A Simulation-based Cost-efficiency Study of Hierarchical Heterogeneous Machines for Compiler- and Hand-Parallelized Applications*

ZINA BEN MILED, JOSÉ A. B. FORTES, RUDOLF EIGENMANN
School of ECE, Purdue University, W. Lafayette, In 47907-1285
VALERIE TAYLOR
Department of ECE, Northwestern University, Evanston, IL 60208-3118

Abstract

This study is a simulation-based analysis of the performance of compiler- and hand-parallelized versions of the CMU benchmarks on heterogeneous machines organized as a hierarchy of processor-and-memory (HPAM). The characteristics of these benchmarks are first established and then each benchmark is mapped onto different multilevel heterogeneous organizations. The performance of these mappings is compared to that achieved by executing the benchmarks on the *optimal* one-level homogeneous machine organization for a given benchmark and machine budget. This study establishes that heterogeneous machine can have higher cost-efficiency than the optimal homogeneous machine. Experimental analysis shows that the heterogeneous multilevel machine organization can benefit from hardware and software support for reconfiguring three-level and two-level machines into two-level and one-level organizations. Furthermore, these experiments indicate that the performance of a given application, when executed on a multilevel heterogeneous organization, depends not only on the parallelism distribution but also on the ratio of total communication time to total computation time.

Keywords: Heterogeneous, Cost-efficiency, Reconfiguration, Simulation, Parallel and Performance.

1 Introduction

The potential for heterogeneous architectures to provide cost-efficient high performance computing has been shown analytically in several previous studies ([1], [2] and references therein). This paper reports the results of a simulation-based study of heterogeneous machines (HPAM) organized as a hierarchy of processor-and-memory[1]. An HPAM consists of several levels of processor-and-memory (PAM) systems. Relative to the first level of the hierarchy, each additional level has an increasing number of slower processors. This study addresses three important questions: (1) Given a fixed budget can an HPAM organization compete with the optimal one-level homogeneous machine for a given benchmark? (2) What characteristics of the benchmarks dictate how well a given application maps onto an HPAM organization? How critical is it to provide hardware and software support for reconfiguring a physical HPAM machine with k of levels into a virtual machine with $j < k$ levels (i.e., making one level behave as an extension of its successor level)?

*This research was supported in part by NSF grants ASC-9612133, ASC-9612023 and CDA-9617372.

DoP	fft1	radar	airshed
1	75.1%	96.2%	5.4%
25			22.2%
40		1.4%	
150			3.3%
400			61.2%
1000			7.9%
8192		2.4%	
65536	24.9%		

Table 1: PARALLELISM DISTRIBUTION FOR THE COMPILER-PARALLELIZED CMU BENCHMARKS.

The optimal homogeneous machine varies from one benchmark to another. Furthermore, for a given benchmark, it also varies with the budget allocated to the machine. This variation is in both the number and the speed of the processors.

Section 2 of this paper describes general characteristics of the CMU benchmarks[3] used to carry this study. Due to space limitation, the results corresponding to three out of the five CMU benchmarks are included in this paper. Additional findings about the selected benchmarks as well as the remaining two benchmarks from the CMU suite can be found in [4]. The simulation environment and the notation used are discussed in Section 3. The results obtained by mapping the CMU benchmarks onto different HPAM organizations are included in Section 4. Conclusions and directions for future work are provided in Section 5.

2 Benchmark Characteristics

Parallelism in each benchmark is detected by two different approaches: compiler (Polaris[5]) and "hand" parallelization. The first version of these benchmarks is representative of HPAM codes that might result from a parallelizing compilation. The benchmarks that result from the second approach allow the study of performance of HPAM architectures when used by expert programmers.

2.1 Compiler Parallelization

Let DoP (degree of parallelism), at a given point in time in the execution of a benchmark, denote the maximum number of assembly level code instructions that can be executed concurrently as expressed by the explicit parallelism in the benchmark. Data parallelism in each benchmark was detected using Polaris.

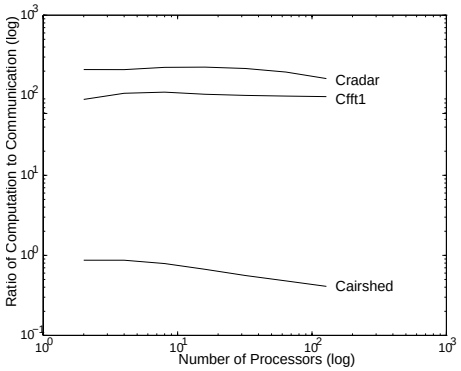

Figure 1: RATIO OF THE TOTAL TIME SPENT IN COMPUTATION TO THE TOTAL TIME SPENT IN COMMUNICATION FOR THE COMPILER PARALLELIZED BENCHMARKS.

HPAM_Sim[6] was then used to evaluate the percentage of time (measured in cycles) during which a benchmark executed with each distinct DoP. Table 1 shows the DoP time percentages for the CMU benchmarks (for clarity, percentages of less than 1% of the total number of cycles are omitted). The information included in Table 1 does not reflect the communication behavior of the benchmarks. In order to complement Table 1 and to gain insight into the communication behavior of the benchmarks, the ratio of the total time spent in computation to the total time spent in communication was estimated for each benchmark (Figure 1). This ratio (denoted CCratio) is obtained by tracing the communication and computation tasks for one processor (i.e., processor zero) of a one-level organization. The total time spent in communication not only includes message passing operations but also any copy or setup operations that are only needed for communication. Although this ratio is computed for processor zero only, in most of the benchmarks it is a relatively accurate estimate of the computation and communication times distribution of the benchmark. This ratio depends on the underlying communication model which is discussed in Section 3.

Table 1 and Figure 1 show that the CMU benchmarks have different parallelism and communication distributions. The benchmark 1D Fast Fourier Transform (fft1) benchmark has two predominant $DoPs$: a large sequential portion (i.e., $DoP = 1$) and the remaining portion of the application, which is highly parallel (i.e., $DoP = 65536$). Expectedly, fft1 should benefit from a two-level organization with one fast processor in the first level and many processors in the second level.

The Narrow Band Tracking Radar (radar) benchmark is mostly sequential. Given the three distinct $DoPs$ in radar, one would expect radar to map well onto a three-level HPAM. However, the sequential portion of radar accounts for more than 90% of the total number of cycles. Thus, the faster the first level, the more performance gain the heterogeneous machine will have over the homogeneous machine.

As shown in Table 1, airshed has five distinct $DoPs$ including a small sequential portion. However, the parallel portions of airshed have a high communication overhead (Figure 1). These portions mainly consist of reduction operations. Thus, in order to observe any

DoP	fft1	radar	airshed
1	0.98%	68.11%	0.14%
40		28.72%	
512		3.17%	
700			99.86%
65536	99.02%		

Table 2: PARALLELISM DISTRIBUTION FOR THE HAND-PARALLELIZED CMU BENCHMARKS.

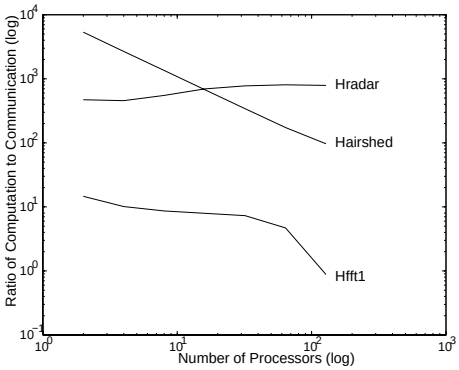

Figure 2: RATIO OF THE TOTAL TIME SPENT IN COMPUTATION TO THE TOTAL TIME SPENT IN COMMUNICATION FOR THE HAND-PARALLELIZED BENCHMARKS.

performance gain by using more than two levels or a two-level organization with large number of processors, the machine would need hardware or software support for fast reductions.

2.2 Hand Parallelization

This section includes a brief summary of the hand-parallelized version of the CMU benchmarks. It might be possible for programmers who are more familiar (than the authors of this paper) with the algorithms underlying these benchmarks to extract more parallelism. Hand parallelization in this study relied on the detailed analysis (by other authors) included in [3]. Hand parallelization (Table 2) exposes more parallelism with $DoP = 65536$ in fft1. There are two types of tasks with the above degree of parallelism: communication intensive tasks and non-communication intensive tasks. For the hand-parallelized version of fft1, of the 99.02% execution time in Table 2, 63.44% and 35.28% correspond to the first and second types of tasks, respectively. Due to the communication intensive nature of the first type of tasks, it might not be practical to fully exploit parallelism up to $DoP = 65536$. Thus, a more accurate estimate for the DoP of these tasks is 256. Compared to the compiler-parallelized version of this benchmark, the CCratio is lower for the hand-parallelized version (Figure 2). One would expect this benchmark to best map onto a one-level homogeneous machine.

Although radar is still mostly sequential, after hand parallelization, more parallelism was exposed for $DoP = 40$ and $DoP = 512$. The compiler version of radar had a portion of the code corresponding to $DoP = 8192$. For practical reasons, and because it corresponds to a small percent of the execution time, this portion was lumped with the portion of the program that corresponds to $DoP = 512$ in the hand-parallelized version of the benchmark.

Polaris exposed mostly parallelism due to reduction

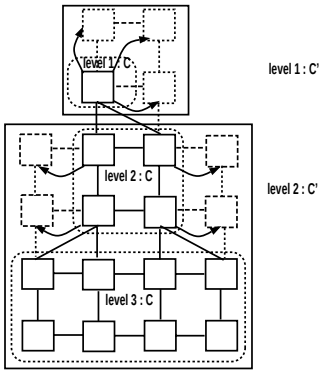

Figure 3: CONFIGURATION C' = $\{L_{2,4}, L_{1,16}\}$ AND $C = \{L_{8,1}, L_{2,4}, L_{1,8}\}$ OF $H = [L_{8,1}, L_{2,4}, L_{1,8}]$.

operations in airshed. In the hand-parallelized version, parallelism across the input grid points was detected. This parallelism is hard to detect with a parallelizing compiler because it requires extensive inter-procedural analysis. As shown in Figure 2 and Table 2, airshed is mostly parallel and has a high CCratio. Thus, it is expected to be best mapped onto a homogeneous one-level organization. The notation Hbenchmark and Cbenchmark (e.g. Hfft1 and Cfft1) is here on used to refer to the hand-parallelized and compiler-parallelized versions of each benchmark, respectively.

3 Simulation Environment

For a given processor, the average clock cycle per instruction (CPI) is the ratio of the CPU clock cycles for a program to the instruction count[7]. In this study, the *non-normalized* speed of a processor is defined as the ratio of the CPI to the CPU clock cycle time. The *normalized speed* of a processor refers to the non-normalized speed of the processor divided by the non-normalized speed of the reference processor. The notation α and $\overline{\alpha}$ is used to distinguish between normalized and non-normalized processor speeds, respectively.

Let $L_{\alpha,\beta}$ represent an HPAM level that consists of β processors with speed α. The notation L_{α,β^+} is used to represent a family of levels with a number of processors greater than or equal to β and processor speed α (e.g. $L_{\alpha,2^+}$ represents $L_{\alpha,2}, L_{\alpha,4}, L_{\alpha,8}$ ...). Similarly, $L_{\alpha^+,\beta}$ is used to denote the family of levels with β processors and processor speed greater than or equal to α. Let $H = [L_{\alpha_1,\beta_1}, L_{\alpha_2,\beta_2}, \cdots, L_{\alpha_n,\beta_n}]$ represent an n-level HPAM physical machine. Each HPAM machine can have several virtual HPAM configurations. Let $C = \{L_{\gamma_1,\delta_1}, L_{\gamma_2,\delta_2}, ..., L_{\gamma_m,\delta_m}\}$ represent an m-level configuration, where $m \leq n$. The *native* configuration of the above HPAM machine (H) is the configuration $C = \{L_{\alpha_1,\beta_1}, L_{\alpha_2,\beta_2}, ..., L_{\alpha_n,\beta_n}\}$ which matches the physical configuration. Figure 3 shows the native configuration ($C = \{L_{8,1}, L_{2,4}, L_{1,8}\}$) as well as the configuration $C' = \{L_{2,4}, L_{1,16}\}$ of the HPAM machine $H = [L_{8,1}, L_{2,4}, L_{1,8}]$. In C', the second level of H behaves as an extension of the third level and the first level emulates a level with a larger number of slower processors (than the first level of C).

3.1 Cost Model

For the model proposed in [8], the cost of a given level $L_{\alpha,\beta}$ is estimated as $a\alpha^b\beta$, where a and b are constants. In this study, the reference processor ($L_{1,1}$) has normalized speed *one* and cost a. The *number of equivalent reference processors* (*neq*) is used to compare different HPAM machines. A fixed value of *neq* corresponds to a fixed budget. The *neq* of $L_{\alpha,\beta}$ is the cost of $L_{\alpha,\beta}$ relative to the cost of the reference processor (i.e., $\frac{a\alpha^b\beta}{a} = \alpha^b\beta$). Similarly, for an entire HPAM machine, the *neq* is:

$$neq\,(H = [L_{\alpha_1,\beta_1}, \cdots, L_{\alpha_n,\beta_n}]) = \sum_{i=1}^{n} {\alpha_i}^b \beta_i \qquad (1)$$

In [8] and [9] a value of *two* was used for b. However, in [8] it was indicated that $b = 2$ is a conservative estimate for the cost of fast processors and that the cost function becomes exponential for high-end processors. The value $b = 2$ is used throughout this study and the effect of varying b is discussed in Section 4.6. For a given value of b and a machine H, *neq* represents the size of the one-level homogeneous machine that can be built using processors with normalized speed one and the same budget allocated for H.

3.2 Communication Model

Network Topology: In concept HPAM can have any interconnect. However, in this study each level in HPAM uses a mesh for intra-level interconnect. Let processors be represented by $(row, column)_{level}$ tuples such that the topmost row and the leftmost column of a given level are labeled zero. Also, let level l consist of an $r_l \times c_l$ processor mesh (r_l and c_l are powers of two). Furthermore, let levels be numbered in increasing order starting from the top level. A given processor $(0, a)_l$ is connected to processor $(r_{l-1} - 1, b)_{l-1}$. The relationship between a and b is given by:

$$b = \lfloor a/f_l \rfloor \text{ where } f_l = c_l/c_{l-1}, \text{ and } c_l \leq c_{l-1} \qquad (2)$$

The above inter-level network was selected because it allows upper level to behave as an extension to lower levels while using existing connectivity efficiently and without the need for additional connectivity.

Communication Latency: For a given machine, the communication latency (t_{comm}) is the sum of three major components[10]: t_{send}, t_{net} and t_{recv}. For a message of size S Bytes, each of these components can be expressed as follows:

$$\begin{aligned} &t_{send} = C_{send} + L\tfrac{S}{W}, \quad t_{net} = L_1(\tfrac{S}{W_1} + n), \\ &t_{recv} = C_{recv} + L\tfrac{S}{W}. \end{aligned} \qquad (3)$$

In the expression of t_{send} and t_{recv}, C_{send} and C_{recv} are constants. In these same expressions, the terms that vary with the size of the message are mainly due to any copy operation required during a send or a receive. In the expression of t_{net}, L_1 is the *per-hop-delay*, n denotes the number of *hops* (between the source and the destination of the message) and W_1 is the width of the point-to-point link between a processor and its nearest neighbor. On most commercial multiprocessor systems, the communication delay is dominated by the software startup latency ($t_{send} + t_{recv}$)[11]. For example, in [10], the different parameters in Equation 3 corresponding to the IBM-SP1 were estimated at : $C_{send} = 20\mu s$, $L/W = 0.02\mu s/Byte$, $L_1/W_1 = 0.03\mu s/Byte$ and $C_{recv} = 35\mu s$. The software startup latency is usually estimated as half the time it takes for one processor

to send and receive a small message to/from a nearest neighbor ("echo" test).

The variation in software startup latencies can be as high as an order of magnitude for systems using traditional message passing protocols (e.g. Cray-T3D+PVM: $21\mu s$ and SP1+MPL: $270\mu s$ [13]). The focus of several studies such as [14] and [15] is to reduce software startup latency. The common idea behind these approaches is to use the advantages of shared memory communication paradigms in message passing. This is accomplished in [14] by using active messages and in [15] by using virtual mapped memory communication. Both models try to maximize the overlap between computation and communication while performing message passing in user space. In [15], it was shown that one-way transfer times for the "echo" test that are close to the physical hardware latency can be achieved for a message passing communication paradigm. Given the variation in software startup latencies due to different implementation and that new software and hardware techniques will continue to improve the software startup latency, this study focuses on comparing HPAM to homogeneous machines in the ideal case where $C_{Send} = C_{recv} = 0$. An ongoing study addresses the effect on the performance of HPAM machines due to varying startup software latencies.

For all the experiments included in this paper, we make the simplifying assumption that $L_1 = L$ and $W_1 = W$. This assumption was motivated by the fact that for the IBM-SP1 (described above) as well as for other machines[11], $L_1/W_1 \approx L/W$. Furthermore, W is set to 8 Bytes because technology advances will allow 4 to 8 Bytes wide channels in the near future [12].

The *per-hop-delay* (L) is set to $12/\overline{\alpha}$. This parameter varies with the processor speed in order to reflect the fact that fast networks are used with fast processors and slow networks are used with slow processors. Using a network that is balanced with respect to processor speed allows the cumulative cost of the individual processors in the machine to remain indicative of the overall cost of the machine. For a processor with $\overline{\alpha} =$ 100-Mhz, L is equal to 120ns (which does not represent an aggressive network) and for a processor with $\overline{\alpha} = 1$Ghz, L is equal to 12ns (which represents a more aggressive network). For example, the reported *per-hop-delay* for the Paragon is 40 ns[11].

Contention: HPAM_Sim allows the simulation of the target machine network with or without contention. Simulation of network contention is more time consuming. For Hfft1, there is more than 20% difference between the simulation estimated execution times obtained with and without contention. Thus, for this benchmark the contention mode of HPAM_Sim was used. For Cfft1, Cradar, Hradar, Cairshed and Hairshed, the percent difference between the simulation estimated execution times using the contention mode and the non-contention mode is less than 1%. Thus, for these benchmarks the non-contention mode was used.

4 Simulation Results and Analysis

The two versions of each of the selected CMU benchmarks were mapped onto different HPAM configurations using HPAM_Sim. There are two different factors

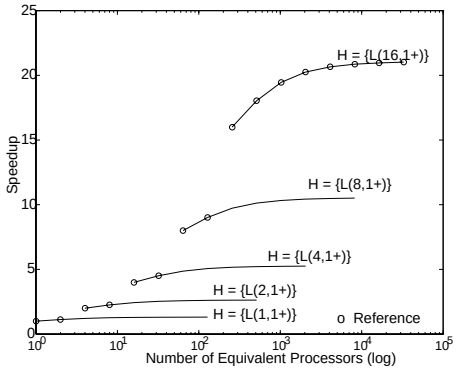

Figure 4: SPEEDUP OF Cfft1 ON $H = [L_{1^+,1^+}]$.

that can determine the assignment of a given task to a given level : the DoP and the communication load. For the remainder of this study speedups are computed with respect to the reference processor.

4.1 1D Fast Fourier Transform

Cfft1 was executed on a one-level homogeneous multiprocessor. The speedups obtained by varying the number of processors and processor speeds are shown in Figure 4. In this figure, the points indicated by "o" represent the one-level machine with optimal speedup for a given value of *neq*. The function defined by these points is denoted $R(.)$ and is used as a reference. Linear interpolation is used to obtain the values of R corresponding to values of *neq* that have not been experimentally generated.

Let $S_H(x)$ denote the speedup obtained for machine H when $neq = x$. The *percent gain* of H with respect to the optimal one-level homogeneous machine is defined as follows:

$$\text{percent gain}_H(x) = 100\frac{S_H(x) - R(x)}{R(x)} \qquad (4)$$

If positive, the percent gain indicates the additional speedup achieved over the optimal one-level homogeneous machine for the same budget.

Figure 5 shows the percent gain of Cfft1 on a two-level machine with one processor of varying speed in the first level. In this figure the "zero" line corresponds to the reference speedup. Except when the speed in the first level is one, each two-level machine reaches a peak performance above zero. The percent gain at this peak increases as the processor speed in the first level increases with respect to the processor speed of the second level. Additional experiments indicated that the performance degrades as the number of processors in the first level increases.

For a two-level machine, the sequential portion of Cfft1 can be mapped to the first level while the parallel portion can be mapped to the second level. Although this mapping resulted in a percent gain larger than zero, additional gain was obtained by using a hybrid configuration. This hybrid configuration consists of alternating between the native configuration of the machine and a one-level configuration as needed. For example, the mapping of Cfft1 on $\{L_{16,1}, L_{1,1^+}\}$ alternates between the native configuration $\{L_{16,1}, L_{1,1^+}\}$ and the configuration $\{L_{1,neq(L_{16,1})+1^+}\}$.

Figure 6 shows the percent gain obtained by executing Cfft1 on a three-level machine when the processor speed in the second level is two. The behavior of Cfft1

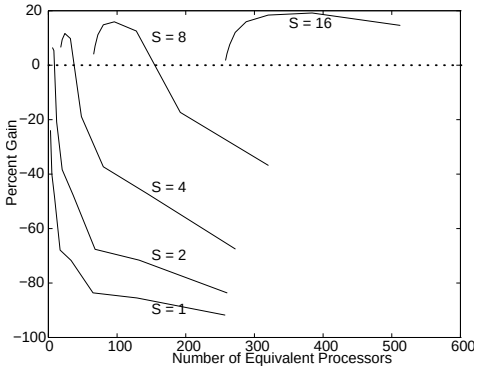

Figure 5: PERCENT GAIN OF Cfft1 ON $H = [L_{S,1}, L_{1,1^+}]$.

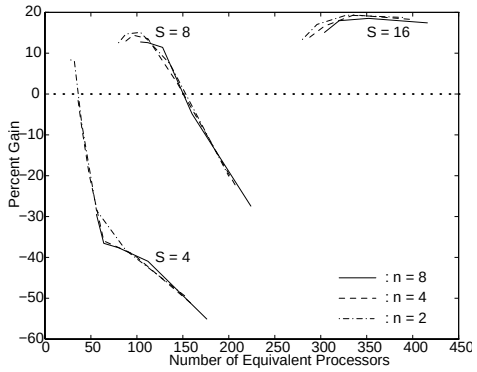

Figure 6: PERCENT GAIN OF Cfft1 ON $H = [L_{S,1}, L_{2,n}, L_{1,1^+}]$.

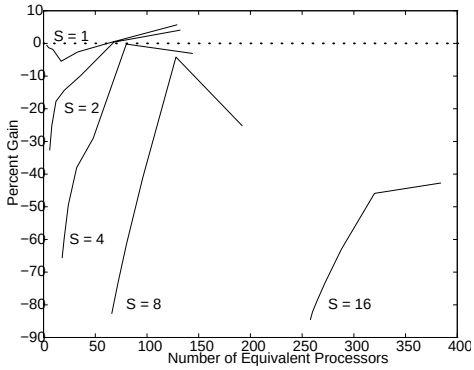

Figure 7: PERCENT GAIN OF Hfft1 ON $H = [L_{S,1}, L_{1,1^+}]$.

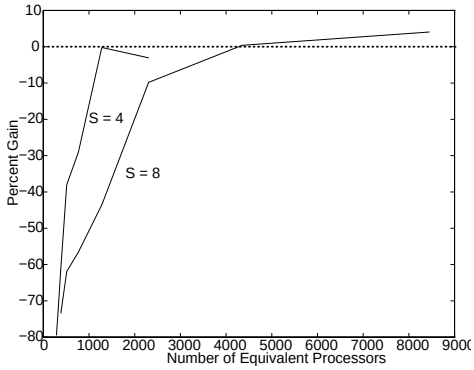

Figure 8: PERCENT GAIN OF Hfft1 ON $H = [L_{16,1}, L_{S,1^+}]$.

on a three-level machine is similar to its behavior on a two-level machine. The peak performance increases as the speed of the processor in the first level increases. A test was also conducted to evaluate the effect of varying the processor speed in the second level of the three-level machine on the percent gain. This test revealed that as the processor speed in the second level increases, the performance of the machine degrades. The configuration for the three-level machine was also hybrid. However, in this case the native configuration was not used. Only the two-level and one-level configurations of the three-level HPAM machine were used. For both the two-level and three-level machines, the peak performance will continue to increase as the processor speed in the first level increases with respect to the processor speed in the remaining levels.

The above-mentioned results lead to several conclusions about the behavior of Cfft1 and the adequate machine organization for its execution. For machines with a small budget (i.e., $neq \leq 50$), given the low return of a multilevel machine (percent gain $\leq 10\%$), the machine of choice is a one-level homogeneous multiprocessor machine. As the budget increases, the two-level and three-level machines out-perform the one-level machine. For the range of processor speed tested, the gain reaches 20% and increases as the processor speed in the first level increases. Thus, for mid-range to high-end budgets, Cfft1 is best executed on a two-level or three-level machine with a very fast processor in the first level and slow processors in the remaining levels. Although it was expected that Cfft1 will only map well onto a two-level machine, Figure 6 shows that it is possible for the three-level machine to achieve a percent gain greater than zero if a hybrid configuration is used.

Previous studies ([19] and references there in) have shown that an increase in performance can result from assigning the appropriate number (determined by the characteristics of the task) of identical processors to each task in the application. The mapping of Cfft1 on $H = [L_{1,1}, L_{1,1^+}]$ shows that, by itself, assigning the appropriate number of processors to each task might not be sufficient to obtain an increase in performance. In addition to adequate processor-task assignment, the heterogeneous features of HPAM are needed to achieve a gain in performance with respect to a one-level machine.

Several mappings of Hfft1 on a two-level machine were attempted. One of these attempts consisted of mapping the transpose operations in Hfft1 to the first level and usually fastest level of a two-level machine. This mapping resulted in a lower performance than actually mapping Hfft1 using the one-level configuration of the two-level machine. When Hfft1 is executed on a two-level machine the percent gain is negative (Figure 7). The loss in performance is small when the difference in speed between the two levels is also small. As this difference increases, the gap between the speedup of the one-level and the two-level machine increases. This is also indicated by Figure 8 which shows the percent gain for the machines $H = [L_{16,1}, L_{4,1^+}]$ and $H = [L_{16,1}, L_{8,1^+}]$. Moreover, for a given processor speed, the gap between the speedup of the two machines reduces as neq increases. For example, the percent gain of $H = [L_{16,1}, L_{8,1^+}]$ increases as the value of neq increases. It reaches +5% for $neq \geq 4000$.

Increasing the speed of the processors in the second level with respect to the first processor leads to an improvement in the performance of the two-level machine

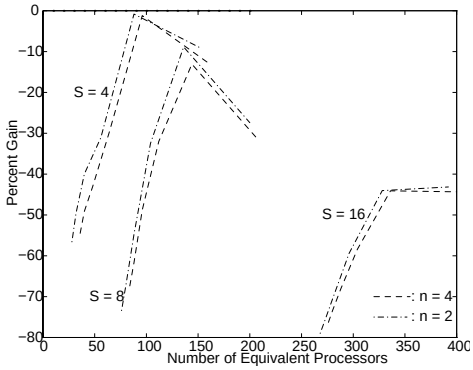

Figure 9: PERCENT GAIN OF Hfft1 ON $H = [L_{S,1}, L_{2,n}, L_{1,1^+}]$.

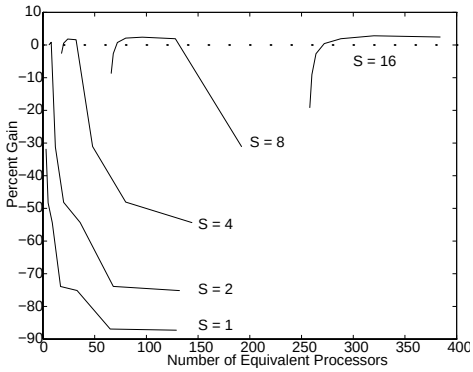

Figure 10: PERCENT GAIN OF Cradar ON $H = [L_{S,1}, L_{1,1^+}]$.

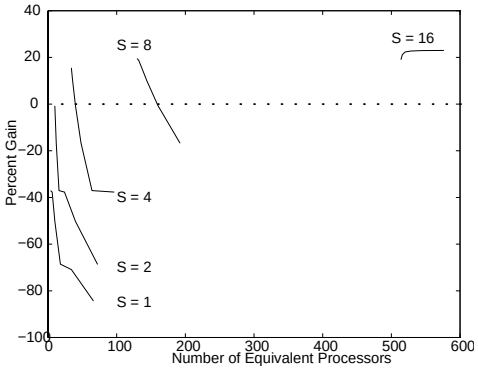

Figure 11: PERCENT GAIN OF Cradar ON $H = [L_{S,2}, L_{1,1^+}]$.

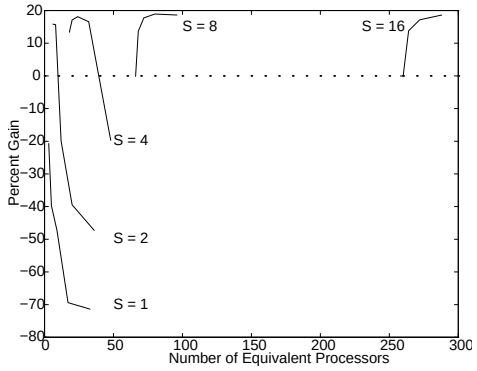

Figure 12: PERCENT GAIN OF Hradar ON $H = [L_{S,1}, L_{1,1^+}]$.

with respect to the one-level machine. This mainly occurs when the processor speed in the second level is as close as possible to the processor speed of the first level. This indicates that better performance for the two-level machine is achieved when it has a configuration that is closest to the one-level machine.

Mapping Hfft1 onto a three-level machine also resulted in a negative percent gain (Figure 9). The negative performance gain for Hfft1 on a two-level and three-level machines is due, among other factors, to the low CCratio of Hfft1.

4.2 Narrow Band Tracking Radar

Two different mappings of Cradar on two-level machines were attempted. The first consisted of assigning the sequential tasks to the first level and the tasks with $DoPs = 40$ and 8192 to the second level. The second approach consisted of using a hybrid configuration that alternates between the native configuration and the one-level configuration of the machine (This hybrid configuration is the same as the one used for Cfft1 on two-level machines). However, unlike Cfft1, Cradar is best executed without the use of any hybrid configuration.

Figure 10 shows the percent gain achieved when Cradar is executed on a two-level machine with one processor of varying speed in the first level. This figure indicates that the peak performance attained is 2%. This peak performance slowly increases as the speed of the processor in the first level increases. Figure 11 shows the percent gain of two-level machines when the number of processors in the first level is two. For these machines the sequential task was manually distributed among the two processors of the first level (automatic distribution of this task may not be possible with current parallelizing compiler technology). As indicated by Figure 11, a 25% peak performance was attained by $H = [L_{16,2}, L_{1,1^+}]$. The "superlinear" increase in performance achieved by adding a fast processor to the first level is due to the fact that this mapping overlaps communication between levels with computation in either the first or second processor of the first level.

Cradar was also mapped onto a three-level machine. This mapping consisted of assigning the sequential portion of the code to the first level, the tasks with $DoP = 40$ to the second level. The tasks with $DoP = 8192$ were distributed across the processors of the second and third levels. Thus, this mapping alternates between the native configuration and a two-level configuration of the three-level machine. In this case it was found that even if the number of processors in the second level increases, the peak performance is still less than or equal to 2%. This finding and a comparison between Figure 10 and Figure 11 indicate that Cradar is best executed on a two-level machine with two fast processors (speed ≥ 16) in the first level and about 40 very slow processors in the second level (for tasks with $DoP = 40$).

Several mappings of Hradar on two-level machines were tested. The first approach mapped the sequential and the tasks with $DoP = 512$ to the first level, while mapping the tasks with $DoP = 40$ to the second level. The second approach consisted of mapping the sequential tasks and the tasks with $DoP = 40$ to the first level, whereas the tasks with $DoP = 512$ were mapped to the second level. Finally, the third approach mapped the sequential tasks to the first level and the remaining tasks to the second level. Because the tasks with $DoP = 512$ are associated with a high degree of communication (transpose operations), the first map-

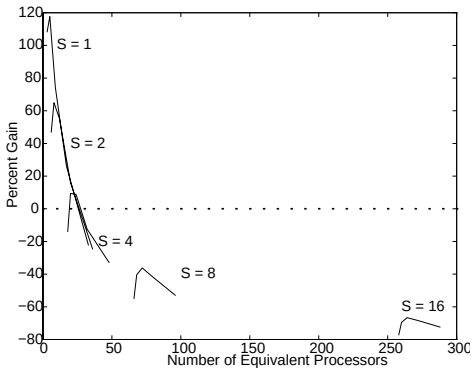

Figure 13: PERCENT GAIN OF Cairshed ON $H = [L_{S,1}, L_{1,1^+}]$.

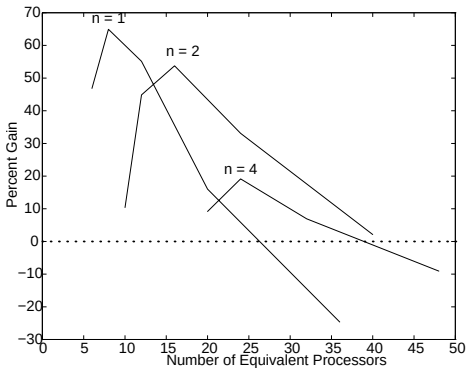

Figure 14: PERCENT GAIN OF Cairshed ON $H = [L_{2,n}, L_{1,1^+}]$.

ping out-performed the other two mappings in most of the cases tested. Thus, the communication properties of a task are as important as the DoP of the task in determining the appropriate mapping.

Figures 12, show the percent gain of Hradar using the first mapping mentioned above. Unlike Cradar, Hradar reaches an 18% peak performance as shown in Figure 12. Additionally, it was experimentally established that, while the peak performance of Cradar increases when the number of processors in the first level increases, the peak performance of Hradar decreases as the number of processors in the first level increases. This is due to the fact that a portion of Cradar that was executed in the first level has been parallelized in Hradar and has a $DoP = 40$.

Hradar was also mapped to different three-level machines. The mapping consisted of assigning the sequential portion of the code to the first level, the tasks with $DoP = 512$ to the second level and the tasks with $DoP = 40$ to the third level. Compared to the two-level mapping, the three-level machine has an additional 5% increase in peak performance.

4.3 Airshed Simulation

Figure 13 shows the percent gain when Cairshed is mapped onto two-level machines. This figure indicates that substantial gain can be achieved with an adequate task distribution regardless of speed. However, this increase in gain cannot be sustained as the size of the machine increases. When the number of processors increases in the first level of the two-level machine (Figure 14), the high percent gain can be sustained for a wider range of machine sizes. However, increasing the number of processors in the first level is also associated with a drop in performance. With support for fast reductions in the second level, and the use of one or two fast processors in the first level, it may be possible to achieve high percent gain and be able to sustain it for a wide range of machine sizes.

Hairshed was mapped onto a two-level machine using a hybrid configuration approach (i.e., the configuration alternates between the native and the one-level configurations of the two-level machine). Figure 15 shows the percent gain obtained for Hairshed when the processor speed in the first level varies. For any given first level processor speed, the percent gain starts below "zero" and increases toward values above "zero" as neq increases. Unlike the two-level heterogeneous machine organizations, the homogeneous machine $H = [L_{1,1}, L_{1,1^+}]$ does not address the effect of Amdahl's law. Thus, as shown in Figure 15 this machine has zero percent gain over the one-level homogeneous machine. The result of Figure 15 show, against our expectation, that Hairshed can benefit from a two-level heterogeneous machine.

The difference between the processor speed in the first and second levels was also varied. The result of this experiment indicated that the larger the difference the higher is the percent gain. For example, $H = [L_{8,1}, L_{1,1^+}]$ has higher percent gain than $H = [L_{8,1}, L_{2,1^+}]$ which in turn has higher percent gain than $H = [L_{8,1}, L_{4,1^+}]$. Additionally, for the range of values of neq tested, it was experimentally found that two-level machine organizations with small number of processors in the first level have higher percent gain than those with a larger number of processors in the first level.

Hairshed was also mapped onto three-level machines (Figure 16) with one processor in the first-level and varying number of processors in the second and third level. For this case, a hybrid configuration approach that alternates between the one-level and the two-level configuration of a given machine was used.

4.4 The Effects of Varying the Cost Model

In this study, the cost of a processor with speed α was assumed to grow in proportion to α^b, where $b = 2$. In order to test the sensitivity of the experimental results obtained with respect to b, the value of b was varied. Increasing b resulted in a shift to the right for all the plots and also in an increase in the difference between the speedup achieved for the optimal homogeneous machines and the heterogeneous machines. This increase translates into an upward shift if the percent gain is positive and a downward shift if the percent gain is negative. In [8], a similar behavior was observed when homogeneous multiprocessor machines using varying processor speed were compared.

5 Conclusions and Future Work

In this study each multilevel configuration was compared to the optimal one-level homogeneous machine. This optimal machine varies in size and processor speed for a given budget across the benchmarks studied.

Although a strict fixed budget test was used for this study, percent gains of 10% to 30% with respect to the optimal one-level homogeneous machine can be achieved in most cases using a multilevel HPAM machine for mid-range to high-range budgets. For low

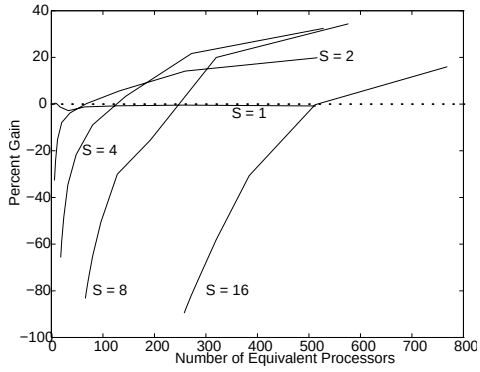

Figure 15: PERCENT GAIN OF Hairshed ON $H = [L_{S,1}, L_{1,1^+}]$.

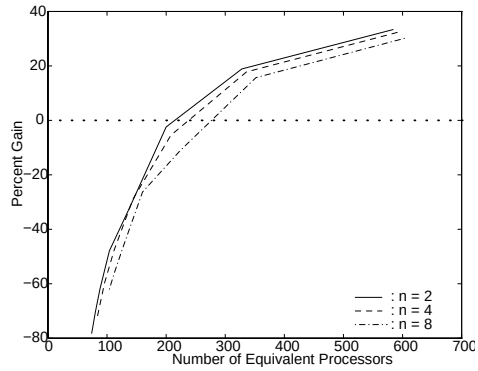

Figure 16: PERCENT GAIN OF Hairshed ON $H = [L_{8,1}, L_{2,n}, L_{1,1^+}]$.

budgets, the one-level homogeneous organization is the organization of choice. The exception benchmark is Hfft1. Because the fixed budget comparison had to be strictly observed and network cost was assumed to track processor cost, slow networks were used with slow processors and fast networks were used with fast processors. Using fast networks in the entire HPAM machine will add to the advantages of HPAM machines over one-level homogeneous machines under the fixed budget criterion.

In this study one benchmark (Hfft1) with a low CCratio had negative percent gain regardless of the HPAM configuration used. Cairshed also has a very low CCratio, which resulted in a small positive percent gain that cannot be sustained over a wide range of budgets. The CCratio can be improved for these benchmarks by (1) using fast networks throughout the entire HPAM machine (2) providing support for fast collective communication across levels and within levels. Future work includes the study of different implementations of collective communication in HPAM machines.

As shown for the case of radar, the communication load of a task can be as important as the degree of parallelism in determining which level the task should be mapped to in a given HPAM machine.

Finally, in 50% and 67% of the cases hybrid configurations of the two-level and three-level machines were used, respectively. This indicates that hardware and software support for reconfiguration is needed in HPAM. Furthermore, this support becomes more critical as the number of levels increases.

References

[1] Z. Ben-Miled and J.A.B. Fortes, A heterogeneous hierarchical solution to cost-efficient high performance computing. *Par. and Dist. Processing Symp.*, 1996, 138–145.

[2] Z. Ben-Miled, R. Eigenmann, J.A.B. Fortes and V. Taylor, Hierarchical processors-and-memory architecture for high performance computing. *Frontiers of Massively Parallel Computation Symp.*, 1996, 138–145.

[3] P. Dinda, T. Gross et. al., *The CMU Task Parallel Program Suite*, Tech. Rep. CMU-CS-94-131, Carnegie Mellon Univ., 1994.

[4] Z. Ben-Miled, A Hierarchical Heterogeneous Solution to High Performance Cost-Efficient Computing. *Ph.D. thesis,* Purdue University, June 1997.

[5] W. Blume, R. Doallo et. al., Parallel programming with polaris. *IEEE Computer*, 1996, 78–82.

[6] Z. Ben-Miled, J.A.B. Fortes, R. Eigenmann and V. Taylor, Towards the design of a heterogeneous hierarchical machine: A simulation approach. *30th Simulation Symp.*, 1997, 126–136.

[7] J.L. Henessy and D.A. Patterson, *Computer Architecture A Quantitative Approach*, (San Francisco, Morgan Kaufmann, 1990).

[8] M.L. Barton and G.R. Whiters, Computing performance as a function of the speed, quantity, and cost of the processors. *Supercomputing*, 1989, 759–764.

[9] J.B. Andrews and C.D. Polychronopoulos, An analytical approach to performance/cost modeling of parallel computers. *Par. and Dist. Computing*, 12, 1991, 343–356.

[10] S.Q. Moore and L.M. Ni, The effects of Network Contention on Processor Allocation Strategies. *IPPS*, 1996, 268-273.

[11] T.H. Dunigan, *Early Experiences and Performance of the Intel Paragon.* Oak Ridge National Laboratory, Tech. Rep. TM-12194, 1994.

[12] D. Basak and D.K. Panda, Designing Large Hierarchical Multiprocessor Systems under Processor, Interconnection, and Packaging Constraints. *ICPP*, I, 1994, 63–66

[13] J.J. Dongarra and T.H. Dunigan *Message-Passing Performance of Various Computers.* Oak Ridge National Laboratory, Tech. Rep. TM-13006, 1996.

[14] T. von Eicken, D.E. Culler et. al., Active Messages: a Mechanism for Integrated Communication and Computation. *ISCA*, 1992.

[15] C. Dubnicki, L. Iftode et. al., Software Support for Virtual Memory-Mapped Communication. *IPPS*, 1996, 372–381.

[16] Intel Supercomputer Systems Division. *Paragon System Overview.* Intel Corp., Santa Clara, CA, 1994.

[17] Cray Research. *Cray T3D System Architecture Overview.* Cray Research Inc., Tech. Rep. HR-04033, 1993.

[18] C.B. Stunkel, D.G. Shea et. al., The SP2 High-Performance Switch. *IBM Systems Journal*, 34(2), 1995, 185–203.

[19] K.P. Belkhale and P. Banerjee, A Scheduling Algorithm for Parallelizable Dependent Tasks. *IPPS*, 1991.