A DISTRIBUTED PROGRAMMING MODEL AND ITS APPLICATIONS TO COMPUTATION INTENSIVE PROBLEMS FOR HETEROGENEOUS ENVIRONMENTS

Yuan Shi
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122
shi@fac.cis.temple.edu
(215)787-6437

(Published in AIP Conference Proceedings 283, Earth and Space Science Information Systems, Pasadena, CA 1992, Editor: Arthur Zygielbaum, pp. 827-848)

ABSTRACT

Recent advances in high performance computing architectures have presented a clear trend that future systems must include computers from different classes. It is conceivable that effective large scale computing in general must be done in a heterogenous distributed environment. The reported research seeks to build a generic virtual processor model on top of heterogeneous computing and communication devices. By using a specification based approach, we can effectively customize the available heterogeneous devices for every computer application. In this paper, we shall present the computational results of three field applications in scientific visualization, engineering simulation and financial simulation using a Scatter-And-Gather method (or virtual vector processing). To aid objective evaluation of the virtual processor model, we also include the program re-engineering costs for achieving such performances.

Keywords: Distributed Heterogeneous Computing, Distributed Operating System.

1. INTRODUCTION

High computing efficiency can be achieved by parallelizing an application over a given computing architecture. In this article we intend to generalize the commonly known approaches to parallelize an application over a set of heterogeneous computing architectures by organizing coarse grain parallel components.

There are three basic types of parallelizable components in every computing application: SIMD, MIMD and pipelined. In order to exploit the full potential of the available computing powers and existing parallelism of a given application, the granularity of parallel components must vary to optimally offset the communication latency. If the distributed environment is non-volatile, i.e. it is single user oriented, a parallel compiler can produce fairly optimized codes for a given hardware architecture. Heterogeneous software and communication protocols in typical volatile distributed environments have made both building a generic distributed operating system and a "heterogeneous parallel compiler" very difficult.

The main focus of the reported research is to promote a virtual processor model that can be constructed dynamically on top of heterogeneous computers and a software system (SYNERGY, 1990 U.S. Patent pending) to make such a model feasible. In particular, we shall report the computation and re-engineering results using coarse grain SIMD components for three field applications.

2. THE VIRTUAL PROCESSOR MODEL

The proposed virtual processor model consists of only three types of components: virtual SIMD, virtual MIMD and virtual pipeline. A processor assignment with respect to an execution environment for a computing application defines the virtual processor for that application. Obviously there is a virtual processor defined for every currently running distributed or non-distributed application.

The central idea of this virtual processor model is to customize a set of distributed computers for a given application by fitting a network of virtual SIMD, MIMD and pipelined components to an application's natural dataflow structure. Numerous tuning devices must be constructed to counter react to unexpected situations in typical volatile environments.

The interface of the virtual processor consists of a databus network of a given application and a distributable program -> processor mapping. The databus network is a network of distributable programs interconnected through databuses. Each distributable program is an independent process that can be dynamically loaded onto a range of processors. Each databus is a user defined abstract (distributed) data object for which a set of pre-defined operations can be applied. For example, a generic queue (or mailbox) can be defined as a databus along with its operations: open, close, read, write and post. A tuple space can also be a databus along with operations: open, close, put, read and get. A small set of such objects is suffice for most scientific computing problems.

The databuses are the essential media for building virtual parallel components. For example, the use of two or more tuple space objects can be used to construct a virtual vector processor and virtual pipe must employ a series of generic queues. Techniques used in vectorizing compilers to discover vectorizable elements can be applied here to discover coarse grain vectors with minor modifications.

A raw sequential program cannot be readily executed on such a virtual processor, if good performance is expected. A re-engineering process must be carried out to relax the internal dataflows of the given program. This will be further discussed in the programming example section.

3. EXPERIMENT ENVIRONMENTS

The reported results were obtained in the Visualization Laboratory and the High Performance Scientific Computing Laboratory at the IBM T.J.Watson Research Center, Yorktown Heights, NY, and the Hermes Financial Simulation Laboratory of the Wharton School of University of Pennsylvania, PA.

The hardware setup in the Visualization Laboratory consists of five workstations: three IBM RISC System 6000/530s, one Silicon Graphics 4D/120GTX and one Stardent 2000. These computers are connected through two local area networks: Ethernet and Token Ring. Three different flavored Unix operating systems : AIX, IRIX and Stellix were used in the experiments.

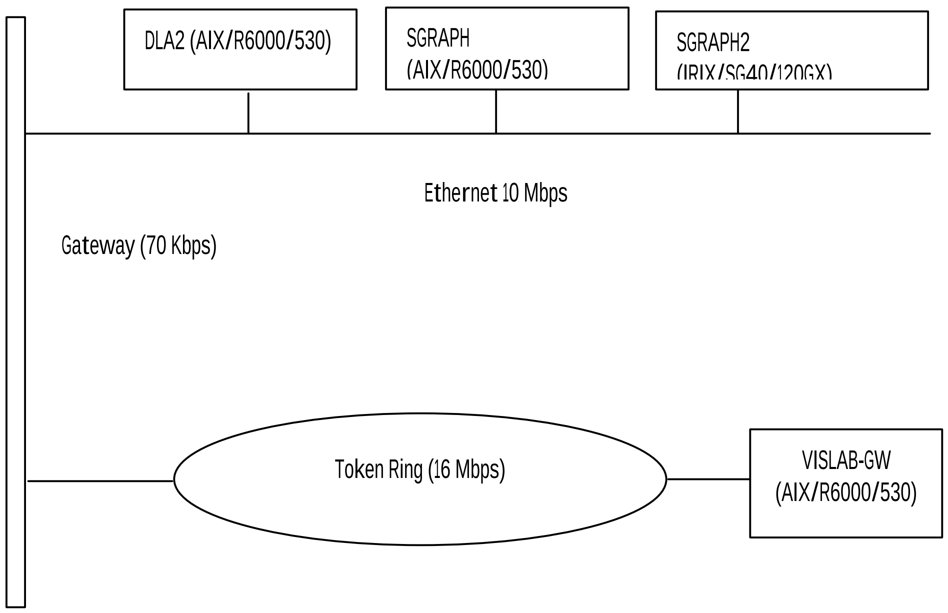

Figure 1. The Vislab Network at YKT

The High Performance Computing Laboratory has 15 IBM RISC/6000-540 workstations connected on a single Ethernet segment serving a number of computation intensive departments. All computers run AIX operating system.

The Hermes Laboratory has one DECstation 5000/200, two DECstation 3100s and four DECstation 2100s. All computers (running various Ultrix versions) are connected on a departmental Ethernet shared by faculty and graduate students.

The main objective of the Vislab experiment was to validate the real life performance limitations of a typical hybrid networking environment running typical visualization/engineering programs. In fact, the utility of node "vislab-gw" over the slow gateway was pretty much in question before the experiments. Other questions to be answered are "Can a computation intensive application load down the Ethernet?", "What will be bottlenecks in such a hybrid network?" and "How bad the bottleneck would affect the overall performance?"

The HPC lab's experiments were to validate the hypothesis that collective workstations can out perform supercomputers in a centralized computing service environment. One 3090 was indeed removed from the lab.

The Hermes Lab' experiment was to identify the limits of a set of heterogeneous processors connected by a volatile departmental network.

4. COMPUTING WITH VIRTUAL VECTOR PROCESSORS

A virtual vector is a Scatter-And-Gather (SAG) subsystem. Such a subsystem typically uses two Tuple Space objects - one for scattering computing elements and one for collecting the results. It can be grossly viewed as a coarse grain vector processor where the processing elements are the heterogeneous computers running the duplicated workers.

From the programmer's view point, a SAG has the databus network shown in Figure 2.

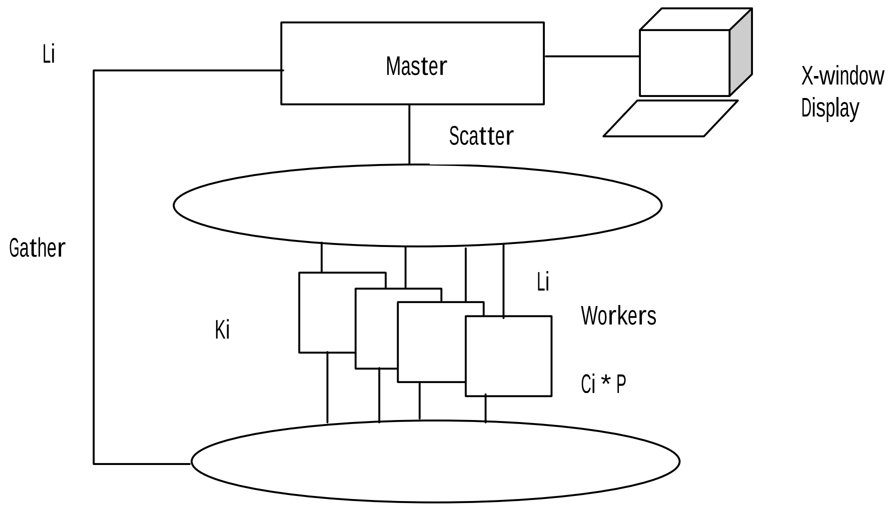

Figure 2. "Scatter-And-Gather" Computation Model

The MASTER program generates tuples to be computed according to a programmer controlled grain size. A tuple is a stream of data bits with a name. One can regard a tuple as a distributed programming variable in analogy with conventional programming variables. A tuple space is merely a collection of tuples. One can insert (put), extract (get) or inspect (read) a tuple in a tuple space. There are K identical worker programs running in parallel on K heterogeneous computers. They compete for tuples from one space and deliver the results to another space. All worker programs repeat until there are no tuples to be processed. The MASTER program must then assemble and process the returned results. The overall speed up factor of a SAG component is K if we disregard the overheads and assume all computers are of equal power. The first attempt to use tuple space for computation speed improvements was made in Linda [1,6,9,16]. We have generalized the tuple space concept to a coarse grain distributed object with adjustable grain sizes [13,14].

In this paper, we have assumed that the total amount of computation is known a priori for each SAG component. Relaxing this condition can lead to many creative designs such as distributed backtracking, branch-and-bound and blackboard-based inference systems. Our preliminary results have been near linear for problems require all optimal solutions and superlinear for problems require only one optimal solution. The details will be reported under a separate cover.

5. SAG PROPERTIES

In a virtual vector, since all processing elements are of unequal powers and are connected through slow communication devices, small grain size will cause excessive tuple communication overhead and large grain size will cause unsynchronized wait due to heterogeneity in computing/communication capacities and application requirements.

Mathematically, a SAG system (Figure 2) can be expressed as follows:

Let n : Number of workers (or distributed computers)
i : 1,2,...,n, index for workers
T : Total processing delay for a SAG application.
D : Total computation quantity (in bytes)
p : Density: computing units per byte ($0<=p<\text{infinity}$)
L_i : Average communication delay per byte to/from worker i.
C_i : Average computation delay per byte at worker i.
T_i : Total delay of worker i.
K_i : Number of tuples consumed by worker i.
d : Tuple size in bytes.

The total delay of the i-th worker T_i is the sum of computation and output delays:

$$K_i\, x d x (L_i + C_i\, r)$$

For simplicity, we assume all workers start computing upon the completion of inserting computing tuples. Thus, the total processing delay T includes the maximal worker delay plus the scattering costs:

$$\mathrm{MAX}_{i=1..n} T_i + \sum_{i=1..n} K_i\, x\, L_i$$

Finally, the total number of consumed tuples must be equal to what have to be computed:

$$d x \sum_{i=1..n} K_i = D$$

Summarizing, we have the following model for a SAG:

$$V = \mathrm{MIN}[\ \mathrm{MAX}_{i=1,2,...,n} T_i + \sum_{i=1,2,...,n} (K_i \, x \, L_i)]$$

Subject to:

$$d x \sum_{i=1...,n} K_i = D (1)$$

where

$$T_i = d x K_i \, x (L_i + C_i \, x r)$$

We can derive a heuristic algorithm in the following steps. By the principle of optimality, the minimal T must be obtained by equating T_i's, thus

$$T_1 = T_2 = ... = T_n = V = \mathrm{MAX}_{i=1..n} T_i (2)$$

This corresponds to balancing the loads and minimizing the synchronization costs among workers at the completion of their processing. Substituting T_i in (2) we obtain:

$$K_i = \frac{V}{d x (L_i + C_i \, x r)} (3)$$

Summing up (3):

$$\sum_{i=1..n} K_i = \frac{V}{d} x \sum_{i=1..n} \frac{1}{L_i + C_i \, x r} (4)$$

Substituting (1) and (4):

$$\frac{D}{d} = \frac{V}{d} x \sum_{i=1..n} \frac{1}{L_i + C_i \, x r} (5)$$

Simplifying (5):

$$V = \frac{D}{\sum_{i=1..n} \frac{1}{L_i + C_i \, x r}} (6)$$

This is the lower bound of total delay for the given application using n heterogeneous distributed computers.

Now if $(K_1, K_2, ..., K_n, d)$ is a solution to the system expressed, then the following are all solutions to the same system:

$(2K_1,2K_2,...,2K_n,d/2)$ $(3K_1,3K_2,...,3K_n,d/3)...(cK_1,...,cK_n,d/c)$

where K_i's are the tuples consumed by respective workers, c is a non-negative integer, d/c is the optimal tuple size satisfying T_i and (1).

To minimize the total delay T, we conclude that $(K_1,K_2,...,K_n,d)$ is the optimal solution.

In reality, we can obtain the sum of K_i's directly using the following algorithm.

a) Craft the application to SAG model.
b) Select a host computer for the SAG Controller and tuple spaces.
c) Allocate the worker to each available computer and log the delay
times for a "typical" tuple size d. This should result in a vector $T = (T_1,...,T_n)$, where n is the number of computers available for workers.
d) Let

$$T_x = \mathrm{MAX}_{i=1..n} T_i$$

e) Let $IDX_i = \mathrm{Round}(T_x/T_i)$. This is the relative power index vector same as that
shown in Table I.
f) The estimated optimal:

$$d = \frac{D}{\sum_{i=1..n} IDX_i}$$

One oversimplified factor in the above formulation is the non-linear effects of packet size to the communication delay. Figure 3 shows the actual measured L_i in the Mandelbrot experiments. Remember that the total amount of computation D is fixed.

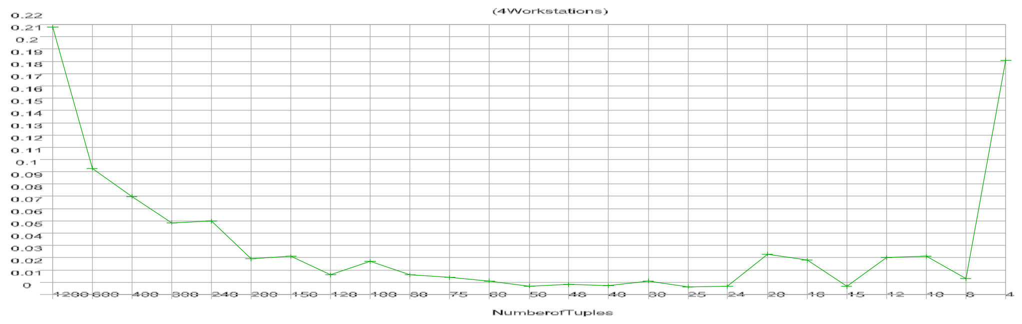

Figure 3. Effects of Packet Size to Communication Delay

For all practical purposes, if d = D in step (c) of the above algorithm, a few experiments with d, d/2, d/3, ..., can approach the actual optimal. The small integers can offset the delays of large packets while maintaining the solution structure. More elaborated effort requires finding the functional relations between the delays and tuple sizes. Every computer will have a different set of coefficients.

Note that above discussion assumes that the distributed environment is non-volatile. In a volatile environment the optimal d is going to change according to network and computer usage fluctuations. It is always a good idea to leave the tuple size tunable external to all subprograms. These tunable "knobs" become the parameters of an overall optimization algorithm.

Furthermore, the optimal d may not be always feasible due to insufficient resources on distributed computers. A practical solution is to find the smallest constant c to yield a sub-optima.

6. PROGRAMMING EXAMPLES

As mentioned earlier, sequential programs must be re-engineered to run on such a virtual processor. We have selected a Mandelbrot set display program and a LCD screen simulator as our re-engineering targets. The total amount of computation are known a priori - there are 1200x980 pixels to process for a fractal display and 1024 x 768 pixels for LCD screen simulation. Both programs have a wide range of possible grain data sizes (from 1 to 1200x980 for Mandelbrot and 768 to 1024x768 units for LCD screen simulation).

In our experiments, the SAG programs were first organized as in Figure 4 to obtain the relative power indexes.

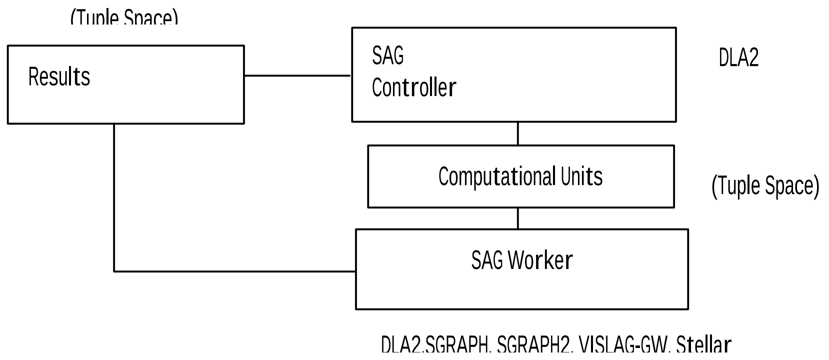

Figure 4. Relative Power Index Finding System

Table I shows the relative contributions the experimental computers made to the two applications with d = $D_{Mendelbrot}$ and D_{LCD} respectively. Note that "DLA2" hosts the client and the two tuple space servers.

The IDXi column represents the power indexes of these computers for the Mandelbrot application with the slowest being 1. Similarly, IDXi' represents relative power indexes for the LCD screen simulator.

Table I. Relative CPU Power Indexes

Worker Location	Mandelbrot Elapsed Time (seconds)	IDXi	LCD Elapsed Time (seconds)	IDXi'
DLA2 (RISC/6000)	598.8	7.04	907.57	7.8
SGRAPH1 (RISC/6000)	623.85	6.77	887.03	7.9
VISLAB-GW (RISC/6000)	510.96	8.25	1128.50	6.2
SGRAPH2(SG40)	1165.62	3.60	2257.13	3.1
STELLAR (Stardent2000)	4216.76	1.00	7043.60	1.0

6.1 THE MANDELBROT SET FRACTAL PROGRAM

The sequential version has the following structure:

```
for i=1 to maxrows
        for j=1 to maxcolls
                z=(0,0)
                for k=1 to max_iter
```

```
                                        x = f(i)
                                        y = g(i)
                                        z = Mandelbrot_Set_Function(z,x,y)
                                        if converges(z)
                                        then        set_screen_color(i,j,k)
                                                    break
                                        else continue
                                        fi
                              rof
                    plot(i,j)
                    rof
          rof
```

This program is parallelizable but not vectorizable due to the control dependency in the innermost loop. The partitioned application system is configured as in Figure 2.

The Master first inserts tuples into a tuple space (Coordinates). Concurrently, K worker programs waiting at K free CPUs, compete for raw coordinates (tuples). A Tuple contains a set of display coordinates. A worker that has acquired a tuple will start computing the Mandelbrot set function using the complex plane coordinates translated from the tuple data. The results are sent to another tuple space (Color_indexes). The Master assembles the results and displays them on an X-window terminal.

The details of the master and worker programs are as follows.

```
Begin Master:
ts1 = cnf_open("coordinates") /* Open a tuple space */
for i=1 to max_tuples
    buffer = pack_tuple(max_tuples)
    stat = cnf_tsput(ts1,buffer,size) /* Insert tuples */
rof
ts2 = cnf_open("color_indexes")
for i=1 to max_tuples
    stat = cnf_tsget(ts2, buffer, size) /* Extract results */
    (i,j,k) = unpack(buffer)
    set_screen_color(i,j,k)
    plot(i,j)
rof
End master.

Begin Worker:
ts1 = cnf_open("coordinates")     /* Open the input space */
ts2 = cnf_open("color_indexes")   /* Open the output space */
stat = cnf_tsget(ts1,buffer,size) /* extract a tuple */
while (not_end_tuple(buffer))
  tuple_size = unpack(buffer)
  for idx=1 to tuple_size
      (i j) = retrieve_ij(idx,tuple_size)
      z = (0,0)
      for k=1 to max_iter
          x = f(i)
          y = g(i)
          z = Mandelbrot_Set_Function(z,x,y)
          if converges(z) then break
                          else continue
          fi
     rof
  rof
  buffer = pack_tuple(i,j,k)
  stat = cnf_tsput(ts2,buffer,size) /* Insert result to output */
  stat = cnf_tsget(ts1,buffer,size) /* Get new tuple from input */
elihw
End worker.
```

In the partitioned programs, the "cnf_ts*" calls are the tuple space manipulation commands. One can easily find a fit for these programs in Figure 2.

According to the analysis in Section 5 and Table I, we can calculate the optimal d:

$$d = 1200 / (1 + 2.28 + 1.87 + 1.95) = 169$$

where 1200 is the number of tuples and (1 2.28 1.87 1.95) are obtained from the first four rows of the IDXi column in Table I. Based on our heuristic, the optimal number of tuples should be around 7.1 or 14.2. The optimal elapsed time is 3.8 minutes (Figure 5).

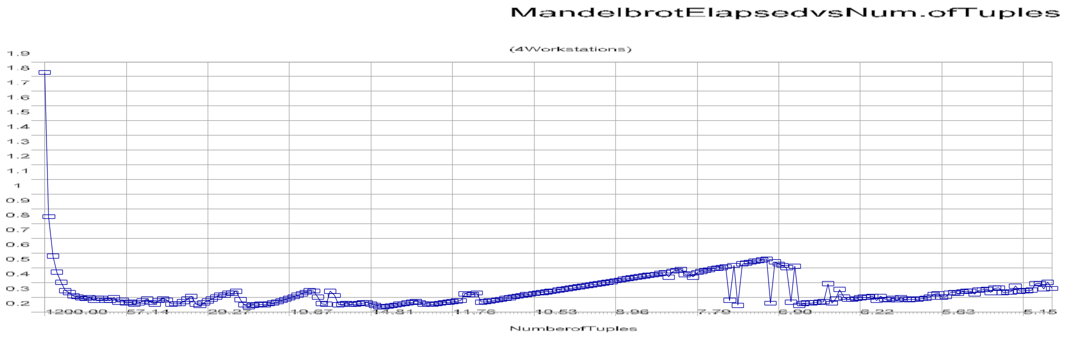

Figure 6 shows MFLOP measurements obtained from experiments on variable tuple sizes. The lower curves are the powers measured at individual computers. The bottom curve represents the power delivered from the RISC 6000 over a different network through a gateway. The top curve is a simple summation of the lower curves. The middle curve with sharp peaks and valleys is the captured distributed power. The optima is found around 14.2 tuples per packet with 25.8 MFLOPS captured performance. Note that the peak of the captured power represents a greater than 71% efficiency ratio when compared to the maximal possible point. It may be interesting to note that a similar program running on a Cray-1 was reported to obtain a 56 MFLOPS performance.

Figure 5. Effects of Varying Grain Sizes for Mandelbrot Display

For more than four workstations, the computation density of this application is not big enough to offset the large communication delays.

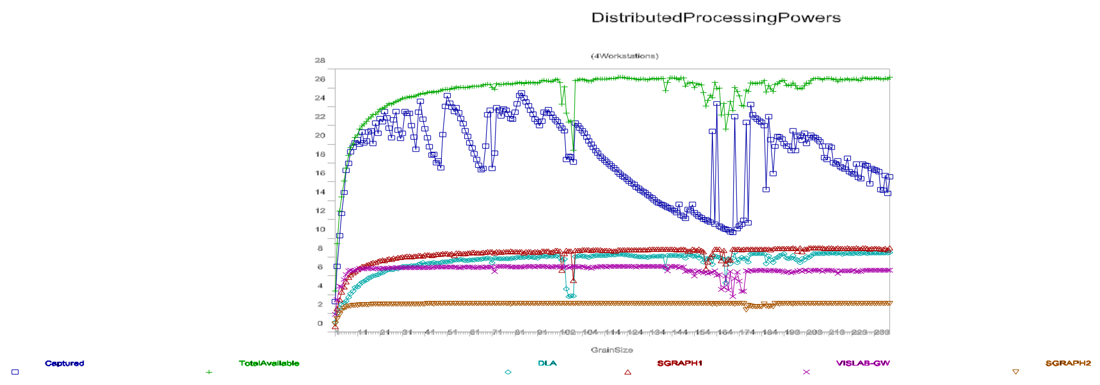

Figure 6. Available and Captured Distributed Processing Powers

6.2 LCD SCREEN SIMULATOR

This is an engineering simulation program. The objective is to simulate the visual effects of a LCD high resolution screen. Every displayed pixel requires solving a differential equation by using the Runge-Kutta method. Its sequential version structure is as follows:

```
input_initialization
for i=1 to max_row
        for j=1 to max_col
                if (i=1) or not similar_val(i,i-1)
                then call R_K(i,j)
                else call copy_col(i-1,j)
                set_color(i,j)
                plot(i,j)
        rof
rof
```

The program tries to save calculation time when the computed value of the first pixel of each column is sufficiently close to the one on previous column. This structure prevents the program being both vectorized and parallelized in a "conventional" parallel compiler. Luckily, we were able to cut the dependency and enhance the quality of the program.

Similar to the Mandelbrot display system, the re-engineering of the program results the following programs:

```
Begin LCD_Master:
tsd = cnf_open("input");
res = cnf_open("results");
```

```
for i=0 to max_col
    for j=0 to max_row
        data_tuple = pack_column_data( i,j )
    rof
    if ((i mod tuple_size) == 0)
    then tuple_name = make_name(i,j)
         cnf_tsput( tsd, tuple_name, data_tuple, size )
         reset_buffer()
    fi
rof
for i=1 to max_col/tuple_size
    cnf_tsget( res, "*", res_tuple, size )
    unpack_results( res_tuple, color, ... )
    for k=1 to tuple_size
        for j=1 to max_row
            set_color( i*tuple_size+k,j )
            plot_pixel( i*tuple_size+k,j )
        rof
   rof
rof
End LCD_Master

Begin LCD_Worker:
tsd = cnf_open("input")
res = cnf_open("results")
there_are_tuples = cnf_tsget( tsd, "*", data_tuple, size)
while ( there_are_tuples )
    extract_data( data_tuple, tuple_size, tuple_name )
    for  i=1 to tuple_size /* cols */
         for j=1 to max_row
             /* if (i=1) or not similar_val(i,i-1) */
               call R_K(name_idx*tuple_size+i,j)
         rof
    rof
    res_tuple = pack_results()
    cnf_tsput(res, tuple_name, res_tuple, size )
    there_are_tuples = cnf_tsget( tsd, "*", data_tuple )
end while
End LCD_Worker
```

Each scattered tuple in this application is a set of screen coordinates and other parameters, such as voltages, gate threshold, etc. A result tuple is a set of screen coordinates along with their color indices.

In this test, the optimal d = 1024/(1 + 2.0 + 2.5 + 2.5) = 128 or equivalently 8 tuples (from the first four positions in the IDX' column in Table I). The optimal elapsed time for four workstations is 4.7 minutes. In this case, the efficiency is more than 98%.

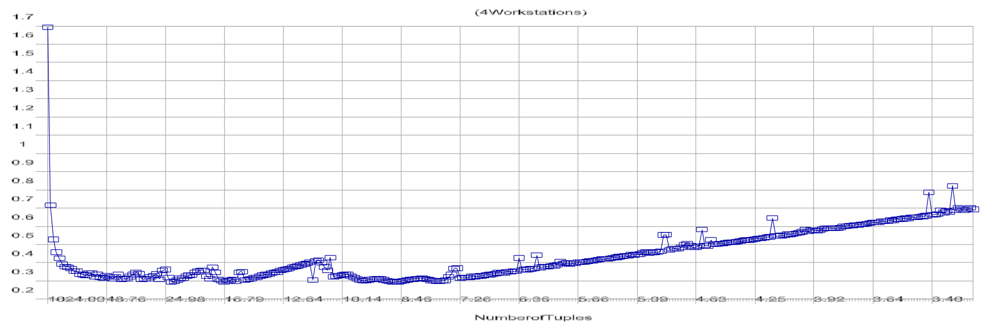

Figure 7. Reduced Elapsed Time vs Grain Sizes for LCD Simulator

6.3 FINANCIAL SIMULATION

This experiment was conducted at the Hermes Financial Simulation Laboratory at the Decision Science Department of the Wharton School of University of Pennsylvania. It requires 1024 Monte Carlo simulations of a single security over 360 time periods. Multiple security simulations can help the financial managers decide the best investment strategies.

The MOSES (Mortgage-backed Security Evaluation System) system was restructured to have the workers computing k simulations where k is the granule size controlled externally to obtain the optimal speed up. Each worker cycle includes two nested loops: k x 360. The innermost loop conducts a Monte Carlo simulation for a given set parameters.

After tuning for the optimal granule size, we have obtained a relative steady 25 second elapsed time using 6 heterogeneous DECstations on the volatile network. The following table shows the performance comparisons with various workstations and supercomputers. The efficiency is 58.9%. The details can be found in [17].

Table III. MOSES Performance Comparisons

Computing Environment	MOSES Elapsed Time	Relative Power

DECstation 2100	155 Seconds	1
DECstation 3100	115 Seconds	1.3
DECstation 5000/200	72 Seconds	2.2
VAX 6400	60 Seconds	2.6
4 DECstations (1x5000,2x3100,1x2100)	30 Seconds	5.2
6 DECstations (1x5000,2x3100,3x2100)	25 Seconds	6.2
Cray X-MP	12 Seconds	12.9
CM-2a (32k Processors)	1.1 Seconds	140.9

7. SUPPORTING SOFTWARE ARCHITECTURE

The supporting software architecture SYNERGY (U.S. patent pending) consists of a two phase development process: a) specification phase and b) execution phase. The specification phase consists of the following activities:

a) Discover the independent internal structures of a given application. This includes finding all candidates for virtual vectorization, pipelining and MIMD type processing.

b) Process programming or program re-engineering using a pre-defined interprocess communication library (IOLIB).

c) Databus network specification using the distributable processes and databuses.

d) Distributed network specification.

e) Compilation of all these specifications to generate runtime control structures expected by the runtime support software.

The runtime support software uses the client/server computing model. It consists of the following elements:

a) Permanent servers on each heterogeneous operating system to perform local process management and IPC port mapping.

b) Dynamic servers for all distributed data object types.

c) A distributed process controller (DPC) that accepts the output of the specification processor and proceeds to negotiate with remote servers for process generation.

The schematic view of the SYNERGY system is shown in Figure 8.

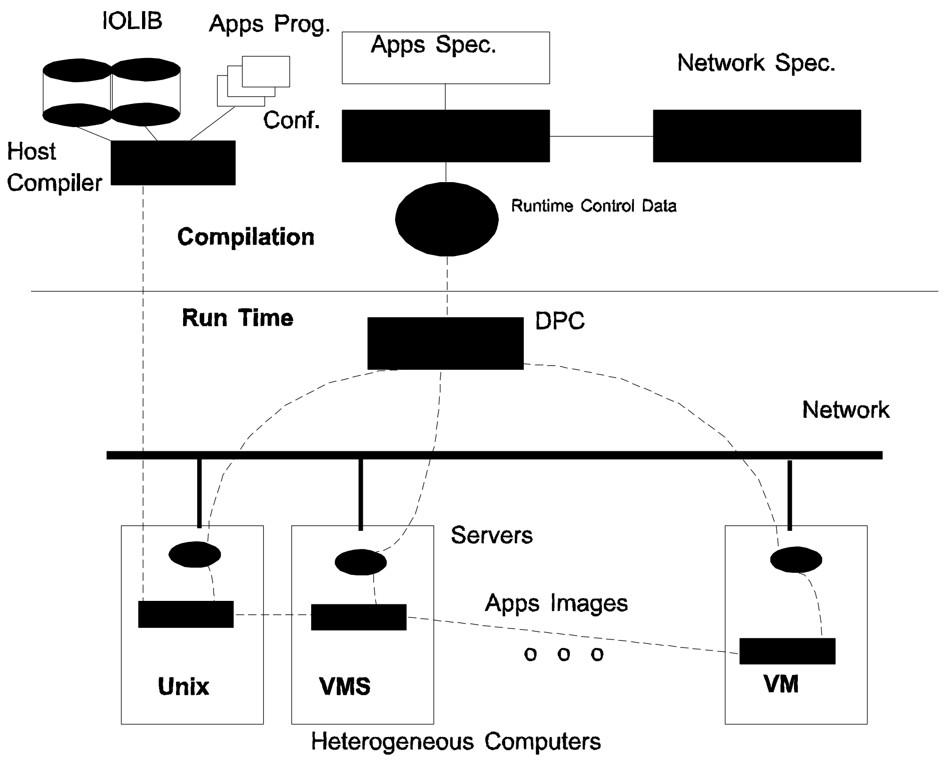

Figure 8. SYNERGY System Architecture

The programming language injection library (IOLIB) is the link between the static user space and the dynamic runtime space. Upon activation of a distributable program, the injected code in the program sets up the program's environment according to initialization data. Since the runtime locations are not part of the compiled code, the programs can be relocated dynamically. The DPC is the controller and runtime interface of the user application. The permanent servers reside on each distributed computer and become part of the local operating system. The dynamic servers are for user defined distributed data objects: tuple space, generic queue and external file. These servers are created before use and killed after mission completion.

Since the generation/activation of runtime support programs are based upon user's specifications, total overhead is minimized according to the application requirements. In general, the larger the application system (data + code), the smaller the overhead percentage.

Table II shows the actual measurements of the overhead in experimental runs.

Table II. Overhead Measurements

	Sequential (original	With SYNERGY Overhead	Percentage

	version)	(partitioned)	
Mandelbrot	507.57	598.80	9.6%
LCD Simulator (Runge-Kutta Reg.)	891	901.5	1.0%
MOSES	72	75	< 1.0 %

8. CONCLUSIONS

The reported computation model allows changes in both application structure and execution environment without affecting the indifferent user codes. This provides a higher level of portability and efficiency for distributed environments than that is currently known [15,9,10,8,7,5,4,3,2,11]. It also offers programmable fault tolerance capabilities that are essential for using any distributed computing resources.

In the reported experiments, we never saturated the networks to the extend that other people could not use the same network. Since we have the control over the tuple sizes, the network loads can be offset by adjusting the tuple sizes to appropriate values. Similarly, major bottlenecks, such as the tuple space servers where many concurrent requests were be made at runtime, are all subject to the control of tuple sizes. The optimal performance of the virtual processor for a given application is in fact the result of using the networks and computers to their most effective degrees.

The drawback of the current SYNERGY system is the necessity of re-engineering existing programs, if the best computation speed is desired. The re-engineering costs are listed as follows:

a) Mandelbrot

100 lines of sequential C code

3 days of re-engineering effort. This includes some X-window programming, debugging and benchmarking.

b) LCD Simulator

1600 lines of sequential C code

14 days of re-engineering effort. This also includes some X-window programming, debugging and benchmarking.

c) Financial Simulation

2500 lines of sequential C code

7 days restructuring effort.

We are now actively researching topics in the following areas:

a) Distributed inference engines using heterogeneous computers over heterogeneous networks.

b) Methodologies for a "Heterogeneous Compiler" that is able to compile a sequential code and produce all candidates for virtual parallel components automatically.

c) A general optimization algorithm for the "Heterogeneous Compiler".

d) Technologies for deploying more efficient network protocols and architectures, such as DQDB for broad band ISDN. Ethernet protocols have proven their limits in dense communication patterns. Multiplexing protocols and network architectures must be evaluated to discover the limits of the SYNERGY technology.

9. ACKNOWLEDGEMENTS

This research is supported in part by a joint study program between the Temple University and the International Business Machines Corporation, the 1990 Temple University Faculty Research Incentive Fund and the Digital Equipment Corporation Fund for Hermes Financial Simulation Laboratory.

REFERENCES

1. S. Ahuja,N. Carriero, and D. Gelernter, "Linda and Friends," IEEE Computer, August 1986, 26-34

2. G.R.Andrew, R.AOlsson, et al. "An Overview of the SR Language and Implementation," ACM TOPLAS, 10,1, January 1988, pp.51-86

3. "NCS User's Guide," Apollo Computer Inc., 1988

4. Robert G. Babb & David C. DiNucci, "Design and Implementation of Parallel Programs with Large-grain Data Flow," THE CHARACTERISTICS OF PARALLEL ALGORITHMS, 1987 MIT Press, Cambridge, 335-349

5. S.P. Barkhordarian, "RAMPS: A Realtime Structured Small-Scale Dataflow System For Parallel Processing," ICPP 1987, 610-614

6. N. Carriero, "Implementation of Tuple Space Machines," Ph.D Dissertation, Department of Computer Science, Yale University, December, 1987

7. D.R. Cheriton, "The V Distributed System," CACM. 31,3, March 1988, pp.314-333

8. C.J.Fleckenstein and D. Hemmendinger, "Using A Global Name Space for Parallel Execution of Unix Tools," CACM, September 1989, 32,9, pp.1085-1090

9. D. Glertner, "Generative Communication in Linda," ACM TOPLAS, 7,1, January 1985, pp.80-112

10. J.K.Ousterhout,A.R.Cherenson,F.Douglis, M.N.Nelson &B.B.Welch, "The Sprite Network Operating System," IEEE Computer, January 1988, pp.23-36

11. C. D. Polychronopoulos, "Parallel Programming and Compilers," Kluwer Academic Publishers, 1988, ISBN: 0-89838-
288-2, QA76.6.p653 1988

12. "Very-High-Level Concurrent Programming", IEEE Transactions on Software Engineering, with N. Prywes, B.Szymanski and A. Pnueli, Vol. 13, No. 9, September 1987, pp.1038-1046.

13. "A CASE Tool Set for Constructing Distributed Heterogeneous Applications," with Avi Freedman, Proceedings of the 1989 3rd International Workshop on Computer Aided Software Engineering, Imperial College, London, July 17-21, 1989

14. "Developing Computation Intensive Applications Using Networked Workstations," with Kostas Blathras, Poster Presentation, Supercomputing'89, Reno, Nevada, Nov. 13-17, 1989

15. A. Tanenbaum & R.V. Renesse, "Distributed Operating Systems," Computing Surveys, Vol. 17, No.4, December 1985

16. R. A. Whiteside & J. Leichter, "Using Linda for Super-computing on a Local Area Network," Proceedings of Supercomputing'88, Orlando, FL, October 1988, 192-199

17. Y. Shi & Z. Stavros, "Distributed Mortgage-backed Security Simulation," Technical Report, 1-2-1991, Decision Science Department, The Wharton School, University of Pennsylvania, PA, 1991.