

Configurational Workload Characterization

Hashem H. Najaf-abadi

NC State University
hhashem@ece.ncsu.edu

Eric Rotenberg

NC State University
ericro@ece.ncsu.edu

ABSTRACT

Although the best processor design for executing a specific workload does depend on the characteristics of the workload, it can not be determined without factoring-in the effect of the interdependencies between different architectural subcomponents. Consequently, workload characteristics alone do not provide accurate indication of which workloads can perform close-to-optimal on the same architectural configuration.

The primary goal of this paper is to demonstrate that, in the design of a heterogeneous CMP, reducing the set of essential benchmarks based on relative similarity in raw workload behavior may direct the design process towards options that result in sub-optimality of the ultimate design. It is shown that the design parameters of the customized processor configurations, what we refer to as the *configurational characteristics*, can yield a more accurate indication of the best way to partition the workload space for the cores of a heterogeneous system to be customized to.

In order to automate the extraction of the configurational-characteristics of workloads, a design exploration tool based on the SimpleScalar timing simulator and the CACTI modeling tool is presented. Results from this tool are used to display how a systematic methodology can be employed to determine the optimal set of core configurations for a heterogeneous CMP under different design objectives. In addition, it is shown that reducing the set of workloads based on even a single widely documented benchmark similarity (between *bzip* and *gzip*) can lead to a slowdown in the overall performance of a heterogeneous-CMP design.

Categories and Subject Descriptors

C.1.1 [Single Data Stream Architectures]: *RISC/CISC, VLIW architectures, Single-instruction-stream*

Keywords

single-thread performance, customization, heterogeneous CMP, design exploration, workload characterization

1. INTRODUCTION

Computer architecture research has effectively become the art of exposing and exploiting localities in workload behavior. Therefore, characterizing and understanding the behavior of common workloads can be considered an essential facet of this research field. The commercial advent of the chip multiprocessor (CMP) in recent years has added to the value of understanding the similarities (and differences) between workloads – by increasing the viability of *heterogeneity*. In the context of

this paper, a heterogeneous-CMP is a CMP in which the cores are architected differently, each sacrificing general-purpose performance for better workload-specific performance, so as to achieve in *commune* an overall performance that would not be attainable otherwise.

A key challenge in the design of such a system is the question of how the workload space should be partitioned for different cores to be customized to. It is this design challenge – which we refer to as the *communal customization* problem – that adds to the importance of understanding the nature of workload similarity. Communal customization is akin to the more established notion of *workload subsetting* [27]. However, the objective of subsetting is to identify workloads that, in the space of workload characteristics, have relatively less Euclidian distance between each other. The assumption is that such workloads are affected similarly by the architectural configuration – thus providing potential for less simulation-based evaluation cost. In contrast, communal customization involves the identification of workloads that can attain *close-to-optimal* performance with the same architectural configuration.

The major argument presented in this paper is that, in the design of a heterogeneous CMP, viewing workload subsetting as a substitute for communal customization – or even as a preliminary step to reduce exploration complexity – may direct the ultimate design towards suboptimality. The fundamental reason for this is that there are complex interdependencies between the different units of a processor design. These interdependencies cause different components of the optimal processor configuration to be affected by the workload behavior *as a whole* (rather than distinct characteristics). In addition, these interdependencies are influenced by the physical properties of the underlying technology and are thus not reflected in the workload characteristics alone.

1.1. Subcomponent Interdependence in Superscalar Processor Design

Workload characteristics that pertain to functionally independent subcomponents of a processor design are commonly viewed as independent gauges of the optimal configuration of those subcomponents. Metrics for the biasness of branches, memory access localities and the distribution of dependent instructions are microarchitecture-independent examples of such characteristics that respectively relate to the branch predictor, data caches and the scheduling unit. However, the optimal configuration of each of these units is influenced by the clock period of the system, while the optimal clock period itself depends on the dynamics of how the different units scale. Thus, in the

search for an optimal design, the unified clock period intertwines the different subcomponents.

For instance it is ill-conceived to consider similar optimal L1 cache configurations for two workloads based solely on the fact that they have similar spatial/temporal locality of memory access and working-set size. This is because other characteristics of the workloads (such as the control-flow behavior) affect the best configuration for other aspects of the design (such as the processor width and pipeline depth), including the clock period. The clock period determines the number of cycles necessary for accessing a cache unit with a particular configuration, which in turn influences the attainable IPC. Also the physical properties of the technology in which the system is implemented determine how the different design aspects scale relative to each other. These properties are not reflected in the raw characteristics of a workload. Moreover, it is not correct to assume that these interdependencies are of mere second-order effect as they involve issues as critical as the latency between back-to-back dependent instructions.

As an illustrative example, consider three workloads α , β , and γ that have equal *importance weights*. The workloads have mostly similar characteristics, other than workloads β and γ having much larger working-sets than α , and γ having greater branch biasness and less dense dependence chains than α and β . Considering the Kiviat graphs of these workloads to be those displayed in Figure 1, it is evident that, from the standpoint of raw workload characteristics, α and β are *relatively more similar* – as they differ only in the size of their working-set. Thus, a naive observation may conclude that customizing one of the two cores for workloads α and β , and the other for workload γ , will result in the best overall single-thread performance.

However, optimal performance with workload β may require a large L1 cache, due to its larger working set, and any compromise may severely degrade performance, while any increase in the L1 cache size may severely degrade the performance of workload α due to the high frequency of loads. Although workload γ also has a large working set, due its less dense dependence chains and higher branch predictability, it will from an IPC standpoint be able to better tolerate cache misses. Therefore, depending on how the different units scale in the given technology, workload γ may be more suitable than β for execution on a configuration that is also suitable for α .

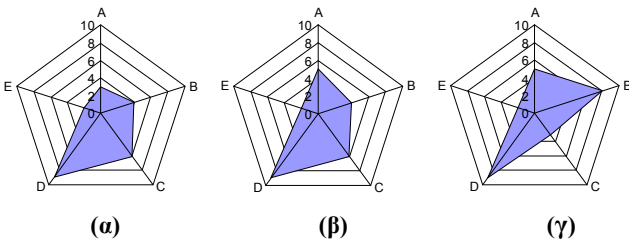


Figure 1: Kiviat graphs of three sample workloads for the architecture-independent characteristics of: A) Working-set size, B) Branch predictability, C) Density of dependence chains, D) Frequency of loads, E) Frequency of conditional branches – all normalized to a scale of 0~10.

1.2. Configurational Characterization

For the purpose of communal customization, an effective form of workload characterization would more transparently reflect how relatively suited different workloads are for being executed on the same architecture – rather than how relatively similar their raw characteristics are. It is intuitive that the characteristics of the best architectural configuration for a workload possess this quality. Such characterization allows for direct and accurate measurement of how well different workloads perform on each other’s customized architectures. Moreover, the best architectural configuration for a workload encapsulates the effect of the different facets of raw workload behavior on the optimal configuration of the different architectural units under the constraints imposed by the interdependencies between them.

1.2.1. Practicality and efficiency

It is undoubtedly more costly to determine the best architectural configuration for a workload – which requires numerous cycle-accurate simulations of the execution of code – than it is to extract its conventional workload characteristics. However, knowing the best architectural configurations for individual workloads enables the best combination of core configurations to become extractable through a *systematic* task of reducing the set of workloads/configurations.

Moreover, determining the best architectural configurations for workloads can be performed priori to the design phase that is critical for time-to-market demands, as the physical properties of future generation technologies are often predictable before commercial feasibility.

1.2.2. Broader implications

Although we frame our discussion around the design of heterogeneous CMPs, the notion of characterizing workloads based on their customized architectural configurations can have broader implications for processor architecture research in general.

The solidity of conventional methodologies in the evaluation of microarchitectural techniques has long been scrutinized [35]. This is in part due to the fact that when any component of the workload characteristics or architectural configuration change the balance between them changes. In actuality, it is this change that should be considered representative of the true effect of a microarchitectural technique – rather than the ‘speedup’ it attains over some baseline architecture.

In other words, for the proper evaluation of a novel architectural technique what should be determined is how the technique influences the balance between the different architectural units and the workload. The customized processor configuration of a workload can be viewed as an *atomic* entity representing this balance. Standardizing the customized configurations of popular benchmark suites can pave the way to a more standard evaluation methodology – the need for which is also stressed in [35].

Nevertheless, we believe that this issue is of less practical significance in the domain of general-purpose processors as such

designs are constrained on multiple fronts by widely diverse workloads. It is when the workload space is to be split among different processing cores that this issue gains significance.

1.3. Outline and Contributions

In this paper, a simulated-annealing exploration process is employed to explore the design space of the superscalar processor for each of the C integer benchmarks from the SPEC2000 suite, and determine their configurational characteristics.

The contributions of this paper are thus threefold: 1) Introducing a superscalar design-space exploration framework based on the popular SimpleScalar simulator [25] and CACTI modeling tool [36]. 2) Determining the configurational characteristics of the integer SPEC benchmarks in a specific technology. 3) Illustrating how configurational workload characterization can enable the coherent and systematic extraction of the best cores to employ in a heterogeneous CMP.

The next section explores related work. Section 3 describes the functionality of our design-space exploration process. The exploration methodology and results are presented in Section 4. These results are used in Section 5 to evaluate different approaches to conducting communal customization. Section 6 concludes the paper.

2. MOTIVATION AND RELATED WORK

2.1. Workload Subsetting

Numerous studies have analyzed different aspects of commercial benchmark suites [1, 2, 3, 4]. Other studies have focused on understanding the similarities between different benchmarks [6, 7] with the objective of reducing the amount of simulation required to evaluate architectural innovations. For instance, Joshi et al. propose a methodology for measuring the similarities between programs based on microarchitecture-independent characteristics [8]. Hoste et al. use workload similarity to predict application performance [11]. Vandierendonck et al., rank SPEC2000 integer benchmarks based on whether they exhibited different speedups on different machines, and use this as a guideline for identifying similar workloads [29].

Yi et al. present a thorough summary of workload subsetting approaches [27], and propose a statistically rigorous approach based on the Plackett-Burman design [32]. A similar technique is proposed by Dujmovic et al [33]. In these approaches, the execution of different workloads is evaluated on different processor designs in order to expose their architectural bottlenecks – which are considered to be indicative of similarity. The assumption on which these approaches are based is that interaction between the different workload characteristics (and their corresponding architectural units) is negligible. However, under realistic design constraints the pipelined nature of traditional processor design brings about interdependence in the configuration of seemingly uncorrelated architectural units.

The uniform clock period determines the pipeline depth and the slack observed in different stages. Pipeline slack can greatly impact the performance of non-linear pipelines such

as the superscalar processor. Figure 2 illustrates how the unified clock can affect the optimal sizing of the issue queue and L1 cache. In each scenario the solid lines represent the delay of the issue queue, the dashed lines represent the access delay of the L1 cache, and the scales at the bottom represent the clock cycles. The propagation delay of the issue queue and the delay of the L1 cache are based on a representative sizing of these units.

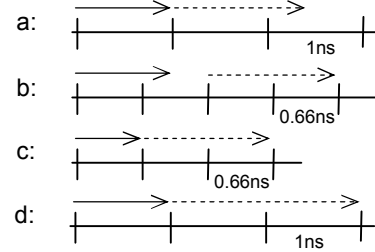


Figure 2: Illustrative scenarios for the design of a processor with different clock periods, issue queue and L1 cache sizes.

In scenario **a**, access to the L1 cache observes considerable slack. The overall slack can be reduced by changing the clock period, as displayed in scenario **b**. Doing so, however, deepens the pipeline which may improve or degrade overall performance. So is the case in scenario **c**, where the slack is further reduced by downsizing the issue queue size. Finally in scenario **d**, instead of scaling down the clock period, the size of the L1 cache is increased to make full benefit of the available two cycles. Depending on the working set of the application this extra L1 cache capacity may not be of any value, and scenario **b** or **c** may be of better overall performance.

2.2. The Heterogeneous CMP

Kumar et al. propose the single-ISA heterogeneous CMP as an approach to enhance the throughput of multithreaded workloads [14]. In follow-up work they show that the best way to design a heterogeneous CMP is actually to tune each individual core for “a class of applications with *common characteristics*” [10]. The focus of our paper inherently revolves around the question of how workloads should be ‘characterized’ for this purpose and what the criterion for ‘commonality’ should be. To the best of our knowledge, the only prior studies that touch upon the issue of designing the cores of a heterogeneous CMP is the aforementioned work by Kumar et al. [10] and a more recent study by Lee and Brooks [37].

Kumar et al. reduce the set of benchmarks based on similarity in workload characteristics. By doing so, and limiting the architectural diversity of the cores, an exhaustive search of different core-combinations across different groupings of workloads is made feasible. This approach is ad hoc in that there is no solid justification (other than exploration cost) to reduce the benchmark set to a specific size. Lee and Brooks use regression modeling to enable fast exploration of the microarchitectural design space. Then, through an iterative process (K-means clustering) *centroids* are identified for the customized architectures. The closest centroid to the customized architecture of each benchmark is assigned as the *compromise archi-*

ecture of that benchmark. This approach is also ad hoc in that its outcome is highly dependent on how the different architectural parameters are normalized and weighed. It is however the most related approach to that proposed here, and addresses the major focus of this paper.

Therefore, as illustrated in Figure 3, there are in general two broad approaches to communal customization. One is to initially extract a small enough subset of the essential workloads for it to be feasible to conduct an exhaustive exploration of the workload-architecture combinations. The other is to initially determine the optimal architectural configuration for each considered application and then reduce the set of resultant architectures to a representative subset. With the availability of the optimal configuration for each workload the true *closeness* between them can systematically and accurately be measured and the most representative subset determined.

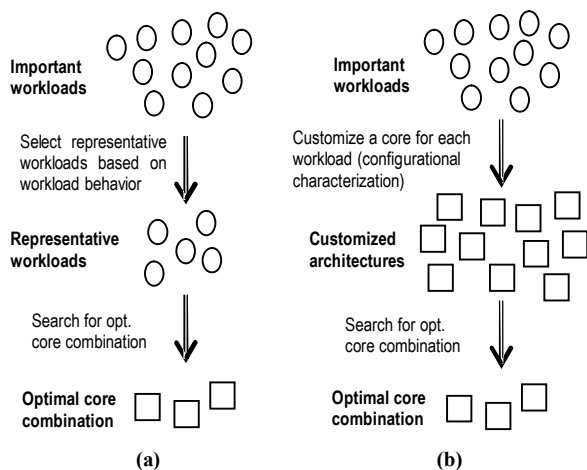


Figure 3: Two general approaches to identifying the optimal core combination for a heterogeneous CMP: (a) An exhaustive search – which for feasibility requires selection of representative workloads, (b) Determining the customized architectures of applications and then reducing the set of architectures.

2.3. Automatic Design Space Exploration

Due to the sheer size of the superscalar processor design space, determining the optimal architectural configuration for a workload is itself a demanding task. A wide spectrum of studies has focused on developing tools to enable efficient exploration of the design space with different degrees of accuracy and architectural variability [12, 15, 16, 17].

At one end of the spectrum are approaches that are more concerned with specific design details. For instance, AMPLE [12] is a wire-driven microarchitectural design space exploration framework in which the size of different units and the floor-planning are customized to workload behavior. Initial microarchitecture parameters for the initial search point of each application are determined based on the application’s characteristics. The clock period is not among the customizable design parameters and different design units are not pipelinable (it only weighs the power benefit of downsizing against its performance degradation).

Due to the non-discrete nature of the clock period, considering it as a customizable parameter considerably increases the processor design space. It is for this reason that prior design exploration studies either limit the design space to a set of pre-designed configurations [10, 33] or consider a fixed clock period across variability in other architectural parameters [12]. Both effectively diminish the true performance potential of customization (and heterogeneity).

On the other end of the spectrum are approaches that are more concerned with the speed of performance evaluation (and exploration). For instance, Lee and Brooks introduce a non-linear microarchitectural regression model, and propose its use to enable fast exploration of the processor design space [37]. While pipeline depth is among the customizable design parameters, it is employed as a speed-power factor and not necessarily a parameter influencing the sizing of different units in a balanced pipeline design.

However, the major issue with such mathematical models is the space in which their accuracy is verified. In general, misleading conclusions may be drawn on the accuracy of a model if the evaluation is conducted in a distorted space. This can occur when the evaluated space is a subset of the actual space due to the absence of variability in certain parameters, or is its superset due to the absence of the enforcement of certain restrictions – or a combination of both factors. The problem in evaluating superscalar regression models is that accounting for independent variability in the pipeline depth of different units and enforcing the constraints imposed by a global clock period results in a design space that can not be concisely delineated (its shape and bounds specified) in parametric form.

Therefore, there is more difficulty in evaluating the accuracy of such models than meets the eye. However, inaccuracy in this area can lead to incorrect conclusions when the model is employed for design-space exploration, principle component analysis or clustering. The advocates of using regression models for design-space exploration argue that employing full-blown simulation is too time consuming and costly. We however, argue that the process is highly parallelizable and that with sufficient resources (which are typically available in large development groups) a design space exploration with reasonable rigor should be achievable in a matter of days. For this reason we believe that basing the exploration process on difficult-to-verify regression models serves little benefit to such a study.

3. XP-SCALAR: A SUPERSCALAR DESIGN-EXPLORATION FRAMEWORK

A light-weight superscalar design-space exploration framework named *xp-scalar* has been developed. The major component of the framework consists of a tool that employs a simulated annealing process to find the best superscalar architectural configuration for executing a specific workload. Also available is a tool for visualizing the performance of the benchmarks on each other’s customized configurations, which eases the identification of discrepancies and can help expedite

the exploration process. The source code of the framework and directions for use are accessible at <http://www4.ncsu.edu/~hhashem/xpscalar.htm>. The tool employs the SimpleScalar v.4 simulator to perform execution-driven simulations, and the CACTI model to approximate the access latency of the different units of the superscalar processor. Both SimpleScalar and CACTI can be employed with or without any modification, as long as the format of the input and output does not change.

In each iteration, either the clock period is varied, and the size of the issue queue, register-file/ROB, load-store queue, L1 and L2 caches, and processor width adjusted to make their access times fit within the number of pipeline stages assigned to them, or the number of pipeline stages of a unit is varied and its configuration appropriately adjusted. Table 1 displays the manner in which the tool uses the CACTI output parameters to estimate the access latency of different architectural units. Note that the issue queue delay consists of wake-up (an associative component) and select (a direct mapped component) delays.

After the different units are scaled to fit the product of the clock period and their pipeline depth, minus the aggregate latch latency, the benchmark is executed on the *sim-mase* simulator (from the SimpleScalar toolset) configured correspondingly. If a configuration executes the workload with greater IPT (Instructions per Time-unit) than the best observed until that point in the exploration, the configuration is recorded as the new optimal solution. When a configuration is reached for which the IPT is less than half that of the optimal configuration, the exploration process rolls back to the optimal solution and is continued.

In this study, power and die area are not considered in the evaluation process, and optimum design is concerned only with performance. It is however found that under realistic assumptions for the access latency to different superscalar sub-components, these aspects of the optimum architectural configuration remain within acceptable limits. Extending the tool to conduct exploration based on a metric that represents some combination of performance, power and die area should not be exceptionally difficult.

4. EXPLORATION RESULTS

4.1. Methodology

The workloads evaluated are the C integer benchmarks from the SPEC2000 suite compiled for the PISA instruction-set. The exploration process was conducted on a quad-core hyper-threaded blade for a period of three weeks. During this period, each workload was also executed on the customized architectures of the other workloads. If a workload was found to per-

form better on some other workload’s optimal configuration, that configuration would replace its own configuration in order to expedite the exploration process. The evaluation of each architectural configuration during the exploration process consists of the execution of a 100-million instruction Simpoint [34]. A considerably large number of such evaluations need to be conducted for each benchmark in order for the evolutionary process to approach the optimum design. Therefore, in the initial stages of the exploration, each evaluation was limited to the first 10 million instructions.

Three microarchitecture-independent technology-dependent factors were found to be influential on the ultimate customized configurations attained for the benchmarks. Table 2 displays the values considered for these parameters in this study. The *memory access latency* determines the amount of time required to access the main memory, i.e., the latency of a load that misses in all cache levels. The *front-end latency* is the amount of time required for an instruction to be retrieved, decoded and renamed, i.e., the extra branch misprediction penalty in the SimpleScalar simulator. CACTI does not produce accurate modeling for block sizes smaller than 8 bytes. Therefore, we consider this lower bound as the width of the issue queue entries. Another important design constant is the latch latency which affects the optimum pipeline depth of different subcomponents. These values are in general accordance with common processor designs.

Table 3 displays the initial architectural configuration employed across all benchmarks. Note that only the access latencies (in clock cycles) of the caches are indicated. This is because the cache configurations are randomly varied to fit the product of the clock period and number of access cycles during the first iteration of the exploration process if the default does not fit.

4.2. Customization Results

Table 4 displays the characteristics of the optimum architectural configuration for each of the considered benchmarks. The optimum processor width is observed to vary between 3 and 7. The optimum ROB size varies between 64 and 1024. The optimum clock frequency varies between 1.72 GHz and 5.2 GHz. The optimum size for the L1 cache capacity is in the range of 8K to 256K, while that of the L2 cache is in the range of 128K to 4M bytes.

Please see the xp-scalar website for more up-to-date results from further evolution of exploration process, and the effect of improvements in the accuracy of modeling the latencies of different units – a process that with feedback from the community will be on-going. The major conclusions drawn here are unlikely to be annulled with change in the modeling.

Table 1. The CACTI parameters used to determine the access latency of various units based on architectural parameters.

Arch. Unit	Line size	Associativity	No. of sets	No. read ports	No. write ports	Used component of CACTI output
L1 data cache	line size of cache	assoc. of cache	no. of sets of cache	2	2	Access time
L2 data cache	line size of cache	assoc. of cache	no. of sets of cache	2	2	Access time
wakeup-select	8 bytes	fully associative	2 x size of issue queue	Issue width	0	Tag comparison
	8 bytes	direct mapped	size of issue queue	Issue width	0	+ Total data-path without output driver
reg. file (ROB)	8 bytes	direct mapped	size of ROB	2 x issue width	issue width	Access time
LSQ	8 bytes	fully associative	size of LSQ	2	2	Total data-path without output driver

Table 2. Fixed design parameters across all configurations.

memory access latency	50ns
front-end latency	2ns
bit-width of IQ entries	64
latch latency	0.03ns

Table 3. Initial configuration used across all benchmarks.

No. of cycles for memory access	172
No. of pipeline stages of front-end	6
Dispatch, issue, and commit width	3
ROB size	128
Issue queue size	64
Min. lat. for awakening of dep. instr.	1
Pipeline depth of Scheduler/Reg-file	1
Clock period (ns)	0.33
L1D access latency	4
L2D access latency	12
Load-store queue size	64
Pipeline depth of LSQ	2

5. COMMUNAL CUSTOMIZATION

5.1. Cross-Configuration Performance

Once a customized architectural configuration for each benchmark has been established, the performance of each benchmark on the configuration of other benchmarks can be determined. This allows for the performance difference between different architectures to be extracted, and the architectures that provide close-to-optimal performance across numerous benchmarks identified. The best core configurations are not necessarily among the workload-customized cores. However, the broader the workload diversity, the better coverage there will be of the design space.

Table 5 displays the IPT of each of the SPEC2000 benchmarks executed on the optimal architecture of all the other benchmarks. From these results, the percentage slowdown of each benchmark when executed on the architectures of other benchmarks over the performance on its own architecture can be

extracted. The importance of carefully choosing the cores of a heterogeneous CMP is evident in these results with up to ~50% slowdown (for *mcf*) observed for the execution of benchmarks on the customized architecture of other benchmark.

5.2. The Best Core Combination

Before the best set of core configurations to employ in a heterogeneous system can be identified, the design-goal needs to be determined and a figure of merit that represents that design goal.

If the goal is to minimize the total execution time of a set of *consecutive* benchmarks, as is customary in single-core microarchitecture research, a representative figure of merit is the harmonic-mean of the performance of each benchmark when run on the most suitable core available for it. Such a design goal however does not account for core-contention. It may thus cause preference towards adoption of configurations that perform extremely well with a few benchmarks without considering the burden this may place on other more general configurations. If the objective is to increase the average performance with which an arbitrary benchmark from a set of benchmarks will be executed when submitted in isolation to the system, a representative figure of merit is the average performance of each benchmark on its most suitable core available.

A more real-world design goal is to minimize the total execution time of a set of benchmarks that can be executed concurrently on separate cores (if available). A representative figure of merit for this can be attained by first dividing the performance of each benchmark when run on the most suitable core available for it, by the number of benchmarks with which it shares that core, and then taking the harmonic mean. We refer to this as the *contention-weighted harmonic-mean*.

Table 4. The customized architectural configurations for the SPEC2000 benchmarks.

	bzip	crafty	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
No. of cycles for memory access	112	321	173	186	198	120	198	321	172	213	172
No. of pipeline stage of the front-end	4	12	6	7	7	4	7	12	6	8	6
Dispatch, issue, and commit width	5	8	4	4	4	3	4	5	5	7	5
ROB size	512	64	128	256	64	1024	512	256	512	512	256
Issue queue size	64	32	32	32	32	64	32	32	64	32	64
Min. lat. for awakening of dep. Instr.	0	3	1	1	1	0	1	3	1	2	1
Pipeline depth of Scheduler/Reg-file	1	3	1	2	1	1	2	4	2	4	2
Clock period	0.49	0.19	0.33	0.31	0.29	0.45	0.29	0.19	0.33	0.27	0.3
L1D associativity	2	1	1	1	1	2	1	1	8	4	2
L1D block-size	32	8	8	8	128	128	64	8	64	32	32
L1D no. of sets	1k	16k	2k	32k	256	1k	2k	2k	128	1k	128
L1D access latency	2	5	2	4	3	5	3	3	3	5	2
L2D associativity	4	16	4	8	1	4	8	16	4	16	8
L2D block-size	64	64	256	64	128	128	512	64	128	128	128
L2d no. of sets	8k	128	128	1k	4k	8k	32	128	2k	128	1k
L2D access latency	15	7	4	6	5	27	12	7	12	6	12
LS-queue size	128	64	256	256	128	64	256	128	256	256	64

Table 5. The performance of each benchmark (rows) on the customized architectures (columns) of other benchmarks.

	bzip	crafty	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
bzip	3.15	2.02	1.73	2.41	2.11	2.56	2.09	2.03	3.05	2.24	2.95
crafty	0.78	2.31	1.15	2.11	1.91	0.48	1.97	2.06	1.29	2.12	1.30
gap	1.39	2.75	3.02	2.60	2.92	0.89	2.89	2.79	2.00	2.47	2.05
gcc	1.17	2.17	1.42	2.27	2.03	0.75	2.02	1.63	1.79	2.06	1.80
gzip	1.78	2.56	2.02	2.88	3.13	1.28	3.01	2.14	2.39	2.57	2.37
mcf	0.74	0.40	0.30	0.45	0.29	0.93	0.32	0.41	0.52	0.42	0.52
parser	1.86	2.11	2.19	2.08	2.47	1.32	2.62	1.86	2.39	2.15	2.30
perl	0.85	2.02	0.90	1.81	1.67	0.54	1.65	2.07	1.32	1.81	1.30
twolf	1.65	0.98	0.81	1.26	0.88	1.18	1.10	0.91	1.83	1.16	1.77
vortex	1.68	2.98	2.55	3.09	2.91	1.07	3.41	2.78	2.61	3.43	2.54
vpr	1.56	1.33	1.13	1.72	1.09	1.05	1.36	1.29	2.00	1.51	2.09

Table 6 displays the best set of cores to employ for different core-counts, in order to maximize the harmonic-mean, average and contention-weighted harmonic-mean of the IPT (respectively represented by *har*, *avg* and *cw-har*) of the integer SPEC2000 benchmarks. These results were attained from the results of Table 5, by conducting a complete search of all possible core-combinations. A tool for automating this task is also part of the xp-scalar framework. These results show that a well-designed two-core heterogeneous CMP, can provide ~10% and ~20% speedup in average and harmonic-mean IPT respectively, over the best single-core configuration.

Table 6. The best core combinations and their performance.

	customized core(s)	avg. IPT	har. IPT
best config for avg. & har. IPT	gcc	2.06	1.57
2 best configs for avg. IPT	parser, twolf	2.27	1.76
2 best configs for har. IPT	gcc, mcf	2.12	1.88
2 best configs for cw-har. IPT	bzip, crafty	2.18	1.87
3 best configs for avg. IPT	crafty, parser, twolf	2.35	1.82
3 best configs for har. IPT	crafty, mcf, twolf	2.27	2.05
4 best configs for avg. & har. IPT	crafty, mcf, parser, twolf	2.32	2.08
each benchmark on its own customized architecture	-	2.38	2.12

Figure 4 displays the single-thread performance attainable from executing the benchmarks when the number of core configurations available is limited. These results show that the choice of available configurations can greatly impact individual benchmark performance. For instance, the benchmarks *twolf* and *parser* displays around 40% and 25% speedup respectively over the best single configuration when the best two configurations for average IPT are employed. Similarly, the benchmark *mcf* attains close to 2x speedup over the best single configuration when the best two cores for harmonic mean performance are available. However, the availability of the customized architectural configuration of *mcf* provides hardly any benefit for the other benchmarks (only *bzip* attains a slight performance enhancement). This shows how other parameters, such as the *importance-weight* of benchmarks, can influence the best core combination. For instance if *mcf* were to have a considerably lower importance-weight than the other benchmarks, the best two configurations for harmonic-mean performance would potentially be different.

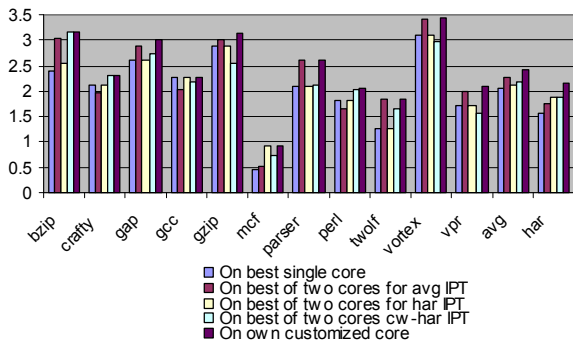


Figure 4: The IPT of the execution of different benchmarks on the best available core, with different configurations.

5.3. Reducing the benchmarks by subsetting

Two of the benchmarks of the SPEC2000 suite that have been widely found to be similar based on workload characteristics,

are *bzip* and *gzip* [30], with one ending up as the representative benchmark of the other. However, as the results of Tables 5 and 6 show, these two benchmarks have very different customized architectural configurations. Specifically, the benchmark *bzip* attains 33% slowdown when executed on the customized configuration of *gzip*, and the reverse scenario results in *gzip* observing 43% slowdown. This illustrates the fact that the effect of directing workloads to architectures based on the characteristics of the workloads can be drastically adverse.

More importantly, using workload characteristics to eliminate benchmarks from the exploration process in the design of a heterogeneous system can lead to suboptimality in the final design solution. If *gzip* is assigned as the representative benchmark for *bzip*, a reevaluation of the dual-core combination for harmonic-mean IPT finds the configurations of *bzip* and *crafty* to be the best solution. These two configurations however result in a harmonic-mean IPT of ~1.87, and a ~0.5% slowdown compared to when *gcc* and *mcf* are employed. Although the effect is small, this example shows the effect of excluding a *single* benchmark based on subsetting, and proves how relative similarity in workload characteristics can be misleading if interpreted incorrectly.

The regions of code and compilation settings employed in the aforementioned studies may differ from that employed here. Nevertheless, relative similarity in workload behavior is a discrete property and major similarities are not expected to be affected by minor variations in the characteristics. Therefore, we believe that as long as the regions of code are roughly representative of the whole benchmark, such a cross-publication comparison is legitimate.

5.4. Assigning Surrogates

While a complete search of the core combinations can provide an accurate solution for the best core combination, it provides little insight into how the different benchmarks relate to each other with respect to their optimal architectural configurations. It thus provides no avenue for less-quantifiable factors, such as design-complexity (which relies largely on human judgment), to weigh in on the choice of core combination. There is also, on a lesser note, the fact that the complexity of a complete search grows quadratically with the number of benchmarks considered.

A more valuable representation of this information would hierarchically reflect how the optimal configurations of certain benchmarks can serve as *surrogates* for others. This information is embedded in the cross-configuration performance results (Table 5), but needs to be extracted into a more human-readable representation. While the use of the dendrogram is customary in displaying subsetting properties, its use for displaying the potential for surrogating (assigning the customized architecture of one benchmark to another) can potentially be misleading.

Specifically, the elements that fall into the same subset according to a dendrogram are more related to each other than they are to components of other subsets. Therefore, to increase clustering, two clusters fully merge into a super-cluster that en-

compasses *all* the elements of both. This however, is not the case when assigning surrogate architectures. A benchmark may be better off with a totally different surrogate when its current surrogate is itself assigned a surrogate. For instance, from Tables 5 and 6, the benchmark *bzip* attains best performance on the customized configuration of *twolf* from among the three best configurations for harmonic-mean IPT. However, from among the best two for harmonic-mean IPT, *bzip* attains better performance on the configuration of *mcf*, while *twolf* does so on that of *gcc*.

A *complete surrogating-graph* is a graph that conveys all the cross-configuration information (such as that in Table 5). In the following subsections we explore greedy approaches to formulate *reduced surrogating-graphs*. An issue of importance is that if the customized architecture of workload A is the surrogate of another workload B, whether A should itself be allowed to be assigned a surrogate, e.g. the architecture of C. Another issue is that if workload A has been assigned the surrogate architecture of another workload B, whether A's own architecture should be allowed to become the surrogate of another workload C – which effectively translates into B's architecture becoming the surrogate of C. These issues, which we refer to as *forward-propagation* and *backward-propagation* of surrogates, are illustratively demonstrated in figure 5.

In all the following illustrations a circled benchmark means that the architectural configuration of the benchmark serves as surrogate for benchmarks connected to it by a downward edge. The number on each edge indicates the order of the corresponding surrogate assignment.

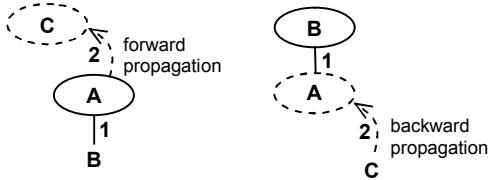


Figure 5: Forward and backward propagation of surrogates with three benchmarks/architectures (A, B and C).

5.4.1. Non-propagation of Surrogates

Figure 6 displays how workloads are grouped together and assigned customized architectures without propagation of surrogates. Employing the remaining architectures (i.e., those of *gap*, *twolf*, *vortex*, and *crafty*) results in a harmonic-mean IPT of 1.83, and an average of 5.66% in performance slowdown across all benchmarks compared to the ideal case of all benchmarks being executed on their own customized architectures. Three of these benchmarks are among the four determined as the best for average and harmonic-mean IPT through a complete search – for which a harmonic mean IPT of 2.08 was attained (see Table 6). The bulk of the slowdown is due to the very last assignment; surrogating *mcf* the customized architecture of *twolf* as surrogate. Adding *mcf* to the set of architectures results in a harmonic-mean IPT of 2.1 and reduces the average slowdown to ~1.6%. This however is achieved at the cost of 5 cores.

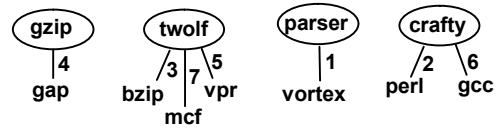


Figure 6: The reduced surrogating-graph through the greedy assignment of surrogate architectures to benchmarks – with no propagation of surrogates.

The issue with this approach is that by prohibiting the propagation of surrogates, the assignment of surrogates becomes limited to a number of architectures that happened to have been extremely good surrogates for a limited number of workloads. Moreover, this approach is not extendable and will not provide a solution for a heterogeneous design with a smaller number of cores. The methodology will eventually reach a point where none of the remaining workloads can be surrogated.

5.4.2. Propagation of Surrogates

Figure 7 displays the outcome of the greedy assignment of surrogates to benchmarks with forward and backward propagation of surrogates. Double-line edges indicate the architecture that is the surrogate of some other workload itself being surrogated. The top circled workload in each group of connected workloads indicates the workload whose architecture is used as surrogate for all the workloads in the group. For traceability, Appendix A displays the cross-configuration percentage slowdowns of the benchmarks with the links that are selected according to this greedy assignment of surrogates marked with a star (*).

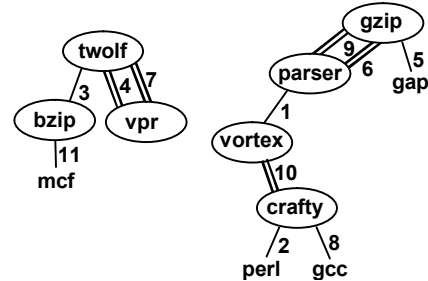


Figure 7: The reduced surrogating-graph through greedy assignment of surrogate architectures to benchmarks – with full propagation of surrogates.

It is observable that a very different grouping is attained with this approach compared to when propagation is disallowed. A heterogeneous system that employs the customized architectures for *gzip* and *twolf* results in a harmonic-mean IPT of 1.74 and an average slowdown of ~18% across all benchmarks compared to an ideal system. An interesting scenario that occurs twice in this example is that, in the greedy assignment of surrogates, the benchmark such as *vpr* (*parser*) is surrogated by the customized configuration of the benchmark *twolf* (*gzip*) which is itself already surrogated by *vpr* (*parser*). This phenomenon, which we refer to as *feedback-surrogating*, prevents the assignment of surrogates from being continued until only a single configuration is remaining.

Propagation of surrogates untangles the process of assigning surrogates. However, it may result in a benchmark being surrogated by an unsuitable architecture, while more suitable options are available. This is intensified when the both forward and backward propagation are employed together. For instance in Figure 7, the surrogate assignment with order 10 results in both forms of propagation. – rendering the architecture of *gzip* as the surrogate for *perl* and *gcc*.

Figure 8 displays the outcome with only forward-propagation of surrogates. The corresponding assignments are marked in appendix A with underlining. The architectures determined as the best two candidates for a heterogeneous system are the customized architectures of *mcf* and *vpr*. The harmonic-mean IPT with these benchmarks is 1.75.

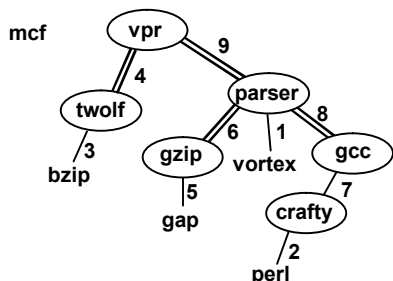


Figure 8: The reduced surrogating-graph through the greedy assignment of surrogate architectures to benchmarks – with forward propagation of surrogates.

In these examples, all benchmarks are considered to have an equal *importance weight*. To consider different importance weights, the slowdowns due to surrogating must be weighed by the importance weight of corresponding workloads in order to guide the exploration process to a design that is more favorable for workloads that consume most of the system’s time. The frequency of job submissions of a particular workload type may be considered an indication of the importance weight of a workload. The product of the frequency of workload submission and the execution time of the workload can also be used to weigh the importance of a workload. However, the execution time of a workload depends on the configuration on which it is executed. This will further complicate the exploration process, unless rough approximations of the relative execution times are employed.

5.5. Multi-threaded Performance

The major difference between the multithreaded and single-threaded scenarios is the issue of *contention* for core access. For instance in the design represented by Figure 8, ten benchmarks employ the customized configuration of *vpr* as their surrogate architecture, which is inconsequential as long as jobs are considered to be submitted individually. But with concurrent jobs, the effect of contention in access to a core that is the surrogate of numerous workloads becomes an issue of concern. Contention can be dealt with in two manners. Either submitted workloads stall until their assigned surrogate core is free, or they are directed to the next most suitable available core. In the

former case, if the objective is to minimize the average execution time of submitted jobs the optimum design is no different than that for single-thread performance. If the objective is to minimize the average turnaround time of jobs, then in addition to minimizing the average slowdown of workloads the aggregate importance weight of workloads assigned to a configuration should be balanced too. This problem is similar to the problem dubbed as the Balanced Partitioning of Minimum Spanning Trees (BPMST) problem [31].

In the case where jobs are directed to the available core that is most suitable, the optimum core combination is more complicated to determine as it is dependent on the distribution of job submissions. Little research has been conducted on the distribution of job submissions with respect to their workload behavior in multi-processor systems. However, with a Poisson distribution and an average submission period proportional to the average execution time, temporary hot-spots and redirection of workloads to cores other than their surrogate customized core will be infrequent, allowing a BPMST-based assignment of surrogate cores to provide an acceptable solution. As the burstyness of the distribution increases however the benefit of heterogeneity will diminish.

We defer the analysis of approaches to communal customization for multi-threaded performance and the effect of different distributions of job submission of for future work.

6. CONCLUSION

A superscalar design space exploration tool that allows variation in the sizing of different units of the superscalar processor is employed to determine the optimal architectural configuration for each of the integer SPEC benchmarks. The best core combinations to employ based on different criterions were determined, and it is shown that through initially reducing the set of workloads based on similarity in raw characteristics, may result in lower performance. This shows that the optimal architectures for executing workloads provide a more valuable source of information about the similarities between the workloads with respect to their resource needs.

As a summary, Table 7 illustrates the overall single thread performance of a 2-core heterogeneous system attained through different techniques.

Table 7: Summary of performance results for a dual-core CMP

Scenario \ Metric	Harmonic mean IPC	Slowdown compared to ideal
Ideal (every workload employing its own customized arch).	2.12	0%
Homogeneous system with all cores designed for best overall performance (gcc).	1.57	26%
Heterogeneous system with core arch's determined through complete search (gcc, mcf).	1.88	11%
Heterogeneous system with core arch's determined through greedy assignment of surrogates with propagation of surrogates.	1.74	18%

Acknowledgments

This research was supported in part by NSF grants CCF-0429843 and CCF-0702632, and an Intel grant. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

References

- [1] S. Bird, A. Phansalkar, L. John, A. Mericasa, R. Indukuru, "Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor", *SPEC Benchmark Workshop*, Jan. 2007
- [2] T Mitra, T Chiueh, "Dynamic 3D Graphics Workload Characterization and the Architectural Implications", *MICRO-99*, p.62-71, Nov. 16-18, 1999, Haifa, Israel.
- [3] G. A. Abandah and E.S. Davidson, "Configuration Independent Analysis for Characterizing Shared-Memory Applications," *Proc. 12th Int'l Parallel Processing Symp.*, 1998, pp. 357-398.
- [4] E. Sohmaier, H. Shan, "Architecture Independent Performance Characterization and Benchmarking for Scientific Applications," *Proc. of the Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecom. Systems (MASCOTS'04)*, pp. 467-474 2004.
- [6] L. Eeckhout, H. Vandierendonck and K. De Bosschere "Quantifying the Impact of Input Data Sets on Program Behavior and its Applications." *Jour. of Instruction-Level Parallelism*, Vol. 5, April 2003.
- [7] H. Vandierendonck, K. Bosschere, "Many Benchmarks Stress the Same Bottlenecks", *Proc. of the Workshop on Computer Arch. Eval. using Commercial Workloads (CAECW-7)*, pp. 57-71, 2004.
- [8] A. Joshi, A. Phansalkar, L. Eeckhout, L. John, "Measuring Benchmark Similarity Using Inherent Program Characteristics," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 769-782, Jun., 2006.
- [9] A. S. Dhodapkar and J. E. Smith, "Managing Multiconfiguration Hardware via Dynamic Working Set Analysis," *Proc. Int'l Symp. Computer Architecture, IEEE CS Press*, 2002, pp. 233-244.
- [10] R. Kumar, D. M. Tullsen, N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors", *Parallel Architectures and Compilation Techniques (PACT)*, pp. 23-32, 2006.
- [11] K. Hoste, A. Phansalkar, A. Georges, L. John, "Performance prediction based on inherent program similarity", *Proceedings of the 15th international conference on Parallel architectures and compilation techniques (PACT)*, 2006.
- [12] M. Ekpanyapong, S. Kyu Lim, C. Ballapuram, and H. S. Lee, "Wire-driven Microarchitectural Design Space Exploration," *IEEE International Symp. on Circuits and Systems*, p1867-1870, 2005.
- [13] J. Huh, D. Burger, and S. W. Keckler, "Exploring the Design Space of Future CMPs," *Proceedings of PACT*, Sept. 2001.
- [14] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. "Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction", In *International Symposium on Microarchitecture*, Dec. 2003.
- [15] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis, "Microarchitecture Evaluation with Physical Planning", *Proc. Of the Design Automation Conference*, Anaheim, pp. 32 - 36, June 2003.
- [16] Y. Ma, Z. Li, J. Cong, X. Hong, "Micro-architecture Pipelining Optimization with Throughput-Aware Floorplanning", *ACM/IEEE Asia South Pacific Design Automation Conference*, Japan, Jan. 2007.
- [17] D. Marculescu, A. Iyer, "Application-driven processor design exploration for power-performance trade-off analysis", *ICCD*, 2001.
- [18] K. Skadron and P. S. Ahuja, "Hydrascalar: A multipath-capable simulator," *Newsletter of the IEEE Tech. Committee on Computer Architecture*, Jan 2001.
- [19] P. J. Joseph, K. Vaswani, M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," *MICRO-39*, pp. 161-170, 2006.
- [20] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's Law Through EPI Throttling", In *Proceedings of International Symposium on Computer Architecture*, 2005.
- [21] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures", In *Proceedings of International Symp. on Computer Architecture*, 2005.
- [22] S. Ghiasi and D. Grunwald, "Aide de camp: Asymmetric dual core design for power and energy reduction", In *University of Colorado Technical Report CU-CS-964-03*, 2003.
- [23] S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for heterogeneous processors in server systems", *Computing Frontiers*, 2005.
- [24] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of both latency and throughput", *ICCD*, 2004.
- [25] T. Austin, E. Larson, D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer*, vol.35, no.2, Feb. 2002.
- [26] R. H. J. M. Otten, L. P. P. Van Ginneken, "Stop criteria in simulated annealing", *ICCD*, 1988.
- [27] J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D. Lilja, and L. K. John, "Evaluating Benchmark Subsetting Approaches" *International Symposium on Workload Characterization*, October 2006, pp 93-104
- [28] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Four Generations of SPEC CPU Benchmarks: What has changed and what has not", *Tech Report TR-041026-01-1*. Oct. 2004.
- [29] H. Vandierendonck, K. De Bosschere, "Experiments with Subsetting Benchmark Suites," *Workshop on Workload Charact.*, 2004.
- [30] A. Phansalkar, A. Joshi, L. Eeckhout, L. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites", *ISPASS*, pp. 10-20, 2005
- [31] M. Andersson, J. Gudmundsson, C. Levcopoulos, G. Narasimhan, "Balanced Partition of Minimum Spanning Trees", *International Conference on Computational Science*, pp. 26-35, 2002
- [32] J. Yi, D. Lilja, and D. Hawkins, "A Statistically Rigorous Approach for Improving Simulation Methodology," *Proc. 9th Ann. Int'l Symp. High-Performance Computer Architecture*, pp. 281-291, 2003.
- [33] J. Dujmovic and I. Dujmovic, "Evolution and Evaluation of SPEC benchmarks", *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 2-9, 1998.
- [34] M. Laurenzano, B. Simon, A. Snively and M. Gunn, "Low Cost Trace-Driven Memory Simulation Using SimPoint," *Workshop on Binary Instrumentation and Applications* (held in conjunction with PACT2005), Sep. 2005, St. Louis, MO.
- [35] "The Use and Abuse of SPEC: An ISCA Panel," *IEEE Micro*, vol. 23, no. 4, pp. 73-77, Jul/Aug, 2003.
- [36] Steven J. E. Wilton and Norman P. Jouppi. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677-688, May 1996.
- [37] B. Lee and D. Brooks, "Illustrative design space studies with microarchitectural regression models", In *International Symp. on High-Performance Computer Architecture*, 2007.

Appendix A:

The percentage slowdown of each benchmark on the customized cores of other benchmarks (see main text for description of markings).

	bzip	crafty	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
bzip	0%	35%	45%	23%	33%	18%	33%	35%	*3.1%	28%	6%
crafty	66%	0%	50%	8%	17%	79%	14%	10%	44%	*8%	43%
gap	53%	8%	0%	13%	*3.3%	70%	4%	7%	33%	18%	32%
gcc	48%	*4.4%	37%	0%	10%	66%	11%	28%	21%	9%	20%
gzip	43%	18%	35%	7%	0%	59%	*3.8%	31%	23%	17%	24%
mcf	20%	56%	67%	51%	68%	0%	65%	55%	44%	54%	44%
parser	29%	19%	16%	20%	*5%	49%	0%	29%	6%	17%	12%
perl	58%	*2%	56%	12%	19%	73%	20%	0%	36%	12%	37%
twolf	9%	46%	55%	31%	51%	35%	39%	50%	0%	36%	*3.2%
vortex	51%	13%	25%	9%	15%	68%	*0.5%	18%	23%	0%	25%
vpr	25%	36%	45%	17%	47%	49%	34%	38%	*4.3%	27%	0%