Impact of Computing-in-Memory on the Performance of Processor-and-Memory Hierarchies*

Renato J. O. Figueiredo†, José A. B. Fortes†, Zina Ben Miled‡,
Valerie Taylor†† and Rudolf Eigenmann†

†School of ECE-Purdue University
West Lafayette, IN 47907
{figueire,fortes,eigenman}@ecn.purdue.edu

‡Department of EE-Purdue University
Indianapolis, IN 46202
miled@engr.iupui.edu

††Department of ECE-Northwestern University
Evanston, IL 60208-3118 taylor@nile.ece.nwu.edu

Abstract

A Hierarchy of Processor-And-Memory (HPAM) can be viewed as an extension of the notion of a memory hierarchy. The extension entails the inclusion of processors with different performance in different levels of the memory hierarchy. Technology trends, applications behavior and previous research suggest that a multiprocessor system organized as a Hierarchy of Processor-And-Memory may offer considerable advantages over conventional multiprocessor organizations. This paper quantifies and analyzes the advantages of computing-in-memory, for a multi-level HPAM system, as compared to a conventional multiprocessor system with the same cost. The analysis entails using performance models and simulation data. Underlying the comparative study is the assumption that the cost of a processor is in proportion to the square of its performance. The 9 benchmarks used in the evaluations belong to the CMU, Perfect Benchmarks and SPEC95 suites. While the evaluation methodology takes into account the heterogeneity of HPAM, the emphasis is placed on modeling the impact of computing-in-memory on the relative performance of the multiprocessors under study. The results indicate that, with rare exceptions, HPAM outperforms conventional multiprocessor designs of identical cost by as much a 80% in the benchmarks and the ranges of model parameters considered in the study. In the very few exceptions to this conclusion HPAM is never outperformed by more than 20%.

[0]This research was supported in part by NSF grants ASC-9612133, ASC-9612023, CCR-9357781 and CDA-9617372. Renato Figueiredo is also supported by a CAPES grant.

1 Introduction

A system organized as a Hierarchy of Processor-And-Memory (HPAM) [23, 21, 22] can be viewed as an extension of the familiar notion of memory hierarchy. The extension consists of including processors into one or more levels of the memory hierarchy. Assuming that the top (i.e. first) level of the hierarchy is the fastest, any given memory level is extended with processors that are slower, less expensive and in larger number than those in the preceding level. Heterogeneity and computing-in-memory are thus inherent to an HPAM system. This paper quantifies and analyzes the advantage of computing-in-memory in HPAM as compared to conventional multiprocessors with the same cost.

There is considerable evidence that differences in the rate of improvement of processor, memory and I/O technology may call for novel computer organizations within the next 10 years. In particular, the increasing gap between CPU cycle times and memory access latency has motivated several proposals for either "bringing the processor to the (DRAM) memory" [3, 15] or "bringing the memory (SRAM) to the processor" . The main purpose of the proposed designs is to avoid the so-called "memory (latency) wall" [20] as well as the "memory bandwidth wall".

An HPAM also relies on computing-in-memory in order to avoid long memory access latencies. The analogy to memory hierarchy is present in two forms. One, already mentioned in the first paragraph, is the use of larger numbers of lower cost, slower processors in lower levels of the hierarchy (just as larger, less expensive slower memory technology is found in lower levels of the memory hierarchy). The second analogy is in relation to locality of references, the fundamental property of real programs that well-designed memory hierarchies exploit in order to provide small average access times. In [21] and subsequent studies it has been empirically established that there exists spatial and temporal locality of (degree of) parallelism. This behavior is exploited by HPAM by keeping "parallel" data "close" to the processors in the level that can process it with the desired degree of parallelism.

In a conventional multiprocessor system all processors are identical. According to [12] , as long as the cost and performance of processors (and implicitly the associated memory system) are linearly related, the best multiprocessor system is one which consists of as many of the best available processors as allowed by a given system budget.

However, also according to [12], a superlinear relation (as observed in a revisited [14] version of Grosch's law [8]) between cost and performance renders homogeneous solutions non-optimal for large enough number of processors. There is an optimal point in the cost-performance function (of the number of processors) beyond which homogeneous solutions are no longer desirable [9, 12, 2]. The intuition behind this conclusion is related to Amdahl's law. An heterogeneous machine executes sequential code on the fastest available processor and parallel code on the largest possible number of processors so that neither code segments become a bottleneck. In contrast, for a similar budget and a superlinear cost-to-performance relationship, a homogeneous machine would either use slow(er) processors for the sequential code segments or few(er) processors for the parallel code segments.

The relation between microprocessor cost and performance is a complex one, which manufacturers consider to be proprietary information. However, several arguments suggest that cost grows superlinearly with performance. The study reported in [14] establishes that such superlinear cost/performance relationship holds for microprocessors in different computational categories. In [12] it is argued that a quadratic cost/performance rule is conservative for high-end microprocessors. Assuming a (conservative) linear relationship between VLSI *area* and performance, yield-based cost models [10, 11] would suggest that chip cost increases in proportion to performance raised to $(\alpha + 1)$, where α is the number of critical masking steps (a typical value being $\alpha = 3$). It is also predicted that future improvements in performance will incur dramatically high development costs (for both microprocessor design and fabrication processes) [17]. Superlinear costs and the implication that homogeneous solutions may be suboptimal motivates the consideration of heterogeneity in future multiprocessor systems. Several studies [21, 23, 2, 12] have analyzed the potential cost-performance gains of such systems assuming (conservatively) that a quadratic law holds (i.e. $(\alpha+1)$=2). This assumption is also made for the HPAM design/models used in this paper.

The rationale and initial quantitative arguments for the HPAM concept appeared first in [21]. In [22], an execution-driven HPAM simulator (HPAM_Sim) that simulates multiple processors and networks of different speeds was used to analyze the impact of heterogeneity on the performance of HPAM. In contrast, the purpose of this paper is to quantify and analyze the advantages of computing-in-memory (i.e. in different levels of HPAM) against computing on a conventional "flat" multiprocessor system. Performance data was gathered by using an execution-driven simulator that models memory behavior "in the large" of selected benchmarks from several suites (Spec95, Perfect Benchmarks [5] and CMU [13]).

The rest of this paper is organized as follows. Section 2 introduces the HPAM machine model and the execution time model of HPAM and flat machines. Section 3 presents the simulation framework and qualitative evaluation methodology. Section 4 presents the simulation results and performance analysis. Section 5 concludes the paper.

2 Machine And Performance Models

A hierarchical processor-and-memory (HPAM) machine is a heterogeneous, multilevel parallel computer. Each HPAM level contains processors, memory and an interconnection network. The speed and number of processors, latency and capacity of memories and networks differ between levels in the hierarchy. The following characteristics hold across the different HPAM levels from top to bottom: individual processor performance decreases, number of processors increases, and memory/network latency and capacity increase.

Tasks are assigned to HPAM levels according to their *degree of parallelism* (DoP). Highly parallel code fractions of an application are assigned to bottom levels, while sequential and modestly parallel fractions are assigned to top levels.

In this paper, the simplest of the possible HPAM design points, with only two processor-and-memory levels, is studied, and its performance is compared to a homogeneous ("flat") machine with the same estimated cost. Both HPAM and flat machines are assumed to support a global shared address space. The

HPAM machine is specified only to the extent necessary to evaluate the impact of computing-in-memory. The relatively straightforward design is shown in Figure 1, along with the flat machine model. The homogeneous machine is composed of a number of interconnected processors with on-chip caches on the first level; extra (off-chip) cache storage on the second level; and main memory on the third level. In the HPAM machine, memory in the second level is augmented with processors. The second HPAM level is hence composed of connected processors with on-chip caches. The first level of HPAM contains only one processor, as powerful as one of the processors in the homogeneous configuration, and the second level contains several identical processors, less powerful than the processor in the first level (by about a factor of two in this paper). Such configuration is referred as a *2-level HPAM* throughout this paper, since only two HPAM levels are "active" (i.e., contain processors). It is assumed, for simplicity, that the network connecting the different levels is a bus.

The following design parameters, based on current technology values, have been assumed for both the flat and HPAM designs:

Processor performance: A base processor with clock frequency of 300MHz is assumed for the first level of both the homogeneous and HPAM configurations. Following the cost-oriented approach on which the HPAM concept is based, the processors of the second level of the HPAM machine must be slower than the base processor. It is assumed that performance, approximated by average number of instructions executed over a period of time, is proportional to the square root of cost [23]. Let FN_1 denote the number of processors with clock cycle $FCLK_1$ (assumed to be 1/300MHz=3.33ns) and average clocks per instruction $FCPI_1 = 1$ in the homogeneous machine; HN_2 the number of processors with clock cycle $HCLK_2$ and average clocks per instruction $HCPI_2 = 1$ in the second level of the HPAM machine; and assume that the first level of HPAM has only one processor. The clock cycle and average CPI of HPAM second-level processors is given by the following equation, which expresses a quadratic relationship between cost and performance.

$$HCLK_2 * HCPI_2 = \sqrt{\frac{HN_2}{FN_1 - 1}} * FCLK_1 * FCPI_1 \tag{1}$$

Bus performance: A bus clock of 60MHz and bus widths of both 16 and 32 Bytes are assumed. This choice is inspired by the use of a 66MHz, 16-Byte bus in Alpha 21164 [4] systems.

Cache sizes: The cache sizes assumed are based on miss rate measurements. The approach described in [16] is followed to obtain cache sizes that are scaled to the working set sizes of the benchmarks under study. Benchmarks from the CMU task-parallel, Perfect Club and Spec95 suites have been used in this study. The conclusion reached is that working sets for the CMU benchmarks are smaller than Perfect Club's, which are smaller than the Spec95 working sets. Therefore, different cache sizes for these suites have been chosen. The cache configurations used are: direct-mapped L1 caches with 32Byte blocks and 2-way set associative L2 caches with 64Byte blocks. The cache sizes used in the simulations are shown in Table 1.

	Level 1 (KB)			Level 2 (MB)		
	CMU	Perf.	Spec	CMU	Perf.	Spec
HPAM	16	16	64	0.5	1	2
Flat	64	64	256	0.5	1	2

Table 1: Cache sizes assumed in the simulations

Number of processors: Since a parallel region of code is assumed to execute concurrently without overheads, small processor counts need to be used to minimize the error that this assumption introduces in the execution model. Homogeneous configurations with 4 and 16 processors, and HPAM configurations with one processor in the first level and 16 processors in the second level are used in this paper.

2.1 Performance Model

The performance model represents the execution times for both HPAM and homogeneous configurations using the notation in Table 2 and Section 2. Figure 1 shows a graphical representation of the terminology used for access times across the hierarchy. The simulator is able to measure the parameters FIC_s, FIC_p, HIC_i, $FA_{1,j}$ and $HA_{i,j}$ of the execution time model.

The execution time expressions for the homogeneous and HPAM configurations are introduced in Equations 2 and 3, respectively. Equation 2 consists of three major terms. The first term is the time to execute the sequential fraction of the code. The second term is the time to execute the parallel fraction of the code, when all processors execute concurrently. The third term is the execution time contribution due to memory accesses to remote levels. It is assumed that the parallel fraction of the code can be executed without overheads due to synchronization and shared-memory traffic inside the level.

FN_1	number of processors in the flat machine
HN_i	number of processors in level i of HPAM
$FCLK_1$	clock cycle of the flat machine processors
$FCPI_1$	CPI, flat machine processors
$HCLK_i$	clock cycle of level i HPAM processors
$HCPI_i$	CPI, HPAM level-i processor
$CSIZE_i$	total cache space available at level i
$BSIZE_i$	block size of the cache at level i.
DoP_t	minimum degree of parallelism for a DO loop to be assigned to 2^{nd} level of HPAM
FIC_s	instruction count, sequential mode (flat)
FIC_p	instruction count, parallel mode (flat)
HIC_i	instruction count, HPAM level-i
$FAs_{1,j}$	number of memory references issued in the sequential code in the flat machine that are serviced by a cache in level j
$FAp_{1,j}$	number of memory references issued in the parallel code in the flat machine that are serviced by a cache in level j
$FA_{1,j}$	$FA_{1,j} = FAs_{1,j} + FAp_{1,j}$
$HA_{i,j}$	number of memory references issued in level i of HPAM that are serviced by a cache in level j
MAT_i	access time of the memory in level i
$FP_{1,j}$	$j = 1$: hit time of 1^{st} level cache $j > 1$: penalty of bringing a cache block from level j to 1 in the flat machine
$HP_{i,j}$	$j = i$: hit time of HPAM i^{th} level $j \neq i$: penalty of bringing a cache block from level j to level i
CLK_{bus}	clock cycle of the inter-level bus
W_{bus}	width of the inter-level bus

Table 2: Notation used throughout the paper

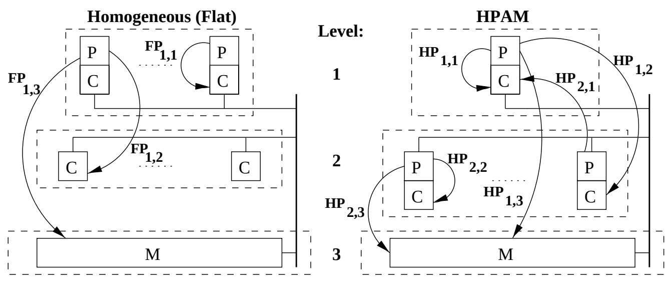

Figure 1: Parameters of the execution time model for 2-level HPAM and homogeneous configurations

Equation 3 is similar to Equation 2, the main difference being that instructions and memory accesses can be issued in any level of the hierarchy that contains processors. It is assumed that $HN_1 = 1$ and $HIC_1 = FIC_s$, and that there is no overlap in execution between levels 1 and 2.

$$FTime = (FIC_s * FCPI_1 * FCLK_1 + FAs_{1,1} * FP_{1,1}) + \left(\frac{FIC_p * FCPI_1 * FCLK_1 + FAp_{1,1} * FP_{1,1}}{FN_1}\right) + \left(\sum_{j=2}^{levels} FA_{1,j} * FP_{1,j}\right) \tag{2}$$

$$HTime = \left(\sum_{i=1}^{levels} \frac{HIC_i * HCPI_i * HCLK_i + HA_{i,i} * HP_{i,i}}{HN_i}\right) + \left(\sum_{i=1}^{levels} \sum_{j=1, j\neq i}^{levels} HA_{i,j} * HP_{i,j}\right) \tag{3}$$

Memory access times: The memory access times, together with the time to transfer a data block over the bus, are used to calculate the penalties $HP_{i,j}$ and $FP_{1,j}$. It is assumed that the on-chip L1 cache is accessed in two CPU clock cycles, as in the Alpha 21164 [4]. For the second-level cache, it is assumed that the same SRAM technology of the L1 cache is used. The memory access time has to be scaled according to the size difference between the two caches. The assumption used is that the second-level cache is divided into HN_2 banks, one bank per processor on the second level of the HPAM machine. Hence, each processor on the second level of HPAM has an on-chip cache of size $\frac{CSIZE_2}{HN_2}$. According to [18], the access time increase due to a larger on-chip cache size can be approximated by a constant factor when the cache size is doubled. This constant factor is approximately 14% for a 0.8μm technology [18]; this value is assumed in the performance analysis to obtain an expression for MAT_2 in terms of MAT_1 and the cache sizes.

Penalties $FP_{1,j}$, $HP_{i,j}$: There are two cases to be considered. Hit times $(i = j)$ are set to on-chip memory access times; hence, $FP_{1,1} = MAT_1$, $HP_{1,1} = MAT_1$ and $HP_{2,2} = MAT_2$. Miss penalties $(i \neq j)$ are given by the access time of the memory providing the data to serve the miss plus transfer time on the bus.

The argument for computing-in-memory is built upon the fact that off-chip accesses are expensive: the off-chip available bandwidth is smaller than the on-chip bandwidth due to packaging constraints (limited number of pins), and the latency is larger due to the necessity of driving off-chip lines with larger capacitance [3, 15]. For the configurations shown in Figure 1, the possible performance gains of an HPAM over a flat machine due to computing-in-memory are based on the fact that an access generated by an HPAM level-2 processor that hits in its local cache (with access time MAT_2) is faster than an access generated by a level-1 processor on the homogeneous configuration that misses on its local cache (access time $FP_{1,2}$). Even though such HPAM level-2 hit is slower than a level-1 hit in the homogeneous machine (access time MAT_1), it can be overlapped with several other "slow" hits in the remaining level-2 processors.

The off-chip overhead is modeled by Equations 4 and 5. The non-negative parameter OH_{PAM} represents potential overheads of accessing memory on a processor-and-memory chip compared to accessing a memory-only chip. Contention on the PAM memory ports when both the on-chip processor and an external processor request a memory access is thus captured by OH_{PAM}. This parameter is varied in the simulations in order to observe its impact on performance.

$$FP_{1,2} = MAT_2 + (1 + \frac{BSIZE_1}{W_{bus}}) * CLK_{bus} \quad (4)$$

$$HP_{1,2} = FP_{1,2} + OH_{PAM} \quad (5)$$

Table 3 shows the actual timings used in the simulations of CMU and Spec95 benchmarks. The range shown for $HP_{1,2}$ corresponds to OH_{pam} ranging from 0 to 50 base processor clock cycles.

W_{bus}	MAT_1	MAT_2	$FP_{1,2}$	$HP_{1,2}$
16B	6.7ns	12.8ns	63ns	63ns to 230ns
32B	6.7ns	12.8ns	46ns	46ns to 213ns

Table 3: Access times used in simulations of CMU and Spec95 benchmarks

Notice that $HP_{2,2} = MAT_2$ is about twice as large as $FP_{1,1} = MAT_1$, i.e., accessing on-chip memory in the second level of HPAM is twice as slow as accessing on-chip memory in the first level. However, when the level-2 processors overlap their memory accesses, the effective memory bandwidth is larger than the bandwidth available in the first level due to the larger number of available processors and independent memory banks.

As for the miss penalties to the main memory (level 3), a fixed value of 300ns is assumed for both configurations based on the main memory latency of an Alpha server [3, 24]: $FP_{1,3} = HP_{1,3} = 300ns$. It is assumed that $HP_{2,1} = HP_{1,2}$ and $HP_{2,3} = HP_{1,3}$.

3 Evaluation Methodology

The following programs have been studied in this paper: Airshed, Stereo, Radar, FFT2 (CMU suite); Ocean, Arc2D, FLO52 (Perfect Benchmarks); Hydro2d and Tomcatv (Spec95 suite).

All the benchmarks studied are originally sequential and written in Fortran. Each benchmark is instrumented with Polaris [19], a source-to-source parallelizing compiler, to identify parallel DO loops. Each identified parallel loop is "wrapped" with traps to a simulator engine at the beginning and end of the loop. The simulator is composed of an execution-driven engine and a multi-level cache simulator built on top of Shade [1]. A trap signals the engine that a parallel region of the code with a specific degree of parallelism has been reached.

The memory hierarchy simulator implements a data cache for each level of an HPAM configuration. Each cache is defined by its size, block size and associativity. Block sizes in a given level of the hierarchy must be either of the same size as a block of the upper level cache or larger by a power-of-two factor. The cache coherence protocol used is based on the MESI protocol with extensions to support memory accesses being potentially generated on every level of the hierarchy [7]. Instruction caches are assumed to be perfect (100% hit rate) by the simulator.

4 Results and Analysis

Table 4 presents the basic characteristics of the benchmarks studied: total instructions count (IC), degree of parallelism threshold used to determine parallel loops (DoP_t), fraction of instructions that are executed in sequential mode (IC_s), and local hit rates for levels 1 and 2 of the HPAM configuration. The hit rate of HPAM level L_i shown in Table 4 is defined as the number of data references issued by processors in level i of HPAM that hit in level i divided by the total number of references issued in level i.

The values of DoP_t shown in Table 4 correspond to the degree of parallelism threshold used to decide whether a DO loop is executed in HPAM in parallel

Bench.	IC	DoP_t	IC_s	Hit, L_1	Hit, L_2
Airshed	6.49E10	25	1.7%	93.3%	99.8%
Stereo	3.64E9	256	23.0%	97.4%	98.4%
Radar	2.63E9	400	97.4%	96.6%	99.7%
FFT2	3.59E9	64k	86.5%	99.4%	99.8%
Ocean	1.88E11	64	89.9%	98.9%	99.9%
Arc2D	1.49E11	16	0.04%	97.1%	99.5%
FLO52	3.10E10	16	1.2%	96.5%	99.9%
Hydro2d	4.13E11	64k	1.2%	97.0%	99.4%
Tomcatv	2.65E11	511	7.0%	98.4%	99.4%

Table 4: Benchmark summary: number of instructions, minimum degree of parallelism executed in the second level of HPAM (DoP_t), % sequential execution and L1 and L2 hit rates for the HPAM configuration

(in the second level) or sequentially. The flat machine executes all non-sequential loops in parallel. The value of DoP_t is chosen so that the percentage of code with degree of parallelism DoP, $1 < DoP < DoP_t$, is negligible.

The level hit rates shown in Table 4 show that the applications under study exhibit good locality with respect to degree of parallelism. A more detailed look at the distribution of accesses across the hierarchy [6] is presented in Table 5 for the benchmark Airshed. The total number of accesses is the same for both HPAM and flat machines; the way accesses are divided across the levels, however, is different.

The access distribution shows how accesses generated by the i^{th} HPAM level tend to be locally served (values shown in bold face). In this example, the sum of HPAM level-1 and level-2 hits ($HA_{1,1} + HA_{2,2}$) is larger than the number of flat level-1 hits ($FA_{1,1}$). In addition, the number of HPAM level-1 misses serviced by level 2 ($HA_{1,2}$) is about 3.3 times smaller than the corresponding number of misses on the flat machine ($FA_{1,2}$). This means that the HPAM processors hit their level caches more often than the flat design, even though the total cache capacity of the HPAM machine is smaller (see Table 1).

4.1 Performance Comparison

The approach taken to study the performance benefits of computing-in-memory consists of comparing a 2-level HPAM machine to a homogeneous machine. The basis of comparison is execution time, as modeled in Section 2.1. The two architectures are compared under two different scenarios. The choice of the scenarios is based on a *constant-cost* approach: a base homogeneous machine with four identical processors is assumed, and two different designs with equivalent cost are generated by following the cost/performance model of Equation 1:

Flat		
$FA_{1,1}$	$FA_{1,2}$	$FA_{1,3}$
4,421,280,188	**43,643,411**	3,604,499
HPAM		
$HA_{1,1}$	$HA_{1,2}$	$HA_{1,3}$
209,486,013	**13,417,183**	1,635,899
$HA_{2,1}$	$HA_{2,2}$	$HA_{2,3}$
6,561,554	*4,235,536,996*	1,890,453

Table 5: Distribution of accesses across levels for the benchmark Airshed on both 2-level HPAM ($HA_{i,j}$) and flat ($FA_{1,j}$) configurations.

Scenario 1: The homogeneous machine has 4 identical processors, each clocked at 300MHz and with a CPI of one. The HPAM machine has 1 processor on the first level identical to the processor used on the homogeneous machine; this processor is responsible for executing the sequential fractions of the application. The second level contains 16 processors, each of them clocked at 129MHz (value obtained from the model presented in Section 2, Equation 1). This scenario is based on the HPAM concept that fast processors are expensive and should be used in sequential computation while parallel computation should be handled by a larger number of slower processors.
Scenario 2: The HPAM machine is identical to the one described for Scenario 1. However, the homogeneous machine has sixteen processors instead of four. Hence both HPAM and flat machines execute parallel code with the same number of processors. The same principle used in the square-root cost model of Equation 1 is used to determine the clock speed of each processor based on the base clock speed of 300MHz: with the same cost of four 300MHz processors, it is possible to implement sixteen processors (for the flat machine) clocked at $\sqrt{4/16} * 300 = 150$MHz.

The simulation results presented in the rest of this section refer to these two scenarios. For a given scenario, the simulation plots show speedup, defined as execution time on the homogeneous machine divided by execution time on the HPAM machine, for varying PAM overhead OH_{pam}, with the bus width assuming values of 16 and 32 Bytes. The benchmark *Tomcatv* is first examined. Figure 2 shows the speedup of HPAM over a homogeneous configuration assuming Scenario 1. In this case, HPAM performs 34% to 42% better than the homogeneous case for the two bus widths considered and for OH_{pam} up to 50 clock cycles of the base CPU shown in the graph. Notice that for a bus width of 32 Bytes (dotted line of Figure 2), a level-1 cache block can be fetched to the cache in one bus cycle. Even with a wide bus, the time to access the level-2 cache from level 1 in the homogeneous configuration is larger than the time to access the level-2 cache from level 2 in the HPAM case, since the off-chip datapath is clocked at a slower rate than the on-chip datapath.

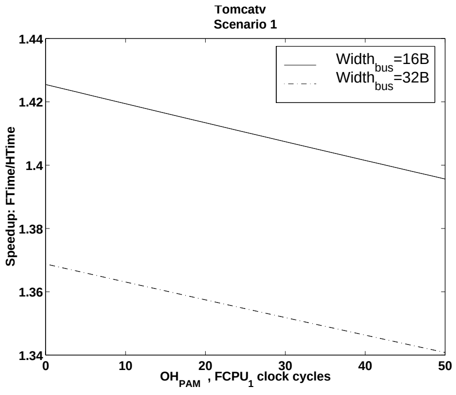

Figure 2: Speedup curve for Tomcatv, Scenario 1. Speedup is the ratio of the execution times, and the X axis is the PAM overhead modeled on Equation 5.

Bench.	Scenario 1, $OH_{pam} =$		Scenario 2, $OH_{pam} =$	
	0	50	0	50
Airshed	1.63	1.51	0.96	0.89
Stereo	1.08	1.01	1.19	1.11
Radar	1.00	0.87	1.78	1.54
FFT2	1.01	0.98	1.70	1.64
Ocean	1.01	0.97	1.81	1.73
Arc2d	1.64	1.63	0.92	0.92
FLO52	1.70	1.66	0.89	0.88
Hydro2d	1.55	1.53	1.01	1.00
Tomcatv	1.37	1.34	1.07	1.05

Table 6: Summary of results: Speedups (HPAM versus flat architecture) for scenarios 1 and 2, and PAM overheads of 0 and 50 base CPU clock cycles

In scenario 2, the homogeneous configuration is capable of executing the parallel fraction of *Tomcatv* faster than the HPAM machine can, because each homogeneous processor is 16% faster than the HPAM processors on the second level. However, the serial fraction of the code is executed in the homogeneous machine at 50% of the speed of the HPAM level-1 processor. As *Tomcatv* has about 7% of its execution in sequential mode (see Table 4), the time spent in the sequential fraction of the code is significant in the total execution time. As HPAM executes this portion of the code twice as fast as the homogeneous machine, the total execution time is 4% to 11% smaller for HPAM.

For an application with more available parallelism, *Airshed*, HPAM performs worse than the homogeneous configuration in Scenario 2. In this scenario, the homogeneous configuration executes the parallel fraction of the code faster than HPAM, and as the execution of the sequential fraction (about 1.7% of the code) is no longer the dominant factor in the performance, the overall execution time is smaller in the homogeneous case. However, HPAM performs significantly better for the same application if Scenario 1 is assumed.

Table 6 summarizes the results obtained for scenarios 1 and 2. The HPAM configuration outperforms the homogeneous configuration in Scenario 1 except for the cases where the application is highly sequential. Two reasons contribute to this performance advantage. First, The larger number of processors in the HPAM configuration. By following a constant-cost approach, assuming the quadratic cost/performance rule shown in Equation 1, the second level of the HPAM system has a larger aggregate performance than the homogeneous machine. Second, good locality of references with respect to degree of parallelism, combined with computing-in-memory in the second level. Good locality is responsible for keeping the inter-level communication low, while computing-in-memory provides fast access to on-chip memory for processors in the second level.

As for Scenario 2, HPAM outperforms the homogeneous configuration when the fraction of the code that is sequential is significant, such as in *Radar*. When the application is highly parallel, the homogeneous machine outperforms HPAM because each processor in the homogeneous machine is faster than an HPAM second-level processor, and because the fast processor on the first level of HPAM is idle during the parallel execution in the second level.

5 Conclusions

The collected experimental data shows that applications exhibit good locality of reference with respect to degree of parallelism. An HPAM machine benefits from such locality property in several ways. First, communication between HPAM levels is reduced, since most of the references issued from a processor tend to be serviced by a processor in the same level. In addition, computing-in-memory also benefits from locality, since processors in the lower hierarchy levels have fast access to on-chip data.

The performance analysis presented in this paper has shown that for several applications with different parallelism profiles, execution times are smaller for a 2-level HPAM system when compared to a homogeneous design, under a constant cost model. Computing-in-memory improves system performance in HPAM by allowing a large number of processors to overlap their accesses to local memory, thus exposing large available bandwidth in the slow memory levels.

In this paper, a very simple HPAM design point was presented, with only two active levels. The results indicate that the performance advantages of such a conservative HPAM architecture are already evident with current technology. Such conclusion motivates the ongoing research efforts in designing wider and deeper HPAM hierarchies, characterizing and mapping a broader class of applications to HPAM with optimized level scheduling algorithms, and simulating with finer accuracy a larger set of HPAM design points.

References

[1] B. Cmelik and D. Keppel. Shade: A fast instruction-set simulator for execution profiling. In *Proceedings of the 1994 SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 128–137, 1994.

[2] D. Menasce and V. Almeida. Cost-performance Analysis of Heterogeneity in Supercomputer Architectures. *Proc. Supercomp. Conf.*, pages 169–177, Nov 1990.

[3] D. Patterson et al. A Case for Intelligent RAM: IRAM. *IEEE Micro*, Apr 1997.

[4] Digital Equipment Corporation. *Alpha 21164 Microprocessor Hardware Reference Manual*, 1994.

[5] M. Berry et al. The perfect club benchmarks: Effective performance evaluation of supecomputers. *International Journal of Supercomputer Applications*, 3(3):5–40, 1989.

[6] R. Figueiredo et al. Impact of computing-in-memory on the performance of processor-and-memory hierarchies. Technical Report TR-ECE-98-1, ECE Department, Purdue University, 1998.

[7] R. J. Figueiredo, J. A. B. Fortes, and Z. Ben-Miled. Spatial data locality with respect to degree of parallelism in processor-and-memory hierarchies. In *Proc. 3rd VECPAR*, June 1998.

[8] H.A Grosch. Grosch's Law Revisited. *Computerworld*, Apr 1975.

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

[10] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1996.

[11] L. Gwennap. Estimating IC Manufacturing Costs. *Micropocessor Report*, August 1993.

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

[13] P. Dinda et al. The CMU Task parallel Program Suite. Technical Report CMU-CS-94-131, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, Jan 1994.

[14] P. Ein-Dor. Grosch's Law Re-Revisited: CPU Power and the Cost of Computation. *Communications of the ACM*, 28(2):142–151, Feb 1985.

[15] P.M. Kogge and T. Sunaga and H. Miyataka and K. Kitamura and E. Retter. Combined DRAM and Logic Chip for Massively Parallel Systems. *16th Conference on Advanced Research in VLSI*, 1995.

[16] S.C. Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. 22nd Annual Int. Symp. on Comp. Arch.*, July 1995.

[17] Semiconductor Industry Association. The National Technology Roadmap for Semiconductors, 1994.

[18] S.J.E. Wilton and N.P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report WRL ResearchReport 93/5, Digital Equipment Corporation, 1993.

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

[20] W.A. Wulf and A.A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM Computer Architecture News*, 1995.

[21] Z. Ben-Miled and J.A.B. Fortes. A heterogeneous hierarchical solution to cost-efficient high performance computing. *Par. and Dist. Proc. Symp.*, Oct 1996.

[22] Z. Ben-Miled, J.A.B. Fortes, R. Eigenmann, and V. Taylor. 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.

[23] Z. Ben-Miled, R. Eigenmann, J.A.B. Fortes, and V. Taylor. Hierarchical processors-and-memory architecture for high performance computing. *Frontiers of Massively Par. Comp. Symp.*, Oct 1996.

[24] Z. Cvetanovic and D.D. Donaldson. AlphaServer 4100 Performance Characterization. Technical report, Digital Equipment Corporation, 1997.