10-1-1999

Impact of Heterogeneity on DSM Performance

Renato J. O.. Figueiredo
Purdue University School of Electrical and Computer Engineering

José A. B. Fortes
Purdue University School of Electrical and Computer Engineering

IMPACT OF HETEROGENEITY ON DSM PERFORMANCE

RENATO J. O. FIGUEIREDO
JOSÉ A. B. FORTES

TR-ECE 99-13
OCTOBER 1999

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Impact of Heterogeneity on DSM Performance

Renato J. O. Figueiredo and José A. B. Fortes
School of Electrical and Computer Engineering:
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285
{figueire,fortes)Qecn.purdue.edu

[0]This research was supported in part by NSF grants CCR-9970728 and EIA-9975275. Renato Figueiredo is supported by a CAPES grant.

Contents

List **of** Tables

List of Figures

Abstract

This paper explores area/parallelism tradeoffs in the design of distributed shared-memory (DSM) multiprocessors built out of large single-chip computing nodes. In this context, area-efficiency arguments motivate a heterogeneous organization consisting of few nodes with large caches designed for single-thread parallelism, and a larger number of nodes with smaller caches designed for multi-thread parallelism. This paper quantitatively studies the performance of such organization for a set of homogeneous multiprocessor programs from the SPLASH-2 benchmark suite. These programs are mapped onto the heterogeneous processors without source code modifications via static thread assignment policies. A constant-area simulation analysis shows that a 4-node heterogeneous DSM with 21 processors outperforms its homogeneous counterpart with 4 processors by an average of 36% for the studied multiprocessor workload, while having the same performance for sequential codes. Also studied are the implications of the degree of heterogeneity in the functional units of such heterogeneou.3 DSM on overall system cost and performance. This paper presents a sensitivity analysis based on a factorial design experiment that determines the relative impact of heterogeneity on performance. The studied benchmarks are affected, on average, primarily by heterogeneity in processor performance (59.3%), followed by cache sizes (18.2%), memory latency (14.6%) and network latency (5.6%).

1 Introduction

The predicted advent of billion-transistor chips [30] will enable the implementation of multiprocessors in a single chip. Large multiprocessors will then be possible by using single-chip multiprocessors as the building blocks. For a given silicon area (i.e. budget) the question arises as to how to design and organize such future multiprocessors. In this context, this paper shows quantitatively that heterogeneous distributed shared-memory multiprocessor designs outperform their homogeneous counterparts in the execution of unmodified multiprocessor workloads.

At a fundamental level, proposed billion-transistor chip designs differ in how resources are allocated to exploit parallelism. Devoting circuit area to complex structures capable of enhancing the performance of a single thread of code [26, 19, 31, 8] is appealing from a software perspective, since higher performance is obtained from unmodified sequential binaries. However, uniprocessor architectures that aggressively exploit instruction-level parallelism (ILP) require increasingly area-expensive structures.

Designing area-efficient structures that are replicated to expose parallelism to multiple threads of code [14, 17, 11] is appealing in terms of hardware design simplicity and efficiency. A chip-multiprocessor (CMP) exploits the area of a very large die by replicating smaller processing units and caches. Replication, in conjunction with design simplicity, allows for high clock rates of individual processing units and larger aggregate issue width due to more efficient area utilization [14].

Combining both kinds of chips can potentially lead to area-efficient multiprocessors capable of fast execution of both sequential and parallel codes. In this paper, a heterogeneous design that meets the goals of high area-efficiency and good performance of low-parallelism tasks is analyzed in the context of an implementation that uses multiple multiprocessor chips. The design consists of a hierarchy of processors and memories that includes a large number of simple processors for parallel computation as well as a few complex processors for fast execution of sequential and/or moderately parallel code (Figure 1(B)). Such an organization, also called HPAM (Hierarchy of Processor-And-Memory), was proposed and studied in [3, 2, 4, 5, 12].

Previous studies showed that, for message-passing programs with one or more degrees of parallelism, HPAM machines have higher cost-efficiency than conventional designs. However, the shared-memory paradigm has become increasingly important for parallel processing due to the availability of low-cost, bus-based multiprocessor nodes [7, 18] and of distributed shared-memory (DSM) protocols [34, 13, 28]. In particular, current trends in microprocessor design [1] suggest that microprocessors of the future will have enhanced support for multiprocessing and directory-based coherence protocols. The question arises as to whether a heterogeneous DSM (HDSM) organization would be able to efficiently execute unmodified homogeneous shared-memory programs. This paper addresses this question under the conservative assumption that there is no support for on-demand migration of code and data across hierarchy levels as a consequence of run-time determined degree of parallelism (DoP).

In summary, this paper differs from past work in several important ways:

- a shared-memory heterogeneous implementation is assumed and simulated
- unmodified, thread-based shared-memory multiprocessor programs are used for quantitative evaluation. and
- these programs are written in a style that hides run-time variations of parallelism

This paper makes two main contributions. The first one is a simulation-based performance analysis of HDSMs executing unmodified, shared-memory multiprocessor programs. To this end, three different static assignment schemes of homogeneous programs to heterogeneous nodes are quantitatively studied. The main conclusion from this analysis is that the HDSM configuration outperforms an equal-area homogeneous counterpart by an average of 36% for multiprocessor workloads. Another conclusion is that the static thread assignment that maximizes performance depends on application characteristics, particularly communication and synchronization. The second contribution is a quantitative assessment of how

heterogeneity in the design of the processor, memory and network subsystems impacts the performance of HDSM machines. One of the main conclusions is that the performance of HDSMs has low sensitivity to the speed of memories in the highly parallel levels.

The quantitative results reported in this paper were obtained through execution-driven simulation of shared-memory parallel scientific benchmarks from the SPLASH-2 suite [35]. Benchmarks are simulated individually to study single-program parallel speedup. The simulation environment used in the performance studies is based on a modified version of the Wisconsin Wind Tunnel-I1 multiprocessor simulator [22] that supports heterogeneity of different functional units. The performance criterion is application execution time, measured as the number of simulated target machine cycles.

The rest of this paper is organized as follows. Section 2 discusses the machine model assumed in this paper, and the experimental methodology used in the performance analysis. Section 3 presents a static approach to the mapping of homogeneous applications to HDSMs, introduces the benchmarks that are used in the performance analysis, and describes the simulation environment. Section 4 presents performance results and analysis. Section 5 discusses related work, and Section 6 presents conclusions. Appendix A describes the methodology used to validate the heterogeneous simulation environment.

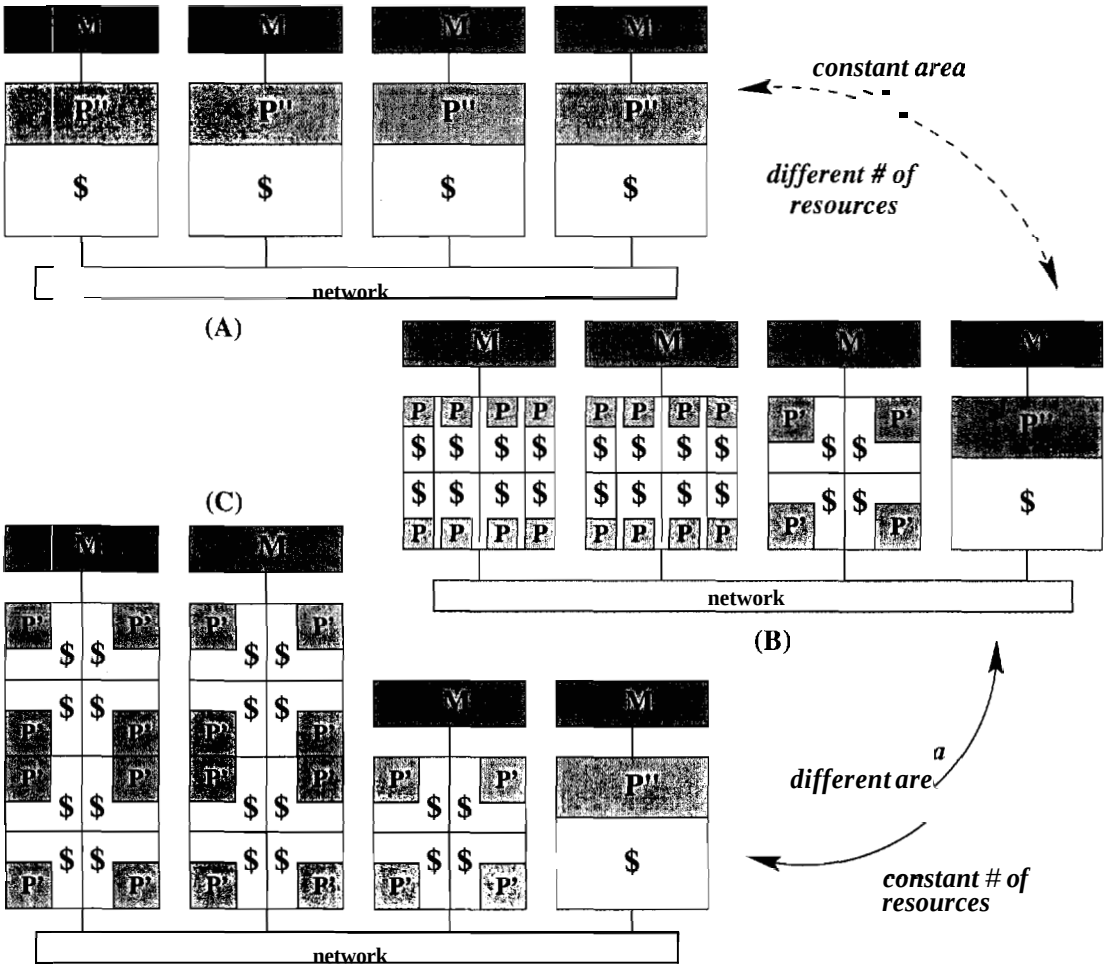

Figure 1: *Multiprocessors under study:* **(A):** *4-node homogeneous DSM with fast uniprocessor (P") nodes;* **(B):** *4-node HDSM with same silicon area as (A) where heterogeneity is present in processors and caches;* **(C):** *example of HDSM configuration (not constrained by equal-area requirement) used to analyze the impact of heterogeneity on performance.*

2 Experimental Methodology

2.1 Machine Model

The machine model assumed in this paper is a DSM whose nodes have single-chip multiprocessors with integrated on-chip directory-based coherence support and memory controller. Both homogeneous and heterogeneous DSM configurations are considered. The heterogeneous machine consists of three levels connected. via a point-to-point network (Figure 2). Each level, in turn, consists of one or more nodes. Each node consists of one or more processors, a remote-access device (RAD), and off-chip main memory, all connected by a bus. The RAD is responsible for providing a shared address space across the DSM nodes and maintaining coherence across remotely cached data.

The caches of CMPs are configured as in conventional symmetric multiprocessor shared-memory designs, where each processor has a private data cache [23]. All simulated caches are direct-mapped. Cache coherence is maintained via a bus snooping protocol inside the node, and via the *Stache* [28] replication policy together with a conventional invalidation-based directory protocol across nodes. Stache employs part of each node's DRAM memory as a large, fully-associative cache similar to the caches in cache-only memory architecture (Simple-COMA [29]) machines. This paper assumes that the stache protocol is handled by a dedicated hardware protocol controller.

Figure 2 depicts the heterogeneous DSM machine model assumed in this paper. The configuration shown in this figure is based on the processor-and-memory hierarchical design approach [3]: the number of processing elements increases from top to bottom levels (1, 4 and 16 processors in levels 1, 2 and 3, respectively), the cache sizes and the performance of processors and memories decrease from top to bottom levels.

Heterogeneity across machine levels is modeled by assigning different per-level values for the following architectural parameters:

#nodes(i): Number of nodes in level **i**.

- *#procs(i)*: Number of processors in a level-i node
- *clock(i)*: Clock period of a level-i processor

$size(i): Size of a level-i cache.

accesst(i): Level-**i** main memory access time.

- *latency(i,j)*: Network latency between levels **i** and **j**.

2.2 Performance Analysis Roadmap

The performance of HDSMs is studied from two different perspectives. In the first analysis (constant-area, Figure 1 dashed arrow), an HDSM is compared to homogeneous configurations under the assumption of constant total die area. The systems under comparison differ only with respect to the organization of the processing elements in each node. Thus, memory access times and network latencies are the same for the systems under comparison. The actual response times of memory and network transactions, however, may differ across heterogeneous nodes due to contention on both the memory bus and network interface (fully modeled in the simulations). This first analysis is divided into two parts. Subsection 4.1 compares a constant-area HDSM to a fast uniprocessor, and subsection 4.2 presents a speedup analysis of the constant-area HDSM with respect to a homogeneous multiprocessor of same area.

In the second analysis (constant-resources, Figure 1 solid arrow), the relative effect of heterogeneity of processor, memory and network on the performance of HDSMs is determined. In this analysis, only heterogeneous configurations are considered. The configurations under comparison have the same number

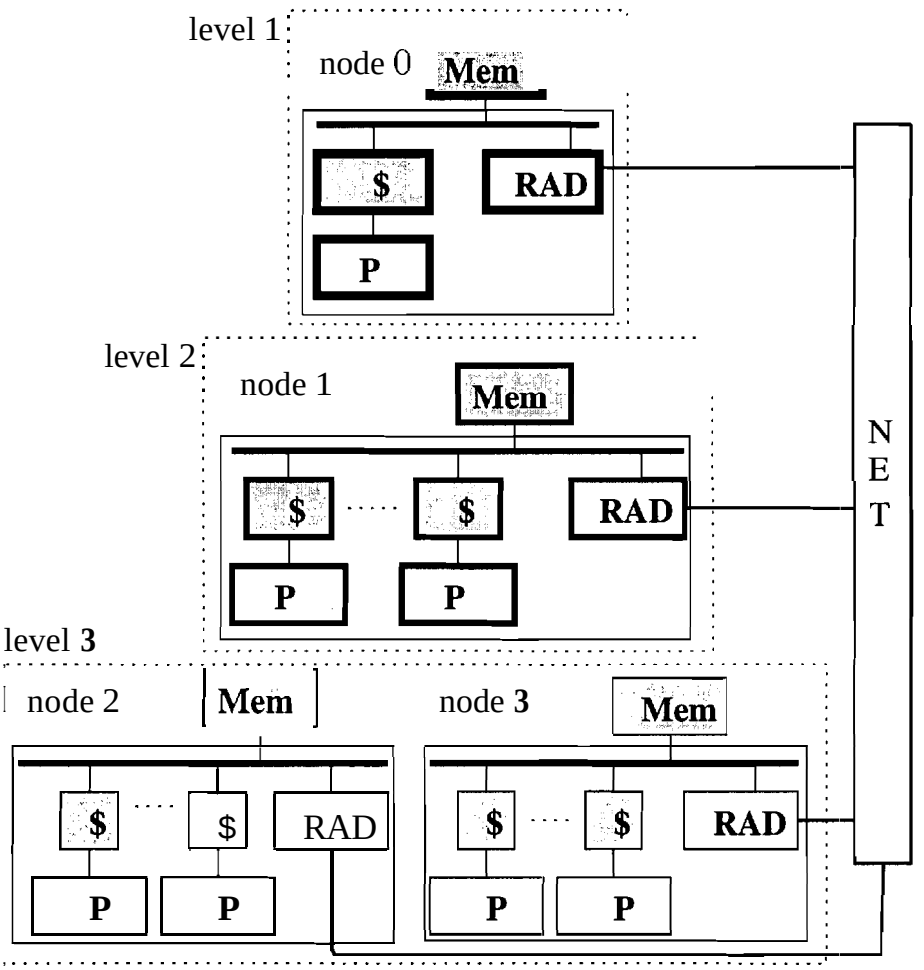

Figure 2: *4-node heterogeneous DSM with 3 levels, consisting of 1, 4 and 16 processors (only 2 processors are shown in level 2 and 4 processors in level 3). The boxes that represent each functional unit are drawn such that the wider the contour, the faster the functional unit; boxes representing caches are drawn such that the area of each box is proportional to the cache's size.*

of resources (nodes, processors, caches, memories and network), while the performance and capacity of each resource may vary across heterogeneous levels.

The use of slower parts in the parallel levels of an HDSM is motivated by potential savings in total system cost. The primary goal of the constant-resources analysis is to study the performance impact of using less expensive parts in the parallel levels of an HDSM. The design space of heterogeneous systems is studied by considering processor, cache, memory and communication hardware as dimensions along which a machine can be made heterogeneous. To analyze the impact of heterogeneity along each dimension on application performance a factorial design experiment is done. The methodology and results of this analysis are presented in Subsection **4.3.**

2.3 Heterogeneous Node Configurations

In the constant-area analysis, the machines under comparison differ only in the internal (on-chip) organization of processors and caches: the homogeneous nodes have a single, fast processor and large caches, while

	level i=1	level i=2	level i=3
#nodes(i)	1	1	2
#procs(i)	1	4	8
clock(i)	*clock(1)*	2*clock(1)*	4*clock(1)*
$size(i)	1MB	128KB	64KB
accesst(i)	56*clock(1)*	56*clock(1)*	56*clock(1)*
latency(i,j)	50*clock(1)*	50*clock(1)*	50*clock(1)*

Table 1: Three-level?four-node heterogeneous machine configuration. Values are normalized with respect to the clock *period* of the fastest processor *(clock(1)).* For network latencies, level j is defined as $j = 2, 3, 1$ for $i = 1, 2, 3$, respectively. The homogeneous machine under comparison has four fast (level-I) nodes.

in the heterogeneous machine there are also nodes with more processors and smaller individual caches. The organizations of the computing nodes are such that die area is bounded by the area of the high-performance node of the homogeneous machine. Point-to-point inter-processor network connections and standard interfaces to main memory are assumed to be the same across the heterogeneous nodes. Since computing nodes have the same die area and same external interfaces to memory and other processors, their packaging is assumed to be the same for all configurations.

Hence, in this analysis, heterogeneity is present only in terms of *#nodes(i)*, *#procs(i)*, *clock(i)* and *$size(i)*. Table 1 shows the values assumed for these parameters across the heterogeneous nodes, as well as the value:; of parameters corresponding to main memory and interconnection network latencies (*accesst(i)*, *latency(i,j)*) which are common to all nodes. The cache of the level-1 processor is dimensioned to hold the secondary working set of most SPLASH-2 [35] programs (Subsection 4.2 presents a small-cache analysis where cache sizes are smaller than the secondary working set). A base clock cycle of 1GHz is assumed for the level-1 processor. The main memory latency is assumed to be 56ns, and the interconnection latency between two computing nodes is assumed to be 50ns. The resulting average simulated ratio of remote/local memory latency is 4.8.

The choices of number and performance of processors and of cache sizes shown in Table 1 are based on the assumption of bounded-area chips. The heterogeneous design has a single high-performance uniprocessor node and three chip-multiprocessor (CMP) nodes (Figure 1(B)). The configuration of the high-performance uniprocessor node is based on a next-generation, 100-million transistor microprocessor, the Alpha 21364 [1]. The specifications of this microprocessor include the processor core of an Alpha 21264 [10], 1.5MB of level-2 cache, memory controller and directory protocol support, all integrated into a single die.

The configurations of processors and caches in the CMP nodes are motivated by the tradeoff between processor design complexity and performance discussed in Section 1. The actual parameters used to define the organization of these nodes are based on a case study of the area/performance tradeoff between two microprocessors from the Alpha family (Table 2). This case is presented in the remaining of this section.

Table 2 presents a comparison of the area and performance characteristics of the 21064 and 21264 designs. The number of transistors [9, 10] is used as a technology-independent estimate of die area; Spec95 results [33] are used as performance metrics. Both absolute and normalized (norm) values are reported in the table. The number of transistors is normalized to the 21064 design. The following paragraph describes how the normalized performance indices are obtained from the clock rates and the Spec95 indices.

The order-of-magnitude performance improvement in terms of Spec95 results observed for the 21264 is achieved via a combination of architectural improvements (higher ILP) and a higher clock rate. The better clock rate of the faster chip is highly dependent on advances in semiconductor process technology: the 21264 and 21064 under comparison are fabricated in $0.35\mu m$ and $0.75\mu m$ technology, respectively. It is therefore reasonable to assume that the simpler design can achieve a clock speed comparable to the

CPU	Clk	Transistors		$Perf_{int}$		$Perf_{fp}$	
	MHz	$*10^6$	norm	cint95	norm	cfp95	norm
21064	100	1.7	*1.0*	1.9	*1.0*	2.8	*1.0*
21264	575	15.2	*8.9*	30.3	*2.8*	47.7	*3.0*

Table 2: *Performance and transistor count of Alpha microprocessors. Spec95 results (cint95, cfp95) based on single-processor AlphaStation 200 and AlphaServer 8400 configurations. Normalized integer ($Perf_{int}$) and floating-point ($Perf_{fp}$) performances for processor P are given by the expression:* $norm(p) = \left(\frac{clk(21064)}{perf(21064)}\right) * \left(\frac{perf(P)}{clk(P)}\right)$.

ILP-enhanced design if fabricated under the same technology[1].

The normalized indices (with respect to the 21064 processor, Table 2) account for clock speed differences to yield an estimate of the relative performance between the two microprocessors under the assumption of same fabrication technology. By factoring out clock speeds, the normalized performance numbers provide an approximation to the speedup due to architectural enhancements.

The data in Table 2 show that a nine-fold increase in transistor count results in three-fold (normalized) speedups due to architectural enhancements. In other words, the same transistor budget of the high-performance processor can be used to design nine simpler engines with a third of the performance, under the assumption of equal clock speed. Based on these findings, the model of the CMP used in the third level of the heterogeneous machine conservatively assumes 8 processors, each with a quarter of performance of the level-1 uniprocessor. The level-2 CMP is modeled assuming the same quadratic area/performance relationship, yielding 4 processors, each with half of the performance of the level-1 uniprocessor. In this paper, heterogeneity of processor performance is modeled via scaling of clock rate. Table 1 shows the simulation parameters used to represent the heterogeneous processors under this model (a detailled discussion of the scaling model is presented in Subsection 3.1).

The level-2 and level-3 CMPs are assumed to have private L2 caches for each processor. The sizes of the private caches are obtained by scaling down the size of the level-1 uniprocessor cache (1MB). The scaling model assumes that the CMPs with 4 and 8 processors have private caches of sizes 1MB/8 and 1MB/16, respectively, effectively reducing the aggregate on-chip cache size by a factor of two. This assumption is conservative in accounting for potential increases in interconnection requirements for the multiprocessor design, since the large on-chip 1MB cache accounts for the majority of the die area of the high-performance uniprocessor [1]. Figure 1(B) depicts the organization of the resulting 4-node multiprocessor system.

[1] A simpler pipeline usually can be clocked at a higher rate [25]; this potential benefit is, conservatively, not captured in the normalization model.

3 Simulation of Homogeneous Applications and Heterogeneous Architectures

The set of homogeneous applications used in this study belongs to the SPLASH-2 [35] suite. It consists of the following benchmarks (and respective input sets): Barnes, a simulator of the interaction of systems of bodies in three dimensions using the Barnes-Hut hierarchical N-body method (16K particles); Cholesky, a blocked sparse Cholesky factorization kernel (tk15.0); FFT, a complex 1-D six-step FFT algorithm (64K points); **FMM**, another simulator of system of bodies using the adaptive Fast Multipole Method (16K particles); LU, a kernel that factors a dense matrix into the product of lower and upper triangular matrices (512x512 matrix); Ocean, a simulator of large-scale ocean movements based on eddy and boundary currents (258x258 ocean); Radix, an integer radix sort kernel (1M integers, 1024 radix); Raytrace, a 3-D renderer that uses ray tracing (car image); Water-Nsquared and **Water-Spatial**, applications that use two different algorithms to evaluate forces and potentials that occur over time in a system of water molecules (512 molecules).

These homogeneous applications are developed under a single-program, multiple-data (SPMD) model. Parallelism is expressed via PARMACS directives. In this model, the application distributes the workload evenly across N threads which are forked after an initialization phase, executed in parallel, and joined at the end of execution. Hence, these applications have been designed to exhibit a single DoP during their parallel execution, given by the total number of threads spawned. When a given application is executed in a homogeneous machine, each processor is assigned the same number of threads (typically one, for processors that do not have hardware multi-threading support). If the same (unmodified) application is executed across heterogeneous processors, the workload remains equally distributed across N threads, thus threads assigned to slower processors may take longer to complete than those in faster processors, and fast processors may become idle while waiting on data and/or synchronization from threads in slower processors.

Two solutions may be applied to increase the utilization of faster processors in this scenario: one is to redistribute the work across threads by modifying the application code, and the other is to redistribute the work by rneans of assigning more threads to more powerful processors. While the first :solution may involve extensive code analysis, the second solution (which is used in this paper) can be implemented with little programrning effort and/or operating system support. However, if the homogeneous thread assignment is one thread per processor, the second solution implies the creation of a larger number of threads. For a constant workload, this implies that the amount of communication and synchronization may increase for the same workload.

In order to map the homogeneous applications onto a heterogeneous architecture without requiring code analysis, and to investigate possible performance advantages of heterogeneous thread assignment schemes, three static thread assignment policies were considered:

1. single-thread assignment: assigns a single thread to each processor in the machine.

2. virtual-processor assignment: aims at improving workload distribution by assigning proportionally more threads to more powerful processors; it assigns *VP(i)* virtual processors to a physical processor i, where *VP(i)* is the ratio between the performance of processor *i* and the performance of the slowest processor in the system.

3. *single-level* assignment: assigns a single thread to each processor of a given machine level, while the remaining levels do not participate in computation.

For simulation purposes, in the work reported in this paper the static assignment policies were implemented not in the simulated operating system, but in each benchmark. Switch/case tests were added to the thread creation subroutine (PARMACS macro CREATE) to emulate the behavior of an operating system that supports the three assignment policies.

3.1 Simulation Environment

The simulation environment used to model heterogeneous DSMs is based on a modified version of the Wisconsin Wind Tunnel-I1 [22]. The modifications allow an HDSM machine with up to three levels to be defined. For each level, the number of nodes per level, the number of CPUs per node, the CPU speed, memory speed, cache size and inter-node network latency are specified via command-line options. The experiments conducted to perform the validation of this simulation environment, along the lines of the methodology applied to the validation of the original Wind Tunnel [27], are presented in detail in Appendix A.

In the simulation experiments, a processor with an average speedup due to ILP enhancements of n (with respect to a scalar processor) is approximately modeled as a scalar pipeline with clock rate scaled by n. Although this clock-scaling model has been found to introduce errors in execution time estimates [24], two reasons contribute to reduce the magnitude of potential errors. First, the number of aggressive ILP processors in the simulated multiprocessors is smaller than previously studied systems (4 in the homogeneous case and 1+4 in the heterogeneous case). Appendix A presents quantitative data obtained using the RSIM [24] simulator that shows reductions in the average error observed for a 4-processor system with respect to an 8-processor system from 74% to 24%. Second, the same simulation model applies to both heterogeneous and homogeneous systems. For a given benchmark, the potential errors arise in both configurations and should have little effect in relative comparisons.

The virtual-processor assignment requires support for the execution of multiple threads on processing nodes. The simulator models a coarse-grain multi-threading scheme based on voluntary context switches initiated by threads at synchronization points. The average context-switch overhead for the heterogeneous configuration is 870 processor cycles. The homogeneous configuration is not affected by this overhead since there is a single thread per processor.

4 Performance Analysis

In this section, simulation results for both conventional and heterogeneous designs are presented and analyzed. Subsection 4.1 analyzes the performance of HDSM under three static assignment policies with respect to a single processor. Subsection 4.2 analyzes the performance of HDSMs with respect to conventional multiprocessors with the same number of nodes (but different number of processors). Subsection 4.3 presents a sensitivity analysis which quantifies the relative impact of heterogeneity of processor, memory and network on the performance of HDSMs via a factorial design experiment.

4.1 Performance With Respect to Uniprocessor

In this subsection, the HDSM machine specified in Table 1 is compared to the high-performance (level-1) uniprocessor. This comparison determines which static assignment policy under study maximizes the parallel speedup achieved for each benchmark.

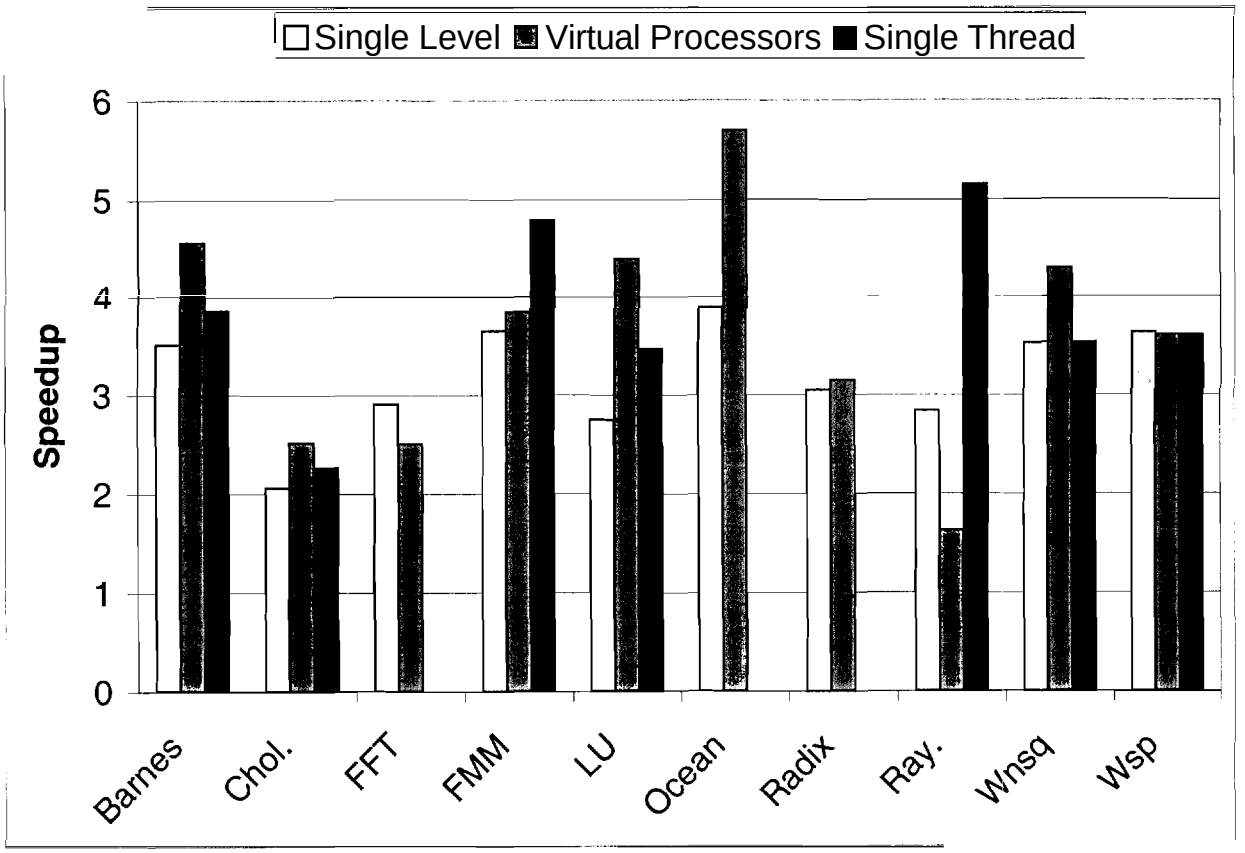

Figure 3: HDSM *speedups with respect to level-1 uniprocessor. Average best-case speedup is 4.12. The single-thread assignment does not apply to benchmarks requzring power-of-two processors (*FFT, Ocean *and* Radix)

Figure 3 presents the HDSM speedups with respect to the uniprocessor for the single-level, virtual-processor and single-thread assignment policies. The single-level scenario only consider:; level-3 assignment; single-level assignments to levels 1 and 2 result in worse performance than the level-3 assignment for all simulated SPLASH-2 benchmarks. The virtual-processor scenario assigns 4, 2 and 1 threads to processors in levels 1, 2 and 3, respectively, except for benchmarks requiring power-of-two number of threads, where 3 threads are assigned to level-2 processors.

Intuitively, the virtual-processor assignment scheme should provide the best performance, due to its load-balancing property. However, the simulation results show that none of the studied policies maximizes the speedup across the entire set of benchmarks.

The virtual-processor assignment achieves better performance than the other two schemes for Barnes, Cholesky, *LU*, Ocean and *Water-nsquared* due to better utilization of the level-1 and level-2 processors. The single-level assignment prevents these processors from participating in computation, while in the single-thread scheme fast processors remain idle while waiting for slower processors on synchronization points. The virtual-processor assignment successfully improves load balancing for these benchmarks by assigning heterogeneous workloads to processors, based on their relative performance.

For the remaining five benchmarks, the performance of the virtual-processor scheme ranges from comparable to inferior to the other two schemes. For the kernel *FFT*, the single-level assignment yields the best performance. The reason for this performance advantage lies in the communication characteristics of this application.

The single-level scheme assigns threads to two wide (8-processor) nodes, while in the virtual-processor case threads are assigned to all four nodes. For the highly communication-intensive transpose phase of FFT, the inter-processor communication overhead is higher in the 4-node configuration than in the 2-node case. For this benchmark, the benefit of faster communication outweighs the higher parallelism exposed by the virtual-processor assignment, and the single-level assignment becomes the policy that yields best performance. The same behavior is observed, to a lesser extent, in Radix and Water-spatzal.

The single-thread policy achieves the best performance for FMM and Raytrace. Single-thread performs better than the virtual-processor assignment in these cases due to the synchronization characteristics of the two applications and the coarse-grain multi-threading model assumed in the simulations, as discussed in the following paragraphs.

Previous work has shown that *FMM* spends a significant fraction of its execution time in synchronization [35] via locks and barriers. Since the simplistic multi-threading model assumed in the simulations only supports voluntary context switches (each taking, on average, 870 cycles), a thread ready to acquire a lock can be delayed by other threads in the same processor for a long period of time. The delayed thread, in turn, can potentially prevent dependent threads in other processors in the system to proceed. The larger number of threads required by the virtual-processor assignment, combined with the underlying context-switching model, can thus induce unnecessary serialization and have a negative impact on performance. In *FMM*, the context-switching overhead of the virtual processor assignment (four threads) on the level-1 processor accounts for 15.9% of the total execution time.

The benchmark *Raytrace* synchronizes through locks; as with FMM, the virtual-processor assignment fails to deliver good performance. However, *Raytrace* uses a dynamic task-queue algorithm that achieves good load balancing under the single-thread policy, rendering the virtual-processor assignment unnecessary. In this algorithm, threads continually fetch work from a shared pool of tasks. A thread executing in a fast processor is thus likely to execute more work than one executing in a slower processor, since its task completion rate is likely to be higher. This behavior is confirmed by measurements of the number of tasks executed by each heterogeneous processor in the system. The workload simulated in this paper has a total of 1024 tasks in the shared work-pool; the average number of tasks executed by threads of levels 1, 2 and 3 are 156, 94 and 31, respectively. The fastest processor thus performs, on average, 66% more work than each half-speed level-2 processor (and 408% more work than each quarter-speed level-3 processor) without requiring a virtual-processor assignment. Due to this dynamic load-balancing scheme, *Raytrace* achieves the best speedup across single-thread assigned benchmarks.

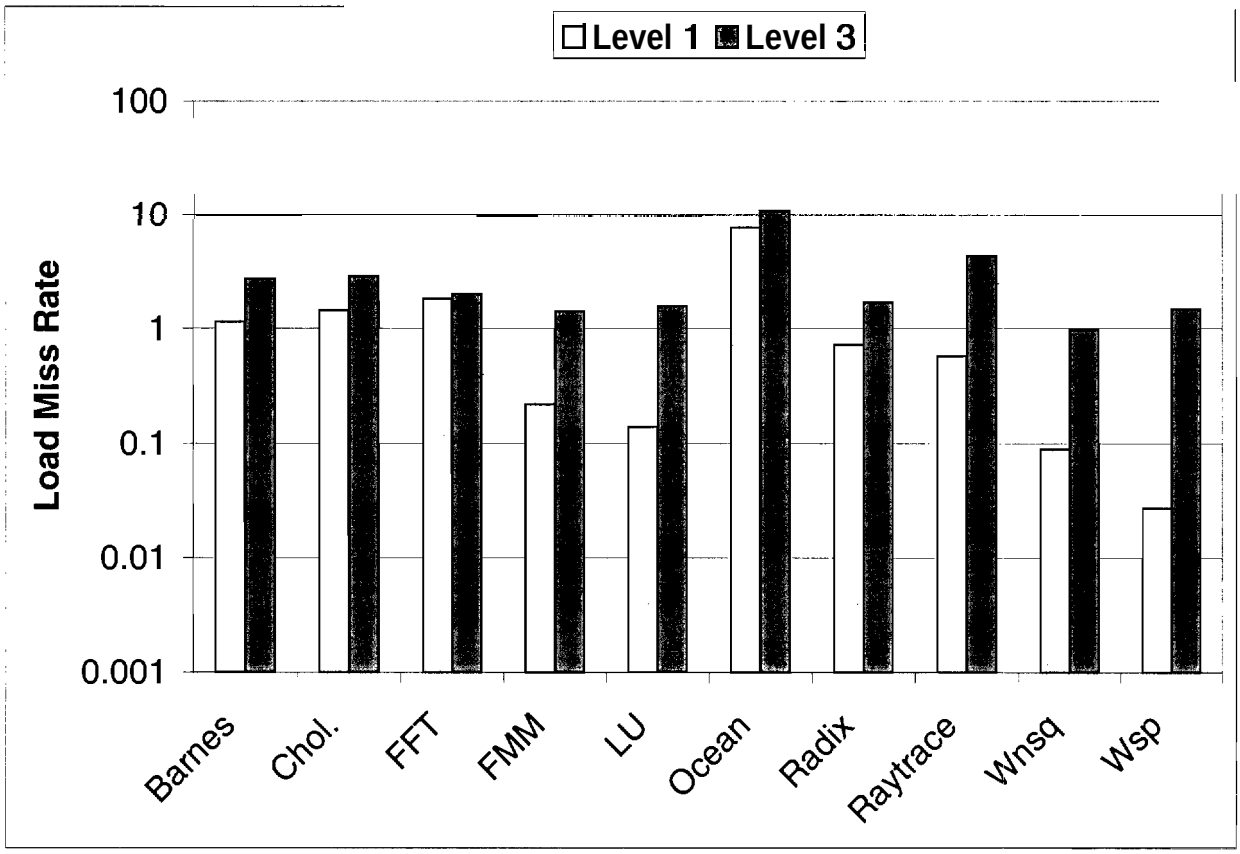

Figure 4: *Average load miss rate (in percentages, log_{10} scale) per processor for HDSM levels 1 and 3.*

The memory-intensive benchmark *Ocean* stands out with the best obtained speedup (5.7) for the HDSM model. The reason for this behavior lies in the ability of the architecture to expose high memory bandwidth. A more detailed look at the behavior of the memory subsystem of the HDSM reveals how it exploits the heterogeneous caches to provide high memory bandwidth.

Figures 4 and 5 show the average load miss rates obtained for the simulated benchmarks, and the average absolute number of misses per processor, respectively, under the virtual-processor assignment policy for levels 1 and 3. The miss rates increase from levels 1 to 3, since caches are smaller in the lowest level. However, the absolute number of misses is the smallest in the caches of level 3 for five of the benchmarks (Figure 5). Hence, larger caches are used more frequently (and with lower miss rates) than smaller caches, but the large number of independent small caches in the lowest level collectively provides high memory bandwidth.

Summarizing the results and findings of this subsection, the arithmetic mean of the best-assignment speedups for the four-node HDSM is 4.12. The performance analysis has showed that load-balancing mechanisms, either encoded in the application's algorithm or provided by virtual processors, and the large aggregate memory bandwidth provided by many independent caches are key to achieving good performance

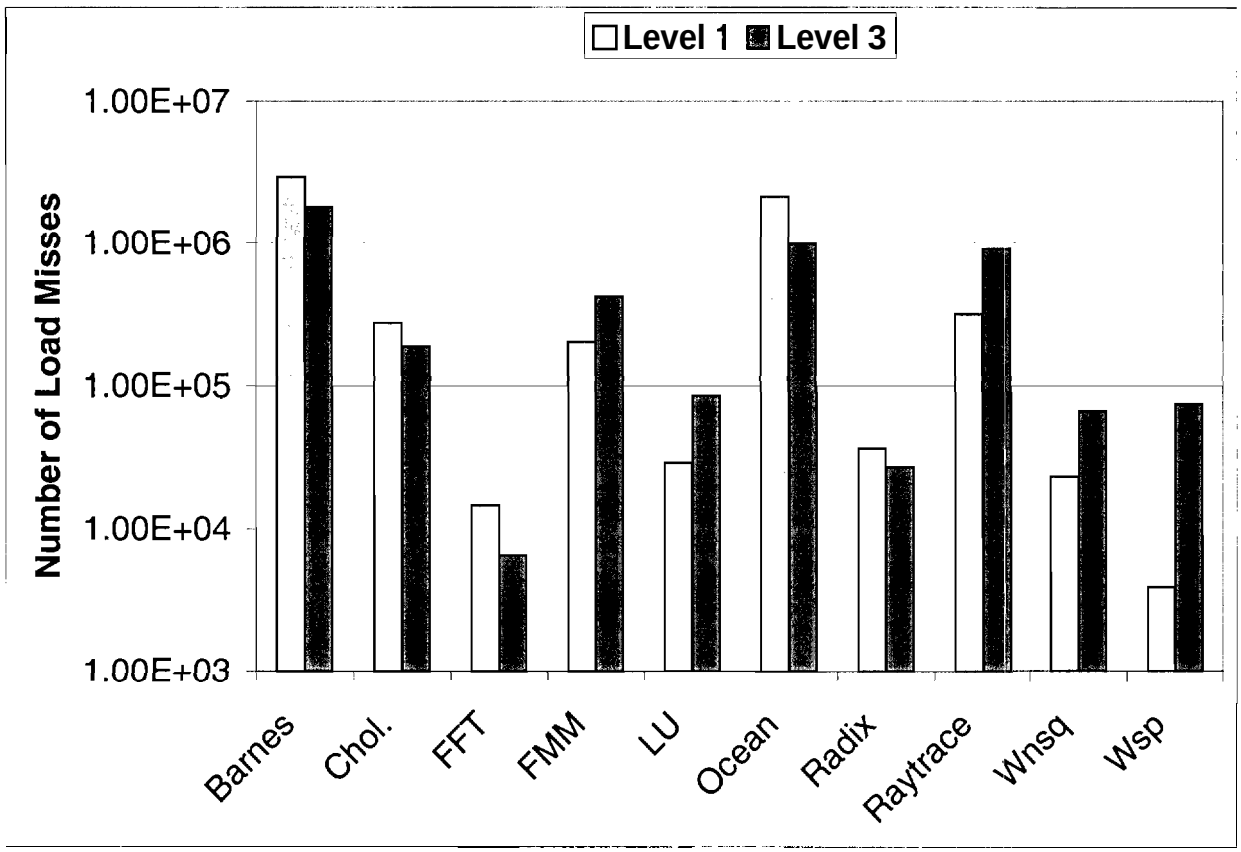

Figure 5: *Average absolute number of misses (log_{10} scale) per processor for HDSM levels 1 and* ***3.***

in most benchmarks. The next subsection analyzes HDSM performance with respect to a conventional four-node design.

4.2 Constant-Area Performance Analysis

In this subsection, an HDSM multiprocessor is compared to a conventional, homogeneous multiprocessor under a constant-area assumption. This comparison provides a quantitative analysis of the potential performance benefits of designing DSM machines with heterogeneous nodes.

Two different scenarios are considered in the analysis. The first scenario *(large cache)* compares a four-node HDSM as specified in Table 1 to a four-node homogeneous multiprocessor whose nodes are all fast level-1 uniprocessors. The second scenario *(small cache)* differs from the first scenario only with respect to cache sizes: all cache sizes are eight times smaller in both homogeneous and heterogeneous configurations. The purpose of this scenario is to study the performance of the HDSM model when caches may not be large enough to hold the secondary (and possibly the primary) working sets of SPLASH-2 benchmarks [35].

A homogeneous scenario consisting of four 8-processor chip-multiprocessors could also be conceived. For the SPLASH-2 parallel workloads, it is likely to outperform both studied homogeneous and heterogeneous

	Homogeneous Set			Heterogeneous Set		
	level i=1	level i=2	level i=3	level i=1	level i=2	level i=3
clock(i)	*clock(1)*	*clock(1)*	*clock(1)*	*clock(1)*	$2clock(1)$	$4clock(1)$
$size(i)	1MB	1MB	1MB	1MB	128KB	64KB
accesst(i)	$56clock(1)$	$56clock(1)$	$56clock(1)$	$56clock(1)$	$84clock(1)$	$112clock(1)$
latency(i,j)	$50clock(1)$	$50clock(1)$	$50clock(1)$	$50clock(1)$	$100clock(1)$	$100clock(1)$

Table 3: *Homogeneous and heterogeneous sets of values for the factors* clock(i), $size(i), accesst(i) *and* latency(i,j) *across levels ($j = 2, 3, 1$ for $i = 1, 2, 3$). Values are normalized with respect to the clock period of the fastest processor (clock(1)).*

configurations. However, such 32-processor machine would have a very poor sequential performance. Since all processors would be of the slowest type, this design can be as much as four times slower than either

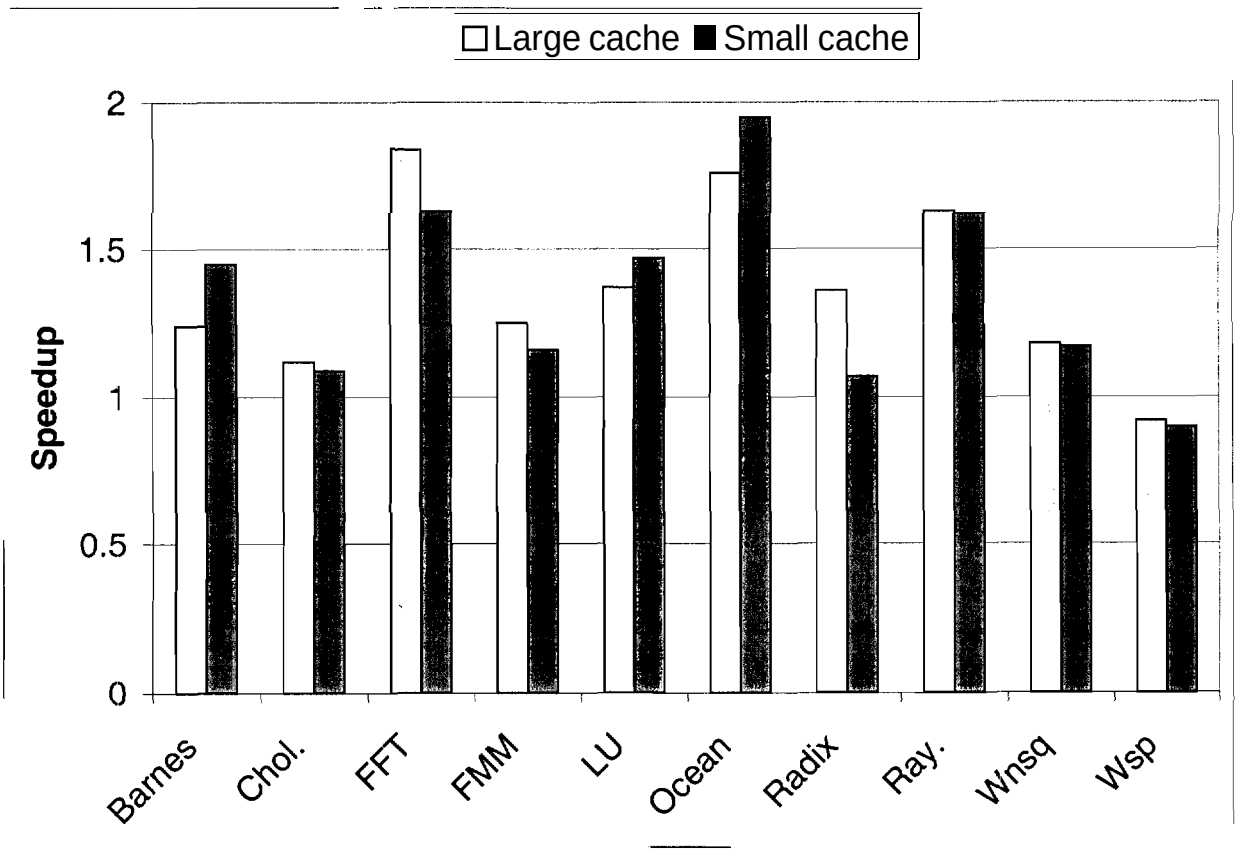

Figure 6: *HDSM speedup relative to homogeneous design. Average speedup is 1.37 for the large cache scenario, and 1.35 for the small cache scenario.*

of the DSMs studied. Although compiler [6] and run-time [15] parallelization techniques can improve the performance of sequential code on this design for some types of applications, the analysis presented in this subsection does not include such scenario.

Figure 6 shows the speedups, calculated as ratios of simulated homogeneous and heterogeneous DSM execution times, for the SPLASH-2 benchmarks. For all benchmarks but Water-spatial (11% slowdown), the HDSM model outperforms the homogeneous counterpart, by as much as 95% for Ocean. Speedups of 25% or more are also observed for Barnes, FFT, FMM, LU and Raytrace.

Good relative speedups for Ocean and *Raytrace* are expected from the analysis presented in the previous subsection; the HDSM model achieves uniprocessor speedups in excess of 5.0 for these benchmarks in a 4-node system. However, significant speedups relative to the homogeneous DSM are also observed for FFT (63% to 84%) even though the HDSM speedup with respect to a uni-processor (Figure 3) is less than 4.0. Similarly to the virtual-processor analysis presented in the previous subsection, this performance edge in FFT is due to the smaller inter-processor communication overhead experienced by the 2-node HDSM versus the 4-node homogeneous DSM.

The HDSM executes the communication-intensive transpose phase of the FFT algorithm 143% faster than the homogeneous multiprocessor, while executing the compute-intensive phases only 33% faster. In the homogeneous configuration, the transpose phase time involves communication among four nodes, while in the 2-node HDSM model, much of the inter-processor communication is intra-node, since each (wide) node has eight processors. As a result, the contribution of the transpose phase to the total execution time becomes smaller (41%) in the HDSM machine than in the homogeneous one (56%).

The HDSM relative performance for the small-cache scenario differs from the large-cache scenario by less than 8.0% for six of the studied benchmarks. For the remaining benchmarks, the relative small-cache HDSM speedup is smaller than the large-cache speedup for Radix and FFT, but larger for Barnes and Ocean. On average across all benchmarks, the two scenarios yield similar HDSM relative speedups: 37% (large cache) and 35% (small cache).

In summary, the HDSM configuration significantly outperforms a homogeneous counterpart for the multiprocessor workload considered in this paper, under both large and small cache scenarios. In the next subsection, a factorial design analysis determines the impact of heterogeneity on HDSM performance, and identifies which application characteristics lead to good performance.

4.3 Constant-Resources Performance Analysis

In this subsection, the impact of heterogeneity of the processor, memory and network subsystems on the performance of HDSMs is analyzed (Figure 1, solid arrow) via a factorial design methodology. Several design points are used in the simulation of each application. The values of *#nodes(i)* and *#procs(i)* are common to all simulated configurations: levels 1, 2 and 3 consist of 1, 1 and 2 nodes with a total of 1, 4 and 16 processors, respectively. Each design point is characterized by four triples, each specifying the value of *clock(i)*, *$size(i)*, *accesst(i)* and *latency(i,j)* for each of three levels i=1, 2 and 3. Each triple can take one of two values, a homogeneous one (i.e. the elements of each triple are all identical) and a heterogeneous one (i.e. the elements of each triple have different values).

The values assumed for each possible triple are shown in Table 3. Except for cache sizes, all numbers are in units of the base clock cycle, i.e. the clock cycle of the fastest processor in the hierarchy (the processor of level 1). The factorial design experiment considers 16 possible design points which correspond to all possible combinations of homogeneous and heterogeneous triples.

The heterogeneous triples shown in Table 3 differ from the configuration of Table 1 (studied in Subsections 4.1 and 4.2) only with respect to the memory and network latencies. In this subsection, the assumption of same memory and network technology is relaxed in order to study the sensitivity of HDSM performance to heterogeneity in memory and network latencies. They differ by at most a factor of two across heterogeneous levels.

To individually assess the performance impact of heterogeneity of each of the factors listed in Table 3,

a 2^k factorial design has been performed. Using the terminology of [16], such experimental design is used to determine the effect of each of k *factors* in variations of a *response variable,* where each *factor* has two alternatives or *levels.* In this paper, the $k = 4$ *factors* under study are *clock(i), $size(i), accesst(i)* and *latency(i,j).* The two *levels* that each factor can assume are the homogeneous and heterogeneous triples shown in Table **3.** The *response variable* used is simulated execution time.

The 2^4 factorial design experiment considers sixteen different combinations of *levels*; each combination corresponds to a distinctly heterogeneous configuration. As an example, one possible configuration may have homogeneous network latencies and cache sizes, while having heterogeneous processor and memory speeds. Each benchmark is simulated once for each distinct configuration; the obtained simulated execution times yield a 16-entry vector. This vector is then mathematically analyzed to determine the effect of each of the $k = 4$ *factors* in the variations of execution time. The analysis consists of calculating, for each factor, the inner-product between the execution-time vector and a sign vector (with entries from the set $(-1,+1)$) associated with the factor, and dividing the inner product by the total variation of the response variable y, given by $\sum_{i=1}^{16}(y_i - \overline{y})^2$.

The results obtained from the factorial design experiment are summarized in Figure 7. Variations in

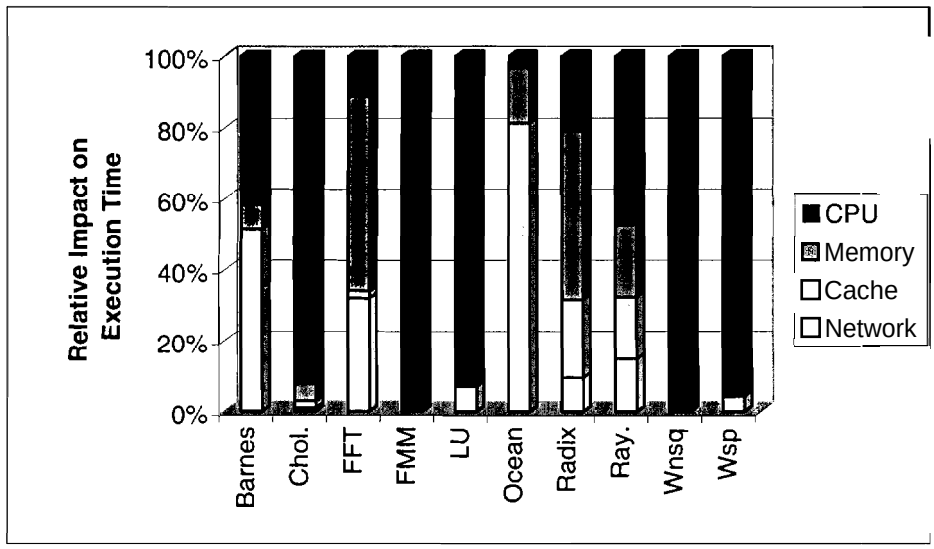

Figure 7: *Breakdown of the relative impact of heterogeneity of processor, memory and network subsystems on the execution time of the studied benchmarks.*

execution time are due to differences in speed and capacity of processors, memories and network: execution time is larger if a given factor is assigned a heterogeneous triple than a homogeneous triple (if all other factors remain unchanged).

In general, the minimum (maximum) execution time obtained in the design space under consideration corresponds to an architecture where all factors are assigned homogeneous (heterogeneous) tuples. For the SPLASH-2 benchmarks, the ratio between maximum and minimum execution times across the sixteen simulated configurations ranges from 1.33 (for FFT) to 5.54 (for Barnes).

The benchmarks are affected in different ways by variations in the processor, memory and network architectural factors. Figure 7 shows that, for the benchmarks FFT, Radzx and Raytrace, heterogeneity in the network and memory subsystems combined have the most significant impact in execution time: for *Ocean* and Barnes, heterogeneous cache sizes are responsible for 79.6% and 48.6% of the increase in execution time, respectively. The remaining five benchmarks are mainly affected by heterogeneity in processor speed.

In the average across all benchmarks, the variation in execution time is mostly due to heterogeneity in processor speed (59.3%), followed by heterogeneity in cache sizes (18.2%), memory access times (14.6%) and network latency (5.6%).

The results from the factorial design analysis indicate that heterogeneity in memory and network may be attractive from a cost-performance standpoint. To investigate this scenario, an experiment comparing fully heterogeneous DSMs against the configuration of Table 1 (processor/cache-heterogeneous) was performed. An average slowdown of 19.2% was observed across the ten SPLASH-2 benchmarks. If the cost reduction associated with slower memories and networks in the lower hierarchy levels outweighs this performance penalty, a cost-performance design goal would favor the all-heterogeneous design.

The results of this subsection, in conjunction with the results of Subsection 4.2, provide insights on the application characteristics that are best exploited by the HDSM design. An analysis of Figures 6 and 7 shows that the benchmarks with significant memory component in the factorial experiment have good speedups with respect to the homogeneous configuration. In particular, Ocean is the benchmark with the largest cache+memory term from the factorial analysis and achieves the best relative speedup among the SPLASH-2 programs. As discussed in the previous subsection, the HDSM model delivers good performance for memory-intensive programs due to its high aggregate memory bandwidth.

5 Related Work

The multiprocessor designs of this paper assume the future availability of very large chips capable of multiprocessing and/or very high performance for sequential codes. Proposals of the so-called billion-transistor architectures focus on the design of single-chip microprocessors that make use of very dense logic to exploit parallelism at different levels. Instruction-level parallel (ILP) processors proposed in [32, **26, 19,** 31] exploit parallelism in a single thread of execution. Chip-multiprocessors [14] exploit parallelism across multiple threads. Simultaneous multi-threaded (SMT) processors [8] target both single- and multi-thread parallelism in a single chip. Such designs will serve as building blocks for large multiprocessor configurations that use multiple multiprocessor chips. This paper considers system-level implications of the use of heterogeneous building blocks on the performance of future-generation DSMs.

Several studies have shown that heterogeneous multiprocessor systems may be more cost-effective than homogeneous multiprocessors [20, **21,** 3]. These studies have used analytical cost/performance models and/or simulation of message-passing workloads explicitly parallelized for a heterogeneous configuration. In contrast, this paper quantitatively analyzes the performance of unmodified shared-memory parallel programs in homogeneous and heterogeneous DSM multiprocessors of equal chip area..

6 Conclusions and Future Work

This paper shows that HDSM multiprocessors organized as processor-and-memory hierarchies constitute a high-performance approach to computer design. In addition, this paper identifies tradeoffs between performance and heterogeneity in the design of processors, caches, memories and network of such class of machines

Simulation experiments show that a 4-node HDSM achieves average speedups of 4.12 (with respect to a uniprocessor) and 1.36 (with respect to a 4-node homogeneous multiprocessor) for ten SPLASH-2 benchmarks. The heterogeneous organization is particularly effective for memory-intensive programs. While levels with small processor counts provide fast response for latency-sensitive tasks, levels with large number of processors provide large aggregate bandwidth for memory-intensive parallel tasks.

Three static thread assignment mechanisms that map homogeneous programs to heterogeneous organizations have been evaluated. The policy based on virtual processors provides good performance for memory- and CPU-intensive applications with low synchronization requirements. The single-thread assignment policy provides better performance for applications with high lock-based synchronization requirements. The single-level assignment policy results in the best performance for communication-intensive applications. These conclusions motivate ongoing work in dynamic thread assignment mechanisms that use run-time information and speculation to decide on the mapping of tasks to heterogeneous resources. Given the sensitivity of this class of applications to shared-memory protocol processing and latency overhead, future research on HDSMs will also focus on distributed shared-memory protocols and latency-tolerance mechanisms that account for node heterogeneity.

The impact of heterogeneity of the processor, memory and network subsystems on the performance of HDSMs is application-dependent. The studied applications are affected, on average, primarily by heterogeneity in processor speed (59.3%), followed by cache sizes (18.2%), memory latency (14.6%) and network latency (5.6%). The performance of HDSMs has thus low sensitivity to the use of slow memory technology in the highly parallel machine levels.

Motivated in part by the results of this paper, several research directions are being pursued on HDSM designs. One research effort investigates multithreading and synchronization mechanisms to efficiently support virtual-processor assignments. A second research effort focuses on the design of memory-dominated HDSMs for data-intensive applications. The third area of research is on software and hardware mechanisms for static and dynamic detection and exploitation of DoP locality. Spatial and temporal DoP locality have been empirically established in [3, 2, 12], but are not exploited by the HDSMs considered in this paper. Many opportunities exist for performance improvements, including compiler techniques, prefetching mechanisms and data speculation.

The many opportunities for further improvement of performance of HDSMs organized as processor-and-memory hierarchies reinforce the fact that there can be significant cost-performance advantages of this type of machines over conventional homogeneous designs.

7 Acknowledgments

This research was supported in part by NSF grants CCR-9970728 and EIA-9975275. Renato Figueiredo is supported by a CAPES fellowship. The authors thank the Wisconsin Wind Tunnel development team for making the source code of the simulator available, and thank Babak Falsafi for assistance with the installation and use of WWT-II.

References

[1] Peter Bannon. Alpha 21364: A Scalable Single-chip SMP. Microprocessor :Forum, Oct 1998.

[2] Ben-Miled, Z., Eigenmann, R., Fortes, J.A.B., and Taylor, V. Hierarchical Processors-and-Memory Architecture for High Performance Computing. Frontiers of Massively Parallel Computation Symp., Oct 1996.

[3] Ben-Miled, Z. and Fortes, J.A.B. A Heterogeneous Hierarchical Solution to Cost-efficient High Performance Computing. Par. and Dist. Processing Symp., Oct 1996.

[4] Ben-Miled, Z., Fortes, J.A.B., Eigenmann, R., and Taylor, V. A Simulation-based Cost-efficiency Study of Hierarchical Heterogeneous Machines for Compiler and Hand Parallelized Applications. 9th Int. Conf. on Par. and Dist. Computing and Systems, Oct 1997.

[5] Ben Miled, Z., Fortes, J.A.B., Eigenmann, R., and Taylor, V. On the Cost-efficiency of Hierarchical Heterogeneous Machines for Compiler- and Hand-Parallelized Applications. International Journal of Parallel and Distributed Systems and Networks, 1998.

[6] Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lawrence, T., Lee, J., Padua, D., Paek, Y., Pottenger, B., Rauchwerger, L., and Tu, P. Parallel Programming with Polaris. IEEE Computer, Dec 1996.

[7] Culler, D. and Singh, J. P. Parallel Computer Architecture. Morgan Kaufmann, 1998.

[8] Eggers, S. J., Emer, J. S., Levy, H. M., Lo, J. L., Stamm, R. L., and Tullsen, I). M. Simultaneous Multithreading: A Platform for Next-Generation Processors. IEEE Micro, pages 12–19, Sep. 1997.

[9] Dobberpuhl, D. et al. A 200MHz 64b Dual-Issue CMOS Microprocessor. In International .Solid Statt: Circuits Conference Digest of Technical Papers, pages 106–107, 1992.

[10] Gieseke, B. A. et al. A 600MHz Superscalar RISC Microprocessor with Out-of-Order Execution. In International Solid State Circuits Conference Digest of Technical Papers, pages 176–177, 1997.

[11] Waingold, E. et al. Baring It All to Software: Raw Machines. IEEE Computer, pages 86–93. Sep. 1995'.

[12] Figueiredo, R. J. O., Fortes, J. A. B., and Ben-Miled. Z. Spatial Data Locality With Respect to Degree of Parallelism in Processor-And-Memory Hierarchies. In Proceedings, 3rd International Meeiing on Vector and Parallel Processing, June 1998.

[13] Gustavson, D. B. The Scalable Coherent Interface and Related Standard Projects. IEEE Micro, Feb. 1992.

[14] Hanimond, L., Nayfeh, B. A., and Olukotun, K. A Single-Chip Multiprocessor. IEEE Computer, Sep. 1997.

[15] Hanimond, L., Willey, M., and Olukotun, K. Data Speculation Support for a Chip Multiprocessor. In Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII), pages 58–69, Oct 1998.

[16] Jain, R. The Art of Computer Systems Performance Analysis. John Wiley & Sons, 1991.

[17] Kozyrakis, C. E. et al. Scalable Processors in the Billion-Transistor Era: IRAM. IEEE Computer, Sep. 1997.

[18] Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W., Gupta, A., Hennessy, J., Horowitz, M., and Lam, M. The Stanford DASH Multiprocessor. IEEE Computer, Mar 1992.

[19] Lipasti, M. H. and Shen, J. P. Superspeculative Microarchitecture for Beyond A D 2000. IEEE Computer, Sep. 1997.

[20] Menasce, D. and Almeida, V. Cost-performance Analysis of Heterogeneity in Supercomputer Architectures. Proc. Supercomputing Conf., Nov 1990.

[21] Moncrieff, D. et al. Heterogeneous Computing Machines and Amdahl's Law. Parallel Computing, 22(3), 1996.

[22] Mukherjee, S. S., Reinhardt, S. K., Falsafi, B., Litzkow, M., Huss-Lederman, S., Hill, M. D., Larus, J. R., and Wood, D. A. Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator. In Workshop on Perf. Analysis and Its Impact on Design (PAID), June 1997.

[23] Nayfeh, B. A., Hammond, L., and Olukotun, K. Evaluation of Design Alternatives for a Multiprocessor Microprocessor. In Proc. *23rd* International Symposium on Computer Archztecture, pages 67–77, 1996.

[24] Pai, V. S., Ranganathan, P., and Adve, S. V. The Impact of Instruction-Level Parallelism on Multiprocessor Performance and Simulation Methodology. In Proc. 3^{rd} International Symposium on High-Performance Computer Architecture, Feb 1997.

[25] Palacharla, S., Jouppi, N. P., and Smith, J. E. Complexity-Effective Superscalar Processors. In Proc. 24th International *Symposium* on Computer Architecture, pages 206–218, 1997.

[26] Patt, Y. N., Patel, S. J., Evers, M., Friendly, D. H., and Stark, J. One Billion Trnasistors, One Uniprocessor, One Chip. IEEE Computer, pages 51–57, Sep. 1997.

[27] Reinhardt, S. K., Hill, M. D., Larus, J. R.. Lebeck, A. R., Lewis, J. C., and. Wood, D. A. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In ACM SIGMETRICS, May 1993.

[28] Reinhardt, S.K., Larus, J.R., and Wood, D.A. Tempest and Typhoon: User-Level Shared-Memory. In Proceedings of the *21st* Annual Internatzonal Symposium on Computer Architecture. pages 325–337, Apr 1994.

[29] Saulsbury, A., Wilkinson, T., Carter, J., and Landin, A. An Argument for Simple COMA. In Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture, pages 276–285, Jan 1995.

[30] Semiconductor Industry Association. The National Technology Roadmap for Semiconductors. San Jose, CA, 1997.

[31] Smith, J. E. and Vajapeyam, S. Trace Processors: Moving to Fourth-Generation Microarchitectures. IEEE Computer, pages 75–78, Sep. 1997.

[32] Sohi, G., Breach, S., and Vijaykumar, T. N. Multiscalar Processors. In *Proc. 23nd* International Symposium on Computer Architecture, pages 414–425, 1995.

[33] Standard Performance Evaluation Corporation. http://www.spec.org.

[34] Stenstrom, P. A Survey of Cache Coherence Schemes for Multiprocessors. *IEEE* Computer, June 1990.

[35] Woo, S. C. et al. The SPLASH-2 Programs: Characterization and Methodological Consideration;~In Proceedings of the 22nd Annual Int. Symp. on Computer Architecture, July 1995.

Appendix A - Simulator Validation

Heterogeneity in processor performance is modeled in this paper by scaling the clock speed. This model has enabled the simulation of a wide range of applications and machine configurations. In this appendix, the mechanisms used to validate the modifications introduced in the Wisconsin Wind Tunnel-I1 simulator to support heterogeneous, clock-scaled processors are described. In addition, this appendix presents simulation results obtained from experiments with an ILP-enabled multiprocessor simulator (RSIM) that provide a quantitative estimate of the magnitude of potential errors in execution time estimates associated with the modeling of ILP processors via clock scaling.

Previous studies [24] have shown that the clock speed scaling method introduces errors in execution time estimates when compared to detailed ILP simulations. However, due to the small number of level-1 and level-2 ILP processors simulated in this paper, the potential errors in execution time estimates are smaller than previously reported for 8-processor systems. This claim is supported by data obtained for two 4-nocle multiprocessor configurations with 1MB level-2 caches (based on the specifications of Table 1) using a detailed ILP multiprocessor simulator, RSIM. Table 4 summarizes the results from this experiment for three benchmarks common to this paper and [24].

The average error in execution time estimates introduced by modeling 4-issue out,-of-order processors by in-order processors with clock rates scaled by four is 24% for a 4-node system for the benchmarks *LU*, *Radix* and FFT, while reported average errors for these benchmarks on an 8-node system are 74% [24]. In addition, since both homogeneous and heterogeneous configurations use clock-scaled processors (4 and 5 processors, respectively), potential errors should have little effect in relative comparisons.

Even in a worst-case scenario, when potential errors should only increase the execution time of the heterogeneous design, the data shown in Table 4 does not modify the conclusion that, in a relative comparison, the heterogeneous configuration has better performance for the programs *FFT*, *LU* and *Radix*.

The modifications of the WWT-II simulator to allow heterogeneous functional units consist mainly of changes in object constructor calls for each processor, memory and network modules in each node; no additional modules have been added to the original program. Even though the modifications introduced in the program are marginal, it is important to validate the heterogeneous model to obtain confidence in the results obtained. Since no heterogeneous cache-coherent shared-memory machines have been built to date, nor have shared-memory heterogeneous multiprocessor simulators been developed, the strategy used to validate the modifications was to perform simulations in both the original and the modified simulators for a set of micro-kernels specifically designed for this study. Such validation methodology is similar to the one used for the original Wind Tunnel [27], but specific kernels have been developed for the validation because heterogeneous shared-memory parallel benchmarks were not available.

Three micro-kernels have been developed to perform the validation of heterogeneity of number of pro-

	Speedups				Errors
Issue	1	4	1	4	
Order	in	out	in	out	4-node
Clock	1x	1x	4x	1x	
Nodes	1	1	4	4	
FFT	1.0	3.50	10.44	12.87	+23.3%
LU	1.0	4.42	12.26	15.77	+28.7%
Radix	1.0	2.72	12.09	10.06	-20.2%

Table 4: *,Speedups (with respect to base in-order uniprocessor) and errors (multiprocessor execution times). Speedups are shown for ,$-issue out-of-order uniprocessor; single-issue 4-node multiprocessor with clock scaling; and out-of-order ,$-issue, 4-node multiprocessor.*

	Orig.	Modified	
$NP_{1,2,3}$	1,1,1	1,4,16	
$CLK_{1,2,3}$	1,1,1	1,1,1	1,2,4
Level 1	117.7	117.7	117.7
Level 2	117.7	29.5	58.9
Level 3	117.7	7.5	29.7

Table 5: *Per-level execution times (in* 10^6 *base clock cycles) for* Kernel *1 with 1MB caches for three simulations: original WWT-II with 3 uniprocessor nodes, modified WWT-II with 1,4,16 processors and clock speeds of 1,1,1 and 1,2,4*

cessors per node, size of caches, speed of processors, and memory and network latencies:

Kernel 1: The first kernel is designed to be highly CPU-intensive and efficiently parallelizable. This kernel surns the contents of two 2048-integer arrays into a scalar (by striding linearly through the arrays with a stride of one integer) 8192 times on each of the three nodes of a heterogeneous machine. Using private variables to store partial sums for each processor, this kernel can be efficiently parallelized. Since the sharing pattern of the shared arrays is purely read-only, there are no coherence-induced cache invalidations. In addition, if the working set fits in the processor caches, there are no conflict and capacity misses. This kernel is used to validate the modifications that include heterogeneity in both the number of processors per node and speed of processors.

Kernel 2: The second kernel is similar to **Kernel 1**, except that cache sizes that may be smaller than the working data set size (16 KBytes) are used and the number of iterations is reduced to 256. By using a cache size smaller than the size of the shared arrays, any given block used in previous iterations is evicted from the cache by the time a new iteration issues an access to that block; hence, in this configuration most memory accesses do not hit in the processor caches (except for hits due to spatial locality of a single cache block) and need to go to main memory. This scenario is used to validate the modifications that include heterogeneity in cache sizes and memory access times.

Kernel 3: The third kernel is designed to be communication-intensive. In this 'kernel, a single processor in each node tests the value of a shared variable *token*, enters a critical section when the value of *token* matches the node's ID, and increments *token* by one (modulo 3) to allow another node to enter its critical section. The token is acquired and released by each processor 8192 times. Execution time for this kernel is highly dependent on the latency of both invalidation and read request coherence-induced messages that are sent during the execution of the critical section and during the test of the value of the token, respectively. This kernel is thus used to validate the modifications that include heterogeneity in network latency.

Table 5 shows the per-level execution times for three simulations of *Kernel 1.* Three 3-level machine configurations are simulated: one homogeneous (one processor per node, each processor of same speed) and two heterogeneous (1,4,16 processors in levels 1,2,3 respectively). For the homogeneous configuration, the execution times on levels 1, 2 and 3 are all equal to 117.7 million cycles. For the heterogeneous configuration with processors of identical speeds, the expected speedups for levels 2 and 3 are 4.0 and 16.0, respectively. The simulation results show that the execution times on levels 2 and 3 are 3.99 and 15.69 times smaller than the execution time on level 1, respectively.

For the remaining heterogeneous configuration, the expected level-2 and level-3 speedups are 2.0 and 4.0, respectively, since slower processors are used in levels 2 and 3. The simulated results for the level-2 and level-3 speedups (shown in Table 5) are 2.0 and 3.94.

The simulation results for *Kernel* 2 with heterogeneous cache sizes are summarized in Table 6. The simulatecl architectures for this experiment have a single processor (of the same speed) in each of the three levels. Cache sizes of 16KBytes are sufficient to hold the kernel's working set. The per-level execution

	Orig.	Modified		Orig.
$CSZ_{1,2,3}$	16,16,16	8,16,16	8,8,16	8,8,8
Level 1	3.9	77.8	77.8	77.8
Level 2	3.8	3.8	77.8	77.7
Level 3	3.9	3.9	3.9	77.8

Table 6: *Execution times (in* 10^6 *base clock cycles) for* Kernel 2 *with homogeneous and heterogeneous cache sizes (in KB). The memory access time is set to 56 cycles.*

	Original			Modif.
$MAT_{1,2,3}$	28,28,28	56,56,56	112,112,112	28,56,112
Level 1	48.4	77.8	136.5	48.4
Level 2	48.3	77.7	136.5	77.7
Level 3	48.4	77.8	136.5	136.4

Table 7: *Execution times (zn* 10^6 *base clock cycles) for* Kernel 2 *with homogeneous and heterogeneous memory access times (in base clock cycles). The cache size is set to a value smaller than the working data sets (8 KBytes)*

times when all levels have 16KBytes caches are in the range 3.8-3.9 million cycles. Cache sizes of 8KBytes are smaller than the working set; the per-level execution times grow to 77.7-77.8 million cycles when such caches are used in all levels. When caches of different sizes are used across levels and the modified simulator is used, the per-level execution times of 8KByte-cache levels and of 16KByte-cache levels match the simulation results of the original simulator with less than 0.1% error.

The simulation results for *Kernel 2* and heterogeneous memory access times are summarized in Table 7. The simulated architectures have a single processor per node (of same speed) and small caches (8KBytes). The homogeneous configurations with access times of 28, 56 and 112 clock cycles in all levels have per-level execution times in the ranges 48.3-48.4, 77.7-77.8 and 136.5-136.5, respectively. The heterogeneous configuration with memory access times of 28, 56 and 112 clock cycles in levels 1, 2 and 3 has per-level execution times that match the values obtained with the original simulator with less than 0.01% error.

The simulation results for *Kernel 3* (heterogeneity in network latency) are summarized in Table 8. The simulated architectures have three levels with a single processor each (of same speed), and identical caches and memories. The home of the token shared by the three processors is the level-2 node; according to the Stache protocol used in the simulations, all coherence communications are done through the home node. Hence, in this kernel, all messages regarding the shared token are exchanged either between levels 1 and 2 or between levels 2 and 3.

When all inter-level network latencies are 50 cycles, both original and modified simulators yield the same result for the total execution time of *Kernel 3* (12.1 million cycles). When the inter-level network latency between levels 1,2 or (2,3) is increased, the results for the modified simulator show that total execution time increases to 15.4 (15.3) million cycles. When the inter-level network latency between levels (1,3) is increased, the total execution time does not change. These results are consistent to the expected behavior from the discussion on home node messaging of the previous paragraph. Finally, when all inter-level network latencies are 50 cycles, both original and modified simulators yield a total execution time of 17.8 million cycles.

The results shown in Tables 5 through 8 indicate that the modifications introduced in the original Wisconsin Wind Tunnel-I1 code to express heterogeneity in several architectural parameters are consistent

$NL(1,2)$	$NL(1,3)$	$NL(2,3)$	*Exec* **time**
50	*50*	*50*	*12.1*
50	*100*	*50*	*12.1*
100	*50*	*50*	*15.4*
50	*50*	*100*	*15.3*
100	*100*	*100*	*17.8*

Table 8: *Total execution times (in* 10^6 *base clock cycles) for* Kernel *3 with homogeneous and heterogeneous network latencies. Values obtained from both the original and the modified simulators ore shown in italics.*

with the expected behavior of the heterogeneous cluster model studied in this paper.