Jim Lincoln - Transfinity 12/11/00

3 ideas - quick return

- email--
- dedicated servers

1. • Compression server to plug into corporate LAN for all other servers for storage + transmission (Packet Logix)

2. • Sell to largest individual sites - yahoo, etc. own hosting...

3 • Ricochet - 128K mobile basis wireless...

FDL/Transfinity

12/13/00

EMC-
- SAN
- NAS

Doc Imaging

1. Compression Server -

- who would be partner?
  - IBM, Dell, HP, Compaq.
  - . any would be fine -
  - · any would provide large enough mkt opportunity
  - · However, making a deal would take take time (Dell + Compaq may be easiest)
  - · Rocketlogic is not proven industrial strength at This time
- How to integrate Compression Server into current systems
  - · applications have to interface with storage systems and transmission systems
  - · both input and output require application programming
  - · could use Network File Systems (Unix) - but varies by NFS vendor (SCO, HP, Linux, IBM, SUN...) (Installable File System)
  - · could tie with Storage System providers
- implement compression on selected platform
- implement decompression software on several platforms
- no problem with transmission system
- look at Storage System Vendors as primary partners
  Document Imaging Vendors (Embedded solutions)
- could lead to follow on Rocketlogic sales

2. Sell to Web Hosting Sites

how do WH sites manage their storage systems

probably using 3rd party storage mgt systems

Akamai, Inktomi may have built own system

one shot sales

3. checking on Ricochet

- concern about wireless market size
3rd parties deal with Metrocom

reviews on co. are mixed -
generate lots of noise
relatively expensive modem
+ $70-80 / month
600K subscribers

what would benefit be from
Transfinity compression

ISP-like
small unit - just starting
may be significant technical problems

Subj: **First Dallas Fax Number**
Date: 11/22/2000 1:39:09 PM Eastern Standard Time
From: jl@fdlimited.com
To: jackerman@freshdirect.com, p_barber@4gnetwork.com, james_baker@painewebber.com, dblanton@vortexpartners.com, ibonner@us.ibm.com, mboucher@candw.ky, bray@monarchpartners.com, wadeb@connect.net, rbuchel@jonesday.com, BCarroll@clbn.com, jcordes@e2communications.com, john.cracken@rocketlogix.com, larrydantzler@familyclick.com, john@globalesp.com, James@VE-Group.Com, sdunayer@interserv.com, ghellis@swbell.net, jfarris@e2communications.com, mforman@furmanselz.com, MGirtz@MUNSCH.com, pgoldean@jonesday.com, burtgrad@aol.com, Mike@GlobalESP.Com (Harold, Mike), aghilliii@hotmail.com, Leo@globalcenter.net, chris.hipp@rocketlogix.com, rhogsed@criticaldevices.com, ahoover@corp.sbc.com, tjenn@texas.net, sjosset@clbn.com, jkiser@lycoenergy.com, mackcrest@aol.com, mcguire_mike@emc.com, jtmccafferty@jonesday.com, rusty@vline.net, mmurphy@clbn.com, menorton@jonesday.com, mrachesky@prodigy.net, eric.rasmussen@rocketlogix.com, TR@familyclick.com, lschaufe@lehman.com, Vic_Schmerbeck@htst.com, peter@antennasoftware.com, sherbn@aol.com, Aaron@clbn.com, lang@ve-group.com, grant@clbn.com

Please make note of First Dallas's new fax number below...

Thanks you,

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4175

```
--------------- Headers ---------------------
Return-Path: <jl@fdlimited.com>
Received: from rly-za03.mx.aol.com (rly-za03.mail.aol.com [172.31.36.99]) by air-za04.mail.aol.com (v77.14) with ESMTP;
Wed, 22 Nov 2000 13:39:08 -0500
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-za03.mx.aol.com (v76_r1.19) with
ESMTP; Wed, 22 Nov 2000 13:38:39 -0500
Subject: First Dallas Fax Number
To: jackerman@freshdirect.com, p_barber@4gnetwork.com,
    james_baker@painewebber.com, dblanton@vortexpartners.com,
    ibonner@us.ibm.com, mboucher@candw.ky, bray@monarchpartners.com,
    wadeb@connect.net, rbuchel@jonesday.com, BCarroll@clbn.com,
    jcordes@e2communications.com, john.cracken@rocketlogix.com,
    larrydantzler@familyclick.com, john@globalesp.com, James@VE-Group.Com,
    sdunayer@interserv.com, ghellis@swbell.net,
    jfarris@e2communications.com, mforman@furmanselz.com,
    MGirtz@MUNSCH.com, pgoldean@jonesday.com, burtgrad@aol.com,
    "Harold, Mike" <Mike@GlobalESP.Com>, aghilliii@hotmail.com,
    Leo@globalcenter.net, chris.hipp@rocketlogix.com,
    rhogsed@criticaldevices.com, ahoover@corp.sbc.com, tjenn@texas.net,
    sjosset@clbn.com, jkiser@lycoenergy.com, mackcrest@aol.com,
    mcguire_mike@emc.com, jtmccafferty@jonesday.com, rusty@vline.net,
    mmurphy@clbn.com, menorton@jonesday.com, mrachesky@prodigy.net,
    eric.rasmussen@rocketlogix.com, TR@familyclick.com,
    lschaufe@lehman.com, Vic_Schmerbeck@htst.com,
    peter@antennasoftware.com, sherbn@aol.com, Aaron@clbn.com,
```

BURTON GRAD ASSOCIATES, INC.

7 WHITNEY STREET EXTENSION
WESTPORT, CONNECTICUT 06880
(203) 222-8718
(203) 222-8728 FAX
BURTGRAD@AOL.COM

November 8, 2000

Mr. Jim Lincoln
First Dallas Ltd.
300 Crescent Court, Suite 100
Dallas, TX 75201

Dear Jim:

Burton Grad Associates, Inc. (BGAI) proposes to perform the additional requested technical and business due diligence review of Transfinity for First Dallas Ltd. (FDL).

Objectives

First Dallas wants to have a further independent technical and business due diligence study performed prior to determining whether it wishes to complete an initial investment in Transfinity. This additional study will focus on the specific market directions which Transfinity proposes to pursue. BGAI will assess these plans to be sure that there are no serious market, development, technical or financial issues which would significantly affect estimates of current value or projections of future profits from Transfinity. FDL will separately perform the legal and financial due diligence work it needs as well as determine the effectiveness of Transfinity's organization.

BGAI, an independent consulting firm with extensive experience in due diligence and valuation studies for computer software and services companies, is pleased to perform this additional due diligence study so that FDL can proceed with its planned investment.

Work Plan

1. BGAI will examine the current Transfinity business plan and market planning materials.

2. BGAI will conduct telephone interviews with technical and business executives of Transfinity and review all relevant materials describing the market opportunities, proposed technical requirements, pricing and offering plans, and competitive positions.

3. BGAI will prepare an additional due diligence report for FDL on its findings and recommendations about Transfinity without disclosing any Transfinity-identified confidential program technical information.

As agreed, BGAI will deal with the markets and attempt to answer the questions described in Appendix A.

Staffing

The project will be managed by Burton Grad, president of BGAI, with BGAI Associate Sidney J. Dunayer as the principal consultant.

FDL and Transfinity will designate liaisons to work with BGAI.

Schedule

The interviews will be scheduled for November 8-13 based on mutual availability.

A summary report covering the BGAI additional findings and recommendations will be delivered to FDL on November 15, 2000, if all materials can be obtained and analyzed in a timely fashion.

Confidentiality

All information received and work performed will be treated as fully confidential and not disclosed to any third party without prior written consent from FDL.

BGAI will sign a letter with FDL agreeing to observe the rules of its non-disclosure understandings with Transfinity. Separately, BGAI and its employees and consultants will be bound by a special non-disclosure agreement between BGAI and Transfinity.

Costs and Payments

The due diligence work will be done on a time and expense basis. The following are the BGAI consultant fees:

Burton Grad	$2,800/day
Sid Dunayer	$1,500/day

Based on the information about Transfinity available to us at this time and the information requests from FDL, we estimate that the project will require two to three days for Dunayer and around one and one-half days for Grad. Therefore, the consulting fees for BGAI should not exceed $9,000 unless FDL requests additional analysis, reports or extensive personal debriefings.

There are not expected to be any invoiced expenses for this project.

Payment will be invoiced on completion of the due diligence project.

Payment is due within 15 days of FDL receiving the invoice. If the project is extended beyond November 30, 2000, then BGAI will invoice monthly for its services.

If the above description is satisfactory, please sign below to authorize BGAI to initiate work on this project.

Sincerely,

Burton Grad

Burton Grad, President

Enclosures
BG: 5400.PRO

Accepted for First Dallas, Ltd.

by ______________________________
Signature Date

____
Name

____
Title

Subj: **Transfinity Market Focus**
Date: 11/8/2000 10:16:02 AM Eastern Standard Time
From: jl@fdlimited.com
To: burtgrad@aol.com
CC: sdunayer@interserv.com

214 fax
880 4062
Lincoln

Gentlemen:

Just as idea to ponder, I know if you don't like it you will inform me quickly.

Instead of looking at this from a 30,000 foot level market approach, why don't we just start from the bottom up. Think about the likely candidates who need it now and will pay. Then broaden that market focus based upon that customer.

This seems it might make sense, since we are not going to be building an 'Industry specific product' just license a solution to a customer.

I see the counter that with this approach you are starting with a tremendous number of variables vs. the market refinement approach and starting with ten variables and focusing on candidates within that market.

Your thoughts?

jl

fax to
Sid
chron copy.

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

```
————————— Headers —————————
Return-Path: <jl@fdlimited.com>
Received: from rly-xa05.mx.aol.com (rly-xa05.mail.aol.com [172.20.105.74]) by air-xa03.mail.aol.com (v76_r1.23) with
ESMTP; Wed, 08 Nov 2000 10:16:02 1900
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-xa05.mx.aol.com (v76_r1.19) with
ESMTP; Wed, 08 Nov 2000 10:15:44 -0400
Subject: Transfinity Market Focus
To: burtgrad@aol.com
Cc: sdunayer@interserv.com
X-Mailer: Lotus Notes Release 5.0.2c  February 2, 2000
Message-ID: <OF826873FA.7D3F87C9-ON86256991.0052D188@htst.com>
From: jl@fdlimited.com
Date: Wed, 8 Nov 2000 09:10:57 -0600
X-MIMETrack: Serialize by Router on htstnt001/htst(Release 5.0.3 |March 21, 2000) at 11/08/2000
 09:11:03 AM
MIME-Version: 1.0
```

Subj: **Tfny**
Date: 11/7/2000 5:25:01 PM Eastern Standard Time
From: jl@fdlimited.com
To: burtgrad@aol.com
CC: sdunayer@interserv.com, lang@ve-group.com

Burt,

Sounds like you have sketched a good plan for the first scope of this project.

Just as a reminder - please think of how the following companies might be attacked:
RealNetworks
AOL - Netscape vs. MSFT - Explorer

or add any major players that might warrant selecting them out before finding them through market selection (IBM v EMC)

Thanks,

jl

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

----------------------- Headers --------------------------------
Return-Path: <jl@fdlimited.com>
Received: from rly-zc02.mx.aol.com (rly-zc02.mail.aol.com [172.31.33.2]) by air-zc05.mail.aol.com (v76_r1.23) with ESMTP; Tue, 07 Nov 2000 17:25:01 1900
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-zc02.mx.aol.com (v76_r1.19) with ESMTP; Tue, 07 Nov 2000 17:24:47 -0500
Subject: Tfny
To: burtgrad@aol.com
Cc: sdunayer@interserv.com, lang@ve-group.com
X-Mailer: Lotus Notes Release 5.0.2c February 2, 2000
Message-ID: <OF7E44DF2F.CB6A3D8D-ON86256990.007A2996@htst.com>
From: jl@fdlimited.com
Date: Tue, 7 Nov 2000 16:20:02 -0600
X-MIMETrack: Serialize by Router on htstnt001/htst(Release 5.0.3 |March 21, 2000) at 11/07/2000
04:20:07 PM
MIME-Version: 1.0
Content-type: text/plain; charset=us-ascii

Payment is due within 15 days of FDL receiving the invoice. If the project is extended beyond November 30, 2000, then BGAI will invoice monthly for its services.

If the above description is satisfactory, please sign below to authorize BGAI to initiate work on this project.

Sincerely,

Burton Grad

Burton Grad, President

Enclosures
BG: 5400.PRO

Accepted for First Dallas, Ltd.

by ____________________ 11/08/00
Signature Date

JAMES W. LINCOLN
Name

MANAGING DIRECTOR
Title

BURTON GRAD ASSOCIATES, INC.
5 SAINT JOHN PLACE
WESTPORT, CONNECTICUT 06880
(203) 222-8718
(203) 222-8728 FAX
BURTGRAD@AOL.COM

First Dallas Ltd.
300 Crescent Court
Suite 1000
Dallas, Texas 75201

Invoice #2998

November 27, 2000

Project: #278-7

Attention: Mr. James Lincoln II

INVOICE

Project: Additional Technical Due Diligence of Transfinity

Consulting Services: November 1-17, 2000

Burton Grad	1 day @ $2,800/day	$2,800.00
Sidney J. Dunayer	3 days @ $1,500/day	4,500.00
	Total Invoice	**$7,300.00**

Payment is Due Within 15 Days of Receipt of This Invoice

SIDNEY J. DUNAYER, INC.
418 Tenth Street
Brooklyn, New York 11215-4009

(718) 768-9089

20 November 2000

Mr. Burton Grad
Burton Grad Associates, Inc.
5 St. John Place
Westport, Connecticut 06880

Dear Burt:

For services rendered:

First Dallas - Additional technical review Of Transfinity (3 days)	$4,500.00
Less Advance	(1,500.00)
Total Due	$3,000.00

Payment is expected within 15 days from the date of this invoice.

Our tax I.D. number is 11-2666620.

If you have any questions about this invoice, please feel free to call me.

Sincerely Yours,

Sidney J. Dunayer
President

OK 11/21/00

BURTON GRAD ASSOCIATES, INC.
5 SAINT JOHN PLACE
WESTPORT, CONNECTICUT 06880
(203) 222-8718 FAX: (203) 222-8728
E-MAIL: BURTGRAD@AOL.COM

Date: November 20, 2000

To: Jim Lincoln

From: Burton Grad

Subject: Possible executive and management people for various FDL investments

At your request, I have listed the names of some of the people I worked with at Sterling Software and Sterling Commerce who I felt were strong executives or technical managers. I do not know whether these people are still with the companies or what they are currently doing.

Sterling Software

John Mecke (Dallas)
Jim Johnson (?)
Rick Bodson (Dallas)
Jeff Bingaman (Reston)
Dave Hodgson (Reston)

Sterling Commerce

Neil Baker (Dublin)
Chris Clothier (Dublin)
Greg Dietz (Dallas)
Pat Davis (Ann Arbor)
Randy Harvey (Dublin)
Paul Olson (Dublin)
Dave Pond (Dublin)
Ed Hafner (Dublin)
Clark Woodford (Dublin)
Mary Trick (UK)
Morgan Crew (Dublin)

Rudy Buckel –
214 – 969 – 2990 --

Cavusight - 221 1715

Aaron ~~[illegible]~~ Siler - Streaming media Golind

7

ISP...

- 469-737-4545

~~Big Dough.com~~

~~8821~~

~~Duquesne~~

~~Capital -~~

Subj: **TFNY**
Date: 11/18/2000 12:53:19 AM Eastern Standard Time
From: jl@fdlimited.com
To: burtgrad@aol.com

Burt,

Got the report, much appreciated.

Going forward on TFNY:

We discussed a couple of times people who might be valuable. Who you spend a little time and make me a list and send it over to me.

We will be working with them over the next two weeks digesting more DD and valuing the opportunity.

I will keep you informed.

jl

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

——————— Headers ———————
Return-Path: <jl@fdlimited.com>
Received: from rly-ye03.mx.aol.com (rly-ye03.mail.aol.com [172.18.151.200]) by air-ye03.mail.aol.com (v77.14) with ESMTP; Sat, 18 Nov 2000 00:53:19 -0500
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-ye03.mx.aol.com (v76_r1.19) with ESMTP; Sat, 18 Nov 2000 00:52:59 -0500
Subject: TFNY
To: burtgrad@aol.com
X-Mailer: Lotus Notes Release 5.0.2c February 2, 2000
Message-ID: <OFDF9D3563.E327D241-ON8625699B.001FADDB@htst.com>
From: jl@fdlimited.com
Date: Fri, 17 Nov 2000 23:48:07 -0600
X-MIMETrack: Serialize by Router on htstnt001/htst(Release 5.0.3 |March 21, 2000) at 11/17/2000 11:48:13 PM
MIME-Version: 1.0
Content-type: text/plain; charset=us-ascii

BURTON GRAD ASSOCIATES, INC.

7 WHITNEY STREET EXTENSION
WESTPORT, CONNECTICUT 06880
(203) 222-8718
(203) 222-8728 FAX
BURTGRAD@AOL.COM

November 17, 2000

Mr. Jim Lincoln
First Dallas Ltd.
300 Crescent Court, Suite 100
Dallas, TX 75201

Dear Jim:

At your request, BGAI has performed further technical and business analyses regarding Transfinity. The focus has been on the applicability of Transfinity's patented compression algorithms to various specific markets from functional, technical and business value standpoints.

Transfinity claims that it has exclusive patents to a new, dramatically better compression technique which can significantly reduce the amount of space needed for information that needs to be stored on transmitted. Transfinity has successfully demonstrated the use of these patented algorithms for graphic and image files which were previously compressed using either the GIF or JPEG industry standards. In these cases, Transfinity has shown more than a 50-55% further compression.

Many web sites and other business and institutional files contain a substantial amount of graphic and image material which accounts for a very large percentage of the actual storage and transmission requirements.

In addition, Transfinity claims that its patented algorithms will significantly reduce the storage and transmission requirements for audio and video files. If this can be demonstrated, then there are substantial opportunities for its use in the entertainment and educational markets.

Based on these claims, four particular market opportunities which were identified by Transfinity are being reviewed in this report:

Market	Sample Companies
A. ISP for consumers:	AOL, Earthlink, Telcos
B. Web Hosting/Delivery:	Akamai, Inktomi, Verio, Exodus
C. Corporate Intranet:	GE, GM, IBM
D. Streaming media and Video-on-demand:	Cable, ATT, RealNetworks, Microsoft

For each of these opportunities, BGAI examined the following questions:

1. How could Transfinity's compression technology be used in this market?

2. What would need to be done technically by Transfinity to make its compression technology usable in this market?

3. What would be the financial benefit to customers in this market from use of Transfinity's compression technology?

4. What are BGAI's conclusions, concerns and recommendations for this market?

The remainder of this report consists of an analysis of each of the market opportunities plus an overall summary and business analysis followed by specific action recommendations.

A. **ISP for Consumers**

Transfinity has initially concentrated on developing and delivering a software solution to be used by ISPs to provide their customers with faster delivery of web-based materials. Because of the current performance of the compression software, Transfinity believes that it will be necessary for the ISP to set up a multi-level caching system in order to produce faster delivery in response to customer web requests. Transfinity proposes to charge the ISP one dollar per month per signed up customer for use of the Transfinity software.

1. Functional Assessment: The ISP would need to use a proxy server to handle the web requests from signed-up customers. This server would obtain the web material, expand it to its original form, compress it using Transfinity software and then deliver the results to the customer. At the customer site, the information would be decompressed, using Transfinity software, for normal viewing.

   By using compression, the ISP could potentially deliver graphics oriented web pages at a faster transfer rate over low speed, 28.8K and 56K, lines. This effectively increases the bandwidth of the low speed connection. In order to make this work, the ISP would need to do some form of page caching for the compressed data. The caching functions would most likely be implemented within an HTTP proxy server.

   Since it is not expected that the expansion and compression process can presently be done in real time (with no apparent delay to the customer), the ISP would need to set up both central and distributed caching for frequently used material. This, of course, raises issues regarding selecting what materials to cache, maintaining currency on cached materials, managing the multi-level caching system, etc.

2. Technical Assessment: We believe that the technical problems needed for performing the compression and decompression can be readily solved. However, unless customers can see much faster response to their web requests, this approach will not be beneficial to the ISPs from a marketing standpoint. This means that performance must not only be real time, but must actually be considerably faster than what can be achieved from normal web requests and delivery. Failing faster compression, then the only other solution would be a high level of cached material on a centralized/decentralized basis.

   Transfinity already has the necessary components to make the technology usable for ISPs. However, they have not yet proven the scalability of their solution. As the number of web pages that require caching increases, there is a proportional increase in resources, i.e., disk storage and CPU cycles, required to store and deliver the compressed content. In the Transfinity solution, the workload is distributed over many servers. Since the servers communicate with each other, they should be able to make their combined caches appear as one big logical cache. There is no empirical data available that would allow us to determine the scalability limits of the Transfinity solution.

   While Transfinity has proposed providing a comprehensive compression and caching system, we believe that this would not be a successful strategy for this marketplace. Effective, competitive ISPs will have to implement their own multi-level caching system if they are going to provide competitive delivery speed regardless of whether the ISP is using the normal web compression (GIF or JPEG for graphic/image materials) or whether they are using the Transfinity algorithms. Therefore, the technical effort required to produce a caching system and the marketing effort required to sell it, along with the consulting effort needed to install and support it would be a serious personnel and financial burden.

3. Value Assessment: The ISP might be willing to pay for the use of the Transfinity software, without charge to its customers, if it would reduce the ISP's transmission costs or give the ISP a competitive advantage. Alternatively, the ISP could try to charge its customers for the use of this higher speed delivery mode in the belief that many of its customers would pay an extra two dollars per month for the improved performance.

   For large ISPs, such as Earthlink and AOL, the number of web pages that would need to be cached might be so voluminous as to cause performance degradation within the caching system itself. This could nullify any benefit derived from compression and caching.

   For smaller ISPs, the number of web pages that are cached would be many times smaller than for the larger ISPs. The compression and caching would provide some performance improvement to the end user. There might be some additional benefit to the ISP itself in that they would reduce inbound traffic from the Internet when the pages are in the cache. Note that in any case, the ISP would have to invest in more hardware (servers and caches) and in the Transfinity software.

It is hard to believe that either large or small ISPs would be willing to pay per customer unless they could charge for this premium service. This would be viewed by the ISP as somewhat of an adverse selection process. Those customers who wanted higher speed service might well pay for DSL or cable connections. Those who didn't "need" higher speed service might be reluctant to pay any premium price. In our opinion, the value projections in Transfinity's ISP financial plans don't hang together.

As an alternative marketing approach, Transfinity could:

- charge an initial licensing fee, based on the number of subscribers for an ISP
- charge for the technical work required to design, program, test and install the system for that ISP
- charge an annual licensing and support fee adjusted to the number of subscribers.

This approach leaves the decision of whether or not to charge customers or simply benefit from any operational savings entirely to the ISP. In most cases, we believe, the ISP would provide all customers with the decompression software without charge and selectively use the Transfinity compression software when and where it makes economic sense in terms of which web sites were accessed from which locations and the calculations for transmission cost reductions. Since no commitment would be made in terms of improved performance, there would not be any customer complaints if delivery was not faster.

4. Conclusions, Concerns and Recommendations: Determining a specific cost benefit for an ISP would require an analysis of the reduction in bandwidth achieved versus the costs of establishing additional compression servers and centralized caches. This would in turn require an analysis for each ISP as to the mix of requests that it services in terms of graphics/images vs text, frequency of revisiting same sites, frequency of site modifications, compression performance (speed and reduction), etc.

   The best way to work out whether the Transfinity compression is marketable to a number of ISPs would be to select one of moderate size and proceed to go through the cost benefit analysis. This should lead to a value per 1000 customers and hence a projection as to the minimum size target ISP and a basic pricing plan.

   If there is not an annual saving of at least $200,000-$300,000 per 100,000 subscribers, then this will not be a feasible market to pursue. Note that we assume that each installation will need to be custom designed and built, not so much for the compression and decompression capabilities, but rather to handle the particular hardware, systems software, internal protocols and interfaces that each ISP uses.

B. **Web Hosting/Delivery**

Companies which provide outsourced web content hosting and companies which provide cached content delivery services can benefit by using Transfinity compression to reduce their storage requirements. In addition, the content deliverers can significantly reduce the transmission bandwidth they need by delivering the Transfinity compressed files close to the user before decompression.

1. Functional Assessment: The need for compression is quite obvious for web host sites (or actually for any content storage site). Since most stored content does not change rapidly or even frequently, compression can take place in an asynchronous fashion with little or no real time requirement. Also, since the nature and form of the content is determinable and consistent, very sophisticated multi-step compression processes can be applied in order to achieve greater reductions in storage space required. Applying the Transfinity compression techniques to various forms of content (text, graphics/images, audio and video) would be practical and economical since the decompression programs do not need to be modified for the different compression processes used.

   Similarly, those content host organizations which also provide cached delivery can immediately reduce the transmission bandwidth needed through sending the compressed files to the local caching points where decompression can take place just prior to delivery to the users.

   Since a web host is the delivery source for a website, they would compress the content before hand and then store and transmit the compressed data, eliminating the need for any special caching. The web host would only require software to compress the graphics images and to edit the HTML pages so that they refer to the compressed images.

2. Technical Assessment:

   Transfinity already has developed software to do compression and to edit the HTML pages. The current implementations were designed to work with the Transfinity caching solution. Only a minimal effort should be required to modify the existing software to a form more suitable to the web host requirements.

3. Value Assessment: The most obvious benefit to the web host is the reduced requirements for disk storage and outbound bandwidth. For larger web hosts, this savings might be significant. Web hosts typically use graduated pricing plans that provide a fixed amount of disk storage and monthly bandwidth for some cost, with any additional usage being charged an additional fee. Any solution that decreases the amount of disk storage required by a website or that increases the effective bandwidth provides a direct benefit to the website publisher. Because of this, it is likely that the web host would charge the web publishers a fee for this benefit.

If the fee isn't outrageous and the software is easy to use, then the web publishers would likely embrace the solution. This market might also be expanded to include those web publishers that also provide their own hosting and delivery facilities.

Web hosts like Akamai and Inktomi make multiple copies of the web site dispersed over many servers that are geographically situated so as to improve performance. For these web hosts, the savings might be very significant not only in the disk storage saved, but also in the transmission time necessary to synchronize all the servers.

4. Conclusions, Concerns and Recommendations:

Web hosting has become a popular business while cached content delivery has become quite large, but with only a few significant players. In both cases, these companies should find the Transfinity compression techniques of real value with clear, measurable savings on storage and transmission for both. The cached delivery companies would have the larger savings because of their multiple cache storage sites and their greater transmission requirements (to the last mile). The web hosting companies would save some on transmission by reducing the first mile transmission bandwidth.

While we don't know what the operations costs are for either of these two markets, it would not be surprising if storage and transmission accounted for 25% or more of their operations costs. Even with the hardware needed for compression, a cost reduction of 40% would seem feasible, hence a total reduction of cost by 10%, which would move directly to the profit line, less of course whatever Transfinity charged for its software license, professional services and support/maintenance.

License pricing should probably be based on the amount of stored content with a different rate for the two kinds of customers.

Again, we would recommend finding sample companies, first in the web hosting area and then in the cached content delivery area to determine their precise technical requirements, analyze their potential savings and construct an appropriate pricing model.

C. **Corporate Intranet**

All large corporations have set up substantial internal storage/communications systems to permit all of their business units to share commonly needed information and provide timely updates and additions. For security reasons, many of these organizations manage their own intranets and content storage facilities. The information being stored has now expanded beyond straightforward text and financial files to include graphics and images as well as some audio and video materials. Potentially, large corporations and other institutions (education, health service, government) would be able to reduce costs and improve internal content delivery performance by utilizing Transfinity's compression algorithms.

1. Functional Assessment: Since the corporation has known internal content and a limited number of internal requestors, the ability to compress the largest and most used files would significantly reduce both storage requirements and delivery bandwidth needed. This application does not depend on the corporation marketing or charging for a service, but would be a simple cost reduction analysis.

2. Technical Assessment: If one views a corporation as a combination content publisher, content host provider and internal content deliverer, then the technical requirements are quite basic. The compression algorithms can be honed to provide maximum reduction in the storage space required which, in turn, would produce the largest savings in transmission time. Caching would be optional for the corporation depending upon the frequency of change and use of the materials. All employees would have the decompression program.

3. Value Assessment: As noted earlier, this would be a straightforward determination of cost savings to the corporation versus the packaging and pricing formula from Transfinity. A simple annual licensing scheme with a front-end sign-up plus custom development and implementation charges would provide Transfinity with good initial cash flow and a strong recurring revenue stream. The pricing formula should recognize corporate size in terms of amount of content and the number of employees with system access.

4. Conclusions, Concerns and Recommendations: This seems like the easiest market to address both from a technical and value standpoint. One question is whether the internal content will have as high a ratio of image files as would commercial web sites. Therefore, the savings may depend more on the ability to compress text and financial files rather than the graphics/images reductions. There would probably be little use of stored audio or video materials (except for entertainment companies).

   Therefore, the approach would be to select a few corporations, representing different industries, and determine the applicability and benefit to the customer of using the Transfinity compression techniques. This would identify what characteristics would qualify a prospect and whether the potential savings would be attractive to the company and whether the potential revenues would be attractive to Transfinity.

   This is obviously an area where a marketing partnership would be desirable. The major consulting organizations would be potential partners as well as the various professional service companies who help design and install corporate networks. Some of the Telcos might also be interesting partners (as well as being themselves intranet customers).

D. **Streaming Media and Video-on-Demand**

Video on Demand refers to an interactive delivery mechanism where the consumer can request a video at any time, with full controls that are similar to a VCR. Special purpose high end servers, usually built using Sun Sparc processors with very high speed disk hardware, are utilized to handle the load. Currently, a typical server would handle about 200 concurrent sessions.

Video on Demand can be delivered in two ways: pre-cached and streamed. In the pre-cached method, several videos are downloaded to a device such as a set-top box and the playback occurs locally. In the streamed approach, a continuous stream of data is pushed at software that can then render the data stream. There are several competing standards for streaming: RealNetworks, Apple QuickTime and MPEG4. Both RealNetworks and Apple QuickTime support "plug-in" compressors/decompressors

This market potentially would benefit most from the ability to reduce the bandwidth required for delivery of audio and video material. The current "standards" for compressing audio and video material are all "lossy" to different extents. Critical to determining the opportunities for Transfinity in this market will be whether they can make significant reductions in the bandwidth required without any greater loss of detail. The needs and values will differ between streaming media (real time source compression and delivery) and video-on-demand (stored files, decompressed at user delivery in real time).

1. Functional Assessment: Video data can be very large, say in the hundreds of megabytes for a full length picture. The more you can compress this data, the less storage space it takes and the less time it takes to transmit it. The speed of the decompressor is critical if we wish to present a smooth picture. Currently, there are a number of video compression systems available. One of the more popular ones is known as Sorenson Video Compression. This method has shown excellent compression ratios with little noticeable image quality loss. Transfinity has not demonstrated the ability to compress large video streams and even if they do, they may not achieve results better than current competitors.

2. Technical Assessment: Transfinity would need to create a compressor/decompressor that was tailored for streaming video. This would then need to be packaged to be used as a plug-in for the target system, RealNetworks or QuickTime. Since the primary method for creating QuickTime movies is Mac based software, this code would need to be ported to Mac OS.

3. Value Assessment Obvious benefits are increased storage density and reduced bandwidth. With significant data stream reduction, a server might also be able to handle more concurrent sessions. Since this is an evolving market, it is quite difficult to establish any parameters for prospective pricing, revenue or profits.

4. Conclusions, Concerns and Recommendations: This is potentially a high payoff area, but it is quite specialized and very competitive in terms of compression techniques. Exploring this opportunity from both a technical and marketing standpoint would be relatively costly and may require months of work. This should be put aside for now and reexamined at a later time.

E. **Summary and Business Analysis**

Of the four market areas analyzed above, two seem to be technically quite feasible and of significant customer value (web hosting/delivery and corporate intranet). One other is technically feasible, but may be of limited or questionable customer value (ISP for consumers). The fourth area is potentially quite valuable, but one part is technically very difficult (streaming media) and the other part, while it may be feasible, is unproven (video-on-demand).

Other potential markets include document imaging, storage device manufacturers, interactive game companies, value added networks and wireless communication providers. We're sure that there are many more potential market opportunities.

We have conceived of Transfinity as primarily a technology licensing company with sufficient design, implementation and maintenance skills to satisfy its customers' installation and support requirements. This suggests that the following are the business foundations on which Transfinity's future success should be built:

1. Transfinity's technology must be wholly owned and exclusively proprietary for any applications of potential interest (not just the four analyzed above, but others of economic significance).

2. Transfinity's technology must be demonstrably substantially better than the industry standards for each application opportunity (graphics, images, audio, video, etc.).

3. Transfinity's implementations should be fast enough in each application area to provide demand-driven response rates either through real time compression or through appropriate pre-compression and storage.

4. Transfinity's offers and prices should encourage smaller prospects to become customers while providing substantially greater revenue from larger customers. The marketing structure and strategy should also ensure reasonable profit on initial licensing and installation with high profits on the recurring revenue from continuing license and support renewals.

5. Transfinity should carefully focus its marketing and technical efforts on the easier target areas and not be distracted (technically or financially) by high cost, high risk opportunities.

6. While marketing partnerships seem attractive in certain marketplaces, these partnerships may cost Transfinity too much effort for the likely payback and may require too much management time so that other strategic opportunities would be neglected.

7. Transfinity is currently under-managed and lacks proper strategic focus. It needs dedicated executive, marketing and technical staffing with clearly defined financial resources. It needs

a practical short term business plan. The longer term business plan should evolve after some initial technical and marketing achievements.

F. **Recommendations**

Given this statement of the foundations on which we believe Transfinity should be built, we recommend that FDL work with Transfinity to carry out the following steps:

1. Thoroughly assess the ownership and scope of the Transfinity patents and any other Transfinity technical assets (programs, trade secrets, knowledge).

2. Spell out the precise financial resources to be devoted to Transfinity and set up clearly independent books and plans, not overlapping with VE Group or Global ESP.

3. Define the business relations between Transfinity and all other related entities (VE Group, Global ESP, Gemini, Mike Harold, Joe Morgan, etc.).

4. Conduct initial market opportunity studies on two of the four initially defined areas plus two other high potential markets. Based on these studies, set strong priorities on technical and marketing investments, excluding secondary or difficult markets for the next 12 months.

5. Recruit executive management and key marketing and technical personnel.

6. Prepare a well thought through short term business plan reflecting the market/technical priorities and avoiding excessive ambition in the short term.

We still believe that Transfinity could be a very successful company and that an early limited investment by FDL with opportunities for further investments at attractive prices would be desirable.

Sincerely,

Burton Grad
cc: Sid Dunayer
5400.RPT

Axes for Strategic Planning
Matrix

ISP - consumer transmission (dial-up)

ASP - commercial

Web Hosting Providers

Content Delivery Providers

=

- Intranet
- Internet

Content Producers - CD-ROM/storage devices

Content Deliverers - content storage
content transmission
~~Communication~~

~~wire~~ * content ~~Users~~ categories - consumers - enduser/content receiver

- businesses
- govts
- educ institution

media companies

*Content Categories
- text/numeric - HTML
- graphics/pictures (GIF's, JPEGS)
- audio/video streaming

~~video~~

4-5 days
11/15 target

Document Imaging - Xerox - financial institutions

wireless. black box - telephone producers.

Appendix A

Proposed Markets from Transfinity

4) Video-on-demand - ~~[illegible]~~ Cable, ATT, RealNetworks, Microsoft

1) ISP for consumers - AOL, Telcos, EarthLink

2) Web Hosting/delivery - Akamai, Inktomi, Vario, Exodus

3) Corporate Intranet - GE, GM, IBM

~~(Browser Co.)~~

Questions to be researched for each market:

1) How ~~could~~ Transfinity's compression technology be used in ~~the~~ this market

2) What would technically need to be done by Transfinity to make its technology usable in this market

3) What would be the value ~~?~~ (to customers) in this market from use of Transfinity's compression technology?

~~$8-9K~~
~~on - [illegible] Lincoln~~
~~send~~

Transfinity --- 11/6/00

18M pre money
2M investment
1M @ 25M

Target close - 11/30

S.d - ok for ltd image license

proposal -
identifying add'l mkts
customers -
license terms -
mkt assessment
technical

new business plan
approach to AOL
Microsoft
Real Networks

ISP to Consumer

Hosting Service

ASP

sdemayor @ interserv.com

Corporate Lans

Subj: **Market Assessment**
Date: 11/6/2000 4:21:19 PM Eastern Standard Time
From: jl@fdlimited.com
To: lang@ve-group.com
CC: burtgrad@aol.com

Lang,

I have asked Burt to give us a preliminary assessment of potential markets for Transfinity (Tfny).

He will need access to any materials that might speed up this process, such as:

Research reports- Gartner, etc
Internal research
Etc

Burt will be getting back to us in terms of cost and scope of project in the next few days.

Remind me when we talk next and we can discuss the proposal and payment for BGAI's services.

jl

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

———————— Headers ————————

```
Return-Path: <jl@fdlimited.com>
Received: from rly-yd02.mx.aol.com (rly-yd02.mail.aol.com [172.18.150.2]) by air-yd01.mail.aol.com (v76_r1.19) with ESMTP;
Mon, 06 Nov 2000 16:21:19 -0500
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-yd02.mx.aol.com (v76_r1.19) with
ESMTP; Mon, 06 Nov 2000 16:21:03 -0400
Subject: Market Assessment
To: lang@ve-group.com
Cc: burtgrad@aol.com
X-Mailer: Lotus Notes Release 5.0.2c  February 2, 2000
Message-ID: <OF0AE7F531.2E48EAA4-ON8625698F.0061CB9D@htst.com>
From: jl@fdlimited.com
Date: Mon, 6 Nov 2000 15:16:23 -0600
X-MIMETrack: Serialize by Router on htstnt001/htst(Release 5.0.3 |March 21, 2000) at 11/06/2000
 03:16:24 PM
MIME-Version: 1.0
Content-type: text/plain; charset=us-ascii
```

Subj: **Transfinity Business Plan**
Date: 11/6/2000 12:26:50 PM Eastern Standard Time
From: jl@fdlimited.com
To: burtgrad@aol.com

File: Transfinity BusPlan 092700.pdf (136040 bytes)
DL Time (50666 bps): < 1 minute

FYI

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

---- Forwarded by Jim Lincoln/htst on 11/06/00 11:26 AM ----

"Wedgeworth, Lang" <Lang@VE-Group.Com>
10/18/00
03:51 PM

To: "Jim Lincoln (E-mail)" <jl@fdlimited.com>
cc:
Subject: Transfinity Business Plan

Lang Wedgeworth

T:972.550.1133 ext. 100   1231 Greenway Drive, Suite 300
F:972.753.4407   Irving, Texas 75038
<<Transfinity BusPlan 092700.pdf>>

(See attached file: Transfinity BusPlan 092700.pdf)

---------------- Headers ----------------
Return-Path: <jl@fdlimited.com>
Received: from rly-yh02.mx.aol.com (rly-yh02.mail.aol.com [172.18.147.34]) by air-yh02.mail.aol.com (v76_r1.23) with ESMTP; Mon, 06 Nov 2000 12:26:49 -0500
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-yh02.mx.aol.com (v76_r1.19) with ESMTP; Mon, 06 Nov 2000 12:25:42 -0400
Subject: Transfinity Business Plan
To: burtgrad@aol.com
X-Mailer: Lotus Notes Release 5.0.2c February 2, 2000
Message-ID: <OF697372CE.5F22FD10-ON8625698F.005F498C@htst.com>
From: jl@fdlimited.com

Subj: **Transfinity Power Point Presentation**
Date: 11/6/2000 12:25:47 PM Eastern Standard Time
From: jl@fdlimited.com
To: burtgrad@aol.com

File: Transfinity Edgewidth.ppt (275968 bytes)
DL Time (50666 bps): < 1 minute

FYI

James W. Lincoln
Managing Director
First Dallas, Ltd

300 Crescent Court
Suite 1000
Dallas, Texas 75201
T: 214.880.4100
F: 214.880.4062

---- Forwarded by Jim Lincoln/htst on 11/06/00 11:24 AM ----

"Wedgeworth,
Lang" To: "Jim Lincoln (E-mail)" <jl@fdlimited.com>
<Lang@VE-Grou cc:
p.Com> Subject: Transfinity Power Point Presentation

11/05/00
02:37 PM

Jim:

Attached is a copy of the captioned file.

Lang Wedgeworth

T:972.550.1133 ext. 100 1231 Greenway Drive, Suite 300
F:972.753.4407 Irving, Texas 75038
<<Transfinity Edgewidth.ppt>>

(See attached file: Transfinity Edgewidth.ppt)

---------------- Headers ----------------
Return-Path: <jl@fdlimited.com>
Received: from rly-zc01.mx.aol.com (rly-zc01.mail.aol.com [172.31.33.1]) by air-zc03.mail.aol.com (v76_r1.23) with ESMTP; Mon, 06 Nov 2000 12:25:47 -0500
Received: from htstnt001.htst.com (htstnt001.highland-wealth.com [208.206.19.15]) by rly-zc01.mx.aol.com (v76_r1.19) with ESMTP; Mon, 06 Nov 2000 12:23:53 -0500
Subject: Transfinity Power Point Presentation
To: burtgrad@aol.com
X-Mailer: Lotus Notes Release 5.0.2c February 2, 2000

Transfinity Business Notes

1) Who owns the patents? and patent pending?
What rights have been granted to whom
with what restrictions?
what is reln with Gemini, VE Group,
Transfinity, GlobalESP, various
individuals

2) Is Transfinity in the compression
business? caching business?
encryption business?
What functions can the n-bit
technology be applied to
besides ~~[illegible]~~ compression?

3) What are the various types of
files and communications
to which the n-bit technology
can be applied?
what level of compression is
realistic for different types
of files and communications

4) what ~~is~~ are the packaging and pricing models for
various markets — only
reference is to $1/mo. per
user. — ?
Shouldn't there be up front charges
for ISP, ASP, etc.

5) what is fair value of Transfinity
and compression technology?

Questions for Transfinity --

✓ • Scaleability - how much volume/server
how to connect user to avail server

✓ • operating Process - steps in handling user request

✓ • User Reqts - DeCompression, distrib server address

✓ • Combination ISP/Web host -

✓ • Performance - how fast to do how much

✓ • Code quality - proof of concept, alpha, beta [illegible]

✓ • design approach -

✓ • need for cache - when + how

✓ • relative significance of GIF's + JPEG's in Web sites

✓ • development plan -

[copy from Jim Lincoln • business plan -

FDL/VE Group

10/18/00

Transfinity -

Lang Wedgeworth
Mike Harold
Dennis Tucker
} Transfinity

John Dean
Jim Lincoln - FDL
Sid Dunayer, Burton Grad - BGAI

=

Browser Plug-in -
Opportunities - Web Hosting
ASP's
Content Providers
Video on Demand

Process --
User installs Browser Plug-in
Proxy setting on his browser -
Requests forwarded to Distributor Server (at ISP)
Checks in its cache - That dist server + other linked distrib servers -.
if in cache then info back to user
distrib server request from Control Server
is it in control Server cache -
goes to Internet + gets original page
edits to input specification
sends to user -- Transfinity page
requests content from Web site
+ pulling down all links
decompress GIFs + JPEGs, etc.
Analysis of files: Mime-type
can update control server or plug in
Compression (bypass if not done in time)
Distribution.
Time to live

Transfinity

Subj: **Talk**
Date: 11/13/2000 6:41:58 PM Eastern Standard Time
From: Mike@GlobalESP.Com (Harold, Mike)
To: burtgrad@aol.com ('burtgrad@aol.com')
CC: jl@fdlimited.com ('jl@fdlimited.com'), Lang@VE-Group.Com (Wedgeworth, Lang)

Burt,

Thanks for your call. I left several messages on your cell. Sorry I missed you. Please call me on my cell at 972-342-7694 or send me an email to set a date and time. I understand from Lang that you want to talk about market opportunities.

Initially I thought that 1) media distribution, 2) video on demand and 3) storage should be the markets and the sequence. Several of the dominant ISPs, caching/web hosting companies and ASPs (as well as one of the world's largest content companies) have told us a different story.

Even though broadband is readily available, these companies are experiencing a great deal of pain in both the storage and communication of data. Some of the data is their customers'. A lot of the data is their own. We now understand that the ability to reduce the cost of both storing and communicating log files, billing information, provisioning information, etc. could make the difference between profitability and loss for many of these companies. They are drowning in data.

We also now know that the "last mile" is on the radar screen for many companies. The greater the back office bandwidth becomes, the more embarrassing their inability to increase the speed of content delivery to their customers' customers.

I still think Transfinity needs to focus on last mile solutions as a priority. But I now believe that compression for storage and network communications represents as great an opportunity. For the time being, video on demand is less important.

These comments are not at all intended as conclusions. I would very much like to hear your thoughts on the matter and look forward to hearing from you.

Mike

```
------------- Headers ------------------
Return-Path: <Mike@GlobalESP.Com>
Received: from rly-zc02.mx.aol.com (rly-zc02.mail.aol.com [172.31.33.2]) by air-zc04.mail.aol.com (v76_r1.23) with ESMTP;
Mon, 13 Nov 2000 18:41:58 -0500
Received: from Enterprise.VE-Group.Com (dsl2-5.onlinetoday.com [205.242.60.5]) by rly-zc02.mx.aol.com (v76_r1.19) with
ESMTP; Mon, 13 Nov 2000 18:41:28 -0500
Received: by enterprise.ve-group.com with Internet Mail Service (5.5.2650.21)
    id <V9M1F8KQ>; Mon, 13 Nov 2000 17:41:24 -0600
Message-ID: <9680D30D92EBD311B58000A0C9E3D98778BF@enterprise.ve-group.com>
From: "Harold, Mike" <Mike@GlobalESP.Com>
To: "'burtgrad@aol.com'" <burtgrad@aol.com>
Cc: "'jl@fdlimited.com'" <jl@fdlimited.com>,
     "Wedgeworth, Lang"
     <Lang@VE-Group.Com>
```

TRANSFINITY

Business Plan

1. Executive Summary
2. Market Opportunities
3. Management and Development Team
4. Financial Projections
5. Appendix: Transfinity's N-Bit Compression Technology

Contact: James Dodd, CFO

972.550.1133 Ext. 101
214.674.7222 Mobile

james@transfinity.com

The information contained herein is confidential. Recipients shall not disclose such information to third parties or copy the materials.

October 2000

This business plan does not constitute an offer to sell or solicitation of an offer to buy any securities, nor does it constitute an offer to sell or solicitation of an offer to buy from any person in any state or other jurisdiction in which an offer would be unlawful.

Transfinity Corporation

1 Executive Summary

Transfinity Corporation is an early stage technology company that has developed and patented a compression technology that compresses previously compressed files. This capability represents a significant technological advance and creates a panoply of market opportunities in the Internet infrastructure as well as more traditional Telco spaces.

Transfinity's compression creates value in a multitude of markets. Because of the pervasiveness of the Internet, and the seemingly insatiable demand for faster delivery of content, Transfinity will first address several markets associated with Internet content delivery and Internet access. Transfinity's technology increases apparent data transfer speed for the most common file types by a factor of approximately 3X. The technology is complementary to all data transfer media, including twisted pair telephone (POTS), ISDN, DSL, cable, fibre, and wireless. As a result, our technology provides not only for the intelligent and faster distribution of Internet content, but also has multiple applications in the bandwidth and storage markets. Transfinity is also targeting the visual media markets. We are currently developing the capability to convert video and streaming media files into a file type proprietary to Transfinity. By applying our compression and encryption technologies to these files, we believe we can speed the transmission and enhance the security of video and streaming media content. With additional resources, each of these applications can be migrated to the wireless environment.

Technologists now realize that the Internet itself is a computer. The same solutions that are required to manage data communications inside a PC, (i.e. synchronization, caching, memory access, data management, etc.) are also required to run the computer that is the Internet. This new computing paradigm will require technologies that intelligently distribute data to the point of access closest to the user and deliver it on demand through any connection type. Transfinity has developed compression and caching solutions that reduce latency time, shorten the path between content providers and users, and increase the bandwidth and storage capacity of the network at every point of access.

Transfinity's software turns the network into a computer by distributing the processing, storage, and communication of Internet content. Transfinity's software is both componentized and distributed and can be placed anywhere on the network. This means that content can be compressed at any point in the network and stored at many points in the network. This approach allows everyone involved – users, service providers, and content providers – to benefit from the dramatic increases in throughput that result from the combination of our proprietary compression, caching, and content routing.

Technology Overview

Transfinity's core technology is its compression. In brief, Transfinity's technology can compress compressed files. And when used to compress uncompressed files, Transfinity's compression provides order of magnitude improvements over other methods. In addition, Transfinity has developed compression capabilities that support and significantly increase the compression ratios of "lossy" compression methods such as JPEG. Transfinity's technology is intended to enhance, rather than replace, existing hardware and software storage and bandwidth standards and practices. Our solution does not require costly hardware additions. Rather, Transfinity compression is effected primarily via software that is transparent to the end user. In addition, the footprint for our decompression software is small (approximately 300 Kb). This accommodates an easily installed browser plug-in for the end user. Importantly for the wireless market, it also allows the software to reside on hand held telephones, PDAs, and other communications devices.

Transfinity's software dramatically increases content delivery speed to the end user via an automatically installed browser plug-in at the end-user's PC. As a result, a browser running through a 28K modem will perform as if it is running at 90K. A browser running through a 56K modem will run faster than an ISDN connection. A DSL or 3G wireless connection will triple in speed and support full-screen, high-definition video. Although broadband is being widely deployed, no one has solved the last mile problem. The majority of users currently do not connect to the Internet via broadband, and cost considerations seem to ensure this will be the case for some time to come. And although third generation wireless is fast, it's not fast enough to support the growing demand for rich content. Transfinity significantly improves the performance of any last mile transmission medium with minimal additional capital cost. The number of potential users of the Transfinity solution is enormous. We believe there are immediate and substantial opportunities in the Internet Service Provider (ISP) and Telco markets to provide low cost, easily installed, faster Internet connections via Transfinity technology.

We believe that Transfinity's technology can enhance the transmission speed and security of visual media distributed over the Internet. Transfinity's unique solution combines encryption with its proprietary image formats so that images are only viewed inside the browser. Visual media protected by Transfinity technology could then be viewed only by those authorized by the content provider. This capability requires no additional hardware for the end user. Transfinity technology currently supports animation, and we have demonstrated full motion video via a 56K modem in a controlled environment. Our product development team will next focus on provisioning streaming video over dial-up and DSL.

Transfinity's compression technology also increases the effective bandwidth and/or storage capacity of any data medium by a factor of approximately 2X. This means that service providers can support twice as many users or twice as much content with their current bandwidth infrastructure. For example, in the cable television market, Transfinity compression can reduce the bandwidth usage of downloaded content by a factor of 2, and thereby allow the cable operator to support twice as many subscribers with a single downstream DOCSIS channel.

Market Opportunities

The data transfer and storage markets that appear most receptive to near term deployments of Transfinity technology include:

- □ Internet service providers- ISP dial- up
- □ Internet content delivery intermediaries
    - Application service providers- ASP
    - Internet content delivery (caching) providers
    - Web hosting companies
- □ Large corporate users of LAN, WAN, and e-mail systems
- □ Video and streaming media enablers
- □ Telcos seeking inexpensive bandwidth enhancements for existing systems

The Transfinity solution is robust, scalable, low cost, readily installed, and applicable to numerous data transmission, bandwidth, and storage markets. Transfinity will soon deploy its technology as an Internet access and content distribution solution for the ISP dial-up market. Alpha testing in the controlled environment of Company headquarters is complete. Further testing will begin shortly by offering selected individuals Transfinity compression-enabled Internet access accounts. Beta testing will occur at a Tier 2 or Tier 3 ISP. This will, at a relatively minimal capital expense to Transfinity, demonstrate to target licensees such as Tier 1 ISPs and Telcos the practical application and potential value of this new technology.

Given the similarities in design architectures, once our technology has been successfully deployed as an ISP and Telco solution, we should require minimal additional market testing prior to formal rollout of our product offerings into the ASP, caching, web hosting, and large corporate markets. Deployment into the corporate network and ASP e-mail market will require an additional six man-months to develop a PC client compress/decompress program applicable to the various protocols emanating from the web. We believe our streaming media and video enhancements will be ready for market within nine months. With sufficient resources, we believe we can develop similar applications for the wireless environment within 15 months.

Company Overview

Transfinity Corporation is positioned as an enabling technology research and development company. Our revenue model is licensing. We will license our technology to selected leaders in the content delivery, data transfer, bandwidth, and streaming media markets. Our headquarters are located in Irving, Texas, a suburb of Dallas. Our technology is protected by issued patents and patents pending.

Transfinity currently has 9 employees. Nearly all have technical backgrounds, specifically in software architecture and development. Our Senior Vice President of Technology was formerly the chief architect for the Logistics, Electronic Commerce, and Catalog Division of FedEx. He holds patents and patents pending in the fields of compression, encryption, arbitrary precision mathematics, virtual machine architectures, distributed computing, and media distribution.

2 Initial Market Opportunities

The Internet's potential to deliver content to anyone, anywhere, at anytime is becoming more and more a reality. Multiple fibre optic backbones now connect all metropolitan areas in the U.S. The rapid deployment of DSL and cable modems promises to provide higher bandwidth to many households. And 3G wireless communications will soon provide "wire line" users with bandwidth comparable to "land line."

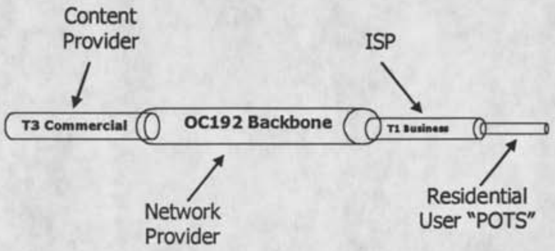
But despite these advances, serious issues remain. The bottleneck at the delivery point of the Internet into the typical household is perhaps the most frustrating. Approximately 90% of U.S. households still use plain old telephone service or POTS over 28K and 56K modems. It will be years before higher bandwidth connections such as DSL predominate. And even then, rural communities, which represent 25% of potential U. S. Internet users, will not have access to these higher bandwidth connections. No matter how fast the fibre optic backbone becomes, the Internet is only as fast as the connection to the ultimate content consumer– the end user. Transfinity addresses this and other problems associated with Internet access and content delivery.

Transfinity has identified several key user markets that offer high potential for near term licensing arrangements: 1) the ISP dial up market where our compression and caching solution can increase browser speeds by approximately three fold; 2) the ASP market that would benefit from significant reductions in communications and storage costs and an increase in transmission speed for clients; 3) the Internet caching and web hosting markets where our technology can increase content transmission speed, improve content security, and reduce data storage costs; and 4) the large corporate market where communications costs and bandwidth requirements can be substantially improved with Transfinity technology; and 5) video and streaming media enablers that currently suffer from severe bandwidth constraints.

With additional resources, we expect to a) develop the final enhancements required to compress all common file types associated with e-mail attachments, b) continue ongoing development of our streaming media and video enhancements, and c) continue research and development of wireless compatible versions of our core technology. We expect to market to the large corporate WAN and e-mail market within six months, the streaming media market within nine months, and the wireless infrastructure community by first quarter 2002.

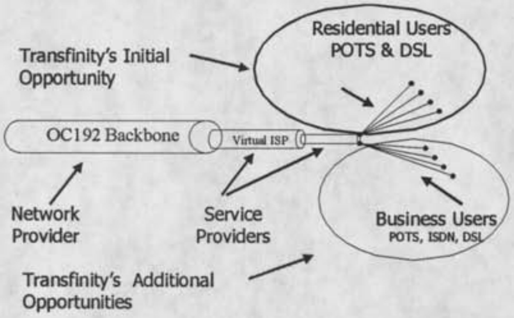
2a ISP Dial-UP Market

The explosive growth of the World Wide Web has created an insatiable demand for bandwidth. In response, companies such as Nortel, MCI Worldcom, Global Crossing, Enron, Level3, and others have constructed broadband infrastructures that transmit data from the content provider to the content consumer. However, this infrastructure encounters a significant bottleneck when it typically shifts at a distribution node from high bandwidth, optical fibers to the "twisted-pair" copper wire that connects most end-users to the network.



The Last Mile "Bottleneck"

Transfinity's low cost, high-speed data compression and caching will dramatically reduce the delays inherent in the traditional "last mile" bottleneck. We see multiple opportunities.



New distribution vehicles and technologies are being deployed that attempt to deliver broadband to the end-user. Each has its respective technical advantages and disadvantages. (Refer to the table on page 7 for a comparison of the technical pros and cons.)

For the foreseeable future the majority of Internet users will continue to use POTS dial-up service. According to eMarketer, in 1999, 54.8 million (94.5%) of Internet users in the United States accessed the Internet through POTS. Because of the overall growth in Internet access, eMarketer estimates that even with growth in broadband services, POTS users will still number 55 million in 2003, or 70% of total Internet users.

DSL and cable modems will be the first of the broadband technologies to be deployed. Goldman Sachs estimates that by 2003, 80% of houses in the United States will be DSL-ready, whereas

72% of households will be cable modem-ready. At that point the real test for broadband will come-- when given the option of using any of these services, will end users choose high-speed access, notwithstanding the additional costs? A recent study by Jupiter Communications indicated that end-users would not choose these services if they cost significantly more than POTS dial-up service. 52% of people surveyed said that they would not be willing to pay more than they are currently paying their ISP.

The significant disadvantage for each of these new technologies is the *time and cost of deployment*. While slow, the current distribution system via "plain old telephone service" ("POTS") – an investment of over $100 billion by the phone companies – is in place and fully amortized. Conversely, each of the newer technologies is not yet widely deployed and will require a huge capital investment that must be recouped. Where the new distribution pipes are deployed, in some instances the actual performance is below expectations.

We have drawn two conclusions: *1) When overlaid onto POTS, Transfinity's compression solution has near-to-intermediate term advantages over these other developing distribution pipes, certainly in terms of cost to deploy and cost to the end user, but also in many cases in terms of performance*. For example, a POTS/Transfinity user running a browser on a 56K modem will experience performance roughly equivalent to that of ISDN without Transfinity compression. A browser running through a DSL line using Transfinity's compression will approximate a T-1 connection without Transfinity compression. *2) These alternative broadband connections are not necessarily technological threats to Transfinity's compression, but instead represent additional market opportunities*. Transfinity's technology applies equally to any type of connection, whether POTS, ISDN, DSL, cable modem, fibre, or wireless. Our technology will significantly improve the performance of each of these alternative distribution media.

Transfinity's compression technology can increase Internet access speed by a factor of approximately 3X, irrespective of transmission medium. Transfinity will demonstrate its technology by targeting first the 55 million Internet users in the United States who use and will continue to use POTS dial-up access. Our solution is transparent to the user and simple and relatively inexpensive for an ISP or Telco to implement.

Transfinity's business plan includes the following steps to demonstrate the value of its compression technology in the ISP sector:

- Partner with a Tier 2 or Tier 3 ISP to provide a full-scale demonstration of our technology's capabilities.

- Based upon the pricing sensitivities uncovered in the demonstration project, refine the pricing schedules for our licensing business.

- License our software to domestic and international Tier 1 ISPs and Telcos.

Last Mile Technology Comparison

TECHNOLOGY	HOW IT WORKS	CAPACITY (KBIT/Sec)	ADVANTAGE	LIMITATIONS/ DISADVANTAGES
POTS (Plain Old telephone Service)	Transmits data on phone lines.	28 analog 56 digital	In place; no deployment costs.	• Slow: Current maximum rate is 56k
DSL (Digital subscriber line)	Transmits digital data on phone lines at frequencies higher than those used for voice. Frequencies are separated at the home. Individual homes get dedicated lines.	Downstream: up to 1500 Upstream: 600-800	Can use existing phone lines	• Requires more infrastructure • Service limited to 5.5 km from phone switching node • Top speeds possible only on short lines • Not available for all phone customers.
Cable modem	Data travels to home on TV (coaxial) cable in a frequency band used for video channel. Upstream transmission is at a lower frequency or on phone wires.	Downstream: ~1000 Upstream: typically 50-100	Uses existing coaxial cable	• Individual data rate drops with number of users • Poor security • Not available on all cable systems
Wireless (terrestrial)	Local antenna broadcasts microwaves, picked up by home antenna. Broadcasts video signals and can transmit data.	Comparable to DSL	Mobile. No buildout to user	• Multipath interference from buildings • Trees, terrain, and rain can block signals • Interference possible from other cells • Signals travel limited distance (like cell phones)
Wireless (satellite)	Satellite broadcasts data signals to individual receivers. Might be added to direct broadcast satellite service, or to mobile low-earth-orbit service such as Teledesic.	To be defined	No cable, no local broadcast antennas	• Better suited to broadcasting because of large satellite coverage area • Limited data rates likely
Fibre to the home	Fibre carries data to homes. Could also carry broadcast video, either in same signal or other wavelengths.	200,000 to 1,000,000	Highest speed	• Cost of construction

2b Application Service Providers

The ASP market is expected to grow dramatically over the next few years as businesses seek to gain efficiencies by outsourcing IT services. The outsourced messaging, or e-mail, industry is the fastest growing segment of the ASP market space and the one most easily outsourced. The Yankee Group estimates that outsourced messaging market revenues will increase from $1.2 billion in 1999 to over $5.4 billion by the year 2003.

Transfinity provides key benefits to providers and users of outsourced mailboxes. The Transfinity solution differentiates ASP services in terms of performance and cost. Our ability to compress mail attachments will greatly improve both upload and download times. Where connections are priced by the minute, faster transfer times for mail messages and their attachments will decrease communications costs. In addition to images, the Transfinity solution will be extended to support other common file types such as Microsoft Office files, .pdf files, and Lotus Notes files.

Importantly, Transfinity's technology is compatible with existing messaging systems. Transfinity's software compresses e-mails and their attachments before they are sent to the service provider and decompresses them before the user sees them. As a result, the software is transparent to the user. Clients use their systems the way they normally do, without having to compress or decompress the mail messages or their attachments. By being non-invasive, Transfinity's software remains compatible with existing messaging systems such as Lotus Notes and MS Exchange.

2c Internet Caching and Web Hosting Providers

As demand for faster Internet content delivery has grown, so has the Content Delivery Service Provider (CDSP) or commercial caching market. Typically, CDSPs use a combination of proxy servers and caching strategies to store static information closer to the source of an information request. When also using compression and high-speed storage techniques such as memory or disk caching, caching providers can dramatically improve the response time for a given request. According to the Yankee Group, the market for caching hardware and software is expected to grow from $180 million in 1999 to $1.7 billion in 2003. CDSPs receive the majority of their revenues from content providers. In a related development, many companies and content providers have decided to outsource the management of their Internet sites and the hosting of their mission critical web servers. Storage and bandwidth requirements are critical elements of the business models for web hosting companies.

Transfinity does not view these companies as competitors. Rather, they are potential channel partners. All of them are sensitive to data transmission and storage costs, and none of them offer a last mile solution for their clients' end users. In other words, CDSPs and web hosting providers can lower their costs and improve performance to their ultimate end users by incorporating Transfinity technology. Transfinity compression and caching could be offered as a premium service or as a distinguishing feature versus the competition. In addition, Transfinity offers a security feature that should be valued by this channel's content provider customers. Transfinity encryption uses digital certificate and public key technology. The content is secure as it travels from the content provider (caching provider or web host) to the consumer. Once the content is made available to the typical end user, he could see it in the browser and yet be unable to copy or save it without extraordinary measures.

2d Large Corporate Users

The rapid growth of the Internet has greatly increased the value of data. With the convergence of voice and data over both local and wide area networks, the volume of data traffic is increasing at an exponential rate. In order to meet the growing requirement for voice, text, images, and video-on-demand, networks must have access to high bandwidth communications. The majority of LANs are built on TCP/IP and Ethernet protocols. Ethernet has advanced to support one-gigabit transmission rates. In the corporate LAN environment, video conferencing and media distribution drives the requirement for bandwidth and storage. For the time being, WANs (interconnected LANs) continue to rely on traditional leased line connections using protocols such as ISDN and T1. The need to support rich data types such as images and audio has made the use of these transport mechanisms increasingly expensive. For this reason, many companies have begun to introduce compression-based networking devices and integrated circuits in an effort to add bandwidth to existing communication infrastructures.

Transfinity believes there can be a substantial opportunity to market its compression and caching technologies directly to large corporates whose communications costs are substantial and whose employees require expanded bandwidth. Transfinity could dedicate its compression solution into the corporate's LAN and WAN networks, as opposed to third party service providers, and charge in essence a corporate wholesale rate.

2e Streaming Media and Video-on-Demand Market

The promises of streaming media and video-on-demand remain largely unfulfilled. This is largely due to security issues and the bandwidth constraints associated with the delivery of video to the home. . The Transfinity security design encrypts the content and prevents the user from viewing it outside of the browser. The content would remain both compressed and encrypted except when in memory. And even in memory, it would remain in Transfinity's proprietary format. This security applies equally well to streaming media and video.

Transfinity is also developing solutions for the bandwidth constraints associated with streaming media and video. The presentation of streaming media requires a reliable connection with a display rate of between 12-24 frames per second. The presentation of video requires a reliable connection with a display rate of between 24-30 frames per second. Using its compression technologies, Transfinity can demonstrate the delivery of streaming media at a rate of 15-20 frames per second over a 56k dial-up connection without the requirement for a media server on the part of the service provider. Transfinity is also researching the distribution of full-screen video-on-demand through DSL, cable, and 3G wireless at a rate of 24 frames per second. Providing consumers the ability to view any movie, television episode, or video at any time represents one of Transfinity's biggest opportunities.

3 Management and Development Team

Transfinity currently has 9 employees. Nearly all have technical backgrounds, specifically in software architecture and development. Employees hold equity or option interests representing, in total, approximately 20% ownership of the Company.

Lang Wedgeworth
Co-Founder and President

Mr. Wedgeworth is responsible for managing Transfinity's business affairs, including its intellectual property portfolio. Prior to joining Transfinity, Mr. Wedgeworth spent over twenty-two years in the private practice of law, with his primary focus upon transactional and business law. Mr. Wedgeworth holds a JD from the University of Texas and a BA from Southern Methodist University.

Mike Harold
Co-Founder and Senior Vice President of Technology

Mr. Harold has over twenty years experience in the computing industry. An expert in distributed computing, systems integration, operating systems, and database design, he has implemented solutions in a wide range of industries including Transportation, Health Care, Logistics, Finance and Education. Mr. Harold is responsible for the technical vision and architectural direction of the Company. Prior to founding Transfinity, he was the chief architect for the Logistics, Electronic Commerce, and Catalog division of FedEx. Mr. Harold has patents and patents pending in the fields of encryption, compression, arbitrary precision mathematics, virtual machine architectures, distributed computing, media distribution, and various business processes.

James Dodd
Chief Financial Officer

Mr. Dodd brings seventeen years of experience in investment banking, corporate finance, and institutional investment management, with particular expertise in raising capital for high growth companies. Prior to joining Transfinity, Mr. Dodd was responsible for international financial institution marketing at Chesapeake Capital, a $1 billion hedge fund. Previously, Mr. Dodd was a Managing Director in the corporate finance department of Hoak, Breedlove, Wesneski, a Dallas-based investment and merchant bank. Earlier in his career, Mr. Dodd was Senior Managing Director of the investment banking operations of Signet Banking Corporation and a Senior Director in the capital markets group of Continental Bank, where he focused on the capital needs of communications and media companies. Mr. Dodd holds an MBA from the University of Chicago and an AB from Cornell University.

Dennis Tucker
Chief Technology Officer

Mr. Tucker holds a BS in Computer Science from the University of North Texas and an AS in Machine Tool Technology. Mr. Tucker has been working with computers of various forms for over twenty years. Mr. Tucker has owned several businesses in the past, including an ISP. Mr. Tucker also has held positions with other companies as a project manager, supervisor, software engineer, and consultant. Mr. Tucker has been working in the field of compression and multimedia for over seven years. He is responsible for the design and direction of the Transfinity development effort.

Shin-Ping Liu
Chief Scientist and Project Manager

Ms Liu brings over ten years of experience in Information Technology, including compression technology, streaming video, distributed computing, and e-commerce. Ms. Liu is responsible for the R&D direction and project management of Transfinity. Prior to joining Transfinity, she was a multimedia Web designer in streaming video, audio, and synchronized multimedia presentation for a university special education institute. Ms Liu holds a patent and patent pending in the fields of compression implementation and content delivery systems. Ms Liu holds an MS in Computer Science from the University of North Texas (UNT) and a BBA from Tunghai University in Taiwan. She is a Ph.D. candidate in Information Science at the University of North Texas.

Robert Dempsey
Senior Software Engineer

Mr. Dempsey has 14 years software development experience in such diverse and technically challenging areas as message processing and high-reliability image archiving. At Transfinity, Mr. Dempsey has responsibility for the design and implementation of the Control Server, the Distribution Server, and the Compression Server front-end. He also maintains much of the development infrastructure such as the source code control system, problem tracking system, and the baseline OS installation and performance tuning of the Control and Distribution Servers. Prior to joining Transfinity, Mr. Dempsey was the development manager for medial image archives at Eastman Kodak Company where he had overall product development responsibility for a $1MM+ product line. Mr. Dempsey holds an electrical engineering degree from the University of Notre Dame and is a member of the IEEE and American Mensa Association.

4 Financial Projections

Financial Summary

	Q4-2000	Q1-2001	Q2-2001	Q3-2001	Q4-2001	Q1-2002	Q2-2002	Q3-2002	Q4-2002
Total Revenue		$1,171,875	$2,343,750	$3,515,625	$4,687,500	$5,859,375	$7,031,250	$8,203,125	$9,375,000
Cummulative Revenue		$1,171,875	$3,515,625	$7,031,250	$11,718,750	$17,578,125	$24,609,375	$32,812,500	$42,187,500
Total Expense	$ 1,132,628	$ 2,079,973	$ 2,055,243	$ 2,140,705	$ 2,276,933	$ 2,922,844	$ 2,805,603	$ 3,109,819	$ 3,500,078
Cummulative Expense	$ 1,132,628	$ 3,212,600	$ 5,267,843	$ 7,408,548	$ 9,685,480	$ 12,608,324	$ 15,413,926	$ 18,523,745	$ 22,023,823
Gross Profit/Loss	$ (1,132,628)	$ (908,098)	$ 288,508	$ 1,374,920	$ 2,410,568	$ 2,936,531	$ 4,225,648	$ 5,093,306	$ 5,874,923
Cummulative Profit/	$ (1,132,628)	$ (2,040,725)	$ (1,752,218)	$ (377,298)	$ 2,033,270	$ 4,969,801	$ 9,195,449	$ 14,288,755	$ 20,163,678

License Revenue for Transfinity

			2000	2001				2002			
Qtr	New Users	Total Users	Q4 Revenue	Q1 Revenue	Q2 Revenue	Q3 Revenue	Q4 Revenue	Q1 Revenue	Q2 Revenue	Q3 Revenue	Q4 Revenue
Q4-2000	0		$ -								
Q1-2001	390,625	390,625		$ 1,171,875							
Q2-2001	390,625	781,250			$ 2,343,750						
Q3-2001	390,625	1,171,875				$ 3,515,625					
Q4-2001	390,625	1,562,500					$ 4,687,500				
Q1-2002	390,625	1,953,125						$ 5,859,375			
Q2-2002	390,625	2,343,750							$ 7,031,250		
Q3-2002	390,625	2,734,375								$ 8,203,125	
Q4-2002	390,625	3,125,000									$ 9,375,000

Total Gross Revenue per Quarter	$0	$1,171,875	$2,343,750	$3,515,625	$4,687,500	$5,859,375	$7,031,250	$8,203,125	$9,375,000
Cummulative Revenue	$0	$1,171,875	$3,515,625	$7,031,250	$11,718,750	$17,578,125	$24,609,375	$32,812,500	$42,187,500

Assumptions:

1) Projections
Fiscal year equals calendar year
Projections are $ based, not accrual based

2) User Fees are based upon a license fee of $1.00 per month per Internet user.
No income assumed from content delivery entities; only from ISPs or ASPs.
Constant number of active U.S. Internet users at 125 million of which 75% are users at home (ISP) and 25% are users at work (ASP).
Transfinity captures 0.3125% of that market per quarter beginning Q1-2001 with a total of 2.5% of the active U.S. users at the end of Q4-2002.

Transfinity Corporation
Total Operational Expenses

	Q4-2000	Q1-2001	Q2-2001	Q3-2001	Q4-2001	Q1-2002	Q2-2002	Q3-2002	Q4-2002
Current Employees	9	17	23	34	39	41	42	42	42
New Employees	8	6	11	5	2	1	0	0	0
Total Employees	17	23	34	39	41	42	42	42	42
Current Employee Salary	$ 350,000	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250
New Employee Salary	$ 222,500	$ 150,000	$ 262,500	$ 130,000	$ 55,000	$ 204,250	$ -	$ -	$ -
Total Employee Salary	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250	$ 1,374,250
Staffing									
Headhunter/Other Recruiting Costs (1)	$ 48,500	$ 33,000	$ 58,000	$ 28,500	$ 12,000	$ 41,350	$ -	$ -	$ -
Moving (2)	$ 8,000	$ 6,000	$ 11,000	$ 5,000	$ 2,000	$ 1,000	$ -	$ -	$ -
Signing Bonus (3)	$ 66,750	$ 45,000	$ 145,500	$ 84,000	$ 95,250	$ 100,275	$ 16,500	$ 61,275	$ -
Staffing Total	$ 123,250	$ 84,000	$ 214,500	$ 117,500	$ 109,250	$ 142,625	$ 16,500	$ 61,275	$ -
Salaries and Benefits									
Health Insurance (4)	$ 22,950	$ 31,050	$ 45,900	$ 52,650	$ 55,350	$ 56,700	$ 56,700	$ 56,700	$ 56,700
Life Insurance (5)	$ 2,040	$ 2,760	$ 4,080	$ 4,680	$ 4,920	$ 5,040	$ 5,040	$ 5,040	$ 5,040
Payroll Tax and Other expenses (6)	$ 95,888	$ 115,125	$ 169,575	$ 179,850	$ 189,788	$ 221,179	$ 208,613	$ 215,329	$ 206,138
Salaries (7)	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250	$ 1,374,250
Cell Phone (8)	$ 6,225	$ 6,975	$ 10,950	$ 10,275	$ 9,825	$ 9,750	$ 9,450	$ 9,450	$ 9,450
Seminar and Continuing Ed (9)	$ 8,500	$ 11,500	$ 17,000	$ 19,500	$ 20,500	$ 21,000	$ 21,000	$ 21,000	$ 21,000
Texas Workmens Comp (10)	$ 4,250	$ 5,750	$ 8,500	$ 9,750	$ 10,250	$ 10,500	$ 10,500	$ 10,500	$ 10,500
Salaries and Benefits Total	$ 712,353	$ 895,660	$ 1,241,005	$ 1,391,705	$ 1,460,633	$ 1,698,419	$ 1,685,553	$ 1,692,269	$ 1,683,078
Cost of Goods									
Server Hosting Expense (11)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
License Fees (13)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
Decision Software & Support (14)	$ -	$ 600,000		$ -	$ -	$ 100,000	$ -	$ -	$ -
Cost of Goods Total	$ -	$ 600,000	$ -	$ -	$ -	$ 100,000	$ -	$ -	$ -
Overhead									
Marketing (18)	$ -	$ 20,438	$ 34,063	$ 40,875	$ 40,875	$ 43,125	$ 71,875	$ 86,250	$ 86,250
Market Research (16)	$ -	$ 40,000	$ 10,000	$ 4,000	$ -	$ 44,000	$ 10,000	$ 4,000	$ -
Accounting (19)	$ 19,000	$ 9,000	$ 9,000	$ 9,000	$ 19,000	$ 9,000	$ 9,000	$ 9,000	$ 19,000
Banking (20)	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300
Customer Support (21)	$ -	$ 225	$ 450	$ 675	$ 675	$ 1,350	$ 2,250	$ 2,700	$ 2,925
Delivery Charges (22)	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300
Human Resources (23)	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500
Legal (24)	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000
Liability/Property Insurance (25)	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650
Online Ad Serving (26)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
Postage (27)	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150
Subscriptions, Dues and Books (28)	$ 1,275	$ 1,725	$ 2,550	$ 2,925	$ 3,075	$ 3,150	$ 3,150	$ 3,150	$ 3,150
Supplies (29)	$ 1,275	$ 1,725	$ 2,550	$ 2,925	$ 3,075	$ 3,150	$ 3,150	$ 3,150	$ 3,150
Travel (30)	$ 153,000	$ 261,000	$ 315,000	$ 396,000	$ 481,500	$ 720,000	$ 859,500	$ 1,102,500	$ 1,557,000
Overhead Total	$ 193,450	$ 353,013	$ 392,513	$ 475,300	$ 567,100	$ 842,675	$ 977,825	$ 1,229,650	$ 1,690,375
Office Expense (32)									
Office Space Rent	$ 15,000	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175
Workstations	$ 40,000	$ 30,000	$ 55,000	$ 25,000	$ 10,000	$ 5,000	$ -	$ -	$ -
Servers	$ -	$ 1,000	$ 11,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000
Phone System	$ 4,075	$ 2,725	$ 3,550	$ 4,025	$ 4,475	$ 4,550	$ 4,550	$ 4,550	$ 4,550
Network	$ 6,000	$ -	$ 1,500	$ -	$ 1,500	$ -	$ -	$ -	$ -
Misc.	$ 38,500	$ 36,400	$ 59,000	$ 49,000	$ 45,800	$ 51,400	$ 43,000	$ 43,900	$ 43,900
Office Expense Total	$ 103,575	$ 147,300	$ 207,225	$ 156,200	$ 139,950	$ 139,125	$ 125,725	$ 126,625	$ 126,625
Total Quarterly Expenses	$ 1,132,628	$ 2,079,973	$ 2,055,243	$ 2,140,705	$ 2,276,933	$ 2,922,844	$ 2,805,603	$ 3,109,819	$ 3,500,078
Cummulative Expenses	$ 1,132,628	$ 3,212,600	$ 5,267,843	$ 7,408,548	$ 9,685,480	$ 12,608,324	$ 15,413,926	$ 18,523,745	$ 22,023,823

Assumptions:

(1) Recruiting Expense

30% of personnel use headhunter and cost is

25% of salary

Other recruiting costs include advertising, online posting, job fair, etc and are

$ 500 per New Employee

(2) Moving Expense

$ 5,000 for every employee requiring relocation

20% of employees require relocation

(3) Signing Bonus Expense

15% of New Employee Salaries (50% paid at signing, 50% deferred 6 months)

Transfinity Corporation
Total Operational Expenses

	Q4-2000	Q1-2001	Q2-2001	Q3-2001	Q4-2001	Q1-2002	Q2-2002	Q3-2002	Q4-2002
Current Employees	9	17	23	34	39	41	42	42	42
New Employees	8	6	11	5	2	1	0	0	0
Total Employees	17	23	34	39	41	42	42	42	42
Current Employee Salary	$ 350,000	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250
New Employee Salary	$ 222,500	$ 150,000	$ 262,500	$ 130,000	$ 55,000	$ 204,250	$ -	$ -	$ -
Total Employee Salary	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250	$ 1,374,250
Staffing									
Headhunter/Other Recruiting Costs (1)	$ 48,500	$ 33,000	$ 58,000	$ 28,500	$ 12,000	$ 41,350	$ -	$ -	$ -
Moving (2)	$ 8,000	$ 6,000	$ 11,000	$ 5,000	$ 2,000	$ 1,000	$ -	$ -	$ -
Signing Bonus (3)	$ 66,750	$ 45,000	$ 145,500	$ 84,000	$ 95,250	$ 100,275	$ 16,500	$ 61,275	$ -
Staffing Total	$ 123,250	$ 84,000	$ 214,500	$ 117,500	$ 109,250	$ 142,625	$ 16,500	$ 61,275	$ -
Salaries and Benefits									
Health Insurance (4)	$ 22,950	$ 31,050	$ 45,900	$ 52,650	$ 55,350	$ 56,700	$ 56,700	$ 56,700	$ 56,700
Life Insurance (5)	$ 2,040	$ 2,760	$ 4,080	$ 4,680	$ 4,920	$ 5,040	$ 5,040	$ 5,040	$ 5,040
Payroll-Tax and Other expenses (6)	$ 95,888	$ 115,125	$ 169,575	$ 179,850	$ 189,788	$ 221,179	$ 208,613	$ 215,329	$ 206,138
Salaries (7)	$ 572,500	$ 722,500	$ 985,000	$ 1,115,000	$ 1,170,000	$ 1,374,250	$ 1,374,250	$ 1,374,250	$ 1,374,250
Cell Phone (8)	$ 6,225	$ 6,975	$ 10,950	$ 10,275	$ 9,825	$ 9,750	$ 9,450	$ 9,450	$ 9,450
Seminar and Continuing Ed (9)	$ 8,500	$ 11,500	$ 17,000	$ 19,500	$ 20,500	$ 21,000	$ 21,000	$ 21,000	$ 21,000
Texas Workmens Comp (10)	$ 4,250	$ 5,750	$ 8,500	$ 9,750	$ 10,250	$ 10,500	$ 10,500	$ 10,500	$ 10,500
Salaries and Benefits Total	$ 712,353	$ 895,660	$ 1,241,005	$ 1,391,705	$ 1,460,633	$ 1,698,419	$ 1,685,553	$ 1,692,269	$ 1,683,078
Cost of Goods									
Server Hosting Expense (11)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
License Fees (13)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
Decision Software & Support (14)	$ -	$ 600,000		$ -	$ -	$ 100,000	$ -	$ -	$ -
Cost of Goods Total	$ -	$ 600,000	$ -	$ -	$ -	$ 100,000	$ -	$ -	$ -
Overhead									
Marketing (18)	$ -	$ 20,438	$ 34,063	$ 40,875	$ 40,875	$ 43,125	$ 71,875	$ 86,250	$ 86,250
Market Research (16)	$ -	$ 40,000	$ 10,000	$ 4,000	$ -	$ 44,000	$ 10,000	$ 4,000	$ -
Accounting (19)	$ 19,000	$ 9,000	$ 9,000	$ 9,000	$ 19,000	$ 9,000	$ 9,000	$ 9,000	$ 19,000
Banking (20)	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300
Customer Support (21)	$ -	$ 225	$ 450	$ 675	$ 675	$ 1,350	$ 2,250	$ 2,700	$ 2,925
Delivery Charges (22)	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300	$ 300
Human Resources (23)	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500	$ 1,500
Legal (24)	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000	$ 15,000
Liability/Property Insurance (25)	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650	$ 1,650
Online Ad Serving (26)	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -
Postage (27)	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150	$ 150
Subscriptions, Dues and Books (28)	$ 1,275	$ 1,725	$ 2,550	$ 2,925	$ 3,075	$ 3,150	$ 3,150	$ 3,150	$ 3,150
Supplies (29)	$ 1,275	$ 1,725	$ 2,550	$ 2,925	$ 3,075	$ 3,150	$ 3,150	$ 3,150	$ 3,150
Travel (30)	$ 153,000	$ 261,000	$ 315,000	$ 396,000	$ 481,500	$ 720,000	$ 859,500	$ 1,102,500	$ 1,557,000
Overhead Total	$ 193,450	$ 353,013	$ 392,513	$ 475,300	$ 567,100	$ 842,675	$ 977,825	$ 1,229,650	$ 1,690,375
Office Expense (32)									
Office Space Rent	$ 15,000	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175	$ 77,175
Workstations	$ 40,000	$ 30,000	$ 55,000	$ 25,000	$ 10,000	$ 5,000	$ -	$ -	$ -
Servers	$ -	$ 1,000	$ 11,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000	$ 1,000
Phone System	$ 4,075	$ 2,725	$ 3,550	$ 4,025	$ 4,475	$ 4,550	$ 4,550	$ 4,550	$ 4,550
Network	$ 6,000	$ -	$ 1,500	$ -	$ 1,500	$ -	$ -	$ -	$ -
Misc.	$ 38,500	$ 36,400	$ 59,000	$ 49,000	$ 45,800	$ 51,400	$ 43,000	$ 43,900	$ 43,900
Office Expense Total	$ 103,575	$ 147,300	$ 207,225	$ 156,200	$ 139,950	$ 139,125	$ 125,725	$ 126,625	$ 126,625
Total Quarterly Expenses	$ 1,132,628	$ 2,079,973	$ 2,055,243	$ 2,140,705	$ 2,276,933	$ 2,922,844	$ 2,805,603	$ 3,109,819	$ 3,500,078
Cummulative Expenses	$ 1,132,628	$ 3,212,600	$ 5,267,843	$ 7,408,548	$ 9,685,480	$ 12,608,324	$ 15,413,926	$ 18,523,745	$ 22,023,823

Assumptions:

(1)	Recruiting Expense	
	20%	of personnel use headhunter and cost is
	25%	of salary
	Other recruiting costs include advertising, online posting, job fair, etc and are	
	$ 500	per New Employee
(2)	Moving Expense	
	$ 5,000	for every employee requiring relocation
	20%	of employees require relocation
(3)	Signing Bonus Expense	
	15%	of New Employee Salaries (50% paid at signing, 50% deferred 6 months)

Transfinity Corporation
Total Operational Expenses

(4)	Health Insurance Expense		
	$	500	per person per month
		90%	of premium is subsidized by company
(5)	Life Insurance Expense		
	$	40	per person per month
(6)	Payroll Tax and Other Expenses		
		15%	of Salaries Expense
(7)	Salaries Expense		
	See Salaries Expense Worksheet for detail		
(8)	Cell Phone Expense		
	$	75	per month per person
	$	300	per new employee
(9)	Seminars and Continuing Education		
	$	2,000	per person per year
(10)	Workman's Compensation		
	$	1,000	per person per year
(11)	Server Hosting Expense		
	See Server Hosting Expense Worksheet for detail. This is a placeholder only		
(13)	License Fees		
			This is a placeholder only
(14)	Decision Software & Support		
	$	500,000	One-time purchase of decision software with support
		20%	annual maintenance cost
(16)	Market Research		
	$	40,000	Purchase of outside studies (I.e. Forrester) per year
	$	10,000	per Focus Group
	$	4,000	per Usability Test
(17)	Online Marketing		
	$	0.50	per transaction accrued for Online Marketing
(18)	Marketing		
		2.50%	of Total Revenue (Trade Mag Ads, Trade Shows, etc.)
(19)	Accounting Expense		
	$	3,000	Outsorced Accounting Expense per month
	$	10,000	Yearly Audit Expense
(20)	Banking Expense		
	$	100	per month
(21)	Customer Support Expense		
	$	75	per user for ASP license per month
			users equal # personnel in Customer Support
	See Call Center Expense Worksheet for detail. This is a placeholder only		
(22)	Delivery Expense		
	$	200	per month
(23)	Human Resources		
	$	500	Outsourced portions of Human Resources per month (e.g. payroll, benefits mgmt, etc.)
(24)	Legal Expense		
	$	15,000	per quarter
(25)	Liabilty/Property Insurance Expense		
	$	500	per month
(26)	Online Ad Serving Expense		
	See Online Ad Serving Expense Worksheet for detail. This is a placeholder only		
(27)	Postage Expense		
	$	80	per month
(28)	Subscription, Dues and Books		
	$	300	per person per year
(29)	Supplies Expense		
	$	25	per person per month
(30)	Travel Expense		
		2	trips per month for "management"
	$	1,500	per trip average cost
	$	8	trips per month per "traveler" (e.g. sales, professional services, etc.)
	$	1,500	per trip average cost
			See "Salaries Expense" for managemtnt/traveler designation
(32)	Office Expense		

Transfinity Corporation
Salary Projections

														1																								
		Q4-2000				Q1-2001				Q2-2001				Q3-2001				Q4-2001					Q1-2002				Q2-2002				Q3-2002				Q4-2002			
Employee Position	Starting Quarterly Salary	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Increased Quarterly Salaries (15% increase)	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries	Current	New	Total	Total Salaries
Executives																																						
CEO	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	$57,500	1		1	$57,500	1		1	$57,500	1		1	$57,500	1		1	$57,500
Sr. VP -- Technology	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	1		1	$50,000	$57,500	1		1	$57,500	1		1	$57,500	1		1	$57,500	1		1	$57,500
CFO	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	$51,750	1		1	$51,750	1		1	$51,750	1		1	$51,750	1		1	$51,750
Comptroller	$20,000			0	$0			0	$0		1	1	$20,000	1		1	$20,000	1		1	$20,000	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000
Executive Secretary	$12,500			0	$0			0	$0		1	1	$12,500	1		1	$12,500	1		1	$12,500	$14,375	1		1	$14,375	1		1	$14,375	1		1	$14,375	1		1	$14,375
TOTAL -EXECUTIVE		3	0	3	$145,000	3	0	3	$145,000	3	2	5	$177,500	5	0	5	$177,500	5	0	5	$177,500	$204,125	5	0	5	$204,125	5	0	5	$204,125	5	0	5	$204,125	5	0	5	$204,125
Marketing and Sales																																						
VP-Sales/Marketing	$40,000		1	1	$40,000	1		1	$40,000	1		1	$40,000	1		1	$40,000	1		1	$40,000	$46,000	1		1	$46,000	1		1	$46,000	1		1	$46,000	1		1	$46,000
Professional-Mkt,PR	$17,500		1	1	$17,500	1		1	$17,500	1		1	$17,500	1		1	$17,500	1		1	$17,500	$20,125	1		1	$20,125	1		1	$20,125	1		1	$20,125	1		1	$20,125
Professional-Sales Executive	$30,000			0	$0		1	1	$30,000	1	2	3	$90,000	3	1	4	$120,000	4	1	5	$150,000	$34,500	5		5	$172,500	5		5	$172,500	5		5	$172,500	5		5	$172,500
Professional-Sales Support	$20,000			0	$0			0	$0		1	1	$20,000	1		1	$20,000	1		1	$20,000	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000
TOTAL -SALES/MARKETING		0	2	2	$57,500	2	1	3	$87,500	3	3	6	$167,500	6	1	7	$197,500	7	1	8	$227,500		8	0	8	$261,625	8	0	8	$261,625	8	0	8	$261,625	8	0	8	$261,625
Technology Management																																						
CTO	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	1		1	$45,000	$51,750	1		1	$51,750	1		1	$51,750	1		1	$51,750	1		1	$51,750
Chief Scientist	$40,000	1		1	$40,000	1		1	$40,000	1		1	$40,000	1		1	$40,000	1		1	$40,000	$46,000	1		1	$46,000	1		1	$46,000	1		1	$46,000	1		1	$46,000
TOTAL TECHNOLOGY MANAGEMENT		2	0	2	$85,000	2	0	2	$85,000	2	0	2	$85,000	2	0	2	$85,000	2	0	2	$85,000	$97,750	2	0	2	$97,750	2	0	2	$97,750	2	0	2	$97,750	2	0	2	$97,750
Content Delivery System Team																																						
Project Manager	$35,000	1		1	$35,000	1		1	$35,000	1		1	$35,000	1		1	$35,000	1		1	$35,000	$40,250	1		1	$40,250	1		1	$40,250	1		1	$40,250	1		1	$40,250
Sr. Developer	$30,000	2	2	4	$120,000	4	2	6	$180,000	6	3	9	$270,000	9	1	10	$300,000	10		10	$300,000	$34,500	10		10	$345,000	10		10	$345,000	10		10	$345,000	10		10	$345,000
Manager-Documentation	$25,000		1	1	$25,000	1		1	$25,000	1		1	$25,000	1		1	$25,000	1		1	$25,000	$28,750	1		1	$28,750	1		1	$28,750	1		1	$28,750	1		1	$28,750
Manager-QA	$30,000		1	1	$30,000	1		1	$30,000	1		1	$30,000	1		1	$30,000	1		1	$30,000	$34,500	1		1	$34,500	1		1	$34,500	1		1	$34,500	1		1	$34,500
TOTAL CONTENT DELIVERY		3	4	7	$210,000	7	2	9	$270,000	9	3	12	$360,000	12	1	13	$390,000	13	0	13	$390,000	$138,000	13	0	13	$448,500	13	0	13	$448,500	13	0	13	$448,500	13	0	13	$448,500
Technical Support Team																																						
Manager	$30,000		1	1	$30,000	1		1	$30,000	1		1	$30,000	1		1	$30,000	1		1	$30,000	$34,500	1		1	$34,500	1		1	$34,500	1		1	$34,500	1		1	$34,500
Consultant	$25,000			0	$0		1	1	$25,000	1	2	3	$75,000	3	2	5	$125,000	5	1	6	$150,000	$28,750	6	1	7	$201,250	7		7	$201,250	7		7	$201,250	7		7	$201,250
TOTAL TECH SUPPORT		0	1	1	$30,000	1	1	2	$55,000	2	2	4	$105,000	4	2	6	$155,000	6	1	7	$180,000	$63,250	7	1	8	$235,750	8	0	8	$235,750	8	0	8	$235,750	8	0	8	$235,750
Operations																																						
Manager-HR	$20,000		1	1	$20,000	1		1	$20,000	1		1	$20,000	1		1	$20,000	1		1	$20,000	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000	1		1	$23,000
Manager-Office	$15,000			0	$0		1	1	$15,000	1		1	$15,000	1		1	$15,000	1		1	$15,000	$17,250	1		1	$17,250	1		1	$17,250	1		1	$17,250	1		1	$17,250
Admin. Assistant	$10,000			0	$0			0	$0		1	1	$10,000	1		1	$10,000	1		1	$10,000	$11,500	1		1	$11,500	1		1	$11,500	1		1	$11,500	1		1	$11,500
Manager-IT	$25,000	1		1	$25,000	1		1	$25,000	1		1	$25,000	1		1	$25,000	1		1	$25,000	$28,750	1		1	$28,750	1		1	$28,750	1		1	$28,750	1		1	$28,750
System Administrator-IT	$20,000			0	$0		1	1	$20,000	1		1	$20,000	1	1	2	$40,000	2		2	$40,000	$23,000	2		2	$46,000	2		2	$46,000	2		2	$46,000	2		2	$46,000
TOTAL-OPERATIONS		1	1	2	$45,000	2	2	4	$80,000	4	1	5	$90,000	5	1	6	$110,000	6	0	6	$110,000	$103,500	6	0	6	$126,500	6	0	6	$126,500	6	0	6	$126,500	6	0	6	$126,500
Total Employees, Salary		9	8	17	$572,500	17	6	23	$722,500	23	11	34	$985,000	34	5	39	$1,115,000	39	2	41	$1,170,000		41	1	42	$1,374,250	42	0	42	$1,374,250	42	0	42	$1,374,250	42	0	42	$1,374,250
Cumulative Salary					$572,500				$1,295,000				$2,280,000				$3,395,000				$4,565,000					$5,939,250				$7,313,500				$8,687,750				$10,062,000

Transfinity Corporation
Office Expense

	Q4-2000	Q1-2001	Q2-2001	Q3-2001	Q4-2001	Q1-2002	Q2-2002	Q3-2002	Q4-2002
Current Employees	9	17	23	34	39	41	42	42	42
New Employees	8	6	11	5	2	1	0	0	0
Total Employees	17	23	34	39	41	42	42	42	42
Office Space Rent (1)	15,000	77,175	77,175	77,175	77,175	77,175	77,175	77,175	77,175
Workstations (2)	8	6	11	5	2	1	0	0	0
Workstation Expense	24,000	18,000	33,000	15,000	6,000	3,000	0	0	0
Upgrades (HW & SW)	8,000	6,000	11,000	5,000	2,000	1,000	0	0	0
Maintenance (HW & SW)	8,000	6,000	11,000	5,000	2,000	1,000	0	0	0
Worksation Total	40,000	30,000	55,000	25,000	10,000	5,000	0	0	0
Office & Development Servers (3)	0	0	1	0	0	0	0	0	0
Server Expense	-	-	10,000	-	-	-	-	-	-
Upgrades (HW & SW)	-	333	333	333	333	333	333	333	333
Maintenance (HW & SW)	-	667	667	667	667	667	667	667	667
Server Total	-	1,000	11,000	1,000	1,000	1,000	1,000	1,000	1,000
Phone System (4)									
Phone switch (leased)	2,500	700	700	700	900	900	900	900	900
Lines (Local)	300	300	300	400	500	500	500	500	500
Lines (Long Distance)	1,275	1,725	2,550	2,925	3,075	3,150	3,150	3,150	3,150
Phone System Total	4,075	2,725	3,550	4,025	4,475	4,550	4,550	4,550	4,550
Local Area Network (5)	6,000	-	1,500	-	1,500	-	-	-	-
Network Total	6,000	-	1,500	-	1,500	-	-	-	-
Miscellaneous Expenses									
Furniture (7)	$16,000	$12,000	$22,000	$10,000	$4,000	$2,000	$0	$0	$0
Printers	$0	$1,000	$3,000	$0	$300	$1,000	$1,000	$1,000	$1,000
Copy machine (8)	$5,500	$0	$0	$0	$500	$5,500	$0	$0	$500
Fax Machine (9)	$0	$0	$0	$0	$0	$0	$0	$0	$0
Scanner (10)	$0	$0	$0	$0	$0	$0	$0	$0	$0
Refrigerator (11)	$0	$0	$0	$0	$0	$500	$0	$500	$0
Microwave (12)	$0	$400	$0	$0	$0	$400	$0	$400	$400
Other (13)	$17,000	$23,000	$34,000	$39,000	$41,000	$42,000	$42,000	$42,000	$42,000
Miscellaneous Total	$38,500	$36,400	$59,000	$49,000	$45,800	$51,400	$43,000	$43,900	$43,900
Subtotal by quarter	**$103,575**	**$147,300**	**$207,225**	**$156,200**	**$139,950**	**$139,125**	**$125,725**	**$126,625**	**$126,625**
Running total	**$103,575**	**$250,875**	**$458,100**	**$614,300**	**$754,250**	**$893,375**	**$1,019,100**	**$1,145,725**	**$1,272,350**

Assumptions:

	(1) Office Space Rent Expense
	Office space - Original space requirements based on needs at after Quarter 9
350	Amount of Square Footage required for each employee
$ 21	Cost per square foot per year
42	9th Quarter Employee Total
	(2) Work Stations Expense
$ 3,000	Cost of workstations per employee
1/3	Hardware & Software Upgrades cost as a percentage of Workstation cost
1/3	Hardware & Software Maintenance cost as a percentage of Workstation cost
	(3) Office & Development Server Expense
$ 10,000	Average cost per server
1/3	Hardware & Software Upgrades cost as a percentage of Server cost
2/3	Hardware & Software Maintenance cost as a percentage of Server cost
10	Quarters to amortise Upgrade & Maintenance Expense
4	Minimum server requirement for Office - 20 employee capacity
9	Minimum server requirement for Development
8	Employees per server after minimum
	(4) Cost of Phone System

Transfinity Corporation
Office Expense

$	2,000	Phone Switch Installation (capacity 20 employees)
$	500	Phone Switch Per Quarter Usage
$	200	Phone Switch System Upgrade every 20 employees
$	300	Local Lines Usage Per Quarter
$	100	Local Lines Additional Usage Charge Per 20 Employees
$	75	Long Distance Charges Per Quarter per Employee
		(5) Cost of Local Area Network
$	3,000	LAN Installation (16-Port Hub, Cable Plant) for 10 employees
	50%	Additional LAN installation for each 10 employees (Will base installation cost off of Q9 employee total)
		(6) Cost of Wide Area Network
$	200	Labor Cost for Installation of Wide Area Network
$	3,000	Hardware (Routers, etc.) for each 5 employees
$	1,000	Upgrade system after every 30 employees
$	350	DSL & ISDN Connections per month
		Miscellaneous Expenses
$	2,000	(7) Cost of furniture per set
		Printers
	30	Number of non-management employees per printer
	1000	Cost per printer for non-management
	1	Number of senior management employees per printer (exec + VP)
	300	Cost per printer per management
	3000	Cost of color printer for Marketing Department
		(8) Copy Machine
	75	Number of employee per copy machine
$	5,000	Cost per copy machine
$	500	Maintenance per year for copy machine
	36 months	Life of copy machine
$	10,000	Cost of Color copier for Marketing Department
		(9) Fax Machine
	50	Number of employees per fax machine
$	500	Cost per fax machine
	36 months	Life of fax machine
		(10) Scanner (one scanner for office)
$	500	Cost per scanner
	36 months	Life of scanner
		(11) Refrigerator
	50	Number of employees per refrigerator
$	500	Cost per refrigerator
	36 months	Life of refrigerator
		(12) Microwave
	30	Number of employees per microwave
$	400	Cost per microwave
	36 months	Life of microwave
$	1,000	(13) Other Miscellaneous Expenses per employee

Appendix: Transfinity's N-Bit Compression Technology

Transfinity's compression technology differs from all other compression technologies in its ability to compress previously compressed data. The possibility that previously compressed data can be compressed again and again is greeted by most computer scientists with incredulity. How is it possible that information theory is wrong in its fundamental assumption that there is a calculable value that represents the absolute lower limit of a message's meaning as measured in bits? Information science has been based on the implicit assumption that the primary unit of information is the byte (i.e. eight bits). Transfinity's technology is based on the assumption that the primary unit of information is not the byte, but rather the binary digit or bit. This simple assumption has allowed Transfinity to make major advances in numerous areas within computing including encryption, compression, and arbitrary precision mathematics.

Transfinity's compression is protected by issued patents and patents pending. N-bit compression provides significant increases in compression over other lossless compression methods. Representative examples of compression ratios obtained by Transfinity with its technologies are as follows:

Lossless Compression		Lossy Compression		Lossless & Lossy Compression	
DICOM	62%	JPEG(Our Algorithm)	71%	Special Medical Image	88%
AVI	60%				
GIF	30%				
ZIP	18%				
AVERAGE	**42.5%**				

There are numerous compression standards in the telecommunications, networking, and Internet communities. Organizations such as the Internet Engineering Task Force (IETF), the International Telecommunications Union (ITU), the International Telegraph and Telephone Consultative Committee (CCITT), the International Standards Organization (ISO), and the World Wide Web Consortium (W3C) support many compression practices and standards. Any encryption and/or compression products introduced into the market must accommodate both existing and emerging standards.

Transfinity N-bit™ compression is not intended to replace or compete with existing standards. It is not a new standard. It is a breakthrough technology that will have wide-ranging effects across multiple commercial markets and industry segments. It has applications related to both bandwidth and storage. It can be implemented in hardware using both conventional RISC and DSP processor architectures. It can be implemented as software. N-bit™ compression has application in areas as diverse as telecommunications, networking, content and file management, and disk and tape storage.

A1 N-bit™ Compression– Theoretical Background

Information theory states that there is a limit to the degree to which data can be compressed based upon its entropy. The assumption is that the content or meaning of the data is somehow contained within the digital representation of the data. A bit string has little or no meaning in and of itself. Very little of the meaning of any data is contained within its binary representation. The large majority of its meaning is extrinsic. By transferring more and more meaning or content to the external source or model that represents the data, greater and greater lossless compression is possible.

The entropy of a given data set may be changed in several ways. One way is to vary the length in bits of the input symbols to the model. This, in effect, changes the model for each symbol length. So, in practice, the model can be varied until desired entropy is obtained. Changing the model to find the desired entropy allows data that was previously uncompressible to be compressed. Data compressed at some symbol size will reach an entropy limit and converge at the calculated probabilities for that data set. To change the entropy limit and the calculated probabilities, one has only to change the model.

The frequency with which the symbols occur in a given symbol set may also be changed by applying one or more rules that modify the binary values of the input symbols prior to compression. A binary grammar may be defined to describe those data transformations that may be used to manipulate bit values within or between symbols.

The general method used to enable re-iterative N-bit™ lossless compression consists of the following steps:

1. The statistical analysis of the input source to determine optimal symbol size, the method or methods to be used to explicitly change the values of the symbols prior to compression, and the method or methods to be used to change the total number of symbols included in a compression pass.

2. The optimization of the compression process in terms of speed and efficiency based on the results of Step 1.

3. The compression of the data using one or more lossless compression methods that use variable bit length words to describe the symbols.

4. Repeat the process by using the output of the previous compression effort as input to the next.

In summary, since the entropy of a symbol is defined as the negative logarithm of the probability of its occurrence in a given symbol string, the base size of the binary-symbol set, the number of unique symbols, and the total number of symbols in the symbol string may be modified to optimize the compressibility of any source of binary input, regardless of whether that source of input is also the output of a previous compression process.

A2 The Core Product

Transfinity's content distribution technology is designed to increase bandwidth and storage capabilities and to decrease communication time on all networks using existing network hardware. This goal will be achieved by utilizing a combination of compression, proxy, firewall caching, and intelligent distribution servers.

Transfinity's Optimal™ Content Distribution System (TOCDS) consists of the following components:

- The Compression Server to manage compression tasks
- The Distribution Server to distribute the content
- The Control Server to manage the distribution servers
- Multiple sub-systems to manage caching, security, etc.

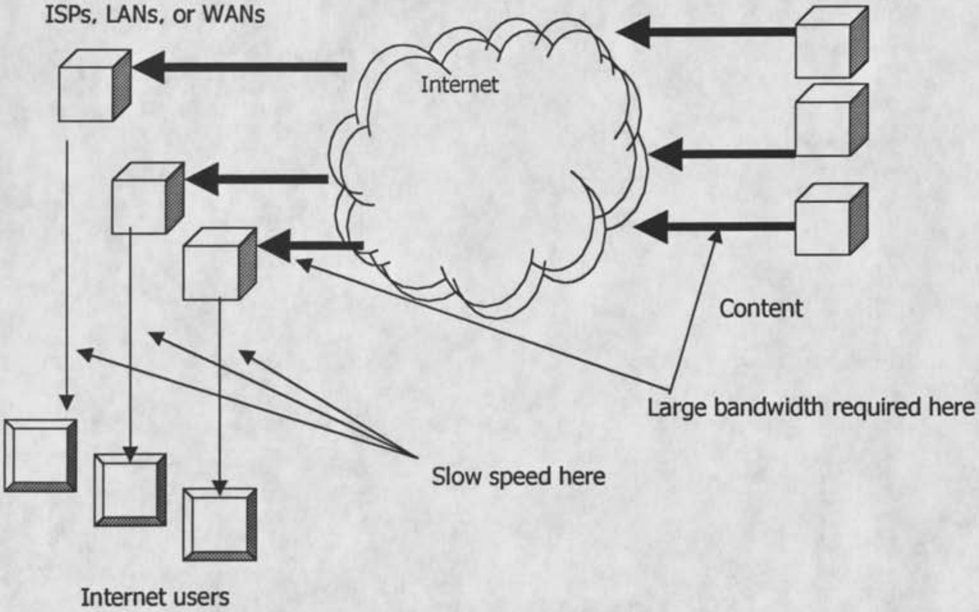
- Various plug-ins, applets, and applications to support the viewing and editing of specific media formats

The Compression Server and Compression Libraries are used in conjunction with any or all of Transfinity's other servers. The system is designed to intercept content on the network at a designated point and to compress and *cache* the content. The system also has the ability to intelligently distribute the content.

The general concept underlying TOCDS is to intercept content on the network at some point, compress, and cache the content. The intercept point may be the source or anywhere along the path, including the destination. The servers are all part of TOCDS, and all servers share all public content through the Control Server.

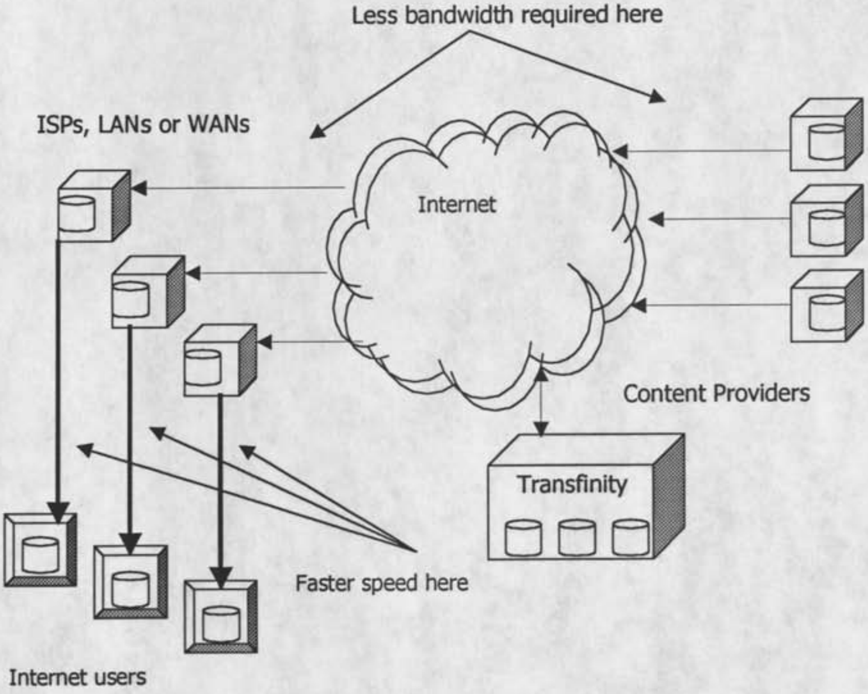
A diagram of the Internet without the TOCDS solution is provided in Figure 1 below:

Figure 1



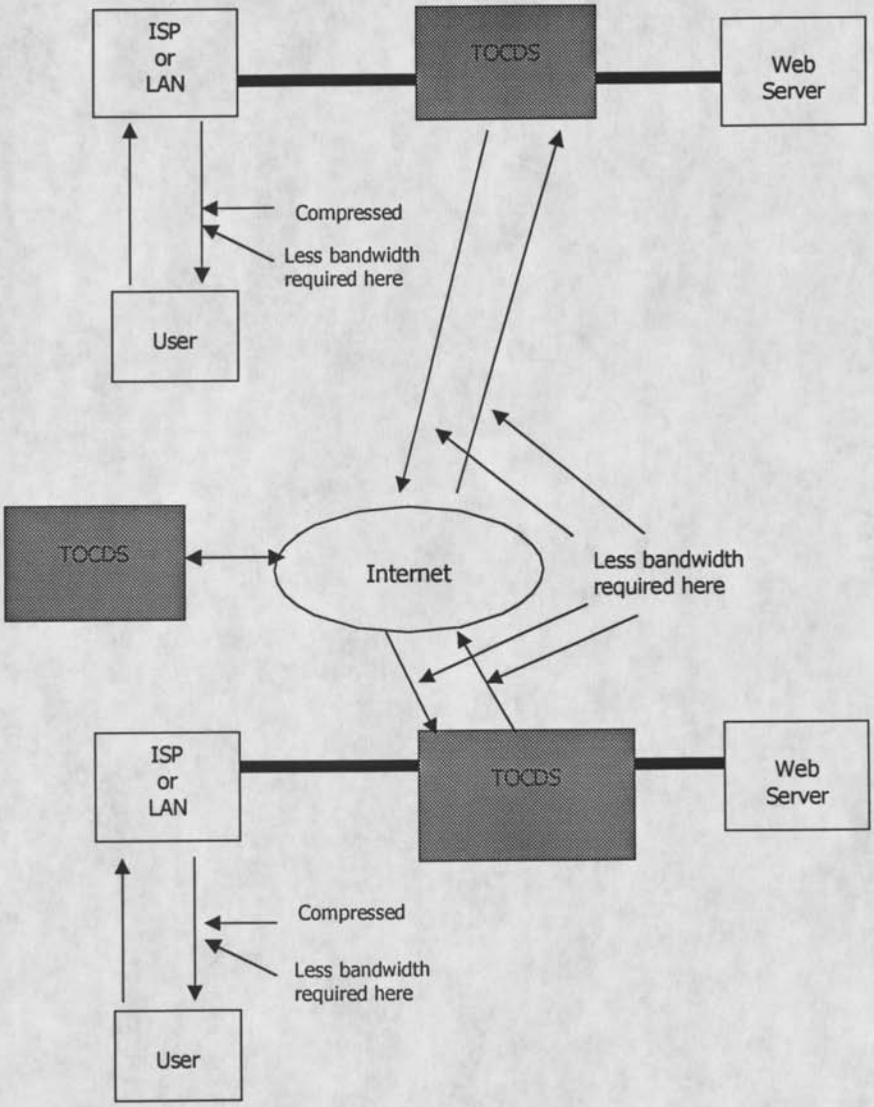
A diagram of the Internet using the TOCDS solution is shown in Figure 2 below:

Figure 2

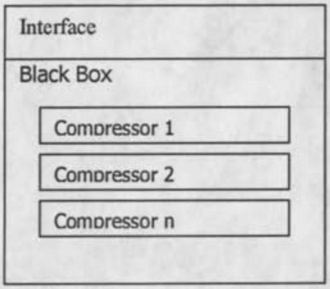


A diagram of the TOCDS general solution is provided in Figure 3 below:

Figure 3



The *Compression Server* contains Transfinity's patented and patent pending technology and is a core component of the product. The Compression Server's job is to compress the content captured by the servers. The Compression Server is intended to be treated as a "Black Box" by other programs and processes. The Compression Server can compress any type of content using any combination of lossy and lossless compression methods. The Compression Server can use various compression methods to adapt to changing content and new compression libraries can be plugged into the existing architecture when upgrades are required.



The Compression Server manages compression tasks from the other servers. Compression requests are made through RMI or TCP/IP, so that the servers may reside on different machines and networks if desired. The Compression Server manages the Compression Libraries through a native interface. It is the Compression Server's job to interpret compression requests, pass parameters to the Black Box Compressor, interpret the output from the compressor and reply to the requesting server.

The *Compression Libraries* are plug-and-play software components that provide different types of compression capabilities to the Compression Server. The Compression Libraries implement Transfinity's patented N-Bit ™ compression technology as well as other compression methods.

Compression Libraries will include:

- N-bit lossless compression
- JPEG, MPEG and fractal lossy compression
- Industry specific file compression (e.g. DICOM)
- Streaming media compression
- File conversion routines (e.g. converting .gif files to JPEG format)

The Compression Libraries will support both real-time, packet level compression in hardware as well as content specific compression in software.

From the users viewpoint, about a 50% increase in bandwidth will be possible through the use of compression technology, and about a 75% increase in speed will be possible through the use of compression and caching technology, with a small reduction in visual quality on some content. From the Service Providers viewpoint, large reductions in incoming network traffic due to the caching nature of TOCDS will result in large bandwidth savings. Their customers will experience increased apparent transmission speed and bandwidth.

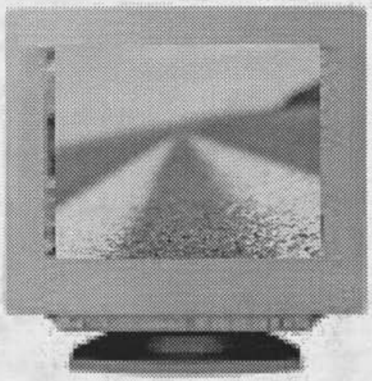
TRANSFINITY

Edgewidth Solutions for the Internet

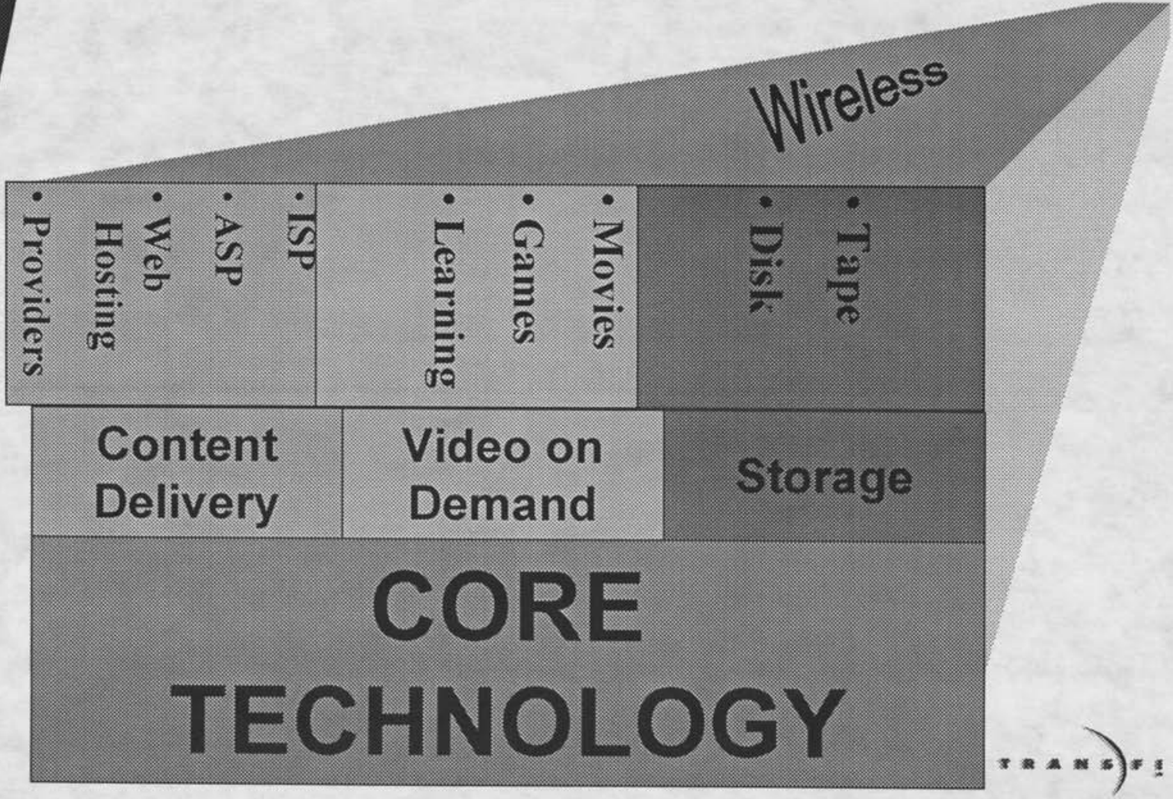
TRANSFINITY

Transfinity is an Internet infrastructure company that is using its patented technologies to target Internet Edgewidth opportunities that include:

- "Last Mile" Internet users
- Video on Demand
- Tape and Disk Storage



The core technology creates multiple revenue opportunities:



Wireless

- ISP
- ASP
- Web Hosting
- Providers

- Movies
- Games
- Learning

- Tape
- Disk

Content Delivery

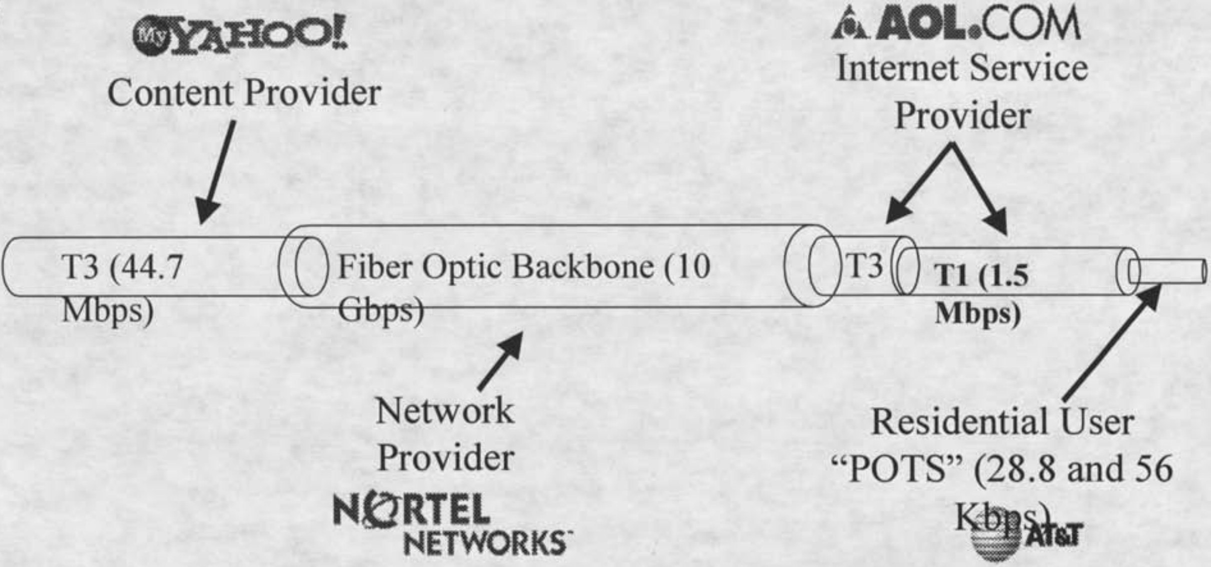
Video on Demand

Storage

CORE TECHNOLOGY

TRANSFINITY

Content Delivery -- The Internet's "last mile" bottleneck . . .



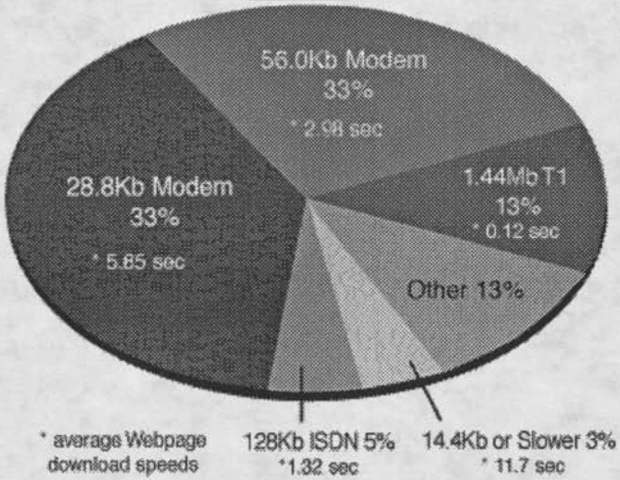
MyYAHOO!
Content Provider
T3 (44.7 Mbps)
Fiber Optic Backbone (10 Gbps)
Network Provider
NORTEL NETWORKS
AOL.COM
Internet Service Provider
T3
T1 (1.5 Mbps)
Residential User "POTS" (28.8 and 56 Kbps)
AT&T

Goes from **10 billion** bits per second down to **28 thousand** bits per second for most users.

TRANSFINITY

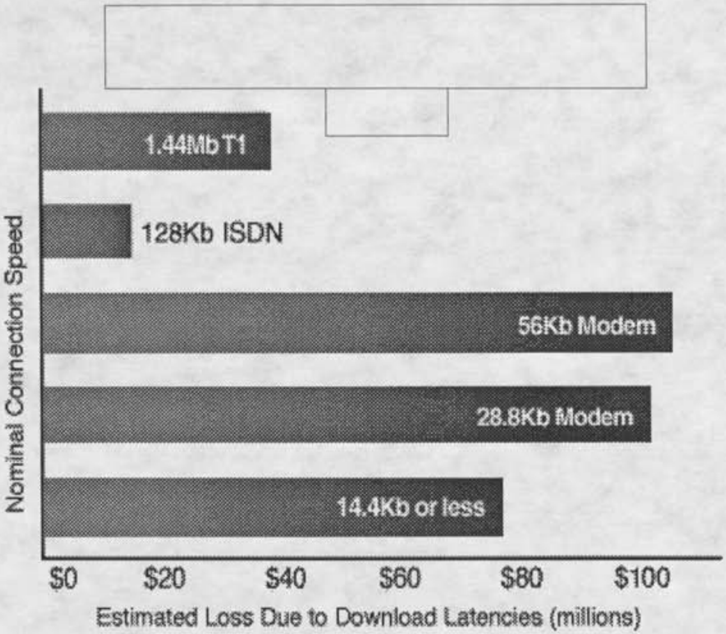
"Plain Old Telephone Service" (POTS) represents the majority of U.S. Internet users:

Estimated connection speeds of U.S. Internet users



Gilder Technology Report

Download latencies represent millions of lost dollars



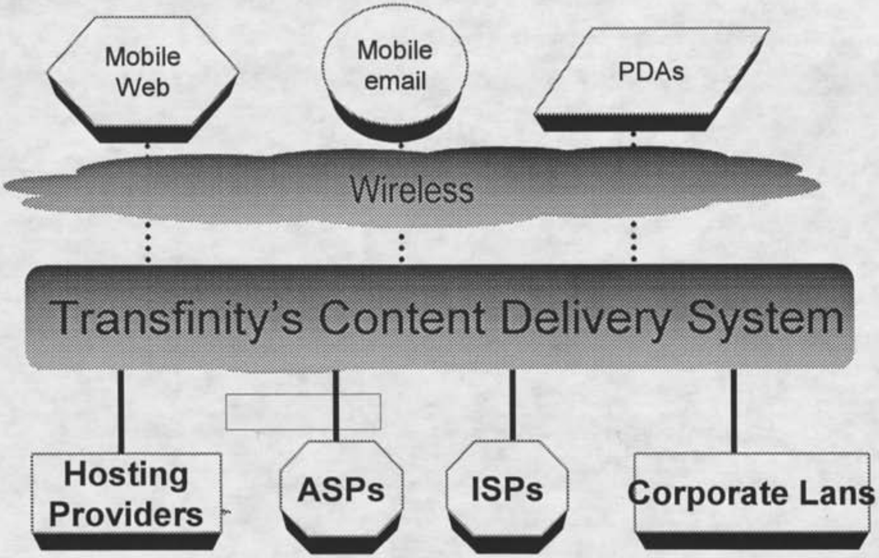
Gilder Technology Report

TRANSFINITY

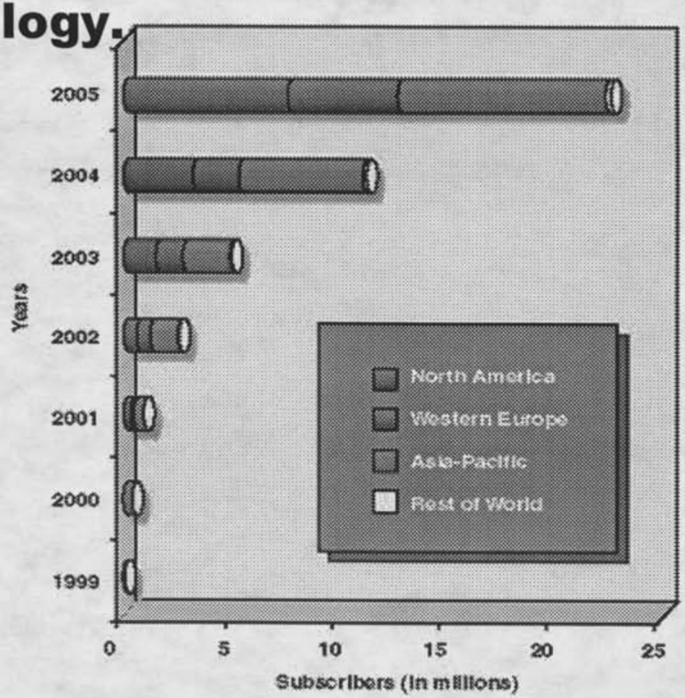
ASPs, Hosting Services and wireless mCommerce represent additional opportunities:

- Over half of America's networked businesses intend to implement an ASP solution in the next 12 months *(Information Technology Association of America).* This translates to millions of Internet users
- Wireless users
  - The number of wireless portal users in the U.S. will grow to nearly 25 million in the next five years *(The Strategis Group)*
  - Global wireless users could exceed 500 million by 2001 and the number could reach 1 billion by 2004 *(Philadelphia Stock Exchange)*
  - Nearly 50% of all e-commerce will be mobile *(Wireless Developer Network)*

Transfinity's solution reaches from the ISP, ASP, Hosting Provider and Corporate Lan to the edge of the mobile Internet.



Video on Demand offers another major market opportunity for Transfinity's technology.



Source: *Cahners In-Stat*

Video-over-DSL subscriber projections.

TRANSFINITY

TRANSFINITY

Edgewidth Solutions for the Internet

Transfinity's
Optimal Content Distribution System (TOCDS)
for ISPs and Content Providers

June, 2000

CONFIDENTIAL

TRANSFINITY'S OPTIMAL™ CONTENT DISTRIBUTION SYSTEM (TOCDS)

BUSINESS SUMMARY

Transfinity Corporation, a technology firm located in Dallas, can provide an Internet Service Provider ("ISP") with the ability to increase its customers' browser speed by 300 percent, regardless of whether access is via POTS dial-up, DSL, cable modem or wireless. A browser running through a ***28K modem will look like it is running at 90K***. A browser running through a ***56K modem will run faster than an ISDN connection***. This is not a costly hardware solution. ***Transfinity does this using software*** that is transparent to the customer and simple to implement on the part of the ISP.

Transfinity's solution will increase an ISP's customer base by providing its customers with a faster response time than they can get from any other ISP or content provider. This enhanced performance/service, fully implemented through software, can be made available to all of an ISP's subscribers at no additional cost. Alternately, an ISP may want to create a "premium" service, where for a nominal month fee users can achieve a significant improvement in speed without a significant increase in costs.

The Technology

A number of new Internet companies are gaining global attention by trying to deliver on the Internet's promise to provide information instantly to anyone, anywhere at anytime. These companies are using a variety of technologies that include intelligent routing, caching and proxy servers to improve performance and reduce response time for Internet information consumers. Transfinity is positioned to enter this market with ***a solution that not only provides for the intelligent distribution of Internet media, but also greatly increases bandwidth and storage capacity through the use of a revolutionary new compression technology.***

Transfinity has developed compression methods that offer a 50% improvement over all other compression methods. Transfinity has the only compression technology capable of losslessly compressing previously compressed data. In addition, Transfinity has developed compression capabilities that support and significantly increase the compression ratios of "lossy" compression methods such as JPEG and MPEG. Transfinity's technology is not intended to replace existing standards, but is designed for use in conjunction with other hardware and software storage and bandwidth standards, practices and technologies. Transfinity's compression takes the form of a browser plug-in that requires no additional resources on the part of the user's CPU.

Transfinity's technology is protected by patents and patents pending.

Conclusion

In the United States, nearly 90% of Internet households still use Plain Old Telephone Service (POTS) with their 28K or 56K modems. A lot of time and money has been spent in the last few years on broadband technologies such as fiber optic, cable and DSL. These technologies are expensive. Unless and until they are low-cost and easy to implement, most Internet households will not use them. It does not matter how much bandwidth there is if it does not reach the end customer.

TECHNICAL SUMMARY

The Core Product

Transfinity's content distribution technology is designed to increase bandwidth and storage capabilities and to decrease communication time on all networks using existing network hardware. This goal is achieved by utilizing a combination of compression, proxy, firewall caching and intelligent distribution servers. Users of this technology will include operators and users of ISPs, Content Providers, Local and Wide Area Networks (LANs and WANs), Virtual Private Networks (VPNs) and Application Service Providers (ASPs).

Transfinity's content distribution technology is based upon its intellectual property. This intellectual property consists of:

- Patents
- Patents pending
- Trade secrets and
- Copyrights

Transfinity's Optimal^{TM} Content Distribution System (TOCDS) consists of the following components:

1. The Compression Server to manage compression tasks
2. The Distribution Server to distribute the content
3. The Control Server to manage the distribution servers
4. Multiple sub-systems to manage caching, security, etc.
5. Various plug-ins, applets and applications to support the viewing and editing of specific media formats.

The Compression Server and Compression Libraries are used in conjunction with any or all of Transfinity's other servers. The system is designed to intercept content on the network at a designated point and to compress and cache the content. The system also has the ability to intelligently distribute the content.

General Description

The general concept underlying TOCDS is to intercept content on the network at some point, compress and cache the content. The intercept point may be the source or anywhere along the path including the destination. The servers are all part of TOCDS and all servers share all public content through the Control Server.

A diagram of the Internet without the TOCDS solution is provided in Figure 1.

A diagram of the Internet using the TOCDS solution is provided in Figure 2.

A diagram of the TOCDS general solution is provided in Figure 3.

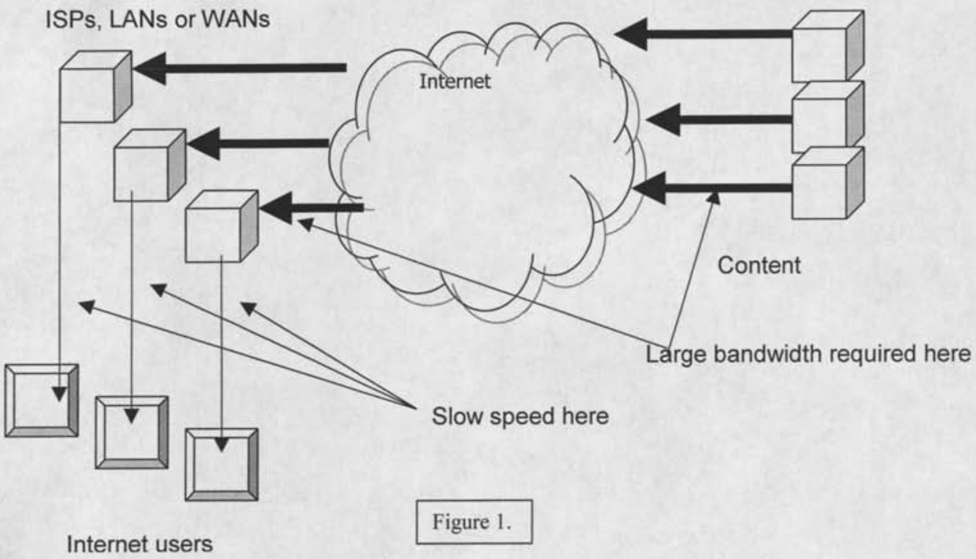


Figure 1.

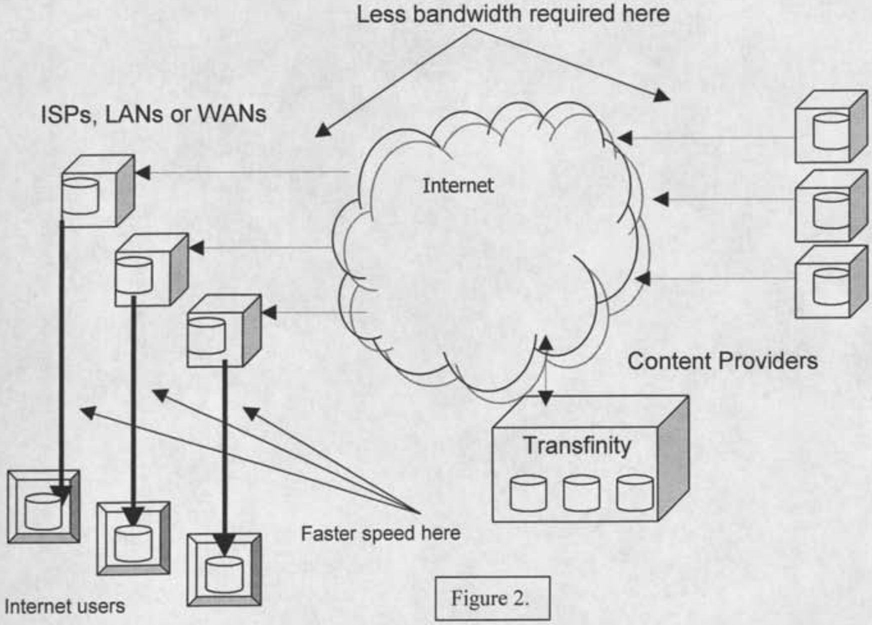


Figure 2.

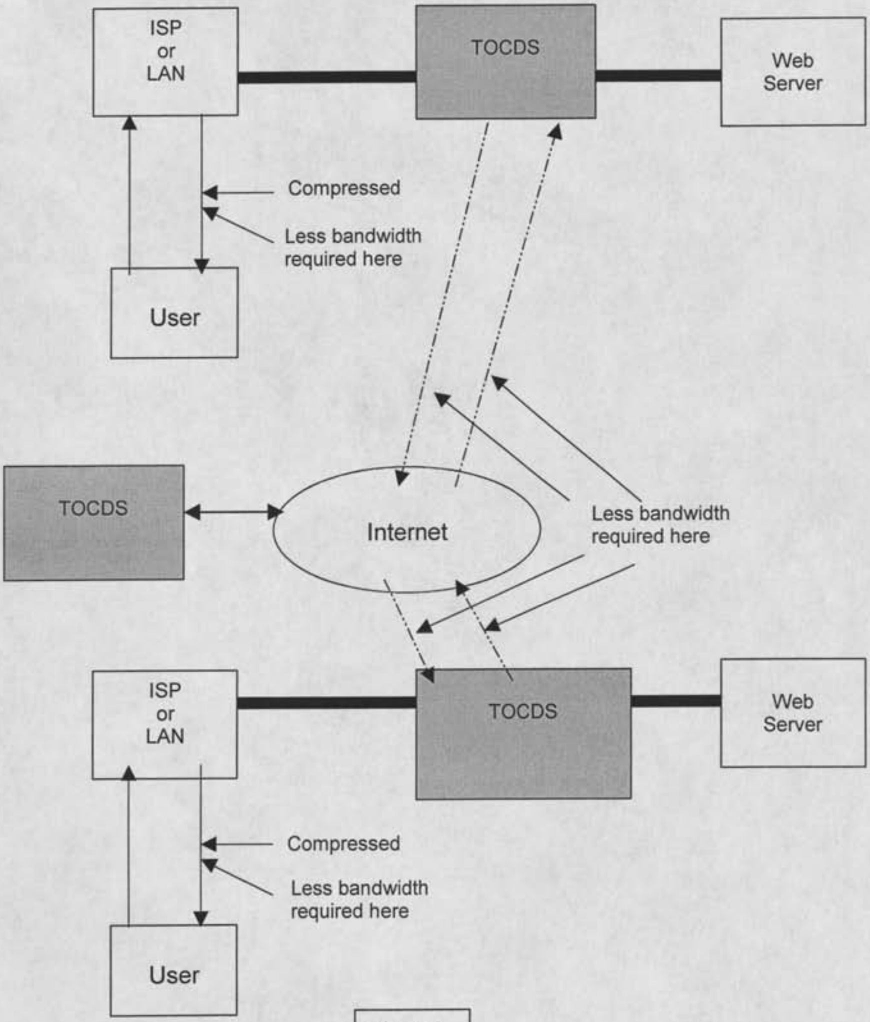
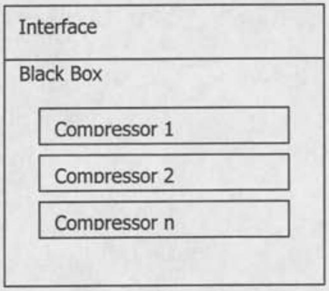

ISP
or
LAN
TOCDS
Web
Server
Compressed
Less bandwidth
required here
User
TOCDS
Internet
Less bandwidth
required here
ISP
or
LAN
TOCDS
Web
Server
Compressed
Less bandwidth
required here
User

Figure 3.

The Compression Server

The Compression Server contains Transfinity's patented and patent pending technology and is a core component of the product. The Compression Server's job is to compress the content captured by the servers. The Compression Server is intended to be treated as a "Black Box" by other programs and processes. The Compression Server can compress any type of content using any combination of lossy and lossless compression methods. The Compression Server can use various compression methods to adapt to changing content and new compression libraries can be plugged into the existing architecture when upgrades are required.



The Compression Server manages compression tasks from the other servers. Compression requests are made through RMI or TCP/IP, so that the servers may reside on different machines and networks if desired. The Compression Server manages the Compression Libraries through a native interface. It is the Compression Server's job to interpret compression requests, pass parameters to the Black Box Compressor, interpret the output from the compressor and reply to the requesting server.

The Compression Libraries

The Compression Libraries are plug-and-play software components that provide different types of compression capabilities to the Compression Server. The Compression Libraries implement Transfinity's patented N-Bit ™ compression technology as well as other compression methods.

Compression Libraries will include:

- N-bit lossless compression
- N-bit lossy compression
- Conversion from standard image types to Transfinity images
- Conversion from standard streams to Transfinity streams
- Conversion from standard file types to Transfinity file types

The Compression Libraries will eventually support both real-time, packet level compression in hardware as well as content specific compression in software.

TOCDS Benefits

From the user's viewpoint, a 50% increase in bandwidth will be possible through the use of compression technology and a 300% increase in speed will be possible through the use of compression and caching technology with a small reduction in visual quality on some content.

From the Service Provider's viewpoint, large reductions in incoming network traffic due to the caching nature of TOCDS will result in large bandwidth savings.

TOCDS Applications

TOCDS are part of a network of Distribution Servers that provide increased bandwidth and speed anywhere that server technology can be applied. TOCDS can be configured in a variety of ways to perform various tasks. One possible configuration would be a proxy; caching and compressing content for an ISP or LAN. Another possibility would be as a firewall, interpreting requests and serving up compressed content from an inner network of e-commerce VPNs and standard web servers.

The Theory Behind Transfinity's Lossless Compression

Transfinity's lossless compression is based on patents and patents pending. The theory underlying these patents is based on a change in our understanding of binary data. From Transfinity's viewpoint, the bit, not the byte, should provide the basis for all computing systems and methods.

A description of the theory follows:

For any ordered set (i.e. class) S of symbols there exists a class Ω (called Omega) of *N* (i.e. variable) length symbol strings similar to the set of natural numbers {1, 2, 3, . . .} where Ω is further defined as the class of all ordered combinations of *N* length symbol strings where:

$N = \{1, 2, 3, \ldots\}$

For example, where

S = {0, 1, 2}

Ω = {0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, 000, . . .}

Or where

S = {a, b, c}

Ω = (a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, . . .}

Furthermore, the class Ω is equivalent for all S.

The Class Ω of N-bits

The class Ω in binary form would be represented as:

{0, 1, 00, 01, 10, 11, 000, . . .}

This is the class Ω of *N*-bits.

Furthermore, the binary representation of members of the class Ω of *N*-bits does not require that any member be interpreted as a number value. A member of the class Ω may be identified with a member of any type (of class). As a result the class Ω is capable of defining any number of types as subclasses.

The class of bytes is defined as a subclass of the class Ω:

{00000000, 00000001, 00000010, . . ., 11111111}

By definition, the class Ω of *N*-bits is the parent class of all classes whose members may be represented as binary symbol strings. A member of Ω (e.g. 10010101010) does not represent a specific object or number value. The member is nothing more than a binary symbol string - an abstract, virtual symbol over which the "template" or "idea" of the object is placed.

By specifying the length in bits and the binary values of members of the class Ω as a subclass, it is possible to define classes of any type commonly used in computing. Subclasses of the class Ω include the natural and real number systems, microprocessor instruction sets, virtual machine instruction sets and objects of any type (including state models, state machines, texts, images, sounds, etc.). Simply stated, the class Ω of *N*-bits provides a means whereby all classes, attributes and relations representable in binary or digital form may be defined as members of Ω or its subclasses.

Entropy and Compression

The term entropy as it is used in information theory is a measure of how much information is contained or encoded in a message. A message in turn is defined as a string of symbols. The higher the entropy of a message, the greater is its information content. Data compression is to information theory what set theory is to higher mathematics and as such becomes the means by which we understand the fundamental nature of information. The lower the entropy, the smaller is its information content. Information theory states that there is a limit to the degree to which data can be compressed based upon its entropy.

The assumption is that the content or meaning of the data is somehow contained within the digital representation of the data. This is not so. A string of bits has little or no meaning in and of itself. Very little of the meaning of any data is contained within its binary representation. The large majority of its meaning is extrinsic. By transferring more and more meaning or content to the external source or model that represents the data, greater and greater lossless compression is possible. Lossless compression means that data, once compressed, can be returned to its original state without the loss of a single bit.

Lossless compression has followed an evolutionary path beginning with the work of Claude Shannon in the late 1940's. Lossless compression methods that have resulted from this work include:

1. The Shannon-Fano algorithm developed simultaneously by Shannon at Bell Labs and R.M. Fano at MIT uses a simple method to identify binary codes for symbols in a given symbol string and to create a binary tree to organize the resulting codes.

2. Huffman coding uses unique prefixes to describe the variable length binary codes that result from the compression process. First described in 1952 by D.A. Huffman, this coding algorithm is believed to be the most efficient method of generating variable length binary codes when provided with a table of probabilities for a given symbol set.

3. Adaptive Coding may be applied to any lossless compression method that uses statistics to create a tree or table to describe the data it is compressing. Compression methods originally used fixed or static models to contain the relations between the original symbols and the resulting encoded values. These trees or dictionaries are transmitted

along with the data and may result in significant decreases in compression ratios. Adaptive coding modifies the model as the data is processed. The result is greater compression.

4. Lempel-Ziv compression uses a dictionary to describe the relations between the data that appears in a fixed or sliding window and the data (i.e. symbols or symbol phrases) that have occurred in previously seen text. First described by Jacob Ziv in 1977 and Abraham Lemple in 1978, these methods are useful in compressing data in devices such as modems, tape drives and network devices that see a constant stream of symbols.

5. Arithmetic coding uses the entire symbol string to create a single floating point number greater than or equal to 0 and less than 1. Although the entropy associated with a symbol in a given symbol string may be fractional, methods such as Huffman coding describe the coded symbols in terms of whole bits. Arithmetic coding allows the resulting binary values to be described in fractional terms. The result is a more efficient compression process than is available with Huffman coding.

The class Ω of *N*-bits provides a means whereby the entropy of a given data set may be increased or decreased at will regardless of the original content of the message.

This is done by varying the length in bits of the input symbols to the model. This, in effect, changes the model for each symbol length. So, in practice, the model can be varied until desired entropy is obtained. Changing the model to find the desired entropy allows data that was previously uncompressible to be compressed. Data compressed at some symbol size will reach an entropy limit and converge at the calculated probabilities for that data set. To change the entropy limit and the calculated probabilities, one has only to change the model.

The entropy of a given symbol in a symbol string is defined as the negative logarithm of the probability of its occurrence in the string. The entropy of a symbol string (i.e. message) is defined as the sum of the entropy for all the symbols in the string. The formula for determining the entropy of a given symbol in a binary message is:

Entropy (i.e. number of bits) = - Log base 2 (number of like symbols/total symbols in message)

As an example, the binary value 01000010 is a standard representation of the letter "B" using the American Standard Code for Information Interchange (i.e. ASCII). The binary value 01000010 means "B" only because it was decided years ago to represent the English alphabet using eight bits and 01000010 was designated the letter "B".

By changing the length in bits of the input the entropy for the letter "B" can be made to vary:

Symbol	Probability		Entropy
01000010	1/1		0
	Total	=	0
0100	1/2		1
0010	1/2		1
	Total	=	2
01	1/4		2
00	2/4		1
10	1/4		2

Total = 5

The base size of the binary symbol set may be modified to optimize the compressibility of any source of binary input regardless of whether that source of input is also the output of a previous compression process. As a result, the theoretical limit to the number of times a symbol string may be repeatedly compressed remains undetermined.

BURTON GRAD ASSOCIATES, INC.

7 WHITNEY STREET EXTENSION
WESTPORT, CONNECTICUT 06880
(203) 222-8718
(203) 222-8728 FAX
BURTGRAD@AOL.COM

September 28, 2000

Mr. Jim Lincoln
First Dallas Ltd.
300 Crescent Court
Suite 100
Dallas, TX 75201

Dear Jim:

Technical Due Diligence Report on VE Group, LLC

The following are BGAI's findings, conclusions and recommendations about VE Group's technologies, programs and planned product directions. Both Sid Dunayer and I are available for futher discussions by telephone. We have separated the findings and conclusions by the two business units: Transfinity and GlobalESP. The summary and recommendations have been combined for the two businesses. Enclosed as attachments are A-1 and A-2: Burton Grad and Sid Dunayer professional profiles; B-1: Information Request List; B-2: Interviewees; C: Technical Review of VE Group, LLC by Sid Dunayer; and D: Interview Notes by Burton Grad.

Findings: Transfinity

- Transfinity has built a program which uses its patented algorithms and proprietary logic to provide the ability to compress the transmission bandwidth required for certain image files by 10% to 68%.

- The compression technology is well done, using a creative new mathematical bit level approach which is quite different from the other known byte level compression techniques.

- Transfinity plans to produce and license the compression and decompression programs to ISPs and Telcoms so that they can reduce the bandwidth (hence time and resources) needed to transmit Internet content requested by users' browsers.

- The programs constructed so far deal only with GIF and JPEG images, but the technology should be applicable to other image files as well as to voice and text/data files, although with somewhat different compression results.

- The system is designed around the idea of using a proxy server by the ISP to actually obtain the requested Internet files; these files are then put through the compression process, transmitted to the user, and then decompressed for user viewing.

- Dennis Tucker who produced the programs, in conjunction with Shingting Liu and Robert Dempsey, believes the programs are very complicated. He also believes that if images are compressed under one operating system (like NT), then they must be decompressed under the same operating system.

- The actual code was not reviewed since Dunayer felt that the current implementation was really a proof of concept, not a deliverable product. He did review demos using the system against both canned files and also against certain files which Dunayer gave them. The demos were successful with over 50% reductions.

- The decompression program is data driven, so it should not require reprogramming in order to handle different mathematical models used during the compression process.

- There was no business case or market study available for BGAI to look at for the Transfinity business, although we were told that one was in process and would be ready shortly.

- Statements were made by Lang Wedgeworth and Michael Harold that sole rights to the patents and proprietary materials were owned by VE Group (Transfinity) through various legal documents involving Michael Harold, Joe Morgan, FedEx and Gemini Systems.

- The programming work has been performed using a mixture of C, C++, Java, JavaScript and HTML.

- An intermediate image format is created which is used for transmission and is the input to the decompression process.

- The stored image is first decompressed and then compressed into the Transfinity image format.

- Version 1 compresses GIF and JPEG files.

- Version 2 will compress "everything" else.

- Version 3 will provide "real-time" compression without necessarily requiring a cache.

<u>Findings: GlobalESP</u>

- GlobalESP has built a B2B product to demonstrate the capabilities of using the proprietary development tools and technologies which they designed and implemented.

- Proposals have been made to at least two prospective buyers where GlobalESP would partner with Answerthink to use these tools to build B2B-type systems.

- The tools and technologies can be used for constructing networks to interface a wide range of distributed applications which were built using heterogeneous technologies.

- GlobalESP plans to work with various system integrators to propose and implement application systems.

- VE Group (GlobalESP) has applied for a patent covering their distributed virtual machine concept.

- GlobalESP has constructed a transactional directory services capability which they believe to be of high value.

Conclusions: Transfinity

- The n-bit compression technology is of very substantial value, and it seems likely that the patents should be valid.

- This technology can make a real major difference in reducing the size of both storage and transmission of images, voice and text/data.

- The current Transfinity application for browser use through ISPs and Telcoms seems awkward and may not be effectively marketable or profitable.

- The content Web managers may be a far more valuable target since it makes more sense to store the content in compressed form for both storage and transmission savings. The issue of decompression would need to be addressed.

- This is a natural business for licensing the technology usage to providers and customers with some kind of per-unit usage fee.

- Investing any large amount of time in programming applications is probably not productive.

Conclusions: GlobalESP

- The capability of building distributed virtual machines is of great significance and should be of substantial market value.

- Because the tools are programmed entirely in Java, they are fully portable to operate on any actual machine (Unix, NT, etc.).

- The services provided include capability for:
    - translating objects from host to virtual machines
    - connection services for host to participate in any network
    - workflow services

- The programs are professionally written

- The current B2B implementation is of little value and does not effectively show the capabilities of the unique tools and concepts.

- The potential for cross-platform distributed computer networks is large.

- Exploiting this technology advance will be quite difficult for a startup company.

<u>Summary and Recommendations: VE Group</u>

1. There is significant technology:

    - n-bit compression for more efficient storage and transmission of images, voice and data/text
    - Distributed heterogeneous Web-based applications system development tools

2. The specific current packaging and marketing direction for these technologies is weak and probably misdirected:

    - Browser-oriented reduction of bandwidth for ISPs
    - E-commerce retail-oriented applications

3. Neither of these business cases looks profitable since each requires substantial investment and significant marketing power to succeed, neither of which VE Group has now nor can get easily.

4. What is the company worth at its present stage?

    - Must be viewed in terms of what someone would pay for exclusive rights to each technology, rather than what the business value would be for the proposed applications and markets.

    - There is extra value because of the quality of certain of the people and their knowledge

5. A set of questions need to be asked:

    - Who are the potential technology buyers?
    - Will they license on a non-exclusive basis?
    - Are the big consulting firms the right partners for GlobalESP?
    - How do the technologies make money for their users and for their suppliers?
    - Are there any analogous companies which are technology rich but marketing poor?
    - Who are the real competitors?

6. In our preliminary opinion, both businesses are worth an initial investment, not to productize and market what has been done, but rather to determine the appropriate markets, the proper way to reach these markets, how to package and price the offerings and how to organize and manage a technologically focused company.

7. The technologies are of profound value and FDL should find a way to become a participant at a reasonable price and then determine how to ensure that each business rethinks its technical, marketing and financial plans.

Very truly yours,

Burton Grad

Burton Grad

BG:5319
cc: Sidney J. Dunayer

Professional Profile

Communications and Network Related Projects

Major International Chemical Manufacturer

Requirements analysis and design of the global network connecting the various product design centers worldwide. The network is currently implemented using Token-Ring and Ethernet local area networks connected via private TI/T3 service, Fiber links, Asynchronous and Synchronous dial connections, X.25 packet connections and SAA connections to the mainframes. Through this network, the chemists worldwide can share data and work together on new creations. The actual mechanism used to route any given "transaction" is dependent on the required response time for that transaction. Those that are "urgent" or require a timely response are routed via an appropriate network connection. The lower priority data replication messages are batched and sent using a cheaper network route.

Software Products Company

As part of a strategic planning study, analyzed various current and proposed message/document interchange models to establish requirements for an integrated messaging system, including analysis of transport mechanisms and use of available communications software packages.

Major Software Products and Services Company

As part of a study to determine whether to centralize company development and processing services, prepared requirements statement for installing an integrated communications network to cover development, processing services and corporate administration as well as telephone and fax services.

Network Services Provider

As part of a technical due diligence for an acquisition, performed an analysis to determine possible methods for connecting the newly acquired customers to the client's VAN. Analysis included the possibility of connecting the VAN to the packet network used by these customers. In this way, the packet service could reroute the customer transactions to the VAN. As customers were migrated from the packet network to the VAN, service on the packet network would decrease and eventually would cease, at which time the connection to the packet network would no longer be required.

Major Financial Institution

Designed and implemented a corporate-wide customer service network including the use of small computers (replacing mainframes), leased lines, dial-in backup units and other interconnect facilities for regional processing centers.

Attachment B-1

Information Request List

A. Development

1. Organization and training of development people
2. Development methodology
3. Scheduled enhancements/customer commitments
4. Current maintenance activities
5. Current development activities
6. Testing and quality assurance procedures
7. Effort and cost records for development
8. Program update procedures
9. Installation procedures
10. Availability and procedures for international usability and service
11. Use of third party developers
12. Detailed review of schedule and progress for new program completion

B. Technical Review

1. Supported platforms and systems for the technologies
2. Major features of the technologies:
    - functions performed
    - ease of installation and use
    - maintainability
    - audits and controls
    - security
3. Development languages and special tools used
4. Number of modules per program and lines of code
5. Provenance of all program modules (where did design and code come from)
6. Inclusion of proprietary notices in source and object modules, both current and previous versions
7. Method of change control
8. Volume and magnitude of change history
9. Architecture of the programs
10. Internal system documentation level and updates
11. Documentation of specifications and design
12. Prerequisites for running the programs
13. Examination of source code
14. Review of usage/demo of operational code
15. Unit and system test cases
16. Relevant patents and patents applied for

Attachment B-2

Interviewees

John Dean

Bob Dempsey

James Dodd

Mike Harold

Shinting Liu

Dennis Tucker

Ramesh Venkataramaiah

Lang Wedgeworth

Attachment C
Page 1

Technical Review of VE Group, LLC

Sid Dunayer – 26 September 2000

People Interviewed: Mike Harold, Dennis Tucker, Shinting Liu, Bob Dempsey, Ramesh Venkataramaiah, Lang Wedgeworth and James Dodd.

Technical Review (Transfinity)

1. **Prerequisites for running the products**
   The system presented required some server components to run under Linux and others to run under Windows/NT.

2. **Major features of the products**
   The primary feature of the product is the compression of certain types of Internet content so as to decrease the bandwidth needed to transmit that content.

3. **Development languages and special tools**
   The system is written in a combination of C, C++, Java, Javascript and HTML.

4. **Number of programs and lines of code**
   As I felt that the product demonstrated was not of commercial quality, I did not ask for this data.

5. **Provenance of all program modules**
   All code was reportedly written at Transfinity.

6. **Inclusion of proprietary notices in source and object modules**
   As source code was not reviewed, I was not able to determine if the necessary notices were in place.

7. **Method of change control and change records to date**
   There does not appear to be any formal change control in place. I did not ask for data on change records.

8. **Architecture of the system**
   The system is designed using a front-end Proxy Server, to field requests from a Web browser and one or more back-end servers to provide compression and control of the entire process.

9. **Internal system documentation level**
   Transfinity has no formal internal documentation. Dennis Tucker feels that the source code is the documentation.

10. **Documentation of specifications and design**
Transfinity only has high-level design documents. Dennis Tucker indicated that this is the method in which they work and that the programmers are expected to work from the high-level design material.

11. **Review of the source code**
As I felt that the system demonstrated was only a proof of concept, I did not review the source code.

12. **Demo of operational code**
Transfinity demonstrated the code using data of their choosing. I then asked that they use data from a web site that I supplied. The demo did not go smoothly and the software had problems with the Web site I supplied. They were able to fix this later on.

13. **Unit and system test cases**
Transfinity has some data files that they use for demos and minimal system testing.

14. **Relevant patents and patents applied for**
Patent numbers 5,893,084 and 5,600,726 cover the methods used to implement the compression algorithms. Both of these patents are assigned to Gemini Systems, Inc.

Technical Review (GlobalESP)

1. **Prerequisites for running the products**
The tools can be used on any platform that supports JAVA.

2. **Major features of the product**
The GlobalESP tools provide the necessary components for developing distributed computing applications that may run on different operating environments and utilize standard Internet protocols for communications.

3. **Development languages and special tools**
The tools are written entirely in pure JAVA.

4. **Number of programs and lines of code**
There are approximately 50 modules and about 750K lines of code.

5. **Provenance of all program modules**
GloablESP reportedly wrote all programs.

6. **Inclusion of proprietary notices in source and object modules**
There were some copyright notices in the code, but there were many places where the notices were missing.

7. **Method of change control and change records to date**
Change control is performed using MKS. As this is a new product, there are no records of changes made to date.

8. **Architecture of the system**
The architecture is too complex to be described here, but it is well described in the patent application for this process.

9. **Internal system documentation level**
Internal system documentation is created using JavaDoc. This is probably adequate to define programming interfaces, but there is still a need for more detailed system flows.

10. **Documentation of specifications and design**
GlobalESP has good design specifications and notes.

11. **Review of the source code**
A review of the source code found that it was well-structured, easy to read, but only sparsely commented.

12. **Demo of operational code**
GlobalESP demonstrated an E-Commerce application written using the tools. The demo went smoothly.

13. **Unit and system test cases**
GlobalESP has a limited test library.

14. **Relevant patents and patents applied for**
Mike Harold has applied for a patent covering the methods used to implement the tools. The patent application lists Mike Harold as the inventor and does not assign the patent to any other entity.

Observations

- The Transfinity compression algorithm appears to do extremely well with content they are focusing on, JPEGs and GIFs. In a test on data I provided, the Transfinity compressor shrunk the image I provided by 68%, i.e., to about one-third its original size. Due to the nature of the algorithm used, it is very likely that they will also do well with other types of Internet content, such as audio and video.

- The current Transfinity offering is aimed at ISPs. It promises to reduce the bandwidth required to deliver Internet content to end-users. Transfinity claims that smaller ISPs would not run their own compression servers, but rather, would buy the service from Transfinity.

- The system, as implemented, is very complex and requires the end user to configure his browser to talk to the proxy server. As the ISP cannot force the user to use the proxy server, those that do not reconfigure will completely bypass the Transfinity system.

- Transfinity has apparently not done any type of study to determine the market for the product.

- Dennis Tucker appeared difficult to work with. He repeatedly criticized a member of his staff during the demo when problems arose. He further made statements that made no sense. For example, when I inquired as to why the compression server had to run on Windows/NT, he stated that when you compress data on one platform and decompress it on another, there are sometimes problems. The statement is utter nonsense.

- The lack of any formal documentation will make it difficult for others to understand what Transfinity has implemented. The lack of a formal development methodology makes it difficult to determine project progress and status.

- GlobalESP has implemented a professional set of tools to aid in the development of distributed computing applications. These applications may be traditional software applications or they may be applications built around the functionality of network appliances. The tools are written in a portable language, JAVA, and utilize standard industry protocols and specifications, TCP/IP, XML and UML.

- GlobalESP has been unable to demonstrate the power of the tools to date. In an attempt to remedy this, they utilized the tools to create an e-commerce application. While the application runs well, it hardly shows the true power of the tools. Furthermore, it tries to position the tools as a B2B solution when indeed that is not really the case.

- I found the GlobalESP team very professional. They have good documentation and internal communication, and the code was better than I would have expected for a new product.

VE Group Interview Notes

Lang Wedgeworth (VE Group)

1996 Started patent process
Had option to buy 50% of Gemini Systems LLC; bought in for $200,000

1997-5/98 Mike Harold went to work at FedEx to produce a virtual outsourcing company
Employment agreement excluded patents for: encryption, compression, any to any translation, precision math
Raised $500,000 -- Global ESP

5/99 Brought in Gemini and PNG Investments = $2.1 million

New company was VE Group: 45% Gemini, 35% PNG, 20% employees
Hired 5-6 people from FedEx
Global ESP was primary project
70%-80% of cost
Approached EY Consulting
Used grocery ordering as an application to build and use tools

1/00 Presented to British American Tobacco with EY

4/00 EY then used i2 and cut out Global ESP

9/99 Dennis Tucker and Shinting Liu worked on compression and produced proof of concept

1/00 Hired Bob Dempsey
Worked on server side

5/00 Raised $1.2 million -- same sources

8/00 Hired John Dean for market knowledge

John Dean (GlobalESP)

- Wanted to develop common set of tools for Answerthink (which acquired Relational Technologies)
- Relation negotiations with Answerthink (channel for sales and implementation of software)
- 5-8 people on implementation team (from consultant)
- 1-3 people from GlobalESP
- Pricing model: per project area

- Put up Web site
- No "go to market" plan
- No real alliances

- Use of Java for language and operating environment
- Team core had been together

Customers

- Shaw Group, Stone and Webster Construction
  In final run-off; Houston

Software	$400k	
Consulting	500k	(3.5 people)
Tech Solutions	1.5 m	
Phase I	2.4 m	Operating System

4 months integration schedule

- Dallas Market Center (Trammell-Crowe owns buildings)
  Partnership with Answerthink

- Can do rapid application development
  Functionality
  100% Java
  XML built-in

- $5 million needed in next year

- Invest in: consulting arm, Support, R&D (not many) and Sales and Marketing aimed at integrators

Ramesh Venkataramaiah (GlobalESP)

1996	FedEx project engineer, network oriented, integration
1998	Network Centric systems, multi-level, replace packet transmission
1999	Work flows only worked with a particular data model Built interfaces or protocols Created two teams: application services, system infrastructure services Vertical markets, to some extent, determine application specs
2000	Responsible for grocery chain implementation; will go live early 10/00 Requirements: 15-30 days to produce specs High-level design Detail design/programming QA: Ed O'Brien Hired trainer

Mike Harold (GlobalESP)

- Interested in math and linguistics
- Describe a "grammar" for a broad problem; worked with Joe Morgan
- Variable word length (not byte restricted)
- Applied to encryption, arbitrary precision math, compression
- Distributed virtual machine, patent pending
- Transactional Directory Services
- Tucker brought in to work on security for Global ESP
- Considered use in storage devices; then low level communications
- Media distribution system
- Problem with receiver side (narrow bandwidth)
- Browser plug-in
- Receiver end hardware (pda's, cell phones)

Mike Harold (Transfinity)

- n-bit technology additional user
- Storage applications
- Indexing of specific items
- Pattern matching
- Encryption

Dennis Tucker (Transfinity)

- Finished proof of concept 9/99
- System to apply/demo on Internet
- Overall system design
- Hired two other programmers for compression plug-in and server
- C, C++, Java, JavaScript, HTML
- Design documents
- Project plans; task lists
- Browser interfaces (IE, Netscape) have caused some problems
- Transfinity image format
- Compression plug-in for each "data type"
- For GIFs and JPEGs can get 50-55% average compression
- Decompress first, then compress with Transfinity techniques

- V1: compresses GIFs and JPEGs to Transfinity images
- V2: compresses everything else: Zip, etc.
- V3: Real-time compression; may eliminate cache

- Final
  Executed
- Copy sent to JM & SF
  JDRP 11/06

FIRST DALLAS, LTD.
300 Crescent Court, Suite 1000
Dallas, Texas 75201

November 4, 2000

where is non-disclosure for BofA to sign?

Transfinity Corporation
1231 Greenway Drive
Suite 300
Irving, Texas 75038

Re: Proposal to Purchase Securities

Gentlemen:

This letter is intended to summarize the principal terms of a proposal being considered by First Dallas, Ltd. (the "Investor") regarding a possible investment by it (or a partnership of which it is a general partner) in Transfinity Corporation (the "Company" and, together with the Investor, the "Parties"), in the form of a purchase of convertible preferred stock and warrants for the purchase of common stock of the Company.

PART I

The Parties intend to commence negotiating a mutually agreeable written definitive agreement providing for the possible investment (a "Definitive Agreement"), which will be prepared by the Investor's counsel. The execution of any such Definitive Agreement, however, is subject to the satisfactory completion of the Investor's ongoing due diligence investigation of the Company and its business. Based on the information regarding the Company and its business provided to the Investor as of the date of this letter, the parties expect that the Definitive Agreement will reflect the following terms:

1.1 General. At the closing of the possible investment (the "Closing"), the Company would issue and sell to the Investor, and the Investor would purchase from the Company, assuming 1,000 shares currently issued and outstanding, preferred stock convertible into 111.11 shares of common stock (the "Common Stock") of the Company and having an aggregate par amount of $2,000,000 (the "Preferred Stock") and warrants (the "Warrants"), in each case having the terms set forth in Annex I hereto. The aggregate purchase price of the Preferred Stock and Warrants will be $2,000,000.

1.2 Registration Rights. The Investor would receive usual and customary demand and piggyback registration rights with respect to the Preferred Stock and Warrants and any shares of Common Stock issuable upon conversion or exercise thereof.

1.3 Right to Participate in Future Financings. In the event that the Company issues and sells Common Stock or securities convertible into or exchangeable for Common Stock (other than shares of Common Stock reserved for issuance to employees, shares of Common Stock issuable upon exercise of the Company's outstanding options or warrants or shares of Common Stock issued in connection with a public offering), the Investor would be given the opportunity to participate in such transaction, on the same terms as those given to the other purchasers in such transaction, to the extent necessary to enable the Investor to maintain its percentage ownership of Common Stock, computed on a fully diluted basis. This right would terminate upon the Company's initial public offering of Common Stock.

1.4 Representations, Warranties and Indemnities. The Company and each of the Company's stockholders would make comprehensive representations and warranties to the Investor regarding the Company and its business, and would provide indemnities and other appropriate protections for the benefit of the Investor.

1.5 Closing. Barring unforeseen circumstances, it is anticipated that the Closing would occur on or before the date that is 60 calendar days following the date hereof.

PART II

2.1 Due Diligence Investigation. Subject to the terms of a mutually agreeable non-disclosure agreement to be signed by the Parties concurrently with the execution of this term sheet the Company will (a) afford to the officers, directors, employees, agents, partners, stockholders and other representatives, including without limitation consultants, accountants and attorneys (collectively, "Representatives"), of the Investor full access during normal business hours to the facilities, personnel and books and records of the Company so as to afford the Investor an opportunity to make such review, examination and investigation of the business, assets, liabilities, condition (financial or otherwise), results of operations and prospects of the Company as the Investor may desire to make and (b) keep the Investor fully apprised and informed of all significant developments relating to the business, assets, liabilities, condition (financial or otherwise), results of operations and prospects of the Company.

2.2 Parties Obligations. Unless and until the Definitive Agreement has been so executed and delivered, neither the Investor, the Company, nor any of their respective Representatives will have any legal obligation to the Company with respect to any possible investment, except to the extent specifically set forth in this Part II.

2.3 Costs. If the proposed transaction closes, each Party will be responsible for and bear all of its own costs and expenses (including any broker's or finder's fee) incurred at any time in connection with the possible investment described herein; provided however, the Company will reimburse the Investor for its reasonable out of pocket expenses and reasonable third-party expenses, including attorney's fees and expenses, incurred in connection with the proposed transaction not to exceed $75,000.00, upon the occurrence of both of the following two conditions (a) prior to the Termination Date (herein defined) the Company fails to sign a

Definitive agreement, and (b) at any time during the period commencing upon the date of this Term Sheet and ending on the Termination Date, the Company enters into a binding agreement with a third party with respect to an investment in the Company. Such reimbursement shall be made by wire transfer within fifteen business days of the delivery by Investor to the Company a written demand for such reimbursement together with supporting documentation for such reasonable out-of-pocket and reasonable third party expenses.

2.4 Waivers. No failure or delay in exercising any right hereunder will operate as a waiver thereof, nor will any single or partial exercise thereof preclude any other or further exercise of any other right.

2.5 Binding Provisions. The provisions of this Part II will be binding on and inure to the benefit of the Parties and their respective successors and assigns.

2.6 Remedies. The Parties acknowledge that money damages would not be a sufficient remedy for any violation of the provisions of this Part II and, accordingly, the Parties will be entitled to specific performance and injunctive relief as remedies for any violation thereof, in addition to all other remedies available at law or equity.

2.7 Governing Law. The provisions of this Part II will be governed by and construed in accordance with the laws of the State of Texas, without giving effect to the principles of conflict of laws thereof.

2.8 Jurisdiction. Each of the Parties hereby consents to personal jurisdiction in any action brought in any federal or state court within the State of Texas having subject matter jurisdiction in this matter for purposes of actions arising out of the provisions of this Part II.

2.9 Termination. The provisions of this Part II and all obligations thereunder will terminate 60calendar days after the date hereof the "Termination Date", unless extended pursuant to a written instrument executed and delivered by the Parties.

2.10 Certain Obligations of the Company. The Company will cause each of its officers, directors and stockholders to comply with the provisions of this Part II as if each of them were a party hereto.

2.11 Counterparts. This letter may be executed in one or more counterparts, each of which will be deemed to be an original copy of this letter and all of which, when taken together, will be deemed to constitute one and the same instrument.

Please indicate your agreement with the provisions set forth in this letter by signing the enclosed copy hereof and returning it to the undersigned no later than 3 days after the date hereof.

Very truly yours,

FIRST DALLAS, LTD.

By:
Name: JAMES W. LINCOLN III
Title: MANAGING DIRECTOR

Accepted and agreed as of
the date first above written:

TRANSFINITY CORPORATION

By: Lang Wedgeworth
Name: LANG WEDGEWORTH
Title: PRESIDENT

ANNEX I

FIRST DALLAS, LTD.

Summary of Terms

Issuer

Transfinity Corporation (the "Company").

Initial Purchasers

First Dallas, Ltd. (the "Investor").

Securities to be Purchased

Convertible preferred stock (the "Preferred Stock") and warrants (the "Warrants").

Preferred Stock and Warrants

$2,000,000 aggregate par amount.

Dividend

The Preferred Stock will have no preferential dividend rights.

Conversion

The holders of the Preferred Stock may convert the Preferred Stock, at any time, in whole or in part, into shares of the Company's Common Stock ("Common Stock").

The number of shares into which Preferred Stock may be converted will be determined by dividing the sum of the aggregate par amount of the Preferred Stock to be converted by a specified dollar amount (the "Conversion Price"). The initial Conversion Price will be $18,000.

The Preferred Stock will be automatically converted into shares of Common Stock upon an initial public offering of Common Stock; provided, however, that if the price per share in the public offering (the "IPO Price") is less than $36,000 (subject to adjustment in the event of dividends, stock splits and such other events), then the Conversion Price will be equal to 50% of the IPO Price.

Warrants

Warrants exercisable in whole or in part on or before December 1, 2001 for the purchase of 3.86% of the Company on a fully diluted basis, taking into account as granted all options authorized under any Company plan ($25mm Pre-money Valuation; $1,000,000.00 proceeds to the Company). The Warrants are transferable subject to restrictions of applicable securities laws.

Anti-Dilution Provisions

The Conversion Price will be subject to customary adjustments in the event of (i) stock dividends, stock splits and similar events and (ii) the issuance of Common Stock or other securities convertible or exchangeable into Common Stock at a price per share less than the then-existing Conversion Price.

Board of Directors

First Dallas will have the right to elect two members of the board of directors and any executive committee (Pro forma six member board). Compensation committee (independent majority i.e. excluding any member receiving compensation) will approve executive compensation.

Tag Along Rights

Investor will have Tag Along rights to participate with VE Group in any sale of stock of the Company.

Restrictive Covenants

The Preferred Stock will contain restrictive covenants usual and customary for instruments of this type, including without limitation on: significant acquisitions or dispositions of assets, mergers, payment of dividends, transactions with affiliates, incurrence of indebtedness, issuance of securities senior to Investor's. Prior to closing, the Company and Investor will agree upon market applications for the Company's technology the Company will pursue, and the Company will agree not to change the Company's market direction without Investor's approval.

Affirmative Covenants

The Preferred Stock will contain affirmative covenants usual and customary for instruments of this type, including access to management and detailed financial and operating information. The Definitive Agreement will provide for notice and time to cure by the Company for non-material breaches of Affirmative Covenants.

Events of Default and Redemption Provision

In the event of default by the Company on indebtedness for borrowed money, or bankruptcy of the Company or breach of the Preferred Stock covenants, holders of the Preferred Stock voting as a class will be entitled to elect all members of the Board of Directors of the Company, provided the Company will have 30 days in which to redeem the Preferred Stock at par value following notice by the Investor that it is declaring an event of default.

Pre-Closing Actions

Prior to issuance of the Preferred Stock and Warrants, all intellectual property and other assets necessary for the conduct of Company's business will have been transferred to the Company and the Company will have adopted an employee incentive option plan authorizing the issuance of options for the purchase of 15% of the Company's Common Stock computed on a fully diluted basis.

US005893084A

United States Patent [19]

Morgan et al.

[11] **Patent Number: 5,893,084**

[45] **Date of Patent: *Apr. 6, 1999**

[54] **METHOD FOR CREATING SPECIFIC PURPOSE RULE-BASED N-BIT VIRTUAL MACHINES**

[75] Inventors: **Joseph M. Morgan**, Amarillo, Tex.; **Michael D. Harold**, Shreveport, La.

[73] Assignee: **Gemini Systems, Inc.**, Shreveport, La.

[*] Notice: The term of this patent shall not extend beyond the expiration date of Pat. No. 5.600.726.

[21] Appl. No.: **725,249**

[22] Filed: **Oct. 4, 1996**

Related U.S. Application Data

[62] Division of Ser. No. 419,001, Apr. 7, 1995, Pat. No. 5,600, 726.

[51] **Int. Cl.**^{6} **G06F 17/00**

[52] **U.S. Cl.** **706/50**; 706/45; 341/67; 341/95

[58] **Field of Search** 341/51. 95. 67. 341/50; 395/54. 406; 706/45. 47. 50; 707/101

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,587,725 12/1996 Sakanishi et al. 345/195

OTHER PUBLICATIONS

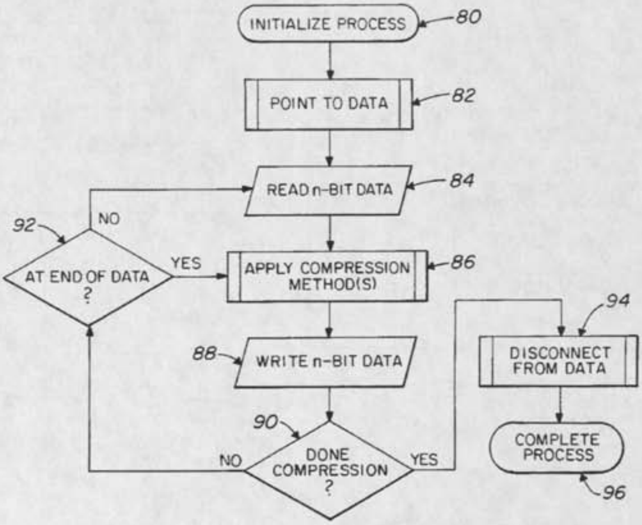
Hwang et al. "Optical Arithmetic Using High–Radix Symbolic Substitution Rules." Computer Arithmetic Symposium. 1989. pp. 226–232.

Primary Examiner—Tariq R. Hafiz
Assistant Examiner—Jason W. Rhodes
Attorney, Agent, or Firm—Jones. Day. Reavis & Pogue

[57] **ABSTRACT**

A system and method for implementing one or more specific purpose rule-based n-bit virtual processing machines. Specific purposes include. but are not limited to. encryption. compression. and arbitrary precision arithmetic. Each virtual machine consists of a command processor. a rule-base. and an interface between the command processor and the rule-base. Each of the elements of a specific purpose rule-based n-bit virtual machine—the command processor. the rule-base. and the rule-base interface—is preferably implemented as software. In the preferred embodiment. the system uses a stored rule-base as its instruction set and provides for input and output in the form of variable length bit strings of length n where n is any number greater than zero. Each of the rules within the rule-base performs one or more binary string operations against one or more variable length n-bit strings. The function of the rule-base is to provide a set of application specific rules that allows the machine to perform a particular task such as encryption. data compression. or arbitrary precision arithmetic. The system includes a method for providing a software interface to the rule-base. This interface may be a separate program or may be contained within the command processor. The command processor receives input in the form of one or more n-bit data types. performs rule-based operations on the data. and returns output in the form of one or more n-bit data types. Specific system and methods for performing data encryption. data compression. and arbitrary precision arithmetic using the invention are described.

16 Claims, 2 Drawing Sheets



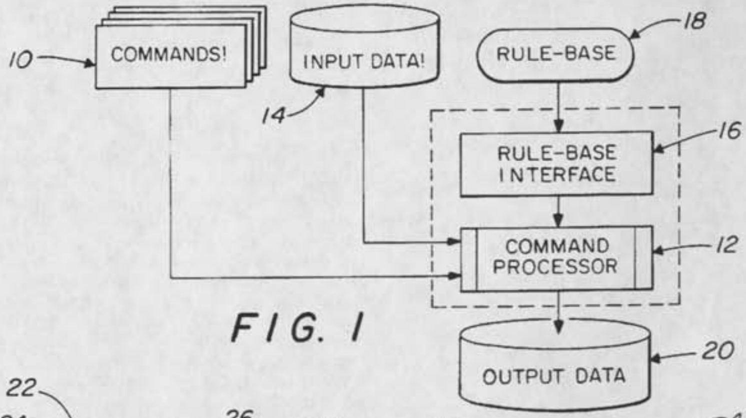


FIG. 1

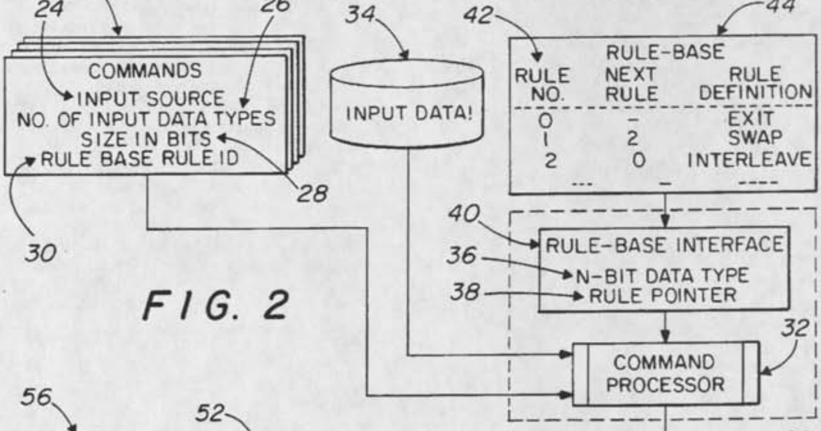


FIG. 2

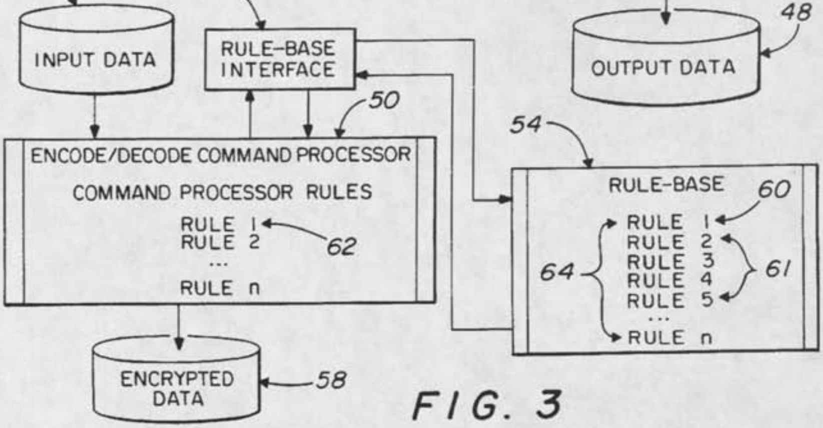


FIG. 3

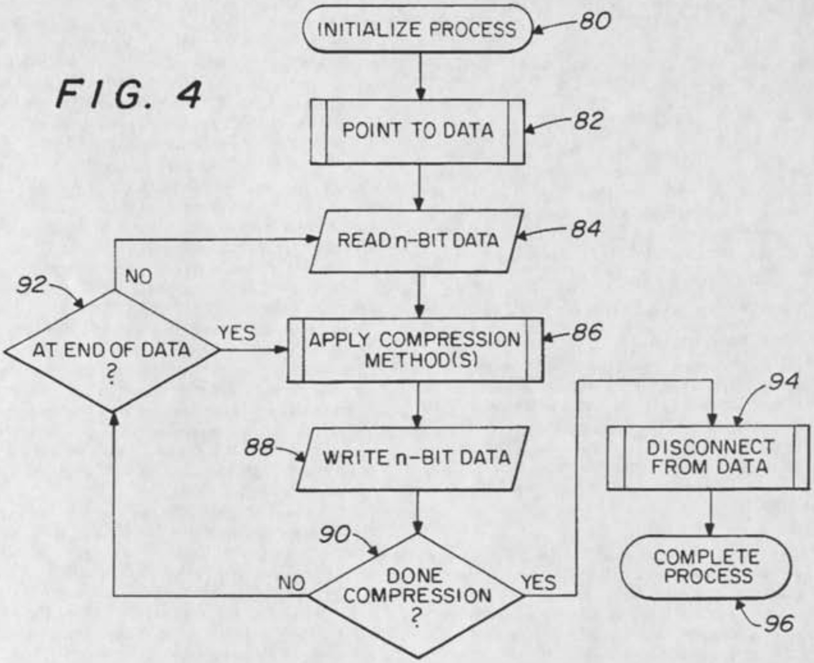

INITIALIZE PROCESS
80
POINT TO DATA
82
READ n-BIT DATA
84
92
NO
AT END OF DATA ?
YES
APPLY COMPRESSION METHOD(S)
86
94
DISCONNECT FROM DATA
88
WRITE n-BIT DATA
90
DONE COMPRESSION ?
NO
YES
COMPLETE PROCESS
96

FIG. 4

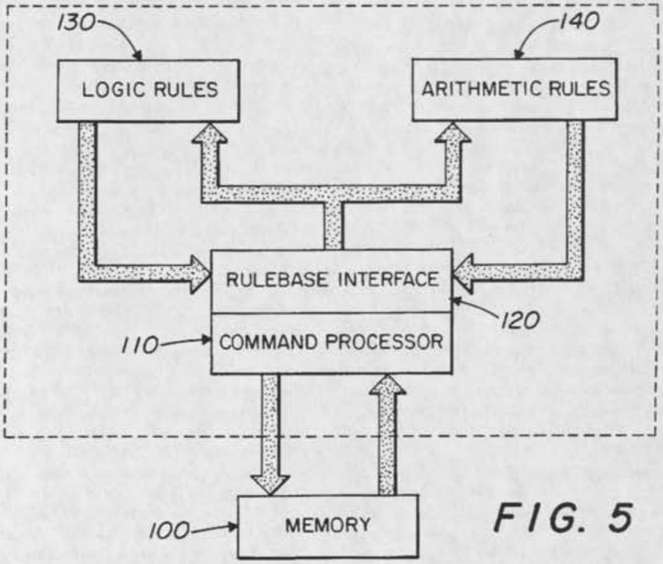

130
140
LOGIC RULES
ARITHMETIC RULES
RULEBASE INTERFACE
120
110
COMMAND PROCESSOR
100
MEMORY

FIG. 5

METHOD FOR CREATING SPECIFIC PURPOSE RULE-BASED N-BIT VIRTUAL MACHINES

This is divisional of application Ser. No. 08/419,001 filed on Apr. 7, 1995, now U.S. Pat. No. 5,600,726.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to computer systems and more particularly to a software architecture for implementing specific purpose rule-based n-bit virtual machines to accomplish such tasks as data typing, encryption, compression, arbitrary precision arithmetic, pattern recognition, data conversion, artificial intelligence, device drivers, data storage and retrieval and digital communications.

2. Description of the Related Art

Existing systems designed to process data vary widely in their specific implementations. However, few are designed for the utilization of a rule-base and there are no others known that use, as their primary data type, an arbitrary X number of bits as input, and an arbitrary Y number of bits as output, where X may or may not be equal to Y.

With respect to virtual software machines, of specific mention is U.S. Pat. No. 4,961,133 filed Oct. 2, 1990, wherein Talati et al. disclose a "Virtual Execution Environment on a Target Computer Using a Virtual Software Machine". This invention deals with preprocessing and compiling source program code in such a way as to be operating system independent and to enable the code to execute across heterogeneous computers via a virtual interface system. Though the invention disclosed by Talati et al. involves providing a virtual software machine, it does not address the problem of directly manipulating machine instructions of any given n-bit length via a rule-base to machine instructions of any target n-bit length on a target machine.

With respect to data encryption, most systems apply some form of mathematical operation or bit-wise operation, such as exclusive-or (or XOR) against the input data to be processed based upon an encryption key or password. Normally, the encryption process is highly specialized, encrypting the data in the same theoretical manner from the beginning to the end of the data stream. These methods lend themselves to differential crypto-analysis, a method capable, through analytical means, of deciphering the encrypted message.

Of specific mention is Matasuzaki et al. U.S. Pat. No. 5,351,299 filed Sep. 27, 1994, whose encryption process is very difficult or impractical to break with more standard analytical methods. This method utilizes the standard idea of XORing data together by use of manipulation of a user-provided password. To decrypt, one XORs the encrypted data again, in reverse order, with the same manipulation of the same user-provided password.

Though Matasuzaki et al. break data up into N blocks of M-bit data, the specified Embodiment I states that "each bit outputted from hash function unit is dependent on all the bits inputted thereto." It also states in the embodiments that the primary input blocks are blocks of multiples of 8 bits, and further broken down into blocks of M bits, defined in 8 bits or multiples of 8 bits. This method severely limits introduction of arbitrary block encryption rules and does not allow for a prime number of bits, such as 11 or 13.

The U.S. Pat. No. 5,285,497 to Thatcher Jr. filed Feb. 8, 1994, specifies encoding variable length Huffman encoded bits in a unique way. However, it does not address the bits as a data type arbitrarily, but in a form having a meaning directed by the Huffman compression means. The invention also requires the use of a specialized microprocessor, a fixed number of specialized encryption rules, and is specific to compressed, digital data streams.

Another unique encryption means as stated in U.S. Pat. No. 5,097,504 to Camion et al. identifies a signature based encryption means where the signature is recorded with the encrypted message and the encryption keys are stored on another, preferably inviolable, medium. This system applies a highly mathematical and specific encryption means, introducing, again, the problem and limitation of not having a flexible and rather arbitrary rule-base that is easily changeable and modifiable.

In U.S. Pat. No. 5,321,749, Virga presents an extremely unique encryption means that converts the input data into a bitmap and encrypts the bitmap to be targeted for decryption in an optical scanning device. The embodiment specifies XORing randomly generated bits produced from a user-specified password with the encoded bitmap. The bitmap is then converted to specific visual alphabet that can be easily recognized by a receiving scanning device. This method, however, allows an analytical hashing means to decipher the seed(s) generated from the user-specified password with a relatively small amount of time.

With respect to compression, there are many means of compression, all of them having the primary objective of locating the most common occurring data types and encoding them, on average, with a data type of a smaller size.

As an example, suppose the input data is comprised of the characters "ABCAB". A compression means may locate the most commonly occurring character pair, "AB", and encodes them with a single character "Z", thereby reducing the input data to ZCZ.

Though the above is an extremely simple example, the many compression means in existence today vary widely and have many implementations in hardware and software. However varying these compression means may be, a primary limitation exists for all of them. The limitation is that when compression has been achieved by use of the desired compression means, the data can no longer be compressed. This is due to the fact that the compressed output of the data results in a distribution of the input data type such that there is no longer a character or set of characters that occurs more frequently than another character or set of characters. Therefore, further compression is not possible or practical and some compression means will actually explode the size of the input data if the distribution of the characters of the input data type is relatively constant.

With respect to arbitrary precision arithmetic, many algorithms have been written to overcome the limitations of a computer to provide very high levels of precision in mathematical calculations. Though these methods can and do provide any desired precision with mathematical calculations, the calculations are performed algorithmically with the requirement to overcome the internal 8, 16, 32, or 64-bit limitations of the computer's hardware and internal memory mapping. These algorithms require a very high CPU load, demanding much of the computer's internal resources.

With respect to pattern recognition and data conversion, the invention disclosed herein provides enhancement to existing means of the same, introducing arbitrary data typing and a user-defined rule-base, the combination of which is absent in current systems.

In U.S. Pat. No. 5,321,606 filed Jun. 14, 1994, Kuruma et al. describe a user-defined set of transformation rules that define the nature of the grammar of the input data to be converted. The invention solves the problem of writing a specific parser or compiler where the limitations rely upon a specific grammar existent in the input data and a specific output term in the output data. Yet, this invention specifies that the output involves "structures of output terms in association with terminal symbols and nonterminal symbols".

In U.S. Pat. No. 4,890,240 filed Dec. 26, 1989, Loeb et al. describe a rule-based, artificial intelligence system where the rules are specifically defined in two parts, a left-hand side and a right-hand side; whereas, the left-hand side is considered an "if" statement and the right-hand side is considered a "then" statement. This invention is specific to overcoming prior problems in RETE processing and not to arbitrary pattern matching and identification with an externally provided rule-base.

U.S. Pat. No. 5,038,296 filed Aug. 6, 1991, U.S. Pat. No. 5,084,813 filed Jan. 28, 1992, and U.S. Pat. No. 5,101,491 filed Mar. 31, 1992 all refer to rule-based systems for generating program code. Though one of the objectives of the present invention is data transformation of program code from one n-bit machine instruction via an externally provided rule-base to a different n-bit machine instruction, it is not directed at code generation and the invention disclosed herein is not limited as such.

SUMMARY OF THE INVENTION

The present rule-based n-bit virtual machine, or processor, may be implemented in software and/or hardware. When implemented as software, a rule-based n-bit virtual machine converts a general purpose computer into a machine that performs an application specific function.

Further, a virtual processor may execute its instructions either in batch mode or interactively.

It is, therefore, a primary object of this invention to provide a means by which one or more of a data type of n-bit size is selected or received as input, processed by a rule or rules designed for processing one or more of a data type of n-bit size, and outputting or transmitting one or more of the processed data type of an n-bit size. In all data-type cases, the value of n is any number greater than zero. The size, in bits, and number of the input data type do not necessarily have to correspond with the size, in bits, and number of the output data type. Also, any given rule designed to process n-bit data types may or may not be specifically designed to work on an n-bit data type of a particular size in bits.

It is yet another object of the present invention to provide an architecture for creating a specific purpose virtual machine using software that manipulates n-bit data types and rule-based instruction sets.

It is another object to create a command processor controlled by a program and that accepts input in the form of one or more n-bit data types and outputs data in the form of one or more n-bit data types where n is any number greater than zero. The upper value of n is limited only by the physical or virtual address space of the computer.

It is still another object to create an interface program between the command processor and the rule-base called the rule-base interface. Separating the command processor from the rule-base allows the rule-base to be stored in different forms such as, but not limited to, a relational database table, a C or C++ language header file, an object class library, a dynamic link library, an EPROM assembly language subroutine, or a microcode instruction set.

It is a further object of the invention to provide a new method for creating applications relating to various fields within computing including, but not limited to, data typing, data encryption, data compression, arbitrary precision arithmetic, pattern recognition, data conversion, artificial intelligence, data storage and retrieval, and digital communications.

It is yet a further object of the present invention to demonstrate the advantages of the method by describing in detail specific implementations of the invention related to data encryption, data compression, and arbitrary precision arithmetic.

In accordance with one aspect of the invention, a subsidiary object is to provide a new method and system of data encryption. This new method of encryption will employ a command processor and a rule-base and will input and output data as variable length n-bit data types.

In accordance with one aspect of the invention, a further subsidiary object is to provide a new system of data compression. This concept involves, but is not limited to, the implicit redistribution of n-bit data-type frequencies by the explicit compression of data using n-bit data types as input and output. This concept allows the data to be compressed reiteratively.

In accordance with another aspect of the invention, a further subsidiary object is to provide a new system of arbitrary precision arithmetic. This new system of arithmetic will employ a command processor and a rule-base and will input and output data as variable length n-bit words.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the present invention will be more fully disclosed when taken in conjunction with the following DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS in which like numerals represent like elements and in which:

FIG. 1 is a diagram illustrating the organization of a system identifying the principal elements and processes associated with the present invention;

FIG. 2 is a block diagram further illustrating the interrelation of the elements of the invention;

FIG. 3 is a diagram illustrating an implementation of the invention as a data encryption system;

FIG. 4 is a process flow chart illustrating an implementation of the invention as a loss-less data compression method; and

FIG. 5 is a diagram illustrating an implementation of the invention as an arbitrary precision arithmetic method.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

With reference now to the drawings, FIG. 1 shows the organization of the principal elements used in the novel processes of a system and method for implementing a specific purpose rule-based n-bit virtual software driven data processing machine. A program used by the command processor **12** receives, as input, one or more commands identifying the input data source **14** and the instructions and/or arguments **10** which will be used in accessing the rule-base **18**. Once a command is received by the command processor **12**, data is input to the command processor in the form of one or more n-bit streams or strings from the data source **14**. The data is passed to the rule-base interface **16** by the command processor **12**. The rule-base interface **16** in turn uses the data to identify and select the rule or rules that are to be used in processing the data. The data is dispatched as one or more arguments to the selected rule within the rule-base **18** and the rule is applied. After the data has been modified in accordance with the specified rule or rules, the modified data is returned to the rule-base interface **16** along with any arguments appended by the last rule applied. These arguments may be used by the rule-base interface **16** to determine the next rule or rules to be applied to the data. These arguments may also consist of key words or messages identifying the current state of the data conversion process. The rule-base interface **16** may iteratively submit the data to one or more rules within the rule-base **18** based on the value or values, if any, of the arguments returned by the previous rule or rules. Once the conditions for the modification of the data by the rule-base have been satisfied, the data is returned as one or more n-bit streams to the command processor **12**. The command processor **12** then outputs the data as one or more n-bit streams **20**. The size in bits and number of n-bit streams output by the command processor **12** is not required to correspond to the size in bits and number of n-bit streams which were originally input.

A simple example of the process is shown in FIG. **2**. Command line input **22** generates data that identifies the source **24** of the input data, the number of input data types **26**, the size of the input data type **28** and the rule-based rule ID **30** that is to be applied to the data. The command line data is processed by the command processor **32** after which the input from data source **34** is read and a rule pointer **38** is generated from the rule ID **30**. The n-bit input data **36** and the rule pointer **38** are passed to the rule-base interface **40**. The rule-base interface **40** in turn uses the rule pointer **38** to identify the rule **42** to be applied to the data and passes the data to the appropriate identified rule **42** within the rule-base **44**. The data is modified by the rule-base **44** and the modified data is returned to the rule-base interface **40**. The rule-base interface **40** in turn passes the data to the command processor **32** which outputs the data as one or more n-bit data types **48**. The objects of the invention are achieved by the novel application of the rule-base and the use of n-bit data types for input, processing, and output functions. This method, when applied to specific applications, may result in major improvements in the performance and capabilities of existing software application driven processing machines. Furthermore, the use of variable length n-bit data types provides numerous opportunities to create new, specific purpose virtual computing environments which are capable of performing tasks that are not possible using eight bit technologies.

For example, and not by way of limitation, the following description illustrates one method of implementing a virtual machine or computer capable of performing rule-based n-bit encryption:

Rule-based n-Bit Encryption (RNE) encrypts data as a string of binary digits using a command processor, a rule-base interface, and a rule-base. In order to fully realize the benefits of RNE, it is important to realize that all of the data elements of the method, including the data processed by the command processor, the rule-base, and the data itself, are perceived as one or more strings of binary data.

In RNE the bit is the primary data structure. Any or all of the elements of the RNE virtual machine may be input, processed, and output as data. The binary representation of the elements of the RNE virtual machine or processor is a bit stream composed of one or more n-bit data types and is not organized as bytes except where the physical and/or system limitations of the computer require it.

As described in FIG. **3**, RNE consists of four principal elements: an encode/decode command processor **50**, a rule-base interface program **52**, a rule-base **54**, the input data or message **56**, and the encrypted data or message **58**.

The rule-base **54** is composed of one or more rules **60**. Each rule **60** may contain variable data and literal data. Each rule **60** may, but is not required to, receive one or more arguments as input. Each rule may, but is not required to, output one or more arguments. The encrypted message is the result of the RNE process. The command processor **50** rules and/or the rules of the rule-base **54** may or may not be contained in the encrypted message **58** at the time transmission occurs.

The command processor **50** is used to access the rule-base **54** and to manage the actual encryption/decryption process. One or more rules **62** may be contained within the command processor **50** to uniquely identify the command processor **50** and/or provide an index or offset into the rule-base **54**.

For example, one of the rules **62** contained in the command processor **50** might implement a hashing rule that would use input data **56** to select an encryption key to generate a pointer into the rule-base **54**. This would allow any number of public and private encryption keys to implement unique encode/decode rule sets within a single rule-base **54**. This would also allow encryption keys to be implicitly user-defined within the command processor **50**. Only matched command processors **50** could decode each other's messages. Neither encryption nor decryption is dependent upon an explicit encryption key, a specific encode/decode rule, or a specified data type. However, any encryption key may be defined either implicitly or explicitly.

The command processor **50** may also be used to parse a password or access code in the input data and to pass the resulting values as arguments to one or more rules **60** or rule sets **61** within the rule-base **54**.

As an example, an encryption key might be constructed having 3 bytes or 24 bits. Each of the bytes would represent a rule set. Each bit within a byte would represent the application of a specific rule or rule set within a rule set. The bytes 10010101, 11100011 and 00101010 (identified as rule sets 1, 2, and **3**,respectively) might be used by the first element of RNE, the command processor **50**, to apply the following rules or rule sets:

For rule set number 1, rules or rule sets 1, 4, 6, and 8 would apply.

For rule set number 2, rules or rule sets 1, 2, 3, 7, and 8 would apply.

For rule set number 3, rules or rule sets 3, 5, and 7 would apply.

The total number of combinations for any given implementation using a key having 24 bits is 16,777,216. Because RNE is based on bit string manipulation, there is no upper limit on the length of the encryption key.

The second element of RNE, the rule-base **54**, is a set of rules **64** used by the command processor **50** to decode or encode binary strings of data. The rules **64** may be used individually, or as a set to decode or encode data.

The rule-base **54** is not defined as a specific type of data structure. The rule-base **54** may, for example, be stored as a secure table in a relational database, as a C or C++ language header file, as an object class library, as a dynamic link library, or as an EPROM assembly language subroutine, or as a microcode component within a microprocessor.

In addition, access to the rules **64** within a rule-base **54** may be accomplished by the command processor **50** using one or more data structures including, but not limited to, linked lists, tree structures, relational tables, object class libraries, hash tables, or hyper-link stacks.

Following are examples of n-bit binary string operations which may be used to encrypt the input data. Consider, first, examples of vector rules and their explanation.

Examples of Vector Rules

The following rules provide simple examples of the ways bit strings may be manipulated. The number and combination of possible rules is infinite.

Inversion

Invert the following n bits:

Where n=7 1011100 becomes 0100011

Transposition

Transpose the following n pairs of bits:

Where n=3 10 11 10 becomes 01 11 01

Interleaving

Interleave with a ratio 1:1 the following pair of n bits:

Where n=4 1011 1001 becomes 1100 1011

Shift Left

Shift the following n bits x bits to the left:

Where n=5, x=1 11011 becomes 10111

Shift Right

Shift the following n bits x bits to the right:

Where n=5, x=1 11011 becomes 11101

Consider, next, examples of matrix or two-dimensional rules and an explanation of their use.

Examples of Matrix Rules

For each of the following examples, the bit stream 0110 0010 1110 0101 1100 will be the input stream.

Rule 1

Step 1. Enter bits from left to right, top to bottom, into a matrix with 5 rows and 4 columns.

0	1	1	0
0	0	1	0
1	1	1	0
0	1	0	1
1	1	0	0

Step 2. Rotate the matrix 90 degrees to the left resulting in a matrix with 4 rows and 5 columns.

0	0	0	1	0
1	1	1	0	0
1	0	1	1	1
0	0	1	0	1

Step 3. Write the bits from left to right, top to bottom
Result: 0001 0111 0010 1110 0101

Original: 0110 0010 1110 0101 1100

Note: This is equivalent to a reverse interleave of X, n-bit data types, where X=5, n=4. The larger the values of X and n, the larger the adjacency displacement.

Rule 2

Step 1. Enter the bits from left to right, top to bottom, into a matrix with 4 rows and 5 columns.

0	1	1	0	0
0	1	0	1	1
1	0	0	1	0
1	1	1	0	0

Step 2. Rotate the matrix 180 degrees.

0	0	1	1	1
0	1	0	0	1
1	1	0	1	0
0	0	1	1	0

Step 3. Invert the bits.

1	1	0	0	0
1	0	1	1	0
0	0	1	0	1
1	1	0	0	1

Step 4. Write the bytes from left to right, top to bottom.
Result: 1100 0101 1000 1011 1001
Original: 0110 0010 1110 0101 1100

Rule 3

It is possible to use multiple arrays to encrypt bit streams and to combine vector rules with array operations. The following rule uses three matrices. The first matrix (a) is 3 rows by 4 columns. The second matrix (b) is 2 rows by 3 columns. The third matrix (c) is 1 row by 2 columns.

Step 1. Read the first 12 bits of the input stream into 3 4-bit data types.
0110 0010 1110

Step 2. Treating each data type individually, shift each bit 1 bit to the left.
1100 0100 1101

Step 3. Fill matrix (a) entering each data type into each of the three rows.

(a)

1	1	0	0
0	1	0	0
1	1	0	1

Step 4. Enter the remaining bits from left to right, top to bottom, into matrices (b) and (c).

(b)

0	1	0
1	1	1

(c)

0	0

Step 5. Swap matrices (b) and (c).

(c)

0	0

(b)

0	1	0
1	1	1

Step 6. Rotate all three matrices 90 degrees to the left.

(a)

0	0	1
0	0	0
1	1	1
1	0	1

(c)

0
0

-continued

(b)

0	1
1	1
0	1

Step 7. Write the bits from left to right, top to bottom.

Result: 0010 0001 1110 1110 1001

Original: 0110 0010 1110 0101 1100

Additional rules may be created for any n-dimensional data representation. There is no upper limit on the number of possible rules or the manner or order in which they are implemented.

The use of public and/or private specific purpose keys such as encryption keys is optional. The encryption and decryption of data is not dependent on an explicit encryption key, a specific encode/decode rule or a specific data type. An uncountable number of encryption keys may be applied against a single implementation of RNE each one of which enforces a unique rule-set, each set containing unique encode/decode rules, in a unique order. Each encryption key may be explicitly or implicitly defined. Thus by providing multiple encryption keys to a single encrypted message, parts of the message will be available only to some users and not others when the message is distributed across multiple machines for multiple users.

The successful encryption or decryption of the data is not dependent on the size of the encryption key or the speed of capacity of the processor. Without access to the encryption key, the command processor rules, and the rule-base, it is impossible to establish a correspondence between the original data and the encrypted data.

For example, and not by way of limitation, the following representative program listing in C details a second example implementation of Rule-based n-Bit Encryption (RNE).

```
/*_______________________________________________________
BME.C          Binary Matrix Encryption
               An Example Implementation of Rule-Based n-Bit Encryption.

Program Development Log:

Release of Version 1.10
_______________________________________________________*/

#include <stdio.h>
#include <io.h>
#include <malloc.h>
#include <conio.h>
#include <math.h>
#include <graph.h>
#include <string.h>
#include <mstring.h>
#include <stdlib.h>
#include <mstdlib.h>

#define YES         1
#define NO          0

#define IN          'I'
#define OUT         'O'
#define OFF         0
#define ON          1

#define FULL        0
#define NOTFULL     1
```

```
/*____________

Matrix Definition
_____________*/

typedef struct matrixTAG { unsigned int x, y;
                                              unsigned long size;
                                              char *matrix; } MATRIX;

/*____________

Function Definitions
_____________*/

char AddByteToMatrix(unsigned char);
void AllocateBuffers(void);
void AllocateLargestMatrix(void);
void CleanUp(char *);
void ClearMatrix(void);
void ClearBitCan(void)
void DefineMatrix(void);
void DisplayProgress(void);
void DisplayTitle(void);
int FillMatrix(void);
void FlushOutBuffer(void);
unsigned char GetBitStrVal(char *);
int GetNextByte(void);
void GetOptimalScale(void);
void InvertMatrix(void);
void NoArguments(void);
void OpenInFile(char *);
void OpenOutFile(void);
void PutNextByte(int);
void SendBitAtXY(long, long, unsigned int);
void SetEncryptionMethod(char *);
void SetInversionFreq(char *);
void SetMode(char);
void SetNextMethod(void);
void SetPassWord(char *)
void SwapScale(void);
int TimeToInvert(void);
void WriteBit(char);
void WriteReverse(void);

/*______________________________________________________________

Matrix Encryption Functions

The following matrix encryption functions are the most simplistic variations on
arbitrarily approaching the matrix. Even so, these functions confuse the data
so much that no one could, with any degree of certainty decipher even one of
the rules contained here.

There are ways to make and process three-dimensional matrices, and virtually
any 2D or 3D shape would work. There are no means possible to figure out
the total number of rules that can be applied to this method, since, of course,
another bit could always be added, the shape or size changed, or some other
variations utilized like adding an arbitrary bit every X number of bits. Thus,
one would not know from where all the bits came, or to where they should go.

The number of rules is really the limit of the imagination's ability to create
them.
______________________________________________________________________*/

#define NUM_RULES 7                         /*There are actually 8 rules*/
                                            /*butTopRightDown           */
                                            /*doesn't do anything.      */
void BottomLeftUp(void);
void BottomRightUp(void);
void BottomUpLeft(void);
void BottomUpRight(void);
void TopDownLeft(void);
void TopDownRight(void);
void TopLeftDown(void);


/*_____________________

File Pointers and File Names
_________________________*/

FILE *InFile, *OutFile;
char OutFileName[L_tmpnam];
```

```
/*________________

Global Variables
________________*/

#define MAX_BUFFER 16384

unsigned char *InBuffer, *OutBuffer;
int OutByteCount = 0;

MATRIX Matrix;
unsigned long MatrixCount = 0L;

char *PassWord;
unsigned PassWordLen = 0

char BitCan[9];
unsigned char LeftOver[9];


/*________________________________________
```

The following data type is an array of pointers to the various encryption rules. It acts as the rule-base interface. Since the rules are set up as an array of function pointers, not only can rules be added or subtracted with great ease, they can be reordered. In other words, this particular example program can be compiled as shown to produce an encryption engine. Then the rule order can be changed in the pointer array defined below. Then the program can be recompiled to create another encryption engine. The two resulting encryption engines would be theoretically identical, but the first would not be able to decode a data stream encoded by the second, and vice-versa. With enough rules, therefore, as many unique encryption engines could be created as desired. Each one of those unique encryption engines would have hundreds of billions of password-method implementations

Also, because the rules are set in an internal data-array, it would be easy to add rule-order rules, so that an operation like the checksum of the incoming data would trigger a rule that reorders the rule pointers on the fly. In other words, not only does the end-user have some control over the encryption, but the data itself would be a player in the total scheme of things. Thus, one who

```
is uninformed would have to know beforehand the unencrypted version of the
data before attempting to decrypt it.
_____________________________________________________*/

void (*Rule[NUM_RULES](void) = {BottomUpLeft, BottomUpRight,
                                 BottomLeftUp, BottomRightUp,
                                 TopLeftDown,  TopDownLeft,
                                 TopDownRight };

int *EncryptionMethod;
unsigned RuleID = 0;
char Mode = IN

unsigned long FileSize = 0L;
unsigned long TotalBytesRead = 0L;


/*_________________

User Definable Switches
_________________*/

unsigned char Inversion = OFF;
unsigned long InversionFreq = 0L;


/*____________________________________________________

Program Code Follows

BME mode filename password1 password2 {/i=x}

        -   mode is either I or O.
        -   filename is the file to encrypt.
        -   password1 defines the sizes of the matrices. This way, the matrices
            vary in size as the file is encrypted. Therefore if a hacker were to
            experience some miracle of guessing the correct matrix size, the
            correct method, and whether it had been inverted, the rest of the
            file would still be junk to him. Pass it twice, and he would never
            know that he got it right in the first place!
        -   password2 defines the way that the rules are used. Each letter of
            password 2 represents, rather arbitrarily (see later) a rule to use for
```

encryption/decryption. That way, the rules to use for each matrix are randomly selected by the use of password2. This is very helpful. Because the command processor can rotate through password1 and password2, a given matrix size is most likely going to be encrypted different ways each time it is used.

- For this implementation and just to complicate matters, the /i=x switch was added to provide the requirement that for each frequency of X matrices, invert the Xth matrix. This way, the uninformed would not even know the original value of any given bit in the file! So a '1' is present. Was it originally '1' or '0'? There is no way to know without knowing who encrypted the file.

```
______________________________________________________________*/

void main(int argc, char *argv[])
{
     char lastbit = 0;

     DisplayTitle();

     if (argc <5)
               NoArguments();

     AllocateBuffers();

     SetMode(argv[1][0]);

     OpenInFile(argv[2]);
     OpenOutFile();

     SetPassWord(argv[3]);

     DefineMatrix();

     SetEncryptionMethod(argv[4]);

     switch (argc) {
               case 6:
                         SetInversionFreq(argv[5]);
     }

     while (FillMatrix() !=EOF) {
        ++MatrixCount;

        DisplayProgress();

        if(TimeToInvert() == YES)
                InvertMatrix();

        (*Rule[RuleID])();

        DefineMatrix();

        SetNextMethod();

    }
    if(strlen(Matrix.matrix)) {
            if(strlen(Matrix.matrix) !=Matrix.size) {

                Matrix.size = strlen(Matrix.matrix);
                GetOptimalScale();

                if (Matrix.size == 0) {

                    lastbit = *(Matrix.matrix + Matrix.size - 1);
                    --Matrix.size;

                    *(Matrix.matrix + Matrix.size) = '\0';

                    GetOptimalScale();
                }
            }

            (*Rule[RuleID])();

            if(lastbit)
                    WriteBit(lastbit);
    }

    CleanUp(argv[2]);
    exit(0);
}
```

```
/*______________________________________________________________

These functions are involved in setup and shutdown.
______________________________________________________________*/


/*______________________________________________________________

SetMode - This is where from the command line, either a 'I' or a 'O' is used.
It really doesn't matter which is used as long as the opposite letter is used to
decrypt as was done to encrypt.  In other words, if I is used to encrypt, O is
used for the decrypt, and vice-versa.
______________________________________________________________*/

void SetMode(char mode)
{
          Mode = toupper((int)mode);

          if(Mode !=IN && Mode !=OUT) {
               printf("\nInvalid Mode: [%c].  Use IN or OUT\n",Mode);
               exit(1);
          }
}

/*______________________________________________________________

Buffer the input and output.
______________________________________________________________*/

void AllocateBuffers(void)
{
          InBuffer  =  (unsigned  char  *)malloc(MAX_BUFFER  *
          sizeof(unsigned char));
          OutBuffer  -  (unsigned  char  *)malloc(MAX_BUFFER  *
          sizeof(unsigned char));

          if (InBuffer == NULL || OutBuffer == NULL) {
                    puts("\nNot Enough Memory\n");
                    exit(1);
          }
```

```
}

/*____________________________________________________

Open the file to encrypt or decrypt.
___________________________________________________*/

void OpenInFile(char *fname)
{
        if((InFile = fopen(fname, "rb")) == (FILE *)NULL {
                printf("\nCan't Open %s for Input\n", fname);
                exit(1);
        }

        FileSize = filelength(fileno(InFile));
}

/*____________________________________________________

Open a file in which to write the encrypted/decrypted data.  A temporary
name is used here for example only.
___________________________________________________*/

void OpenOutFile(void)
{
    tmpnam(OutFileName);

    if((OutFile = fopen(OutFileName, "wb")) == (FILE *)NULL) {
            printf("\nCan't Open %s for Output\n", OutFileName);
                exit(1);
    }
}

/*____________________________________________________

This is the user's first password argument derived from the fourth command
line argument.  If there is not an even number of characters, one character is
added so that an even number of characters is used for the matrices.  It does
```

not have to be done this way. It could be done any number of ways in this particular application. Here, the first password is used to define the matrix sizes. It is made to be an even number of characters so there is a Y for every X. If a new character is not added to bring the first password length to an even number of characters, that new character is simply assigned the same value as the first character of the password. That, too, is arbitrary and can actually be done any number of ways.

Here, the ASCII decimal value of the character is used as the matrix axis value. For example, suppose the password is 'ABCD'. The ASCII decimal values for A, B, C, and D are 65, 66, 67, and 68, respectively. The first matrix axes are defined using A and B. The size of the matrix, therefore, is 65 bits by 66 bits = 4290 bits. The next matrix axes are defined using C and D, thus it's size is 67 by 68 bits. Since, in this example, there are no more characters, simply begin again at the beginning of the password with A and B. Since the password is case-sensitive, all 256 ASCII characters are usable.

```
________________________________________________________*/

void SetPassWord(char *passwordarg)
{
        unsigned int len, size;

        size = len = strlen(passwordarg);

        if(len % 2)
                ++size;

        PassWord = (char *)malloc((size + 2) * sizeof(char));

        if(PassWord == NULL) {
                puts("\nOut of Memory\n");
                exit(1);
        }

        strcpy(PassWord, passwordarg);

        if(len != size) {
                *(PassWord + len) = *passwordarg;
                *(PassWord + size) = '\0';
        }

        PassWordLen = size;
```

```
    }
/*______________________________________________________________

Here, the second password provided by the user is used as a tool to decide
which rule to use.

The ASCII character set is implicitly divided into blocks, each the size in
characters as the number of rules available. Then the relative position of each
password2 character within its block is found. That position is one of the
numbers 0 to NUM_RULES - 1. This points, then, to a rule in the rule-base.

This is a good way to do it since, at any time, the number or order of rules in
the rule-base can be changed. If the order or number of rules is changed, then
this code does not have to be changed.
______________________________________________________________*/

void SetEncryptionMethod(char *method)
{
        unsigned int len, val, x;
        int *eptr;

        len = strlen(method) + 1;

        EncryptionMethod = (int *)malloc(len * sizeof(int));

        for(eptr = EncryptionMethod; *method !='\0'; method++,
        eptr++) {

            val = (*method / NUM_RULES) * NUM_RULES;

            for (x = 0; x <NUM_RULES; x++)
                    if ((val + x) == *method)
                            *eptr = x;
        }
        *eptr = -1;

        RuleID = *EncryptionMethod;
}
```

/*

This code takes the optional command line switch and parses it for inversion frequency. The program then inverts each Xth matrix before encrypting it.

Consider an example. Suppose the user enters the following command line:

BME I filename ABC 12345 /i=3

The first password 'ABC' is lengthened to 'ABCA' so that it will contain an even number of characters. The letters are then converted to the ASCII decimal equivalents (65, 66, 67, 65) and used, alternately, as X and Y matrix axis values. The second password '12345', by application, translates to rule pointers 0, 1, 2, 3, and 4, respectively. The '/i=3' sets the inversion frequency to 3, telling the program to invert every third matrix prior to its alteration by its assigned rule.

The following table describes each matrix size, which rule is use, and whether or not the matrix is inverted.

Matrix Size in Bits			Rule Used	Inverted?	# Bytes
X	Y	Size	0 to Total	Y or N	Accum.
65	66	4290	0	N	536 1/4
67	65	4355	1	N	1080 5/8
65	66	4290	2	Y	1616 7/8
67	65	4355	3	N	2161 1/4
65	66	4290	4	N	2697 1/2
67	65	4355	0	Y	3241 7/8
65	66	4290	1	N	3778 1/8
67	65	4355	2	N	4322 1/2
65	66	4290	3	Y	4858 3/4
67	65	4355	4	N	5403 1/8

...and so on to 30 different entries before it begins to repeat.

If the command line is symbolized as:

```
BME mode filename P1 P2 /i=F.
```

the way to calculate the number of table entries is:

	# Chars in P1 (rounded up to the nearest even number)
Divided by	2
Multiplied by	# Chars in P2
Multiplied by	F(if defined)

Therefore, this example is:

```
P1 = 'ABC', P2 = '12345', F=3
```

The number of characters in P1 = 3, rounded up to the nearest even number equals 4. The number of characters in P2 = 5. Therefore, the number of table entries for this example is:

$$(4/2) * 5 * 3 = 2 * 5 * 3 = 30$$

```
______________________________________________*/

void SetInversionFreq(char *freq)
{
     freq +=3;

     InversionFreq = atol(freq);
     if (InversionFreq <1L)
               InversionFreq = 0L;
     else
               Inversion = ON;
}


/*______________________________________________
```

This function writes out all remaining data, closes files, deletes the original file, renames the encrypted temporary file to the name of the original file, and then cleans up all allocated memory.

```
______________________________________________*/
```

```
void CleanUp(char *fname)
{
     FlushOutBuffer();

     fclose(InFile);
     fclose(OutFile);

     unlink(fname);
     rename(OutFileName, fname);

     DisplayProgress();

     free(Matrix.matrix);
     free(InBuffer);
     free(OutBuffer);
     free(PassWord);
     free(EncryptionMethod);


/*____________________________________________

Provide instructions if procedures not correctly followed.
____________________________________________*/

void NoArguments(void)
{
     puts("\nBME mode infile password method {/i=n}\n");
     puts ("Modes are: IN | OUT");
     exit(1);
}


/*_______________

Other Functions
_______________*/

void Display Title(void)
{
     puts("\nBME - Binary Matrix Encryption");
     puts("Version 1.0 - Release 09.94");
```

```
        puts("(c) 1994: Michael D. Harold & Joseph M. Morgan\n");
}

void Display Progress(void)
{
        printf("\rProcessing:  (%6.2f%c)",  ((float)TotalBytesRead  /
        (float)FileSize * 100., 37);
}

/*______________________________________________________

The following functions handle the matrix.
_____________________________________________________*/

/*______________________________________________________

This function manages the rotation through password1 using adjacent
characters to set the matrix size. Note that this implementation forces a matrix
axis to be at least 2.
_____________________________________________________*/

void DefineMatrix(void)
{
        static unsigned int PassWordPtr = 0;
        static char MatrixDefined = NO;

        if (MatrixDefined == NO)
                AllocateLargestMatrix();

        Matrix.x = (long)*(PassWord + PassWordPtr++);
        Matrix.y = (long)*(PassWord + PassWordPtr++);

        if (PassWordPtr == PassWordLen)
                PassWordPtr = 0;

        if (Matrix.x <2)
                Matrix.x = 2;

        if (Matrix.y <2)
        Matrix.y = 2;

    Maxtrix.size = Matrix.x * Matrix.y;

    MatrixDefined = YES;
}

/*________________________________________________

Instead of allocating and freeing memory as the matrices shift their size, the
largest matrix is allocated and that space used.
________________________________________________*/

void AllocateLargestMatrix(void)
{
    unsigned size, MaxSize = 0, x, y;
    char *ptr = PassWord;

    while (*ptr !='\0') {
            x = *ptr++;
            y = *ptr++;

            size = x * y;
            if (size > MaxSize)
                    MaxSize = size;
    }

    Matrix.matrix = (char *)malloc((MaxSize + 1) * sizeof(char));

    if (Matrix.matrix == (char *)NULL) {
            puts("\nOut of Memory\n");
            exit(1);
    }
}

/*________________________________________________

This function manages the conversion from byte-based reading to bit-based
reading.  Note that a matrix is simply an n-Bit word.  So, an n-Bit word of
```

```
matrix size is filled with bits from the input.  To optimize the process (and it
could be better), bits are added to the matrix 8-at-a-time.
____________________________________________________________*/

int FillMatrix (void)
{
     int c;

     ClearMatrix();

     if (strlen(LeftOver)) {
               strcpy(Matrix.matrix, LeftOver);
               LeftOver [0] = '\0';
     }

     if (strlen(Matrix.matrix) == Matrix.size)
               return 1;

     while ((c = GetNextByte()) != EOF) {

          TotalBytesRead++;

          if (AddByteToMatrix((unsigned char)c) == FULL) break;
     }

     if (c == EOF && !(feof(InFile))) {
          puts("\nUnexpected EOF");
          exit(1);
     }

     return c;
}

/*____________________________________________________________

To keep things neat, even though unnecessary, the following program sets the
matrix to NULL.
____________________________________________________________*/

void ClearMatrix(void)
{
        unsigned long x;

        for (x = 0; x < =Matrix.size; x+ +)
                *(Matrix.matrix + x) = '\0';
}


/*____________________________________________

This function adds bits to the matrix, up to 8-at-a-time.  If it is unable to add
all 8 bits to the matrix, it copies the remaining bits to a holding bin called
LeftOver.  Note its use above in FillMatrix.
____________________________________________*/

char AddByteToMatrix(unsigned char byte)
{
        char bytestr[9];
        int nbits;

        chartobinstr(byte, bytestr);

        nbits = (Matrix.size - strlen(Matrix.matrix));
        nbits = (nbits > 8) ? 8 : nbits;

        strncat(Matrix.matrix, bytestr, nbits);

        if (nbits < 8)
                strcpy(LeftOver, (bytestr + nbits));

        return  (strlen(Matrix.matrix)  = =  Matrix.size)  ?  FULL  :
        NOTFULL;
}


/*____________________________________________

This next three functions control the physical input and output of data via two
16K buffers.
____________________________________________*/

int GetNextByte(void)
}
```

```
    static int BytePtr = 0;
    static int BytesInBuffer = 0;

    if (BytePtr == BytesInBuffer) {
            BytesInBuffer = fread(InBuffer, sizeof(unsigned char),
            MAX_BUFFER - 1, InFile);

            if (BytesInBuffer == 0)
                    return EOF;

            BytePtr = 0;
    }

    return *(InBuffer + BytePtr++);
}

void PutNextByte(int c)
{
    *(OutBuffer + OutByteCount) = c;

    if (++OutByteCount == MAX_BUFFER - 1) {
            fwrite(OutBuffer,sizeof(unsigned char), MAX_BUFFER - 1,
            OutFile);
            OutByteCount = 0;
    }
}

void FlushOutBuffer(void)
{
    fwrite(OutBuffer, sizeof(unsigned char), OutByteCount, OutFile);
}


/*______________________________________________________________

This function handles the output of single bits. It accumulates bits, for
example only, into an 8-bit buffer to translate back into bytes to accommodate
the architecture. Once 8 bits are accumulated, it is converted into a real byte
and sent to the output buffer.
________________________________________________________________*/

void WriteBit(char bit)
```

```
{
    static int BitCanPtr = 0;
    BitCan[BitCanPtr++] = bit;

    if (BitCanPtr == 8) {
            PutNextByte(GetBitStrVal(BitCan));
            BitCanPtr = 0;
            ClearBitCan();
    }
}

void ClearBitCan(void)
{
    int x;

    for (x = 0; x < 8; x++)
            BitCan[x] = '\0';
}


/*______________________________________________________

The following function receives a bit stream and converts it into an actual
unsigned character value.
_______________________________________________________*/

unsigned char GetBitStrVal(char *bitstr)
{
    unsigned char value, x;

    for (x = 128, value =0; *bitstr !='\0'; bitstr++, x /=2)
            if (*bitstr == '1')
                    value += x;

    return value;
}
```

```
/*______________________________________________________

The following function determines when it is time to invert the matrix.
Returns YES or NO.
______________________________________________________*/

int TimeToInvert(void)
{
    if (Inversion == OFF || MatrixCount < InversionFreq ||
    !InversionFreq)
        return NO;

    if (!(MatrixCount % InversionFreq))
        return YES;

        return NO;

}

/*______________________________________________________

Following are various rules by which to encrypt the matrix. Envision the X
axis as vertical and the Y axis as horizontal.
______________________________________________________*/


/*______________________________________________________

The following rule writes bits beginning from the bottom right corner of the
matrix, moving left through that row, then moving up one row to the end and
continuing in that manner until the matrix is completely written out.
______________________________________________________*/

void BottomLeftUp(void)
{
    register long x, y;
    unsigned int len;

    len = strlen(Matrix.matrix);

    for (x = Matrix.x - 1; x>=0; x--)
```

```
                for (y = Matrix.y - 1; y>=0; y--)
                        SendBitAtXY(x, y, len);
}


/*___________________________________________________________

The following rule starts at the bottom left corner, moves right through the
row, moves up one row to the beginning and continues until the matrix is
written out.
___________________________________________________________*/

void BottomRightUp(void)
{
        register long x, y;
        unsigned int len;

        len = strlen(Matrix.matrix);

        for (x = Matrix.x - 1; x>=0; x--)
                for (y = 0; y < Matrix.y; y++)
                        SendBitAtXY(x, y, len);
}


/*___________________________________________________________

The next rule starts at the bottom right corner, moves up through the column,
then moves left one column at the bottom, and continues.
___________________________________________________________*/

void BottomUpLeft(void)
{
        register long x, y;
        unsigned int len;

        if (Mode == OUT)
                SwapScale();

        len = strlen(Matrix.matrix);

        for (y = Matrix.y - 1; y >=0L; y--)
```

```
            for (x = Matrix.x - 1; x > =0L; x--)
                    SendBitAtXY(x, y, len);
}

/*____________________________________________________

This rule starts at the bottom left corner, moves up through the column, moves
right to the bottom of the next column, and continues.
___________________________________________________*/

void BottomUpRight(void)
{
    register long x, y;
    unsigned int len;

    if (Mode = = OUT) {
            strrev(Matrix.matrix);
            SwapScale();
    }

    len = strlen(Matrix.matrix);

    for (y = 0L; y < Matrix.y; y+ +)
            for (x = Matrix.x - 1; x > =0; x--)
                    SendBitAtXY(x, y, len);
}

/*____________________________________________________

This rule starts at the top right corner, moves down through the column,
moves left to the bottom of the next column, and continues.
___________________________________________________*/

void TopDownLeft(void)
{
    register long x, y;
    unsigned int len;

    if (Mode = = OUT) {
```

```
                strrev(Matrix.matrix);
                SwapScale();
        }

        len = strlen(Matrix.matrix);

        for (y = Matrix.y - 1; y > =0; y--)
                for (x = 0L; x < Matrix.x; x + +)
                        SendBitAtXY(x, y, len);
}


/*______________________________________________________________

The next rule starts at the top left corner, moves down through the column,
then right to the top of the next column, and continues in like manner.
_____________________________________________________________*/


void TopDownRight(void)
{
        register long x, y;
        unsigned int len;

        if (Mode = = OUT) {
                SwapScale();
        }

        len = strlen(Matrix.matrix);

        for (y = 0L; y < Matrix.y; y+ + )
                for (x = 0L; x <Matrix.x; x+ + )
                        SendBitAtXY(x, y, len);
}


/*______________________________________________________________

The next rule starts at the top right corner, moves left through the row, down
and to the end of the next row, and continues in like manner.
_____________________________________________________________*/
```

```
void TopLeftDown(void)
{
    register long x, y;
    unsigned int len;

    len = strlen(Matrix.matrix);

    for (x = 0; x < Matrix.x; x++)
            for (y = Matrix.y-1; y >=0; y--)
                    SendBitAtXY(x, y, len);
}


/*_____________________________________________________

This rule starts at the top left corner, moves right through the row, down and
to the beginning of the next row, and so on.  Since this is exactly the way the
matrix is constructed, it doesn't actually perform any encryption.  It is put here
to complete the rule set, but this implementation does not use it.
_____________________________________________________*/

void TopRightDown(void)
{
        register long x, y;
        unsigned int len;

        len = strlen(Matrix.matrix);

        for (x = 0; x < Matrix.x; x++)
                for (y = 0; y < Matrix.y; y++)
                        SendBitAtXY(x, y, len);
}


/*_____________________________________________________

These are functions common to the matrix conversion rules.
_____________________________________________________*/
```

```
/*______________________________________________________________

This function is required to decrypt certain rules.  The matrix axes have to be
swapped to properly recover the data.
______________________________________________________________*/


void SwapScale(void)
{
            unsigned long n;

            n = Matrix.x;
            Matrix.x = Matrix.y;
            Matrix.y = n;
}


/*______________________________________________________________

This function inverts the matrix, turning every '1' bit to a '0', and every '0'
bit to a '1'.
______________________________________________________________*/

void InvertMatrix(void)
{
      char *ptr;

      for (ptr = Matrix.matrix; *ptr !='\0'; ptr++)
            *ptr = (*ptr=='1')?'0':'1';
}


/*______________________________________________________________

This function manages the rotation through the rule pointers defined by
password2.
______________________________________________________________*/


void SetNextMethod(void)
{
      static int MethodPtr = 1;
```

```
     RuleID = *(EncryptionMethod + Method Ptr++)

     if (*(EncryptionMethod + MethodPtr) == -1)
          MethodPtr = 0;
}


/*________________________________________________________

This function locates the bit in the matrix at x, y and writes it to the output
stream.
________________________________________________________*/


void SendBitAtXY(long x, long y, unsigned int len)
{
     unsigned long offset;

     offset = (x * Matrix.y) + y;

     if (offset < (long)len)
          WriteBit(*(Matrix.matrix + offset));
}


/*________________________________________________________

This function defines the scale by the number of bits in the n-bit data type.
Very few files will divide up evenly into the shifting matrix series. Therefore,
there will usually be an incomplete matrix at the end. These rules won't
properly encrypt/decrypt a partial matrix. Therefore, this function resets the
matrix scale to an optimal axis based upon the number of remaining bits.
However, if the length of the remaining n-bit word is prime, the last bit is held
and removed from the matrix, and then the optimal axis is calculated. Once
the new matrix is encrypted, if there is a remaining bit, it gets written out.
________________________________________________________*/

void GetOptimalScale(void)
{
          unsigned ResultNumber;
          unsigned HighLimit;
          unsigned x;
```

```
        HighLimit = Matrix.size;

        Matrix.x = Matrix.y =0;

        for (x = 2; x < HighLimit; x++) {

                if (Matrix.size % x)
                        continue;

                ResultNumber = HighLimit = Matrix.size / x;

                Matrix.x = x;
                Matrix.y = ResultNumber;
        }

        Matrix.size = Matrix.x * Matrix.y;
}

/* End of program code */
```

FIG. 4 illustrates, by example, the elemental steps of data compression. Compression is begun by the initiation process step **80** which includes such items as the declaration and initialization of variables, the allocation of memory, and parsing user input. The next step **82** involves pointing to the input data, output data, and other support data needed by the compression method(s) at step **86**.

From there, the process begins by reading in an implementation of a defined amount of data. It is specific to this invention that the data is read as one or more n-bit data types at step **84** consistent and parallel with any given implementation. The data is then processed by the compression method(s) at step **86**. The resulting data, now in compressed form, is written to output data at step **88**.

It is then determined at step **90** whether or not compression has been completed. If not, it is determined if the end of the input data has been reached at step **92**. If not, the process forks backs to the read process at step **84**, otherwise it forks to the application of the compression method(s) at step **86**.

If it has been determined by the implementation that compression has been completed at step **90**, then the necessary data is released at step **94**, and the process is completed at step **96**.

For example, and not by way of limitation, the following description illustrates one method of implementing an n-bit virtual software machine capable of performing reiterative loss-less data compression.

Most methods of loss-less data compression are based on a method in which repetitive patterns or symbols within a data file are first identified and then replaced with symbols which occupy, on average, less space in memory than the original symbols. Examples of loss-less data compression techniques include Huffman Coding, Arithmetic Coding, Dictionary-Based Compression and Lempel-Ziv Coding. Each of these methods relies on the substitution of a smaller binary string for a larger binary string based on one or more repetitive patterns of symbols or symbol patterns, or the frequency of bytes or patterns within the uncompressed data. The desired result of this process is a file which is smaller than the original.

In order for compression to occur, an uneven frequency distribution of symbols or symbol patterns must be present in the uncompressed file. The greater the unevenness of the frequencies of the symbols or symbol patterns in the original data, the greater the compression.

Currently, all known methods of loss-less data compression result in an even distribution of symbols in the compressed file. Because loss-less data compression methods rely upon the uneven distribution of symbols or symbol patterns, the even frequency distribution makes further compression undesirable or impossible.

With this invention, the concept of Reiterative n-Bit Compression (RNC) uses variable length n-bit data types to explicitly or implicitly redistribute the frequency with which symbols occur within the data. After one iteration, the frequency distribution of the symbol set representing the data may be modified explicitly by changing the size in bits of the input and/or output data type. This explicit frequency redistribution of the symbol set allows the data to be compressed reiteratively.

The following table illustrates the differences in the resulting distribution of some 8-bit characters following compression with different sized n-bit data types.

Characters	Space	A	E	I	O	U
	Original Distribution Count of Each of the Above Characters					
Data Type Size in Bits	4376	498	749	836	352	283
	Distribution Count Following Compression					
4	143	114	114	112	46	33
6	217	122	117	138	59	48
8	139	42	59	42	25	29
10	260	107	120	129	40	42
12	161	64	64	61	13	6

The first row of the above table lists the characters being evaluated for their frequency within the original file. The third row shows each character's distribution in the original input file prior to its compression. Rows 5 through 9 list each character's distribution following compression of the original file using input data types of varying bit lengths. The first column lists the length, in bits, of the input n-bit data type.

Referring to the table, the "space" character appears 4376 times in the original file. After compressing the file using a 4-bit data type, the "space" appears 143 times. In contrast, after the file is compressed using a 10-bit data type, the "space" appears 260 times. When the file is compressed using a 6-bit data type, the character 'U' appears 48 times, compared with only 6 times following compression using a 12-bit data type.

By forcing the redistribution of bits, the compression ratio can be forced to vary, thereby permitting optimization of the next compression pass. The following table shows an example of compression results of data. The first column lists the size, in bytes, of the file before a given compression pass. The second column lists the size of the n-bit data type. The third column lists the number of the actual compression pass. The fourth column lists the size of the file, in bytes, following the compression pass.

Some compression passes have been attempted more than once with varying sized input data types. This allows for selection of the best compression ratio for a given n-bit data type. Compression attempts that do not result in compression are italicized. The compression pass that achieved the most compression is boldfaced.

Size	Nbits	Pass	Comp. Size
18119	10	1	14174
14174	10	2	14157
14157	10	3	14156
	9	3	14136
	8	3	13900
	7	3	14091
	6	3	14059
	5	3	14558
13900	8	4	13832
	7	4	13900
	6	4	13907
	9	4	13897
	10	4	13894
	11	4	13911

It should also be noted that frequency redistribution can be achieved by implementing such data conversion algorithms as alternating block-based inversion, bit-shifting, or exclusive OR operations tailored for the target, or a multiple of the target data type.

As another example, and not by way of limitation, the following description illustrates one method of implementing a virtual machine or computer capable of performing rule-based n-bit arbitrary precision arithmetic: Rule-based n-Bit Arithmetic (RNA) performs binary arithmetic operations on fixed or floating point data of arbitrary precision where such precision is limited only by the real or virtual address space of the computer. Arithmetic operations include, but are not limited to, binary addition, subtraction, multiplication, and division.

As part of its method, RNA contains two new data types whose notations are unique and are not described in any previously existing data types or data notations. One of these notations represents integer values. The other notation represents floating point values.

The majority of computers and computer languages now conforms to the following internal representation of integer values:

1. The left-most bit, the sign bit ("S"), is used to contain the sign of the number, with the usual interpretation that 0 means positive or plus and 1 means negative or minus.
2. The "decimal" or radix point is assumed to be affixed at the left or right end of the number.
3. The remaining bits represent the binary values ("B") of the number.

A standard signed 16-bit value would be stored as follows:

S (1 bit)	B (15 bits)	
		ˆ radix

With regard to floating point numbers, the following specific notation endorsed by the IEEE (Institute of Electrical and Electronic Engineers) is the standard:

1. The left-most bit is the sign bit ("S").
2. The next eight bits are the exponent ("E"). The exponent is interpreted as an integer in excess-127 code. Excess-127 code allows the exponent to represent numbers from -127 through 128.
3. The remaining bits are the mantissa ("M"). The value of the mantissa is normally defined as 1 plus the value of "M" treated as a binary fraction with the radix at the left end.

A standard signed, single precision floating point value would be stored as follows:

S (1 bit)	E (8 bits)		M (23 bits)
		ˆ radix	

This data type has an upper limit of 64 bits (double precision) and 128 bits (quadruple precision).

The following data types represent RNA integer and floating point values.

RNA integer values are represented as follows:

1. The left-most bit, the sign bit ("S"), is used to contain the sign of the number with the interpretation that 0 means positive and 1 means negative.
2. The first n-Bit value to the right of the sign bit identifies the length in bits ("L") of the binary representation of the number. The size and limit of this value is implementation specific.
3. The second n-Bit value to the right of the sign bit represents the binary values ("B") of the number beginning with the least significant digit (LSD) and extending to the most significant digit (MSD).

A signed n-Bit integer value would be stored as follows:

S (1 bit)	L (n bits)	B (L bits)

For example, any binary integer value from 1 to 16,777,216 significant digits in length could be stored with the following n-Bit data type:

S (1 bit)	L (24 bits)	B (L bits)

RNA floating point values are represented as follows:

1. The left-most bit, the sign bit ("S"), is used to contain the sign of the number with the interpretation that 0 means positive and 1 means negative.
2. The first n-Bit value to the right of the sign bit identifies the length in bits ("L") of the binary representation of the number. The size and limit of this value is implementation specific.
3. The second n-Bit value to the right of the sign bit identifies the radix point of the number ("R"). The size in bits of this field is identical to the size of the previous n-Bit field, (L).
4. The third n-Bit value to the right of the sign bit represents the binary values ("B") of the number beginning with the least significant bit (LSB) and extending to the most significant bit (MSB).

A signed n-Bit floating-point number would be stored as follows:

S (1 bit)	L (n bits)	R (n bits)	B (L bits)

For example, any binary floating-point value from 1 to 16,777,216 significant digits in length could be stored with the following n-Bit data type:

S (1 bit)	L (24 bits)	R (24 bits)	B (L bits)

Binary addition, subtraction, multiplication, and division are accomplished with n-Bit data types using standard methods. Addition may be performed using binary adders. Subtraction may be performed by using "true complement" notation and adding the minuend to the complemented subtrahend. Multiplication is the result of repeated binary addition. Division is the result of repeated binary subtraction.

To add the following two n-Bit floating-point numbers:

```
            11.001      +  1.011011
            11.001      =
S = 0        L = 0101       R = 0010      B = 11001
             1.011011   =
S = 0        L = 0111       R = 0001      B = 1011011
            11.001
          +  1.011011
          ___________

(LSB)  100.100011   (MSB) =
S = 0        L = 1001       R = 0011      B = 100100011
```

Negative numbers may be stored in complemented form to facilitate the ease of substraction. By using "true complement" notation (i.e., reversing the value of each binary digit in the representation of the number), a binary adder may be used to accomplish binary addition, subtraction, multiplication, and division. This means that RNA may be implemented in microcode or at the level of hardware circuitry. At this level, RNA may be implemented as a microprocessor function or as a specific purpose hardware component of a general purpose computer.

FIG. **5** describes an example of an Arithmetic Logic Unit (ALU) which implements Rule-based n-Bit Arithmetic (RNA). One or more n-Bit values are retrieved from memory **100** by the command processor **110**. The command processor **110** interprets the values as data or instructions, depending on their locations in memory **100**, and submits them to the rule-base interface **120**.

If the instruction is a logic instruction, the data and the instruction are submitted to the logic rule-base **130**. Logic rules include, but are not limited to, AND, OR, NOT, NAND, and NOR logic functions. Once the appropriate rule has been applied to the data, the result is returned to the rule-base interface **120**. Additional logic and/or arithmetic rules may be applied to the data before it is returned to memory **100**.

If the instruction is an arithmetic instruction, the data and instruction are submitted by the rule-base interface **120** to the arithmetic rule-base **140**. Arithmetic rules include, but are not limited to ADDITION, SUBTRACTION, MULTIPLICATION, and DIVISION. Once the arithmetic rule **140** has been applied to the data, the result is returned to the rule-base interface **120**. Additional arithmetic and/or logic rules may be applied to the data before it is returned to memory **100**.

The advantages of RNA are:

1. It provides greater precision than any other arbitrary precision arithmetic method.
2. The size of the numbers used in RNA is limited only by the real or virtual address space of the computer.
3. RNA may be implemented in hardware of software.
4. RNA is processor independent.
5. RNA provides faster calculations of very large arbitrary precision numbers.

A specific purpose RULE-BASED n-BIT VIRTUAL SOFTWARE MACHINE, as uniquely described by this invention, is any specific purpose virtual software machine which uses a rule-base as an instruction set to perform binary string operations on n-bit data types.

The COMMAND PROCESSOR is a machine that uses a program which receives n-bit data types and command language instructions as input and performs operations upon the input using one or more rules. Each rule is a type of processor instruction which performs a binary string operation upon one or more n-bit data types. After the input data has been processed, the command processor outputs data in the form of one or more n-bit data types.

An n-bit data type is defined as a data type consisting of n bits (or binary digits) where n is any number greater than zero. There is no inherent upper limit on n-bit data types. Variable length n-bit data types are used as standard input and output and are maintained and managed by the invention.

The RULE-BASE INTERFACE is defined as a method of transferring data between the command processor and the rule-base. The data in the form of one or more n-bit data types is passed to the rule-base interface by the command processor. The rule-base interface, in turn, identifies the rule or rules that are to be used in processing the data. The data is dispatched as one or more arguments to the selected rule within the rule-base and the rule is applied. After the data has been modified in accordance with the specified rule or rules, the modified data is returned to the rule-base interface. The rule-base interface may iteratively submit the data to one or more rules. Once the conditions for the modification of the data by the rule-base have been satisfied, the data is returned as one or more n-bit data types to the command processor. The command processor then outputs the data.

The RULE-BASE INTERFACE manages access to the rules within the rule-base using any access method including, but not limited to, linked lists, tree structures, relational database tables, and hyper-link stacks.

The RULE-BASE is a collection or set of rules. Each rule applies a binary string operation to the input data. A BINARY STRING OPERATION is any operation which performs bit level operations on one or more binary strings representing n-bit data types. A binary string operation may emulate processor instructions such as binary ANDs, ORs, XORs, and COMPLEMENT operations. Combinations of these operations may emulate processor instruction sets with the additional advantage of providing virtual n-bit data and instruction registers within the virtual machine in which to perform these operations. Binary string operations may also emulate more complex operations such as addition, subtraction, multiplication, division, vector, and matrix operations. These operations are implemented in such a way that there is no inherent upper limit on the length of the n-bit data types used as input or output.

The types of data structures which may be used to implement the rule-base as it is defined in the invention include, but are not limited to, the following: relational database tables, C or C++ language header files, any generation computer language function(s) or subroutine(s), object class libraries, and EPROM assembly language subroutines, and microcode instruction sets.

Although the invention and several of its preferred embodiments have been described and illustrated in detail, the same is by way of example only and should not be taken by way of limitation. The spirit and scope of the present invention are limited only to the terms of the appended claims.

What is claimed is:

1. A method for compressing information data from a data source comprising the steps of:

coupling at least one n-bit data string of input data as variable length n-bit data types where n is all integers greater than 0 and includes both odd and even numbers and is limited only by the physical address space of the computer and contains bits representing said information data and including control bits to a virtual command processor;

storing a plurality of data compression rules in a rule-base memory for processing the n-bit data string;

coupling a rule-base interface between said virtual command processor and said rule-base memory for identifying specific data compression rules stored in said rule-base memory according to said control bits in said n-bit input data string received from said virtual command processor;

modifying the n-bit data string according to the identified compression rules in the rule-base to compress the information data bits; and

transferring said compressed information data bits to said virtual command processor for output as variable length n-bit data types.

2. A method as in claim 1 further comprising the steps of:

identifying said data source with said control bits in said n-bit data string; and

including bits in said control bits that represent at least one argument to be used when accessing said rule-base.

3. A method as in claim 2 further comprising the steps of:

coupling one or more of said arguments to said identified rule within said rule-base; and

applying the identified rule to the n-bit information data to modify said information data and to perform said data compression.

4. A method as in claim 3 further including the steps of:

appending additional arguments, as needed, to said modified information according to said identified rule in said rule-base; and

returning the modified information data to said rule-base interface along with said needed arguments.

5. A method as in claim 4 further including the steps of:

iteratively submitting said modified data to at least another one of said rules stored in said rule-base in accordance with said arguments appended by said identified rule for further modification until said data modification satisfies all of said arguments; and

returning said satisfied modification data to said virtual command processor as one or more n-bit data strings that are not required to correspond to the n-bit size and number of n-bit input data strings coupled to said virtual command processor from said data source.

6. A method as in claim 1 further comprising the steps of:

storing said rules in a memory in said rule-base; and

defining said rules as binary string operations.

7. A method for creating a specific purpose virtual processor comprising the steps of:

coupling at least one n-bit data string of input data as variable length n-bit data types where n is all integers greater than 0 and includes both odd and even numbers and is limited only by the physical address space of the processor and contains bits representing said specific purpose and including control bits to a virtual command processor;

storing a plurality of rules for said specific purpose in a rule-base memory for processing the n-bit data string;

coupling a rule-base interface between said virtual command processor and said rule-base memory for identifying specific ones of said specific purpose rules stored in said rule-base memory according to said control bits in said n-bit input data string received from said virtual command processor;

modifying the n-bit data string according to the identified specific purpose rules in the rule-base to create a specific purpose application; and

transferring said modified n-bit data string representing said specific purpose application to said virtual command processor for output as variable length n-bit data types.

8. A method as in claim 7 further comprising the steps of:

identifying said data source with said control bits in said n-bit data string; and

including bits in said control bits that represent at least one argument to be used when accessing said rule-base.

9. A method as in claim 8 further comprising the steps of:

coupling one or more of said arguments to said identified rule within said rule-base; and

applying the identified rule to the n-bit information data to modify said information data to perform said specific purpose.

10. A method as in claim 9 further including the steps of:

appending additional arguments, as needed, to said modified information according to said identified rule in said rule-base; and

returning the modified information data to said rule-base interface along with said needed arguments.

11. A method as in claim 10 further including the steps of:

iteratively submitting said modified data to at least another one of said rules stored in said rule-base in accordance with said arguments appended by said identified rule for further modification until said data modification satisfies all of said arguments; and

returning said satisfied modification data to said command processor as one or more n-bit data strings that are not required to correspond to the n-bit size and number of n-bit input data strings coupled to said virtual command processor from said data source.

12. A method as in claim 10 further including the step of manipulating said information data with rules from said rule-base such that accomplishment of said specific purpose is not dependent upon an explicit key, a specific purpose rule, or a specified data type.

13. A method as in claim 10 further comprising the step of user-defining the implementation of said specific purpose since no single algorithm or rule must be defined.

14. A method as in claim 7 further comprising the steps of:

storing said rules in a memory in said rule-base; and

defining said stored rules as binary string operations.

15. A virtual software processor for defining an n-bit data type in terms of a desired output where the value of n is all integers greater than 0 and includes both odd and even numbers and is limited only by the physical address space of the processor, said machine comprising:

an input means for receiving, as input, one or more of said n-bit data types as Y variable length n-bit words where Y>0;

a storage means for storing user-defined rules in a rule-base,

processing means coupled to said input means and said storage means for performing operations on said input n-bit data types using one or more of the user-defined rules of said rule-base to define said input n-bit data types in terms of said desired output; and

output means coupled to said processing means for outputting sad desired output as variable length n-bit words that do not necessarily hag to correspond with the size, in bits, and number, Y, of the variable length n-bit input words.

16. A method of implementing a virtual processor capable of performing rule-based n-bit arbitrary precision arithmetic logic functions comprising the steps of:

coupling at least one n-bit data string of input data as variable length n-bit data types where n is all integers greater than 0 and includes both odd and even numbers and is limited only by the physical address space of the processor and contains bits representing a desired arithmetic logic function and including control bits to a virtual command processor;

storing a plurality of arithmetic operations in a rule-base memory for processing the n-bit data string;

coupling a rule-base interface between said virtual command processor and said rule-base memory for identifying specific arithmetic logic functions stored in said rule-base memory according to said control bits in said n-bit input data string received from said virtual command processor;

modifying the n-bit data string in accordance with the identified arithmetic operations in the rule-base to perform said desired arbitrary precision arithmetic operations; and

transferring said performed arithmetic operations to said virtual command processor for output as variable length n-bit data types.

* * * * *

US005600726A

United States Patent [19]

Morgan et al.

[11] **Patent Number: 5,600,726**

[45] **Date of Patent: Feb. 4, 1997**

[54] **METHOD FOR CREATING SPECIFIC PURPOSE RULE-BASED N-BIT VIRTUAL MACHINES**

[75] Inventors: **Joseph M. Morgan**, Amarillo, Tex.; **Michael D. Harold**, Shreveport, La.

[73] Assignee: **Gemini Systems, L.L.C.**, Shreveport, La.

[21] Appl. No.: **419,001**

[22] Filed: **Apr. 7, 1995**

[51] **Int. Cl.^{6}** **H04L 9/00**

[52] **U.S. Cl.** **380/49**; 380/4; 380/25

[58] **Field of Search** 380/4, 23–25, 380/28–30, 49, 18

[56] **References Cited**

U.S. PATENT DOCUMENTS

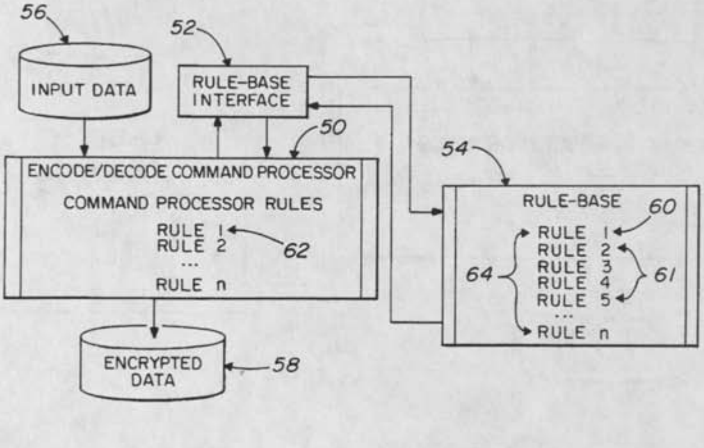
4,386,416	5/1983	Giltner et al. .	
4,454,575	6/1984	Bushaw et al. .	
4,697,243	9/1987	Moore et al. .	
4,788,543	11/1988	Rubin .	
4,890,240	12/1989	Loeb et al. .	
4,893,339	1/1990	Bright et al.	380/28
4,961,133	10/1990	Talati et al. .	
4,961,225	10/1990	Hisano	380/28
5,009,833	4/1991	Takeuchi et al. .	
5,031,215	7/1991	Pastor	380/30
5,038,296	8/1991	Sano .	
5,084,813	1/1992	Ono .	
5,097,504	3/1992	Camion et al.	380/30
5,101,491	3/1992	Katzeff .	
5,121,496	6/1992	Harper .	
5,131,087	7/1992	Warr .	
5,150,410	9/1992	Bertrand .	
5,153,918	10/1992	Tuai	380/25
5,212,768	5/1993	Itsuki et al. .	
5,228,116	7/1993	Harris et al. .	
5,255,386	10/1993	Prager .	
5,276,855	1/1994	Kitahara .	
5,278,901	1/1994	Shieh et al.	380/25
5,285,497	2/1994	Thatcher, Jr.	380/28
5,305,384	4/1994	Ashby et al.	380/28
5,315,655	5/1994	Chaplin	380/25
5,321,606	6/1994	Kuruma et al. .	
5,321,749	6/1994	Virga	380/18
5,351,299	9/1994	Matsuzaki et al.	380/29
5,384,846	1/1995	Berson et al.	380/23

Primary Examiner—Salvatore Cangialosi
Attorney, Agent, or Firm—Jones, Day, Reavis & Pogue

[57] **ABSTRACT**

A system and method for implementing one or more specific purpose rule-based n-bit virtual processing machines. Specific purposes include, but are not limited to, encryption, compression, and arbitrary precision arithmetic. Each virtual machine consists of a command processor, a rule-base, and an interface between the command processor and the rule-base. Each of the elements of a specific purpose rule-based n-bit virtual machine—the command processor, the rule-base, and the rule-base interface—is preferably implemented as software. In the preferred embodiment, the system uses a stored rule-base as its instruction set and provides for input and output in the form of variable length bit strings of length n where n is any number greater than zero. Each of the rules within the rule-base performs one or more binary string operations against one or more variable length n-bit strings. The function of the rule-base is to provide a set of application specific rules that allows the machine to perform a particular task such as encryption, data compression, or arbitrary precision arithmetic. The system includes a method for providing a software interface to the rule-base. This interface may be a separate program or may be contained within the command processor. The command processor receives input in the form of one or more n-bit data types, performs rule-based operations on the data, and returns output in the form of one or more n-bit data types. Specific system and methods for performing data encryption, data compression, and arbitrary precision arithmetic using the invention are described.

12 Claims, 2 Drawing Sheets



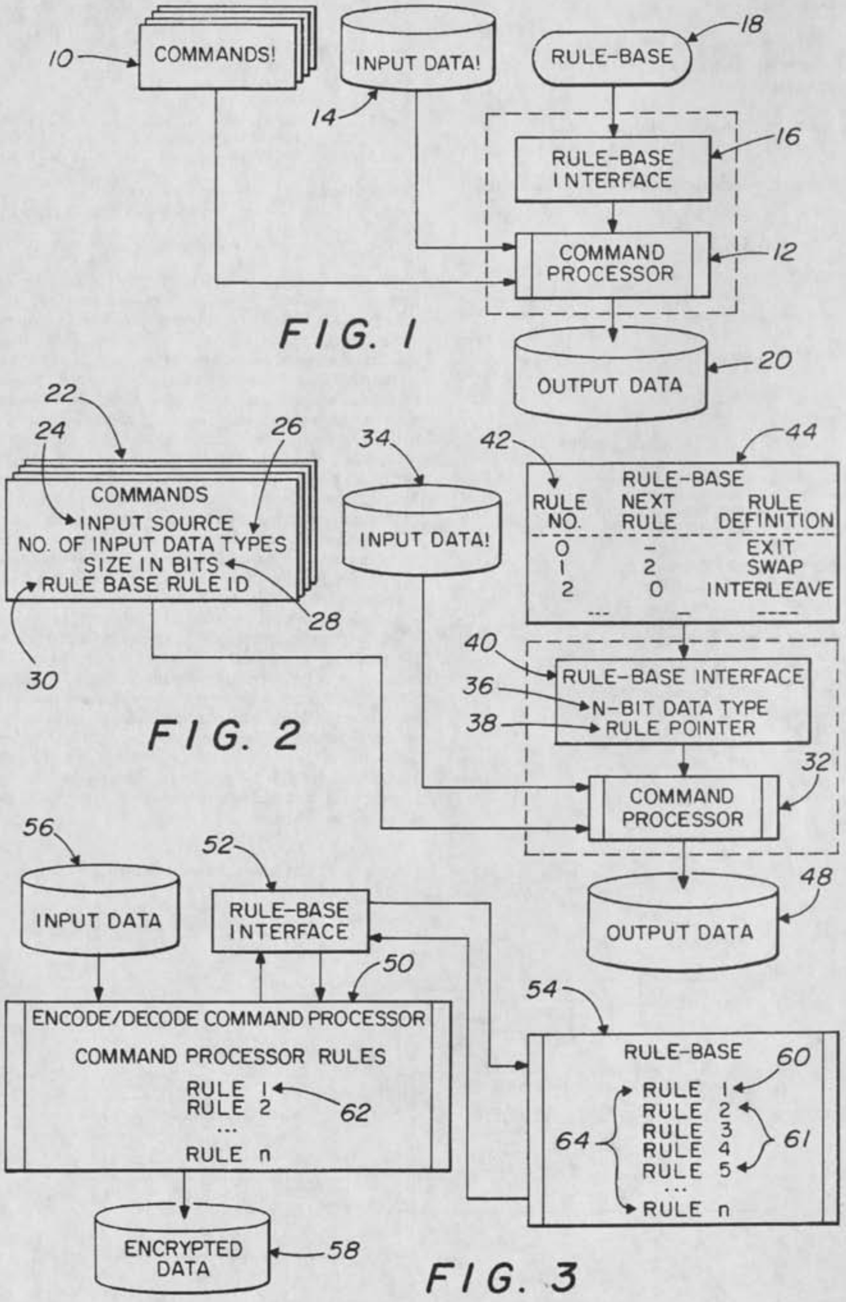
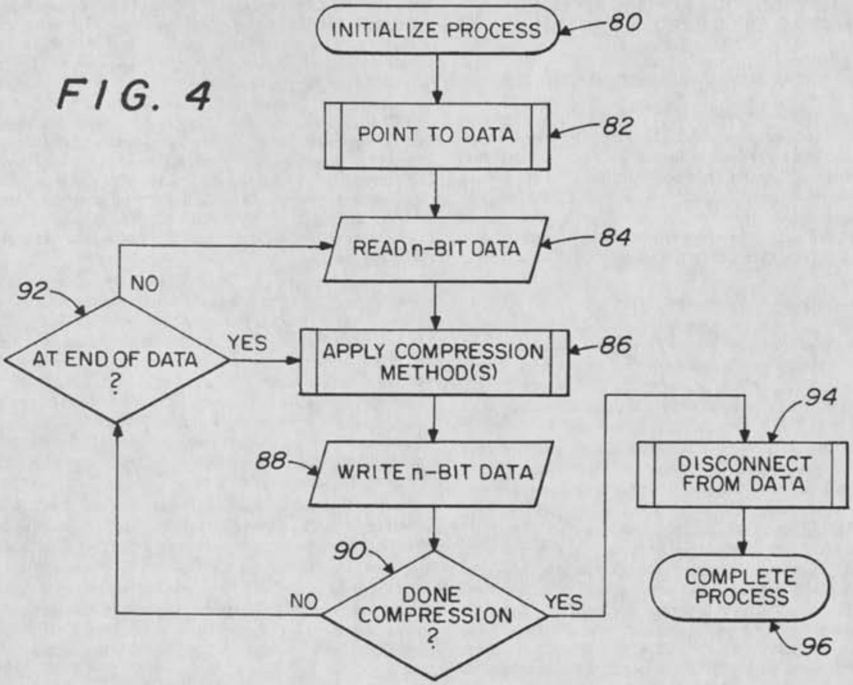


FIG. 1

FIG. 2

FIG. 3

FIG. 4


INITIALIZE PROCESS
80
POINT TO DATA
82
READ n-BIT DATA
84
NO
92
AT END OF DATA ?
YES
APPLY COMPRESSION METHOD(S)
86
94
88
WRITE n-BIT DATA
DISCONNECT FROM DATA
90
NO
DONE COMPRESSION ?
YES
COMPLETE PROCESS
96

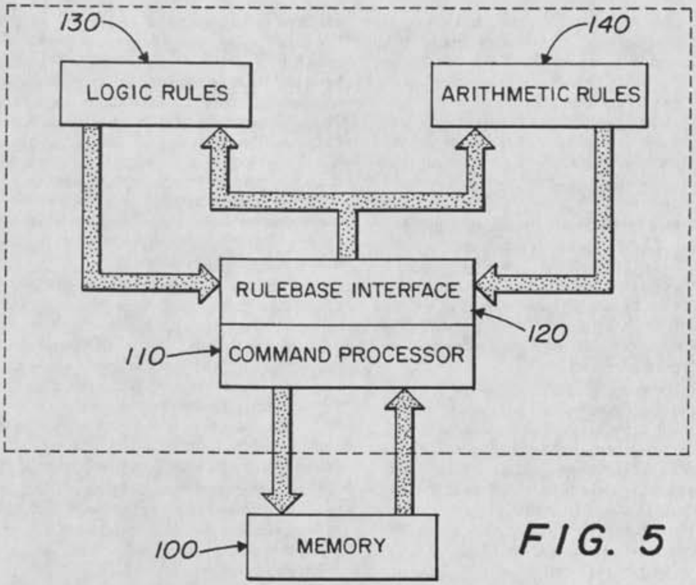

130
140
LOGIC RULES
ARITHMETIC RULES
RULEBASE INTERFACE
120
110
COMMAND PROCESSOR
100
MEMORY

FIG. 5

METHOD FOR CREATING SPECIFIC PURPOSE RULE-BASED N-BIT VIRTUAL MACHINES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to computer systems and more particularly to a software architecture for implementing specific purpose rule-based n-bit virtual machines to accomplish such tasks as data typing, encryption, compression, arbitrary precision arithmetic, pattern recognition, data conversion, artificial intelligence, device drivers, data storage and retrieval and digital communications.

2. Description of the Related Art

Existing systems designed to process data vary widely in their specific implementations. However, few are designed for the utilization of a rule-base and there are no others known that use, as their primary data type, an arbitrary X number of bits as input, and an arbitrary Y number of bits as output, where X may or may not be equal to Y.

With respect to virtual software machines, of specific mention is U.S. Pat. No. 4,961,133 filed Oct. 2, 1990, wherein Talati et al. disclose a "Virtual Execution Environment on a Target Computer Using a Virtual Software Machine". This invention deals with preprocessing and compiling source program code in such a way as to be operating system independent and to enable the code to execute across heterogeneous computers via a virtual interface system. Though the invention disclosed by Talati et al. involves providing a virtual software machine, it does not address the problem of directly manipulating machine instructions of any given n-bit length via a rule-base to machine instructions of any target n-bit length on a target machine.

With respect to data encryption, most systems apply some form of mathematical operation or bit-wise operation, such as exclusive-or (or XOR) against the input data to be processed based upon an encryption key or password. Normally, the encryption process is highly specialized, encrypting the data in the same theoretical manner from the beginning to the end of the data stream. These methods lend themselves to differential crypto-analysis, a method capable, through analytical means, of deciphering the encrypted message.

Of specific mention is Matasuzaki et al. U.S. Pat. No. 5,351,299 filed Sep. 27, 1994, whose encryption process is very difficult or impractical to break with more standard analytical methods. This method utilizes the standard idea of XORing data together by use of manipulation of a user-provided password. To decrypt, one XORs the encrypted data again, in reverse order, with the same manipulation of the same user-provided password.

Though Matasuzaki et al. break data up into N blocks of M-bit data, the specified Embodiment I states that "each bit outputted from hash function unit is dependent on all the bits inputted thereto." It also states in the embodiments that the primary input blocks are blocks of multiples of 8 bits, and further broken down into blocks of M bits, defined in 8 bits or multiples of 8 bits. This method severely limits introduction of arbitrary block encryption rules and does not allow for a prime number of bits, such as **11** or **13**.

The U.S. Pat. No. 5,285,497 to Thatcher, Jr. filed Feb. 8, 1994, specifies encoding variable length Huffman encoded bits in a unique way. However, it does not address the bits

as a data type arbitrarily, but in a form having a meaning directed by the Huffman compression means. The invention also requires the use of a specialized microprocessor, a fixed number of specialized encryption rules, and is specific to compressed, digital data streams.

Another unique encryption means as stated in U.S. Pat. 5,097,504 to Camion et al. identifies a signature based encryption means where the signature is recorded with the encrypted message and the encryption keys are stored on another, preferably inviolable, medium. This system applies a highly mathematical and specific encryption means, introducing, again, the problem and limitation of not having a flexible and rather arbitrary rule-base that is easily changeable and modifiable.

In U.S. Pat. No. 5,321,749, Virga presents an extremely unique encryption means that converts the input data into a bitmap and encrypts the bitmap to be targeted for decryption in an optical scanning device. The embodiment specifies XORing randomly generated bits produced from a user-specified password with the encoded bitmap. The bitmap is then converted to specific visual alphabet that can be easily recognized by a receiving scanning device. This method, however, allows an analytical hashing means to decipher the seed(s) generated from the user-specified password with a relatively small amount of time.

With respect to compression, there are many means of compression, all of them having the primary objective of locating the most common occurring data types and encoding them, on average, with a data type of a smaller size.

As an example, suppose the input data is comprised of the characters "ABCAB". A compression means may locate the most commonly occurring character pair, "AB", and encodes them with a single character "Z", thereby reducing the input data to ZCZ.

Though the above is an extremely simple example, the many compression means in existence today vary widely and have many implementations in hardware and software. However varying these compression means may be, a primary limitation exists for all of them. The limitation is that when compression has been achieved by use of the desired compression means, the data can no longer be compressed. This is due to the fact that the compressed output of the data results in a distribution of the input data type such that there is no longer a character or set of characters that occurs more frequently than another character or set of characters. Therefore, further compression is not possible or practical and some compression means will actually explode the size of the input data if the distribution of the characters of the input data type is relatively constant.

With respect to arbitrary precision arithmetic, many algorithms have been written to overcome the limitations of a computer to provide very high levels of precision in mathematical calculations. Though these methods can and do provide any desired precision with mathematical calculations, the calculations are performed algorithmically with the requirement to overcome the internal 8, 16, 32, or 64-bit limitations of the computer's hardware and internal memory mapping. These algorithms require a very high CPU load, demanding much of the computer's internal resources.

With respect to pattern recognition and data conversion, the invention disclosed herein provides enhancement to existing means of the same, introducing arbitrary data typing and a user-defined rule-base, the combination of which is absent in current systems.

In U.S. Pat. No. 5,321,606 filed Jun. 14, 1994, Kuruma et al. describe a user-defined set of transformation rules that define the nature of the grammar of the input data to be converted. The invention solves the problem of writing a specific parser or compiler where the limitations rely upon a specific grammar existent in the input data and a specific output term in the output data. Yet, this invention specifies that the output involves "structures of output terms in association with terminal symbols and nonterminal symbols".

In U.S. Pat. No. 4,890,240 filed Dec. 26, 1989, Loeb et al. describe a rule-based, artificial intelligence system where the rules are specifically defined in two parts, a left-hand side and a right-hand side; whereas, the left-hand side is considered an "if" statement and the right-hand side is considered a "then" statement. This invention is specific to overcoming prior problems in RETE processing and not to arbitrary pattern matching and identification with an externally provided rule-base.

U.S. Pat. Nos. 5,038,296 filed Aug. 6, 1991, 5,084,813 filed Jan. 28, 1992, and 5,101,491 filed Mar. 31, 1992 all refer to rule-based systems for generating program code. Though one of the objectives of the present invention is data transformation of program code from one n-bit machine instruction via an externally provided rule-base to a different n-bit machine instruction, it is not directed at code generation and the invention disclosed herein is not limited as such.

SUMMARY OF THE INVENTION

The present rule-based n-bit virtual machine, or processor, may be implemented in software and/or hardware. When implemented as software, a rule-based n-bit virtual machine converts a general purpose computer into a machine that performs an application specific function.

Further, a virtual processor may execute its instructions either in batch mode or interactively.

It is, therefore, a primary object of this invention to provide a means by which one or more of a data type of n-bit size is selected or received as input, processed by a rule or rules designed for processing one or more of a data type of n-bit size, and outputting or transmitting one or more of the processed data type of an n-bit size. In all data-type cases, the value of N is any number greater than zero. The size, in bits, and number of the input data type do not necessarily have to correspond with the size, in bits, and number of the output data type. Also, any given rule designed to process n-bit data types may or may not be specifically designed to work on an n-bit data type of a particular size in bits.

It is yet another object of the present invention to provide an architecture for creating a specific purpose virtual machine using software that manipulates n-bit data types and rule-based instruction sets.

It is another object to create a command processor controlled by a program and that accepts input in the form of one or more n-bit data types and outputs data in the form of one or more n-bit data types where n is any number greater than zero. The upper value of n is limited only by the physical or virtual address space of the computer.

It is still another object to create an interface program between the command processor and the rule-base called the rule-base interface. Separating the command processor from the rule-base allows the rule-base to be stored in different forms such as, but not limited to, a relational database table, a C or C++ language header file, an object class library, a dynamic link library, an EPROM assembly language subroutine, or a microcode instruction set.

It is a further object of the invention to provide a new method for creating applications relating to various fields within computing including, but not limited to, data typing, data encryption, data compression, arbitrary precision arithmetic, pattern recognition, data conversion, artificial intelligence, data storage and retrieval, and digital communications.

It is yet a further object of the present invention to demonstrate the advantages of the method by describing in detail specific implementations of the invention related to data encryption, data compression, and arbitrary precision arithmetic.

In accordance with one aspect of the invention, a subsidiary object is to provide a new method and system of data encryption. This new method of encryption will employ a command processor and a rule-base and will input and output data as variable length n-bit data types.

In accordance with one aspect of the invention, a further subsidiary object is to provide a new system of data compression. This concept involves, but is not limited to, the implicit redistribution of n-bit data-type frequencies by the explicit compression of data using n-bit data types as input and output. This concept allows the data to be compressed reiteratively.

In accordance with another aspect of the invention, a further subsidiary object is to provide a .new system of arbitrary precision arithmetic. This new system of arithmetic will employ a command processor and a rule-base and will input and output data as variable length n-bit words.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the present invention will be more fully disclosed when taken in conjunction with the following DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS in which like numerals represent like elements and in which:

FIG. **1** is a diagram illustrating the organization of a system identifying the principal elements and processes associated with the present invention;

FIG. **2** is a block diagram further illustrating the interrelation of the elements of the invention;

FIG. **3** is a diagram illustrating an implementation of the invention as a data encryption system;

FIG. **4** is a process flow chart illustrating an implementation of the invention as a loss-less data compression method; and

FIG. **5** is a diagram illustrating an implementation of the invention as an arbitrary precision arithmetic method.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

With reference now to the drawings, FIG. **1** shows the organization of the principal elements used in the novel processes of a system and method for implementing a specific purpose rule-based n-bit virtual software driven data processing machine. A program used by the command processor **12** receives, as input, one or more commands identifying the input data source **14** and the instructions and/or arguments **10** which will be used in accessing the rule-base **18**. Once a command is received by the command processor **12**, data is input to the command processor in the form of one or more n-bit streams from the data source **14**. The data is passed to the rule-base interface **16** by the command processor **12**. The rule-base interface **16** in turn uses the data to identify and select the rule or rules that are to be used in processing the data. The data is dispatched as one or more arguments to the selected rule within the rule-base **18** and the rule is applied. After the data has been modified in accordance with the specified rule or rules, the modified data is returned to the rule-base interface **16** along with any arguments appended by the last rule applied. These arguments may be used by the rule-base interface **16** to determine the next rule or rules to be applied to the data. These arguments may also consist of key words or messages identifying the current state of the data conversion process. The rule-base interface **16** may iteratively submit the data to one or more rules within the rule-base **18** based on the value or values, if any, of the arguments returned by the previous rule or rules. Once the conditions for the modification of the data by the rule-base have been satisfied, the data is returned as one or more n-bit streams to the command processor **12**. The command processor **12** then outputs the data as one or more n-bit streams **20**. The size in bits and number of n-bit streams output by the command processor **12** is not required to correspond to the size in bits and number of n-bit streams which were originally input.

A simple example of the process is shown in FIG. **2**. Command line input **22** generates data that identifies the source **24** of the input data, the number of input data types **26**, the size of the input data type **28** and the rule-based rule ID **30** that is to be applied to the data. The command line data is processed by the command processor **32** after which the input from data source **34** is read and a rule pointer **38** is generated from the rule ID **30**. The n-bit input data **36** and the rule pointer **38** are passed to the rule-base interface **40**. The rule-base interface **40** in turn uses the rule pointer **38** to identify the rule **42** to be applied to the data and passes the data to the appropriate identified rule **42** within the rule-base **44**. The data is modified by the rule-base **44** and the modified data is returned to the rule-base interface **40**. The rule-base interface **40** in turn passes the data to the command processor **32** which outputs the data as one or more n-bit data types **48**. The objects of the invention are achieved by the novel application of the rule-base and the use of n-bit data types for input, processing, and output functions. This method, when applied to specific applications, may result in major improvements in the performance and capabilities of existing software application driven processing machines. Furthermore, the use of variable length n-bit data types provides numerous opportunities to create new, specific purpose virtual computing environments which are capable of performing tasks that are not possible using eight bit technologies.

For example, and not by way of limitation, the following description illustrates one method of implementing a virtual machine or computer capable of performing rule-based n-bit encryption:

Rule-based n-Bit Encryption (RNE) encrypts data as a string of binary digits using a command processor, a rule-base interface, and a rule-base. In order to fully realize the benefits of RNE, it is important to realize that all of the data elements of the method, including the data processed by the command processor, the rule-base, and the data itself, are perceived as one or more strings of binary data.

In RNE the bit is the primary data structure. Any or all of the elements of the RNE virtual machine may be input, processed, and output as data. The binary representation of the elements of the RNE virtual machine or processor is a bit stream composed of one or more n-bit data types and is not organized as bytes except where the physical and/or system limitations of the computer require it.

As described in FIG. **3**, RNE consists of four principal elements: an encode/decode command processor **50**, a rule-base interface program **52**, a rule-base **54**, the input data or message **56**, and the encrypted data or message **58**.

The rule-base **54** is composed of one or more rules **60**. Each rule **60** may contain variable data and literal data. Each rule **60** may, but is not required to, receive one or more arguments as input. Each rule may, but is not required to, output one or more arguments. The encrypted message is the result of the RNE process. The command processor **50** rules and/or the rules of the rule-base **54** may or may not be contained in the encrypted message **58** at the time transmission occurs.

The command processor **50** is used to access the rule-base **54** and to manage the actual encryption/decryption process. One or more rules **62** may be contained within the command processor **50** to uniquely identify the command processor **50** and/or provide an index or offset into the rule-base **54**.

For example, one of the rules **62** contained in the command processor **50** might implement a hashing rule that would use input data **56** to select an encryption key to generate a pointer into the rule-base **54**. This would allow any number of public and private encryption keys to implement unique encode/decode rule sets within a single rule-base **54**. This would also allow encryption keys to be implicitly user-defined within the command processor **50**. Only matched command processors **50** could decode each other's messages. Neither encryption nor decryption is dependent upon an explicit encryption key, a specific encode/decode rule, or a specified data type. However, any encryption key may be defined either implicitly or explicitly.

The command processor **50** may also be used to parse a password or access code in the input data and to pass the resulting values as arguments to one or more rules **60** or rule sets **61** within the rule-base **54**.

As an example, an encryption key might be constructed having 3 bytes or 24 bits. Each of the bytes would represent a rule set. Each bit within a byte would represent the application of a specific rule or rule set within a rule set. The bytes 10010101, 11100011 and 00101010 (identified as rule sets 1, 2, and 3, respectively) might be used by the first element of RNE, the command processor **50**, to apply the following rules or rule sets:

For rule set number 1, rules or rule sets 1, 4, 6, and 8 would apply.

For rule set number 2, rules or rule sets 1, 2, 3, 7, and 8 would apply.

For rule set number 3, rules or rule sets 3, 5, and 7 would apply.

The total number of combinations for any given implementation using a key having 24 bits is 16,777,216. Because RNE is based on bit string manipulation, there is no upper limit on the length of the encryption key.

The second element of RNE, the rule-base **54**, is a set of rules **64** used by the command processor **50** to decode or encode binary strings of data. The rules **64** may be used individually, or as a set to decode or encode data.

The rule-base **54** is not defined as a specific type of data structure. The rule-base **54** may, for example, be stored as a secure table in a relational database, as a C or C++ language header file, as an object class library, as a dynamic link library, or as an EPROM assembly language subroutine, or as a microcode component within a microprocessor.

In addition, access to the rules **64** within a rule-base **54** may be accomplished by the command processor **50** using one or more data structures including, but not limited to, linked lists, tree structures, relational tables, object class libraries, hash tables, or hyper-link stacks.

Following are examples of n-bit binary string operations which may be used to encrypt the input data. Consider, first, examples of vector rules and their explanation.

Examples of Vector Rules

The following rules provide simple examples of the ways bit strings may **15** be manipulated. The number and combination of possible rules is infinite.

Inversion			
Invert the following n bits:			
Where n = 7	1011100	becomes	0100011
Transposition			
Transpose the following n pairs of bits:			
Where n = 3	10 11 10	becomes	01 11 01
Interleaving			
Interleave with a ratio 1:1 the following pair of n bits:			
Where n = 4	1011 1001	becomes	1100 1011
Shift lift			
Shift the following n bits x bits to the left:			
Where n = 5, x = 1	11011	becomes	10111
Shift Right			
Shift the following n bits x bits to the right:			
Where n = 5, x = 1	11011	becomes	11101

Consider, next, examples of matrix or two-dimensional rules and an explanation of their use.

Examples of Matrix Rules

For each of the following examples, the bit stream 0110 0010 1110 0101 1100 will be the input stream.

Rule 1

Step 1. Enter bits from left to right, top to bottom, into a matrix with 5 rows and 4 columns.

0	1	1	0
0	0	1	0
1	1	1	0
0	1	0	1
1	1	0	0

Step 2. Rotate the matrix 90 degrees to the left resulting in a matrix with 4 rows and 5 columns.

0	0	0	1	0
1	1	1	0	0
1	0	1	1	1
0	0	1	0	1

Step 3. Write the bits from left to right, top to bottom

Result:	0001 0111 0010 1110 0101
Original:	0110 0010 1110 0101 1100

Note: This is equivalent to a reverse interleave of X, n-bit data types, where X=5, n=4. The larger the values of X and n, the larger the adjacency displacement.

Rule 2

Step 1. Enter the bits from left to right, top to bottom, into a matrix with 4 rows and 5 columns.

0	1	1	0	0
0	1	0	1	1
1	0	0	1	0
1	1	1	0	0

Step 2. Rotate the matrix 180 degrees.

0	0	1	1	1
0	1	0	0	1
1	1	0	1	0
0	0	1	1	0

Step 3. Invert the bits.

1	1	0	0	0
1	0	1	1	0
0	0	1	0	1
1	1	0	0	1

Step 4. Write the bytes from left to right, top to bottom.

Result:	1100 0101 1000 1011 1001
Original:	0110 0010 1110 0101 1100

Rule 3

It is possible to use multiple arrays to encrypt bit streams and to combine vector rules with array operations. The following rule uses three matrices. The first matrix (a) is 3 rows by 4 columns. The second matrix (b) is 2 rows by 3 columns. The third matrix (c) is 1 row by 2 columns.

Step 1. Read the first 12 bits of the input stream into 3 4-bit data types.
0110 0010 1110

Step 2. Treating each data type individually, shift each bit 1 bit to the left.
1100 0100 1101

Step 3. Fill matrix (a) entering each data type into each of the three rows.

1	1	0	0
0	1	0	0
1	1	0	1

(a)

Step 4. Enter the remaining bits from left to right, top to bottom, into matrices (b) and (c).

0	1	0
1	1	1

(b)

0	0

(c)

Step 5. Swap matrices (b) and (c).

$$\begin{array}{|c|c|}\hline 0 & 0 \\ \hline\end{array} \qquad (c)$$

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 1 & 1 \\ \hline\end{array} \qquad (b)$$

Step 6. Rotate all three matrices 90 degrees to the left.

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline 1 & 0 & 1 \\ \hline\end{array} \qquad (a)$$

$$\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline\end{array} \qquad (c)$$

$$\begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 1 \\ \hline 0 & 1 \\ \hline\end{array} \qquad (b)$$

Step 7. Write the bits from left to right, top to bottom.

Result:	0010 0001 1110 1110 1001
Original:	0110 0010 1110 0101 1100

Additional rules may be created for any n-dimensional data representation. There is no upper limit on the number of possible rules or the manner or order in which they are implemented.

The use of public and/or private specific purpose keys such as encryption keys is optional. The encryption and decryption of data is not dependent on an explicit encryption key, a specific encode/decode rule or a specific data type. An uncountable number of encryption keys may be applied against a single implementation of RNE each one of which enforces a unique rule-set, each set containing unique encode/decode rules, in a unique order. Each encryption key may be explicitly or implicitly defined. Thus by providing multiple encryption keys to a single encrypted message, parts of the message will be available only to some users and not others when the message is distributed across multiple machines for multiple users.

The successful encryption or decryption of the data is not dependent on the size of the encryption key or the speed of capacity of the processor. Without access to the encryption key, the command processor rules, and the rule-base, it is impossible to establish a correspondence between the original data and the encrypted data.

For example, and not by way of limitation, the following representative program listing in C details a second example implementation of Rule-based n-Bit Encryption (RNE).

rule-base, it is impossible to establish a correspondence between the original data and the encrypted data.

For example, and not by way of limitation, the following representative program listing in C details a second example implementation of Rule-based n-Bit Encryption (RNE).

```
/*________________________________________________________
BME.C          Binary Matrix Encryption
               An Example Implementation of Rule-Based n-Bit Encryption.

Program Development Log:

Release of Version 1.10
_______________________________________________________*/

#include <stdio.h>
#include <io.h>
#include <malloc.h>
#include <conio.h>
#include <math.h>
#include <graph.h>
#include <string.h>
#include <mstring.h>
#include <stdlib.h>
#include <mstdlib.h>

#define YES         1
#define NO          0

#define IN          'I'
#define OUT         'O'
#define OFF         0
#define ON          1

#define FULL        0
#define NOTFULL     1
```

```
/*____________

Matrix Definition
____________*/

typedef struct matrixTAG { unsigned int x, y;
                                              unsigned long size;
                                              char *matrix; } MATRIX;

/*____________

Function Definitions
____________*/

char AddByteToMatrix(unsigned char);
void AllocateBuffers(void);
void AllocateLargestMatrix(void);
void CleanUp(char *);
void ClearMatrix(void);
void ClearBitCan(void)
void DefineMatrix(void);
void DisplayProgress(void);
void DisplayTitle(void);
int FillMatrix(void);
void FlushOutBuffer(void);
unsigned char GetBitStrVal(char *);
int GetNextByte(void);
void GetOptimalScale(void);
void InvertMatrix(void);
void NoArguments(void);
void OpenInFile(char *);
void OpenOutFile(void);
void PutNextByte(int);
void SendBitAtXY(long, long, unsigned int);
void SetEncryptionMethod(char *);
void SetInversionFreq(char *);
void SetMode(char);
void SetNextMethod(void);
void SetPassWord(char *)
void SwapScale(void);
int TimeToInvert(void);
void WriteBit(char);
void WriteReverse(void);

/*______________________________________

Matrix Encryption Functions

The following matrix encryption functions are the most simplistic variations on arbitrarily approaching the matrix. Even so, these functions confuse the data so much that no one could, with any degree of certainty decipher even one of the rules contained here.

There are ways to make and process three-dimensional matrices, and virtually any 2D or 3D shape would work. There are no means possible to figure out the total number of rules that can be applied to this method, since, of course, another bit could always be added, the shape or size changed, or some other variations utilized like adding an arbitrary bit every X number of bits. Thus, one would not know from where all the bits came, or to where they should go.

The number of rules is really the limit of the imagination's ability to create them.
______________________________________*/

#define NUM_RULES 7                   /*There are actually 8 rules*/
                                      /*butTopRightDown           */
                                      /*doesn't do anything.      */
void BottomLeftUp(void);
void BottomRightUp(void);
void BottomUpLeft(void);
void BottomUpRight(void);
void TopDownLeft(void);
void TopDownRight(void);
void TopLeftDown(void);

/*______________

File Pointers and File Names
_______________*/

FILE *InFile, *OutFile;
char OutFileName[L_tmpnam];
```

```
/*________________

Global Variables
___________________*/

#define MAX_BUFFER 16384

unsigned char *InBuffer, *OutBuffer;
int OutByteCount = 0;

MATRIX Matrix;
unsigned long MatrixCount = 0L;

char *PassWord;
unsigned PassWordLen = 0

char BitCan[9];
unsigned char LeftOver[9];


/*______________________________________________
```

The following data type is an array of pointers to the various encryption rules. It acts as the rule-base interface. Since the rules are set up as an array of function pointers, not only can rules be added or subtracted with great ease, they can be reordered. In other words, this particular example program can be compiled as shown to produce an encryption engine. Then the rule order can be changed in the pointer array defined below. Then the program can be recompiled to create another encryption engine. The two resulting encryption engines would be theoretically identical, but the first would not be able to decode a data stream encoded by the second, and vice-versa. With enough rules, therefore, as many unique encryption engines could be created as desired. Each one of those unique encryption engines would have hundreds of billions of password-method implementations

Also, because the rules are set in an internal data-array, it would be easy to add rule-order rules, so that an operation like the checksum of the incoming data would trigger a rule that reorders the rule pointers on the fly. In other words, not only does the end-user have some control over the encryption, but the data itself would be a player in the total scheme of things. Thus, one who is uninformed would have to know beforehand the unencrypted version of the data before attempting to decrypt it.

```
______________________________________________*/

void (*Rule[NUM_RULES](void) = {BottomUpLeft, BottomUpRight,
                                BottomLeftUp, BottomRightUp,
                                TopLeftDown,  TopDownLeft,
                                TopDownRight };

int *EncryptionMethod;
unsigned RuleID = 0;
char Mode = IN

unsigned long FileSize = 0L;
unsigned long TotalBytesRead = 0L;


/*_________________

User Definable Switches
____________________*/

unsigned char Inversion = OFF;
unsigned long InversionFreq = 0L;


/*______________________________________________
```

Program Code Follows

BME mode filename password1 password2 {/i=x}

- mode is either I or O.
- filename is the file to encrypt.
- password1 defines the sizes of the matrices. This way, the matrices vary in size as the file is encrypted. Therefore if a hacker were to experience some miracle of guessing the correct matrix size, the correct method, and whether it had been inverted, the rest of the file would still be junk to him. Pass it twice, and he would never know that he got it right in the first place!
- password2 defines the way that the rules are used. Each letter of password 2 represents, rather arbitrarily (see later) a rule to use for

```
          encryption/decryption. That way, the rules to use for each matrix
          are randomly selected by the use of password2. This is very
          helpful. Because the command processor can rotate through
          password1 and password2, a given matrix size is most likely going
          to be encrypted different ways each time it is used.
     -    For this implementation and just to complicate matters, the /i=x
          switch was added to provide the requirement that for each
          frequency of X matrices, invert the Xth matrix. This way, the
          uninformed would not even know the original value of any given bit
          in the file! So a '1' is present. Was it originally '1' or '0'? There
          is no way to know without knowing who encrypted the file.

____________________________________________________________*/

void main(int argc, char *argv[])
{
	char lastbit = 0;

	DisplayTitle();

	if (argc <5)
		NoArguments();

	AllocateBuffers();

	SetMode(argv[1][0]);

	OpenInFile(argv[2]);
	OpenOutFile();

	SetPassWord(argv[3]);

	DefineMatrix();

	SetEncryptionMethod(argv[4]);

	switch (argc) {
		case 6:
			SetInversionFreq(argv[5]);
	}

	while (FillMatrix() !=EOF) {
        ++MatrixCount;

        DisplayProgress();

        if(TimeToInvert() == YES)
                InvertMatrix();

        (*Rule[RuleID])();

        DefineMatrix();

        SetNextMethod();

    }
    if(strlen(Matrix.matrix)) {
            if(strlen(Matrix.matrix) != Matrix.size) {

                Matrix.size = strlen(Matrix.matrix);
                GetOptimalScale();

                if (Matrix.size == 0) {

                    lastbit = *(Matrix.matrix + Matrix.size - 1);
                    --Matrix.size;

                    *(Matrix.matrix + Matrix.size) = '\0';

                    GetOptimalScale();
                }
            }

            (*Rule[RuleID])();

            if(lastbit)
                    WriteBit(lastbit);
    }

    CleanUp(argv[2]);
    exit(0);
}
```

```
/*____________________________________________________

These functions are involved in setup and shutdown.
___________________________________________________*/


/*____________________________________________________

SetMode - This is where from the command line, either a 'I' or a 'O' is used.
It really doesn't matter which is used as long as the opposite letter is used to
decrypt as was done to encrypt. In other words, if I is used to encrypt, O is
used for the decrypt, and vice-versa.
___________________________________________________*/

void SetMode(char mode)
{
        Mode = toupper((int)mode);

        if(Mode !=IN && Mode !=OUT) {
            printf("\nInvalid Mode: [%c]. Use IN or OUT\n",Mode);
            exit(1);
        }
}

/*____________________________________________________

Buffer the input and output.
___________________________________________________*/

void AllocateBuffers(void)
{
        InBuffer  =  (unsigned  char  *)malloc(MAX_BUFFER  *
        sizeof(unsigned char));
        OutBuffer  -  (unsigned  char  *)malloc(MAX_BUFFER  *
        sizeof(unsigned char));

        if (InBuffer = = NULL || OutBuffer = = NULL) {
                puts("\nNot Enough Memory\n");
                exit(1);
        }
```

```
}

/*______________________________________________________

Open the file to encrypt or decrypt.
____________________________________________________*/

void OpenInFile(char *fname)
{
        if((InFile = fopen(fname, "rb")) == (FILE *)NULL {
                printf("\nCan't Open %s for Input\n", fname);
                exit(1);
        }


        FileSize = filelength(fileno(InFile));
}


/*______________________________________________________

Open a file in which to write the encrypted/decrypted data.  A temporary
name is used here for example only.
____________________________________________________*/

void OpenOutFile(void)
{
     tmpnam(OutFileName);

     if((OutFile = fopen(OutFileName, "wb")) == (FILE *)NULL) {
             printf("\nCan't Open %s for Output\n", OutFileName);
                 exit(1);
     }
}


/*______________________________________________________

This is the user's first password argument derived from the fourth command
line argument.  If there is not an even number of characters, one character is
added so that an even number of characters is used for the matrices.  It does
```

not have to be done this way. It could be done any number of ways in this particular application. Here, the first password is used to define the matrix sizes. It is made to be an even number of characters so there is a Y for every X. If a new character is not added to bring the first password length to an even number of characters, that new character is simply assigned the same value as the first character of the password. That, too, is arbitrary and can actually be done any number of ways.

Here, the ASCII decimal value of the character is used as the matrix axis value. For example, suppose the password is 'ABCD'. The ASCII decimal values for A, B, C, and D are 65, 66, 67, and 68, respectively. The first matrix axes are defined using A and B. The size of the matrix, therefore, is 65 bits by 66 bits = 4290 bits. The next matrix axes are defined using C and D, thus it's size is 67 by 68 bits. Since, in this example, there are no more characters, simply begin again at the beginning of the password with A and B. Since the password is case-sensitive, all 256 ASCII characters are usable.

```
______________________________________________________*/

void SetPassWord(char *passwordarg)
{
          unsigned int len, size;

          size = len = strlen(passwordarg);

          if(len % 2)
                    ++size;

          PassWord = (char *)malloc((size + 2) * sizeof(char));

          if(PassWord == NULL) {
                    puts("\nOut of Memory\n");
                    exit(1);
          }

          strcpy(PassWord, passwordarg);

          if(len !=size) {
                    *(PassWord + len) = *passwordarg;
                    *(PassWord + size) = '\0';
          }

          PassWordLen = size;
```

```
    }

/*______________________________________________________

Here, the second password provided by the user is used as a tool to decide
which rule to use.

The ASCII character set is implicitly divided into blocks, each the size in
characters as the number of rules available. Then the relative position of each
password2 character within its block is found. That position is one of the
numbers 0 to NUM_RULES - 1. This points, then, to a rule in the rule-base.

This is a good way to do it since, at any time, the number or order of rules in
the rule-base can be changed. If the order or number of rules is changed, then
this code does not have to be changed.
______________________________________________________*/

void SetEncryptionMethod(char *method)
{
        unsigned int len, val, x;
        int *eptr;

        len = strlen(method) + 1;

        EncryptionMethod = (int *)malloc(len * sizeof(int));

        for(eptr = EncryptionMethod; *method !='\0'; method++,
        eptr++) {

            val = (*method / NUM_RULES) * NUM_RULES;

            for (x = 0; x <NUM_RULES; x++)
                    if ((val + x) == *method)
                            *eptr = x;
        }
        *eptr = -1;

        RuleID = *EncryptionMethod;
}
```

/*____

This code takes the optional command line switch and parses it for inversion frequency. The program then inverts each Xth matrix before encrypting it.

Consider an example. Suppose the user enters the following command line:

```
BME I filename ABC 12345 /i=3
```

The first password 'ABC' is lengthened to 'ABCA' so that it will contain an even number of characters. The letters are then converted to the ASCII decimal equivalents (65, 66, 67, 65) and used, alternately, as X and Y matrix axis values. The second password '12345', by application, translates to rule pointers 0, 1, 2, 3, and 4, respectively. The '/i=3' sets the inversion frequency to 3, telling the program to invert every third matrix prior to its alteration by its assigned rule.

The following table describes each matrix size, which rule is use, and whether or not the matrix is inverted.

Matrix Size in Bits			Rule Used	Inverted?	# Bytes
X	Y	Size	0 to Total	Y or N	Accum.
65	66	4290	0	N	536 1/4
67	65	4355	1	N	1080 5/8
65	66	4290	2	Y	1616 7/8
67	65	4355	3	N	2161 1/4
65	66	4290	4	N	2697 1/2
67	65	4355	0	Y	3241 7/8
65	66	4290	1	N	3778 1/8
67	65	4355	2	N	4322 1/2
65	66	4290	3	Y	4858 3/4
67	65	4355	4	N	5403 1/8

...and so on to 30 different entries before it begins to repeat.

If the command line is symbolized as:

```
    BME mode filename P1 P2 /i=F,
```

the way to calculate the number of table entries is:

	# Chars in P1 (rounded up to the nearest even number)
Divided by	2
Multiplied by	# Chars in P2
Multiplied by	F(if defined)

Therefore, this example is:

```
    P1 = 'ABC', P2 = '12345', F=3
```

The number of characters in P1 = 3, rounded up to the nearest even number equals 4. The number of characters in P2 = 5. Therefore, the number of table entries for this example is:

```
    (4/2) * 5 * 3 = 2 * 5 * 3 = 30
____________________________________________________*/

void SetInversionFreq(char *freq)
{
    freq +=3;

    InversionFreq = atol(freq);
    if (InversionFreq <1L)
            InversionFreq = 0L;
    else
            Inversion = ON;
}

/*__________________________________________________
```

This function writes out all remaining data, closes files, deletes the original file, renames the encrypted temporary file to the name of the original file, and then cleans up all allocated memory.

```
____________________________________________________*/
```

```
void CleanUp(char *fname)
{
     FlushOutBuffer();

     fclose(InFile);
     fclose(OutFile);

     unlink(fname);
     rename(OutFileName, fname);

     DisplayProgress();

     free(Matrix.matrix);
     free(InBuffer);
     free(OutBuffer);
     free(PassWord);
     free(EncryptionMethod);

/*________________________________________________

Provide instructions if procedures not correctly followed.
________________________________________________*/

void NoArguments(void)
{
     puts("\nBME mode infile password method {/i=n}\n");
     puts ("Modes are: IN | OUT");
     exit(1);
}

/*__________________

Other Functions
__________________*/

void Display Title(void)
{
     puts("\nBME - Binary Matrix Encryption");
     puts("Version 1.0 - Release 09.94");
```

```
    puts("(c) 1994: Michael D. Harold & Joseph M. Morgan\n");
}

void Display Progress(void)
{
    printf("\rProcessing:  (%6.2f%c)",  ((float)TotalBytesRead  /
    (float)FileSize * 100., 37);
}


/*_____________________________________________

The following functions handle the matrix.
______________________________________________*/


/*_____________________________________________

This function manages the rotation through password1 using adjacent
characters to set the matrix size. Note that this implementation forces a matrix
axis to be at least 2.
______________________________________________*/

void DefineMatrix(void)
{
    static unsigned int PassWordPtr = 0;
    static char MatrixDefined = NO;

    if (MatrixDefined == NO)
        AllocateLargestMatrix();

    Matrix.x = (long)*(PassWord + PassWordPtr++);
    Matrix.y = (long)*(PassWord + PassWordPtr++);

    if (PassWordPtr == PassWordLen)
        PassWordPtr = 0;

    if (Matrix.x <2)
        Matrix.x = 2;

    if (Matrix.y <2)
```

```
        Matrix.y = 2;

    Maxtrix.size = Matrix.x * Matrix.y;

    MatrixDefined = YES;
}

/*__________________________________________________

Instead of allocating and freeing memory as the matrices shift their size, the
largest matrix is allocated and that space used.
__________________________________________________*/

void AllocateLargestMatrix(void)
{
    unsigned size, MaxSize = 0, x, y;
    char *ptr = PassWord;

    while (*ptr !='\0') {
            x = *ptr++;
            y = *ptr++;

            size = x * y;
            if (size > MaxSize)
                    MaxSize = size;
    }

    Matrix.matrix = (char *)malloc((MaxSize +1) * sizeof(char));

    if (Matrix.matrix == (char *)NULL) {
            puts("\nOut of Memory\n");
            exit(1);
    }
}

/*__________________________________________________

This function manages the conversion from byte-based reading to bit-based
reading. Note that a matrix is simply an n-Bit word. So, an n-Bit word of
```

```
matrix size is filled with bits from the input.  To optimize the process (and it
could be better), bits are added to the matrix 8-at-a-time.
___________________________________________________________*/

int FillMatrix (void)
{
    int c;

    ClearMatrix();

    if (strlen(LeftOver)) {
            strcpy(Matrix.matrix, LeftOver);
            LeftOver [0] = '\0';
    }

    if (strlen(Matrix.matrix) = = Matrix.size)
            return 1;

    while ((c = GetNextByte()) !=EOF) {

        TotalBytesRead+ +;

        if (AddByteToMatrix((unsigned char)c) = = FULL) break;
    }

    if (c = = EOF && !(feof(InFile))) {
        puts("\nUnexpected EOF");
        exit(1);
    }

    return c;
}

/*__________________________________________________________

To keep things neat, even though unnecessary, the following program sets the
matrix to NULL.
___________________________________________________________*/

void ClearMatrix(void)
{
```

```
          unsigned long x;

          for (x = 0; x < =Matrix.size; x+ +)
                    *(Matrix.matrix + x) = '\0';
}

/*________________________________________________

This function adds bits to the matrix, up to 8-at-a-time.  If it is unable to add
all 8 bits to the matrix, it copies the remaining bits to a holding bin called
LeftOver.  Note its use above in FillMatrix.
_______________________________________________*/

char AddByteToMatrix(unsigned char byte)
{
          char bytestr[9];
          int nbits;

          chartobinstr(byte, bytestr);

          nbits = (Matrix.size - strlen(Matrix.matrix));
          nbits = (nbits > 8) ? 8 : nbits;

          strncat(Matrix.matrix, bytestr, nbits);

          if (nbits < 8)
                    strcpy(LeftOver, (bytestr + nbits));

          return  (strlen(Matrix.matrix)  ==  Matrix.size)  ?  FULL  :
          NOTFULL;
}

/*________________________________________________

This next three functions control the physical input and output of data via two
16K buffers.
_______________________________________________*/

int GetNextByte(void)
}
```

```
     static int BytePtr = 0;
     static int BytesInBuffer = 0;

     if (BytePtr == BytesInBuffer) {
             BytesInBuffer = fread(InBuffer, sizeof(unsigned char),
             MAX_BUFFER - 1, InFile);

             if (BytesInBuffer == 0)
                     return EOF;

             BytePtr = 0;
     }

     return *(InBuffer + BytePtr++);
}

void PutNextByte(int c)
{
     *(OutBuffer + OutByteCount) = c;

     if (++OutByteCount == MAX_BUFFER - 1) {
             fwrite(OutBuffer,sizeof(unsigned char), MAX_BUFFER - 1,
             OutFile);
             OutByteCount = 0;
     }
}

void FlushOutBuffer(void)
{
     fwrite(OutBuffer, sizeof(unsigned char), OutByteCount, OutFile);
}


/*_____________________________________________________

This function handles the output of single bits. It accumulates bits, for
example only, into an 8-bit buffer to translate back into bytes to accommodate
the architecture. Once 8 bits are accumulated, it is converted into a real byte
and sent to the output buffer.
_____________________________________________________*/

void WriteBit(char bit)
```

```
{
     static int BitCanPtr = 0;
     BitCan[BitCanPtr++] = bit;

     if (BitCanPtr == 8) {
               PutNextByte(GetBitStrVal(BitCan));
               BitCanPtr = 0;
               ClearBitCan();
     }
}

void ClearBitCan(void)
{
     int x;

     for (x = 0; x < 8; x++)
               BitCan[x] = '\0';
}


/*____________________________________________________________

The following function receives a bit stream and converts it into an actual
unsigned character value.
______________________________________________________________*/

unsigned char GetBitStrVal(char *bitstr)
{
     unsigned char value, x;

     for (x = 128, value =0; *bitstr !='\0'; bitstr++, x /=2)
               if (*bitstr == '1')
                         value + = x;

     return value;
}
```

```
/*_______________________________________________

The following function determines when it is time to invert the matrix.
Returns YES or NO.
_______________________________________________*/

int TimeToInvert(void)
{
    if (Inversion == OFF || MatrixCount < InversionFreq ||
    !InversionFreq)
        return NO;

    if (!(MatrixCount % InversionFreq))
        return YES;

        return NO;

}

/*_______________________________________________

Following are various rules by which to encrypt the matrix. Envision the X
axis as vertical and the Y axis as horizontal.
_______________________________________________*/


/*_______________________________________________

The following rule writes bits beginning from the bottom right corner of the
matrix, moving left through that row, then moving up one row to the end and
continuing in that manner until the matrix is completely written out.
_______________________________________________*/

void BottomLeftUp(void)
{
    register long x, y;
    unsigned int len;

    len = strlen(Matrix.matrix);

    for (x = Matrix.x - 1; x>=0; x--)
```

```
          for (y = Matrix.y - 1; y>=0; y--)
                SendBitAtXY(x, y, len);
}

/*________________________________________________________

The following rule starts at the bottom left corner, moves right through the
row, moves up one row to the beginning and continues until the matrix is
written out.
________________________________________________________*/

void BottomRightUp(void)
{
     register long x, y;
     unsigned int len;

     len = strlen(Matrix.matrix);

     for (x = Matrix.x - 1; x>=0; x--)
          for (y = 0; y < Matrix.y; y++)
                SendBitAtXY(x, y, len);
}

/*________________________________________________________

The next rule starts at the bottom right corner, moves up through the column,
then moves left one column at the bottom, and continues.
________________________________________________________*/

void BottomUpLeft(void)
{
     register long x, y;
     unsigned int len;

     if (Mode == OUT)
          SwapScale();

     len = strlen(Matrix.matrix);

     for (y = Matrix.y - 1; y >=0L; y--)
```

```
                for (x = Matrix.x - 1; x > =0L; x–)
                        SendBitAtXY(x, y, len);
}

/*_____________________________________________________

This rule starts at the bottom left corner, moves up through the column, moves
right to the bottom of the next column, and continues.
_____________________________________________________*/

void BottomUpRight(void)
{
        register long x, y;
        unsigned int len;

        if (Mode == OUT) {
                strrev(Matrix.matrix);
                SwapScale();
        }

        len = strlen(Matrix.matrix);

        for (y = 0L; y < Matrix.y; y++)
                for (x = Matrix.x - 1; x > =0; x–)
                        SendBitAtXY(x, y, len);
}

/*_____________________________________________________

This rule starts at the top right corner, moves down through the column,
moves left to the bottom of the next column, and continues.
_____________________________________________________*/

void TopDownLeft(void)
{
        register long x, y;
        unsigned int len;

        if (Mode == OUT) {
                strrev(Matrix.matrix);
                SwapScale();
        }

        len = strlen(Matrix.matrix);

        for (y = Matrix.y - 1; y > =0; y–)
                for (x = 0L; x < Matrix.x; x + +)
                        SendBitAtXY(x, y, len);
}


/*______________________________________________________

The next rule starts at the top left corner, moves down through the column,
then right to the top of the next column, and continues in like manner.
______________________________________________________*/


void TopDownRight(void)
{
        register long x, y;
        unsigned int len;

        if (Mode = = OUT) {
                SwapScale();
        }

        len = strlen(Matrix.matrix);

        for (y = 0L; y < Matrix.y; y+ +)
                for (x = 0L; x <Matrix.x; x+ +)
                        SendBitAtXY(x, y, len);
}


/*______________________________________________________

The next rule starts at the top right corner, moves left through the row, down
and to the end of the next row, and continues in like manner.
______________________________________________________*/
```

```
void TopLeftDown(void)
{
     register long x, y;
     unsigned int len;

     len = strlen(Matrix.matrix);

     for (x = 0; x < Matrix.x; x++)
               for (y = Matrix.y-1; y >=0; y--)
                         SendBitAtXY(x, y, len);
}


/*______________________________________________________

This rule starts at the top left corner, moves right through the row, down and
to the beginning of the next row, and so on. Since this is exactly the way the
matrix is constructed, it doesn't actually perform any encryption. It is put here
to complete the rule set, but this implementation does not use it.
______________________________________________________*/

void TopRightDown(void)
{
          register long x, y;
          unsigned int len;

          len = strlen(Matrix.matrix);

          for (x = 0; x < Matrix.x; x++)
                    for (y = 0; y < Matrix.y; y++)
                              SendBitAtXY(x, y, len);
}


/*______________________________________________________

These are functions common to the matrix conversion rules.
______________________________________________________*/
```

```
/*______________________________________________________________

This function is required to decrypt certain rules. The matrix axes have to be
swapped to properly recover the data.
_____________________________________________________________*/


void SwapScale(void)
{
            unsigned long n;

            n = Matrix.x;
            Matrix.x = Matrix.y;
            Matrix.y = n;
}


/*______________________________________________________________

This function inverts the matrix, turning every '1' bit to a '0', and every '0'
bit to a '1'.
_____________________________________________________________*/

void InvertMatrix(void)
{
      char *ptr;

      for (ptr = Matrix.matrix; *ptr != '\0'; ptr++)
            *ptr = (*ptr=='1')?'0':'1';
}


/*______________________________________________________________

This function manages the rotation through the rule pointers defined by
password2.
_____________________________________________________________*/


void SetNextMethod(void)
{
      static int MethodPtr = 1;
```

```
        RuleID = *(EncryptionMethod + Method Ptr++)

        if (*(EncryptionMethod + MethodPtr) == -1)
             MethodPtr = 0;
}

/*____________________________________________________

This function locates the bit in the matrix at x, y and writes it to the output
stream.
____________________________________________________*/

void SendBitAtXY(long x, long y, unsigned int len)
{
        unsigned long offset;

        offset = (x * Matrix.y) + y;

        if (offset < (long)len)
             WriteBit(*(Matrix.matrix + offset));
}

/*____________________________________________________

This function defines the scale by the number of bits in the n-bit data type.
Very few files will divide up evenly into the shifting matrix series. Therefore,
there will usually be an incomplete matrix at the end. These rules won't
properly encrypt/decrypt a partial matrix. Therefore, this function resets the
matrix scale to an optimal axis based upon the number of remaining bits.
However, if the length of the remaining n-bit word is prime, the last bit is held
and removed from the matrix, and then the optimal axis is calculated. Once
the new matrix is encrypted, if there is a remaining bit, it gets written out.
____________________________________________________*/

void GetOptimalScale(void)
{
             unsigned ResultNumber;
             unsigned HighLimit;
             unsigned x;
        HighLimit = Matrix.size;

        Matrix.x = Matrix.y =0;

        for (x = 2; x < HighLimit; x++) {

                if (Matrix.size % x)
                        continue;

                ResultNumber = HighLimit = Matrix.size / x;

                Matrix.x = x;
                Matrix.y = ResultNumber;
        }

        Matrix.size = Matrix.x * Matrix.y;
}

/* End of program code */
```

FIG. 4 illustrates, by example, the elemental steps of data compression. Compression is begun by the initiation process step 80 which includes such items as the declaration and initialization of variables, the allocation of memory, and parsing user input. The next step 82 involves pointing to the input data, output data, and other support data needed by the compression method(s) at step 86.

From there, the process begins by reading in an implementation of a defined amount of data. It is specific to this invention that the data is read as one or more n-bit data types at step 84 consistent and parallel with any given implementation. The data is then processed by the compression method(s) at step 86. The resulting data, now in compressed form, is written to output data at step 88.

It is then determined at step 90 whether or not compression has been completed. If not, it is determined if the end of the input data has been

FIG. 4 illustrates, by example, the elemental steps of data compression. Compression is begun by the initiation process step **80** which includes such items as the declaration and initialization of variables, the allocation of memory, and parsing user input. The next step **82** involves pointing to the input data, output data, and other support data needed by the; compression method(s) at step **86**.

From there, the process begins by reading in an implementation of a defined amount of data. It is specific to this invention that the data is read as one or more n-bit data types at step **84** consistent and parallel with any given implementation. The data is then processed by the compression method(s) at step **86**. The resulting data, now in compressed form, is written to output data at step **88**.

It is then determined at step **90** whether or not compression has been completed. If not, it is determined if the end of the input data has been reached at step **92**. If not, the process forks backs to the read process at step **84**, otherwise it forks to the application of the compression method(s) at step **86**.

If it has been determined by the implementation that compression has been completed at step **90**, then the necessary data is released at step **94**, and the process is completed at step **96**.

For example, and not by way of limitation, the following description illustrates one method of implementing an n-bit virtual software machine capable of performing reiterative loss-less data compression.

Most methods of loss-less data compression are based on a method in which repetitive patterns or symbols within a data file are first identified and then replaced with symbols which occupy, on average, less space in memory than the original symbols. Examples of loss-less data compression techniques include Huffman Coding, Arithmetic Coding, Dictionary-Based Compression and Lempel-Ziv Coding. Each of these methods relies on the substitution of a smaller binary string for a larger binary string based on one or more repetitive patterns of symbols or symbol patterns, or the frequency of bytes or patterns within the uncompressed data. The desired result of this process is a file which is smaller than the original.

In order for compression to occur, an uneven frequency distribution of symbols or symbol patterns must be present in the uncompressed file. The greater the unevenness of the frequencies of the symbols or symbol patterns in the original data, the greater the compression.

Currently, all known methods of loss-less data compression result in an even distribution of symbols in the compressed file. Because loss-less data compression methods rely upon the uneven distribution of symbols or symbol patterns, the even frequency distribution makes further compression undesirable or impossible.

With this invention, the concept of Reiterative n-Bit Compression (RNC) uses variable length n-bit data types to explicitly or implicitly redistribute the frequency with which symbols occur within the data. After one iteration, the frequency distribution of the symbol set representing the data may be modified explicitly by changing the size in bits of the input and/or output data type. This explicit frequency redistribution of the symbol set allows the data to be compressed reiteratively.

The following table illustrates the differences in the resulting distribution of some 8-bit characters following compression with different sized n-bit data types.

Characters	Space	A	E	I	O	U
Data Type Size in Bits	Original Distribution Count of Each of the Above Characters					
	4376	498	749	836	352	283
	Distribution Count Following Compression					
4	143	114	114	112	46	33
6	217	122	117	138	59	48
8	139	42	59	42	25	29
10	260	107	120	129	40	42
12	161	64	64	61	13	6

The first row of the above table lists the characters being evaluated for their frequency within the original file. The third row shows each character's distribution in the original input file prior to its compression. Rows **5** through list each character's distribution following compression of the original file using input data types of varying bit lengths. The first column lists the length, in bits, of the input n-bit data type.

Referring to the table, the "space" character appears 4376 times in the original file. After compressing the file using a 4-bit data type, the "space" appears 143 times. In contrast, after the file is compressed using a 10-bit data type, the "space" appears 260 times. When the file is compressed using a 6-bit data type, the character 'U' appears 48 times, compared with only 6 times following compression using a 12-bit data type.

By forcing the redistribution of bits, the compression ratio can be forced to vary, thereby permitting optimization of the next compression pass. The following table shows an example of compression results of data. The first column lists the size, in bytes, of the file before a given compression pass. The second column lists the size of the n-bit data type. The third column lists the number of the actual compression pass. The fourth column lists the size of the file, in bytes, following the compression pass.

Some compression passes have been attempted more than once with varying sized input data types. This allows for selection of the best compression ratio for a given n-bit data type. Compression attempts that do not result in compression are italicized. The compression pass that achieved the most compression is boldfaced.

Size	Nbits	Pass	Comp. Size
18119	10	1	14174
14174	10	2	14157
14157	10	3	14156
	9	3	14136
	8	3	13900
	7	3	14091
	6	3	14059
	5	3	14558
13900	8	4	13832
	7	4	13900
	6	4	13907
	9	4	13897
	10	4	13894
	11	4	13911

It should also be noted that frequency redistribution can be achieved by implementing such data conversion algorithms as alternating block-based inversion, bit-shifting, or exclusive OR operations tailored for the target, or a multiple of the target data type.

As another example, and not by way of limitation, the following description illustrates one method of implementing a virtual machine or computer capable of performing rule-based n-bit arbitrary precision arithmetic:

Rule-based n-Bit Arithmetic (RNA) performs binary arithmetic operations on fixed or floating point data of arbitrary precision where such precision is limited only by the real or virtual address space of the computer. Arithmetic operations include, but are not limited to, binary addition, subtraction, multiplication, and division.

As part of its method, RNA contains two new data types whose notations are unique and are not described in any previously existing data types or data notations. One of these notations represents integer values. The other notation represents floating point values.

The majority of computers and computer languages now conforms to the following internal representation of integer values:

1. The left-most bit, the sign bit ("S"), is used to contain the sign of the number, with the usual interpretation that 0 means positive or plus and 1 means negative or minus.
2. The "decimal" or radix point is assumed to be affixed at the left or right end of the number.
3. The remaining bits represent the binary values ("B") of the number.

A standard signed 16-bit value would be stored as follows:

S (1 bit)	B (15 bits)

radix

With regard to floating point numbers, the following specific notation endorsed by the IEEE (Institute of Electrical and Electronic Engineers) is the standard:

1. The left-most bit is the sign bit ("S").
2. The next eight bits are the exponent ("E"). The exponent is interpreted as an integer in excess-127 code. Excess-127 code allows the exponent to represent numbers from -127 through 128.
3. The remaining bits are the mantissa ("M"). The value of the mantissa is normally defined as 1 plus the value of "M" treated as a binary fraction with the radix at the left end.

A standard signed, single precision floating point value would be stored as follows:

S (1 bit)	E (8 bits)	M (23 bits)

radix

This data type has an upper limit of 64 bits (double precision) and 128 bits (quadruple precision).

The following data types represent RNA integer and floating point values.

RNA integer values are represented as follows:

The left-most bit, the sign bit ("S"), is used to contain the sign of the number with the interpretation that 0 means positive and 1 means negative.

The first n-Bit value to the right of the sign bit identifies the length in bits ("L") of the binary representation of the number. The size and limit of this value is implementation specific.

The second n-Bit value to the right of the sign bit represents the binary values ("B") of the number beginning with the least significant digit (LSD) and extending to the most significant digit (MSD).

A signed n-Bit integer value would be stored as follows:

S (1 bit)	L (n bits)	B (L bits)

For example, any binary integer value from 1 to 16,777,216 significant digits in length could be stored with the following n-Bit data type:

S (1 bit)	L (24 bits)	B (L bits)

RNA floating point values are represented as follows:

1. The left-most bit, the sign bit ("S"), is used to contain the sign of the number with the interpretation that 0 means positive and 1 means negative.
2. The first n-Bit value to the right of the sign bit identifies the length in bits ("L") of the binary representation of the number. The size and limit of this value is implementation specific.
3. The second n-Bit value to the right of the sign bit identifies the radix point of the number ("B"). The size in bits of this field is identical to the size of the previous n-Bit field, (L).
4. The third n-Bit value to the right of the sign bit represents the binary values ("B") of the number beginning with the least significant bit (LSB) and extending to the most significant bit (MSB).

A signed n-Bit floating-point number would be stored as follows:

S (1 bit)	L (n bits)	R (n bits)	B (L bits)

For example, any binary floating-point value from 1 to 16,777,216 significant digits in length could be stored with the following n-Bit data type:

S (1 bit)	L (24 bits)	R (24 bits)	B (L bits)

Binary addition, subtraction, multiplication, and division are accomplished with n-Bit data types using standard methods. Addition may be performed using binary adders. Subtraction may be performed by using "true complement" notation and adding the minuend to the complemented subtrahend. Multiplication is the result of repeated binary addition. Division is the result of repeated binary subtraction.

To add the following two n-Bit floating-point numbers:

11.001 + 1.011011

11.001 =

S = 0	L = 0101	R = 0010	B = 11001

1.011011 =

S = 0	L = 0111	R = 0001	B = 1011011

```
       11.001
  +  1.011011
(LSB) 100.100011  (MSB) =
```

S = 0	L = 1001	R = 0011	B = 100100011

Negative numbers may be stored in complemented form to facilitate the ease of subtraction. By using "true complement" notation (i.e., reversing the value of each binary digit in the representation of the number), a binary adder may be used to accomplish binary addition, subtraction, multiplication, and division. This means that RNA may be implemented in microcode or at the level of hardware circuitry. At this level, RNA may be implemented as a microprocessor function or as a specific purpose hardware component of a general purpose computer.

FIG. **5** describes an example of an Arithmetic Logic Unit (ALU) which implements Rule-based n-Bit Arithmetic (RNA). One or more n-Bit values are retrieved from memory **100** by the command processor **110**. The command processor **110** interprets the values as data or instructions, depending on their locations in memory **100**, and submits them to the rule-base interface **120**.

If the instruction is a logic instruction, the data and the instruction are submitted to the logic rule-base **130**. Logic rules include, but are not limited to, AND, OR, NOT, NAND, and NOR logic functions. Once the appropriate rule has been applied to the data, the result is returned to the rule-base interface **120**. Additional logic and/or arithmetic rules may be applied to the data before it is returned to memory **100**.

If the instruction is an arithmetic instruction, the data and instruction are submitted by the rule-base interface **120** to the arithmetic rule-base **140**. Arithmetic rules include, but are not limited to ADDITION, SUBTRACTION, MULTIPLICATION, and DIVISION. Once the arithmetic rule **140** has been applied to the data, the result is returned to the rule-base interface **120**. Additional arithmetic and/or logic rules may be applied to the data before it is returned to memory **100**.

The advantages of RNA are:

1. It provides greater precision than any other arbitrary precision arithmetic method.
2. The size of the numbers used in RNA is limited only by the real or virtual address space of the computer.
3. RNA may be implemented in hardware of software.
4. RNA is processor independent.
5. RNA provides faster calculations of very large arbitrary precision numbers.

A specific purpose RULE-BASED n-BIT VIRTUAL SOFTWARE MACHINE, as uniquely described by this invention, is any specific purpose virtual software machine which uses a rule-base as an instruction set to perform binary string operations on n-bit data types.

The COMMAND PROCESSOR is a machine that uses a program which receives n-bit data types and command language instructions as input and performs operations upon the input using one or more rules. Each rule is a type of processor instruction which performs a binary string operation upon one or more n-bit data types. After the input data has been processed, the command processor outputs data in the form of one or more n-bit data types.

An n-bit data type is defined as a data type consisting of n bits (or binary digits) where n is any number greater than zero. There is no inherent upper limit on n-bit data types. Variable length n-bit data types are used as standard input and output and are maintained and managed by the invention.

The RULE-BASE INTERFACE is defined as a method of transferring data between the command processor and the rule-base. The data in the form of one or more n-bit data types is passed to the rule-base interface by the command processor. The rule-base interface, in turn, identifies the rule or rules that are to be used in processing the data. The data is dispatched as one or more arguments to the selected rule within the rule-base and the rule is applied. After the data has been modified in accordance with the specified rule or rules, the modified data is returned to the rule-base interface. The rule-base interface may iteratively submit the data to one or more rules. Once the conditions for the modification of the data by the rule-base have been satisfied, the data is returned as one or more n-bit data types to the command processor. The command processor then outputs the data.

The RULE-BASE INTERFACE manages access to the rules within the rule-base using any access method including, but not limited to, linked lists, tree structures, relational database tables, and hyper-link stacks.

The RULE-BASE is a collection or set of rules. Each rule applies a binary string operation to the input data. A BINARY STRING OPERATION is any operation which performs bit level operations on one or more binary strings representing n-bit data types. A binary string operation may emulate processor instructions such as binary ANDs, ORs, XORs, and COMPLEMENT operations. Combinations of these operations may emulate processor instruction sets with the additional advantage of providing virtual n-bit data and instruction registers within the virtual machine in which to perform these operations. Binary string operations may also emulate more complex operations such as addition, subtraction, multiplication, division, vector, and matrix operations. These operations are implemented in such a way that there is no inherent upper limit on the length of the n-bit data types used as input or output.

The types of data structures which may be used to implement the rule-base as it is defined in the invention include, but are not limited to, the following: relational database tables, C or C++ language header files, any generation computer language function(s) or subroutine(s), object class libraries, and EPROM assembly language subroutines, and microcode instruction sets.

Although the invention and several of its preferred embodiments have been described and illustrated in detail, the same is by way of example only and should not be taken by way of limitation. The spirit and scope of the present invention are limited only to the terms of the appended claims.

We claim:

1. A method for encrypting information data from a data source comprising the steps of:

coupling at least one n-bit data string of input data as variable length n-bit data types containing bits representing said information data and including control bits to a command processor;

storing a plurality of encryption rules in a rule-base memory for processing the n-bit data string;

coupling a rule-base interface between said command processor and said rule-base memory for identifying specific encryption rules stored in said rule-base memory according to said control bits in said n-bit input data string received from said command processor;

modifying the n-bit data string in the rule-base in accordance with the identified encryption rules to encrypt said information data bits; and

transferring said encrypted data to said command processor for output as variable length n-bit data types.

2. A method as in claim **1** further including the steps of:

identifying said data source with said control bits in said n-bit data string; and

including bits in said control bits that represent at least one argument to be used when accessing said rule-base.

3. A method as in claim **1** further comprising the steps of:

storing said rules in a memory in said rule-base; and

defining said rules as binary string operations.

4. A method as in claim **2** further comprising the steps of:

coupling one or more of said arguments to said identified rule within said rule-base; and

applying the identified rule to the n-bit information data to modify said information data and to perform said encryption.

5. A method as in claim **4** further including the steps of:

appending additional arguments, as needed, to said modified information according to said identified rule in said rule-base; and

returning the modified information data to said rule-base interface along with said needed arguments.

6. A method as in claim **5** further including the steps of:

iteratively submitting said modified data to at least another one of said rules stored in said rule-base in accordance with said arguments appended by said identified rule for further modification until said data modification satisfies all of said arguments; and

returning said satisfied modification data to said command processor as one or more n-bit data strings that are not required to correspond to the n-bit size and number of n-bit input data strings coupled to said command processor from said data source.

7. A method as in claim **5** further including the step of encrypting said information data to prevent the determination of the original position of any bit in the bit stream of the input information data based upon the position of any bit in the encrypted bit stream.

8. A method as in claim **7** further including the step of encrypting and decrypting said information data with said rule-base such that neither encryption nor decryption is dependent upon an explicit encryption key, a specific encode/decode rule, or a specified data type.

9. A method as in claim **5** further comprising the step of user-defining the implementation of said encryption/decryption since no single algorithm or rule must be defined.

10. A method as in claim **1** further including the steps of:

providing multiple encryption keys to a single encrypted message such that parts of an encrypted message are accessible only to some users and not others; and

distributing the encryption/decryption of a message across multiple machines for multiple users.

11. A method as in claim **10** further including the step of defining any encryption key explicitly.

12. A method as in claim **10** further including the step of defining any encryption key implicitly.

* * * * *