Exploring the Design Space for a Shared-Cache Multiprocessor

Basem A. Nayfeh and Kunle Olukotun

Computer Systems Laboratory
Stanford University, CA 94305

Abstract

In the near future, semiconductor technology will allow the integration of multiple processors on a chip or multichip-module (MCM). In this paper we investigate the architecture and partitioning of resources between processors and cache memory for single chip and MCM-based multiprocessors. We study the performance of a cluster-based multiprocessor architecture in which processors within a cluster are tightly coupled via a shared cluster cache for various processor-cache configurations. Our results show that for parallel applications, clustering via shared caches provides an effective mechanism for increasing the total number of processors in a system, without increasing the number of invalidations. Combining these results with cost estimates for shared cluster cache implementations leads to two conclusions: 1) For a four cluster multiprocessor with single chip clusters, two processors per cluster with a smaller cache provides higher performance and better cost/performance than a single processor with a larger cache and 2) this four cluster configuration can be scaled linearly in performance by adding processors to each cluster using MCM packaging techniques.

1 Introduction

Improvements in semiconductor fabrication and packaging technology will soon make it possible to implement computer architectures that exploit high-bandwidth, low-latency communication among multiple high performance functional units, and significant amounts of cache. One possible way of implementing such an architecture would be to integrate the functional units and memory on a single die [7]. This sort of implementation is appealing because it eliminates the cost and performance impact of chip-crossings inherent in a multiple chip architecture in which the chips are individually packaged and interconnected on a printed circuit board (PCB). An alternative implementation technique is to partition the various functional units and memory over several chips and then interconnect these chips on a single substrate using multichip-module (MCM) packaging technology. This alternative has the potential to produce more powerful computers than current single-chip implementations. MCM technology can provide higher performance because more functional units and larger memories can be placed in close proximity to one another. These functional units and memories would be interconnected using low latency MCM interconnections which eliminate most of the extra delay associated with chip-to-chip connections on a PCB. In fact, the electrical properties of the interconnect in most MCM technologies are even better than the properties of the on-chip interconnections. Thus, it may even be advantageous to use the MCM interconnect for long intra-chip connections [3].

One of the ways to organize the functional units and memory offered by a large chip or an MCM is as a multiprocessor. Such a machine might consist of multiple processors that share a single multi-ported cache in a organization similar to the Alliant FX/8 [12]. In the design of such a multiprocessor one has the ability to change the ratio of the number of processors to the amount of cache. Thus, a key design question is what should the ratio of processors to cache memory size be to achieve the best cost/performance?

The objective of this paper is to address the above question by looking at the performance trade-offs between cache size and number of processors for a single-chip implementation and its use in building larger MCM based multiprocessors. The performance of a fixed number of processors and varying cache size was studied by Agarwal [1]. In addition, Rothberg et al. [17], examined the relationship between multiprocessor cache sizes and the number of processors from an applications program perspective for large scale multiprocessors. Our evaluation, however, combines the interaction of the application programs, cache sizes, and small to medium numbers of processors. This allows us to present detailed performance results for an important portion of the processor-cache design space.

The multiprocessor architectures that we consider in this study are small to medium sized cluster-based multiprocessors with four to thirty-two processors. We expect that this sort of architecture will become common place in workstations and compute servers in the future. The benchmarks used in this study fall into two classes: parallel applications running in single user mode and sequential applications that are run in multiprogramming

mode. These benchmarks represent two of the important ways that multiprocessors are used, i.e., as accelerators to reduce the execution time of single applications and as compute-servers for multiple users in a throughput oriented environment.

The rest of this paper presents the study of processor-cache size trade-offs. In the next section we describe the base architecture, simulation methodology, and the benchmark applications that are used in this study. The paper itself is divided into three parts. In the first part, Section 3, we consider the effect on performance of changing the cache size and the number of processors independently. The performance data provides us with a set of curves that characterize the processor-cache design space for the two benchmark classes. In the second part, Section 4, we analyze the area cost of a processor and of the multiported non-blocking cache. In the third part, Section 5, we use these costs to evaluate the cost/performance of a single chip implementation and its ability to scale to larger MCM based multiprocessors. Our conclusions are presented in Section 6.

2 Methodology

2.1 Architecture

As microprocessors have improved in performance, using them to build a multiprocessor system that can achieve high-bandwidth, low-latency communication over a single shared bus has become very difficult because bus performance has not scaled at the same rate as processor performance. This is due to an inherent limitation of the bus topology [4].

Currently, typical multiprocessors have large per-processor caches which are kept coherent using a shared bus and snooping protocols [2]. In such an architecture, large caches increase performance by reducing the communication traffic on the shared bus. Large caches are also required to hold the working sets of multiple applications that are executing concurrently in a multiprogramming environment.

The approach we advocate here to alleviate this problem is to cluster the processors so that there is a high-bandwidth, low-latency connection between processors within a cluster and a lower bandwidth inter-cluster bus. Clustering provides the benefit that data that is actively shared among processors within the cluster can be accessed very efficiently without using the inter-cluster bus.

There are two alternatives for organization of the cluster. The first alternative is to have separate per processor caches which are kept coherent over a high bandwidth intra-cluster bus [13]. This organization has the advantage that the total cache bandwidth scales with the number of processors in the cluster. However, coherence misses and invalidation traffic on the single shared bus can become a performance bottleneck. The second alternative is to make the processors communicate using a multi-ported shared cluster cache. While this alternative may take too many wires and pins if implemented on a PCB, it is well suited to a single chip, or MCM implementation where wires and I/O pins are abundant. This organization has the added benefit that if the two processors are sharing a particular cache line there will only be a single copy of the line in the shared cache as opposed to having two copies in each processor's cache. This makes the shared cache more efficient and avoids the cache coherence overhead. The disadvantage of a shared cache is that it could become a performance bottleneck. Furthermore, if the processors in the cluster are executing independent processes, the conflicts in a shared cache can cause higher miss rates than in a separate cache organization.

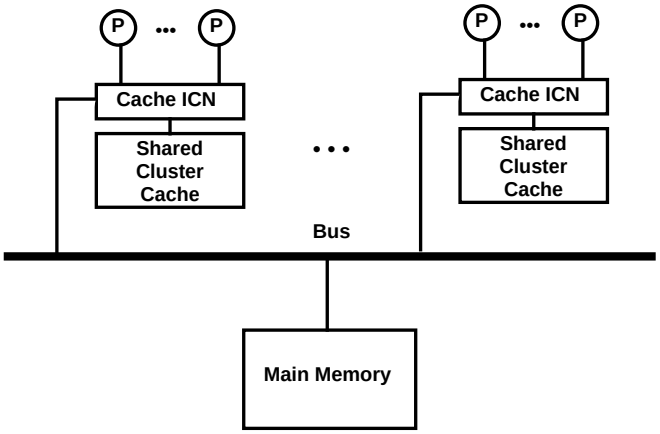

Figure 1: The base architecture

The base architecture that will be used in this study is shown in Figure 1. Each cluster consists of one to eight processors, a Shared Cluster Cache (SCC) for data and a separate instruction cache for each processor. To provide enough bandwidth to prevent the SCC from becoming a performance bottleneck, each processor has a dedicated port to a non-blocking cache with multiple banks [23]. The banks in the SCC are interleaved on cache lines; in other words, consecutive cache lines are contained in consecutive cache banks. The path between the processors and the SCC's banks is provided by the cache interconnection network. Each SCC has access to physical main memory over the shared bus. The SCCs are kept coherent with one another by using a snoopy invalidation based cache coherence protocol.

2.2 Parallel Simulation

2.2.1 Benchmark Applications

Our multiprocessing benchmark applications consist of three of the Stanford SPLASH benchmarks [20] namely: Barnes-Hut, MP3D and Cholesky. All three applications are written in C and use the Argonne National Laboratory macros [5] which facilitate process creation, data sharing, and synchroni-

zation. We refer the reader to [20] for a detailed description of the benchmarks.

Application	Description
Barnes-Hut	Hierarchical N-body simulation of the evolution of galaxies
MP3D	Particle-based simulation of rarefied hypersonic flow
Cholesky	Cholesky factorization of a sparse matrix

Table 1: Multiprocessor Benchmark Applications

For each processor-cache configuration in our design space, as mentioned in Section 1, we ran: Barnes-Hut using 1024 bodies, MP3D for 5 time steps using 10,000 particles and Cholesky using the BCSSTK14 input file.

2.2.2 Simulation Model

In order to simulate our multiprocessor architecture, we use Tango-Lite [8] to supply properly interleaved reference events to a detailed multiprocessor cache simulator. Our multiprocessor cache simulator implements a four SCC based system as described in Section 2.1. We address the issue of contention at the shared cache by considering contention on each individual bank within the SCC. In our model, each SCC has four banks for each processor in its cluster.

The SCCs are kept coherent with each other using an invalidation-based scheme on a snoopy bus. In this scheme a write to a line in a particular SCC causes that line to be invalidated, if present, in each of the other SCCs. We chose a cache line size of 16 bytes to help reduce false-sharing between clusters. We assume that the latency to fetch a cache line from main memory or from another SCC over the snoopy bus is fixed at 100 cycles. This latency is consistent with the ratio between processor clock speeds and bus transaction latencies in the most recent bus-based multiprocessor designs [6].

2.3 Multiprogramming Simulation

2.3.1 Benchmark Applications

Our multiprogramming benchmark workload consists of six SPEC92 [24] integer benchmarks and two SPEC92 floating point benchmarks as shown in Table 2

Application	Type	Description
sc	Integer	Spreadsheet calculations
espresso	Integer	Generates and optimizes PLA structures
eqntott	Integer	Translates boolean equations into truth tables
xlisp	Integer	LISP interpreter
compress	Integer	Lempel-Ziv data compression
gcc	Integer	C compiler
spice	Flt. Point	Analog circuit simulator
wave5	Flt. Point	Maxwell's equations solver

Table 2: Multiprogramming Benchmark Applications

2.3.2 Simulation Model

In order to simulate a multiprogramming workload, we generated annotated versions of the individual benchmarks using pixie [22] which allowed us to collect dynamic memory references and basic block information. The annotated programs were then run at the same time as separate processes with the pixie references being piped into a multiprogram scheduler. The scheduler uses a round-robin scheme with a time quantum of 5 million processor cycles to schedule processes on each processor in the cluster. A total of 100 million pixie references are simulated, which resulted in an average of 30 million instructions per application. These runs were repeated for each processor-cache configuration in our design space.

3 The Processor- Cache Size Performance Trade-off

3.1 Parallel Applications

In this section we study the processor-cache size design space by looking at the three parallel applications: Barnes-Hut, MP3D, and Cholesky. The design space consists of processor configurations ranging from one to eight processors per cluster, with SCC sizes ranging from 4 KB to 512 KB. Since we are focussing on small to medium size multiprocessors, we simulate four clusters resulting in a system configurations of 4 to 32 processors. The simulation results for each application are presented in terms of normalized execution time as a function of SCC size. The simulation results for each application are summarized in graph followed by a discussion of its performance characteristics.

3.1.1 Barnes-Hut

We expected Barnes-Hut to achieve near-linear speedup on our cluster architecture since the invalidation characteristics of four clusters effectively behave as four processors on a snoopy bus. For Barnes-Hut, [14, 20] have shown that four processors achieve linear speedup.

From Figure 2, it is apparent that greater than linear speedup is achieved on configurations with medium to large

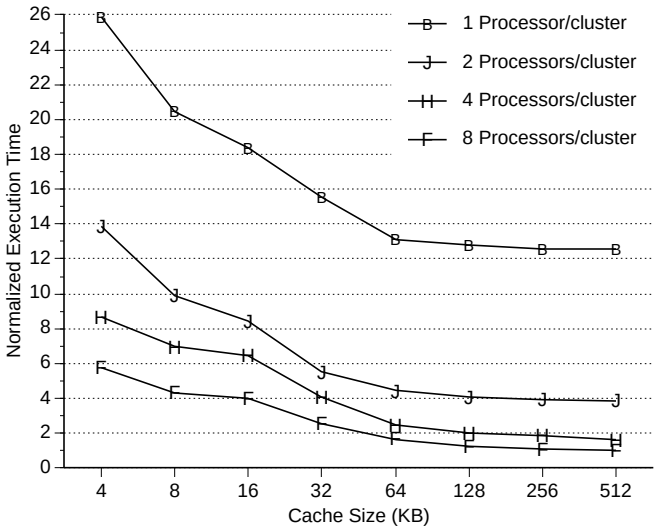

Figure 2: Barnes-Hut performance characteristics.

SCC sizes. These speedups are given in Table 3. The better

SCC Size	1 Proc./ cluster	2 Procs./ cluster	4 Procs./ cluster	8 Procs./ cluster
4 KB	1.0	1.9	3.0	4.5
8 KB	1.0	2.1	2.9	4.8
16 KB	1.0	2.2	2.8	4.6
32 KB	1.0	2.8	3.8	6.1
64 KB	1.0	3.0	5.3	7.9
128 KB	1.0	3.1	6.5	10.3
256 KB	1.0	3.2	6.8	11.8
512 KB	1.0	3.2	7.7	12.5

Table 3: Barnes-Hut speedups relative to one processor per cluster

than expected performance of Barnes-Hut is due to the increased locality of reference within a cluster and can be explained in terms of Barnes-Hut's data structures as follows:

Barnes-Hut is an N-body application that uses a hierarchical octree representation of three-dimensional space, allowing the typical n^2 force computations among n bodies to be reduced to $n \log n$ computations. This is achieved by constructing a tree [19] in which each node approximates the contribution of its children to the force computation. In order to compute the force on a particular body, the tree is traversed starting from the root. The depth of traversal along a particular branch in the tree is dependent upon a closeness criterion which allows the approximation to be used for sufficiently far away bodies. Each processor is responsible for computing the force on, and updating the positions and velocities of a subset of the bodies, which are partitioned into groups based on the structure of the tree [20]. If the partition is such that processors within a cluster are assigned bodies which are adjacent in the tree, we expect to see much greater data reuse and significant reduction in miss rates due to prefetching [21]. The prefetching is due to processors within a cluster traversing the same regions of the tree at around the same times. Thus, one processor effectively brings in data to the cache which will be used by the remaining processors in the cluster before it is replaced or invalidated. However, it is possible that some of the references from multiple processors in the cluster will interfere with each other destructively and begin to increase the overall miss rate. In addition, we noted that as the number of processors per cluster increases, the total number of invalidations actually performed in the system decreases. The net effect of prefetching, invalidations and destructive interference on an SCC's read miss rate is given in Table 4 for three sample SCC sizes. Al-

Processors /cluster	Read Miss Rate		
	8 KB	64 KB	256 KB
1	7.96%	4.55%	4.10%
2	7.82%	1.45%	0.92%
4	8.53%	0.86%	0.17%
8	10.33%	1.26%	0.26%

Table 4: Effects of prefetching and destructive interference on read miss rates for Barnes-Hut.

though prefetching and decreasing invalidations provide dramatic decreases in the read miss rate with multiple processors per cluster for medium to large SCCs, destructive interference is the dominating factor for small SCCs. In the medium to large SCCs, the slight increase in miss rate between four and eight processors per cluster demonstrates that there is still some destructive interference. However, the overall reduction in the medium to large SCC miss rate with multiple processors provides the greater than linear performance as shown in Table 3.

3.1.2 MP3D

MP3D typically does not scale very well in invalidation based architectures with large numbers of processors due to the lack of locality and high numbers of invalidation misses from frequent accesses to globally shared data [13]. The advantage of the cluster based architecture is that invalidation traffic of an n-cluster multiprocessor approaches that of a n-processor snoopy cache coherent multiprocessor. In our simulations we found that adding more processors to each cluster had almost no effect on the invalidation traffic between clusters. This is in marked contrast to adding more processors directly to the

shared bus which typically increases the invalidation traffic, and in the case of MP3D becomes the main limiting factor in obtaining speedup.

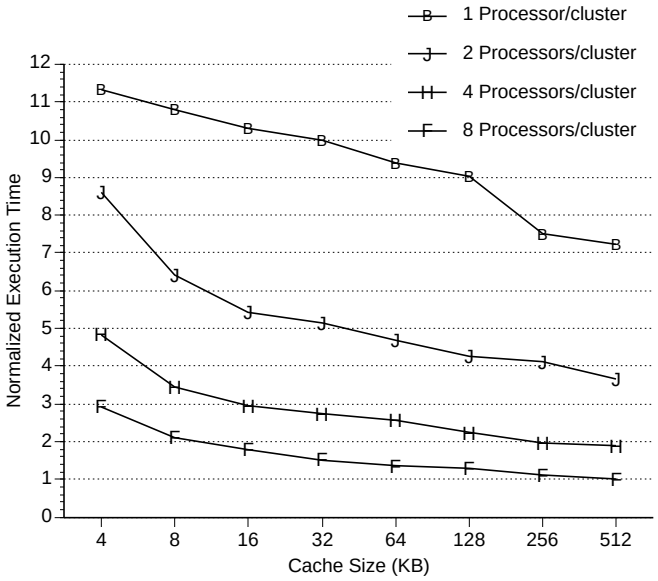

Figure 3: MP3D performance characteristics.

Figure 3 shows the performance characteristics of MP3D. For the smallest SCC size of 4 KB, the self-relative speedup of the eight processors per cluster to one processor per cluster is approximately 3.8, whereas the self-relative speedup for the largest SCC size of 512 KB is 7.2. The reason for this is that unlike the Barnes-Hut application, prefetching does not reduce the miss rates of MP3D due to the lack of locality; however, destructive interference does increase the miss rates of smaller SCCs. Thus, medium to large SCCs achieve almost linear speedup on the MP3D application.

3.1.3 Cholesky

The results for Cholesky using the BCSSTK14 input file are given in Figure 4. As in the case of MP3D, we found that there was almost no increase in invalidations as the number of processors increased. We found that the effects of prefetching for the various processor-cache configurations was existent but not as pronounced as in Barnes-Hut. For example, the read miss rate for an SCC size of 32 KB decreases by 25% as the number of processors per cluster is increased from one to eight.

With these characteristics, we would have expected good speedup, but in fact this was not the case. For the smallest SCC size of 4 KB, the self-relative speedup of the eight processors per cluster to one processor per cluster is approximately 3.0, whereas the self-relative speedup for the largest SCC size of 512 KB is only 3.5. The most probable reason for the lack of speedup is the BCSSTK14 input file itself which is small enough to allow reasonable simulation times, yet suffers from

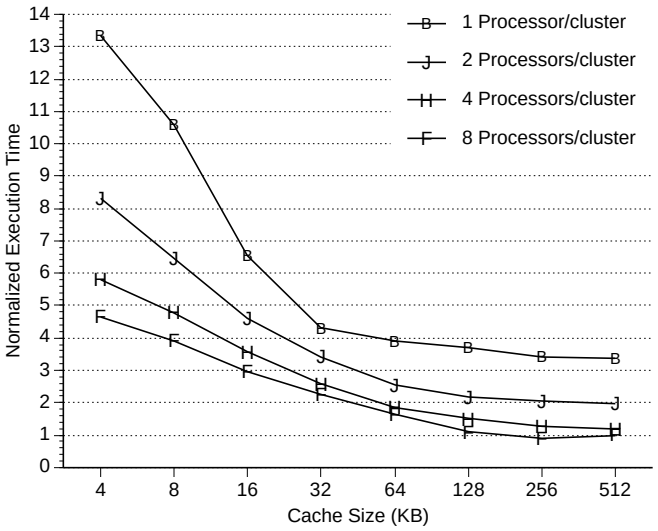

Figure 4: Cholesky performance characteristics.

limited concurrency, bad load balancing and high synchronization overhead [20].

3.2 Multiprogramming Application

A major limitation of multiprogrammed performance on both uniprocessors and multiprocessors is the increased cache miss rates that an application experiences due to the effects of context switches. In the clustered architecture, we expect that the miss rates should increase further due to interference caused by multiple processes making references simultaneously to the same shared cache. Figure 5 shows the

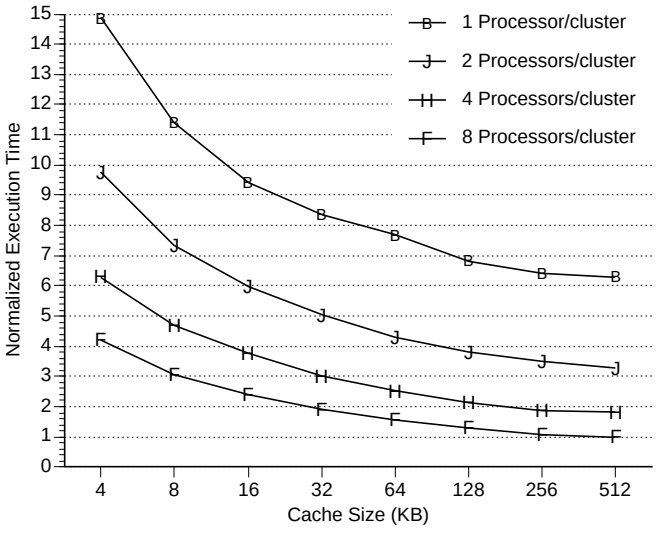

Figure 5: Multiprogramming performance characteristics.

performance of a single cluster with the multiprogram workload. Figure 6 shows the self-relative speedups for executing the total multiprogram workload as a function of the number of processors per cluster. Since these numbers are normalized to the one processor per cluster case for each SCC size, the deg-

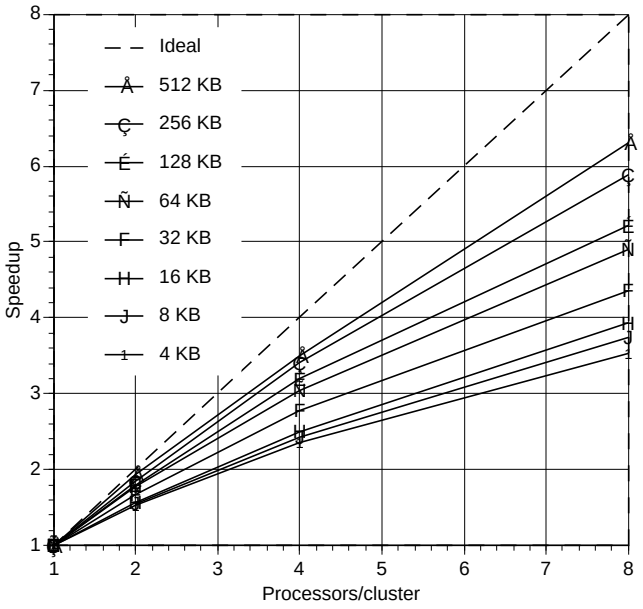

Figure 6: Multiprogramming speedup characteristics.

radation from the ideal performance is due to interference conflicts alone. As expected, increasing the SCC size reduces the degradation and thus, the fraction of the total execution time that the processor is idle waiting for memory requests. For example, the normalized execution time for eight processors per cluster increases by a factor of 4.1 in going from the smallest SCC size of 4 KB to the largest of 512 KB. The trend is similar for the other processor configurations.

4 Cluster Implementation and Costs

In this section we present possible chip implementations which can be used to realize our architecture model as described in Section 2.1. In addition, we evaluate the cost of these implementations. First, we present the implementation assumptions used, followed by a discussion of four chip designs.

4.1 Implementation Assumptions

There are three components of processor performance and cost that are directly affected by the organization of the first level cache and the number of processors on a chip. These are the load latency in processor cycles, the processor cycle time and the total chip area. To evaluate the effect of different cluster sizes on these components we must make assumptions about the semiconductor process, in addition to the processor's area, pipeline, and cycle time.

The semiconductor process technology that we assume is a CMOS technology with 0.4µm gate lengths and three layers of interconnect. This technology should be available by the end of 1996. Using this technology, it will be economical to fabricate chips that are 18 mm on a side with a total die area of 300 mm^2.

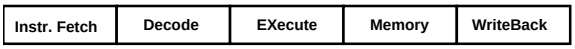

Figure 7: Five-stage integer pipeline.

The processor that we use to estimate chip area is the DEC Alpha 21064 [15]. This is a high-performance 64-bit processor with integrated integer unit (IU), floating point unit (FPU), instruction cache and data cache. This processor is implemented in a 0.68µm gate-length semiconductor process. We have linearly scaled the dimensions of this processor to fit those of a 0.4µm process. In reality, scaling a processor dimensions is more complex than this, however, this is a good first approximation. We note that only the area of the IU, FPU and instruction cache are scaled in this manner. The costs of a one processor per cluster data cache and various SCCs are worked out in detail.

In the architectural studies we assumed the conventional five-stage integer pipeline shown in Figure 7. In this pipeline the data access takes place during the memory stage and, consequently, load instructions have a total latency of two cycles.

Developing an estimate of processor cycle time is difficult because it is very dependent on semiconductor process technology and circuit design. An often used metric when comparing alternatives in the same process technology family, is to measure cycle times in terms of gate delays rather than nanoseconds. In our evaluation, we use the delay of an inverter with a fanout of four (FO4) [11] as our basic unit of gate delay. Using this metric, the Alpha 21064 chip, which represents very aggressive circuit design, has a processor cycle time of 30 FO4 inverter delays [10]. It is certain that future microprocessor implementations will continue to use aggressive circuit design to achieve short processor cycle times; thus, we assume a processor cycle time of 30 FO4 inverter delays in our evaluation.

4.2 One Processor per Cluster

Figure 8 shows the floorplan of a chip that is implemented in 0.4µm semiconductor process technology. The chip has a total area of 204 mm^2 and contains a 64-bit integer unit, a 64-bit floating point unit, a 16 KB instruction cache and a single ported 64 KB data cache composed of eight 8 KB blocks of SRAM. The SRAM block area of 6.6 mm^2 is based on a detailed SRAM cell layout and includes the overhead for the cache tags and the drivers to drive the data bus back to the functional units. Because there are two rows of SRAM, the 2.2 mm width of the SRAM blocks also includes the wiring channels required to connect the bottom row of SRAM blocks to the functional units.

The 64 KB data cache size is dictated by processor cycle time considerations. This is the largest direct-mapped cache that can be accessed in 30 FO4 inverter delays. This access time includes the time for the functional units to drive the cache address lines and the time for the SRAM to drive the data bus back to the functional units.

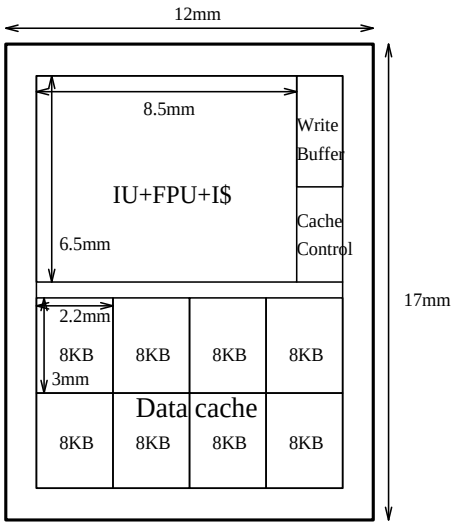

Figure 8: Floorplan of a one processor per cluster chip.

4.3 Two Processors per Cluster

Figure 9 shows the floorplan of a two processor per cluster chip. This chip contains two integer units, two floating point units and two 16 KB instruction caches, and one 32 KB SCC. This two processor chip has a total area of 279 mm^2 which is 37% larger than the single processor chip.

The SCC is triple ported and eight-way interleaved. There are two processor ports and one cache controller port for refilling the SCC. The ports are implemented by a crossbar processor-cache interconnection network (ICN) that takes up an area of 12.1 mm^2, assuming a wire pitch of 1.6 μm. The eight-way interleaved SCC is used to reduce the bank conflicts between the processors. The SRAM blocks used in each of the SCC's banks are based on the same SRAM cell as the single processor data cache. However, each block also contains an arbitration unit, a write buffer and the extra drivers necessary to drive the long wires in the crosssbar ICN. The result is that although the SRAM block has an area of 8 mm^2, it only contains 4 KB of cache. The SRAM block also contains an extra decoder which allows the block to be accessed from the top or the bottom.

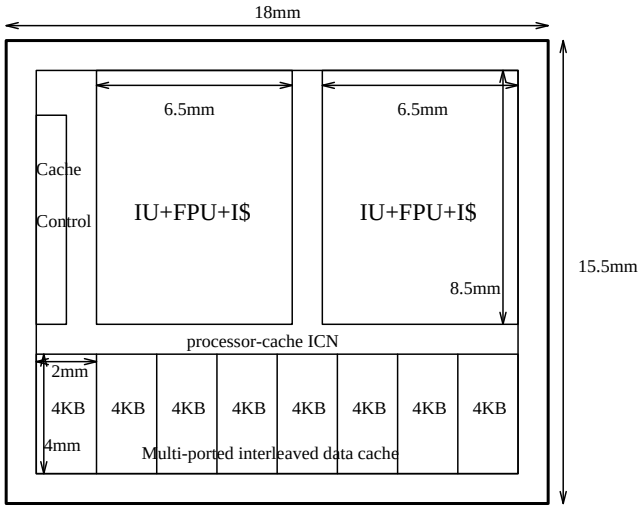

Figure 9: Floorplan of a two processors per cluster chip.

SCC bank arbitration is required to decide which processor will access a bank each processor cycle. Due to the long ICN wires that must be driven, arbitration takes 17 FO4 inverter delays to complete. To avoid increasing the CPU cycle time above 30 FO4 inverter delays, an extra arbitration stage must be added between the EXecute and the MEMory stage of the five-stage pipeline. This increases the load latency to three cycles and further complicates the pipeline bypassing that needs to be implemented to minimize pipeline stalls [9].

4.4 Four Processors per Cluster

A four processor per cluster implementation will not fit on a single chip even in 1996 technology. Therefore, it is necessary to combine two of the two processor chips to implement a four processor cluster. Such a multi-chip implementation will require MCM packaging technology to reduce the performance impact of chip crossings.

In the four processor per cluster design we have assumed current MCM packaging technology from MicroModule Systems [16]. This technology provides two signal routing layers with a 75 μm pitch. Even though the wires in this technology have negligible resistance and a capacitance per unit length that is half of that of the on-chip wires, accessing the cache on another chip in a single cycle would exceed the 30 FO4 inverter delay limit. To avoid increasing the processor cycle time, another data cache access stage after the MEMory stage is required. This increases the load latency to four cycles.

Figure 10 shows the floorplan of the four processors per cluster building block. The major differences between this chip and the two processor per cluster chip are a larger processor-cache ICN, and a larger number of I/O pins. A larger processor-cache ICN, 12 mm^2 versus 10 mm^2, is required to provide two extra ports to each cache bank. The extra I/O pins are required to provide communication with the two processors on the other chip. We estimate that each processor needs 160 address, data and control lines. Even though this results in a chip with a total of 600 signal I/O pads, these pads can still be placed in a pad frame around the perimeter of the chip. The total chip area, therefore, is estimated to be 297 mm^2 which is 46% larger than the single processor chip.

4.5 Eight Processor Cluster

The eight processor per cluster building bock, shown in Figure 11, extends the concept of an SCC based multiprocessor one step further. This design requires the connection of each of the eight banks to nine ports. Two processor-cache ICNs provide these ports. The major challenge in building this chip is

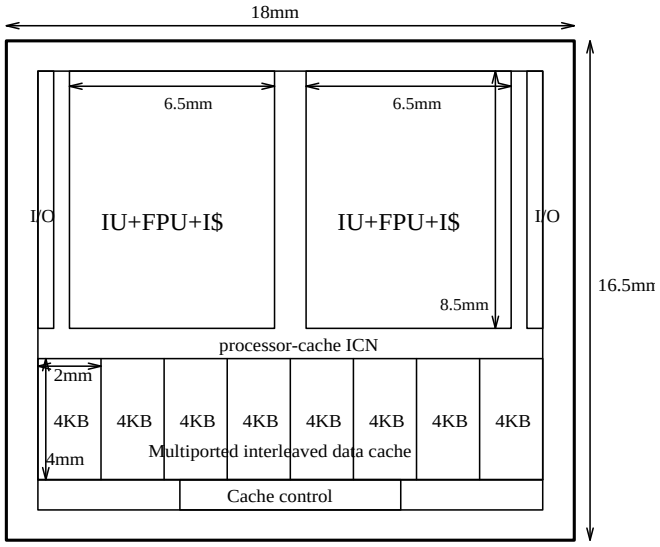

Figure 10: Floorplan of a four processors per cluster building block.

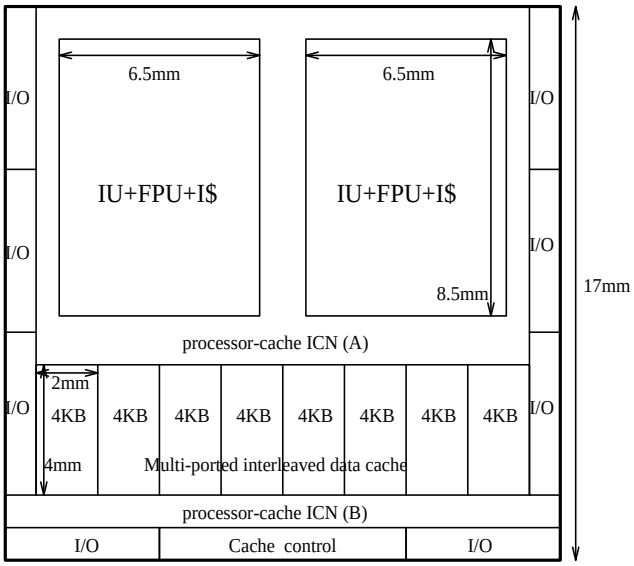

Figure 11: Floorplan of a eight processors per cluster building block.

providing enough I/Os to communicate with six other processors. To provide the 1100 signal I/O pads required on this chip, technology such as IBMs controlled-collapse chip connection (C4) in which pads are placed in an array on a separate layer of metal on top of active circuity [18], must be utilized. Assuming this type of technology, the total area of this chip is 306 mm^2 which is 50% larger than the one processor chip.

5 Comparing Shared-Cache Implementations

Using the processor-cache performance trade-off results presented in Section 3, we can compare the performance of the implementation alternatives presented in Section 4. There are two interesting performance comparisons. These are the single chip cluster implementation comparison and the multiple chip MCM implementation comparison.

5.1 Single Chip Implementations

In this section we compare the implementation of a cluster on a single chip. The single chip implementations are one processor with a 64 KB data cache and two processors with a 32 KB SCC. These implementations cannot be compared directly using the performance values from Section 3 because the one processor implementation has a two cycle load latency and two processor implementation has a three cycle load latency.

To account for different load latencies we used the basic-block profiling tool, pixstats [22], to compare the execution times of the benchmarks on a uniprocessor with a perfect memory system for load latencies of two, three and four cycles. The compiler that was used in this comparison scheduled instructions for a load latency of three cycles, thus the four cycle load latency results are pessimistic. The other functional unit latencies used in this comparison are those of the DEC Alpha 21064 [15]. The results of this comparison, normalized to a load latency of 2 cycles, are shown in Table 5. These results over estimate the effect of load latency on execution time because the cycles wasted in the memory system are not counted. Multiplying the performance values in Section 3 by the factors in this table provides a good approximation to the effect of load latency on program execution time.

Benchmark	Load Latency		
	2 cycles	3 cycles	4 cycles
Barnes-Hut	1.00	1.06	1.13
MP3D	1.00	1.07	1.14
Cholesky	1.00	1.07	1.16
Multiprogramming	1.00	1.08	1.17

Table 5: Relative uniprocessor execution times for various load latencies.

Table 6 shows the relative execution time, taking account of the different load delays, for the two single chip cluster configurations. It is clear that the four two-processor clusters each with a 32 KB SCC performs much better than four single processors each with a 64 KB data cache. On average, the two-processor per cluster configuration is 70% faster than the one processor configuration. We recall from Section 4.3,

that the two-processor chip requires 37% more area than the one-process chip. The conclusion is that implementing two processors on a chip rather than a larger cache not only improves absolute performance, but also improves cost/performance by 24% based on the die area of the processor.

Benchmark	Cluster Configuration	
	1 Processor/ 64 KB	2 Processors/ 32 KB
Barnes-Hut	13.1	5.8
MP3D	9.4	5.5
Cholesky	3.9	3.4
Multiprogramming	7.7	5.4

Table 6: Performance comparison of single chip implementations.

5.2 MCM Implementations

The four and eight processors per cluster configurations, described in Section 4.4 and Section 4.5 respectively, are implemented using slightly modified versions of the two processors per cluster configuration of Section 4.3. This leads to two MCM cluster implementations that we consider: 16 processors configured as four processors per cluster with a 64 KB SCC, and 32 processors configured as eight processors per cluster with a 128 KB SCC. Table 7 shows the performance of these implementations on the four benchmarks, including the effects of a four cycle load latency.

It is interesting to note, however, that the parallel application performance of a 16 processor system built with four processors per cluster is double the performance of an eight processor system built with two processors per cluster, despite the additional load latency. With the exception of Cholesky, the performance increase in going from the 16 to 32 processor system is linear. This indicates that scaling is possible in small to medium sized multiprocessors using two processors per chip as the basic building block.

6 Conclusions

We have studied the performance of a cluster-based multiprocessor architecture in which processors within a cluster are tightly coupled via a shared cluster cache (SCC) for various processor-cache configurations. Our results show that for parallel applications, clustering via shared caches provides an effective mechanism for increasing the total number of processors in a system without increasing the number of invalidations. In fact, on some applications, prefetching effects in the SCC can dramatically reduce the miss rates and provide in some cases greater than linear speedup. This prefetching occurs as a natural consequence of sharing caches and does not require the overhead of explicit prefetching instructions, programmer intervention or compiler support. Multiprogramming workloads, however, may suffer some degradation in performance due to increased interference conflicts in the shared cluster cache. For large caches, these effects are less pronounced.

Benchmark	Cluster Configuration	
	4 Processors/ 64 KB	8 Processors/ 128 KB
Barnes-Hut	2.8	1.4
MP3D	2.9	1.5
Cholesky	1.6	1.3
Multiprogramming	2.9	1.5

Table 7: Performance comparison of MCM based cluster configurations.

After exploring the processor-cache design space, we examined cost/performance issues for four possible cluster configurations. Our results show that in the near future, incorporating two processors on a single chip with a shared cluster cache yields both better performance and cost/performance than a single processor with a larger cache. In addition, modified versions of the two processor chip were found to be an effective building block for small to medium sized multiprocessors.

Acknowledgments

We would like to thank Kun-Yung Chang for his help in developing the implementation and cost models for the shared cluster caches and Bharadwaj Amurtur for his SRAM design parameters. We would also like to thank Jaswinder Pal Singh and Jeremy Levitt for their help with the parallel and multiprogramming applications, respectively. Finally, we would like to thank Mark Horowitz, Anoop Gupta and the reviewers for their insightful comments on earlier drafts of this paper.

This research was supported by a grant from the Powell Foundation.

References

[1] A. Agarwal, "Multiprocessor cache Analysis using ATUM," in *Proc. 15th Annual Int. Symp. Computer Architecture*, 186-195, 1988.

[2] J. Archibald and J. Baer, "An evaluation of cache coherence solutions in shared-bus multiprocessors," *ACM Transactions on Computer Systems*, vol. 4, no. (4), pp. 273–298, November 1986.

[3] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, 1990, Reading, Massachusetts: Addison-Wesley Publishing Company. 1990.

[4] L. A. Barroso and M. Dubois, "The performance of cache-coherent ring-based multiprocessors.," in *20th Annual Int. Symp. Computer Architecture*, San Diego, CA, ACM, 268–277, 1993.

[5] J. Boyle and e. al, *Portable Programs for Parallel Processors*, 1987, Holt, Rinehart, and Winston Inc. 1987.

[6] M. Galles, "The Challenge Interconnect: Design of a 1.2 GB/s coherent multiprocessor bus," in *Hot Interconnects*, pp. 1.1.1-1.1.7, 1993.

[7] P. P. Gelsinger, P. A. Gargini, G. H. Parker, and A. Y. C. Yu, "Microprocessors circa 2000," *IEEE Spectrum*, vol. no. pp. 43–47, October 1989.

[8] S. Goldschmidt, "Simulation of Multiprocessors: Accuracy and Performance," Ph.D. Thesis, Stanford University, 1993.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, 1990, San Mateo, California: Morgan Kaufman Publishers, Inc. 1990.

[10] M. Horowitz, "Clocking in high performance processors," in *VLSI Circuit Symposium*, Seattle, WA, 50, 1992.

[11] M. Horowitz, S. Przybylski, and M. D. Smith, "Tutorial on Recent Trends in Processor Design: Reclimbing the Complexity Curve," Western Institute of Computer Science, Stanford University. 1992.

[12] S. Lackey, J. Veres, and M. Ziegler, "Supercomputer expands parallel processing options," in *Computer Design*, pp. 21-27, 1985.

[13] D. E. Lenoski, "The Design and Analysis of DASH: A Scalable Directory-Based Multiprocessor," Ph.D., 1991.

[14] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy, "The DASH Prototype: Implementation and Performance," in *19th International Symposium on Computer Architecture*, Queensland, Australia, IEEE/ACM, 92-103, 1992.

[15] E. McLellan, "The Alpha AXP architecture and the 21064 processor," *IEEE Micro*, vol. 13, no. (3), pp. 36–47, 1993.

[16] MMS, "MMS "D" series thin-film interconnect products," Micro Module Systems, June 1993.

[17] E. Rothberg, J. P. Singh, and A. Gupta, "Working Sets, Cache Sizes and Node Granularity Issues for Large-Scale Multiprocessors," in *20th Annual International Symposium on Computer Architecture*, San Diego, CA, IEEE/ACM, 1993.

[18] D. P. Seraphim and R. C. L. A. C.-Y. Li, *Principles of Electronic Packaging*, 1989, New York, New York: McGraw-Hill, Inc. 1989.

[19] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. L. Hennessy, "Load balancing and data locality in parallel N-body techniques," Technical Report CSL-TR-91-496, Stanford University, 1991.

[20] J. P. Singh, W.-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," in *Workshop of the International Symposium on Computer Architecture*, Toronto, Canada, IEEE/ACM, 1991.

[21] A. J. Smith, "Cache Memories," in *ACM Computing Surveys*, 473-530, 1982.

[22] M. D. Smith, "Tracing with Pixie," Technical CSL-TR-91-497, Stanford University, Computer Systems Laboratory, Nov. 1991.

[23] G. Sohi and M. Franklin, "High Bandwidth Data Memory Systems for Superscalar Processors," in *Proc. 4th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, 53-62, 1991.

[24] SPEC, "SPEC Benchmark Suite Release 2.0," System Performance Evaluation Cooperative, 1992.