The Benefits of Clustering in Shared Address Space Multiprocessors: An Applications-Driven Investigation

Andrew Erlichson, Basem A. Nayfeh,
Jaswinder P. Singh †, and Kunle Olukotun

Computer Systems Lab
Stanford University
Stanford, CA 94305

† Department of Computer Science
Princeton University
Princeton, NJ 08544

Abstract

Clustering processors together at a level of the memory hierarchy in shared address space multiprocessors appears to be an attractive technique from several standpoints: Resources are shared, packaging technologies are exploited, and processors within a cluster can share data more effectively. We investigate the performance benefits that can be obtained by clustering on a range of important scientific and engineering applications in moderate to large scale cache coherent machines with small degrees of clustering (up to one eighth of the total number of processors in a cluster). We find that except for applications with near neighbor communication topologies this degree of clustering is not very effective in reducing the inherent communication to computation ratios. Clustering is more useful in reducing the the number of remote capacity misses in unstructured applications, and can improve performance substantially when small first-level caches are clustered in these cases. This suggests that clustering at the first level cache might be useful in highly-integrated, relatively fine-grained environments. For less integrated machines such as current distributed shared memory multiprocessors, our results suggest that clustering at the first-level caches is not very useful in improving application performance; however our results also suggest that in an machine with long interprocessor communication latencies, clustering further away from the processor can provide performance benefits.

Key Words and Phrases: Clustering, Applications, Shared Memory.

1 Introduction

Clustering is an architectural technique used in shared address space multiprocessors that allows clusters of processors to share a particular level of the memory hierarchy. Processors within a cluster can communicate data with each other more efficiently than with processors in other clusters. The motivation for clustering comes from a consideration of both parallel application behavior and the technologies with which parallel processors are built. Clustering has the potential to reduce the average memory access time of parallel applications whose processes tend to access similar portions of the address space. These processes will tend to find data in the shared memory or cache of the cluster rather than outside the cluster. Parallel processors are also built using a hierarchy of packaging technologies, including single chip, multiple chips on a board, and multiple boards in a cabinet. Typically, the bandwidth and latency of communication degrades as more levels in the packaging hierarchy are traversed. Thus, there is an incentive to implement the processors and the shared portion of the memory hierarchy of a cluster using as few of the packaging levels as possible. With present packaging and semiconductor technology, it is possible to cluster multiple processors on a single board. These processors might share the secondary cache or main memory level. As integration density increases, it will be possible to integrate a small number of processors and a shared first-level cache on a single chip.

In terms of application performance there are both advantages and disadvantages to clustering. The advantages are twofold. First, larger degrees of clustering reduce the cold and communication misses in a program through implicit prefetching and obviated communication. Second, clustering also has the potential to reduce capacity misses through overlap of the working sets of clustered processors. The main disadvantages of clustering are the increased contention at the clustered resource and the interference among the reference streams of the clustered processors, particularly when the clustered level of the hierarchy is a cache with small associativity. Two points are useful to note. First, as the clustering is done further away from the processors in the memory and interconnection hierarchy, both the performance benefits and the costs of clustering decrease. For example, successful prefetching into a shared second-level cache is potentially less beneficial than prefetching into a first-level cache, but contention and interference are less dramatic as well. Second, both the benefits and costs increase as the number of processors in the cluster increase.

The goal of this study is to provide a quantitative characterization of parallel application behavior in clustered machine organizations. This study focuses on inherent application characteristics and how amenable they are to clustering. In addition, we also approximate the specific implications for clustering at the first-level cache in the memory hierarchy. Clustering at the first-level cache is possible in processors which have external first-level caches such as those produced by Hewlett Packard [1]. However, other forms of clustering are possible, such as a shared second-level or a shared memory/bus. The specific results for these clustering configurations will depend on latencies, associativities, and contention. In the future we will study realistic clustering configurations at the first-level cache and at other levels in the memory hierarchy, in detail to provide insight into the effect of lower-level design decisions on the performance of clustered systems. The study is application-driven: it ignores multi-programming and operating systems issues, and studies a suite of realistic applications, important computational kernels and applications running stand-alone. To make the study tractable, we focus our attention on the following restricted problem: Given a fixed number of processors P with a fixed total amount of cache and enough physical memory to fit the problems of interest, what are the potential benefits of clustering the processors at some level of the memory hierarchy?

We first examine the inherent sharing characteristics of the applications and what these imply for the potential benefits of clustering, in the absence of capacity, contention or interference artifacts. We do this by simulating infinite per-processor or per-cluster caches with realistic miss latencies. This shows the potential benefits of prefetching and obviated communication. Then, we examine the impact of finite

capacity, still in the absence of contention. These results examine the performance benefits of overlap in working sets of the applications at different cache sizes. They are useful both for characterization and because they can immediately help designers make coarse decisions about the level of the memory hierarchy at which it makes sense to cluster. Finally, we focus on one level of clustering, sharing the first-level cache, in some detail to see whether the potential benefits observed are retained when realistic concerns like contention and the increased access time of a clustered cache are introduced.

The rest of this paper is organized as follows. Section 2 describes the potential benefits and drawbacks of clustering in more detail. Section 3 describes our experimental methodology, including the simulation environment, the choice of applications and data set sizes, and the experiments we perform. Section 4 discusses the implications of application sharing patterns for inherent communication. Finite capacity effects are examined in Section 5, and Section 6 presents some concrete examples with contention and latency effects. Finally, Section 7 summarizes the main points of the paper.

2 The Benefits and Drawbacks of Clustering

In this section we will abstractly consider two types of clusters, a shared cache cluster and a shared main memory cluster. A *shared cache cluster* has processors sharing a cache backed by main memory. The clusters are then connected together with some general purpose interconnect or a bus. A *shared main memory cluster* has individual processor caches connected by a snoopy bus with the backing shared main memory. Again the clusters are connected together via a general purpose interconnect. We consider the benefits of clustering in these two types of systems before narrowing our investigations to shared cache clusters with distributed directories.

The primary benefit of clustering is that it increases the number of memory references that can be satisfied closer to the issuing processor, and hence within the cluster. The decrease in the cluster miss rate comes from three sources: prefetching, reduced inherent or false sharing communication across clusters, and overlapped working sets. We will discuss each of these sources of miss rate reduction in more detail.

The reduction in miss rate from prefetching arises when two processors in the same cluster reference a nonlocal data item in temporal proximity. Since the first reference prefetches the data item into the cluster for the second reference, only the first reference generates communication outside the cluster. Prefetching effects occur both for the same data item as well as for different data items that are captured by the same cache line.

In an invalidation based cache coherence protocol, such as the one we assume, the decrease in miss rate from reduced inherent or false sharing communication across clusters is a direct result of the decrease in invalidations. Some invalidations that would have gone to another cluster now stay within the same cluster since the processor that would have had its cache invalidated is within the cluster. If caches are shared, the invalidations are eliminated entirely and subsequent accesses by other processors within the cluster are potentially converted from misses to hits. In a clustered memory architecture, the invalidations are sent to processors that have copies of the data item, whether inside or outside the cluster, but ownership is kept within the cluster if all cachers are within the cluster. Subsequent accesses by other processors within the cluster are satisfied by cache to cache transfers, thereby avoiding the long latency of fetching the data item from another cluster.

Clustering exploits the overlapped working sets of different processors. In many parallel applications a large fraction of the working set is comprised of data that are read-only during a phase of computation. In a shared-cache organization, there is a single shared copy of this data in the cluster cache instead of a copy in each processor's cache. This reduces the number of capacity misses for a given size cluster cache when compared to the same size sum of individual processor caches. This beneficial effect will be most pronounced in shared first-level cluster cache organizations where the size of the individ-

ual processor caches is small. In a shared main memory cluster overlapped parts of the working sets are duplicated in the cluster but the parts replaced by one processor may not have been replaced by other processors in the cluster, providing cache to cache sharing opportunities.

The actual performance benefits of prefetching and reduced communication are limited by several factors. In the case of prefetching, the data can be invalidated or replaced after being prefetched but before being used. Data prefetched by a processor within the cluster but replaced before a subsequent processor uses the data must be fetched again. Similarly, data prefetched but invalidated by another cluster must also be fetched again if the data is used. Finally, data prefetched by a first processor may not be fetched far enough in advance of a second processor's reference so that the delay of the reference is completely hidden from the second processor. The replacement of prefetched data or data that would otherwise be spared invalidation because of clustering is due to finite cache capacity, and so is dependent on cache size. Invalidation by outside clusters to prefetched data is inherent in the application behavior, although possibly timing dependent. This effect will remain for infinite caches. The degree to which prefetching is successful in hiding the latency of non-local references is related to the latency of a non-local reference and the temporal proximity with which cluster processors reference the same data.

Communication misses are only reduced if the data item that remained in the cluster is not replaced or invalidated before use by another processor. In shared cache systems, any processor in the cluster can cause replacement of a data item before reuse. In clustered main memory systems only the processor cache that took ownership can replace the data item; however, any other cluster could invalidate the data before reuse. Finally, if there is no subsequent reuse an obviated invalidation has no hit rate benefits.

There are two potential drawbacks to clustering: it may increase the miss rate of the shared memory, and it increases the hit time of the shared memory.

Clustering has the potential to increase the miss rate. The increase in miss rate is caused by destructive interference. In shared cache systems, destructive interference occurs when one of the processors replaces data in the cache that another processor needs. The effect on performance of destructive interference is exacerbated in caches with low degrees of associativity because these caches have higher conflict miss ratios. In clustered memory systems destructive interference does not exist, since the caches are separate and it is not possible for a processor to replace the contents of another processor's cache.

The increase in hit time of the shared memory affects both shared cache and clustered memory systems; however it will have the greatest effect on performance in a shared cache system because references to the cache have a higher frequency than references to main memory. In a shared cache system, providing the cache bandwidth to support multiple cache accesses per cycle requires a multiple-port, non-blocking, multi-banked cache [10]. Such caches have higher hit times than single ported, single bank caches [5]. Furthermore, when bank conflicts occur there are further increases in the hit time. Longer primary cache hit times will degrade performance by increasing the CPU cycle time or increasing the latency of loads in CPU cycles. In clustered memory systems the caches are typically connected to the cluster memory via a snoopy bus. The snoopy bus increases the latency of fetching data from the memory because it adds arbitration, queuing and electrical delays.

3 Experimental Methodology

In this section we describe and explain the experimental methodology used to generate the results in Sections 4, 5, and 6. We begin by describing the architectural assumptions and the simulation methodology used. Then, we describe the applications that are used in our simulations.

3.1 Architectural Assumptions and Simulation Methodology

In this clustering study, the architecture that we simulate is a shared address space multiprocessor with single shared-caches and distributed directories. The architecture is shown in Figure 1. It consists of nodes containing processors clustered around a shared cache. The nodes are connected by a high speed network, with the memory physically distributed among them. The cluster caches are kept coherent using an invalidation based protocol. The directory is implemented as a full bit vector with replacement hints. In all the simulations we perform, the total number of processors in the system is fixed at 64, while the number of processors per cluster varies. We investigate 1, 2, 4, and 8 processors per cluster configurations. The cache line size is 64 bytes in all our experiments.

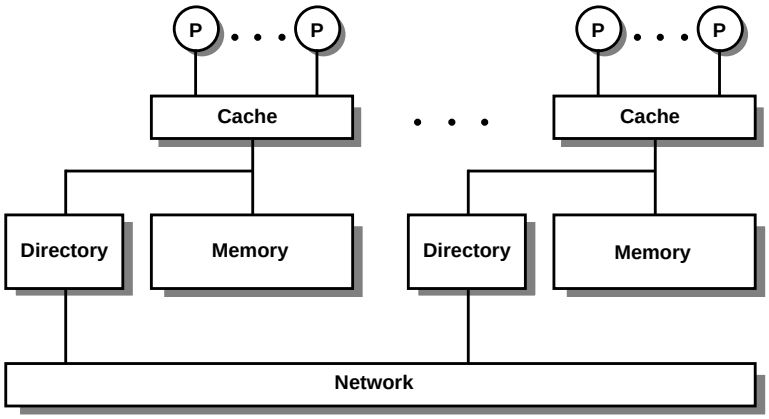

Figure 1: The Simulated Architecture

Misses are broken up into 3 categories, READ, WRITE and UPGRADE. READ misses occur when the processor makes a read access and does not find the data in the cache. WRITE misses are those where a write access does not find the data in the cache. UPGRADE misses are those where a write finds the data in the cache but in the SHARED state. Our cache states are INVALID, SHARED and EXCLUSIVE. READ misses always fetch the data in the SHARED state. Only READ misses are assigned latency, as the processor blocks waiting for the result. It is assumed that the latency of WRITE and UPGRADE misses could be completely hidden by store buffers and a relaxed consistency model.

READ misses to lines pending in the cache from outstanding READ or WRITE misses are said to MERGE MISS and will block until the associated data returns to the cache. The directory supports three cache states for a line, NOT_CACHED, EXCLUSIVE, and SHARED.

The latency of memory accesses used in our study are shown in Table 1. The latencies of local and remote misses are consistent with the relative processor speeds, DRAM speeds and network speeds of shared address space multiprocessors that have been built, such as the Stanford DASH Multiprocessor [3]. Future distributed shared memory multiprocessors will have longer latencies. We therefore examine the effect of longer latencies on the potential of clustering for communication misses using infinite caches. Latencies with finite caches are more complex, since we are only using a single level of cache. The impact of capacity and conflict misses could be reduced by using a larger secondary cache as a backup, and the interactions then depend on several cache parameters. However, communication misses are inherent and are not helped by using a deeper cache hierarchy. For this reason we will present results on the effect of longer latencies for communication misses only, and comment on caveats where appropriate.

In Table 1, “home” refers to the cluster at which the memory associated with the reference is allocated. Memory references not cached at a cluster must be sent to the home. The home can satisfy

Table 1: Latency of Memory Operations.

Memory Operation	Latency in Cycles
Hit in cache	1
Miss to local home, satisfied by home cluster (Dir = SHARED OR NOT_CACHED)	30
Miss to local home, satisfied by remote cluster (Dir= EXCL)	100
Miss to remote home, satisfied by home (Dir = NOT_CACHED OR SHARED)	100
Miss to remote home, satisfied by third party cluster (Dir = EXCL)	150

the reference directly in all cases except when the data is dirty in a remote cluster. A reference is local to the home if the memory is allocated at that referring cluster. References that are not local to the home require two network hops and take 100 cycles. References requiring three hops, because the data is dirty in another cluster take 150 cycles. Memory is allocated to clusters when first touched on a round robin basis. Some application programs explicitly place data when such placement improves performance. All stack references are allocated locally.

To exclude the effect of conflict misses from the performance characterizations, the caches that are simulated are fully associative caches with an LRU replacement policy. While caches are not fully associative in practice, we are interested in characterizing the inherent behavior of the applications in a clustered architecture rather than evaluating the performance of specific implementations. For this reason we do not want to include the effect of conflict misses that are due to limited associativity. Such lower level, real-system, issues are of course important in finally determining whether to cluster or not, and will be the subject of future work.

The event-driven simulator used to generate our performance results is based on Tango-lite [2]. This simulator produces application execution times by simulating with single cycle cache hits. We use this base shared cache hit time for our initial results presented in Sections 4 and 5. We then account for increases in hit time, due to sharing, using a procedure outlined in Section 6.

3.2 Applications

We use a set of applications that are representative of many parallel scientific, engineering and graphics computations. The applications also display a range of communication patterns, memory referencing characteristics, and working set sizes. The applications and the input data set size used in our simulations are listed in Table 2. MP3D is the one example that is not representative of a class of well-written applications for parallel machines, but is chosen to demonstrate the effects of unstructured high-volume read-write communication and large working sets. Detailed descriptions of the applications can be found in [6] [7] [8] [4] [11]. The applications are not modified to explicitly restructure algorithms to take advantage of the new level in the memory and communication hierarchy introduced by clustering. However, processes are assigned to processors in ways that tend to keep communication within a cluster to the extent possible without modifying the parallel algorithms.

Table 3 lists the major communication patterns of the applications, and describes the sizes of their important working sets and how these working sets scale with data set size n. The working set sizes for the data set sizes we simulate are indicated in parentheses. We provide brief descriptions of the applications below.

Table 2: Applications and Problem Sizes

Application	Representative Of	Problem Size
Barnes	Hierarchical N-body codes	8192 particles, $\theta = 1.0$
FFT	Transform methods, high-radix	64K complex points, radix N
FMM	Fast Multipole N-body Method	8192 particles
LU	Blocked dense linear algebra	512-by-512 matrix, 16-by-16 blocks
MP3D	High-comm. unstructured accesses	50,000 particles
Ocean	Regular-grid iterative codes	130-by-130 grids, 25 grids
Radix	High-performance parallel sorting	256K integer keys, radix=256
Raytrace	Ray tracing in computer graphics	Balls4
Volrend	Volume rendering in computer graphics	Human head from CT scan

Table 3: Communication Structure and Working Set Sizes

Application	Major Communication Pattern	Working Set Size	Working set as a function of n & p
Barnes	Low volume, unstructured, but hierarchical	Relatively Small (12KB)	very slow:$O(\log n)$
FMM	same as above	small (4KB)	constant
FFT	All-to-all, structured	small (4KB)	slow: $O(\sqrt{n})$
LU	Low communication, along row and column	small (2KB)	constant
MP3D	High communication, unstructured	large	$O(\frac{n}{p})$
Ocean	Nearest-neighbor, multigrid	size of local partition of grid	$O(\frac{n}{p})$
Radix	All-to-all, relatively unstructured	two: one small, one large	large is $O(\frac{n}{p})$
Raytrace	Read only, unstructured	large	unclear
Volrend	Read only, quite unstructured	quite small	$O(\sqrt[3]{n})$

Barnes simulates the evolution of galaxies using the Barnes-Hut hierarchical N-body method. It represents the space containing the particles as an octree, and processors traverse the octree partially once for each particle they own. Communication miss rates in realistic situations are low, and the communication is unstructured. The working sets are quite small, and overlap substantially because processors overlap in the parts of the tree they touch. FMM is similar to Barnes in these respects, but has a smaller working set.

FFT is a one-dimensional n-point Fast Fourier Transform. The n points are organized as a sqrt(n) by sqrt(n) matrix, of which each processor is assigned a contiguous set of rows. Each processor computes a one dimensional FFT on its rows. The communication is in a blocked matrix transpose, in which each processor reads a different block of data from every other processor.

LU is a blocked LU factorization of a dense matrix. The processors are organized in a grid. The communication volume is low, and is along rows and columns of the processor grid. The working set is essentially a single block, which is 16-by-16 8-byte elements in size, and is disjoint for different processors.

MP3D is our communication stress test. It is a particle-in-cell code that is written with vector rather than parallel machines in mind. The communication volume is large, and the communication patterns are very unstructured and are read-write in nature.

Ocean is a regular grid nearest-neighbor application with a multigrid solver. Every processor is assigned a square subgrid of every grid, and traverses its subgrid communicating with its neighbors at the boundaries. The important working set is simply the size of a processor's partition of a grid, and the working sets are disjoint.

Radix is a sorting kernel that sorts a large number of integers. The communication phase is composed of processors using the values of their keys to write these keys into random locations in a shared array that is distributed among the processors.

Raytrace and Volrend are applications from computer graphics. Both have a pixel plane that is initially divided among processors in the same manner as the grid in Ocean. However, dynamic task queues are used, and processes steal groups of pixels from other processes' queues when they become idle. Processors write only pixels that they are assigned or that they steal. The main data structure in both programs is a large volume data set that is read only and is distributed randomly among memories. The communication volume due to sharing of the read-only data sets and false sharing of the pixel grid is quite small. The difference in the two applications is that the rays that a processor shoots through its assigned pixels do not reflect in Volrend, but do so in Raytrace. Thus, Raytrace has much larger and more unstructured working sets. Both applications impose a shared octree data structure on the volume for efficiency.

To keep simulation times tractable, many of the problem sizes we simulate are quite small for 64-processor runs. However, we use 64 processors since several of the applications are well-tuned enough to provide very low communication miss rates with fewer processors, particularly with our 64 byte cache lines, and higher communication rates emphasize the benefits of clustering. It is important to note that the reduction in communication volume due to clustering is in almost all cases dependent on the communication patterns, so that as long as these patterns stay the same the percentage reduction in memory stall time would be the same even if the communication volume were lower or higher. Using a large number of processors with a relatively small problem does tend to inflate load imbalance and synchronization wait times, as we shall see.

One limitation of our study is that the applications we use, other than MP3D, are all quite highly optmized for parallel performance, for example to reduce communication and increase locality of data reference. This is again because we want to understand the implications of inherent application characteristics. Real programs may be less optimized in several important ways: (i) algorithmic: less optimal assignment of computation to processors, resulting in more inherent communication in the program, or not using techniques like blocking (ii) poorer design of data structures, which can lead to worse data locality (more difficult data placement, worse false sharing, etc.) and even more communication due to non-inherent sources, (iii) poor alignment and distribution of data structures among physical memories, even though data structures are designed well for this purpose, which can again lead to much more communication, and (iv) locality and contention problems with seemingly less important variables. The greater communication generated by these programs can lead to more benefits from clustering, while the larger working sets may lead to less benefits from overlapped working sets. Studying the impact of these levels of optimization, and the extent to which clustering increases the robustness of a machine to less optimized programs would also be interesting for future work.

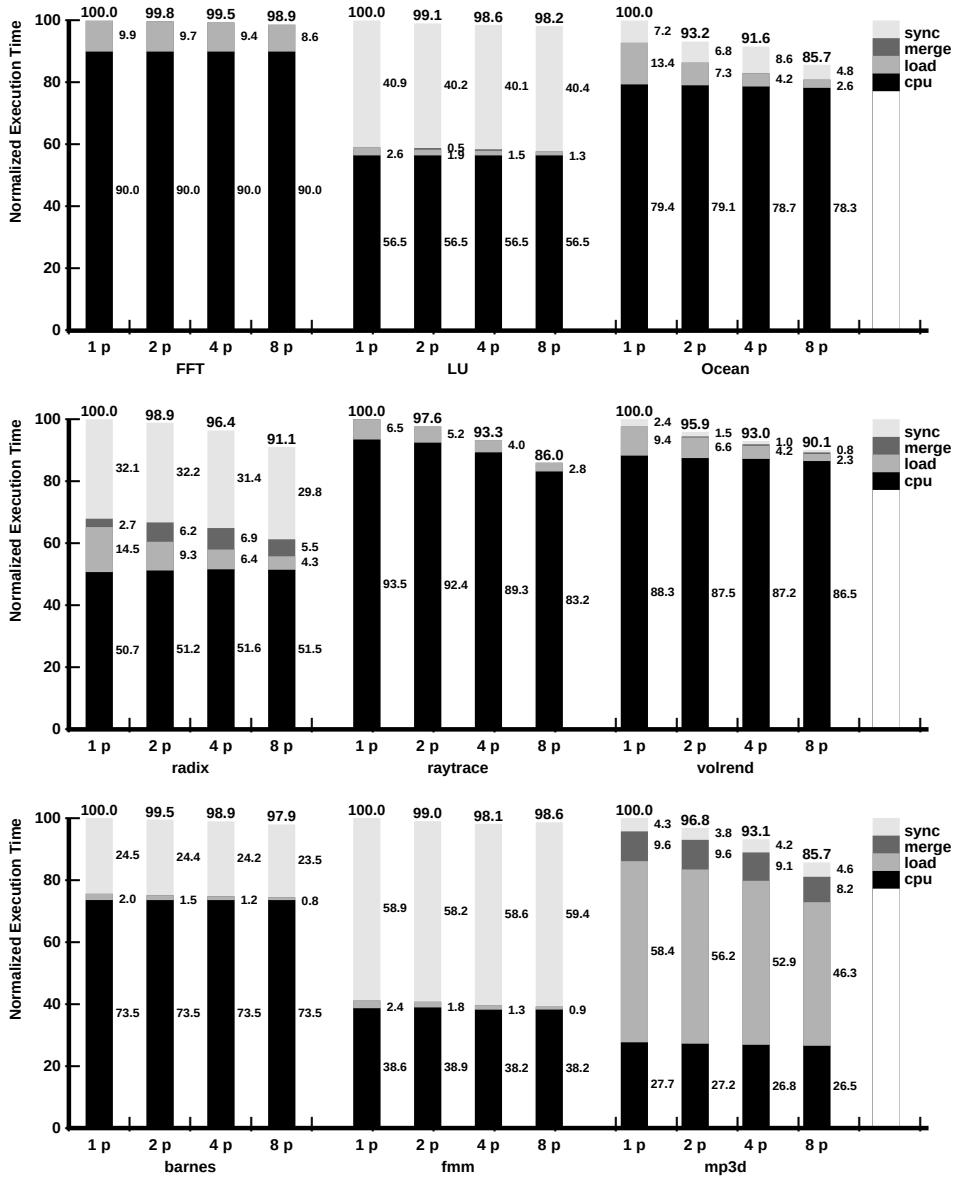

Figure 2: The Benefits with Infinite Caches

4 Implications of Clustering for Inherent Communication and Cold Misses

We begin our study of clustering by investigating the ability of clustering to reduce communication by eliminating invalidation and cold misses. To do this we evaluate the performance of a shared cache system with infinite caches. The use of infinite cluster caches makes it possible to eliminate all other sources of cache misses except (a) compulsory (cold) misses that are caused by the first access to the line or (b) invalidations from other clusters. This makes it possible to present a clear picture of the inherent sharing behavior of the applications and the performance benefit that clustering achieves from prefetching and reduced invalidations. As mentioned earlier we ignore contention, but use realistic miss penalties. Realistic miss penalties are necessary because it is important to consider the how much time elapses between references to the same data item in order to accurately assess the performance benefits of prefetching.

The execution times for the applications for clusters sizes of 1, 2, 4 and 8 processors sharing infinite caches with the miss penalties presented in Section 3.1 are shown in Figure 2. Each execution time is normalized to that of the one processor cluster. Each of the execution times is divided into CPU busy time, load miss stall time, load merge stall time and synchronization wait time. Load merge stall time represents the time the processor spends waiting for a cache line that has been prefetched by another processor but has not yet arrived.

Since the effectiveness of clustering in reducing communication depends on the communication patterns and topology of the application, we orient our discussion around these. We begin by examining the applications that have regular and well-structured access and communication patterns (LU, FFT, and Ocean), and then consider those with more unstructured access patterns (Barnes, FMM, Radix, Raytrace, Volrend and MP3D).

For small to moderate numbers of processors, the communication to computation ratios of the structured applications is typically very low, so the miss rates in infinite caches are low as well [6]. The results for LU show that the eight processor cluster has over 98% of the execution time of the single processor cluster with these latencies. Communication occurs in blocked LU decomposition when processors access the diagonal or perimeter blocks. Since processors in the same row (or column) of the processor grid access the same blocks, there is some prefetching benefit in a clustered cache. However, the communication volume is so low in LU factorization that reducing it has little effect on performance. A close examination of the two processor cluster execution time shows that load stall time is reduced by almost a factor of two. However, most of this time is replaced by merge stall time. This indicates that prefetching is occurring, but that the prefetch references do not happen soon enough. The reason for this is that processors in a cluster access the remote (e.g. diagonal) blocks at about the same time. The merge stall time is reduced as more processors are added to the cluster, since now there is a greater ability for the processors to stagger their communication of remote blocks.

FFT also does not benefit much from reduction in communication misses from clustering. In this case, the all-to-all processor communication means that clustering will only decrease the communication by a factor of $\frac{P-C}{P-1}$, where C is the number of processors in the cluster and P is the total number of processors in the machine. This is not a large reduction when $C \ll P$, and the performance impact of this reduction in communication is shown by the slight reduction in load stall time. FFTs with larger numbers of processors will have more communication relative to computation, but the communication patterns dictate that the benefits from clustering will not be too much better for machine configurations where $C \ll P$.

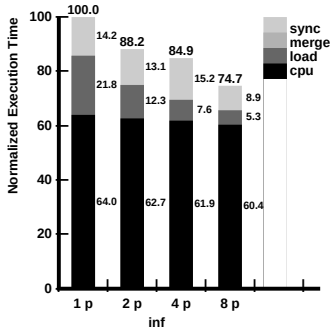

Figure 3: Ocean, Infinite Cache, Small Problem

In contrast to LU and FFT, Ocean shows a significant decrease in execution time as the size of the cluster is increased. The reason for this is the nearest neighbor communication. Ocean is representative of a large class of scientific computations that perform nearest-neighbor iterative computations on regular

grids. Communication in these applications occurs at the four borders of the square subgrid assigned to each processor. The processors are assigned to adjacent subgrids in the same row, thus doubling the size of the cluster doubles the number of subgrids that are local to a cluster and halves the amount of communication traffic to other clusters, except for subgrids at the grid boundary. This effect is demonstrated by the fact that the load miss time is reduced by almost half for each doubling in cluster size. However, this reduction does not have a dramatic effect on total execution time because the communication to computation ratio of even the relatively small 130-by-130 grid problem on 64 processors is quite low. Larger problems, more realistic for 64-processor machines, would have even lower ratios. Clustering may benefit more when smaller problems are run on larger numbers of processors. To investigate the effect that the resulting higher communication miss rates would have on clustered system, we simulated Ocean with a smaller 66-by-66 grid. The results, in Figure 3 show that the performance impact of clustering is indeed greater. However, there is also an increase in synchronization due to load imbalance. This leads us to the following conclusion for regular grid-based iterative computations. Clustering does reduce the communication to computation ratio substantially. However, the communication to computation ratio in these computations is a perimeter to surface area (or area to volume in 3-D) ratio, which is small for typical problem and machine configurations. When the problem is small enough relative to the number of processors for communication to be significant, clustering helps performance significantly although other overheads such as load imbalance and synchronization can become substantial. While we do not expect clustering to have much impact on large regular-grid problems, clustering may push out the number of processors that can be used effectively on a fixed problem size.

The performance of the unstructured applications on clustered systems with infinite caches falls into two regions: applications for which clustering provides no performance benefits and applications for which clustering provides modest benefits. Barnes-Hut and FMM fall into the first class of applications in which there is almost no performance benefit as the size of the cluster is increased from 1 to 8 processors. These applications do not have much inherent communication and the reduction because of clustering is not nearly as large as for Ocean, so that clustering has almost no impact on overall performance. Larger numbers of processors would still show little gain in communication, and the synchronization wait times and load imbalance would dominate by the time the communication became large enough for clustering to help. The benefits for Volrend and Raytrace are somewhat larger but still not very large. The prefetching effects of clustering reduce the number of cold misses to the volume data, but communication rates are low enough (and likely to stay that way even for larger problems and machines) that performance is not helped by more than 10% even with 8-way clustering. Radix sort shows significant prefetching effects, particularly on the shared histograms used to determine the sorting permutations, but like in LU the merge times are significant (since processors in a cluster are accessing the same histogram at the same time) and the performance benefits small. Also, the communication in the permutation phases of the radix sort, while less structured is more like the all-to-all communication in FFT.

MP3D is the application that has high communication rates and unstructured read-write communication on its shared space cells. The relative benefits of clustering in reducing communication are small, but since communication time is a large fraction of execution time, the performance benefits are proportionally much larger than in other applications and reach about 15% with 8-way clustering. MP3D shows clearly that even though the benefits of clustering in reducing communication relative to a one-processor per cluster situation are small in the applications with unstructured access patterns, the benefits of clustering on performance could still be significant if communication miss rates were high. However, the processor utilization and parallel speedups are very small in MP3D, which is a stress test rather than a representative parallel application. Overall, in the applications with unstructured access patterns the communication volume is generally very low, and the reduction in communication volume due to clustering not so large as for computations with more structured localized computation like Ocean. At the point at which communication becomes an important performance factor, other overheads such as those

of synchronization wait time and load imbalance dominate.

Next, we briefly examine the impact of clustering on communication misses using longer memory and intercluster latencies, still with infinite caches. The performance effect of communication misses is particularly sensitive to increases in miss penalty as these misses must be satisfied by another cluster.

Table 4 lists the new latencies used to obtain the results for four of our applications, as shown in Figure 4. These latencies are more typical of recently proposed large scale distributed memory machines.

Table 4: Longer Latencies for Memory Operations.

Memory Operation	**Latency in pClocks**
Miss to local home, satisfied by home cluster (Dir = SHARED OR NOT_CACHED)	120
Miss to local home, satisfied by remote cluster (Dir= EXCL)	400
Miss to remote home, satisfied by home (Dir = NOT_CACHED OR SHARED)	400
Miss to remote home, satisfied by third party cluster (Dir = EXCL)	600

Figure 4 shows that FFT still does not benefit much from clustering. Again, the all-to-all communication pattern implies that the factor by which communication misses are reduced is about $\frac{P-C}{P-1}$ for clusters of size C processors each.

Since the maximum number of processors in a cluster is eight, the reduction in communication misses is small. The net effect is simply a higher percentage of the execution time spent stalling for loads with the higher latencies, for all degress of clustering. Of course, the reduction due to clustering would increase as C increased relative to P.

The benefits of clustering increase, in terms of absolute execution time, for MP3D using longer latencies, since most of the time is spent stalled on loads. However, the improvement in the relative benefit due to clustering is not so pronounced since the CPU utilization is so low even with the lower latencies that the factor reduction in overall execution time is very close to the reduction in load stall time, which does not change much with latency. For Ocean and Volrend, however, CPU utilization is high to begin with and the relative benefits of clustering compared to the lower latency case are substantial; there is not a substantial increase in merge stall time which could arise with longer latencies.

Our general conclusion about the potential performance benefits from clustering from a reduction in communication is that they are small in well-written parallel applications. Several applications that have the highest communication rates (FFT's and radix sorts on large numbers of processors) have all-to-all communication, which is not reduced very much by clustering. Furthermore, we find that communication in the unstructured applications is either not reduced very much, or, with the base miss penalties, is not a performance bottleneck to begin with. With longer miss penalties the reduction in communication of some unstructured applications, such as Volrend, can result in significant performance improvements. Also, the communication in an important class of scientific applications with short-range communication in a near-neighbor topology, such as Ocean, can be reduced substantially by appropriate clustering of processors and assignment of processes to processors in clusters. However, these applications usually have low communication rates for typical problem sizes, since communication is proportional to the perimeter (or surface area in 3-D) of a partition while computation is proportional to area (or volume), and people tend to run much larger problems relative to machines than we have used here.

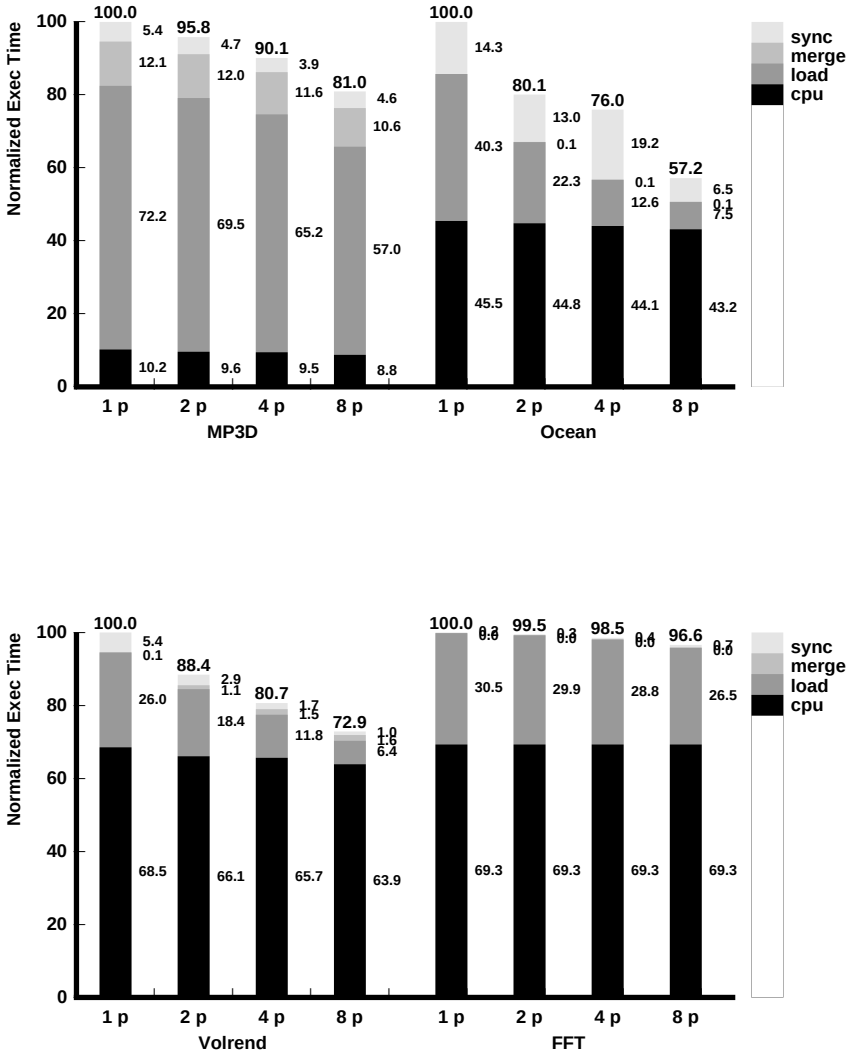

Figure 4: The Benefits with Infinite Caches and Longer Latencies

For example, iterative grid computations such as Ocean most often suffer from local capacity misses than from communication in realistic executions; clustering does not help these misses since they are to disjoint partitions of the grids (in fact it may hurt due to interference and cache conflicts) so its performance benefits are much reduced. Nevertheless, given small enough problem configurations and long enough miss penalties the performance improvements from clustering can be substantial. This suggests that the best argument for clustering is not so much that it would help in typical coarse-grained problem configurations that people run today on this scale of machine (64 processors), but that it pushes out the number of processors that can be used effectively on a problem. Introducing a dose of reality about the performance costs of clustering, as we shall do for shared first-level caches in Section 6, might substantially reduce even the benefits we are seeing. But first, let us examine how the best-case potential benefits

change when the effects of finite capacity are introduced.

5 The Effects of Finite Capacity

In this section we investigate the performance of shared cache clusters with finite capacity caches. We expect to see two competing effects due to the introduction of finite capacity. These are a decrease in miss rates due to the sharing of working sets among clustered processors, and an increase in miss rates from destructive interference between the reference streams of multiple processors. The caches used in our simulations are fully associative in order to characterize inherent behavior more than artifacts of specific low-level organization; therefore we do not expect to see much destructive interference. The goal of this section is to examine whether there are substantial benefits to be obtained by clustering due to processors within a cluster having overlapping working sets. Overlapping working sets have a prefetching effect on misses, and also reduce the total cache requirements making more efficient use of cache real estate, since the clustered cache can now be smaller than the sum of the individual unclustered processor caches.

All the structured scientific computations that partition their data sets into disjoint partitions (Ocean, FFT, etc) exhibit virtually no sharing of working sets across processes. We therefore do not present results for those applications. The results that are interesting from the finite capacity and working set overlap point of view are those for the applications with unstructured access patterns, and we present these in this section.

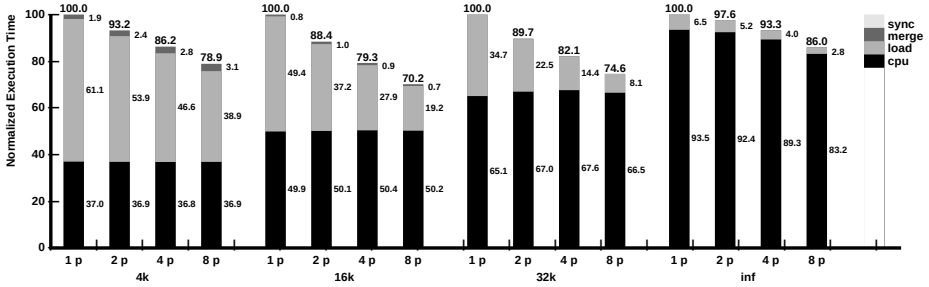

Figure 5: Finite Capacity Effects for Raytrace

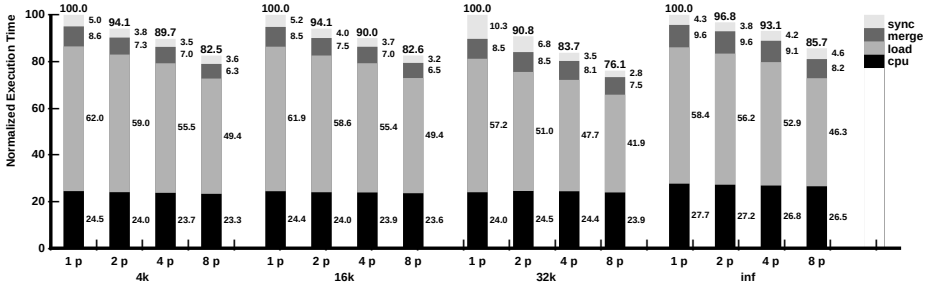

Figure 6: Finite Capacity Effects for MP3D

The execution times for these applications are shown in Figures 5, 6, 7, 8, and 9. This data is reported for finite caches sizes of 4KB, 16KB, and 32KB *per processor*, and for cluster sizes of 1, 2, 4 and 8 processors (i.e. for the set of bars labeled 16KB, the shared cache size with 8 processors is 8×16

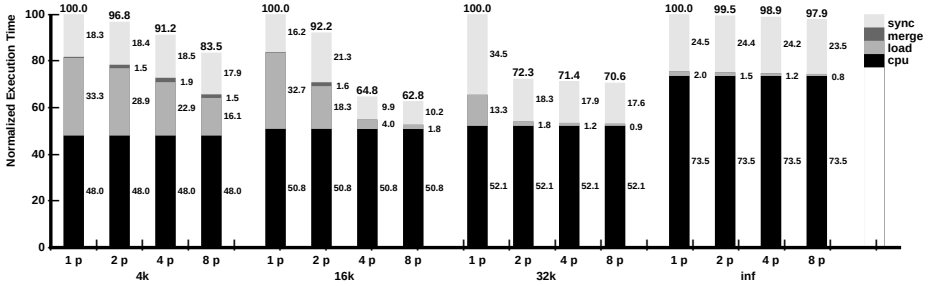

Figure 7: Finite Capacity Effects for Barnes

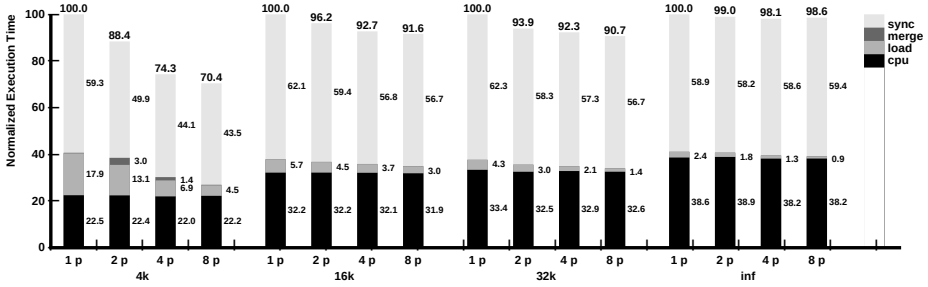

Figure 8: Finite Capacity Effects for FMM

or 128KB). The bars for every cache size (per processor) in every application are normalized to the 1 processor per cache time with that cache size per processor; i.e. the left most bar in every set is at 100%. We can conclude that overlapping of working sets is helping if the reduction in execution time due to clustering with the finite caches is larger than that with infinite caches (also repeated in the graph for convenience).

From the figures, and the estimated sizes of the working sets in Table 2 we come to the following conclusions. All the unstructured applications that heavily read a large shared data set in a phase of computation (Barnes, FMM, Volrend and Raytrace) show substantial advantages from overlapped working sets when the cache size without clustering is smaller than the working set. At some degree of clustering, the combined working set fits in the clustered cache owing to an overlap of individual working sets. When the fully associative cache is larger than the individual working sets to begin with, we clearly do

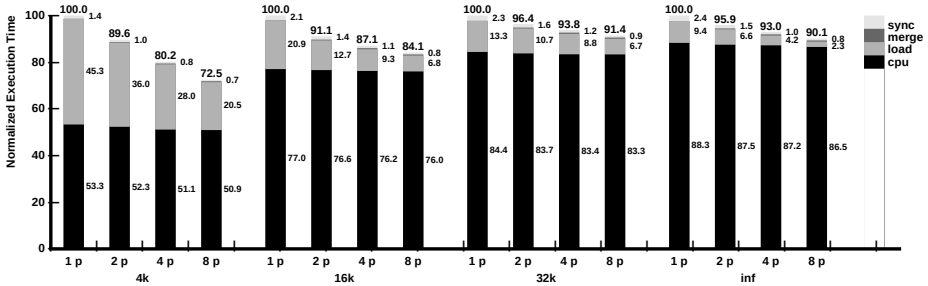

Figure 9: Finite Capacity Effects for Volrend

Table 5: Shared Cache Load Hit Latencies

Memory Operation	Latency in Cycles No Bank Conflict	Latency in Cycles Bank Conflict
Hit in cache (1 processor per cluster)	1	2
Hit in cache (2 processors per cluster)	2	3
Hit in cache (4 and 8 processors per cluster)	3	4

not see many more benefits than with infinite caches. The advantages of clustering in overlapping working sets (at least with fully associative caches) in this case are much larger than those in reducing inherent communication. For a cache size close to the uniprocessor working set, clustering the caches causes a steep drop in execution time at the point where the overlapped working set suddenly fits in the cache. This is because scientific and engineering applications often have sharply defined working sets [6]. But what is more interesting is that even for the 4KB caches that almost always start out much smaller than a processor's working set, the improvement in performance from overlapping working sets is substantial.

All the unstructured applications, with the exception of Raytrace, have relatively small working sets that grow slowly with problem size (see Table 2 and [6]). For example, we can see from the figures that the working set of FMM is at something close to the 4KB cache size, for Barnes is somewhat less than 16KB per processor, Volrend is near the 16KB range, and for Raytrace is larger and not so well defined [13]. The fact that the working sets are small and often overlap substantially indicates that clustering is likely to be useful at the first-level cache rather than at higher levels, and therefore perhaps in a fine-grained highly integrated environment that provides a reasonably sized cache shared among a small number of processors. The advantages are not likely to be reduced much by other effects such as contention and increased load delay slots in accessing the shared first-level cache, as we shall see in the next section. However, for a shared first-level cache, in particullar, they can depend substantially on associativity and cache conflicts, and these shared first-level caches will need to be highly associative. Lower associativity will also increase the cache size at which individual and combined working sets fit, and hence the point where clustering benefits substantially, and will make the cache sizes that fit the working sets less sharply defined. How limited associativity changes the behavior of clustering is something we are currently investigating. However, the fully associative caches we use here are cleaner and more appropriate for a study of the implications of inherent application characteristics such as this one.

6 Including the Costs of Sharing the First-Level Cache

To include the costs of sharing the first-level cache we must evaluate the performance effects of increasing the load hit latency of the cache. We use an estimation procedure which provides rough estimates of the performance costs involved.

We begin by varying the load hit latency in the shared cache from one and four clock cycles [5] depending on the number of processors per cluster. These are summarized in Table 5.

Each shared cache in our model has four banks for each processor in the cluster [5]. For example, a four processor cluster has a shared cache that is 16-way interleaved. We further assume that the processor emits a memory reference to a bank picked at random every cycle and that the processor stalls for a cycle if there is contention for that bank with another processor. In general, the probability C that

a memory reference will conflict with at least one other reference is:

$$C = 1 - \left(\frac{m-1}{m}\right)^{n-1}$$

where m is the number of banks and n is the number of processors. Table 6 shows the probabilities for contention at the shared cache for the cluster sizes we have simulated.

Table 6: Probabilities of Bank Conflict

Processor (n) Per Cache	**Banks (m)**	**Probability of Collision**
1	1	0.0
2	8	0.125
4	16	0.176
8	32	0.199

Finally, we assume that if a load reference is outstanding, then the processor will continue to execute instructions until a dependent instruction is encountered. At this point, the processor will stall until the load reference completes. Given these assumptions, we can analytically calculate the fraction of processor references that will conflict. The basic block profiling tool Pixie [9] can be used to find the relative increase in execution time caused by increasing the load latency from 1 to 2 cycles, 1 to 3 cycles, and 1 to 4 cycles. By taking a weighted average of the execution time increases, where the weights correspond to fractions of conflicting and conflict-free references and the execution times increases correspond to those associated with the increased load hit latency of conflict and conflict-free references to the cluster cache, we can calculate an overall execution time increase factor that accounts for the costs of clustering. These are shown in Table 7. By multiplying this factor by the execution times generated with the event-driven simulator we can estimate the performance effect of the shared cache.

Table 7: Compute Time Expansion Factors for Different Load Hit Latencies

Application	**1 Cycle Latency**	**2 Cycles Latency**	**3 Cycles Latency**	**4 Cycle Latency**
Barnes	1.0	1.036	1.078	1.123
LU	1.0	1.055	1.114	1.173
Ocean	1.0	1.061	1.144	1.243
Radix	1.0	1.051	1.102	1.162
Volrend	1.0	1.051	1.106	1.167
MP3D	1.0	1.08	1.14	1.243

The overall results for this procedure are shown in Tables 8 and 9. These tables show the relative performance of clustering for selected applications using only the base (i.e. smaller) miss latencies, and for two cache sizes: 4KB and infinite. The 4KB cache size is below the single processors working set size of applications. This cache size is used to evaluate the performance of clustering for some of the applications with working set advantages (Barnes, Volrend, MP3D) and one application (Radix) that does not exhibit significant working set advantages. The infinite cache size is used to evaluate the performance of clustering on Ocean, which is the application that exhibits the greatest reduction in communication.

Table 8: Relative Execution Time of Clustering with 4KB Caches.

Application	1-way cluster	2-way cluster	4-way cluster	8-way cluster
Barnes	1.0	0.99	0.95	0.88
Radix	1.0	1.01	1.02	0.96
Volrend	1.0	0.93	0.86	0.79
MP3D	1.0	0.96	0.93	0.86

Table 9: Relative Execution Time of Clustering with Infinite Caches.

Application	1-way cluster	2-way cluster	4-way cluster	8-way cluster
Ocean	1.0	0.99	1.04	0.99
LU	1.0	1.03	1.06	1.05

For the infinite size cache we also present the data for the LU application that does not exhibit much reduction in communication from clustering.

The conclusions that can be made from Table 8 and Table 9 are that for small cache sizes, the working set advantages of the applications are sufficient to provide performance benefits for clustered implementations in most cases, even when costs are factored in. The inclusion of shared cache costs increase the execution of these applications from 2% to 9%. However, with infinite cache sizes the low inherent communication means that even for Ocean which showed the greatest communication reduction from clustering, more than 4-way clustering makes performance worse, at least with our base set of latencies.

7 Summary and Conclusions

We have examined the performance benefits of clustering in shared-address-space multiprocessors. We found that owing to typical communication topologies in structured and unstructured computations, clustering is not very effective in reducing the inherent communication to computation ratios except in near-neighbor problems where the communication to computation ratio is typically low anyway. The communication patterns of many applications, both structured and unstructured, are not amenable to very large reductions in off-cluster communication volume due to small degrees of clustering, particularly when algorithms are not greatly restructured specifically for clustering. At least for our base latencies, the small advantages essentially go away when realistic issues like contention and the increased hit time for accessing a shared cache are factored in. However, clustering can be useful for small problem configurations for applications with highly localized (e.g. near-neighbor) communication as latencies become larger.

Clustering can be more useful due to the overlap in the working sets of clustered processes in unstructured computations. We have seen that it can improve performance substantially for small shared first-level caches. However, since the working sets of most such applications are small and grow slowly, sharing caches has greatest applicability in a highly-integrated, fine-grained system with clustering at the first-level cache. Investigating this scenario carefully requires looking at contention issues, the effects of increased delay slots and compiler scheduling, and the destructive interference due to limited associativity, and we plan to do this in the future. For less highly integrated or fine-grained systems such as current distributed shared memory machines, one configuration where clustering may help is when caches

have very low associativity but are not shared (i.e. no destructive interference), and conflict misses in one processor's cache are satisfied by the cache of another processor in the same shared-memory cluster (cache-to-cache sharing [3]) that also referenced those data [12]. We also plan to follow up this study focusing on inherent application characteristics with more detailed studies of different configurations and real-system issues, as well as less optimized applications. However, for shared-cache systems of this scale (64 processors), we expect experiments with more realistic systems to show that sharing a first-level cache requires high associativity, and that the decision of whether to cluster in small degree or not should be based not on application benefits but on engineering and packaging constraints.

8 Acknowledgments

This research was supported by ARPA contract N00039-91-C-0138. Kunle Olukotun is supported partially by a grant from the Powell Foundation.

References

[1] Tom Asprey, et. al. Performance Features of the PA7100 Microprocessor. *IEEE Micro*, June 1993, pp. 22–35.

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

[3] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The DASH Prototype: Implementation and Performance. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992, pp 92–103.

[4] Jason Nieh and Marc Levoy. Volume Rendering on Scalable Shared Memory MIMD Architectures. *Proc. Boston Workshop on Volume Visualization*, October 1992.

[5] Basem A. Nayfeh and Kunle Olukotun. Exploring the Design Space for a Shared-Cache Multiprocessor. In *Proc. 21st Annual International Symposium on Computer Architecture*, 1994..

[6] Edward Rothberg, Jaswinder Pal Singh and Anoop Gupta. Working Sets, Cache Sizes, and Node Granularity for Large-Scale Multiprocessors. In *Proc. 20th Annual International Symposium on Computer Architecture, 1993.*

[7] J.P. Singh, W.-D. Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. *Computer Architecture News*, 20(1):5-44, March 1992.

[8] Jaswinder Pal Singh et al. Load balancing and data locality in parallel hierarchical N-body simulation. Technical Report CSL-TR-92-505, Stanford University, February 1992. To appear in Journal of Parallel and Distributed Computing.

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

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

[11] Susan Spach and Ronald Pulleyblank. Parallel Raytraced Image Generation. *Hewlett-Packard Journal*, vol. 43, no. 3, pages 76–83, June 1992

[12] Wolf Weber, "Scalable Directories for Cache-Coherent Shared-Memory Multiprocessors" Ph.D. Thesis, Stanford University, 1993.

[13] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture* , pages 24-36, June 1995.