A T9000 Implementation of the p4 Parallel Programming Model

Stephen J. Turner and Adam Back
Department of Computer Science
University of Exeter, Exeter EX4 4PT England

Abstract. Although parallel computers have significant price/performance advantages over conventional sequential machines, they are still not widely used for general purpose computing. One of the main reasons for this is the lack of portability between different classes of parallel architecture, each of which has its own individual programming style. This paper discusses p4, a programming model for portable, heterogeneous, parallel computing, developed by the Argonne National Laboratory, and its implementation on the T9000 series of transputers. Experimental results are presented which demonstrate the performance of p4 relative to the raw performance of the T9000. These results are compared with similar experiments carried out with a T800 version of the p4 system. Since implementations of p4 exist on a wide range of parallel computers, the model provides a convenient route for transferring applications onto T9000 networks.

1. Introduction

Parallel computers have been available for many years, but are still not widely used for general purpose computing. Although they have significant price/performance advantages over conventional sequential machines, there has been a reluctance to move from sequential to parallel programming. One of the main reasons for this is the lack of portability between different classes of parallel architecture. Different machines can require different programming styles and this limits the lifetime of parallel software written for a particular architecture, which in turn inhibits investment in developing large parallel applications.

The p4 parallel programming system [3, 4], which originates from the Argonne National Laboratory, provides a portable programming model for a large range of parallel machines. Its predecessor was the system described in the book "Portable Programs for Parallel Processors" [2], from which p4 takes its name. The model combines message passing with shared memory to form a cluster based model of parallel computing. In this paper, we describe our work in porting the p4 system onto the T9000 series of transputers. The experimental results presented here not only show the relative overhead of using a system such as p4, but also provide an early indication of the performance of the T9000 transputer.

The p4 parallel library provides a programming model for portable, heterogeneous, parallel computing. In section 2, we describe the salient features of the p4 system and its suitability as a model for general purpose parallel computing. Some of the design issues of our implementation of p4 on both the T800 and the T9000 series are discussed in section 3, in particular the creation of p4 processes and the message passing mechanism. Experimental results are then presented in section 4 and these demonstrate the communications performance of p4 relative to the raw performance of the T9000. These results are compared with similar tests carried out using an earlier implementation of p4 on a T805 network [1]. Finally, in the conclusions, this work is discussed in the context of the recently published standard for a message passing interface (MPI) for distributed memory architectures [7].

2. The p4 Parallel Programming Model

2.1. Clusters

The cluster model in p4 groups together processes into clusters. All p4 processes can communicate via message passing, but only p4 processes within the same cluster can share memory. Thus a cluster forms a shared address space. The p4 parallel programming system has been implemented by other researchers on many different classes of parallel architecture: networks of workstations, distributed memory multicomputers, shared memory multiprocessors and virtual shared memory parallel computers.

With standard p4 on Unix workstations, the p4 process corresponds to a Unix process, and a cluster is formed by a set of processes sharing memory on the same processor. Communication between processes in the same cluster is implemented using shared memory, and between processes in different clusters using the socket mechanism. Similarly, on a distributed memory parallel architecture, the p4 processes within a cluster will all reside on the same processor. Message passing is implemented in this case using the underlying communications mechanism provided by the hardware.

With shared memory multiprocessors, the shared memory can be implemented in hardware between p4 processes which are running in parallel, as opposed to the time sharing of processes on a single processor Unix machine. With a virtual shared memory parallel computer, the virtual shared memory provided by the architecture can be used to allow processes in the same cluster to be placed on different processors. Thus with virtual shared memory, the choice of clusters could be based on the structure of the application, rather than on the characteristics of the hardware.

The p4 system provides synchronous and asynchronous message passing facilities between p4 processes. Messages may be typed: it is possible to request a message of a particular user defined type, and the p4 system will automatically buffer messages of different types until they are requested. Similarly, it is possible to request a message from a certain p4 process, and messages from other processes will be buffered until required. Within a cluster, the programming model includes monitors to control shared memory.

Facilities are provided to define a set of machines and a description of how clusters map on to those machines. This configuration is pseudo-dynamic in that it is specified at run time, but it must remain fixed after the p4 library has been initialised. There is also provision for dynamic process creation, although such processes are not able to communicate via message passing. A global barrier mechanism and a cluster barrier mechanism are also available, as are global reduction operations.

2.2. Heterogeneous Operation

The p4 system provides a framework within which it is possible to have a combination of machines of different classes presenting a common model. It allows us to execute programs on a combination of workstations, distributed memory parallel machines and shared memory multiprocessors. The p4 library provides heterogeneous operation

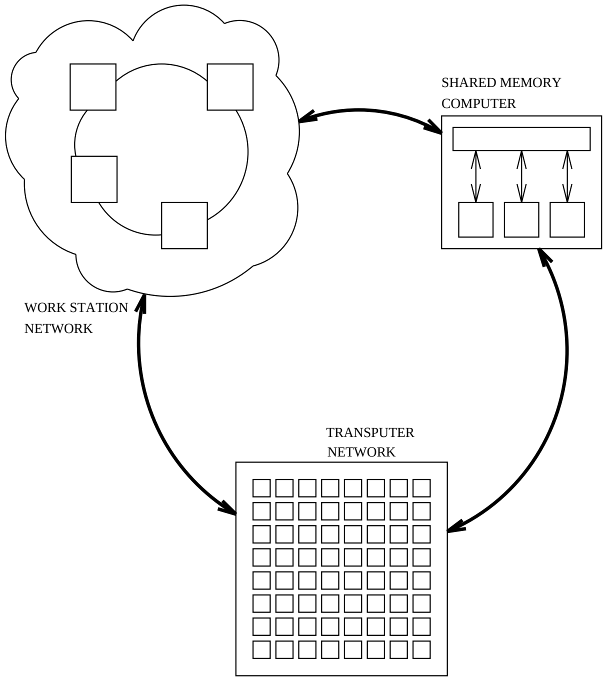

Figure 1: Heterogeneous Computing

across these architectures, so that a single p4 program may be run on a combination of machines simultaneously. Some clusters could be located on Unix workstations, some on the nodes of a transputer network and some on shared memory multiprocessors.

To allow messages to be passed between machines of different architectures, p4 uses the XDR (eXternal Data Representation) library. XDR [12] provides a standard representation for float, double, int, long into which messages must be translated on send and from which they must be translated on receive. In this way messages can be passed between little-endian and big-endian machines, and between machines which do and do not use the IEEE floating point format.

Our transputer implementation is thus able to operate in a heterogeneous manner with Unix workstations, and shared memory multiprocessors, as in figure 1. Transparently to the p4 applications programmer, communication in our example network will be taking different paths depending on the location of the p4 processes involved:

- Transputer ↔ Transputer, via transputer hardware links,
- Transputer ↔ Unix machine, via transputer socket library and Unix sockets.
- Unix machine ↔ Unix machine, via Unix socket library.

3. The Transputer Implementation

3.1. Process Creation

In implementing p4 processes on a transputer architecture, there is a problem in associating a local data area with each p4 process. This local data area is required to hold the p4 process id, XDR data buffers and other data which is required on a per process basis. This is not a problem in the standard Unix implementation of p4 as the Unix *fork()* is used to spawn p4 processes. The *fork()* copies both the data segment and the heap segment of the parent process. In this way, the child process has its own copy of all global variables, and all program data which is stored on the heap.

The transputer process creation functions are implemented in hardware and are orders of magnitude more lightweight than the standard Unix *fork()*. Process creation on transputers does not involve copying the data segment or the heap segment. The problem of finding a place to store local per process data in the transputer process environment is solved efficiently by using an aligned workspace area.

A workspace pointer is used by a process as its stack pointer, and varies according to the current depth of function invocation. The workspace pointer also has a second function as a process identifier. When a process is descheduled by the hardware scheduler its workspace pointer is used to identify it. Negative offsets from the workspace pointer are used by the scheduler and some transputer instructions, to form a linked list of processes and to store information about the process while it is descheduled.

A process can determine the value of its workspace pointer using a single CPU instruction. By allocating a p4 process's workspace at an aligned address, it can determine the start of its workspace easily. The p4 per process data is then stored at the start of the p4 process's workspace, while the process's stack will grow down from the top of the workspace with the workspace pointer pointing to the stack top. In this way, we are able to retain the transputer's advantage of fast dynamic process creation.

Our current version of p4 thus differs from the generic Unix implementation in that neither the data segment nor the heap segment are replicated. These restrictions do not present a problem as far as many p4 applications are concerned. For p4 programs which assume such replication, a pre-processor has been implemented using NewYacc [16] which groups together all global and static data items into a structure, and modifies all accesses to these data items to go indirectly through a pointer to this structure. In this way, when a process is created it is possible to take a copy of its data segment by copying the "data segment" structure.

This pre-processor was originally intended for p4 applications which run on the T800 transputer. Although it can also be used with the T9000 version, we are currently investigating how it might be possible to take advantage of the protection and memory management features of the T9000 [15] to provide improved support for segments.

3.2. Message Passing

The generic Unix implementation of p4 uses the socket mechanism for communication. On parallel machines with special purpose communication hardware, vendor specific communication libraries are normally used. In the T800 version of p4, we implemented the p4 message passing calls using the Virtual Channel Router (VCR) of the Inmos D4314A ANSI C toolset [9]. This provides virtual channels which can be placed between processes on any processor in a transputer network, the necessary through-routing and multiplexing being performed by the VCR software.

In order to implement p4's point to point communication in terms of virtual channels we provide a configuration which connects each processor to each other processor in the network. For each virtual channel, there are message handling processes at each end which run at high priority. The message handler for a particular channel will forward all messages destined for the remote processor via the virtual channel. Further details of the implementation of the message passing mechanism may be found in [1].

On the T9000, virtual channels are multiplexed onto the physical links of the transputer network by the on-chip virtual channel processor. In conjunction with the C104 router, this provides virtual channel routing by hardware. We were able to transfer our implementation of the p4 message passing routines onto the T9000 without any modification using the Inmos T9000 (D4394) toolset [10]. However, it is possible that improvements in the efficiency of the message passing routines could be obtained by making use of some of the specific features of the T9000 such as resource channels [11].

3.3. Priority Scheduling

This is an extension to the p4 model which was designed to enable us to use p4 as a portable base for our research into the use of optimistic parallel execution mechanisms in general purpose computing [14]. This has the requirement that it must be possible to control the scheduling of processes at run time. We wished to be able to do this in a lightweight process environment, but at the same time we did not wish to lose the portability of our system. In an attempt to satisfy both of these goals we designed the following extension to p4.

We provide a lightweight process library where each process has a priority level which determines the scheduling. The library provides facilities to suspend, resume, and kill processes. It also allows the priority of each process to be altered at run time. Processes with a priority level equal to that of the highest runable process are executed using a round robin policy. If at any time a higher priority process becomes ready, the lower priority process is pre-emptively descheduled.

On the T800, a multi-priority scheduler with the capability of pre-emptively descheduling processes has been constructed based on the work of Shea *et. al.* [13]. The library is able to pre-emptively schedule and deschedule processes within two transputer time-slices. That is, if a higher priority process becomes ready while a lower priority process is running, the lower priority process will run for at most two time slices before being descheduled.

Whereas it is very difficult to manipulate the transputer's own scheduler queue safely on the T800, the T9000 offers improved access to the queue and instructions to control the scheduling of processes. In particular, an atomic operation is provided to change the front and back registers of an active list as a pair. This allows the easy implementation of a true pre-emptive scheduling mechanism.

4. Experimental Results

4.1. Communications Performance

A number of experiments have been carried out to test the communications performance of p4 on T9000 transputers, using a PARSYS SN9400 SP system. It is important to note that these tests use pre-production T9000-Gamma silicon, with a clock speed of 20 MHz, well below that expected from production quality devices, and a link speed of 100 Mbits/sec. The figures for the T805 system were obtained with 25 MHz transputers and a link speed of 20 Mbits/sec.

Figure 2 demonstrates the efficiency of the p4 system as compared to the raw performance obtained with the T9000 transputer. These times are obtained with a simple "Ping" application where a fixed-size message is sent from one T9000 transputer to an adjacent transputer and then back to the originator. This is repeated a large number of times in order to obtain an accurate measurement, with the average round trip time in $\mu secs$ plotted for different message sizes. A slight discontinuity can be seen as the message size becomes greater than 32 bytes, as an extra packet is then required to transmit the message.

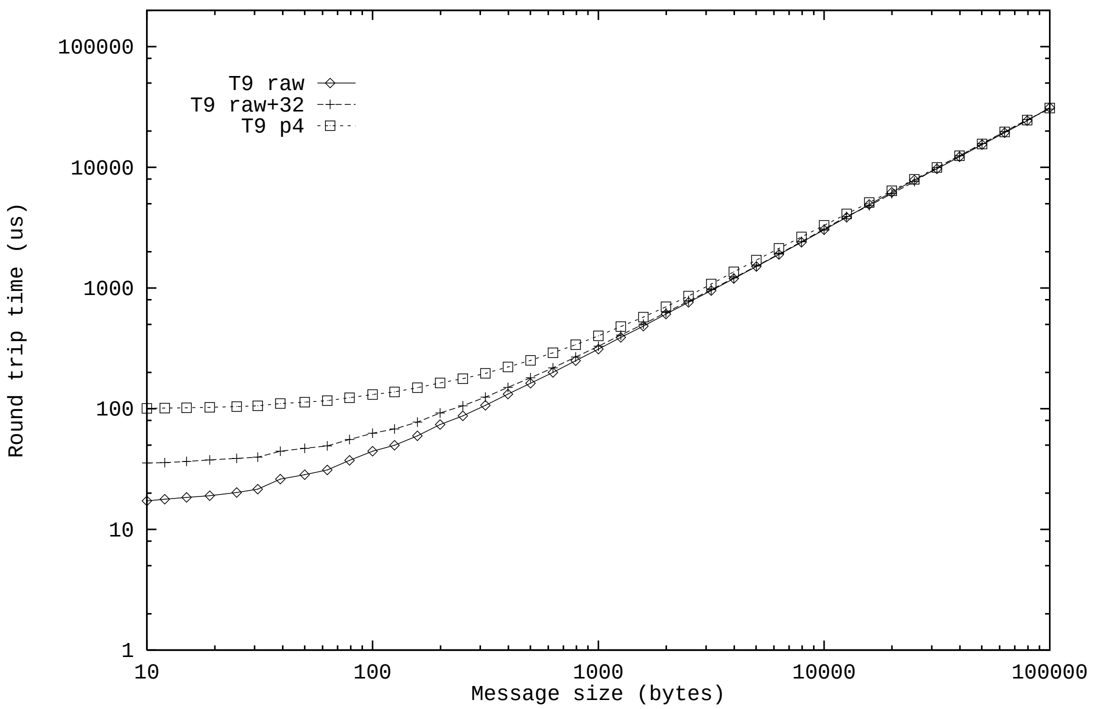

Figure 2: Round trip time T9000 ping

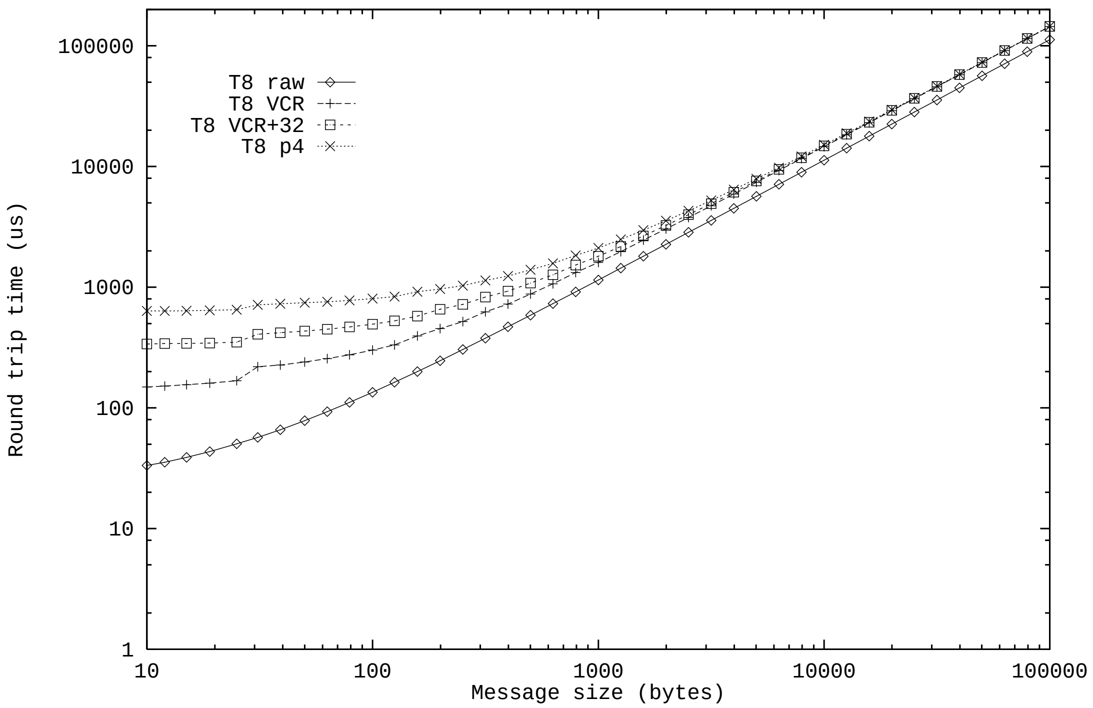

Figure 3: Round trip time T805 ping

In p4, a 32-byte header is first sent as a separate message and contains information about the size of the actual message, the type, sender's p4 process id, whether XDR (eXternal Data Representation) is to be used etc. To distinguish the overhead p4 incurs for this header from the overhead due to buffer management and the provision of typed messages, the line "T9 raw+32" shows the performance obtained with raw communications when a 32 byte header is sent as a separate message before the actual message.

Figure 3 shows the corresponding round trip times obtained with the T805 transputer. Here, the lines labelled "VCR" and "raw" show the performance of fixed sized messages with and without the Inmos VCR communications software [9]. The performance difference between VCR and raw messages is due to packetization and multiplexing of multiple virtual channels on a single hardware link. It can be seen that the VCR software itself incurs a significant overhead on the T805 system and there is a noticeable discontinuity as the packet size is exceeded.

The advantages of the virtual channel processor on the T9000 can be clearly seen in figure 4, which shows the ratio of T8 to T9 round trip times. For smaller messages, the T9000 raw communications (with VCR in hardware) can be up to 10 times faster than using the software VCR on the T805. The line in figure 4 representing the ratio for the p4 system shows a factor of between 6.8 and 4.5 in the round trip times. Further improvements can be expected when production quality T9000 silicon becomes available.

Figure 5 shows the throughput in Mbytes/sec for the Ping application on both T8 and T9 systems. The throughput is calculated as the message size divided by half the round trip time. For small messages, the throughput is significantly reduced by the send and receive overheads on each processor, but for large messages these overheads become negligible in comparison to the total time taken and the throughput should be close to the maximum unidirectional bandwidth of the link.

On the T9000, p4 almost achieves the throughput of raw communications for very large messages, although this is less than the theoretical figure of 9.55 Mbytes/sec that might be expected for a 100 Mbits/sec link [11]. Note that with the Ping application, there is only one virtual channel (in each direction) mapped onto the physical link connecting the two transputers. To achieve maximum bandwidth with a T9000 system, it is desirable to map a number of virtual channels onto the same physical link [8]. This would happen with a more complicated p4 application.

On the T805, the p4 throughput is limited by that of the software VCR, which itself is rather less than the throughput achieved with raw communication. Here the line for raw communication does become close to the maximum unidirectional bandwidth of 1.7 Mbytes/sec for large messages.

In the experiments so far discussed, the transputers concerned have been connected with a direct link. The last two figures analyze the performance of T9000 communications when messages are sent via a C104 router. It can be seen from figures 6 and 7 that sending via a C104 loop increases the round trip time and decreases the throughput. The effect is more noticeable for larger messages: in fact, for messages larger than 600 bytes, the overhead of routing via the C104 exceeds that incurred by the p4 system.

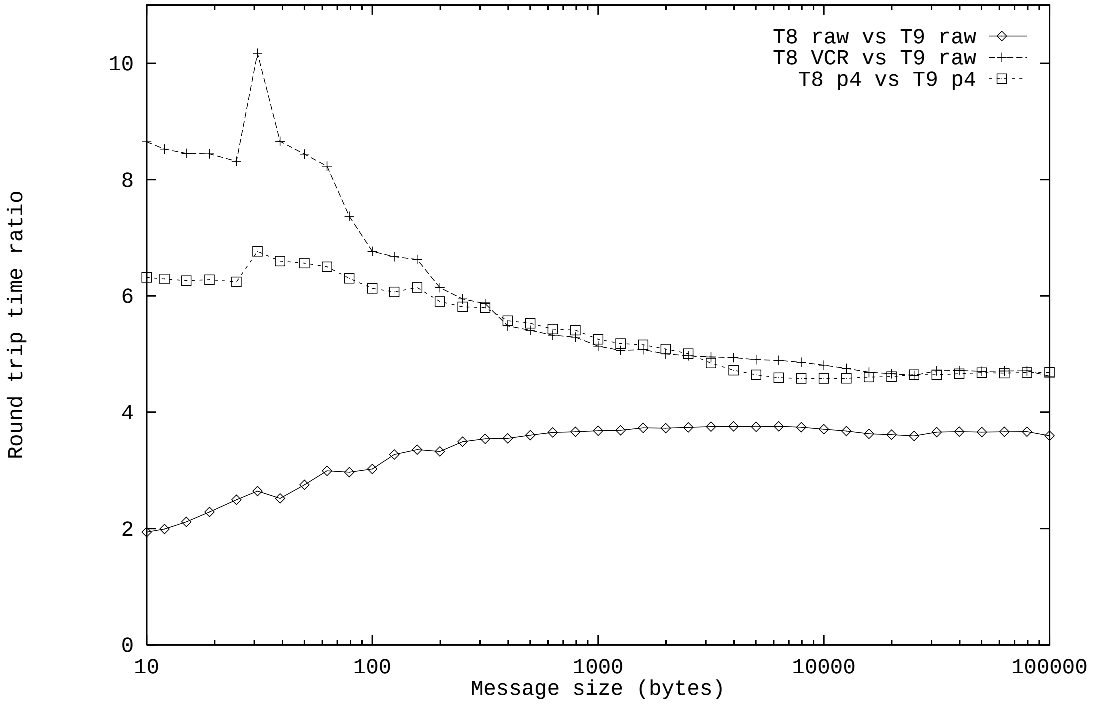

Figure 4: Round trip time ratio T805 vs. T9000

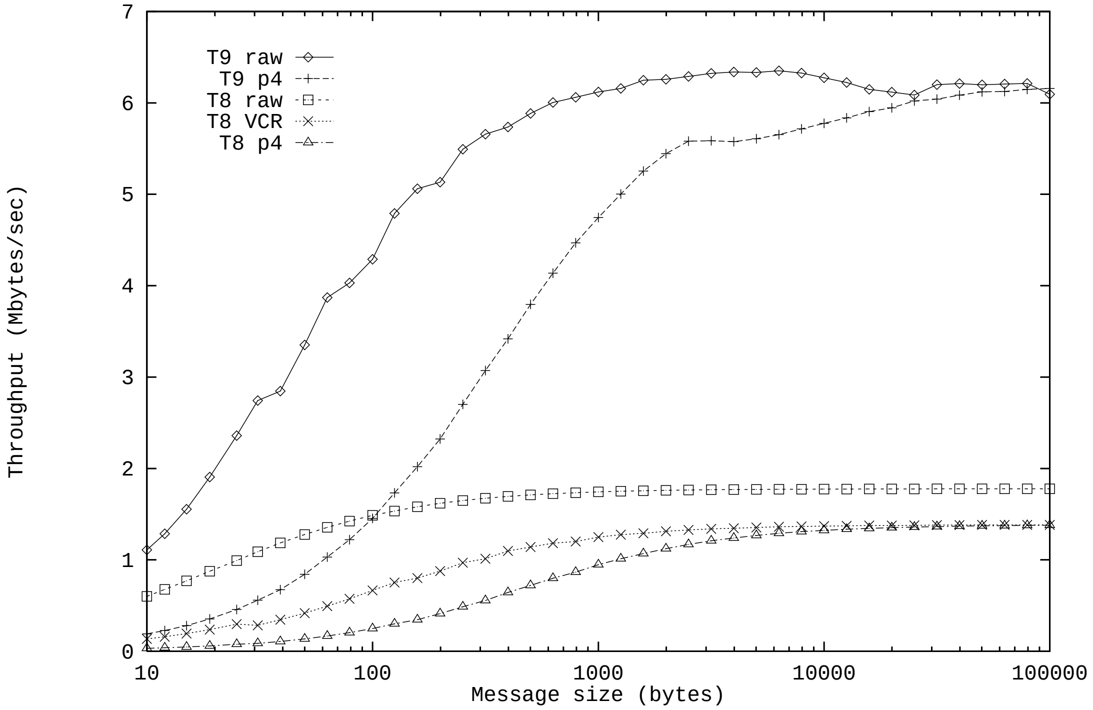

Figure 5: T805 and T9000 Throughput

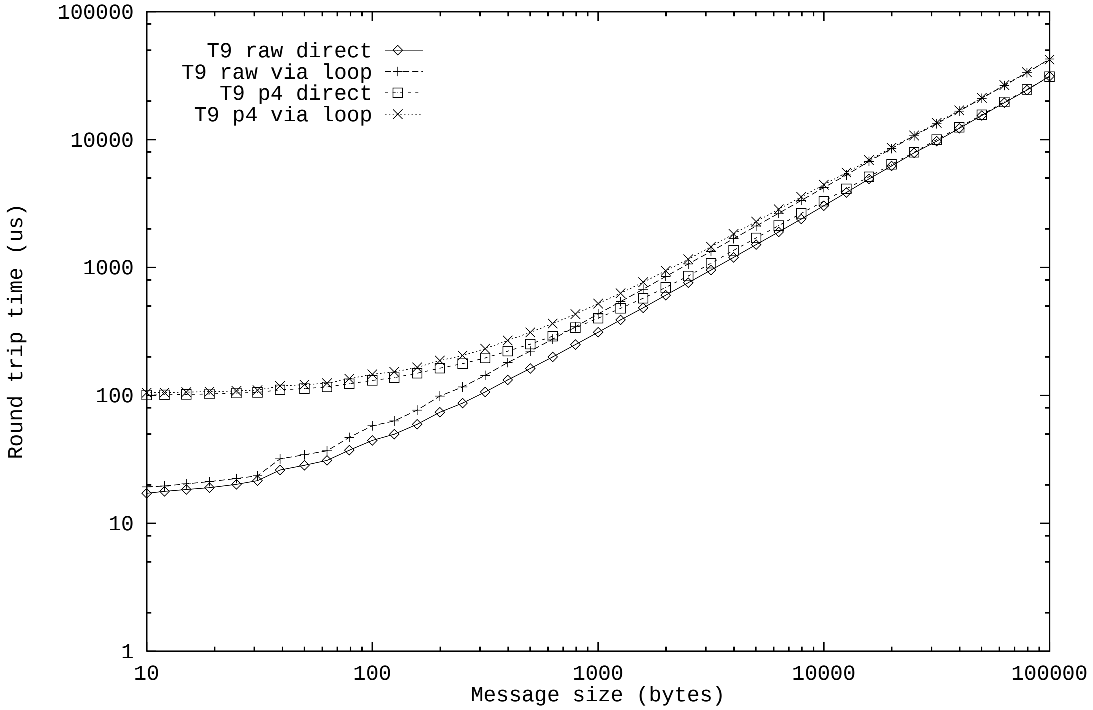

Figure 6: T9000 Round trip time via C104 loop and direct

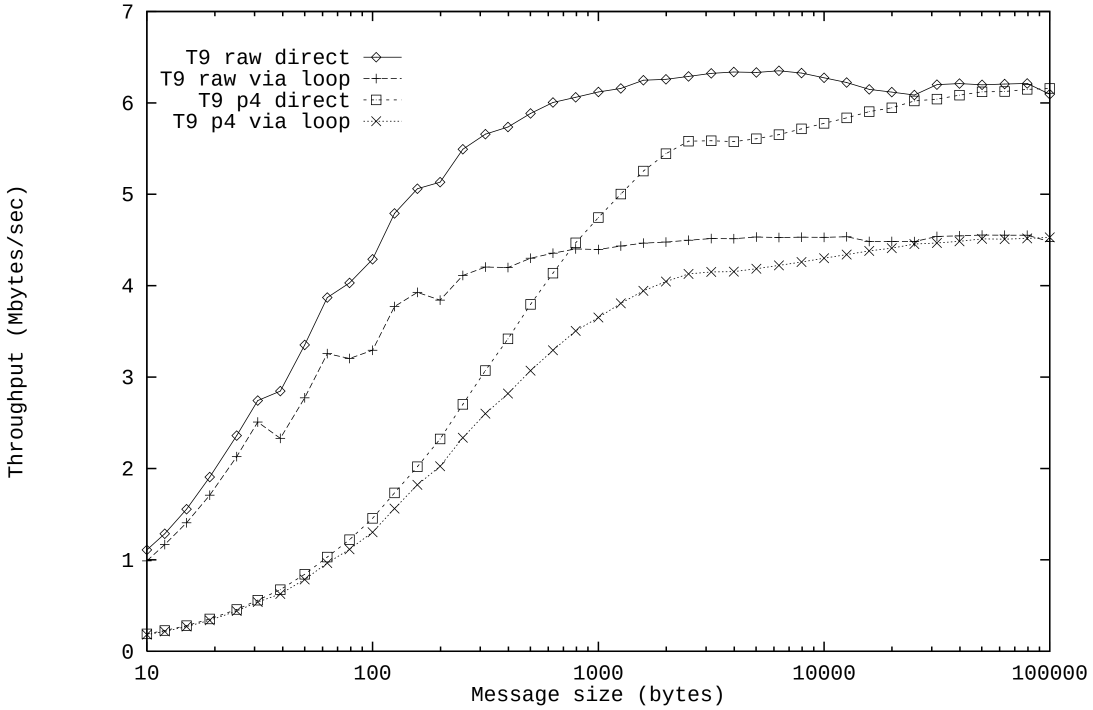

Figure 7: T9000 Throughput via C104 loop and direct

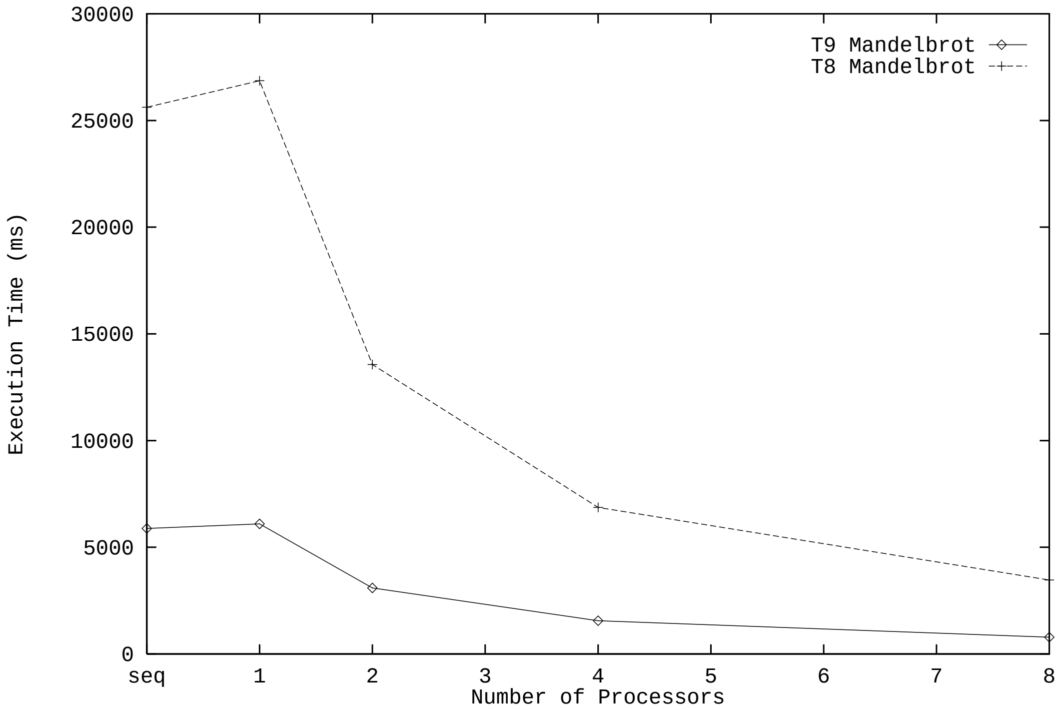

Figure 8: T805 and T9000 Mandelbrot Set

4.2. Processing Power

In order to measure the processing power of the T9000, a compute intensive application was chosen. A parallel Mandelbrot Set program was written using the p4 system, organised as a processor farm. As the necessary through-routing is carried out entirely by the p4 system, it is possible to organise the p4 processes in a way that matches the algorithmic topology of the application [5]. In order to keep the processors busy, it is necessary to buffer an extra item of work for each processor, and this buffering is also provided automatically by the p4 system using asynchronous message passing.

Figure 8 shows the execution times in *msecs* for the Mandelbrot Set application on both T805 and T9000 transputer networks. The two points on the vertical axis are the times for a purely sequential version of the application. It can be seen that the 20 MHz T9000 processor used here is approximately 4.4 times faster than the 25 MHz T805 for this application. The Mandelbrot calculation which forms the basis of this experiment was coded using double-precision floating-point arithmetic.

A more detailed analysis of the T9000 performance can be obtained by examining the code generated for the sequential Mandelbrot Set application. In the inner loop of the Mandelbrot calculation, there were found to be 39 integer instructions and 11 floating-point instructions. (The 13 floating-point load and store instructions are treated here as integer instructions). The time column of table 1 is the total execution time for the application. By counting the total number of iterations for all pixel positions, it is possible to obtain a Mflops figure which is shown in the last column of table 1. The third line of this table shows the projected figure for a 50 MHz T9000 and the last two lines, the figures for an i486 and SGI R4000 processor respectively.

A 20 MHz T9000 should have a peak rate of 10 Mflops and a sustained rate of approximately 50% of the peak rate. These figures are therefore a little disappointing,

Table 1: Sequential Mandelbrot Set using floating-point arithmetic

Processor	Instr.		Time (ms)	Mflops
	int	fp		
25 Mhz T805	39	11	25621	0.86
20 Mhz T9000	39	11	5880	3.8
50 Mhz T9000?	39	11	2352	9.4
66 Mhz i486	18	11	5630	3.9
100 Mhz R4000	8	11	1056	21.3

Table 2: Sequential Mandelbrot Set using integer arithmetic

Processor	Instr.	Time (ms)	Mips
25 Mhz T805	65	31613	4.6
20 Mhz T9000	52	5587	20.9
50 Mhz T9000?	52	2234	52.1
66 Mhz i486	32	5480	13.1
100 Mhz R4000	32	1734	42.0

although some improvement can be obtained by loop unrolling. Note however that the T9000 compiler used for these tests is also a pre-production version and it is likely that the final version of the compiler will generate more optimal code.

Table 2 shows a similar set of figures for a purely integer version of the Mandelbrot set application, again using a sequential algorithm. Note that this test involves the extensive use of integer multiply, which may not be typical of many applications. The third line of this table is again a projection for a 50 MHz T9000. A 20 MHz T9000 should have a peak rate of 80 Mips and a sustained rate of 40 Mips and again it is likely that better figures will be obtained when a production version of the T9000 C compiler becomes available.

5. Conclusions

This paper has shown how the cluster based model of parallel computing, as provided by p4, presents the programmer with a uniform model which enables the development of efficient, portable applications that can be run in a heterogeneous environment. Since implementations of p4 exist on a wide range of parallel machines, new applications can be developed which are not restricted to the transputer or any other particular architecture. Moreover, p4 also provides a convenient route for transferring existing applications from other parallel computers onto T9000 networks.

The experimental results presented in this paper are provisional and further tests need to be performed and analyzed, particularly when production quality versions of the T9000 processor and compiler become available. Nevertheless, some interesting conclusions can be drawn. It can be seen that the communications overhead of p4 is not unreasonable, particularly for larger messages, in view of the additional functionality that p4 provides. Indeed for moderate size messages of between 500 and 1000 bytes, the overhead incurred by p4 is similar to that of sending a message via a C104 router rather than a direct link.

Message passing paradigms such as p4 are widely used on parallel machines, but the lack of a standard has impeded the development of portable software and libraries for message passing machines. Recently, a new standard message passing interface (MPI) has been published [6, 7], which is intended to replace the machine specific and other currently used interfaces. Its development has been strongly supported by many of the major vendors and it seems likely to be adopted for a wide range of systems. As a portable MPI implementation is available which runs on p4, it is hoped that our work will provide a short-cut to the implementation of MPI on transputers.

The transputer implementation of p4 described in this paper may be obtained by anonymous ftp from `atlas.ex.ac.uk (144.173.14.11)`. The current version is in the directory `pub/parallel/software/p4`, in the file `beta-1.2.tar.gz`.

References

[1] A. Back and S. J. Turner. Portability and Parallelism with Lightweight p4. In *Proc. BCS PPSG Conference on General Purpose Parallel Computing*, 1993.

[2] J. Boyle, R. Butler, T. Disz, B. Glickfeld, E. Lusk, R. Overbeek, J. Patterson, and R. Stevens. *Portable Programs for Parallel Processors.* Holt, Rinehart, Winston, 1987.

[3] R. Butler and E. Lusk. User's Guide to the p4 Parallel Programming System. Technical report, ANL-92/17, Argonne National Laboratory, 1992.

[4] R. Butler and E. Lusk. Monitors, Messages, and Clusters: the p4 Parallel Programming System. Technical report, Argonne National Laboratory, 1993.

[5] M. Debbage, M. B. Hill, D. Nicole, and A. Sturges. The Virtual Channel Router. *Transputer Communications*, 1(1):1–16, 1993.

[6] J. J. Dongarra, R. Hempel, A. J. G. Hey, and D. W. Walker. A proposal for a User-Level Message Passing Interface in a Distributed Memory Environment. Technical report, ORNL/TM-12231, Oak Ridge National Laboratory, 1993.

[7] MPI Forum. MPI: A Message-Passing Interface Standard. Technical report, University of Tennessee, 1994.

[8] A. Hipperson. The Virtual Link Communications Performance of Early Pre-Production T9000s. In *Proc. 7th PARSYS User Group Meeting, Oxford.* PARSYS, 1994.

[9] Inmos. *D4314 ANSI C Toolset User Guide*, 1993.

[10] Inmos. *T9000 ANSI C Toolset User Guide*, 1994.

[11] M. D. May, P. W. Thompson, and P. H. Welch. *Networks, Routers and Transputers.* IOS Press, 1993.

[12] SUN Microsystems. *Network Programming Guide*, 1990.

[13] K. M. Shea, M. H. Cheung, and F. C. M. Lau. An Efficient Multi-priority Scheduler for the Transputer. In *Proc. 15th WoTUG Technical Meeting (Aberdeen)*, pages 139–153. IOS Press, 1992.

[14] S. Turner and A. Back. General Purpose Optimistic Parallel Computing. In *Proc. 7th PARSYS User Group Meeting, Oxford.* PARSYS, 1994.

[15] University of Kent. *T9000 Systems Workshop*, 1992.

[16] E. L. White, J. R. Callahan, and J. M. Purtilo. The NewYacc User's Manual. Technical report, University of Maryland, 1989.