Behavior of the Butterfly™ Parallel Processor in the Presence of Memory Hot Spots

Robert H. Thomas

BBN Laboratories Incorporated
Cambridge, MA 02238

Abstract

This paper describes a set of experiments designed to measure the behavior of the Butterfly Parallel Processor in the presence of memory "hot spots". The experiments were motivated by a paper by Pfister and Norton [3] that reported results from simulation studies on multistage switching networks for shared memory parallel processors. Their results indicated that, for machines with a large number of processors, very slight non-uniformities in memory reference patterns can lead to severely degraded performance for the entire machine, including processors that avoid referencing the hot memories. The results were explained in terms of a phenomenon called "tree saturation" where traffic to the hot memories backs up into the switch and interferes with other switch traffic. The experiments reported here show that those results do not generalize to the Butterfly Parallel Processor. The access time for a memory that contains a hot spot is degraded, but the presence of the hot spot has little effect on the performance of programs that avoid the hot memory. Furthermore, tree saturation does not occur in the Butterfly Switch.

1. Introduction

This paper describes a set of experiments that measure the behavior of the Butterfly Parallel Processor [2] in the presence of memory "hot spots". The experiments were motivated by a paper on memory hot spots by Pfister and Norton [3] that presented results of simulation studies of the switching network for RP3, a research parallel processor being developed at IBM Yorktown Heights.

The simulation results showed that non-uniformities in memory reference patterns, which make certain memories "hot", can have a devastating effect on the performance of an entire machine, including processors that avoid referencing the hot memories. Pfister and Norton explained their results in terms of a phenomenon called "tree saturation", where traffic to the hot memories backs up into the switch and interferes with other traffic, including that to non-hot memories. Their results indicated that for machines with a large number of processors (>=100) even slight non-uniformities in reference patterns can lead to tree saturation and severely degraded performance for the entire machine.

Pfister and Norton claim generality for their results, stating that they apply to all multistage blocking networks. Furthermore, their paper claims that attempts to avoid the problem, such as providing multiple paths through the network, do not really help. Finally, the results are used to motivate the use of a second, combining switch in the RP3 architecture.

The switching networks studied by Pfister and Norton were multistage shuffle exchange switches similar in topology to the switch used in the Butterfly Parallel Processor. However, there is one key difference in switch operation: the switches studied were "blocking", whereas the Butterfly Switch is not. In a blocking switch, when contention for an output port of a switching element occurs, the path from the source to that switching element is held until the desired output port can be obtained. When contention for an output port occurs in a non-blocking switch, the message encountering the contention is rejected (to be retransmitted later) and switch resources associated with it (i.e., the path to the point of contention) are released. When the message is retransmitted, it again competes with other messages for switch resources.

Thus, like the switches studied by Pfister and Norton, the Butterfly Switch is multistage. However, unlike them, it is non-blocking. Because the Butterfly Switch is non-blocking, the behavior of a program on a Butterfly system can be expected to be less severely affected by non-uniformities in memory reference patterns (caused either by the program itself or by other programs on the machine).

Nonetheless, obvious questions to ask are: how does the Butterfly Parallel Processor perform in the presence of memory hot spots? Does it exhibit tree saturation? Does the architecture break down in large configuration for programs whose memory reference patterns exhibit moderate or even very slight non-uniformities?

The experiments described below show that the results presented by Norton and Pfister do not generalize to the Butterfly Parallel Processor. The access time for a memory that contains a hot spot is degraded, but the effect of switch contention is very small, even when severe non-uniformities in memory reference patterns are present. The experiments indicate that tree saturation does not occur in the Butterfly Switch.

2. The Butterfly Parallel Processor

This section presents enough information about the Butterfly Parallel Processor to understand the experiments described in this paper. More information about the Butterfly machine can be found in [2].

The Butterfly Parallel Processor is composed of processors with memory and a multistage switch that interconnects the processors. A Butterfly system can be configured with from 1 to 256 processors. One processor and memory are located on a single board called a Processor Node. All Butterfly Processor Nodes are identical. Collectively, the memory of the Processor Nodes forms the shared memory of the machine. All memory is local to some Processor Node; however each processor can access any of the memory in the machine, using the Butterfly Switch to make remote references. From the point of view of an application program, the only difference between references to memory on its local Processor Node and memory on other Processor Nodes is that remote references take a little longer to complete. (The typical memory referencing instruction takes about 6 microseconds when the data referenced is remote and about 2 microseconds when it is local.) The speeds of the processors, memories, and switch are balanced to permit the system to work efficiently in a wide range of configurations.

Each Butterfly Processor Node contains a Motorola MC68000 microprocessor (or a MC68020 with a MC68881 floating point co-processor), at least 1 MByte of main memory, a co-processor called the Processor Node Controller, memory management hardware, an I/O bus, and an interface to the Butterfly Switch. I/O connections can be made to each Processor Node, making I/O configuration very flexible.

The Butterfly machine supports a very efficient operation for transferring blocks of data from one Processor Node to another. The block transfer operation is implemented by Processor Node Controller microcode. Once initiated, a block transfer occurs at the full 32 MBit/second bandwidth of a path through the Butterfly Switch.

3. The Experiments

Two experiments were conducted to measure the performance of the Butterfly Parallel Processor in the presence of hot spots. The objective of the first experiment was to time execution of a typical program, first in an environment without any hot spots, and then in one where N processors were used to generate a hot spot. A matrix multiplication benchmark program [1] was chosen. The objective of the second experiment was to determine the effect hot spots have on typical memory references

by systematically measuring the behavior of the machine under non-uniform memory reference patterns. This was done by timing remote read, write, and block transfer operations for various memories, first in an environment without any hot spots, and then in an environment where N processors were used to generate a hot spot.

Hot spots were generated in two different ways:

1. Via read and write references. N processors were used to make a given memory hot by reading and writing the same location in that memory. This was accomplished by having each processor execute the tight loop:

```
for (i = 0; i < count; i++)
    * hotmemp = * hotmemp;
```

   where *hotmemp* is a pointer (*short* *) to a location in the hot memory.

2. Via block transfer. N processors were used to make a given memory hot by using the block transfer operation to copy data from that memory to their local memories. This was accomplished by having each processor execute the tight loop:

```
for (i = 0; i < count; i++)
    Do_bt (hotmemp, localp, numbytes);
```

   where *Do_bt* initiates a block transfer that moves *numbytes* bytes from the location beginning at *hotmemp* in the hot memory to the location beginning at *localp* in the processor's local memory.

The difference between these two methods is in the duration of the switch messages they generate. Simple read and write references use the switch in 2 microsecond bursts. Each iteration of the loop generates 3 messages, 2 for the read and 1 for the write. Block transfers are broken into 256 byte packets, each of which uses the switch in 64 microsecond bursts. Each iteration of the block transfer loop generates 2 messages for each packet, a short request message and a 64 microsecond response message.

Although all Processor Nodes in a Butterfly system are functionally equivalent, there is a distinguished King Node that is special in two ways: it is the node to which the console terminal is connected; and it controls the machine while the operating system is being booted. Because a terminal handler and window manager run on the King Node, it appears about 8%-10% slower than the other nodes to application programs. To ensure that the measurements were not affected by the processing requirements of the terminal handler and window manager, the King Node was avoided in both experiments.

The experiments were run on a 128 processor Butterfly system. When the experiments were run, 16 processors had been temporarily removed to configure

several smaller systems, leaving 112 processors in the system. Since the King Node was not used, 111 processors were available for the experiments. The switch for this system has 4 columns (stages) of 4-input 4-output switching elements, and is configured to contain 2 paths between each pair of Processor Nodes.

4. Experiment #1: Matrix Multiplication

The matrix multiplication program was timed in a number of environments:

1. Without any hot spots.
2. With a hot spot generated by read and write references, using only cool memories for the matrices. That is, both the hot memory and the memories of processors used to generate the hot spot were avoided.
3. With a hot spot generated by read and write references, using both the hot memory and the cool memories for the matrices.
4. With a hot spot generated by block transfers, using only cool memories for the matrices. As in (2) above, both the hot memory and the memories of processors used to generate the hot spot were avoided.
5. With a hot spot generated by block transfers, using both the hot memory and the cool memories for the matrices.

Data

For runs involving a hot spot, 100 processors were used to generate the hot spot. This left 11 processors with cool memories.

All runs used square matrices of size 192x192. This size was chosen because:

1. The run time for the matrix multiplication is long enough to give statistically interesting results, and short enough to run a series of experiments.
2. The matrix multiplication benchmark is written in a way that makes analysis of the results simpler when the matrix dimensions are multiples of 6 (see below).

The data obtained by timing the matrix multiplication benchmark on successively larger processor configurations for each set of experimental conditions is shown in Table 1.

Discussion

When the matrix multiplication program avoids the hot memory, the presence of the hot spot has negligible impact on the program's performance: there is less than 1% increase in execution time. When the program uses the hot memory, the impact

Matrix Size = 192x192

	Time (seconds) Number processors. 1	2	4	8	11
No hot memory	65.73	32.73	16.37	8.22	—
Hot memory – 100 processors doing simple read/write references					
Avoid hot memory (11 cool memories)	66.02	32.97	16.67	8.47	6.27
Use hot memory (11 cool memories + 1 hot memory)	67.55	33.72	17.10	8.67	6.39
Hot memory – 100 processors doing 768 byte block transfers					
Avoid hot memory (11 cool memories)	66.07	33.13	16.62	8.50	6.25
Use hot memory (11 cool memories + 1 hot memory)	92.01	46.51	23.42	12.05	8.90

Table 1: Data from matrix multiplication benchmark program.

depends upon the way the hot spot is generated. There is a small increase in run time when the hot spot is generated by read and write references (2.76% in the single processor case) and a substantial increase when the hot spot is generated by block transfers (40% in the single processor case). Since block transfer operations keep the memory busy longer than single read and write references, this result is not surprising.

Switches for larger Butterfly machines are typically configured with alternate paths to make the machine resilient to failures in switching elements (which almost never occur) and to reduce contention within the switch. For example, as mentioned in Section 3, the switch for the 128 processor machine used in these experiments has one alternate path (for a total of two paths) between each pair of nodes. The data presented above was collected with the alternate switch paths enabled. Measurements were also made to determine the sensitivity of the timing data to alternate paths by repeating the experiment with the alternate paths disabled.

Use of alternate paths within the switch makes a small difference. When the hot spot is generated by read and write references and the hot memory is used, the program runs about 1% slower when the alternate paths are disabled. When the hot spot is generated by block transfers and the hot memory is used, the program runs about 2 1/2% slower when the alternate paths are disabled.

The following is an analysis of the program's behavior when running on a single processor in the presence of a hot spot generated by block transfers. It shows that the increase in execution time is due almost entirely to the increase in time required to access data in the hot memory.

The matrix multiplication program uses the block transfer operation to make local copies of matrix rows and columns before accessing the individual elements to multiply and add.

To multiply matrices of size 192x192, 36864 dot products must be computed. The program is written to compute dot products in groups of 36. This involves 12 block transfer operations to obtain 6 rows and 6 columns. Thus, 12 block transfers yield 36 results, each result requiring 1/3 block transfer. Therefore, the program performs 12288 block transfer operations.

Twelve memories were used to hold the matrices, one of which was hot. Therefore, 1/12 of the block transfers can be expected to be delayed due to the hot spot. The block transfer delay from a hot memory was measured separately by timing a 768 byte block transfer from a cool memory, and then timing it again when the memory was made hot by 100 processors doing block transfers from it:

	Time to block transfer 768 bytes (microseconds)
No hot memory	322.18
Hot memory 100 processors doing 768 byte block transfers.	25885.81

Therefore, the additional time for the matrix multiplication program to perform block transfers from the hot memory should be about:

(1/12) * 12888 * (25885.81-322.18) = 26.18 seconds

The measured increase in the execution time for the matrix multiplication program for a single processor was

92.01 - 65.73 = 26.28 seconds

Thus, the performance degradation resulting from the hot memory is due almost entirely to contention at that memory. The effect of switch contention on program performance is negligible, even with severely non-uniform memory reference patterns.

Note that communication (accessing remote memory) accounts for about 6%[1] of the execution time of the matrix multiplication program. Our experience with the Butterfly Parallel Processor is that communication typically accounts for 4%-10% of the execution time for an application. Because a relatively small part of total

---

[1] = 100% * (12288 blk xfers * 322.18 microsec/blk xfer) / (66.02 sec).

program execution time is due to communication, remote memory reference times must be severely degraded before memory hot spots can have a signficant effect on overall program performance. The purpose of the second experiment was to measure the effect memory hot spots have on remote memory references as opposed to overall program performance.

5. Experiment #2: Remote References

The second experiment timed references made from a given processor node to memory on every other processor node. Four types of references were timed:

1. Single word (4 byte) read references;

   ```
   t = * p;
   ```

   where *t* is a variable in local memory and *p* is a pointer (*int* *) to the word to be read.

2. Single word (4 byte) write references;

   ```
   * p = t;
   ```

   where *t* is a variable in local memory and *p* is a pointer (*int* *) to the word to be written.

3. Block transfer of data from the remote memory;

   ```
   Do_bt (remotep, localp, numbytes)
   ```

   where *remotep* is a pointer to a block of data on a remote node to be copied, *localp* is a pointer to an area in local memory, and *numbytes* is the number of bytes to be copied to local memory.

4. Block transfer of data to the remote memory;

   ```
   Do_bt (localp, remotep, numbytes)
   ```

   where *localp* is a pointer to a block of data in local memory to be copied, *remotep* is a pointer to an area on a remote node, and *numbytes* is the number of bytes to be copied from local to remote memory.

The measurements for a given reference type were made by timing a tight loop that included the memory reference:

```
Start_timer;
for (i = 0; i < loopcount: i++)
    Make_reference;
Stop_timer;
```

In addition, the empty loop was timed to measure loop overhead:

```
Start_timer;
for (i = 0; i < loopcount; i++) ;
Stop_timer;
```

Data

Runs that involved hot spots used 100 processors to generate the hot spot. Therefore, in those runs there was 1 (remote) hot memory, 10 (remote) cool memories, 1 (local) cool memory, and 99 (remote) memories for processors generating the hot spot.

The timing data in Table 2 shows average times for one iteration of the memory referencing loop for the various memory reference types under the conditions indicated. For the first set of data, which was collected without any hot spots, the "remote" reference times were computed by averaging the loop times measured for each of the 110 remote memories and dividing by *loopcount*. Data from the hot memory measurements was treated similarly. For example, the "hot memory" reference times were computed by dividing the measured times through the reference loop by *loopcount*; and the "cool memory" reference times were computed by averaging the loop times for the 10 cool memories and dividing by *loopcount*. *Loopcount* for this data was 10000. The loop overheads for each of the conditions were measured as described above, and factored out of the data. That is, the times presented exclude the measured loop overheads.

	Reference times (microseconds)					
	read	write	bt-from 256 bytes	bt-from 768 bytes	bt-to 256 bytes	bt-to 768 bytes
No hot memory						
remote	15.41	7.87	111.38	317.17	112.00	339.26
Hot memory – 100 processors doing simple read/write references						
cool memory	16.70	8.75	112.35	316.20	113.94	340.19
hot memory	701.93	306.80	473.99	1393.59	276.61	470.09
Hot memory – 100 processors doing 768 byte block transfers						
cool memory	15.95	9.02	112.88	315.97	113.26	335.85
hot memory	17410.04	153.30	8178.84	25820.95	254.55	827.14

Table 2: Data from remote reference experiment.

Discussion

When there is a hot memory, references to cool memory are slowed down slightly. This is probably due to contention within the switch; switch messages used to reference cool memory collide with the switch messages used to make the memory hot.

When there is a hot memory, simple references to cool memory are slowed down about the same amount as block transfer references to cool memory. For example, remote reads from a cool memory when the hot spot is generated by read and write references are slowed by 1.29 microseconds (16.70 versus 15.41), and 256 byte block transfers from a cool remote memory are slowed by .97 microseconds (112.35 versus 111.38)[2]. This is not surprising since the slow down is due to the increased time for initiating successful message transmission through the switch, and the increase is independent of message size.

References to the hot memory are substantially slower. For most types of references a memory made hot by block transfers is slower than one made hot by read and write references. The major exception is that simple writes are slower when the memory is made hot by read and write references than when it is made hot by block transfers (306.80 versus 153.30). This is due to the buffering strategy in the Processor Node switch interface which, in effect, gives preference to simple writes: when the memory is hot due to read and write references, the write being timed must compete with the writes making the memory hot; whereas when the memory is hot due to block transfers, there are no other writes to compete with.

6. Conclusions

The principal conclusion to be drawn from these experiments is that the results reported by Pfister and Norton do not generalize to the Butterfly Parallel Processor. While memory contention has an important effect on program performance in a Butterfly system, switch contention does not.

The matrix multiplication experiment showed that non-uniformities in memory reference patterns have very little effect on the behavior of a program that avoids the hot memory. When the hot memory is avoided, its presence has virtually no effect on a program's performance, even if the non-uniformities are large.

---

[2] 768 byte block transfers were actually measured to be slightly faster (316.20 versus 317.17).

If a program uses a hot memory, the performance degradation due to the hot memory depends on the extent to which the hot memory is used by the program. That is, the program is appreciably slowed only when it references the hot memory. Although the memory reference experiment showed slight slow down in references to the cool memories, the matrix multiplication experiment showed that the slight slow down has negligible impact on overall program performance.

There is no evidence that the tree saturation phenomenon described by Pfister and Norton occurs in the Butterfly Switch. Severe non-uniformities can lead to a small increase in contention within the switch, but the saturation effect simply does not occur.

Acknowledgements

I would like to thank Will Crowther, John Goodhue, and Walter Milliken, who read several drafts of this paper and provided helpful suggestions for presenting the results of the experiments. The Defense Advanced Research Projects Agency provided support for the development of the Butterfly Parallel Processor.

References

[1] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, T. Blackadar.
Performance Measurements on a 128-Node Butterfly Parallel Processor.
In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 531-540. IEEE Computer Society Press, August, 1985.

[2] *Butterfly Parallel Processor Overview*
BBN Laboratories Incorporated, 1985.

[3] G.F. Pfister and A. Norton.
"Hot Spot" Contention and Combining in Multistate Interconnection Networks.
In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 790-797. IEEE Computer Society Press, August, 1985.