The Importance of Accurate Task Arrival Characterization in the Design of Processing Cores

Hashem H. Najaf-abadi Eric Rotenberg
Department of Electrical and Computer Engineering
North Carolina State University
{hhashem, ericro}@ece.ncsu.edu

Abstract

This paper studies the importance of accounting for a neglected facet of overall workload behavior, the pattern of task arrival. A stochastic characterization is formulated that defines regularity in the task arrival pattern. This characterization is used as the basis for a quantitative evaluation of the importance of accurately accounting for the task arrival behavior in the design of the processing cores of a Chip Multiprocessor (CMP).

The results of this study show that because the methodologies traditionally used for evaluating overall performance do not accurately account for task arrival behavior, they can lead to significantly suboptimal design solutions. For instance, it is found that, for an unvarying mix of benchmarks, the best dual-core design for harmonic mean performance can result in up to 21% suboptimality depending on the task arrival pattern. In addition, it is shown that when the pattern of task arrival is prone to change, simply accounting for average task arrival behavior can result in up to 12% inaccuracy, and suboptimality in employed design solutions.

A practical conclusion that can be drawn from the results of this study is that benchmark vendors need to provide not only a representative mix of instruction level behavior (the traditional application benchmarks), but also representative task arrival patterns. In addition, robustness to variation in the task arrival pattern needs to be accounted for as an overall merit in the evaluation of potential design solutions.

1. Introduction

In the design of any processing system, it is the behavior of the workload to be executed on the system that justifies the selection of one reasonable design solution over another. The less that is known about the workload behavior, the more arbitrary the playing field will be, and the more suboptimal the employed design solutions may in practice turn out to be.

The arrival pattern of tasks with different instruction level behavior is an aspect of overall workload behavior that has not received the attention it deserves. For instance, tasks may tend to arrive independently or in bursts, they may arrive at a high or a low rate, and the rate may be consistent or variable. These factors can impact the best design solutions to be employed in the design of a multi-core system – where the availability of cores can depend on how tasks are assigned to them. Traditionally, however, only the *mix* of instruction level behaviors (represented by the traditional application benchmarks) has been accurately accounted for in the methodologies used for evaluating the performance of processing systems.

Execution throughput is often accounted for in the performance evaluation of the communication resources of parallel systems [1]. However, the focus of this study is on the effect of the task arrival pattern on the best design for the processing cores themselves. Although germane to the design of any processing system, the results of this study are more prevalent to Chip Multiprocessor (CMP) design, where the selection of the core configurations is ingrained in the design of the system.

In general, the turnaround time of a task executed on a given system is determined by not only how fast the system can execute the task, but also how long it will take for resources to become free for the execution of the task in the first place. A system in which design choices have been made based on only the execution time of tasks in isolation, may provide high performance to certain workloads at the cost of lower performance to others. In a real-world setting, the less suitable workloads will occupy system resources and stall the execution of the more suitable workloads. This can result in lower overall performance than an alternative design may have provided. Therefore, accounting for the full effect of workload behavior requires an accurate understanding of how tasks tend to arrive at the system relative to each other.

Note that any approach to measuring overall performance can be viewed as being representative of some form of task arrival behavior. However, the objective of this study is to determine whether such task arrival patterns provide a good enough approximation, or whether it is important to accurately account for more representative task arrival behavior. It is with this goal in sight that we formulate different task arrival characteristics and empirically evaluate their effect on overall performance.

We believe that this study presents a number of valuable contributions. Among them:

- A stochastic characterization of task arrival behavior is formulated, based on which regularity is defined. This lays the groundwork for coherent evaluation of the effect of task arrival behavior on overall performance.
- It is empirically shown that, under typical technology characteristics, accounting for the task arrival behavior is crucial to accurate and insightful performance evaluation of different design solutions for the processing cores in multi-core systems.
- It is shown that when the task arrival behavior varies over time, averaging out the task arrival characteristics can result in notable inaccuracy and incorrect conclusions regarding the best design solutions. This means that the pattern of variation of the task arrival behavior over time also needs to be accounted for, not only an aggregation of characteristics.

978-1-4244-5156-2/09/$26.00 ©2009 IEEE

- Based on the observed results, two recommendations are made for attaining better accuracy in performance evaluation. One is that benchmark vendors, in addition to providing a representative mix of workload behavior (in the form of a set of application benchmarks), also need to provide representative task arrival patterns. The other is that robustness to variation in the task arrival pattern needs to be accounted for in the overall merit of any design solution.

The next section describes the overall methodology used for empirical evaluation throughout this study. Subsection 2.1 introduces the set of core designs used in the evaluation. Subsection 2.2 describes how task arrival is simulated and performance is measured. In subsection 2.3, a model of task arrival behavior and the notion of regularity in task arrival are formulated. Section 3 explores the interaction between task arrival characteristics and the choice of core designs in dual-core systems. Section 4 studies the effect of averaging out the different task arrival characteristics when they are not constant (resulting in *irregularity*). Section 5 explores the interaction between task arrival characteristics and two other design aspects of multi-core systems: scheduling and the number of cores. Section 6 discusses recommendations that stem from results of this study. Section 7 presents an overview of related work. Section 8 discusses further issues regarding the effect of task arrival behavior on overall performance, and Section 9 concludes this study.

2. Experimental Methodology

2.1. The Palette of Core Designs

To evaluate the interaction between the core designs employed in a multi-core system and task traffic characteristics, we use a predefined palette of core designs. This is a matter of convenience: our goal is not to prescribe designs, as in other work [3], but rather to understand the suboptimality caused by composing systems with specific traffic characteristics in mind and then applying alternate traffic characteristics. The palette is comprised of core designs that have each been customized (*i.e.*, tailored for optimum performance) for an individual SPEC2000 integer benchmark in 45nm technology. Customizing cores to benchmarks is a convenient and systematic way of deriving differentiated cores that cover the space of instruction level behavior.

The customized cores were attained through a simulated annealing exploration process. More precisely, the different design parameters were randomly varied, and for each variation the effect on performance was measured through cycle-accurate simulation. If the new configuration performed better than the best configuration observed until then, that configuration was recorded as the best. If it displayed less than half the performance of the best configuration, the exploration would rollback. Otherwise, another design parameter was randomly varied, and the exploration process was continued.

The superscalar width, register-file/ROB size, issue-queue size, L1 and L2 cache configurations, and clock frequency are among the design parameters varied. The performance of each of the intermediate configurations was measured through cycle-accurate simulation of the execution of the benchmark on the *sim-mase* simulator [16] of the Simplescalar V4.0 toolset configured correspondingly. The pipeline depth of each microarchitectural unit was extracted from the propagation delay of the unit, the clock frequency of the system, and the latch delay. The CACTI V5.3 modeling tool [15, 17] was employed to estimate the propagation delay of the different design units.

The load-store queue was modeled as a fully-associative structure with two read and two write ports, with each entry consisting of 8 bytes. The wakeup logic was modeled as a fully-associative structure with as many read and write ports as the issue-width of the design. The register file was modeled as a direct-indexed structure with as many write ports as the issue-width, and two times as many read ports. A number of design parameters were set constant throughout the design space and across all design solutions. Specifically, a fixed delay of 0.3ns was employed for all pipeline latches, a fixed delay of 50ns for memory access, and a fixed delay of 2ns for the frontend of all designs.

The customized cores are documented in Table 1. The performance of each benchmark on every core design is shown in Table 2. In both tables, a column corresponds to the customized core of a benchmark. In Table 2, rows correspond to benchmarks and columns correspond to core designs. Each entry shows the performance of the corresponding benchmark on the corresponding core design. Notice that the performance of a benchmark is highest when executed on its own customized core design, as is shown in boldface. The evaluations presented in following sections have been repeated with the customized cores for 70nm technology, and similar overall conclusions were observed.

Table 1: Microarchitectural configurations customized to individual SPEC2000 integer benchmarks (in 45nm technology).

	customized cores										
	bzip	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr	crafty
No. of cycles for memory access	196	299	298	174	124	298	342	199	224	198	213
No. of pipeline stage of the front-end	7	11	11	6	4	11	13	7	8	7	8
Dispatch, issue, and commit width	4	6	5	6	6	6	4	4	6	6	4
ROB size	512	256	64	256	128	256	256	512	512	256	256
Issue queue size	32	32	32	8	64	32	32	32	128	32	8
Min. lat. for awakening of dep. Instr.	0	1	1	0	0	1	2	0	2	0	0
Pipeline depth of Scheduler/Reg-file	1	2	1	1	0	2	2	1	2	1	1
Clock period	0.2889	0.2004	0.2009	0.3218	0.4393	0.2008	0.1787	0.2851	0.2576	0.2871	0.2688
L1D associativity	2	2	5	1	1	5	3	2	4	2	2
L1D block-size	1024	128	1024	128	32	1024	64	256	128	1024	128
L1D no. of sets	64	8	256	32	64	256	16	256	128	64	8
L1D access latency	1	2	1	1	4	1	8	1	8	1	8
L2D associativity	26	5	7	4	27	7	6	14	4	14	5
L2D block-size	1024	64	16	32	4096	16	128	8192	16	32768	128
L2D no. of sets	512	256	1024	512	256	1024	64	256	512	64	128
L2D access latency	4	8	16	16	4	16	16	1	16	1	16
Load-store queue size	40	56	40	48	64	32	48	40	104	40	32

Table 2: Performance (in billions of instructions per second) for each benchmark executed on each core type (each column represents the customized core of a benchmark, and each row represents a benchmark).

		customized cores										
		bzip	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr	crafty
benchmarks	bzip	**4.1734**	2.1885	2.607	2.4051	3.361	2.867	2.4757	4.0105	2.3943	4.0515	2.8292
	gap	1.4951	**3.7109**	3.2739	3.3555	1.0336	3.2634	3.4548	2.2491	3.064	2.2327	3.3595
	gcc	1.2521	1.7774	**2.922**	2.8324	0.8463	2.9124	1.8726	1.9414	2.5624	1.9202	2.6123
	gzip	2.0661	2.71	3.669	**3.8266**	1.4165	3.6124	2.568	2.8644	2.8268	2.8319	3.7713
	mcf	0.7012	0.3558	0.3026	0.2906	**1.1099**	0.3197	0.4498	0.8665	0.347	1.0361	0.4461
	parser	2.1554	2.8362	3.129	3.0089	1.4966	**3.1753**	2.2698	2.8763	2.6505	2.8166	2.8955
	perl	0.9033	1.4391	2.2446	2.2668	0.5777	2.2576	**2.5087**	1.4271	2.2317	1.4196	2.3861
	twolf	1.9976	1.2075	1.1751	1.2301	1.4213	1.3847	1.1861	**2.6381**	1.2554	2.5647	1.4537
	vortex	1.7695	3.6411	3.4799	3.4652	1.2109	3.6811	3.0811	2.5995	**3.9079**	2.5811	3.5429
	vpr	1.7249	1.6072	1.3358	1.2136	1.2091	1.5366	1.6102	2.3473	1.6937	**2.3849**	1.6204
	crafty	0.8005	1.4256	2.4766	2.338	0.5165	2.5255	2.3666	1.3342	2.4809	1.3402	**2.6707**

2.2. Simulation of Task Arrival Behavior

For the purpose of this study, we consider the inverse of the average turnaround time of tasks to represent *overall performance*. Unlike other metrics, this metric is not based on any assumption about the task arrival pattern. Thus, it can serve in evaluating the effect of different task traffic behavior on performance. However, in order to measure this figure of merit there needs to be a system in which the task arrival behavior is actually modeled.

For the evaluation of task traffic behavior in this study, a simulator has been developed that emulates the queuing and occupation of different processing cores for the different considered workload types. Tasks are generated according to a random process, according to the stochastic parameterization outlined in the next subsection. Tasks are primarily placed in the dedicated task-queue of the core with the most suitable design for the workload type of the task. If the most suitable core is occupied, the task is directed to the next suitable core. If all cores are occupied, the task waits for the most suitable core to become available. When there are cores with the same microarchitectural configuration in the system (*e.g.*, a homogeneous design), tasks are randomly assigned to them based on availability. Once the execution of a task is complete and the core is free, the next task at the head of the task queue employs the core for execution.

Tasks occupy cores for an amount of time proportional to the performance of the workload type of the task on that core type. Specifically, all tasks are considered to consist of 3.2 billion instructions. The amount of time a core is occupied by a task is determined by the rate with which the task's workload type is executed on the microarchitectural configuration of that core (Table 2). As an example, a task of the *bzip* workload type will be executed on its own customized core in $(3.2\times10^9)\div4.17$ nanoseconds (or 0.77 seconds), and will be executed on the customized core of *gap* in $(3.2\times10^9)\div2.18$ nanoseconds (or 1.47 seconds).

2.3. The Facets of Task of Arrival Behavior

In general, task arrival behavior can be characterized in numerous different ways. What is important for the purpose of this study is that the characterization be reproducible with simple stochastic parameters. Moreover, it is necessary that it provide a breakdown of the different factors that affect task arrival behavior in real world environments. It is with this objective that we define the notion of *regularity* in the task arrival pattern.

We consider the task arrival pattern to be regular when the following task arrival characteristics remain invariable throughout the time interval of concern:

- *Workload mix* (the relative frequency of the arrival of tasks of different workload types): Throughout this study, unless specified otherwise, the workload mix is considered to be even across all benchmarks. An uneven workload mix (tasks of different workload types not having an equal arrival rate) can be parameterized by the percentage of overall load belonging to each benchmark.
- *Task arrival load* (the rate and distribution with which tasks are submitted to the system): This characteristic can be parameterized by the average latency between tasks arriving at the system, and its distribution. Throughout this study, we consider the latency between task arrivals to primarily have a normal distribution.
- *Burstyness* (multiple tasks of the same workload type arriving together): This characteristic can be parameterized with three factors: 1) the percentage of tasks that arrive in bursts, 2) the number of tasks that arrive together, and 3) the set of workload types that produce tasks in bursts. Burstyness causes the distribution of the latency between task arrivals to deviate from a normal distribution. Throughout this study, the latency between bursts is considered to have a normal distribution.
- *Workload correlation* (the arrival of tasks of different workload types influencing each other): This characteristic can be represented by 1) the workload types that are correlated, 2) the strength of the correlation, i.e., the percentage of tasks that abide by the correlation, and 3) how the arrival of tasks of a certain workload type are influenced by or influence the arrival of tasks of the other workload types. In this study, we focus on workload correlation in which tasks of correlated workload types arrive together. Other forms of correlation are also conceivable. Note that workload correlation does not necessarily cause the task arrival of any individual benchmark type to deviate from a normal distribution, or the workload mix to become uneven.

Note that these different characteristics are not totally independent of each other. For instance, non-bursty uncorrelated task arrival behavior is more representative of server environments, where users tend to issue single-threaded jobs to the system independently of each other. Thus, the workload mix that is more typical of a server environment may be more fitting for such task arrival patterns. The more multi-threaded the jobs are, the more correlated and bursty the task traffic behavior will be. If the thread-level parallelism is extracted from

loops, there will be burstyness. If it is extracted from different code regions, there will be correlation between the workload types of the different code regions.

3. Interaction between Choice of Core Designs and Task Arrival Characteristics

This section focuses on the choice of core designs of dual-core systems. Note that with a larger number of cores the overall turnaround time and execution throughput of the system may change. However, the overall conclusions of this study regarding the importance of the task arrival pattern will not.

Our approach is to compose various dual-core system designs (from the palette of core designs, documented in Subsection 2.1) that achieve the highest possible performance under specific design-time assumptions about the task arrival load, burstyness, and correlation. Subsequently, the suboptimality of such design solutions is evaluated under alternative task arrival behavior. In this manner the importance of accurate characterization of the task arrival pattern is exemplified.

3.1. Non-Bursty, Uncorrelated Traffic

Figure 1 shows the average turnaround time of tasks submitted to three differently designed dual-core systems. One is the best homogeneous design, employing two copies of the customized core design of *vpr*. The other is the best heterogeneous design for light traffic loads (under an average arrival rate of 0.0055 tasks per second), consisting of the customized core designs of *parser* and *vpr*. The third is the best heterogeneous design for heavy traffic loads (an average task arrival rate of 0.88 tasks per second), consisting of the customized cores of *crafty* and *vpr*. The best homogeneous design is not influenced by the traffic load. The results are for non-bursty, uncorrelated traffic.

Note that in all following figures that contain two graphs, unless indicated otherwise, the graph on the right provides a close-up view of light to medium task arrival rates. The graph on the left shows the turnaround time for the full spectrum of task arrival rates, from no contention to saturation.

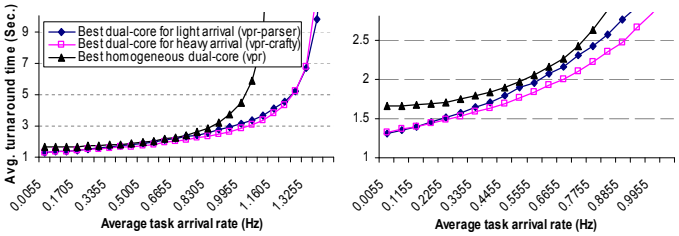


Figure 1: Avg. turnaround time with non-bursty traffic.

These results show that heterogeneity can result in 21% less average turnaround time in light traffic loads and provide 19% greater bandwidth (the maximum task arrival rate that the system can sustain). At the point where the system reaches saturation (the point at which average turnaround time increases unboundedly) the average turnaround time is around 4 times that under a light traffic load. In light traffic loads, the best design for light traffic loads displays only slightly better performance than the best design for heavy traffic loads. However, in heavier traffic loads (0.88Hz), the best design for light traffic loads performs close to 10% poorer than the best design for heavy traffic loads.

3.2. Bursty, Uncorrelated Traffic

The results of Figure 2 show the average task turnaround time with half-frequency double-task burstyness, *i.e.*, half of all tasks arrive at the systems in bursts of two tasks. No benefit remains for heterogeneity in light traffic loads. In fact, the best dual-core design for light traffic loads is homogeneous, consisting of two copies of the customized core of *twolf*. The customized cores of *gcc* and *vpr* comprise the best dual-core design for heavy traffic loads.

Burstyness causes the average turnaround time to increase across all arrival rates. For instance, the average turnaround time of the best heterogeneous design for light traffic loads increases 23%, from 1.34 seconds (for non-bursty) to 1.65 seconds (for bursty). The increase is less for the homogeneous design. However, burstyness does not majorly affect the bandwidth of the system (compare with Figure 1). Thus, in heavy traffic loads, heterogeneity remains of noticeable benefit, still providing 19% greater bandwidth.

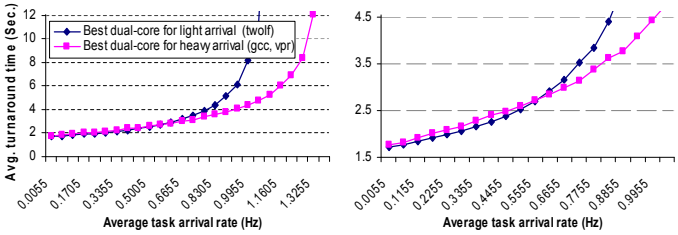


Figure 2: Average turnaround time with half-frequency double-task bursty traffic.

Figure 3 shows the average task turnaround time with full-frequency double-task burstyness, *i.e.*, all tasks arrive at the systems in bursts of two tasks of the same workload type. The best dual-core design for heavy traffic loads is heterogeneous and consists of the customized cores of *parser* and *vpr*. The best dual-core design for light traffic loads is once again homogeneous, consisting of two copies of the customized core of *vpr*.

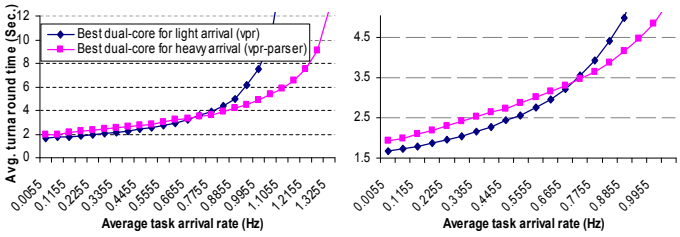


Figure 3: Average turnaround time with full-frequency double-task bursty traffic.

The heterogeneous design is outperformed by the homogeneous design in light traffic loads by more than 17%. Compared to half-frequency burstyness, the heterogeneous design observes a 14% increase in average turnaround time in light traffic loads, while that of the homogeneous design barely changes. However, the heterogeneous design does still provide a bandwidth improvement consistent with non-bursty traffic.

In a heterogeneous design, when tasks of the same workload type arrive in bursts, some tasks may be forced to be executed on unsuitable cores. It is for this reason that heterogeneity results in lower overall performance in the presence of burstyness (assuming light traffic loads). The different core

designs individually achieve better performance under certain workload types by sacrificing performance under others.

3.3. Homogeneity and Task Correlation

In the results presented so far, we see that the task arrival pattern can affect whether homogeneity or heterogeneity is the best design choice, and the best choice of core designs (from our palette) to employ when heterogeneous. But, if the design of the system were to be limited to homogeneity, neither the load nor burstyness of task arrival will majorly affect the best choice of core design (assuming the workload mix is evenly distributed and remains unchanging).

For instance, Figure 4.a shows the average task turnaround time in a homogeneous system employing the customized core design of *vpr*, and that of a homogeneous system employing the customized core design of *crafty*. The *vpr*-based design yields lower turnaround time across the entire arrival rate spectrum and higher execution throughput under heavy loading. In other words, the best homogeneous dual-core system is not sensitive to arrival rate.

What does affect the best core design to employ in a homogeneous system is correlation between the tasks of different workload types. Simply as an example, consider a non-bursty task arrival pattern in which all tasks are preceded by a $1/10^{th}$ task of the *gzip* workload type (consisting of 0.32 billion instructions). Note that the workload mix remains unchanged, only the pattern of task arrival differs.

The results of Figure 4.b show the average task turnaround time for this form of correlated task arrival. Both systems observe ~10% increase in average task turnaround time in light arrival loads. The reason for this is that the short *gzip* tasks stall the execution of their correlated tasks. This adds up to a considerable drop in the execution throughput.

More importantly, the *gzip* benchmark performs considerably poorer on the best core design for uncorrelated task arrival (the customized core of *vpr*). This translates into slightly lower overall performance than the customized core of *crafty* in medium task loads.

Such correlations can accumulate and create larger differences in performance. All in all, these results show that the pattern of task arrival, even with a fixed workload mix, can influence the best homogeneous design.

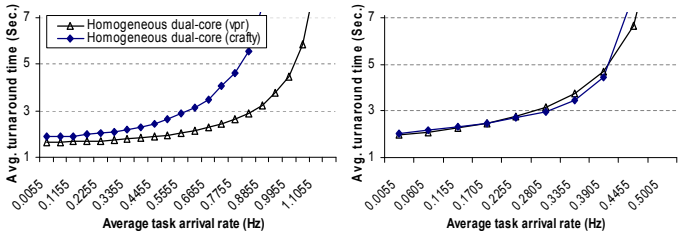


Figure 4: Avg. turnaround time for two dual-core homogeneous designs under (a) uncorrelated, and (b) correlated task arrival (see text for description).

3.4. Section Summary

All in all, the results in this section show that notable suboptimality can result from disparity between the task arrival behavior accounted for in the design of a system and that which is observed at run-time. For instance, designing a dual-core system for high performance in lightly-loaded full-bursty arrival will lead to a homogeneous design solution that displays 21% suboptimal performance in non-bursty arrival. On the other hand, designing a dual-core system for high performance in lightly-loaded non-bursty arrival will lead to a heterogeneous design solution that displays 17% suboptimal performance in full-bursty arrival.

4. Irregular Task Arrival

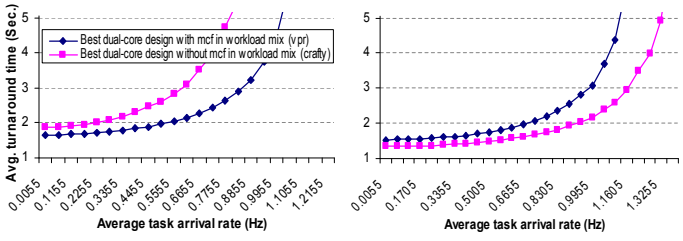
When any of the factors outlined in Section 2.3 are not constant, the task arrival behavior is *irregular*. An issue of interest is whether a regular task arrival pattern can be used to model an irregular pattern. Also, the potential for change in the task arrival behavior raises the issue of design *robustness*: the capacity to perform acceptably well under different task arrival characteristics. The following section addresses these issues.

4.1. Irregularity in the Workload Mix

An assumption made throughout the prior section is that the task traffic is distributed evenly across the different benchmarks. Under this assumption, however, the high performance of the best multi-core design may be mostly due to one of the cores in the system providing extremely high performance for a certain workload type. Thus, in environments where the workload type happens to appear infrequently, the multi-core design may be extremely suboptimal in the overall performance it provides.

For instance, among the SPEC2000 integer benchmarks, *mcf* is the most different. Its customized core provides extremely suboptimal performance for all the other benchmarks and the customized cores of the other benchmarks provide suboptimal performance to *mcf*. Although the customized core design of *mcf* is not among the best cores to employ in any dual-core design, the presence of the benchmark *mcf* in the workload mix heavily influences the best combinations of cores. For instance, when *mcf* is excluded from the workload mix, the best core design to employ in a homogeneous dual-core system for non-bursty task arrival is the customized core of *crafty*. With *mcf* present in the workload mix, the best core design is the customized core of *vpr*.

Figure 5.a compares the average task turnaround time in a homogeneous dual-core system employing the customized core of *vpr*, to one employing the customized core of *crafty*, when the benchmark *mcf* is present in the workload mix. These results show that employing the customized core of *crafty* rather than that of *vpr* increases the average turnaround time by 14% in the lowest traffic loads and decreases the peak bandwidth of the system by 30%.



(a) *(b)*

Figure 5: Average task turnaround time when *(a) mcf* is, and (b) *mcf* is not, in the workload mix of the actual task arrival pattern applied to the system.

Figure 5.b, on the other hand, shows the average task turnaround time with the same dual-core systems when the benchmark *mcf* is not present in the workload mix. These results show that employing the customized core of *vpr* rather than that of *crafty* increases the average turnaround time by 14% in the lowest traffic loads and decreases the peak bandwidth of the system by 17%.

The results of Figure 5 confirm the well-known fact that it is important to select a representative workload mix in performance evaluation, and that accounting for different workload mixes is essential to finding a design solution that is robust. In this simple example, employing the customized core of *vpr* is the more robust solution as it outperforms *crafty* when *mcf* is in the workload mix, and displays comparably less suboptimality when *mcf* is not.

Less established is the fact that any accounting for irregularity in the workload mix of the task arrival pattern is essential to the accuracy of performance evaluation. Averaging out the presence of different workload types in the workload mix can lead to misleading conclusions. For instance, consider a task arrival pattern in which half of the time (assuming phase-based variation) the benchmark *mcf* is present in the workload mix, and is absent in the other half.

The accurate way to measure the overall performance is to determine the average task turnaround times with *mcf* both present and absent in the workload mix, and then take the average of these turnaround times. One may assume it legitimate to take a shortcut and model a task arrival pattern in which *mcf* has half the arrival rate of the other benchmarks. However, the half-presence of a workload type can have a very different effect on performance than it being fully present half of the time and absent the next half.

Experimental results show that the best core design to employ in a homogeneous dual-core system for non-bursty traffic in which *mcf* has half the arrival rate of the other benchmarks is the customized core of *twolf*. However, the customized core of *vpr* is the best core design to employ for traffic in which *mcf* is in the workload mix half of the time and absent the other half.

Figure 6 compares the average turnaround time of these two dual-core homogeneous systems, under traffic in which *mcf* tasks are in the workload mix only half of the time. These results show that averaging out the workload mix results in the adoption of a design that is 12% suboptimal in the execution throughput it can provide.

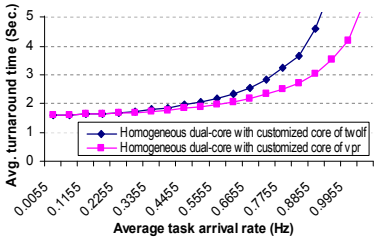


Figure 6: Irregular task arrival with *mcf* in the workload mix only half of the time.

4.2. Irregularity in the Traffic Load

The results of Section 3 show that the traffic load can alter the best cores to employ in a multi-core system. Thus, irregularity in the traffic load also needs to be accurately accounted for. Averaging out different task arrival rates can lead to misleading conclusions.

Consider a task arrival pattern in which tasks arrive at a high rate half of the time and arrive at a low rate in the other half. An accurate way to measure the overall performance is to determine the performance under both heavy and light traffic loads, and then take the average. One may assume it legitimate to take a shortcut and model a task arrival pattern in which the arrival rate is the arithmetic mean of that of the two traffic loads. However, medium-loaded traffic can have a very different effect on performance than it being lightly-loaded half of the time and heavily-loaded the other half.

For instance, experimental results show that the best combination of core designs to employ in a dual-core system, for full-bursty traffic that has an average task arrival rate of 0.88 tasks-per-second half of the time and an average task arrival rate of 0.0055 tasks-per-second in the other half, consists of the customized cores of *crafty* and *vpr*. However, the best dual-core design, for full-bursty traffic with an average task arrival rate of 0.44 (or the approximate average of 0.88 and 0.0055) tasks-per-second, is a homogeneous design employing the customized core of *vpr*.

Figure 7 compares the average turnaround time of these two dual-core systems under full-bursty traffic. The coarse dashed line highlights the average task turnaround time on the homogeneous system, and the fine dashed line highlights that of the heterogeneous system. The center of each arrowhead line marks the average turnaround time with an arrival pattern that is lightly-loaded half the time and heavily-loaded in the other half.

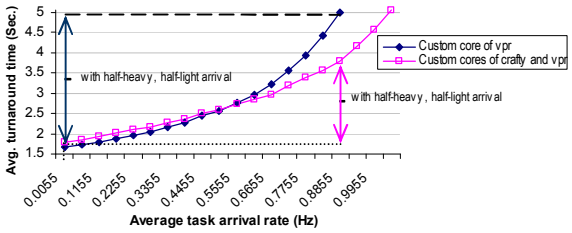


Figure 7: Average turnaround time with full-frequency double-task bursty traffic.

With an irregular arrival rate of 0.88 tasks-per-second half of the time and 0.0055 tasks-per-second the other half, the dual-core system with the customized cores of *crafty* and *vpr* attains an average task turnaround time of 2.79 seconds. On the other hand, the dual-core system with the customized core of *vpr* attains an average task turnaround time of 3.33 seconds. These results show that averaging out the task arrival rate can result in the adoption of a design solution that is 19% suboptimal in average turnaround time.

4.3. Irregular Burstyness

The results of Section 3 show that the burstyness of the task arrival pattern can alter the best cores to employ in a multi-core system. Averaging out different degrees of burstyness can potentially lead to misleading conclusions.

For instance, consider a task arrival pattern in which the traffic is non-bursty half of the time and full-bursty the other half. An accurate way to measure the overall performance is to determine the performance under both non-bursty and full-bursty traffic, and then take the average. One may assume it legiti-

mate to take a shortcut and model a task arrival pattern with half-frequency burstyness. However, half-frequency burstyness may have a very different effect on performance than being non-bursty half of the time and full-bursty the other half.

Experimental results show that the best combination of core designs to employ in a dual-core system for lightly-loaded traffic that is full-bursty half of the time and non-bursty the other half consists of the customized cores of *vpr* and *crafty*. However, the best dual-core design for lightly-loaded half-frequency bursty traffic is a homogeneous design consisting of the customized core of *twolf* (as in the results of Figure 2). Figure 8 compares the average turnaround time of these two dual-core systems under traffic that is non-bursty half of the time and then full-bursty the next half. These results show that averaging out the burstyness can lead to the adoption of a design that is over 10% suboptimal in the average task turnaround time it provides.

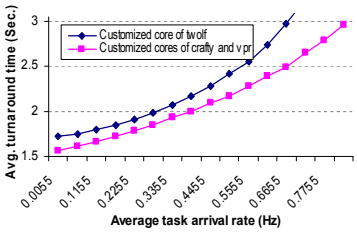


Figure 8: Irregular task arrival with full-bursty traffic half of the time and non-bursty traffic the other half.

5. Other Design Considerations

Sections 3 and 4 focused on the choice of core designs of dual-core systems. This section explores other aspects of the design of a multi-core system that interact with task arrival behavior, namely, the scheduling policy and number of cores. For both design aspects, the degree of heterogeneity is a prominent consideration.

5.1. Scheduling

The way in which tasks are scheduled and queued in a multi-core system can also impact how task arrival behavior affects overall performance. This is an issue of most importance in heterogeneous designs when the task traffic is bursty. The main concern in scheduling is whether tasks should be directed to only the core with the most suitable design or to the most suitable core available. These two approaches are illustrated in Figure 9.

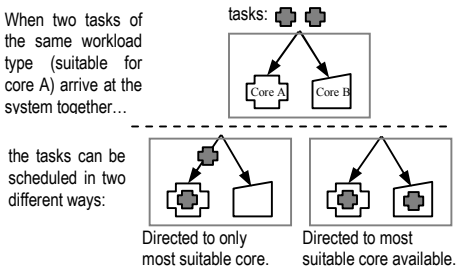


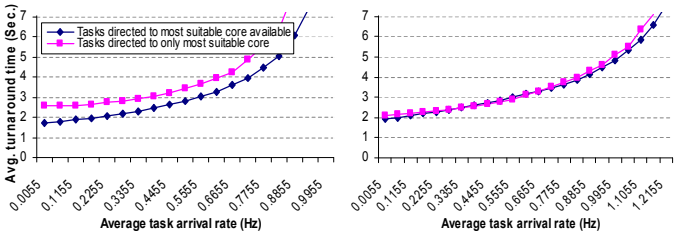
Figure 9: Two different approaches to task scheduling.

Depending on how well the tasks suitable for core A are executed on core B, it may or may not be beneficial to wait until core A is free. The approach taken regarding this issue can influence the best combination of core designs to employ in the system. In results presented thus far, the scheduling mechanism used was to direct tasks to the most suitable core available.

In Figure 10, the average task turnaround times for these two task scheduling approaches are compared under full-frequency double-task bursty traffic. Figure 10.a shows results for a dual-core design with the customized cores of *vpr* and *twolf*. These are the best two cores to employ in a dual-core system in order to maximize the arithmetic mean of performance in light and heavy arrival rates. We refer to this form of heterogeneity as *mild heterogeneity*, as the cores are not heavily biased towards disjoint subsets of the workload space. These results show that directing the tasks to only the most suitable core results in a 50% increase in the average task turnaround time in light traffic loads.

Figure 10.b shows results for the best combination of cores for heavily-loaded task arrival, namely, the customized cores of *parser* and *vpr*. These cores are heavily biased towards disjoint subsets of the workload space. We refer to this form of heterogeneity as *strong heterogeneity*. The results show only a small increase in the average task turnaround time when tasks are directed to only the most suitable core.

With mild heterogeneity, tasks are executed fast enough on the unsuitable cores that stalling them to be executed on only the most suitable cores is detrimental to performance. This, however, is not the case with strong heterogeneity, as tasks that are directed to unsuitable cores tie up the resources with inefficient work. Allowing tasks to migrate from core to core in mid-execution does not change this outcome.



(a) *(b)*

Figure 10: Average turnaround time in a dual-core with (a) mild heterogeneity, and (b) strong heterogeneity.

What can be concluded from these results is that, although mild heterogeneity may provide a good tradeoff between performance in light and heavy traffic loads, it is more sensitive to the task scheduling mechanism than strong heterogeneity.

5.2. The Number of Cores

Major factors that limit the number of cores employed in a system include die area and design complexity. More cores can provide greater overall execution throughput. Being able to sustain a high throughput is desirable as long as it does not come at excessive cost. However, in order to be able to make an accurate assessment of the performance-cost tradeoff, there is need for accurate measurement of performance. It is for this reason that the task arrival behavior can also influence the best number of cores to employ in a system.

Figure 11 compares how average turnaround time varies with traffic load in heterogeneous dual-core and three-core systems. The results are presented for non-bursty traffic. Each

design employs the very best combination of core designs for light traffic loads, determined through an exhaustive search. The three-core design employs the customized cores of *mcf*, *parser* and *twolf*, while the dual-core design employs the customized cores of *parser* and *vpr*.

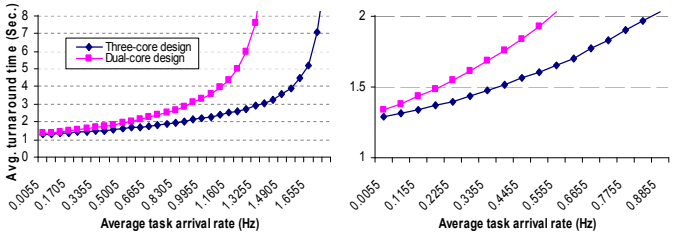


Figure 11: Average turnaround time in heterogeneous dual-core and three-core systems designed for lightly-loaded traffic.

These results show that the extra core of the three-core system provides minor performance advantage over the dual-core design in light traffic loads (~4%). That of the quad-core design (not shown here) over the three-core design is even less. This means that the microarchitectural diversity of strong heterogeneity in a dual-core system is sufficient to cover most of the workload diversity in SPEC2000 integer benchmarks. With more diverse task workload types, a greater number of cores may be able to provide greater performance enhancement in light traffic loads. The throughput advantage of the three-core design over the dual-core design is as expected.

Figure 12 compares the average task turnaround times of the best dual-core, three-core and quad-core designs for non-bursty heavily-loaded task arrival (1.4 tasks-per-second). Unsurprisingly, the system bandwidth increases with the number of cores. Interestingly, the best combination of cores for the three-core design was found to consist of the customized cores of *twolf*, *vpr* and *crafty*, while that of the quad-core design consists of the customized cores of *parser* and *twolf* along with two of *vpr*. This shows that as the number of cores increases, the best combination of cores for heavily-loaded traffic becomes less heterogeneous. Specifically, notice that the customized core of *mcf*, which is a very specialized design, was among the best three cores for lightly-loaded traffic, but not for heavily-loaded traffic.

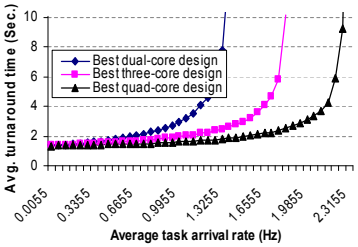


Figure 12: Average turnaround time in heterogeneous systems designed for heavily-loaded traffic.

In Figure 13, the average task turnaround time of the best three-core design for non-bursty heavily-loaded traffic (as seen in Figure 12), is superimposed on that of the best three-core design for non-bursty lightly-loaded traffic (as seen in Figure 11). Comparing these results with those of Figure 1 indicates that, as the number of cores increases, the performance profile of systems optimized for heavy traffic and that of systems optimized for light traffic become less similar. The performance difference in light traffic load increases, and so does the difference in their saturation points.

Also shown in Figure 13, is the average task turnaround time of the best homogeneous three-core design for light traffic loads, employing the customized core of *vpr*. The results show that, in heavier traffic loads, the homogeneous design actually outperforms the heterogeneous design optimized for light traffic loads, although it does ultimately saturate at a lower arrival rate. This is contrary to what was observed for dual-core systems, where the heterogeneous design outperformed the homogeneous design across the board (see Figure 1).

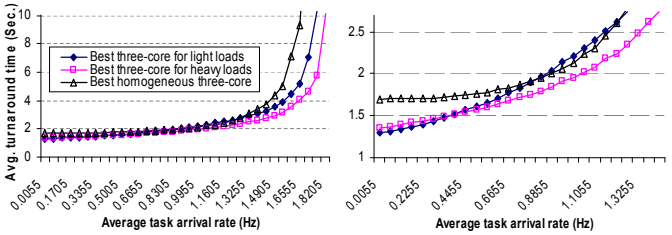


Figure 13: Average turnaround time in the best heterogeneous three-core systems for non-bursty heavy and light traffic loads.

The cause of this difference is the fact that a larger number of cores allows for the employment of core designs that are more application-specific and potentially less balanced in the number of workload types they are suitable for. For instance, the customized core design of *mcf* provides extremely poor performance for benchmarks other than *mcf*, placing burden on the other two cores. This imbalance is not an issue of concern in light traffic loads, but bares its toll in heavier-loaded traffic by causing inefficient utilization of the resources.

6. Practical Implications

An abstract conclusion that can be drawn from the results of this study is that accurately accounting for the pattern of task arrival is critical to fair performance evaluation of different design solutions for a processing system. However, the question is what this implies for practical performance evaluation and design space exploration. We address the implications in the following subsections.

6.1. Task Arrival Benchmarking

Traditionally, benchmark vendors have only provided the mix of instruction level behavior that typical processing systems observe (and, thus, need to perform well under). However, the empirical results presented here show that it is also important to accurately account for the typical pattern with which tasks of such different instruction level behavior arrive at the system. Specifically, averaging out irregularity in the task arrival pattern can lead to misleading conclusions regarding the relative performance of different design solutions.

Averaging out task arrival characteristics is analogous to using a microbenchmark that merely mimics the frequency of occurrence of different types of instructions, in the evaluation of single-thread performance. However, while extracting realistic instruction level behavior can be achieved by simulating the execution of representative applications, extracting realistic task arrival behavior can be more challenging. What is needed to extract representative task arrival patterns is an in-

frastructure for recording the pattern of task arrival and the instruction level behavior of each task in a real environment.

Some may argue that it is impractical to expect yet another facet of workload behavior to be accounted for, as performance evaluation is challenging as it is. We, however, argue that measuring the effect of task arrival behavior on overall performance requires considerably less simulation resources than that necessary for further accuracy in other aspects of the evaluation methodology (e.g., simulating larger regions of instruction-level code). Moreover, the aggregate effect of inaccurate assumptions about different aspects of the task arrival behavior can be considerably large (note that the effect of the different characteristics are evaluated individually in the results presented here). In addition, it is important to realize that had microarchitecture research been mostly based on qualitative evaluations, it would be legitimate to question the essentiality of further evaluation accuracy – but it requires limited pursuit of related literature to realize that that is not the case.

Note that the task arrival characteristics may also depend on architectural parameters. For instance, the degree of burstyness may be dependent on the number of cores in the system. Similarly, the task arrival rate may be dependent on the rate at which tasks are completed in the system (the performance of processing nodes). It is important that such dependencies also be reflected in the task arrival trace. In this case, the task arrival traces need to be in the form of parameterized stochastic models.

6.2. Robustness to Variation in the Task Arrival Behavior

Overall performance has traditionally been measured as some form of mathematical aggregation of performance across a number of different benchmarks of instruction level behavior. However, a design that provides the best performance from the standpoint of one form of aggregation (*e.g.*, harmonic mean) may have suboptimal performance from the standpoint of another (e.g., arithmetic mean).

Any form of aggregation can be considered to be representative of some form of task arrival behavior. Thus, the results of this study can be interpreted as indication of the fact that it is important to account for the robustness of a design solution to different forms of measuring overall performance (and the task arrival pattern each represents).

In fact, just as it is not correct to evaluate the performance of a design based on how well it performs under a specific instruction level benchmark, it is not correct to evaluate a system based on how well it performs under a single specific task arrival pattern – but should rather be evaluated across a range of task arrival patterns. Therefore, a number of different task arrival patterns that are representative of different environments need to be provided by benchmark vendors. In this manner, the performance of a processor design can then be documented for different task arrival patterns.

Finally, it is in light of the importance of robustness to variation in the task arrival pattern that the value of Core-Selectability [19] is fully exposed (see related work).

7. Related Work

The effect of message traffic on multi-computer performance has been widely studied in literature related to the evaluation of interconnection networks [1, 11]. However, the objective of such studies is to determine the best topology, switching techniques and routing algorithms for the interconnection network of a multi-computer system.

The effect of task arrival behavior on the design of the core(s) of processing systems has been of limited attention. The most relevant work is a study of heterogeneous multi-core design by Kumar et al. [2], in which the issue of job arrival rate in multi-core systems is examined. The work shows the benefit of single-ISA heterogeneity in providing greater execution throughput with limited die area, under the assumption that larger cores provide monotonically higher execution performance. In later work, the authors examine heterogeneity in the presence of non-monotonic difference in performance between the cores [3], and find the potential performance benefit to be larger.

Other studies that have focused on core design in multi-core systems do not account for the effect of the task arrival characteristics, and effectively resort to constraining assumptions about task arrival. For instance, Lee and Brooks study the effect of pipeline complexity on the performance-power tradeoff in chip multi-processor designs [4]. But the experimental methodology only accounts for tasks of different workload types arriving at the system in isolation from each other.

Strozek and Brooks propose a customization engine that is built on an automated method for determining an efficient architecture for a group of workloads [12]. This is used to design heterogeneous chip-multiprocessors that can execute a range of applications efficiently. However, in the experimental evaluations, only the execution time of individual benchmarks is accounted for, not the turnaround time. In other words, the effect of contention between tasks is not accounted for.

Winter and Albonesi study task scheduling algorithms in a chip multi-processor system that is rendered heterogeneous due to hard faults or wear-out [5]. Here, too, overall performance is measured by accounting for the performance of different workloads (specifically, the SPEC2000 benchmarks) when submitted to the system in isolation. Thus, it is tacitly assumed that tasks will be arriving at the system in a manner that no contention occurs among them.

The issue of how to direct tasks of different workload types to the most suitable core in a heterogeneous multi-core system is the most widely studied related work (see [6] for a list of related studies). In the empirical evaluations of this paper, we assume that the most suitable core for a task of a certain workload can be determined in an oracle manner.

Another related issue is cache fairness [7], which is concerned with how shared cache is divided between concurrently executed threads/tasks. Shared caches are not employed in the empirical evaluations of this study. This prevents the influence of potentially adverse cache sharing on evaluation results without the need to employ specific cache fairness mechanisms. The effect of the task arrival behavior may differ considerably in systems that employ shared cache and no cache-sharing mechanism, and concurrent threads may impact each other's performance.

Conte [18] previously used a simulated annealing exploration process to optimize the cost of a processor design by varying the number and pipeline depth of functional units, while not allowing the IPC to degrade lower than a fixed threshold. In this study, simulated annealing was also employed to tractably explore the large core design space and derive a palette of core designs. Here, however, the performance of a design solu-

tion was measured as a function of IPC and clock frequency, while accounting for the pipeline depth of different design units.

In a recent related study, we proposed Core-Selectability [19]. In this design approach, each core in a CMP is replaced with a cluster of multiple differently-designed cores, called a *node*, with the option to dynamically select which core to actively employ. Only one core in each node need be actively employed at each instance of time, as the purpose of the different cores is to provide microarchitectural diversity, rather than concurrency. As part of the empirical evaluation of that study, we explored the performance of multi-core designs under bursty and non-bursty task arrival patterns. Core-selectability enables dynamic configuration of a CMP to achieve different homogeneous or heterogeneous designs, rendering the system robust to different task arrival behavior.

8. Discussion: The Shortcomings of this Study

A simplification used in the experimental methodology of this study is that the amount of time a task occupies a core is determined by only the instruction-level behavior of the task (i.e., which benchmark) and the design of the core it will be executed on (i.e., which customized core design is used). However, in a more realistic setting, the need for access to shared resources such as the buses and shared cache can create contention between concurrent threads [14]. More accurate evaluation, that can model such contention, requires a more detailed simulation environment.

It has been shown that there is minimal inter-thread contention between integer threads (*e.g.*, [13]). However, in studying the impact of task arrival behavior with broader workload diversity, that includes floating-point benchmarks, such interplay will need to be accounted for. Accounting for the impact of this interplay will be more challenging as it requires some form of modeling the contention between tasks of different workload types and its influence on execution performance.

In commercial processor design, the overall merit of a design solution is usually dependent on not only measures of performance, but also factors such as design effort and power consumption. Although we do not quantitatively account for such factors in this study, they deserve much attention. For instance, in addition to the observations expressed here regarding performance, the greater design effort associated with heterogeneous designs and the greater power consumption of wide processor designs need to be accounted for. These factors can be more difficult to accurately quantify outside of an actual development environment.

9. Conclusion

The bottom line is that the pattern of task arrival can influence the best core design(s) to employ in the system, and the optimality of a given design solution.

The results of this study show that accounting for an unrepresentative task arrival rate can result in up to 11% suboptimality in the average task turnaround time. And accounting for an unrepresentative degree of burstyness in the task arrival pattern can result in up to 21% suboptimality in the average task turnaround time.

It is also shown that, when there is irregularity in the characteristics of the task arrival pattern, it is not accurate to average out the different characteristics. For instance, averaging out the degree of presence of different workloads in the workload mix can result in up to 12% suboptimality in the multi-core design determined as the best design solution. This means that not only are the parametric rates that characterize the task arrival behavior of crucial importance, but also the pattern with which they vary. Therefore, it is important that benchmark vendors provide not only the mix of typical instruction level behavior, but also representative arrival patterns for tasks with such instruction level behavior.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback. This research was supported in part by NSF grant No. CCF-0811707, and funding from Intel and IBM. Note that any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] W. J. Dally, B. Towles, "Principles and Practices of Interconnection Networks", *Morgan Kaufmann*, 2004.

[2] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, K. Farkas. "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance". *In 31st International Symposium on Computer Architecture, ISCA-31*, 2004.

[3] R. Kumar, D. Tullsen, N. Jouppi. "Core Architecture Optimization for Heterogeneous Chip Multiprocessors". *International Conference on Parallel Architectures and Compilation Techniques, PACT*, 2006.

[4] B. C. Lee and D. Brooks. "Effects of Pipeline Complexity on SMT/CMP Power-Performance Efficiency," *Workshop on Complexity Effective Design 2005 (WCED2005, held in conjunction with ISCA-32)*, 2005.

[5] J.A. Winter and D.H. Albonesi, "Scheduling Algorithms for Unpredictably Heterogeneous CMP Architectures", *38th International Conference on Dependable Systems and Networks*, 2008.

[6] M. Becchi, P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures", *Conf. Computing Frontiers*, 2006.

[7] S. Kim, D. Chandra, Y. Solihin, "Fair Caching on a Chip Multiprocessor Architecture", *Proc. of IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=ac2)*, 2004.

[9] Advanced Micro Devices, "AMD Opteron™ Processor Product Data Sheet", 2007.

[10] K. Chakraborty, P. M. Wells, G. S. Sohi, "Computation spreading: employing hardware migration to specialize CMP cores on-the-fly", *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[11] J. Balfour, W.J. Dally, "Design tradeoffs for tiled CMP on-chip networks", *Proc. of the 20th Int'l Conf. on Supercomputing*, 2006.

[12] L. Strozek, D Brooks, "Efficient architectures through application clustering and architectural heterogeneity", *International Conference on Compilers, Architecture, and Synthesis of Embedded Systems, CASES*, 2006.

[13] Y. Xie, Gabriel H. Loh, "Dynamic Classification of Program Memory Behaviors in CMPs", *In the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2008.

[14] D. Chandra, F. Guo, S. Kim, Y. Solihin, "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor

Architecture", *Proc. of the11th International Symposium on High Performance Computer Architecture (HPCA)*, 2005.
[15] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model", *IEEE Journal of Solid-State Circuits*, 31(5):677–688, 1996.
[16] E. Larson, S. Chatterjee, T. Austin, "The MASE Microarchitecture Simulation Environment," *Int'l Symposium on Performance Analysis of Systems and Software*, 2001.
[17] S. Thoziyoor, N. Muralimanohar, J. Ahn, N. Jouppi, "CACTI 5.1", *Technical Report HPL-2008-20, HP Labs*, 2008.
[18] T. M. Conte, "Systematic Computer Architecture Prototyping," *Ph.D. Thesis, University of Illinois*, 1992.
[19] H. H. Najaf-abadi, N. K. Choudhary, E. Rotenberg, "Core-Selectability in Chip Multiprocessors", *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT),* 2009.