

Self-Tuned Distributed Multiprocessor System

Ronald Pose, Jonathan Wells, Vincent Fazio

Department of Computer Science,
Monash University,
Clayton, Victoria 3168,
AUSTRALIA
E-mail: rdp@cs.monash.edu.au

Abstract. Self-tuning, a technique devised by Ted Kehl, is a new clocking paradigm which incorporates the best features of conventional synchronous logic along with advantages offered by the asynchronous, self-timed paradigm. The principles of self-tuning are explained and an outline of the application of self-tuning to the interconnection network of a large scale distributed shared memory multiprocessor being developed at Monash University is presented. Extensions of the technique to the processors and memory units of the system are envisaged.

1 Introduction

When performance limitations of conventional single processor computer systems first encouraged people to integrate a number of processors and memories into a single system the approach taken was to use either a shared memory bus or a switch interconnecting processors and memories. Various hierarchies and networks of switches have been employed in attempts to scale the systems beyond the capacity of a single switch or bus. Massively parallel computing systems with hundreds or thousands of processors interconnected either with a shared memory structure or via a structure which allows messages to be routed between the processors have been developed, although they have not been as commercially successful as was originally expected.

Massively parallel computing architectures are becoming widely accepted in many computationally intensive areas. One of the prime advantages touted is their scalability, and yet while in principle a good degree of scalability is possible, in practice the unit of scaling is very coarse. The net effect of this is to make incremental expansion of such machines impractical, except for large and expensive expansion to multiples of the original size, and in some cases expansion by powers of two is required.

The main problem with scaling these machines to larger and larger configurations is the nature of the interconnection network. In order to get the bandwidth required to make effective use of the machine, significant resources are dedicated to the interconnection network. The interconnection network tends to

involve the majority of the circuitry and the cost of current massively parallel computer systems. It also tends to be built out of large switching units. Given the relative costs of the switching units and the processors, it is not cost-effective to obtain a machine with fewer than the number of processors required to occupy a switching unit. The unit of scalability is thus this number of processors at best, and depending on the switching topology may require exponential growth patterns to make effective use of the switching investment. While it may seem reasonable to go from a 4 node system to an 8 node system, when it comes to expansion granularity much coarser than this as is becoming common, the incremental costs become prohibitive. One has to find the money in large amounts rather than by expanding the system gradually as needs and finances permit. Another difficulty in scaling and upgrading such systems is that typically all nodes have to be of the same type. In particular if a new model node is released, it cannot usually be combined in a system with older node versions, which often run at slower clock speeds.

The Monash architecture and interconnection topology distributes the switching investment uniformly across the nodes of the system. It is thus scalable from a single processor-memory node up to over a thousand nodes in increments of single nodes. The architecture has been developed from the beginning with incremental scalability in mind. Engineering aspects of the implementation of the architecture have been considered at all levels so as to keep the cost of the system down and to make the system practical to build. The commercial potential of the architecture is also a factor in its design. The interconnection mechanisms are designed to be fairly independent of the processor technology being used. The system is in fact designed not only for incremental expansion but also for incremental upgrading to the latest fastest processors without discarding the existing processors.

A clocking scheme which allows for the desired properties was devised [Castro and Pose 94] and we now apply the new paradigm of self-tuning to a related scheme so as to optimize the interconnection system performance. Self-tuning was developed by Ted Kehl [Kehl 93] as a new clocking methodology borrowing heavily from both the synchronous and self-timed or asynchronous disciplines. This paper concentrates on the new self-tuning paradigm and its application to our distributed multiprocessor system.

2 Self-Tuning

Ivan Sutherland [Sutherland 86] said that synchronous systems rely on *faith* whereas self-timed systems rely on *measurement*. Synchronous systems have *faith* that a signal will move from one section of the circuit to another in a known time and without errors. To develop that *faith*, the designer does worst case timing analysis during design and provides a clocking period sufficient to accommodate these worst case delays. Worst case timing specifications are often derived from statistical analysis of component test data, so that one has very high confidence that a particular

component will meet its worst case specifications. Of course best case timing is very much faster than worst case timing, and on average, components tend to operate much closer to best case than to worst case. Thus to ensure that the system is guaranteed to work, one assumes the worst of every component. It is extremely unlikely that every component, or even that any component will operate at worst case timing, hence one pays a very big speed penalty in designing your system this way. Another way to look at it is that there is a very good chance that your conservatively designed synchronous system will run correctly with a considerably faster clock. In fact there is a small industry of *clock-chipping* kits, which are essentially kits to allow you to speed up your computer by putting in faster clock components and perhaps increasing cooling capacity. The approach taken with *clock-chipping* is to try a number of different clock speeds, choosing the fastest that appears to work correctly.

Self-timed or asynchronous systems do not rely on such *faith*. Instead they *measure* each data transfer, and data movement is not complete until it has been measured as being complete. Measurement is performed using completion detection circuitry. A round-trip signalling mechanism is required. First a request is sent, then completion of the operation is detected, and finally an acknowledgement takes place so that the next operation can proceed. The three parts of the round-trip signalling are done sequentially, and this slows the system down considerably, since the signalling itself often takes more time than the operation being performed. Sutherland concluded that small synchronous circuits are inherently faster than self-timed circuits, although one must beware of the enormous difference between best case and worst case timing in synchronous components.

Self-tuning is a new paradigm which combines the best features of synchronous and self-timed paradigms. As in self-timing, measurements are made, but instead of the wasteful round-trip signalling, one makes the measurements in parallel with the operation of the circuit and one does not wait for the result of the measurements before continuing. The basic principle behind self-tuning is that changes in circuit speed occur gradually relative to clocking speed. Synchronous circuits are based on the idea that circuits never change speed so that a very conservative estimate of the circuit speed must be used. Asynchronous systems do not assume anything about the circuit speed and instead signal the completion of the operations as they occur. Self-tuning makes the measurements as in asynchronous systems but defers any correction of the clock to future cycles. The results of the measurements are used in the future to *tune* the clock so as to match the actual operating speed of the circuitry. The self-tuning strategy only works if the clock speed doesn't change too rapidly relative to the clock speed. Ted Kehl [Kehl 93] has shown experimentally and argued logically that this is indeed the case for well designed circuits. Factors which influence circuit speed, such as temperature and voltage, simply do not change rapidly compared with modern clock speeds in the hundreds of MHz. Of course one must build the measurement circuitry using similar technology as the operational circuitry so that it

indeed tracks the speed of the circuitry being measured, and one must design robust power supply systems which maintain reasonably steady voltages.

There is an expectation that for a few hundred clock cycles at least, there is at most a very slight change in circuit speed and one should allow a little tolerance for such minor variations, but can still get very close to optimal operating speeds. During those few hundred clock cycles the speed of the circuit is measured and the clock is re-tuned to run faster or slower depending on the changes in circuit speed. Since any measurement circuitry will tend to slow down the system, they are kept out of the loop as much as possible. Depending on the system, it may even be possible to use software to do some of the measurements.

Of course there is the bogey-man argument, that says that what if the speed of the circuitry really did change quickly despite your expectations. While one cannot ever rule that out completely, neither can synchronous circuits absolutely guarantee that published worst case specifications really are worst case. There is a small chance that something will go wrong or that one has a faulty component. The strongest argument against those that worry about such an event is that almost all modern computer systems use clocks based on phase-locked loops. A phase-locked loop itself uses a feedback loop where corrections are applied based on measurements made some time earlier. Thus all modern computer systems have been relying on clocking systems based on such principles for many years, without any problems. What self-tuning does is extend this approach so as to optimize the complete system, not just generate a clock.

3 Measurement in Self-Tuned Systems

In order for self-tuning to be feasible it must be possible to measure the actual operating speed of the circuit, or at least to be able to detect whether it is speeding up or slowing down. One also needs to be able to adjust the clock, so normal crystal clocks are not used. Instead a simple ring oscillator arrangement with a variable number of tapped inverters is used.

Measurement takes place by looking for transitions in the data. If one is clocking too fast you will be clocking before the transition has taken place hence will not get the correct data. If one clocks too slowly the correct data will have been sitting there for a while before you get to use it, and this means the system is not running as fast as it could.

A very simple measurement strategy can be employed. The data line is sampled rapidly three times, typically by delaying the clock through a chain of inverters. The values sampled are compared to the incoming data, giving three equality checks. The last one is assumed to be correct. If the middle one is different to the last one then that means that a transition occurred in the time between when the middle one was sampled and when the last one was sampled, hence one should slow down the clock since there is very little margin remaining in the clocking. If the middle one is the

same as the last one, then we are not running any risk of clocking too quickly since the transition of the data must be happening well before the sampled data that we are using. There is however the possibility that we are going too slowly for optimal system operation. We can detect that by seeing if all three equality checks give the same value, since that would mean that any data transitions are happening at least two delays before the sample being used. In that case one can speed up the clock.

With this scheme one never needs to know the actual operating speed of the circuit. It will automatically adjust to allow the system to run at close to its maximum speed.

Another big advantage of self-tuned systems is that not only are they guaranteed to operate at close to the circuitry's actual operating speed, but because one is continually measuring the circuit's correct operation, one could also expect increased robustness.

In order to allow the vital measurements to take place, it is necessary to design self-tuned systems such that their correct operation can be detected in some way. For a communications system correct operation means transferring data correctly. Such a system is fairly simple in that it is relatively easy to design it such that the main possible source of errors is sampling the data before it has settled to its correct value. A simple scheme which detects transitions in the data as described above will suffice.

The self-tuning mechanism to be used in the Monash system is described later, after an outline of the system architecture is given.

4 Nodes

The basic node in the architecture is a combined processor-memory-switching element [Pose, Fazio, Wells 94]. There is only a single kind of node in the system. In fact there are no other active components in the system. The interconnection topology is implemented entirely through low cost passive interconnections between the nodes. A variety of interconnection topologies is possible with this approach with different performance tradeoffs.

The node comprises a number of subsystems including at least one processor unit, at least one memory unit, two parallel bus ports, a high speed serial connection and a switching scheme to route traffic between these elements. The processor unit can read and write the local memory unit and can direct memory references to memory units on other nodes through the bus ports or the fast serial link. The memory units can accept operations from the local processor unit, or from either bus port or the fast serial link. Facilities are also available to allow a processor to instruct a memory unit to copy data into another memory unit in a different node. The switching hardware handles the routing between these elements and indirectly through the larger interconnection topology. In effect the system functions as a distributed shared memory with a non-uniform access time. A node is depicted in figure 1.

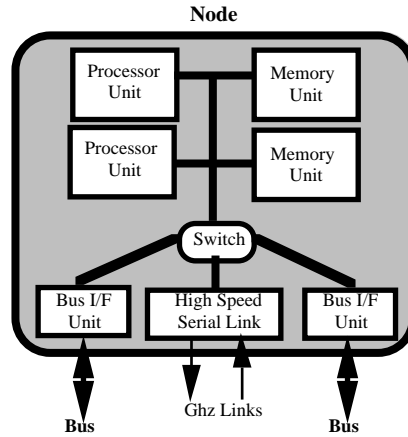


Figure 1. Structure of a Node

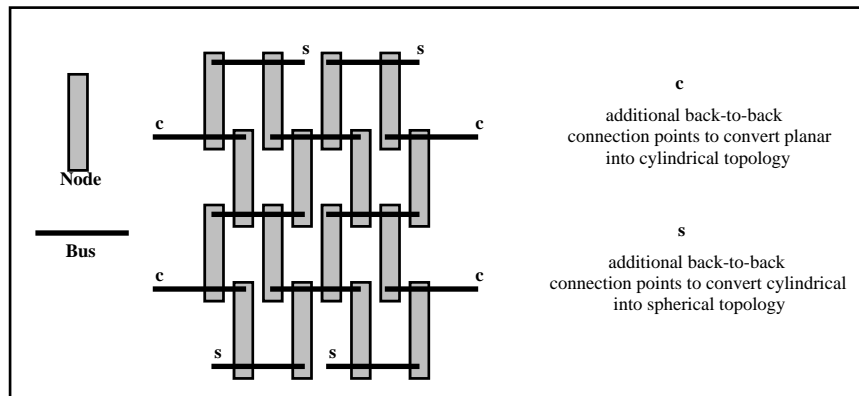


Figure 2. Planar 4 node per bus topology

Because there is only a single type of node the system is well suited to economical commercial mass production. A single node contains memory and processor on a single board and could form the basis for a conventional workstation-style computer sitting on someone's desk. A slightly larger system with a moderate number of nodes can sit in the corner of the room. A large system with over a thousand nodes could live in a machine room. Identical boards are used in each and extra boards may be added one at a time.

4.1 The Processor Unit

A node may contain one or more processor units. While a processor unit typically is based on a 64-bit RISC microprocessor, the only real requirements are that it be a

device which can generate addresses and data and accept data it may have requested. The communication protocol and mechanism used has to be consistent with that employed by other units in the node but need not be identical on every node. A processor unit may be implemented as a plug in module or daughter board to allow for easy maintenance and future upgrades.

Typically a processor unit is matched in speed and technology with the memory units in the same node however this is not essential as long as reliable communication is achieved. It is to be expected that modern high speed processor units will contain appropriate caches to minimize the impact of memory latency and to minimize the amount of memory traffic.

4.2 The Memory Unit

A node may contain one or more memory units. A memory unit typically will comprise a memory array, associated error detection and correction and control logic, and a controlling device which can be instructed to copy a block of data to another memory unit via one of the bus interfaces or over the high speed serial link. Obviously it must also be able to accept such a block of data from a bus interface.

In a node equipped with processor units based on modern 64-bit RISC technology, it would be expected that a memory unit would contain at least 64 Mbytes of memory. Logically though it must be a module which accepts an address, a function and perhaps some data, and may if required return some data. It must be able to communicate with both the bus interfaces and serial link and with the processor units on the node. As with processor units, a memory unit may be implemented as a plug in module or daughter board so as to allow upgrading to higher performance or larger memory sizes.

4.3 External Communications Interfaces

While a processor unit and a memory unit together make up a node which is the equivalent of a modern single board computer, the aim of this system is to allow such machines to be scaled into large multiprocessor systems. For this to be achieved there has to be a means of communicating between nodes. Two different kinds of external communications interfaces have been devised for this purpose. Reasons for having both mechanisms are discussed in the engineering section.

Two bus interfaces are provided on each node. These are 64-bit parallel bus interfaces with associated control logic and an implementation of a two level communication protocol for communicating with other nodes. The other end of the bus interface communicates with all the other units in the node, including the other bus interface and the high speed serial link. It is expected that large buffers would be included in the bus interface. These could act as a kind of write buffer for the other

units allowing them perhaps to continue with other activities while internode transfers are taking place.

Another key feature is that there is no common clock in the system as a whole. Our design philosophy of a single kind of active node interconnected passively precluded the conventional centralized clock distribution approach which anyway is rather difficult to implement in a large scale system. Each node runs at its own speed. This avoids a major clock distribution problem and allows upgrading to faster nodes on an individual basis rather than having to upgrade the whole system. It also means that each bus interface transmits data using its own clock since no other clock is available. By transmitting the clock along with the data bus interfaces can receive data using the clock at which it was transmitted. An architecture employing FIFOs as both buffering elements and synchronization elements has been devised [Castro and Pose 94].

The high speed serial link performs the same function as the bus interface units in that it allows units to communicate with units in other nodes. The technology employed in the high speed serial link is quite different. Instead of being based on a 64-bit parallel bus, the external interface of the high speed serial link is a pair of unidirectional 1 GHz serial connections over coaxial cable or fibre optic cables. Even at 1 GHz this link will be slower than the 64-bit parallel interfaces which could run at over 50 MHz, however the technology employed allows for different physical arrangements of the system as will be discussed later.

4.4 Switching within a Node

The key to our approach is that the switching component which dominates the traditional massively parallel machine has been distributed equally across all nodes and is a moderate overhead on each board which can even be ignored for a single node workstation.

The various units making up a node indeed need to communicate with one another. Because there are only a small number of units in a node it is not necessary to invest in large scale expensive switching apparatus. It is wise however to allow each of the units to operate to its capacity so a single bus interconnecting all the units is not appropriate. At the other extreme a small cross-bar switch interconnecting the units is unnecessary although not too unreasonable given the small number of units. A good performance tradeoff can be achieved by employing a small number of buses which allow the most heavily used inter-unit communications to proceed concurrently.

5 Interconnection Network Topology

The node forms a simple building block from which one can construct large systems interconnected in a variety of topologies. Given that the off-node interconnection

hardware has to be passive to preserve our goal of a single active component type, and that it must be possible to have practical implementations from the engineering viewpoint, we have devised a scheme which allows 6 different topologies to be implemented with differing characteristics.

5.1 Planar Topology with 4 Nodes per Bus

In this topology each node is connected to two buses, one for each bus interface. Every bus in the system has a maximum of 4 nodes on it. Remember that buses are just a passive set of parallel wires. A planar topology with 4 nodes per bus is depicted in figure 2.

5.2 Cylindrical Topology with 4 Nodes per Bus

While it is indeed possible for any node in the system to communicate with any other node by passing data through intermediate nodes it is clear that the path length and hence the communication time will be non-uniform. It is possible to reduce the maximum path length by rolling the plane into a cylinder. For instance if one examines the planar topology depicted in figure 2 one will see that there are buses on the left and right ends of the plane which have only 2 nodes on them. Logically one may combine these buses into single buses with 4 nodes each. This may be likened to rolling the plane into a vertical cylinder but is better realized from the engineering standpoint by producing a flattened cylinder by placing two smaller planar backplanes back-to-back.

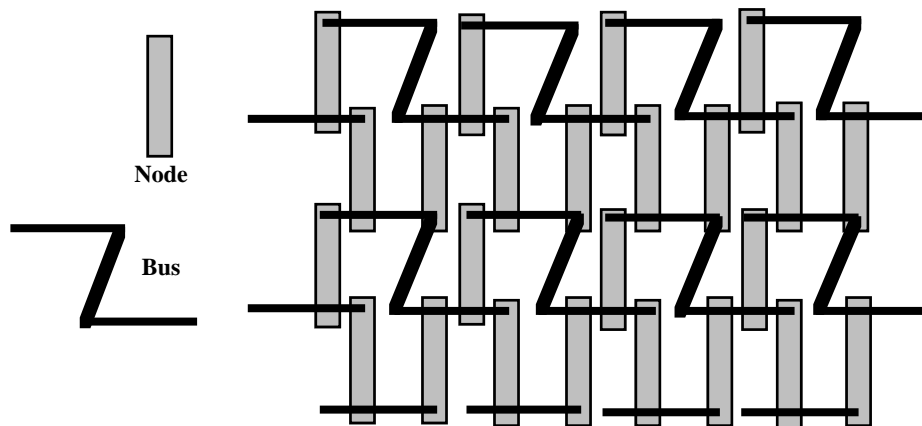
5.3 Spherical Topology with 4 Nodes per Bus

Just as the observation was made that the left and right ends of the planar topology were not fully populated, it is also evident that some buses at the top and bottom of the planar topology and the cylindrical topology are also not fully occupied. Logically these may also be combined by folding in the other direction and you actually end up with a kind of spherical topology. This also works out very well from the engineering viewpoint in that the back-to-back flattened cylinder construction also has the appropriate under occupied buses for a flattened sphere located back-to-back allowing a simple interconnection to combine them. For a given number of nodes a spherical topology has a shorter average path length than a cylinder, which itself has a shorter path length than a planar topology.

5.4 Topologies with 6 Nodes per Bus

While we indeed have reduced the maximum and average path lengths between nodes in going from a planar topology to a cylindrical one to a spherical one, it is possible to further reduce the average path length by increasing the number of nodes per bus. By

doing this one needs fewer buses for a given number of nodes. Of course one increases the traffic on each bus by doing this. A planar topology with 6 nodes per bus is depicted in figure 3.



Note that while this topology can also be connected into cylindrical and spherical arrangements, due to the asymmetrical bus shape one can get different numbers of nodes per bus around the edges of the backplane. It is even possible to end up with 8 nodes on some edge buses but this can be avoided by removing a node.

Figure 3. Planar 6 node per bus topology

Note that this depiction of a 6 nodes per bus topology indicates that the buses are bent. It is useful to employ such buses in our system since we do not want to violate our design principle of a single kind of node, and by arranging bent buses we can plug in nodes identical with those used in the 4 nodes per bus systems. It is also very convenient from the engineering perspective to use such bent buses. Careful comparison of figures 2 and 3 reveal that the vertical parts of the buses in figure 3 correspond to nodes in figure 2. In fact one can take the identical backplane structure that was used for the 4 nodes per bus topology and just replace every second node board with a passive set of wires linking the two buses. This may be achieved with a dummy node board only containing wires with no active components.

Just as in the case of topologies with 4 nodes per bus, it is also possible to fold the plane into a cylinder or sphere. In fact the identical structure of back-to-back backplanes can be employed. All six topologies may be created from common components. One just plugs in the nodes or wires to form the desired topology.

5.5 Another Dimension

Thus far we have only considered the use of the bus interface units to interconnect nodes. Certainly it may be reasonable to construct systems with 32 or 64 or even more nodes plugged into a backplane, however the path lengths will be increasing with the size of the system and loading on the buses could also become too great due to a lot of traffic just passing through.

We still have the high speed serial links running at 1 GHz. Conceptually we have the nodes arranged in planes, cylinders, or spheres, and we have engineered the system to have the node boards plugged into back-to-back backplanes. One could also stack these topologies in another dimension. While that would be difficult to arrange physically with backplane buses, the flexibility afforded by the use of long thin cables for the high speed serial links allows us to realize another dimension of interconnection. This may be visualized by considering the planar topology using a single backplane, and then stacking such backplanes and nodes above one another with the high speed serial links running vertically between the nodes which are directly above one another. The same approach works with cylindrical and spherical topologies since we are not physically constrained using the high speed serial links and can even geographically distribute the system to some extent. Consider the system to be layers of planes, cylinders or spheres.

A limitation of the technology used to build the high speed serial links restricts us to a single transmitter per link. This means that a bus structure is not possible. A single transmitter can send data to receivers on nodes vertically aligned in the conceptual topology. Thus a node can transmit to another node on every layer of this stacked topological structure. In order to ensure that the system is fully interconnected it is necessary to ensure that there is at least one transmitter on each layer. This means that the overall system can have no more layers than there are nodes per layer. It can however have fewer layers.

A fairly large system may well comprise 2048 nodes arranged in 32 layers of cylindrical 4 node per bus topology backplanes, each containing 64 nodes. Each node may well contain 2 64-bit processor units and memory units containing 128 Mbytes. Thus the system would contain 4096 processors able to read and write 256 Gbytes of memory.

6 Self-Tuning Applied to the Interconnection Network

As described so far we have a system of many small buses interconnected in a strange topology. Essentially the data is sent along with a clock pulse or strobe so may be thought to be self-timed. The real question though is how fast one can send the data. That will depend on the operating characteristics of the bus itself, the bus transmitter being used, and the bus receivers on all nodes connected to the bus, as well as the abilities of the nodes themselves to be able to deal with the bus transmissions. This is

complicated by the fact that the nodes themselves might be dissimilar in characteristics. It appears that this is an ideal case for self-tuning techniques.

We would like to be able to measure how fast the interconnection system is actually running without making too many assumptions about its characteristics or what is connected to it. In this case it is reasonable to assume that the speeds will not change dramatically over the short term, perhaps over several minutes or hours, so it may even suffice to have some partial software measurement system do this task.

What we have chosen to do in the first instance is to tune the system by having a node send test transmissions to each of the other nodes on a particular bus, starting at a very slow speed and increasing speed until something stops working correctly. In that way, the fastest transmission speed to each of the other nodes is determined, and the minimum of these speeds is optimal for that node to transmit on the bus. It may well be the case that different nodes will end up transmitting at slightly different speeds but that doesn't matter since the data is essentially self-clocked.

It is not clear how frequently one should re-tune such a system. This is one of the areas we are planning to investigate. What is clear is that it is advantageous for all nodes on a particular bus to be running at similar speeds, since a bus, being a shared medium, will effectively run at the speed of the slowest node.

7 Evaluation of the Interconnection Scheme

The interconnection scheme outlined above has several advantages over schemes employed in current massively parallel machines. We set ourselves a challenging aim in trying to achieve acceptable performance and yet not wanting to bear the costs of such performance in terms of complexity, scalability, and of course the actual expense of building the system. To that end we reduced the complexity down to a kind of simple switching structure which forms a small part of each node and which is controlled by an extremely cheap and simple routing system based on small look-up tables. The cheap incremental scalability is thus ensured as long as we can provide the required performance.

The conventional approach to achieving performance is to shorten the average and maximum paths that data has to traverse and to replicate the data paths so as to allow concurrent operation on many paths. By shortening the paths one reduces latency and by increasing concurrency one increases bandwidth. Existing systems have tended to take the approach of large and expensive switches to increase these parameters or else the use of various higher dimensionality networks such as hypercubes. Both these approaches lead to poor incremental scaling or poor fault tolerance.

We have indeed taken the maxims of increasing the degree of connectivity and increasing the degree of concurrency, however our methods are quite different. Taking the bus as the simplest multiple node interconnection scheme, we have recognized that it has limited bandwidth, and so as not to saturate it, we limit the

number of nodes on it to a small number, 4 or 6. To increase the degree of connectivity we have instead connected the node to two buses, and to a high speed serial link. In this way we have achieved our aim without overloading individual buses. As a side benefit this leads to a degree of fault tolerance. In a similar fashion we can use different topologies with various tradeoffs between bus loading and path length.

It is instructive to compare this topology with that of a mesh which has employed in many systems. A mesh typically has 4 or 6 external connections per node. Because of physical constraints this is only possible if the connections are not too wide. We have achieved an effectively higher degree of interconnectivity with a far simpler system. Using 4 nodes per bus we get each node connected directly to 6 other nodes; using 6 nodes per bus we get each board connected directly to 10 other nodes. That is ignoring the high speed serial links which could significantly increase that connectivity. It thus appears that we can achieve similar performance characteristics with lower cost and better scalability characteristics.

The self-tuning of the interconnection system allows us to have a far more flexible design without the restrictions of having to design for a particular clock speed, and allowing for newer, faster nodes to be added at any time. One could also add slower, cheaper nodes if desired.

8 Project Status

The system outlined in this paper is currently being designed and implemented in the Department of Computer Science at Monash University. A shared memory paradigm is being implemented although this allows for message passing algorithms with no overhead. Performance is related to the locality of memory references thus managing the locations of processes and their data is important. The project also includes a capability-based operating system which provides a persistent global virtual memory. This operating environment is based on the Password-Capability System [Anderson, Pose, Wallace 86] which ran on earlier generation shared bus multiprocessor hardware employing novel address translation techniques [Pose, Anderson, Wallace 87; Pose 89].

9 Conclusion

We have outlined a multiprocessor interconnection scheme with which a non-uniform memory access, distributed memory multiprocessor can be produced. It has superior properties with regard to incremental scalability compared with most competing systems. Attention to engineering considerations has led to an economical system with great commercial potential.

While the absolute performance of this system and its interconnection network will not reach the levels of some of the more elaborate schemes being used elsewhere, the simplicity and low cost of the approach should make the system very attractive in terms of performance per dollar and its incremental scalability and the ability to upgrade gracefully to faster processors adds to its attraction.

The use of the self-tuning paradigm is an ideal way to ensure optimal system performance while allowing for a very flexible design.

References

- [Anderson, Pose, Wallace 1986] Anderson, M.S., Pose, R.D., Wallace, C.S., A Password-Capability System, *The Computer Journal*, Vol. 29, No. 1, 1986, pp. 1-8.
- [Castro and Pose 1994] Castro, M. and Pose, R., The Monash Secure RISC Multiprocessor: Multiple Processors without a Global Clock, *Australian Computer Science Communications Vol. 16, No. 1, 1994*, pp. 453-459.
- [Kehl 1993] Ted Kehl, Hardware Self-Tuning and Circuit Performance Monitoring, *Proc. 1993 IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors, Cambridge Massachusetts, Oct. 3-6, 1993.*, pp. 188-192.
- [Pose, Anderson, Wallace 1987] Pose, R.D., Anderson, M.S., Wallace C.S., Implementation of a Tightly-Coupled Multiprocessor, *Australian Computer Science Communications Vol. 9, No. 1, 1987*, pp. 330-340.
- [Pose 1989] Pose, R.D., Capability Based Tightly Coupled Multiprocessor Hardware to Support a Persistent Global Virtual Memory, *Proc. 22nd Hawaiian Int. Conf. on Sys. Sci. Vol 2, 1989*, pp. 1-10.
- [Pose, Fazio, Wells 1994] Pose, R.D., Fazio, V., Wells, J., An Incrementally Scalable Multiprocessor Interconnection Network with Flexible Topology and Low-Cost Distributed Switching, *IEEE TCCA Newsletter, Summer-Fall 1994, Special Issue on High Performance Interconnection Networks*.
- [Sutherland 1986] Sutherland, I.E., A theory of Logical Effort, *Sutherland, Sproul and Associates Inc., #SSA 4679, 12 Sept. 1986*.