

Storm: A Multibus System for Climate Modelling

D. Abramson †§
G. Cameron †§
M. Dix ‡
M. Makies §

† Division of Information Technology
723 Swanston St
Carlton, 3053
C.S.I.R.O.

‡ Division of Atmospheric Research
C.S.I.R.O.

Contact Information: davida@mel.dit.csiro.au

§ Department of Computer Systems Engineering
Royal Melbourne Institute of Technology
P.O. Box 2476V, Melbourne 3001, Australia.

ABSTRACT

This paper considers the design of a multiprocessor, called STORM, specially targeted for efficient execution of large global climate modelling programs. It begins by discussing the limitations of current shared and distributed memory multiprocessors and shows that a shared memory machine with only partial processor/memory interconnect can provide a scalable platform for climate model programs. The paper presents a detailed analysis of some memory traces taken from CSIRO 4, a real model developed and used by the CSIRO Division of Atmospheric Research in Australia. The paper considers a prototype of STORM built from Sun Sparc Stations.

1. INTRODUCTION

Global Climate Modelling (GCM) is one of the "grand challenge" computing problems, and consumes vast amounts of processor time. Models are being designed to allow researchers to compute the long term effects of changes in the earth's atmosphere. The challenge to computer architects is to provide hardware and software platforms which provide an efficient execution environment for the models, and which are easy to program.

A number of experiments have demonstrated that shared memory multiprocessors can be used effectively to reduce the execution time of Global Climate Models [1, 2]. Such machines range from a small number of very high speed vector processors, like the CRAY Y/MP8 [3] to a larger number of much slower processors like the Encore Multimax family [4].

In Figure 1 we show the results of some experiments in which we parallelised the spectral formulation of the *shallow water equations* on a shared memory multiprocessor, an Encore Multimax. The shallow water

equations are the heart of the *dynamics* of most global climate models, and are responsible for the majority of the interprocessor communication (non-local memory references) in a parallel implementation of a model. These results, which plot execution time against number of processors, demonstrate that near linear speedup can be achieved on a system which support up to 20 processors on one bus. Two different problem sizes are shown.

In most cases the parallelisation of the sequential algorithms is achieved by loop slicing, in which each processor is responsible for a subset of loop iterations. The results obtained above were achieved by parallelisation of the outer loop in the programs. This was sufficient because of the small number of processors, however, with a larger number of processors it would be necessary to parallelise the inner loop as well. The data is stored in shared memory and is accessible to all processors.

In Figure 2 we show the size of the sequential and parallel implementations of two formulations of the shallow water equations, a finite difference and a spectral formulation. These were coded in C using the *Argonne National Laboratories* macros [11]. The graph clearly shows that the distributed memory implementation of the codes are longer than their shared memory equivalent. The extra source required for the distributed memory code is composed of declarations for the message buffers, plus code for transferring data between processes at the correct points. Since the *dynamics* of the climate model are only responsible for a portion of the code, a complete model (like the CSIRO4 model cited later in the paper) would be much larger, and thus the overhead of the message passing code would be smaller. However, it is our contention that the addition of message passing code to a sequential program is more complex and error prone than the loop unravelling of parallel loops performed in the shared memory code. In the case of CSIRO4 it was possible to parallelise the original F77 code for execution on a shared memory machine (a Silicon Graphics Power Series) with

very little modification. Parallelisation for execution on a distributed memory architecture is much more difficult and error prone.

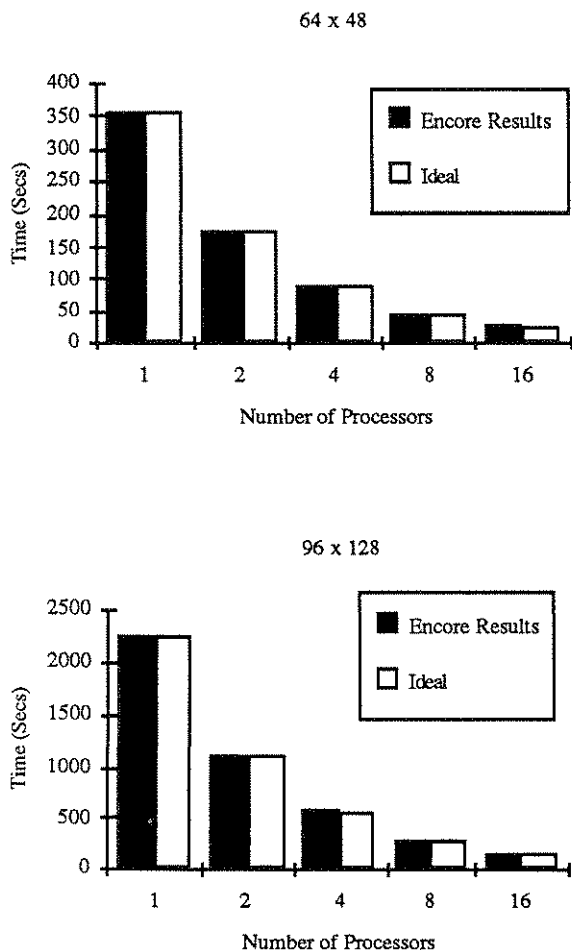


Figure 1 - Execution of two different problems sizes of shallow equations on Encore Multimax

The problem with most shared memory machines is that, in general, they are not scalable to very large numbers of processors, because the processor-memory interconnect becomes a bottleneck. Exceptions to this include a number of experimental scalable shared memory machines that are being constructed, such as Stanford DASH [9], MIT Alewife [10], Georgia Tech Beehive, and a few commercial machines, most notably the the Kendall Square KSR1. However, most current shared memory machines are bus based and thus have limited scalability. Consequently, arrays of message passing processors, which are scalable, have been used to implement such complex models. However, the ease of scalability is replaced by a much more difficult programming paradigm. Further, the message passing hardware is typically slower

than the normal memory bus, and thus message passing arrays can perform badly when large amounts of data must be passed between nodes. In the case of the climate model, information must be passed across latitudes and down longitudes, which requires data to be passed through intermediate nodes of the machine. Whilst such hardware is available on modern message passing machines like the Intel Paragon [12], it is more complex than the scheme described in this paper.

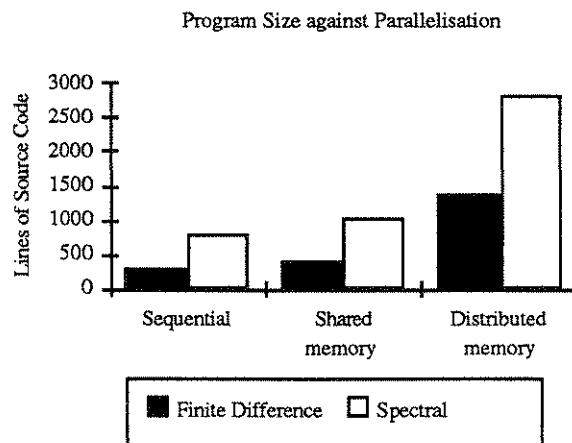


Figure 2 - Program Size against Parallelisation Strategy

Shared memory multiprocessors typically provide equal communication delays from all processors to memory, and are thus ideal for problems in which global communication is required. However, the most commonly used formulation of Global Climate Models, the spectral method, mostly requires communication across latitudes and down longitudes, and uses very little access from all processors to all memory. Thus, a shared memory machine provides more inter-processor bandwidth than is required, and the size of the machine is limited unnecessarily. In this paper we consider the design of a machine which has an interconnection structure more closely matched to the requirements of the spectral model, allowing much larger machines to be constructed. The machine provides the appearance of globally shared memory through a mesh of row and column buses. Programs can be written as though such memory is available, making software development and testing similar to that required for a conventional shared memory processor. In a similar way, virtual memory is a useful abstraction even when the hardware must swap pages to achieve the appearance of more physical memory than is available.

Shared memory machines also provide cache coherence hardware which keeps all caches consistent. Whilst this is a desirable feature for arbitrary programs, global climate models can execute correctly under a less stringent

coherence policy. Consequently, the machine described in this paper reduces the hardware complexity considerably over a general purpose machine by not including cache coherence hardware. Cache coherence is enforced when required through a software managed scheme.

The construction of such a machine poses many design choices. Some examples of questions that need resolution are:

- How much bus bandwidth is required for local, global, vertical and horizontal communication?
- How many globally shared variables are manipulated, and can they be addressed transparently in spite of the physical interconnect?
- What is the effect of local processor cache memories, and is cache coherence hardware required?
- Is it possible to build the machine from standard workstations using their bus interface to provide access to the memory array, or must dedicated processing elements be constructed? The former has a number of advantages, such as allowing a multiprocessor to track technology advances in the workstation arena.

This paper begins by discussing the mathematics of climate modelling and then describes the architecture of the STORM machine. We then present some results of program traces which allow the questions posed above to be answered in sufficient detail to design the machine.

2. CLIMATE MODELLING

2.1 Structure of the model

Global climate models attempt to simulate the processes important in the Earth's weather. They are based on the dynamics and thermodynamics of the large scale flow, as described by the Navier Stokes equations. Physical processes, such as the effect of solar and longwave radiation, the formation of clouds and rainfall and interactions with the surface must also be included.

The partial differential equations that describe the atmosphere cannot be solved directly and some sort of discretization must be used to obtain a numerically solvable system. In climate modelling the spectral method, where the basic variables are represented as sums of spherical harmonics, is the most common approach. The formulation of the model used here is described by [5].

For example, the horizontal field of temperature can be expressed as

$$T(\lambda, \phi) = \sum_{m=-J}^J \sum_{l=|m|}^{l=J} T_l^m Y_l^m(\lambda, \phi)$$

where λ, ϕ are longitude and latitude respectively and Y_l^m is a spherical harmonic function with zonal wavenumber m and total wavenumber l . The spectral representation is well suited to the globe and gives a compact and accurate representation of atmospheric fields. However, in order to evaluate nonlinear terms in the dynamical equations, and to calculate the physical processes, it is also necessary to have the model variables on a longitude and latitude grid.

An outline of the model structure is

- 1) Transform spectral fields to gridpoint fields. Calculate the changes in the temperature and moisture fields at each gridpoint due to physical processes. Add the tendencies to the old fields to obtain modified gridpoint fields and transform these back to spectral space.
- 2) Transform spectral amplitudes to gridpoint space. Evaluate the changes in the fields at each gridpoint due to the nonlinear terms in the equations of motion. Then transform these tendencies back to spectral space.
- 3) Evaluate the linear tendencies in the equations of motion (in spectral space).
- 4) Perform the time integration (in spectral space), to update the fields based on the tendencies calculated in steps 2 and 3 above.

Step 1 is referred to as the physics loop of the model and step 2 as the dynamics loop.

The conversion from the spectral representation to the grid requires a Legendre transform and a Fourier transform. Both of these require global access, but in each case it is only in one direction. It is this restriction that makes the proposed architecture feasible. As an example, the transforms from the temperature represented as a sum of spherical harmonics to the grid point representation are:

- The temperature, represented as $T(m, l)$, begins in spectral space, where m is the zonal wave number and l is the meridonal wave number.
- A Legendre transform, essentially a weighted sum over the total wavenumber, converts this to the intermediate Fourier space, $T(m, \phi)$, where ϕ is the latitude.

- At each latitude a Fourier transform (a weighted sum over the zonal wavenumbers) converts to grid space $T(\lambda, \phi)$, where λ is the longitude.
- In the first transform memory access is only over columns and in the second over rows.

The same procedure is applied to all of the key variables in the model.

3. STORM: A SCALABLE SHARED MEMORY MULTIPROCESSOR

In section 2 we demonstrated that the spectral formulation of a global climate model requires global access across latitude or longitudes in order to perform the spectral transform. Most of the physics computations only require access to local data. Some constants and variables which are altered rarely are required by the entire computation, however, with the assistance of processor cache memories these values are only required rarely, and do not constitute a major memory load. This addressing pattern can be supported with a bus based multiprocessor like the 16 processor example shown in Figure 3.

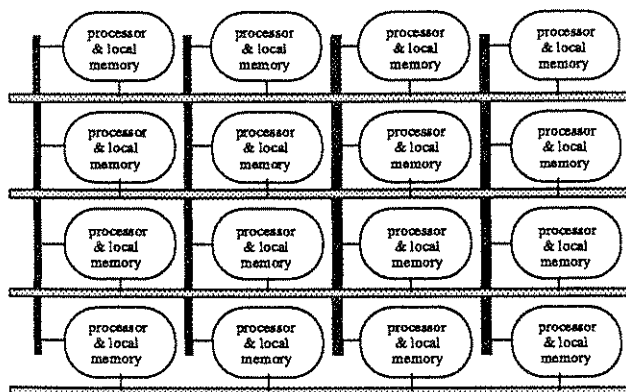


Figure 3 - A Scalable Bus Based Machine

3.1 Memory Layout of GCM on STORM

Unlike most shared memory machines, STORM does not store shared data structures in one physically shared memory. Instead, the memory is partitioned into a number of processor local memories which can be addressed globally through the bus based interconnect. A consequence of this is that data structures must be partitioned so that they reside on the processor on which they are manipulated. However, unlike a message passing machine, data can be accessed by remote nodes by the same mechanism used for local memory; albeit with a higher latency. The data structures are partitioned using three basic mechanisms; one in which structures with *two* or more indexes are spread across all memory modules in *two* dimensions; one in which data structures with *one* or

more indexes are distributed across either the *first* row or *first* column of memory modules; and one in which scalars and indexed data structures are allocated to memory module one.

Figure 3 showed the layout of the machine partitioned in terms of latitude and longitude. Some of the data structures are indexed by two *wave numbers* rather than longitude and latitude. In these cases the wave numbers are used to partition the processor sub ranges. In order to limit the communication requirements during the spectral transform stage of the dynamics computation, a two dimensional partition of the key two dimensional data structures is required. In this way, each processor holds a rectangular section of the grid structure as shown in Figure 4. One dimensional data structures are mapped onto either the first row or column of the machine, depending on the type of their index value. For example, single dimensioned arrays with latitude as an index are held in the first column of the machine, and are thus accessible to all processors with latitude values in the correct range via the horizontal buses. Scalars which are required globally are mapped into processor one, and thus can be accessed by the first row and column by a simple bus access and from any other processor as two linked bus accesses.

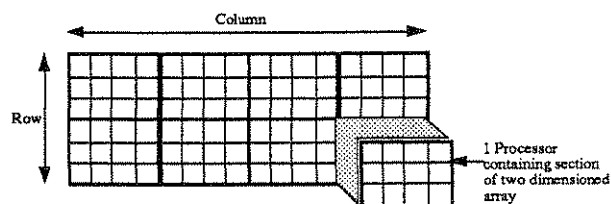


Figure 4 - Mapping Grid structures onto mesh

3.3 Address Translation

In the STORM machine each processor has access to two buses; one for connection to processors which are allocated to the same longitude and one for connection to processors which are allocated to the same latitude. Each processor has direct access to some section of the grid of variables. However, when an address is generated for a section of data outside the processor's local memory, a module selection circuit determines which bus will give access to the data, and an external memory request is initiated. In this way, a processor can *see* any of the key variables with the same longitude or latitude as the variables in its local memory.

Because STORM makes use of standard compilers, the data structures are mapped into a one dimensional virtual address space, rather than the two dimensional array provided by the hardware. The consequence of this is that addresses must be translated from linear virtual addresses into two dimensional addresses by special address

translation hardware shown in Figure 5. The figure shows the sub-division of an address in shared memory. It is composed of 5 regions; 2 bits address for the byte within word; 14 bits of address allocated for up to two indexes; a number of bits for higher order indexes; a number of bits for higher order address bits and three bits to denote a segment descriptor. The segment descriptor allows for different mapping functions for different sized data structures. Thus all data structures of similar type are grouped together into segments, and the segment number is combined with the two lower indexes to form an address into a mapping table. The mapping table then produces a modified address composed of a memory module number and an offset within the module.

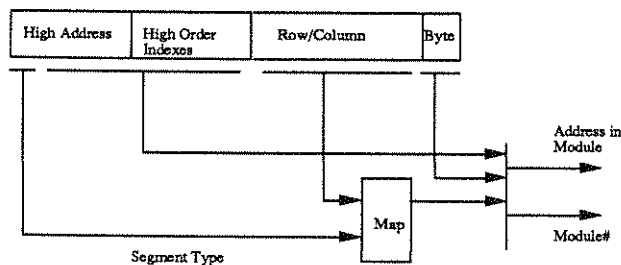


Figure 5 - Address Translation Mechanism

Access to *global* variables is supported by passing a request from one bus to another, as shown in Figure 6. Thus, global variables which are stored in the memory of processor one, can be accessed by any processor in the mesh. However, this access mechanism is much slower and also causes more bus contention than the normal mechanism. It is implemented by linking the finite state machines that control the buses. In the analysis section of the paper we demonstrate that the global references can be made very infrequent by appropriate caching of data.

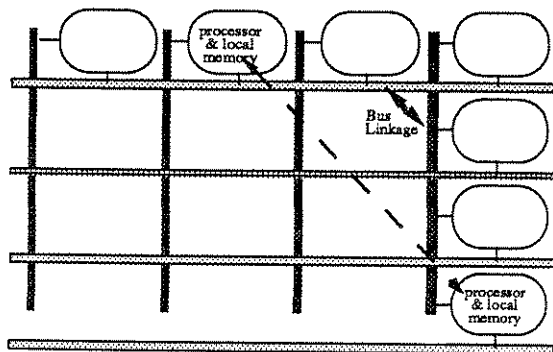


Figure 6 - Bus linking to allow global access to memory

3.2 Loop Slicing

Loop slicing is the only method used to extract concurrency from the originally sequential code. In line with the partitioning scheme described in the previous

section, loop bounds are altered in accordance with the processor identity. Since the data structures are partitioned statically onto the machine, the loop bounds can be re-computed at run time by using the partitioning information.

3.4 Cache Structure and Cache Coherence

Caches are required by processors to reduce the latency of both local and external memory references. They are particularly important in multi-processors because they reduce interprocessor communication. However, if processors are allowed to freely cache variables fetched from non-local memory modules then cache incoherence can arise [6]. A number of cache coherence schemes are possible, ranging from hardware to software managed schemes [7, 8]. Hardware managed schemes have the major advantage that they do not require software support in order to guarantee consistency. There are two main classes of hardware schemes; snooping protocols and directory based protocols. Snooping schemes are restricted to bus based multiprocessors and do not scale well. Directory based schemes can be applied to non bus-based machines. Both schemes require a significant investment in hardware.

Software based schemes have the advantage that they do not require as much hardware support, but do require extensive code analysis in order to be transparent to the user programmer. This is an active area of research for analysis of general purpose programs. Our simulation studies have demonstrated that global climate models have the property that they only require cache coherence to be enforced at the end of each time step of the computation and after the transform steps. Thus it is possible to enforce cache coherence by planting a very small number of directives into the source code of the model. This is the approach taken by STORM.

3.5 Processor Design Issues

An important choice in the design of STORM is whether the processors must be built from chip level microprocessors or whether they could be constructed from board level workstations. The former has the advantage that the hardware can be optimised for the multiprocessor architecture, but the disadvantage that the boards would not be useful for any other task. The major advantage of using commercial workstations is that they contain complete input-output hardware, and operating system and compiler software. It also has the advantage that it is possible to build only the shared memory and interconnection hardware without the need to build the processor and input output system. The disadvantage of this approach is that the memory module selection circuit, which must be placed before the memory holding the data structures, must be connected to the processor by its external system bus. In the case of the SUN Sparc

Stations being considered in this paper, this means placing the address interpretation hardware and the local memory on the SBUS. Since an SBUS transfer is slower than a normal internal memory reference, this configuration has the potential to slow the machine down. In the results section we demonstrate that it is possible to use the SBUS of a Sparc Station without slowing the machine down significantly.

Figure 7 shows the structure of STORM making use of existing workstations. It shows that the local memory of the node is placed on the external side of the SBUS interface, and can thus be accessed by both the local and external processors. The use of address translation hardware means that it is not possible to store shared data in the local memory of the Sparc Station.

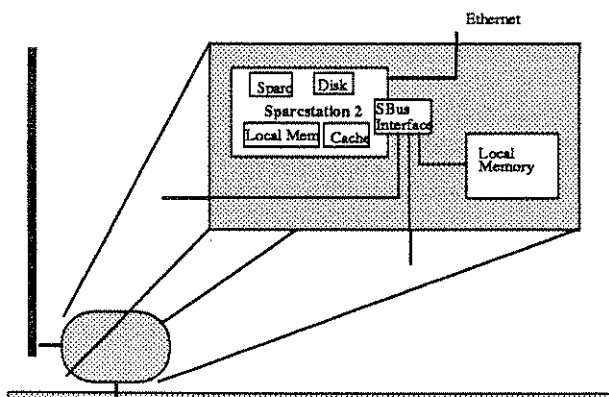


Figure 7 - Design of Processor Node

3.6 Key Parameters required by Simulation

This overview of STORM is only intended to set the scene for the type of instrumentation required, rather than give an in depth description of the STORM design. The performance of the multiprocessor machine is most affected by the bandwidth provided between a processor and its memory sub-system. Consequently, the following questions need answering:

- What is the amount and nature of the traffic between a processor and its local memory?
What is the effect of caching the SBUS traffic?
- What is the amount and nature of the traffic between a processor and memory which is addressable by its row or column bus?
- What is the amount and nature of the traffic between a processor and data held in memory module one?

In the next section the results of the analysis of a real climate model will be presented, which assist in answering these questions.

4. ANALYSIS OF A GLOBAL CLIMATE MODEL

4.1 General Results

In this section the analysis of a 4 layer global climate model, called CSIRO4 [5] is presented. CSIRO4 is composed of 77 routines of FORTRAN code, and contains some 9000 lines of source code. CSIRO4 has been parallelised for execution on a Silicon Graphics shared memory multiprocessor, so its suitability for parallel execution has been already established [2]. CSIRO4 represents near state of the art in climate models, and thus makes a suitable vehicle for instrumentation. The purpose of this section is to present the results of a study of the memory traffic patterns of CSIRO4 when mapped onto the STORM hardware structure. The study allows the key design parameters of STORM to be chosen, plus provides an indication of the suitability of SUN Microsystems Sparc Stations as the processor nodes of STORM. The study focuses on the effect on the uni-processor performance of the Sparc Stations when configured in the STORM machine rather than predicting the multiprocessor performance of the programs. This is because we expect to achieve the same sort of speedup profile that was shown in Section 1.

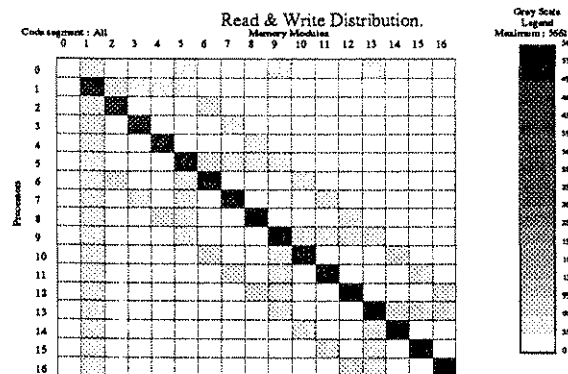


Figure 8 - Instrumentation of CSIRO 4 - Total References for 16 processors

Figures 8, 9 and 10 contains a map of the processor memory traffic for a 16, 64 and 256 processor STORM configuration as shown in Figure 3. The diagram shows the amount of memory traffic generated by each processor by using a grey scale. The map contains a number of results. First, the program conforms to the execution patterns which execute efficiently on the STORM hardware. For example, most of the memory traffic is directed to the local memory, with some amount being directed to processors sharing the same longitudes of

latitudes. This type of traffic is supported by the row and column buses of STORM.

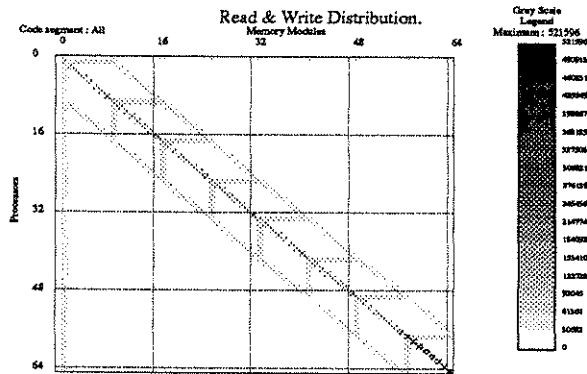


Figure 9 - Instrumentation of CSIRO 4 - Total References for 64 processors

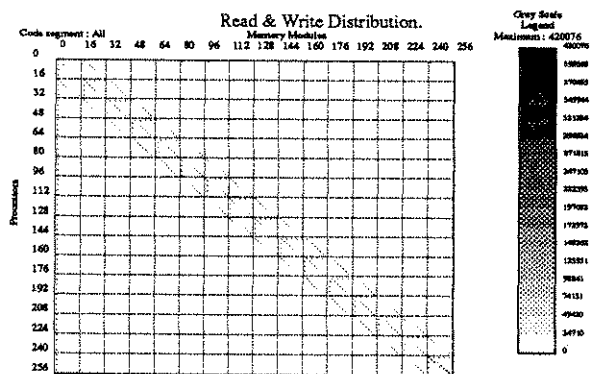


Figure 10 - Instrumentation of CSIRO 4 - Total References for 256 processors

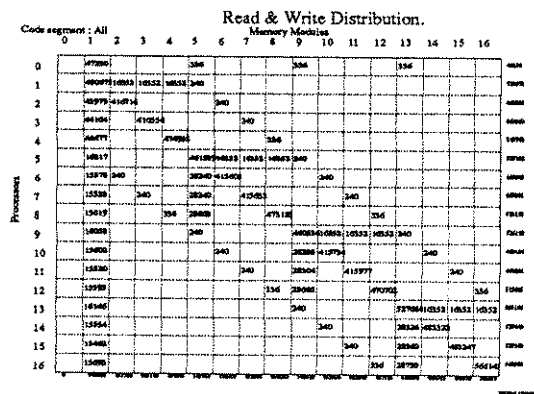


Figure 11 - Numeric results of total references for 16 processors

The map also shows traffic from all processors to memory module one, which contains constants and infrequently modified variables. The results suggest that a large amount of traffic is generated in this way. However, the grey scale scheme contains a non-linearity when the number of accesses becomes very low so that low counts do not appear as zero. An option on the post processing software allows the system to replace the grey scale with absolute numbers, and this is shown in Figure 11 for the 16 processor case. Each cell in these figures contain the absolute number of accesses as well as the fraction of processor accesses directed to the memory module, and the fraction of module references from the processor. These results show that the absolute number of references from all processors to module one is around 3%, which is still quite large given the slow access pathway from any processor to module one. The figure also shows that most processors access their local memories about 90% of the time. The transforms contribute from 3% to 6% of processor traffic, depending on whether they are row transforms or column transforms. Clearly the module one traffic needs to be much lower than shown or the maximum size of the machine will be severely restricted. It is not possible to display the same results for the 64 and 256 processor cases in the limited space, so they are summarised in Figure 12. It is worth noting that as the machine becomes larger the fraction of references from each processor to memory 1 reduces. This is because the problem size is held constant, and thus the references to global constants decreases on a per processor basis.

Type of traffic	Traffic		
	16 procs	64 procs	256 procs
% references to memory 1	3%	3%	1%
% references to same row	3%	3%	3%
% references to same column	3%	6%	6%
% references to local memory	91%	88%	90%

Figure 12 - Numeric results of total references for 16, 64 and 256 processors

Figure 13 shows the effect of allowing module one references to migrate into the processor caches for a 16 processor machine. The results are similar for the 64 and 256 processor machines. The main effect is that the number of module one references is reduced to an extremely low number. The results indicate that the amount of local traffic very clearly dominates the computation, and that the loading on the vertical and horizontal buses is quite low. The amount of truly global

addressing is almost zero. An alternative technique for removing the module one traffic would be to duplicate the global data structures and post updates each time step.

50%, and thus it is important to measure the effect of this low cache hit rate.

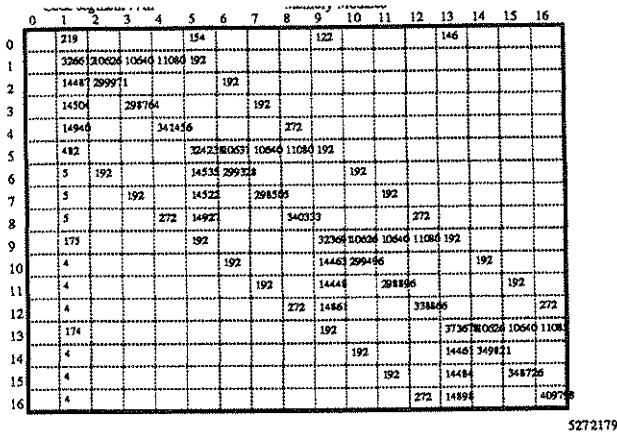


Figure 13 - Numeric results of total references after caching

4.2 Cache Hit Rates

The cache experiments indicate that only 8% of the total memory traffic generated by the model is directed to shared memory. Thus, in computing the degradation in speed of the processor due to the placement of the local shared memory on the SBUS, the cache hit rates should only be applied to 8% of the memory traffic. The remaining 92% performs at the same speed as when the model is run on a uni-processor SPARC Station. Figure 14 shows the cache hit rate plotted against line size for various cache sizes, ranging from 8192 bytes to 65536 bytes as used in the SPARC Station. Of course, the SPARC Station cache is used to hold local and non-local data as well as code. In general, the plots show the cache hit rate increasing with line size, due to the array based addressing used in the code. A few anomalies are present at certain line sizes.

The most important result is that the hit rate for caching only non-local shared memory variables is relatively high (in the order of 85%), where as the hit rate for caching only local shared memory variables is very low (around 30%). This is due to the addressing patterns used on these two classes of data. Shared variables which are held in local memory are typically only accessed a few times in the computation per time step, and thus exhibit a relatively low cache hit rate. However, the globally accessible information in memory module one, plus the transform data, is read multiple times per processor in each step of the computation, and thus benefits much more from a cache memory. This results explains why the access map in Figure 13 shows such a remarkable reduction in module one traffic when the cache is enabled. These results indicate that the cache hit rate for all shared variables would only be expected to be in the order of

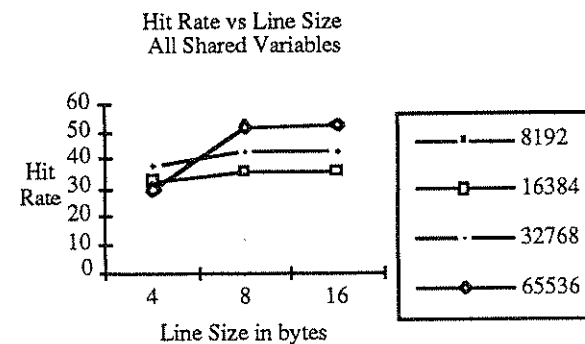
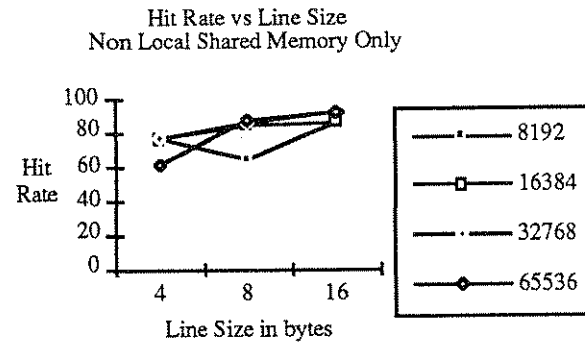
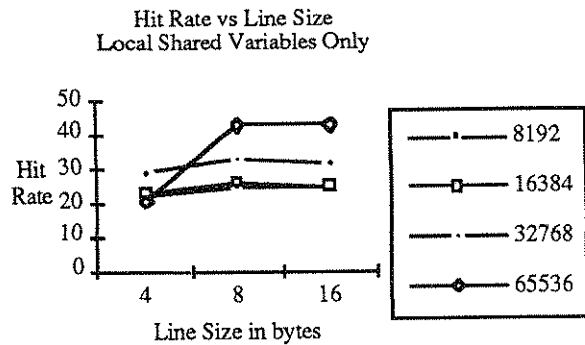


Figure 14 - Hit rates versus line size for varying cache sizes

The results suggest that the low cache hit rates may affect the speed of the processor severely and may slow down the program execution. In Figure 15 we show the slow down of the processor as a function of the achieved cache hit rate and the speed of the external memory and bus system. It considers four different scenarios for ratios of cache memory (C), local Sparc Station memory (LM), external local memory (ELM) and external memory

accessed via the bus (EEM). It shows that a cache hit rate of 50% will yield a slow down of about 10% if the external local shared memory is 10 times slower than the cache, whilst the Sparc Station local memory is only 5 times slower than the cache and the bus based accesses are 20 times slower. We believe these figures can be achieved with current technology.

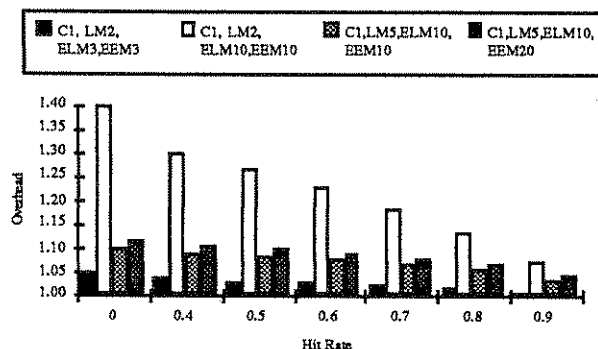


Figure 15 - Slowdown of processor against hit rate and local/external access time

Figure 16 shows the same computation when the Sparc station is 3 times slower than the cache memory and the external local shared memory is 5 times slower for different bus access rates from 10 to 100 times slower. It shows that in order to keep the slowdown within 10% it is necessary to limit the external bus memory access time to 30 times the internal memory access time when the cache hit rate is 50%. It also shows that even when the external bus accessed memory is 100 times slower than the cache, the slowdown is only 30%.

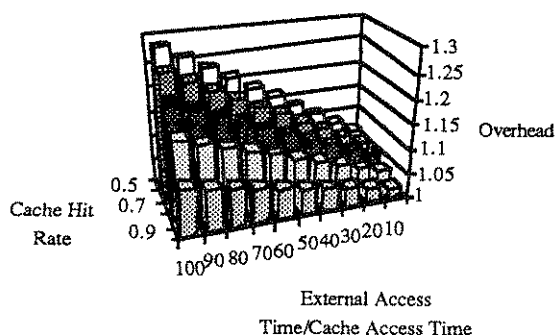


Figure 16 - Slowdown of processor against hit rate and local/external access time

4.4 Summary of analysis

The results presented in this section help answer a number of the questions posed earlier in the paper.

Q What is the amount and nature of the traffic between a processor and its local memory?

A Only 8% of memory traffic is directed to shared data, the remaining 92% is for access to process local data and constants. Of this 8%, 90% of these accesses are directed to the local memory. This means that providing the time to address this local memory is not much slower than the internal memory of the machine, then the processor will not be slowed down significantly. We have presented a number of parametric studies which show how the slowdown of the processor is related to the external memory speed.

Q What is the effect of caching the SBUS traffic?

A Caching the SBUS can reduce the amount of accesses to memory module one from around 3% per processor to almost zero. Caching local shared variables does not have such a dramatic effect. The parametric studies have taken the low cache hit rate into account. An alternative for removing the module one traffic is to duplicate the global data structures.

Q What is the amount and nature of the traffic between a processor and memory which is addressable by its row or column bus?

A About 5% of a processor's traffic to shared memory is required for the row and column transforms. The parametric studies illustrate the effect of slowing down the buses. Quite significant degradation of the external bus speeds does not seem to affect the overall timing.

Q What is the amount and nature of the traffic between a processor and data held in memory module one?

A If memory module one traffic is not cached, each processor sends about 3% of its shared memory traffic to module one. This would limit the size of the machine severely given the low utilisation of the row-column buses. However, caching these variables allows the module one traffic to be reduced to near zero, thus allowing the memory module one bottle neck to be removed.

5. CONCLUSIONS

The results indicate that the proposed architecture can provide an efficient execution platform for climate models based on the spectral transform method. Almost all traffic is directed to the row and column buses, and the processor

cache is able to reduce the number of global accesses to an insignificant amount. Construction of a small prototype is currently in progress to test some of the new architectural features of STORM before a larger configuration is built.

The results indicate that it is possible to use commercial workstations as processor nodes, and to connect them through their external bus interconnects. One important engineering factor in the choice of the Sun Sparc Station was that it allowed us to cache the external SBus accesses. Our demonstrator is being constructed from Sparc Station 2's.

The paper has not concentrated on the speedup we expect from the multiprocessor as a result of parallel execution, but rather the slowdown a single processor would experience from being configured in this way. Given that we can keep the slowdown to a sufficiently low level, the overall speedup should be commensurate with the speedup figures cited in the introduction. These results will be the topic of a future paper.

Acknowledgements

The work described in this paper was conducted under the joint CSIRO/RMIT Program in High Performance Computation. Thanks go to the members of the program team. In particular, Rhys Francis assisted in the formulation of the instrumentation process, and also edited a draft of this paper. Mark Rawling assisted in the instrumentation of the system libraries.

References

- [1] G.-R. Hoffmann and D.F. Snelling, "Multiprocessing in Meteorological Models", Springer-Verlag, 1988.
- [2] Rotstayn, L. and Dix, M., "Parallelization of a Spectral General Circulation Model", Supercomputer, 9(1), pp 33-42, 1992.
- [3] Cray Research, "The CRAY-Y-MP Series of Computer Systems", CRAY Research Inc, 608 Second Ave Sth, Minneapolis, MN 55402.
- [4] Encore Corporation. "The Multimax Technical Summary".
- [5] Gordon, H. "A flux formulation of the spectral atmospheric equations suitable for use in long term climate modelling", Monthly Weather Review, 109, 56-64, 1981.
- [6] Smith, A. J., "Cache memories", ACM Computing Surveys, vol 14, No 3, pp 473-530, September, 1982.
- [7] Veidenbaum, A., "A Compiler-assisted cache coherence solution for multiprocessors", Proceedings of 1986 International Conference on Parallel Processing, pp 1029-1036, St Charles, III, 1986.
- [8] Sweazey, P. and Smith, A. J., "A class of compatible cache-coherence protocols and their support by the IEEE Futurebus", Proceedings of the 13th International Symposium on Computer Architecture, June 1986, Tokyo, Japan, pp 414-423.
- [9] Lenoski D., Laudon J., Gharachorloo K., Weber W., Gupta A., Hennessy J., Horowitz M. and Lam M., "The Stanford Dash Multiprocessor.", IEEE Computer, pp 63-79, March 1992.
- [10] Nussbaum D., Vuong-Adlerberg I., and Agarwal A., "Modeling a Circuit Switched Multiprocessor Interconnect.", Proceedings of the 1990 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Boulder, pp 267- 269, May 1990.
- [11] Lusk, Overbeek, et al, "Portable Programs for Parallel Processors", Holt, Rinehart and Winston Inc, 1987.
- [12] Foster, I., Gropp W. and Stevens R., "The Parallel Scalability of the Spectral Transform Method", Monthly Weather Review, 120, pp 835 - 850, 1992