# The PRESTO Application Suite

Radhika Thekkath and Susan J. Eggers
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195
radhika@cs.washington.edu

Technical Report # 94-09-01

September 9, 1994

## 1 Introduction

This report describes a group of coarse- and medium-grain, explicitly-parallel applications that have been made available via the World-Wide Web at (http://www.cs.washington.edu/research/projects/parsw/Benchmarks/Presto/www/index.html). These programs have been written using the PRESTO user-level threads library [4], by students at the University of Washington and Rice University. PRESTO provides a $C^{++}$-based environment for writing object-oriented parallel programs for shared-memory multiprocessors. The library provides basic classes useful for writing parallel programs, among them are thread manipulation routines for concurrency and synchronization primitives. Details of this programming environment and examples are in the PRESTO user's manual [5]. The PRESTO manual and sources can be obtained via anonymous FTP (ftp://cs.washington.edu/pub/presto1.0.tar.Z).

The applications in this suite have been written for the Sequent Symmetry [10] and calls into the PRESTO library port to the Sequent. PRESTO's thread manipulation calls, such as thread creation, deletion, etc., are fairly standard, and are available in thread packages available on other machines. These programs are therefore easily portable to other architectures by replacing the PRESTO threads calls to those of another library.

The next section describes each application briefly. The applications from Rice University were developed as part of a class project, and made available here, thanks to Prof. John Bennett. Some of these programs are short and do fairly obvious computations, like matrix multiply. Hence many details are omitted, and explanation is provided only when appropriate.

## 2 The Applications

Table 1 lists the applications described in this report. The table also shows the number of lines of source code, and a broad classification of each program in the application domain.

Several research studies have used subsets of these applications [3, 11, 12], mainly in simulation work. These publications provide a detailed analysis of the characteristics of some of the applications.

| Program Name | Lines of C++ Source | Application Domain |
|---|---|---|
| Grav | 1013 | Scientific |
| Patch | 2746 | Graphics |
| Pdsa | 3952 | CAD |
| Vandermonde | 319 | Mathematics |
| Health | 511 | Simulation |
| FullConn | 600 | Simulation |
| fft | 867 | Mathematics |
| gauss | 543 | Mathematics |
| tp | 648 | Graph |
| knap | 589 | List Processing |
| life | 504 | Mathematics |
| merge | 1087 | List Processing |
| mst | 1127 | Graph |
| qsort | 474 | Sorting |
| shellsort | 741 | Sorting |
| sieve | 430 | Mathematics |
| sor | 843 | Mathematics |
| sparse | 578 | Matrix |
| string | 711 | List Processing |
| mult | 510 | Mathematics |
| factorial | 1355 | Mathematics |

Table 1: The workload

## 2.1 Grav

**Author:** Ed Felten.

Grav is an implementation of the Barnes and Hut Clustering algorithm [2] for simulating the gravitational intersection of a large number of stars over time.

The inputs to the program are: the number of stars, the name of the file describing the stars and the number of time steps to run the algorithm. Each star in the file is described by a real value for the mass, a vector (x, y and z values) for the position, an integer id, and a vector for the velocity. The files "stars.50" and "stars.100" provide sample input files for 50 and 100 stars respectively. Compiling and executing the C file "generate.c" provides a way to produce bigger input star systems.

Each time step of the algorithm comprises of 4 stages.

**build:** Build the node tree by adding particles to it.

**collect:** Add up the total mass in this region and calculate the center of mass.

**calc:** Calculate acceleration.

**push:** Update positions and velocity on all stars.

The number of processors and number of threads are constant declarations in "grav.c" and can be easily changed. The limit on the maximum number of stars or particles (256) can be changed by editing the file "grav.h"

## 2.2 Patch

**Author:** Denise Draper.

Patch is a graphics application that performs the first phase of a generalized B-spline construction based on S-patches [9].

Given a mesh, the program constructs a collection of Bezier control points corresponding to control mesh vertices. This is done by first calculating the position and tangent control data, followed by the calculation of the Bezier coordinates. The algorithm employs symmetry by orienting the calculations to each corner of the triangle.

The program takes as input the name of a file that specifies the mesh. The mesh is described by a set of points and a description of the faces of the surface. The program is partitioned so that a single thread works on 4 faces. This partitioning can be varied by editing the file "foofle.c" .

## 2.3 Pdsa

**Author:** unknown.

Pdsa does automatic placement of integrated circuit layouts using a simulated annealing algorithm [13].

This is a special algorithm which permits the circuit to be a combination of macro blocks and standard cells. It works in essentially three steps: first, standard cells are partitioned into flexible virtual blocks using a minimum net-cut criteria; second, the macro blocks are then placed using a simulated annealing optimization that uses routing area costs and net costs. And finally, once the relative positions of the macro blocks is known, the standard cell blocks are placed using a simple annealing optimization.

The inputs to the application must specify the number of processors and the number of major and minor threads. The minor threads are responsible for computing the cost functions, while the major threads do the bulk of the remaining work. An example input file "g2.yal" is provided with the application.

### 2.4 Vandermonde

**Author:** Rich Nieves.

Vandermonde does matrix computations given two input arrays. The work is partitioned so that each thread works on a fixed number of columns in the array. The total work proceeds in two phases with a barrier synchronization of all threads between phases. This two-phase computation is repeated 1000 times by default. The user can override this value at program run time. The user must also specify the input arrays and the number of threads that must be spawned to do the required work. Several input data files are available with varying array sizes.

### 2.5 Health

**Author:** David Wagner.

Health and FullConn (described next) are applications that use the Synapse runtime environment [14]. Synapse is a customized PRESTO environment for conservative parallel simulation. Like PRESTO, it is also implemented in C++, and provides several classes to facilitate writing parallel simulations. It provides virtual time semantics, three deadlock handling mechanisms, and options for performance tuning. The Synapse source files are available, and are used to build the Synapse library.

Health is an application that simulates the Columbian health care delivery system presented by Lomov, Cleary, Unger, and West [8]. They simulated it using the optimistic approach, and conjectured that a conservative simulation would be difficult. The Health problem may be described as follows, the Columbian government provides health care in a multi-tiered fashion of services and referrals. Each level in the system is capable of providing some services, while it must refer other problems to the next higher level. Each village generates a stream of patients to its local health center. Each center has one or more health care providers (HCPs). Each patient arriving at the center, is either served immediately, or waits in a queue until a HCP is available. Each patient's problem is diagnosed, and either the service is provided locally, or the patient is referred to the parent health care center.

The simulation assumes that the health care system is organized as a full, four-way branching tree of some height. The number of HCPs at a center depends on its height in the hierarchy. The inputs to the application must specify the maximum level of the tree structures and the time limit for the simulation. Each village generates patients according to a Poisson process with arrival rate 0.3 patients per time unit. The probability that a patient will be treated at a health care center is 0.9, with the exception that all patients are treatable at the root.

### 2.6 FullConn

**Author:** David Wagner. (Modified by Reid Brown).

FullConn is the second Synapse application provided. Its goal is to achieve a conservative parallel simulation of a multiprocessor system. Such a system consists of a set of processor nodes, which communicate with one another at seemingly random intervals, when the executing parallel program needs to access shared data on another processor.

Hence, this application simulates a fully-connected mesh network of processor nodes. Events in the parallel simulation are fixed-size messages which are kept circulating randomly among the nodes. These messages correspond in the original system to messages that fetch and send shared data between processors. The number of arrivals per processor per time step is specified as input, and each processor maintains input and output FCFS queues. Each node has a service time requirement for the processing of messages; this is the time that a processor would spend computing before it is ready to process the receive or send the next message.

The inputs to the program are: the number of initial arrivals at a node and the stopcount, which is the duration of time after which the messages stop circulating.

### 2.7 FFT

**Author:** Rice University.

The application fft does Fast Fourier Transform computations. In the first phase, it generates sample points of the array X[] according to the size N specified as input. It then evaluates the 'W' term of the FFT algorithm once, holding the data in the W[] array. The data from X[] is then swapped to the R[] array in preparation for the iterative parallel FFT algorithm.

The size of the array X[] can be specified as an input parameter, and must be a power of 2; the default is 1024.

### 2.8 gauss

**Author:** Rice University.

Gauss does gaussian elimination using the pivot algorithm. The input to the program must specify the dimension of the input matrix, as well the parallel partitioning scheme. The three partitioning possibilities are: each thread works on a separate column, each thread works on a group of consecutive columns, or each thread works on a group of columns, where each column in a group is separated by a fixed distance (interleaved).

### 2.9 tp

**Author:** Rice University.

The tp program solves the traveling salesman problem. This program takes a set of cities and their coordinates as input, and finds the shortest path that goes through all the cities.

### 2.10 knap

**Author:** Rice University.

The knap program solves the integer knapsack problem. It takes two integer inputs, the first provides the vector elements of the knapsack, and the second provides the random seed value. The final required knapsack value 'M' can also be specified as part of the input.

## 2.11 life

**Author:** Rice University.

The life application plays the game of Life. It uses a default 16 x 16 size board. Worker threads divide up the number of rows on the board equally among themselves. The game uses two phases, with barrier synchronization between them. The first phase calculates new values into the scratch board, and the second phase stores these new values back into the original board.

## 2.12 merge

**Author:** Rice University.

The merge program merges two sorted lists of N numbers into a single sorted list. The parallel algorithm works by creating twice as many threads as the number of processors, and dividing each list up equally into sublists among the threads. The separation points in each list are the "break" points. The algorithm then finds the insertion point in the other list, corresponding to each of the break points. The break and insertion points are then combined to obtain the starting indices to the independent merges of the sublists. The current size limit of each list is $2^{32} - 1$ numbers.

## 2.13 mst

**Author:** Rice University.

The mst application finds the minimum spanning tree for a given input connected graph using Prim's algorithm [1]. If $V$ is the set of vertices of the graph, the algorithm begins with a set $U$ initialized to 1. And at each iteration, it finds the lowest cost edge $(u, v)$ that connects $U$ to $V - U$ and adds $v \in V - U$ to $U$. This loop is repeated until $U = V$.

Sample input graphs may be built using the source files in the buildGraph directory.

## 2.14 qsort

**Author:** Rice University.

The qsort application implements a parallel quicksort algorithm by forking a thread for each array partition in a recursive fashion. The algorithm works on an integer array and generates the array 1...numelements, given the number of elements in the array ( numelements ) as input.

## 2.15 sieve

**Author:** Rice University.

The sieve application implements the Sieve of Erastosthenes algorithm. It calculates all prime numbers less than some given limit. The maximum allowable limit is $10^7$. The algorithm works by a process of elimination. It removes, in parallel, all multiples of some integer $i$, where $i$ could be a prime. This is repeated for all numbers that are less than the truncated square root of the given limit. Note that this upper bound is safe, since there can be at most one remaining multiple less than the limit, which is the square of this upper bound. All other multiples must be greater than or equal to the limit.

### 2.16 sor

**Author:** Rice University.

The sor program implements the successive over-relaxation algorithm. The program takes the size of the matrix as input, and begins by initializing the boundary values to 1. It then computes the values inside the matrix with the given boundary conditions. If the number of iterations is not specified, it uses a default tolerance of 1% as the terminating condition, i.e., when the maximum difference between the new and old values is less than 1%.

### 2.17 sparse

**Author:** Rice University.

The sparse application implements an equation solver that gets its values from a sparse matrix. The inputs to the program specify the sparsity value, the total matrix dimension and the seed. Using these inputs, the program first automatically generates a sparse integer matrix, and set up the Equation object. This object is a set of coefficient values and a set of participating elements in the equation. A list is used to identify the dependency of the equation on remote objects. The solution to the equation can be obtained when all remote data is available locally. This movement of data is achieved using a set of message passing protocols implemented for this purpose.

### 2.18 string

**Author:** Rice University.

The string program finds the k-difference in a string of patterns to a string of given text. Each of the k differences represent modifications that would be required in the text in order to make an exact match with the pattern. The modifications may be, for example, changing a character, inserting a character, or deleting a character. The algorithm uses dynamic programming to compute successive values. The parallel implementation does not use synchronization as some values may be computed more than once.

### 2.19 mult

**Author:** Rice University.

The mult program does matrix multiply. It takes the matrix size as input and generates two identical matrices which are then multiplied. Each thread is responsible for generating one row of the result matrix.

### 2.20 factorial

**Author:** Rice University.

The factorial application computes the factorial of a given number. The size limitation on the generated factorial value is machine dependent. This parallel version of factorial works by dividing up the work, i.e., the multiplication of the sequence of numbers, among the different threads. Lastly, a single thread accumulates the individual products into the final factorial product.

## 3 Pointers to Other Application Sources

Finally, we include pointers to other application sources[1] available in the public domain, that are known to us. The SPLASH applications are available via anonymous ftp from Stanford (mojave.stanford.edu). The original Split-C version of the EM3D application is available from Berkeley (http://http.cs.berkeley.edu/public/parallel/software.html). Applications from their recent publications [6, 7] are also available via anonymous ftp from the University of Wisconsin (ftp.cs.wisc.edu/pub/WWT).

## Acknowledgements

We thank all the authors who let us use their applications for our studies, and who have now graciously given us permission to make these sources available to other researchers.

## DISCLAIMER

These PRESTO application sources are distributed free of charge without any warranty. These applications were part of a research effort of the Computer Science departments at the University of Washington and Rice University. Neither the departments nor the authors make any claim beyond this. In particular, no liability is assumed.

## References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addision-Wesley, 1983.

[2] J. Barnes and P. Hut. A hierarchical o(nlogn) force-calculation algorithm. *Nature 24*, pages 446–449, 1986.

[3] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Adaptive software cache management for distributed shared memory architectures. *17th Annual International Symposium on Computer Architecture*, pages 125–134, May 1990.

[4] B. N. Bershad, E. D. Lazowska, and H. M. Levy. PRESTO: A system for object-oriented parallel programming. *Software: Practice and Experience*, 18(8):713–732, August 1988.

[5] Brian N. Bershad. The PRESTO user's manual. Technical Report TR. No. 88-01-04, University of Washington, January 1988.

[6] S. Chandra, J. R. Larus, and A. Rogers. Where is time spent in message passing and shared memory programs? *To be published in Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

[1] The authors of this report do not consider themselves in any way responsible for the availability or location of these sources.

[7] J. R. Larus, B. Richards, and G. Viswanathan. LCM: memory system support for parallel language implementation. *To be published in Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

[8] G. Lomov, J. Cleary, B. Unger, and D. West. A performance study of time warp. *In Distributed Simulation*, July 1988.

[9] Charles T. Loop. Generalized b-spline surfaces of arbitrary topological type. Technical Report TR. No. 92-10-01 (Ph.D. thesis), University of Washington, October 1992.

[10] Symmetry Technical Summary. Sequent Computer Systems, Inc.

[11] R. Thekkath and S. J. Eggers. The effectiveness of multithreaded architectures. *To be published in Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

[12] R. Thekkath and S. J. Eggers. Impact of sharing-based thread placement on multithreaded architectures. *21th Annual International Symposium on Computer Architecture*, pages 176–186, April 1994.

[13] M. Upton, K. Samii, and S. Sugiyama. Integrated placement for mixed standard cell and macro-cell designs. *Proceedings of the 27th Design Automation Conference*, pages 32–35, June 1990.

[14] D. B. Wagner. *Conservative Parallel Discrete-Event Simulation: Principles and Practice.* Ph.D. thesis, University of Washington, September 1989.