Super Computing Applications in the CSIRO-DIT High Performance Computation Project

D. Abramson, M. Dix†, R. Francis, I. Mathieson, M. Rawling and P. Whiting

CSIRO.Division of Information Technology
55 Barry St, Carlton, 3053, Australia

†CSIRO Division of Atmospheric Research
Private Bag 1, Aspendale, 3195

1 Introduction

In April 1990 the CSIRO Division of Information Technology created the High Performance Computation Project. The primary thrust of the Project is to determine how to reduce the cost of solving computationally intensive problems which are in the national interest. The objectives to be pursued while solving such problems include learning how best to program commercially available parallel computers as well as understanding design and construction issues so that special purpose computer architectures can be built to solve specific problems where appropriate.

The High Performance Computation Project grew out of the Parallel Systems Architecture Project (PSAP), a three year collaborative effort between CSIRO and RMIT. The original scope of the PSAP was to investigate the dataflow model of computation [1]. This lead to the construction of a new dataflow model of parallel computation [2]; a number of compilers for dataflow-like languages and a number of application studies [3]. The new architecture was embodied in a machine called CSIRAC II. Most of the dataflow related research has moved to Swinburne Institute of Technology under the direction of Professor Egan.

The new research being undertaken in the High Performance Computation Project is *application driven*. The purpose of this style of research is two fold. First, it has the potential to directly produce products and services which can contribute to the wealth of Australia's information industries. Second, it provides a sound base for the research direction, which allows a knowledge base to be built around the science of concurrent computing. Many of the projects described in this paper are multi-disciplinary, and involve experts from diverse fields of science and engineering. Thus while the techniques of high performance computation are interesting in their own right they also represent an enabling technology for advances in many fields.

The Project has access to a range of parallel and high speed computers located in Australia including an Encore 20 CPU shared memory multiprocessor, a network of SUN workstations, a Transputer development system, a MasPar 1024 processor SIMD and a Cray Y-MP. The range of platforms allows us to gain experience and carry out comparative analyses of applications across different architecture classes and programming paradigms.

The remainder of this paper introduces the problems currently under active investigation. These include: Biomolecular Computing, Parallel Discrete Event Simulation, Climate Modelling and Scheduling Architectures. Each section gives a brief overview of the area and summarises results or analysis completed to date. In some cases, more detailed technical reports are available.

2 Biomolecular Computing

Traditionally, drugs have been created either by extracting chemicals from natural sources, or by directly manufacturing the chemical in the laboratory. The structure of such drugs are determined by classical techniques of applied chemistry, such as x-ray diffraction and mass spectrometry. Once created, the drugs are subjected to clinical trials to determine their effectiveness. More recently, biochemists have been *designing* drugs by computer. Drug

molecules are synthesised by modelling the structure of the chemical, which can then be manipulated to create a drug with the desired properties. Such modelling takes account of the forces within and between the molecule as well as the known structure of existing molecules, and therefore requires an extremely large processor resource.

This work is being conducted in collaboration with the CSIRO Division of Biomolecular Engineering, who are actively involved in designing anti-viral drugs. An existing, and relatively stable, software package is being analysed to determine whether a cheaper solution can be found, either by building a special purpose processor, or by implementing the package on a commercial multiprocessor.

2.1 Molecular Dynamics using X-PLOR

Molecular dynamics (MD) is concerned with calculating the physical structure and other properties of complex chemical systems incorporating from hundreds to thousands of atoms. X-PLOR is a program for carrying out such calculations, and as it states in the manual:

> *"X-PLOR is a general-purpose macromolecular refinement program that uses crystallographic diffraction data or nuclear magnetic resonance interproton or other internuclear distance data in combination with energy minimization or molecular dynamics. The program allows one to use simulated annealing to overcome local energy minima."* [6] (p. 13).

For large molecules (of the order of 10, 000 atoms) this is a very time consuming problem: a sample run of 100 iterations with 4000 atoms takes 1000 seconds on the JSF Cray Y-MP. A major factor in this is the inclusion of crystallographic data from X-ray diffraction studies, which invokes a 3-D FFT during each energy refinement step. Without this X-ray energy term, the 100 iteration run takes 80 seconds, but the results are much less accurate.

In X-PLOR, MD is used to model the behaviour of each of the component atoms or multi-atom residues making up a macromolecule over a brief time period (using discrete steps of the order of 10^{-15} seconds). During each time step, the effects of each atom or residue on its neighbours is calculated, resulting in changes to the potential and kinetic energy of the entire system. The potential energy calculation includes components from each covalent bond, the angle between two such bonds and the torsion between a sequence of three bonds, plus non-covalent effects such as van der Waal's attraction, electrostatic potential and hydrogen bonding. X-ray diffraction and nuclear magnetic resonance studies (when available) provide effective constraints to these empirical energy calculations.

The inclusion of X-ray diffraction data is the most time consuming step, due to the large 3-D FFTs involved. However, the next most time consuming component includes the calculation of electrostatic effects. Since this force is more far reaching than any of the other non-covalent forces, a larger enclosing volume must be searched to find influential neighbours during each iteration, and the larger volume may result in finding more neighbours, further increasing the calculation load.

2.2 Profiling Results

X-PLOR has been profiled on a number of machines in order to understand the effects of vectorisation and floating point hardware on its performance. The machines include the JSF Cray Y-MP, a SPARC based Sun 4/65 with a Weitek floating point coprocessor, and an Encore Multimax (NS32332 processor, with no floating point acceleration).

Two benchmark data sets were provided with the X-PLOR software: one (DNA) performs a 500 cycle molecular dynamics run on two segments of DNA (comprising 700 atoms in all); the other (MAAT) performs a single energy minimisation step on a 4000 atom bacterial enzyme. A more realistic workload (N41) performs larger MD runs (1000 cycles) on an antibody fragment

of 4000 atoms complete with diffraction data. Sequential times (in seconds) for these benchmarks are given in Figure 1.

Machine	DNA	MAAT	N41 (xray)	N41(no xray)
Vax 8800	2544	611	---	---
Cray X-MP (COS)	65	38	---	---
Cray Y-MP (UNICOS)	53	23	950	79
Convex C-1	725	282	---	---
SPARC 4/65 (SunOS)	1973	503	> 24hr.	1866
Encore Multimax (UMAX V)	22627	4452	---	---

Fig 1 Benchmark performance results for X-PLOR.

Profiling on both the JSF Cray Y/MP (using perftrace) and a Sun 4/65 workstation (SPARCstation 1+) with floating point but not vector support (using gprof [13]), confirmed that the FFT code running over the X-ray diffraction data dominated execution time. When the X-ray energy term was eliminated, the non-bonded energy calculations dominated on both architectures, since the code vectorised poorly. Time was spent in two major routines: one searching for non-bonded interactions (XPSEA2), and the other calculating the actual energies (ENBSF).

2.3 Parallelisation on the Encore Multimax

Since the ENBSF routine was so dominant in the earlier runs, it was chosen as the first target for parallelisation using shared memory. ENBSF calculates the electrostatic and van der Waal's energy terms, plus the change in position, over all atoms. The strategy adopted for parallelisation required the addition of a FORTRAN interface to shared memory; parallelism operators such as fork and join; an additional X-PLOR symbol (NCPU) to specify the degree of parallelism required; and barrier and spinlock synchronisation mechanisms. As part of the implementation, the original sequential code in ENBSF was pushed down into a subroutine called when NCPU was less than 2. Thus the only cost in sequential execution of the code is a test on NCPU and the duplicate procedure call with its 25 parameters.

For parallel operation, the new ENBSF routine allocated blocks from the shared heap for shared locks and counters and to emulate shared common, which was not readily available. ENBSF then spawned NCPU subprocesses (via the UNIX fork call) which called ENBSF2, a parallel version of the original routine. These multiple copies of ENBSF2 could only return results to ENBSF via parameters held in shared memory, although each had full access to a non-shared copy of the data held by ENBSF. The multiple copies used self-scheduling via a shared loop counter to collectively iterate over all atoms. Shared data accesses were protected via spinlocks where necessary to prevent errors. When the shared loop counter was exhausted, the ENBSF2 routines returned and exited, and after all had done so, the parent ENBSF routine updated its common from shared memory, and disposed of the shared objects.

One problem with this approach is that although the parallelism is introduced at the right level in the code (with minimal code modification and preventing multiple logging reports), the overhead of the UNIX fork/exit/wait operations is excessive, as shown in Figure 2 for a 50 cycle run using the DNA dataset without the X-ray term. It can be seen that the forking time dominates the runtime overheads. These results show that this code is amenable to parallel execution however to achieve maximum performance other major routines need similar modification.

of 4000 atoms complete with diffraction data. Sequential times (in seconds) for these benchmarks are given in Figure 1.

Machine	DNA	MAAT	N41 (xray)	N41(no xray)
Vax 8800	2544	611	---	---
Cray X-MP (COS)	65	38	---	---
Cray Y-MP (UNICOS)	53	23	950	79
Convex C-1	725	282	---	---
SPARC 4/65 (SunOS)	1973	503	> 24hr.	1866
Encore Multimax (UMAX V)	22627	4452	---	---

Fig 1 Benchmark performance results for X-PLOR.

Profiling on both the JSF Cray Y/MP (using perftrace) and a Sun 4/65 workstation (SPARCstation 1+) with floating point but not vector support (using gprof [13]), confirmed that the FFT code running over the X-ray diffraction data dominated execution time. When the X-ray energy term was eliminated, the non-bonded energy calculations dominated on both architectures, since the code vectorised poorly. Time was spent in two major routines: one searching for non-bonded interactions (XPSEA2), and the other calculating the actual energies (ENBSF).

2.3 Parallelisation on the Encore Multimax

Since the ENBSF routine was so dominant in the earlier runs, it was chosen as the first target for parallelisation using shared memory. ENBSF calculates the electrostatic and van der Waal's energy terms, plus the change in position, over all atoms. The strategy adopted for parallelisation required the addition of a FORTRAN interface to shared memory; parallelism operators such as fork and join; an additional X-PLOR symbol (NCPU) to specify the degree of parallelism required; and barrier and spinlock synchronisation mechanisms. As part of the implementation, the original sequential code in ENBSF was pushed down into a subroutine called when NCPU was less than 2. Thus the only cost in sequential execution of the code is a test on NCPU and the duplicate procedure call with its 25 parameters.

For parallel operation, the new ENBSF routine allocated blocks from the shared heap for shared locks and counters and to emulate shared common, which was not readily available. ENBSF then spawned NCPU subprocesses (via the UNIX fork call) which called ENBSF2, a parallel version of the original routine. These multiple copies of ENBSF2 could only return results to ENBSF via parameters held in shared memory, although each had full access to a non-shared copy of the data held by ENBSF. The multiple copies used self-scheduling via a shared loop counter to collectively iterate over all atoms. Shared data accesses were protected via spinlocks where necessary to prevent errors. When the shared loop counter was exhausted, the ENBSF2 routines returned and exited, and after all had done so, the parent ENBSF routine updated its common from shared memory, and disposed of the shared objects.

One problem with this approach is that although the parallelism is introduced at the right level in the code (with minimal code modification and preventing multiple logging reports), the overhead of the UNIX fork/exit/wait operations is excessive, as shown in Figure 2 for a 50 cycle run using the DNA dataset without the X-ray term. It can be seen that the forking time dominates the runtime overheads. These results show that this code is amenable to parallel execution however to achieve maximum performance other major routines need similar modification.

Another problem (on the Encore) is the need to copy common blocks in and out of shared memory. While it would be possible to do fewer copies if forking occurred higher in the call tree such an implementation would require far more major modifications to the source code.Hence to obtain maximum performance while constraining code modification, a solution which allows forking at the lowest level without forcing data copying is required. Lightweight process packages, such as SCHEDULE by Dongarra and Sorenson [11], are being investigated.

No. of cpus	Elapsed time	No. of forks	Forking only	Forking + copies	No forks or copies	Speedup (no forks)
1	2534	0	---	---	---	1.00 (1.00)
5	565	520	53	56	509	4.48 (4.95)
10	366	1040	105	108	258	6.92 (9.71)
15	340	1560	157	160	180	7.45 (13.85)

Fig 2 Speedup results for the routine ENBSF from X-PLOR

3 Parallel Discrete Event Simulation

In an effort to reduce the time to create a new product, most computer systems are subjected to substantial simulation before they are built. This simulation allows engineers to design circuits and then exercise them with test cases before fabrication. Substantial savings both in fabrication costs and circuit specific test construction can be made in this way. Clearly, as the systems to be simulated become larger, the time taken to perform the simulation becomes critical.

This work is being conducted in collaboration with Austek Microsystems, who rely on simulation to test their micro circuits. The existing Austek simulation kernel is being examined to determine whether a parallel version can be implemented. It is envisaged that the parallel kernel will run on a commercial shared memory multiprocessor, which may include an in-house multiprocessor being developed by Austek.

3.1 Techniques for Parallelisation of Discrete Event Simulation

Simple Discrete Event Simulation - Centralised Time

The simplest approach to parallel discrete event simulation uses a global time clock together with a single global event queue. The simulation is made up of several connected node instances that model the process or circuit being simulated. These nodes receive signal events on their inputs and generate new (future) events on their outputs as they are evaluated. All events are kept in a strict time order on the event queue. An important optimisation is available in this type of simulation in that if an output does not change there is no need to post an event for it, this is not necessarily true for other simulation techniques. Concurrency is achieved by processing all events scheduled for the current time in parallel, however, in real simulations of integrated circuits, very few events are scheduled for exactly the same time and thus the effective concurrency can be quite low [5, 17].

The Chandy-Misra Algorithm - Distributed Time

One way to achieve higher concurrency is to process events scheduled for different times in parallel. The Chandy-Misra algorithm does this by replacing the global clock of the previous method with distributed time so that each node instance in the simulation proceeds independently at its own time [8]. The event queue can also be distributed as events are typically sent as time

stamped messages directly to the instances on which they are scheduled, to be queued at that point. As the simulation proceeds instances keep track of their own simulated time, which is the time of their most recent evaluation or dispatch. A dispatch occurs whenever the current time of an instance is less than the earliest time at which each of its inputs is valid. This condition is tested on the arrival of each new event and as each dispatch occurs. Note that it is possible for a single event to lead to several dispatches if all other inputs have multiple events queued on them and the current event is scheduled for a later time than all of these. Of course, each dispatch of an instance will generally lead to new events being generated.

In order to proceed when an output does not change, instances must send null events to their successors, thus further validating the current signal values on their inputs. As long as there are no 'time holes' the simulation is guaranteed to run to completion. In practice however, null events are usually not sent for efficiency reasons and the simulation deadlocks as nodes can not validate all of their inputs. At this stage a deadlock resolution phase is entered whereby all invalid node inputs are validated up until the time of the earliest event on any node. The simulation proceeds by oscillating between the active event driven simulation and deadlock resolution phases.

The key to a successful implementation of the Chandy-Misra algorithm is the minimisation of the number of deadlocks that occur, through lookahead and other methods. In addition, the deadlock resolution phase which typically accounts for 40% to 60% of total execution time needs careful optimisation [12].

The Time Warp Operating System - Virtual Time

Another approach currently gaining in popularity is optimistic simulation based on the concept of virtual time or time warping [15]. As in the Chandy-Misra algorithm the simulation is based on node instances that are locally self-contained and which communicate via time stamped event messages. However no attempt is made to maintain time synchronisation either through global management or input validation. Nodes simply process events as they arrive and treat all other inputs as though they were valid up until that time. The result is that one node may be working far ahead of another in simulation time. With this method it is possible that a node could receive a message that is scheduled for its simulation past, indicating that it had proceeded too far into the future. To correct the simulation the node must 'roll back' to the scheduled time of the new event and resume processing from that time. When a node rolls back it must also cancel any work it did in the meantime by sending cancellation events for each event it had posted. These cancellations may lead to secondary roll-backs and further cancellations until the simulation again stabilises. It can be shown that a well structured simulation (eg, one that has no infinite loops) is guaranteed to make progress and experimental evidence shows that cascading roll-backs are not a problem in many practical simulators although this has not been shown for large scale VLSI logic simulations.

3.2 Performance Results

The Austek simulation kernel has been modified to provide detailed statistics as simulations proceed. For example, in order to determine the likely effectiveness of the simple centralised time scheme, an analysis similar to that shown in figure 3 can be carried out.

The graph in figure 3 shows the number of events that could have been processed in parallel at each time step during part of the simulation of a cache controller chip. Points to note are the burstiness of the event activity and the appearance of long periods during which there are very few events scheduled for the current time.

The Austek simulator preprocesses all events at a given time before any node instances are dispatched so as to prevent multiple dispatches of an instance in the same time step. This would

otherwise occur where independent events were scheduled on each input of a two input node for the same time. Due to this preprocessing and the fact that events themselves have fan-out (ie, events can be sampled by more than one node instance), it is usual for the number of dispatched instances to be different from the number of scheduled events at each time step. Figure 4 shows the number of dispatched instances in the cache simulation over the same period of time as covered by figure 3. As can be seen, what little concurrency there was in figure 3 has been further reduced by the time instance dispatching occurs. This has proven to be the rule rather than the exception for this simulator.

Figure 5 shows the total number of events scheduled and time slots occupied at each time step. This includes all events and slots in the future as well as events in the current time slot. This places an upper bound on the concurrency that could be obtained by the Chandy-Misra or time warp simulation methods, disregarding any change in the event list makeup that would occur if these strategies were actually used. Although the available concurrency is certainly less than this graph indicates, because no account has been taken of event dependencies, it indicates that one of these two methods might be successful in speeding up this simulator.

Current research is concentrating on gathering more accurate and detailed statistics on a potential Chandy-Misra implementation. Although it is expected that time warp would gain more concurrency, the problems associated with roll-backs are considered too difficult to overcome at the moment. For example, how would one roll back a node instance that reads or writes files, or that has a completely arbitrary nature in general? A time warp implementation only seems possible if some potentially severe restrictions are placed on the nature of nodes themselves. However, one of the more powerful aspects of the Austek simulator is the complete generality of node descriptions that it currently allows.

4 Climate Modelling

The environment has attracted a great deal of attention recently: the ozone layer and the greenhouse effect are common household terms. Physicists at the CSIRO Division of Atmospheric Research have a number of advanced models for simulating the atmosphere. They are planning to predict the average climate for periods of up to 70 years into the future. Such simulations are extremely time consuming, and run for many hours on an expensive Cray Y-MP processor. Many meteorological bureaus are turning to parallel processing in an attempt to simulate weather patterns more accurately at lower cost .

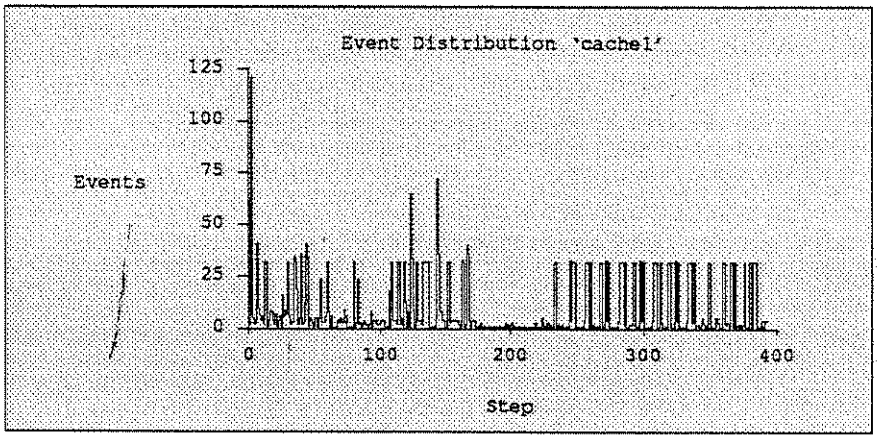


Fig 3 Event list distribution showing the number of events scheduled for each time slot

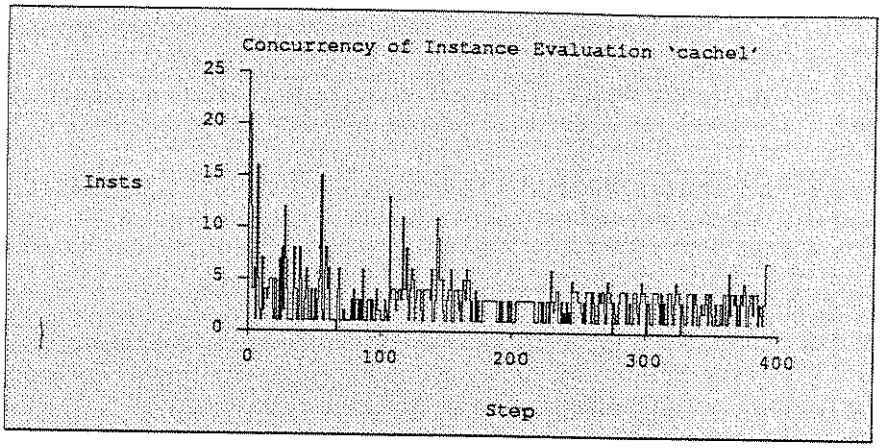


Fig 4 Instance dispatches showing the number of dispatches during each time slot

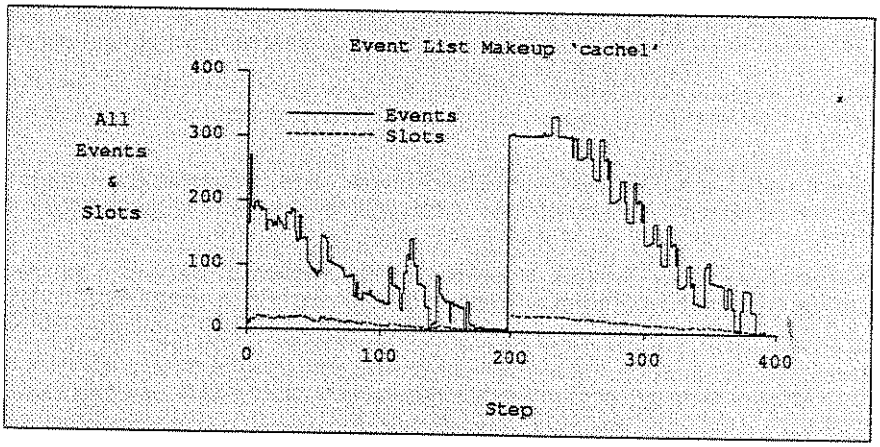


Fig 5 Event list makeup showing all events and slots occupied (present and future)

This project is being conducted in collaboration with the CSIRO Division of Atmospheric Research. A number of operational programs are being analysed to determine the effect of parallel processing. It is possible that a special purpose weather multiprocessor will be constructed to provide access to multiprocessing hardware without the need for dramatic changes in the existing weather models.

4.1 The Shallow Water Equations

The initial study being undertaken is based on the shallow water equations. These equations are a favoured choice for experiments with various model structures and numerical schemes. Although a very simple representation of the atmosphere they do include the two types of horizontal wave motion important in more realistic models, gravity waves and Rossby waves. They are also useful for experiments in parallelising because they include much of the structure (and so problems with data communication and synchronisation) of more complicated models. The shallow water equations describe the motion of a incompressible fluid with a free surface, with the constraint that the horizontal scales of motion are much larger than the vertical. For more details see [14, 16].

The equations may be applied on any domain from a small tank to the global atmosphere or ocean provided it is modelled as a single layer. Applied to the atmosphere the shallow water equations can be taken to represent the flow averaged through the troposphere. Though obviously a great simplification of the full range of dynamics and processes of the real

atmosphere (and as simulated in modern climate and weather prediction models) they are still very useful. On the largest horizontal scales atmospheric motions are approximately barotropic (i.e. don't change with height) and the shallow water equations are a reasonable approximation to the dynamics. Even simpler models (with fixed upper and lower boundaries) were used for the earliest numerical weather prediction experiments and showed some skill [10].

Even for this very simple model the equations are nonlinear and cannot be solved directly, requiring use of numerical methods. One approach is to discretise the equations on a grid, replacing the spatial derivatives by finite differences. The other main approach is a spectral method where the fields are defined as sums of some basis functions. The derivatives are evaluated analytically from the basis functions, giving more accuracy, subject of course to the problems of truncation. Spectral methods are popular in meteorological modelling because the basis functions usually chosen, the spherical harmonics, are similar to the natural large scale modes of the atmosphere and so can give an accurate representation of the flow with relatively few degrees of freedom.

Both finite difference and spectral models treat the time integration in the same way, replacing the time derivative by a finite difference form. Simple leapfrog time integration is usually used, in order to ensure numerical stability, though more complicated schemes are possible.

4.2 Some Early Results

At present only some of the total number of planned experiments have been completed. In this section we present some of these results: Those of the shallow water equations in both Grid and Spectral formulation running on a uni-processor Cray Y-MP as well as an Encore Multimax shared memory machine. The Grid results are also shown for a 1024 processor MasPar. The MasPar spectral code is still being developed.

Cray

Program	Problem Size	Time(secs)	MFlops (ave)	% vectorised
Finite Diff C	32 x 32	6.5	10.8	0.1
Finite Diff C	96 x 96	66.6	9.4	0.1
Finite Diff Fortran	32 x 32	0.7	121.8	94.0
Finite Diff Fortran	96 x 96	4.0	180.5	99.0
Spectral Fortran	32 x 128 x 96	9.1	29.8	60.9
Spectral Fortran FFT[1]	32 x 128 x 96	3.4	64.0	91.8
Spectral Fortran	16 x 64 x 48	2.0	22.7	46.2
Spectral Fortran FFT[1]	16 x 64 x 48	0.6	53.8	92.1
Spectral C	16 x 64 x 48	4.0	9.4	0.2
Spectral C	32 x 128 x 96	24.8	8.9	0.2

Shared Memory

Program	Problem Size	Parallel Times					Peak Speedup
		1	2	4	8	16	
Finite Diff	96 x 96	9480	4976	2407	1211	626	15.1
Finite Diff	32 x 32	690	350	177	93	53	13.0
Finite Diff	16 x 16	161	81	43	23	15	10.7
Spectral	32 x 96 x 128	2248	1127	579	291	150	15.0
Spectral	16 x 64 x 48	354	177	89	46	25	14.0

MasPar

Program	Problem Size	Time (secs)
Finite Diff	32 x 32	2.7

The results presented above show that a shared memory architecture achieves quite good speedup on both the finite difference model and the spectral code. The MasPar machine achieves a good absolute speed in comparison with the CRAY Y/MP. The vectorised 32 x 32 grid code takes 0.7 seconds on the CRAY and 2.7 on the MasPar.

5 Scheduling Architectures

Scheduling occurs in many fields. For example, manufacturing production lines require scheduling of the product runs to maximise factory output; trucks must be scheduled to arrive at the correct bays at the correct times to pick up their loads; the classes and teachers must be scheduled to the periods of the week in schools and colleges.

A new optimisation technique, called simulated annealing, is capable of organising schedules to minimise certain timing constraints. The scheme works by modelling the optimisation problem as a collection of vibrating atoms in a structure. As the structure is cooled, the latent energy in the structure is lowered. In the scheduling problem, this means lowering the number of broken timing constraints. The main problem with simulated annealing is that it is extremely slow. A special purpose architecture has been designed to execute the simulated annealing algorithm on scheduling problems. This processor provides a solution about 1000 times faster than a conventional workstation, and allows simulated annealing to be used for practical problems. It has been applied quite successfully to school timetables. This project is currently investigating other applications for the scheduling architecture.

5.1 The School Timetabling Problem

The particular scheduling problem of interest in this paper is the task of creating a valid school timetable. The problem of creating a valid timetable involves scheduling classes, teachers and rooms in such a way that no *teacher, class or room* is used more than once per period. For example, if a class must meet twice a week, then it must be placed in two different periods to avoid a clash. The timetable is to be distributed across a fixed number of periods per week. A class consists of a number of students. Initially we consider classes to be disjoint, that is, they have *no students in common*. In this scheme, a correct timetable is one in which a class can be scheduled concurrently with any other class. In each period a class is taught a *subject*. It is possible for a subject to appear more than once in a period. A particular combination of a teacher, a subject, a room and a class is called an *tuple*. A tuple may be required more than once per week. Thus, the timetabling problem can be phrased as scheduling a number of tuples such that a tuple, teacher, class or room does not appear more than once per period.

It is possible to define a *cost function* for evaluating a given timetable. This function is an arbitrary measure of the quality of the solution. A convenient cost function calculates the number of clashes in any given timetable. An acceptable timetable has a cost of 0. The optimisation problem becomes one of minimising the value of this cost function. The cost of any period can be expressed as the sum of three components corresponding to a class cost, a teacher cost and a room cost.

The class cost is the number of times each of the classes in the period appears in that period, less one if it is greater than zero. Thus, if a class appears no times or once in a period then the cost of that class is zero. If it appears many times the the class cost for that class is the number of times less one. The class cost for a period is the sum of all class costs. The same computation applies for teachers and rooms. The cost of the total timetable is the sum of the period costs. Therefore, any optimisation technique should aim to find a configuration with the lowest possible cost.

5.2 Using Simulated Annealing

Simulated annealing is a Monte-Carlo technique which can be used to find solutions to optimisation problems. A good review of the theory and practice can be found in [18]. The technique simulates the cooling of a collection of hot vibrating atoms. When the atoms are at a high temperature they are free to move around, and tend to move with random displacements. However, as the mass cools the inter-particle bonds force the atoms together. When the mass is cool, no movement is possible, and the configuration is frozen. If the mass is cooled quickly then chance of obtaining a low cost solution is lower than if it is cooled slowly (or annealed). At any given temperature a new configuration of atoms is accepted if the system energy is lowered. However, if the energy is higher, then the configuration is accepted only if the probability of such an increase is lower than that expected at the given temperature. This probability is given by $P(\Delta E) = e^{-\Delta E/KT}$, where K is Boltzmann's constant. These acceptance criteria are based on the physics of annealing, which is described in [18].

Many optimisation problems can be considered as a number of *objects* which need to be scheduled such that an objective function is minimised. The vibrating atoms are replaced by the objects, and the value of the objective function replaces the system energy. An initial schedule is created by randomly scheduling the objects, and an initial cost (c_0) and temperature (T_0) are computed. Subsequent permutations are created by randomly choosing a number of objects, rearranging them, and computing a change in cost (Δc). If $\Delta c \leq 0$ then the change is accepted. However, if $\Delta c > 0$ then the probability of that change at temperature T is calculated,

$$P(\Delta c) = e^{-\Delta c/T}.$$

If the probability is greater than a randomly selected value in the range (0, 1) then the change is accepted. The most common technique for choosing a random number is to use a pseudo-random uniformly distributed variable on the unit interval. After a number of successful permutations the temperature is decreased by a cooling rate, R, such that $T_n = T_{n-1} * R$, where $0 \leq R < 1$, and T is a real number.

One of the advantages of simulated annealing over algorithms which always seek a better solution (hill climbing algorithms) is that simulated annealing is less likely to get caught in local minima, because the cost can increase as well as decrease.

5.3 Some Performance Results

Because the annealing algorithm is so slow, a special purpose architecture was constructed to match the structure of the algorithm more closely. There is insufficient space in this paper to describe the structure of the architecture, however, it operates as an attached processor to a conventional work station.

The software form of the annealing code has been written in a number of different forms, namely, a very simple model in C and a full model with many additional constraints in sequential Pascal. The annealing accelerator is controlled by a small Pascal program which executes on the

host processor, however, this program does not effect the speed of the hardware, and is thus not relevant to the study.

Below is a table of the number of tuples per second processed by each of these programs on a few different machines. It shows that the accelerator runs about 25 times faster than the optimised simple model C version on the Cray, and 55 times faster than a SUN SPARCstation 1. Interestingly, the Cray version is only two times faster than the SUN version. Analysis of the code indicates that the Cray has performed poorly because there is very little floating point computations and almost no vector operations. Given the Monte-Carlo nature of the algorithm it is unlikely that a vector form could be devised. The accelerator is 260 times faster than a NS32332 Encore Multimax.

If the accelerator is compared with the full model coded in Pascal, then it runs 180 times faster than a SPARCstation, and 900 times faster than a NS32332 in the Encore.

Program	Cray Y/MP	SparcStation 1	IBM PC386	Encore Multimax[1]
C Simple Model	60.0K[2]	28.0K[3]		5.8K
Pascal Full Model	-	8.6K		1.7K
Accelerator Full Model	-	1538.0K	1538.0K	-

Notes:
K = 1000 cycles per second
[1]Encore Multimax with NS32332
[2]Cray SCC Compiler
[3]Sun CC -O4

The current implementation of the accelerator includes many more constraints than those described in this paper. It occupies two standard full length IBM PC boards, and has about 200 integrated circuits. An earlier version of the accelerator which only enforced the simple constraints modelled occupied one IBM PC board, and used about 100 chips. It ran the algorithm at about 2000K operations per second, about 30% faster than the current implementation. The one off component cost for the current accelerator is about $2000 making it extremely cost effective.

6 Summary

The sections of this paper have presented details of the wide range of areas being investigated within the High Performance Computation project of CSIRO-DIT. Each of the investigations is applications driven and the actual work ranges from applications programming through program and architecture performance analysis to specialised hardware development .

In this paper we have concentrated on describing the general problems under investigation and giving some preliminary performance results, rather than describing all of the implementation details. In some cases we have chosen commercial multiprocessors to achieve a faster solution, and in one other we have constructed a special purpose computer architecture. In the latter case the speedup is enormous and has a very low production cost. It is expected that by April 1991 demonstrators will be completed for most of the projects described in this paper.

References

1 D. Abramson, G. K. Egan, "An Overview of the RMIT/CSIRO Parallel Systems Architecture Project," Australian Computer Journal, Vol 20, No 3, August 1988.

2 D. Abramson, G. Egan, "The RMIT Data Flow Computer: A Hybrid Architecture," The British Computer Journal, June 1990

3 D. Abramson, "Using a Dataflow Multiprocessor for Functional Logic Simulation," RMIT TR-112-063R, 1987. Also appeared in Third International Conference on Supercomputing, Boston, Massachusetts, May 15-20, 1988.

4 D. Abramson, G. K. Egan, "Design Considerations for a High Performance Dataflow Multiprocessor," Advanced Issues in Dataflow Computing, Prentice-Hall, in print.

5 M. L. Bailey & L. Snyder, "An Empirical Study of On-Chip Parallelism," 25th ACM/IEEE DAC, 1988.

6 A. T. Brunger, "X-PLOR (Version 1.5) Reference Manual," Yale University, 1988.

7 B. Busbee, "Supercomputing and the environment," Super Computer, March 1990, pp 7-17.

8 K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Comm. of the ACM, April 1981, pp. 198-206.

9 P. S. Chang, G. K. Egan. "An Implementation of a Spectral Baratropic Numeric Weather Prediction Model in the Functional Language SISAL," RMIT TR-112-088R, 1990. Also appeared in 1990 ACM Symposium on Parallel Programming, Seattle, Feb 1990.

10 J. Charrey, R. Fjortoft and J. von Neumann, "Numerical Integration of the barotropic vorticity equation," Tellus, Vol 2, pp. 237-254, 1950.

11 J. J. Dongarra and D. C. Sorenson, "SCHEDULE: Tools for Developing and Analysing Parallel Fortran Programs," pp. 363-394 in "The Characteristics of Parallel Algorithms," L. H. Jamieson, D. B. Gannon and R. J. Douglass (eds.), MIT Press, 1987.

12 R. M. Fujimoto, "Lookahead in Parallel Discrete Event Simulation," Proc. 1988 ICPP, August 1988.

13 S. L. Graham, P. B. Kessler, M. K. McKusick, "gprof: A Call Graph Execution Profiler," Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

14 G. J. Haltiner and R. T. Williams, "Numerical Prediction and Dynamic Meteorology," Wiley, 1980.

15 D. R. Jefferson, "Virtual Time," ACM Transactions on Programming Languages and and Systems 7(3) pp. 404-425, July 1985.

16 J. Pedlosky, "Geophysical Fluid Dynamics," Springer-Verlag, 1979.

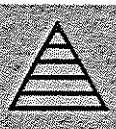
17 K. F. Wong, M. A. Franklin, R. D. Chamberlain and B. L. Shing, "Statistics on Logic Simulation," 23rd ACM/IEEE DAC, 1986.

18 P.J.M. van Laarhoven and E.H.L. Aarts, "Simulated Annealing: Theory and Applications", D. Reidel Publishing Company, 1987, Kluwer Academic Publishers Group.

THIRD AUSTRALIAN SUPERCOMPUTER CONFERENCE

THE UNIVERSITY
of MELBOURNE

DECEMBER 3-6, 1990



ORGANISED BY:

STRATEGIC RESEARCH FOUNDATION

THE UNIVERSITY OF MELBOURNE