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

Technical Report: CSL-TR-94-632

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. We find that in general clustering is not very effective in reducing the inherent communication to computation ratios. Clustering is more useful in reducing working set requirements 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 is not very useful in improving application performance, and the decision about whether or not to cluster should be made on the basis of engineering and packaging constraints.

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 of the cluster rather than outside the cluster. Parallel processors are also built using a hierarchy of packaging technologies. These are a 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. Next generation processors that manage their own secondary cache will likely support directly connectable system interface busses that will allow shared main memory clusters to be constructed easily. 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 behavior and performance there are both advantages and disadvantages to clustering. The advantages are twofold. First, clustering reduces the cold and communication misses in a program through prefetching and obviated communication. Second, clustering also has the potential to reduce the capacity misses through overlap of the working sets. 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, the performance benefits and costs of clustering decrease. For example, successful prefetching into local shared second level cache is potentially less beneficial than prefetching into the first-level cache, and contention and interference decrease as well. Second, as the number of processors in the cluster increase both the benefits and costs increase.

In light of these issues, one way to state the general machine organization problem for a designer is as follows: Given a certain budget and a certain implementation technology, how should one best organize a multiprocessor to maximize performance? This problem involves questions of how to divide the budget among machine resources such as processors, cache, memory and network interface, how the machine will be used (in batch mode or in time-shared multiprogrammed mode), what the appropriate workloads are for the machine, the interactions of different machine organizations with these workloads, and engineering issues regarding latency, bandwidth and packaging.

The goal of this study is to provide a quantitative characterization of parallel application behavior in clustered machine organizations. This characterization can be used in conjunction with latency, bandwidth and engineering constraints to make decisions about clustering. In addition, we also study the specific implications for clustering at the first level cache in the memory hierarchy. The study is application-driven: it ignores time-sharing and operating systems issues, and studies a suite of realistic, important computational kernels and applications running in batch mode. 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. Then, we examine the impact of finite capacity, still in the absence of contention. These results are used to show 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 which level of the memory hierarchy it makes sense to cluster. The memory hierarchy is viewed as a hierarchy of caches, much like a cache only memory architecture or COMA, This is a clean abstraction which allows clustering at any level of the hierarchy. 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. A shared main memory cluster has individual processor caches connected by a snoopy bus with the backing shared main memory. The main memory is assumed to be an effectively infinite attraction memory of a flat COMA style machine[17]. We consider the benefits of clustering in these two types of systems before narrowing our investigations to non-COMA 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, and reduced 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 causes a cluster memory miss. 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 the latter case the prefetching effect is dependent on cache line size and application data layout.

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 is a direct result of the decrease in invalidations. Invalidations that would have gone to another processor node now stay within the same cluster. In a system with shared first level caches 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, but ownership is kept 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.

A reduction in the size of the clustered working sets reduces the miss rate of the cluster memory. 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 individual processor caches is small. In a shared main memory cluster working sets are still duplicated but the parts of the working set replaced by one processor may not have been replaced by other processors, 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[15]. Such caches have higher hit times than single ported, single bank caches[9]. 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, queueing 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 memory multiprocessor with shared caches and distributed directories. The architecture is shown in Figure 1. The architecture consists of nodes of processors clustered around a shared cache. The nodes are connected by a high speed network. The memory is distributed amongst the clusters as in the Stanford DASH architecture. 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. In our simulations we investigate 1, 2, 4, and 8 processors per cluster configurations.

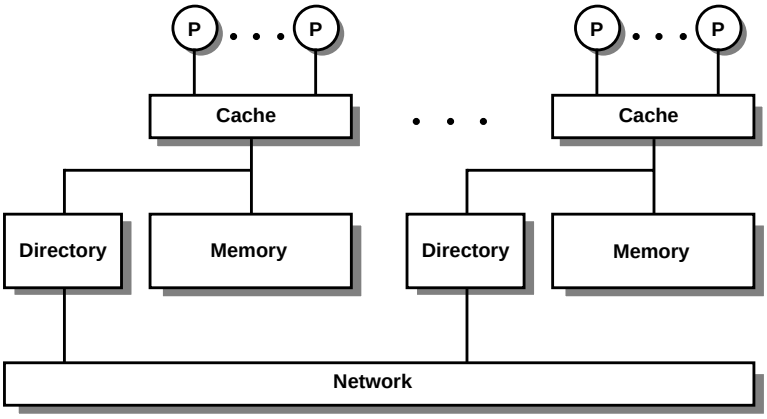

Figure 1: The Simulated Architecture

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 till the associated data returns to the cache. For simulation simplicity, invalidations occur instantaneously, possibly invalidating a line still pending in the cache. The directory supports three cache states for a line, NOT_CACHED, EXCLUSIVE, and SHARED.

Table 1: Latency of Memory Operations.

Memory Operation	Latency in Cycles
Hit in cache (1 processor per cluster)	1
Hit in cache (2 processors per cluster)	2
Hit in cache (4 and 8 processors per cluster)	3
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 latency of memory accesses used in our simulations are shown in Table 1. The primary cache hit latency varies between one and four clock cycles to account for the increases in the hit time of the shared cache as the number of processors per cluster increases. These latency values are based on previous studies of shared primary cache architectures[9]. The latencies of local and remote misses are consistent with the relative processor speeds, DRAM speeds and network speeds of modern systems.

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 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. In practice, fully associative caches are difficult to build; however, we are interested in characterizing the behavior of the applications in a clustered architecture rather than evaluating the performance of specific implementation. For this reason we do not want to include the effect of conflict misses that are due to limited associativity.

The event-driven simulator used to generate our performance results is based on Tango-lite[5]. This simulator produces application execution times by simulating with single cycle cache hits. We use an estimation procedure to account for the increase in hit times from the shared cache. To apply this procedure, we assume that the shared cache has four banks for each processor in the cluster[9]. For example, a four processor cluster has 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. Lastly, we assume that the processor will not stall on a load instruction until the register destination of the load is used. Given these assumptions, we can calculate analytically the fraction of processor references

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

that will conflict. The basic block profiling tool Pixie can be used to find the relative increase in execution time of increasing the load latency from 1 to 2 cycles, 1 to 3 cycles, and 1 to 4 cycles [14]. By taking a weighted average of the execution time increases, where the weights correspond to fractions of conflict and conflict-free references and the execution times increases correspond to those associated with the 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. By multiplying this factor by the execution times generated with the event-driven simulator we can estimate the performance effect of the shared cache. The results of applying this performance estimation procedure are presented in Section 6.

3.2 Applications

We use a set of applications that are representative of many parallel scientific and engineering 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 read-write communication and large working sets. Detailed descriptions of the applications can be found in [10][11][13][8][16].

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 parenthesis. We provide brief descriptions of the applications below.

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

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	$O(\sqrt[3]{n})$

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 divided among processors in the same manner as the grid in Ocean, and processors write only their own assigned pixels. The main data structure in both programs is a large volume data set that is read only and is distributed randomly among processors. The communication volume due to sharing of the read-only data sets and false sharing of the pixel grid is 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, however, impose an octree data structure on the volume for efficiency which is shared, as well as, some small fraction of the volume data that a ray traces.

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 many of the applications are well-tuned enough to provide very low communication miss rates with fewer processors, particularly with our 64 byte cache lines, so that the benefits obtained from clustering would appear artificially small. Even now, the communication volume is relatively low, but large enough to reveal 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 the percentage reduction 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.

4 Implications of Clustering on Inherent Communication and Cold Misses

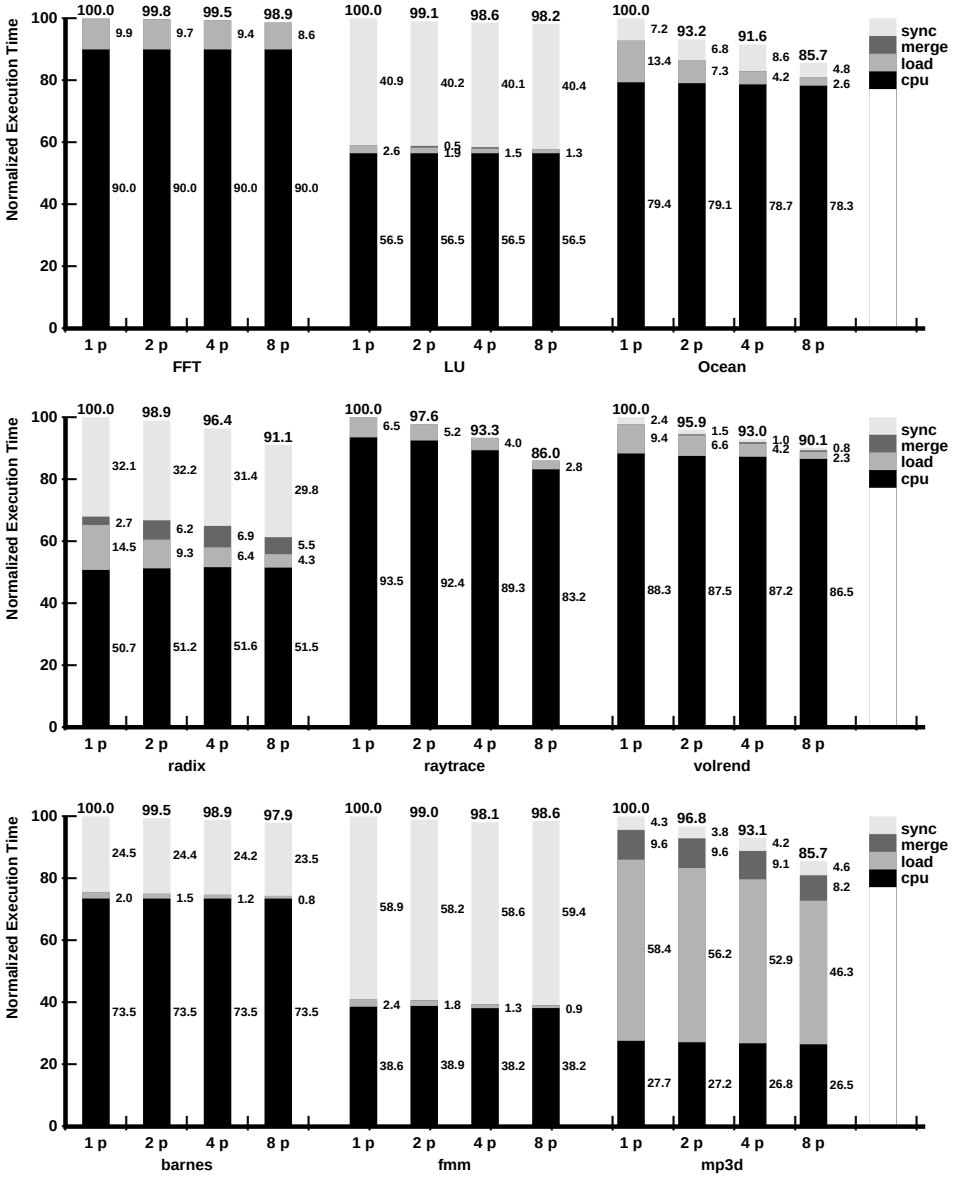

Figure 2: The Benefits with Infinite Caches

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 compulsory (cold) misses that are caused by the first access to

the line or 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. In order to fully isolate the performance benefits of clustering 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 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 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.

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[10]. The results for LU show that the eight processor cluster has over 98% of the execution time of the single processor cluster. 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. Even though there is nearest neighbor communication in LU, the communication volume is so low 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 more than 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 $\frac{P-C-1}{P}$, where C is the number of processors in the cluster and P is the total number of processors in the machine. The small 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, but the communication patterns dictate that the benefits from clustering will not be any better.

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 that the nearest neighbor communication in this application is being captured by the cluster cache. 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.

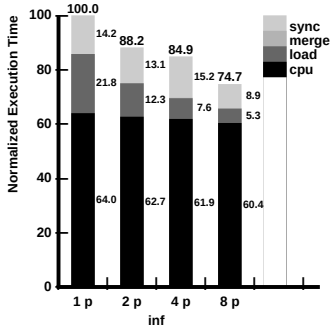

Figure 3: Ocean, Infinite Cache, Small Problem

Smaller problem sizes running on a larger number of processors would have higher communication miss rates. To investigate the effect that the 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, with the smaller problem size there is 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) 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. 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, in these applications the communication volume is generally very low, and at the point at which communication becomes an important performance factor, other overheads such as those of synchronization wait time and load imbalance dominate.

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 is not a performance bottleneck to begin with. The only case where we see a communication topology that is amenable to communication reduction by clustering is in fact a very important one for scientific computing: short-range communication in a near-neighbor topology (such as Ocean). However, these situations 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). The best argument that can be made for clustering in these cases is not so much that it would help in typical coarse-grained configurations that people run today, 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 (with no costs) change when the effects of finite capacity (replacements and the overlap of working sets) 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 (It will be useful to examine how these results change in the presence of limited associativity). 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 both inherent communication and capacity misses. In addition, they reduce the total memory requirements and make 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 exhibit virtually no sharing of working sets. We therefore do not present results for those applications. The results that are interesting from 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.

The execution times for the applications that exhibit significant working set advantages from clustering are shown in Figures 4, 5, 6, 7, and 8. This data is reported for finite caches sizes of 4 KB, 16 KB, and 32 KB per processor, and for cluster sizes of 1, 2, 4 and 8 processors. 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.

From the figures, and the estimated sizes of the working sets in Table 2 we come to the

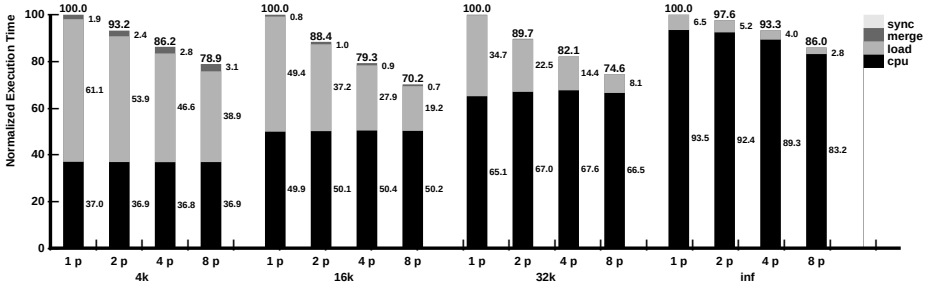

Figure 4: Finite Capacity Effects for Raytrace

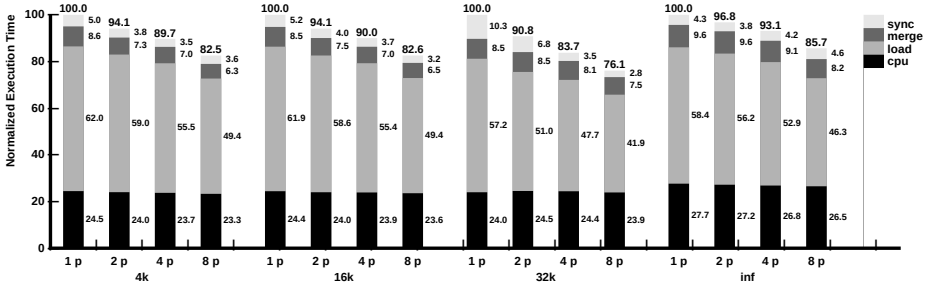

Figure 5: Finite Capacity Effects for MP3D

following conclusions. All the representative 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. When the fully associative cache is larger than the working set, we clearly don't 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[10]. But what is more interesting is that even for the 4K caches that almost always start out much smaller than a processor's working set, the improvement in performance from overlapping working sets is

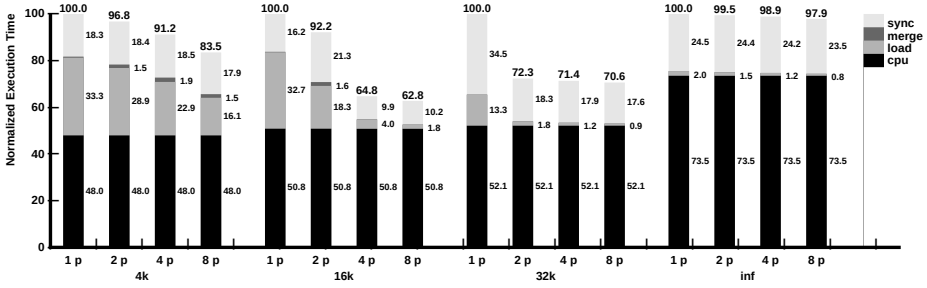

Figure 6: Finite Capacity Effects for Barnes

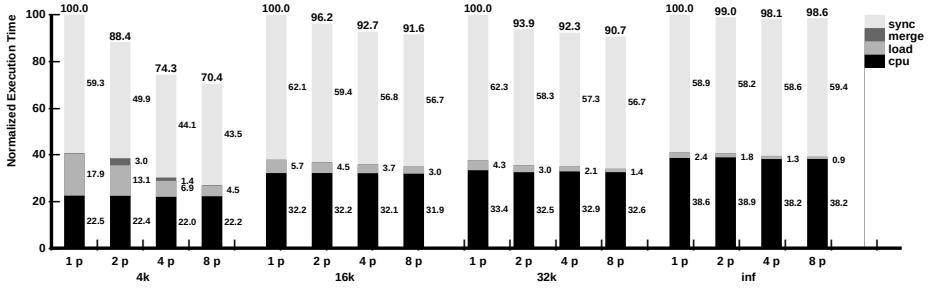

Figure 7: Finite Capacity Effects for FMM

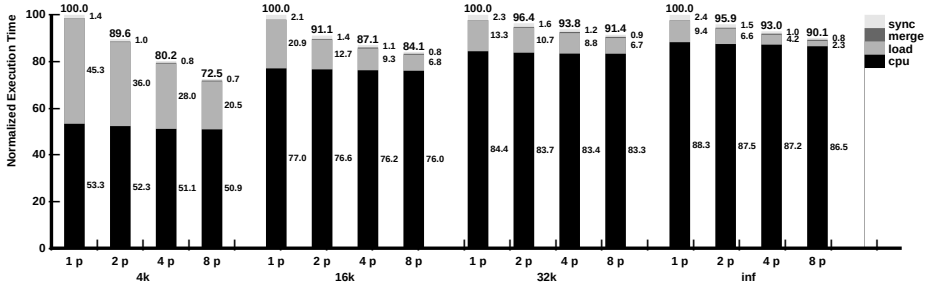

Figure 8: Finite Capacity Effects for Volrend

substantial.

All the unstructured applications (other than Raytrace) have relatively small working sets that grow slowly with problem size (see Table 2 and [10]). 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 16K per processor, Volrend is near the 16KB range, and for Raytrace is larger. 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 (we will, of course, need to look at caches with smaller associativity to validate this claim, and particularly examine their interference effects in the cases of structured access patterns as well), 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.

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 hit time of the cache. To do this we use the method outlined in Section 3. This method only provides rough estimates of performance costs associated with shared caches, but it is useful to get a more accurate assessment of performance of a first-level shared cache

To estimate the amount of contention at the multi-banked non-blocking cache, we assume that each processor makes a reference to the cache every cycle. If the reference stream is random, the probability C that any reference will conflict with at least one other reference is

Table 4: Probalities 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

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

where m is the number of banks and n is the number of processors. Table 4 shows the probabilities for contention at the shared cache for the cluster sizes we have simulated.

Table 5: Load Latency Execution Time Factors

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-sort	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

Table 6: 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-sort	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

Analysis of the programs with Pixie provides the execution time expansion factor for the various load latencies. These are shown in Table 5. We multiply the execution times by the appropriate factors to generate the data in Table 6 and Table 7. These tables show the relative performance of clustering for selected applications. We present the performance for two cache sizes: 4 KB and infinite. The 4 KB 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-sort) 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 only application that exhibits a reduction in communication. For the infinite size cache we also present the data for the LU application that does not exhibit a reduction in communication from clustering.

Table 7: 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

The conclusions that can be made from Table 6 and Table 7 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. The inclusion of shared cache costs increase the execution of these applications from 2% to 9%. However, for infinite cache sizes the lack of inherent communication means that even for Ocean which showed the greatest communication reduction from clustering, more than 4-way clustering makes performance worse.

The performance of radix-sort with a 4KB cache shows that in the worst case, a 4-way cluster, execution time is 2% greater. In contrast, the execution time of lu, increases up to 5% with an infinite cache. This suggests that if the per processor caches are small most applications will exhibit some performance improvement form clustering from overlapping working sets. This improvement can offset the costs of the shared first-level cache. However, the performance benefits of reduced communication are not sufficient to overcome the costs of a shared-first level cache.

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 small advantages essentially go away when realistic issues like contention and the increased hit time for accessing a shared cache are factored in.

Clustering is more useful due to the overlapping in the working sets of clustered processes in unstructured computations, and can improve performance substantially for small caches. However, since the working sets of most such applications are small and grow slowly, this has applicability mostly 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 more carefully, and we plan to do this in the future. For less highly integrated or fine-grained systems such as current distributed shared memory machines, we believe that the decision of whether to cluster or not should be based not on application characteristics but on engineering and packaging constraints.

8 Acknowledgements

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

9 References

[1] David R. Cheriton, Hendrik A. Goosen and Patrick D. Boyle. ParaDiGM: A Highly Scalable Shared-Memory Multi-Computer Architecture. *IEEE Computer*, February 1991.

[2] Jim Christy and David Wilkins. Parallel Ray Tracing Without Database Duplication. CS315B Project Report, Stanford University, June 1992.

[3] Helen Davis, Stephen Goldschmidt and John L. Hennessy. Multiprocessor Simulation and Tracing using Tango. *Proc. Intl. Conf. on Parallel Processing*, August 1991.

[4] Peter J. Denning. The working set model of program behavior. *Communications of the ACM*, vol. 11, no. 5, May 1968, pp. 323-333

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

[6] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[7] Daniel E. Lenoski et al. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. 17th Annual International Symposium on Computer Architecture*, pages 148-159, 1990.

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

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

[10] 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.*

[11] 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.

[12] Jaswinder Pal Singh, John L. Hennessy, and Anoop Gupta. Scaling parallel programs for multiprocessors: methodology and examples. *IEEE Computer*, July 1993.

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

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

[15] 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.

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

[17] Per Stenstrom, Truman Joe, and Anoop Gupta, Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures. In *Proc. 19th Annual International Symposium on Computer Architecture*, pages 80-91, May 1992.

[18] Andrew A. W. Wilson, Jr. Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors. In *Proc. 14th Annual International Symposium on Computer Architecture*, pages 244-252, 1987