Core-Selectability in Chip Multiprocessors

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

Abstract—**The centralized structures necessary for the extraction of instruction-level parallelism (ILP) are consuming progressively smaller portions of the total die area of chip multiprocessors (CMP). The reason for this is that scaling these structures does not enhance general performance as much as scaling the cache and interconnect. However, the fact that these structures now consume less proportional die area opens an avenue to enhancing their performance through truly overcoming the one-size-fits-all approach to their design.**

This paper proposes core-selectability – incorporating differently-designed cores that can be toggled into active employment. This enables differently customized ILP-extracting structures to be at hand in the system while not dramatically adding to the interconnect complexity. The design verification effort is minimized by separating the complexity of different core designs. Moreover, contrary to alternative approaches, the performance and power efficiency of the core designs are not compromised.

Evaluation results are presented that show that, even when limiting the diversity between core designs to only the sizing of microarchitectural structures, core-selectability has the potential to provide notable performance enhancement (with an average of 10%) to scalable multithreaded applications, without increased concurrency. In addition, it can provide significantly greater throughput to multiprogrammed workloads by providing the potential for the system to transform into a heterogeneous design.

Chip Multiprocessor; Heterogeneity; Microarchitecture

I. INTRODUCTION

In the design of a chip multiprocessor (CMP), if the balance in resource provisioning is to be maintained, an increase in the number or performance of processing cores requires an increase in the cache and interconnection resources. This does not necessarily mean that all applications utilize the provisions to the fullest extent, however. By overcoming the instruction-level bottlenecks of applications that underutilize the cache and interconnect, it is possible to enhance their execution performance, and yet maintain the balance in provisioned resources. This will result in better utilization of provisioned cache and interconnect.

A major factor that inhibits instruction-level performance enhancement is the one-size-fits-all approach to the design of the centralized units necessary for extracting instruction-level parallelism (ILP), e.g., the issue-queue, load/store queue (LSQ), reorder buffer (ROB). This is a result of the inherent criticality of these units, which renders them impractical for dynamically changeable design solutions (i.e., reconfiguration). If it were possible to genuinely adjust the configuration of these units to suit the application at hand, notable instruction-level performance could be gained.

Meanwhile, as merely scaling the size of the ILP-extracting units does not necessarily improve their general performance, the actual cores in CMPs have been consuming progressively smaller portions of the physical layout. This opens an avenue to a different form of instruction-level performance enhancement: replacing each core in the CMP with a cluster of multiple differently-designed cores, called a *node*, with the option to dynamically select which core to actively employ. The purpose of these different cores is to provide microarchitectural diversity, rather than concurrency. Thus, only one core in each node need be actively employed at a time. This allows for the cores in a node to share the complex resources that interconnect nodes together in the CMP, and maintain the original provisioning of these resources. We refer to this technique as *core-selectability*, as its main benefit comes from the ability to select the microarchitectural design to be employed.

This technique can achieve what reconfiguration aspires to achieve. It is a scalable solution to using the available transistors to enhance multithreaded performance without overly increasing design complexity, verification effort or power consumption. It allows for the microarchitecture design effort to be partitioned and focused on specific types of workload behavior, rather than attempting to pack everything into one complex design solution. Moreover, it provides the potential for the system to transform into a heterogeneous design (and even different forms of heterogeneity), enabling greater throughput to multiprogrammed workloads [30] and better performance to critical-section intensive multithreaded workloads [29].

In this study we investigate the implications and benefits of implementing core-selectability in a general-purpose chip multiprocessor. In the experimental evaluation, the considered core designs are based on propagation delays observed for different microarchitectural structures attained from a detailed synthesizable HDL model of a superscalar processor in 45nm technology.

The following section provides background and an overview of related work. The implementation of core-selectability is described in Section III. Our evaluation methodology and choice of core designs are outlined in Section IV. Section V presents evaluation results that illustrate the performance benefit of core-selectability under a wide range of multithreaded benchmarks when the cores have the same clock period. In Section VI, we present results showing how core-selectability can improve task turnaround time and execution throughput under multiprogrammed workloads. Fur-

1089-795X 2009

DOI 10.1109/PACT.2009.44

ther discussion is presented in Section VII, and Section VIII concludes this study.

II. Background

A. Customized Design of Processing Cores

Prior work has shown that there is significant performance benefit in employing customized core designs for different workload behavior [11] [30]. Nevertheless, when limited to a single core design, the best solution is one that provides reasonable performance across a wide range of workload behavior. Thus, the different units of a general-purpose processing core need to be designed in anticipation of typical workload behavior. Such a system will perform suboptimally when the actual workload being executed on the system displays atypical behavior – with undersized structures degrading IPC and oversized structures wasting propagation delay. And yet, in a general purpose system, all workload behavior can be atypical.

The characteristics of the employed technology can also impact the best tradeoffs in the design of a processing core. For instance, extracting greater parallelism requires more complex logic (and longer propagation delays), which either results in deeper pipelining or an increased clock period. While increasing the clock period can directly degrade performance, deeper pipelining can impact the cycle delay between the wakeup of dependent instructions, adversely impacting parallelism.

Moreover, intricate circuit-level details can dramatically sway the best design tradeoffs for a given workload behavior. For instance, the unified clock period intertwines the different microarchitectural design units. Thus, in a high-performance design, the scaling of any unit must either result in change in the pipeline depth of that unit or it must be accompanied by proportional scaling in the propagation delay of all other units (to enable frequency scaling). To make things even more complicated, different units of the design tend to scale differently and are not ideally pipelinable. This can result in pipeline slack, which increases effective propagation delay, and degrades performance.

The ability to employ different core designs opens the door to a much broader overall system design space, and the potential for considerable performance gain. One aspect of this design space is the manner in which the workload space should be split up between the cores, for each to be customized to [31]. Another aspect is the customization of core designs to their constituent workload behavior. Correctly exploring this design space requires core customization in which intricate circuit-level details are accurately accounted for. In this design space, abstracting away the circuit-level details can lead to inaccurate assessments and the adoption of severely suboptimal design solutions.

B. A Circuit Level Model

In order to be able to account for circuit-level details, and conduct accurate core customization, we have developed a fully-synthesizable Verilog model of a contemporary pipelined out-of-order superscalar processor [24]. This model has parameterized microarchitectural features, and is aimed at high-fidelity design space exploration. It enables evaluation of the effect of the propagation delay and pipeline depth of different microachitectural units on overall performance under any technology characteristics. The results presented in this study are based on results attained from the synthesis of different configurations in 45nm technology.

As an example, Figure 1 shows the propagation delay of the select logic, extracted from the model, when the issue-queue size and issue width are scaled. These results show that, for a specific propagation delay, there is a tradeoff between the size of the issue queue and the issue width. The best tradeoff depends on the workload behavior.

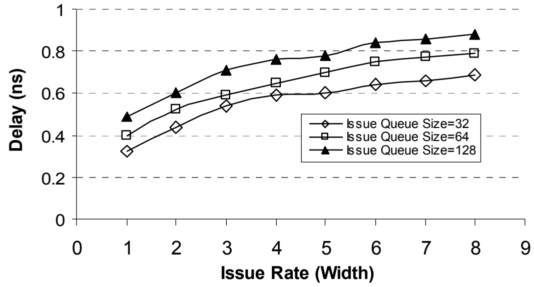


Figure 1. Propagation delay of the selection logic in 45nm technology, when the issue width and issue-queue size vary.

C. The Overhead of Reconfigurability

One approach to enabling different workloads to be executed on suitable core designs is through reconfigurability. In implementing reconfigurability, there is a tradeoff between the flexibility of the design and the overhead introduced to the system. Adaptable architectures [2] have less overhead, but provide less flexibility in the design. In contrast, FPGA-based reconfigurability [1] provides high flexibility at the cost of large overheads. Nevertheless, in critical microarchitectural units the overhead of even the most inflexible forms of reconfigurability can outweigh the would-be benefit.

Although prior studies have proposed adaptable implementations of various microarchitectural units, their focus tends to be on reducing power consumption when needed, with minimal performance degradation. Achieving performance enhancement through adaptability, however, is more challenging.

1) Reconfigurability in the logic

An example of the difficulty in implementing reconfigurability can be observed in the superscalar wakeup logic. The main component of propagation delay in the wakeup logic is the load on the rails that broadcast newly issued instruction tags to the comparators that compare them with those of waiting instructions.

Downsizing the effective issue-queue size can, in theory, be achieved by switching off the load of the unwanted comparators. In practice, however, switching off the capacitance can only be achieved through buffering a portion of the broadcasting rail, as shown in Figure 2. Such buffering will provide the effect of a repeater when the issue-queue is not downsized [2]. But, since the buffer will need to be large enough to drive the rest of the broadcasting rail, it will permanently place a large extra load on the rail, increasing the propagation delay of the downsized portion.

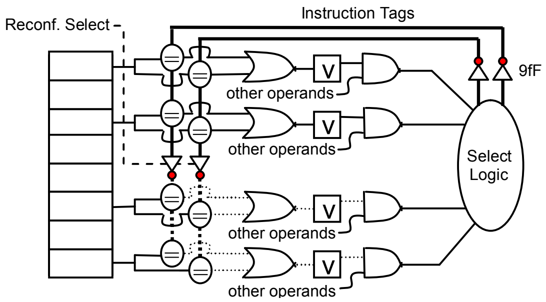


Figure 2. Reconfiguration in the issue-queue size.

Using the optimal repeater placement of SoC Encounter, it was found that in 45nm technology the best design for a 128-entry issue-queue consists of 4 large buffers, each with 9fF input capacitance, driving all the issue-queue comparators in a tree structured layout. Implementing the ability to downsize the issue-queue within this circuitry results in a notably larger propagation delay compared to custom downsized issue-queue designs.

Table 1 shows the propagation delay of the wakeup and select logic in a 4-wide issue-queue with a maximum size of 128 entries that can dynamically be downsized. Also shown in this table is the propagation delay for custom downsized issue-queue designs. These results show that, the reconfigurable design, at its full size (of 128 entries) is 15% slower than a same-sized custom design. For smaller issue-queue sizes, the reconfigurable design becomes even slower relative to same-sized custom designs – with 46% longer propagation delay for the 16-entry size. Therefore, although there may be some benefit in this form of reconfigurability, it is far from the true benefit of customization.

TABLE I. EFFECT OF ISSUE-QUEUE SIZE ON PROPAGATION DELAY WITH AND WITHOUT RECONFIGURABILITY

Issue-Q size	Wakeup Delay (ns)	Select Delay (ns)	Wake & Select Delay (ns)	Reconfig. Delay (ns)
16	0.55	0.54	1.09	1.55
32	0.635	0.59	1.38	1.89
64	0.67	0.65	1.62	2.1
128	0.82	0.76	2	2.3

Implementing reconfigurability in the issue width is even more challenging. The dependent instruction tags of waiting instructions are connected to as many comparators as there is issue width. The extra load of the unnecessary comparators in the downsized setting can not be dynamically removed from the circuit without inserting extra buffering. This buffering introduces extra load in the system, which degrades performance.

Nevertheless, the direct overhead in propagation delay is not the only factor inhibiting performance gain from such reconfigurability.

2) *Reconfigurability in the pipeline structure*

Even if certain microarchitectural units were to be made optimally adaptable at the logic-level, the most difficult factor in attaining performance benefit from such reconfigurability is in connection with the tight pipelining of high-performance processor designs.

In fact, *selectability*, as a general concept, can be employed to create reconfigurability in any microarchitctural unit (providing the ability to *select* from among different implementations of a unit). Moreover, adaptable caches [32] have been shown to be implementable with fairly low overhead. However, it is challenging to draw on such confined reconfigurability without causing slack and imbalance in the pipeline structure, as different design units scale differently and some do not scale at all.

For instance, the operation of the functional units is determined by the ISA and is unaffected by the microarchitectural configuration. Moreover, the functional units reside within the feedback loop of any microarchitecture, and any slack in the pipeline stages of the functional units will impact the raw performance of the system. Such portions of a processor design need to be pipelined with slack in low clock frequency configurations for correct functionality in high frequency configurations.

Dynamic pipeline scaling (DPS) provides the potential for variable depth pipelining of a microarchitectural unit [21]. This can theoretically enable the independent scaling of different design units, while preventing slack in any pipeline stage. In practice, however, DPS introduces overhead in the form of extra latch propagation delay and power consumption. More importantly, it is only practical for multiplying or dividing the clock frequency, which dramatically limits the viable design space.

3) *The verifiability of reconfigurability*

Design verification needs to be accounted for in high level design choices as it has become the major consumer of man-hours in the development of modern processors [33]. It has been shown that design symmetry reduces verification cost by mitigating the effective number of functional states that need to be accounted for [5].

Reconfigurability, however, *desymmetritizes* the design of a processing core by creating differences between portions of the microarchitecture that would otherwise be identical. For instance, in the implementation of an adaptable issue-queue, the entries that are disabled in the down-sized mode differ from those that are not disabled. In addition, the logic that implements reconfigurability itself is inherently not symmetric. This can lead to an explosion in the number of states, and a potentially exponential increase in verification effort.

Verifiability is not only an issue of concern in reconfigurable designs. Any design solution that attempts to push the limit on overall performance will inevitably be more complex and less symmetric, requiring greater design verification effort. Core-selectability allows for the design complexity of each design solution to be limited, by focusing on only the constituent workload behavior.

D. *Increasing Concurrency*

Question: What is preventing manufacturers from employing a larger number of processing cores in their chip multiprocessor designs than they currently are?

Merely replicating the cores themselves consumes no extra design or verification effort. Moreover, each additional core consumes incremental die area. The issue is not chip yield either – with companies such as IBM already employing multi-die-in-package design solutions (or Multi-Chip Module technology) [22]. Therefore, the main reason must lie in the complexity and design effort added cores introduce to the interconnection network and cache hierarchy, in addition to a limited power budget.

As illustrated in Figure 3, in conventional chip multiprocessor design, increasing the number of cores in the system necessitates more interconnect bandwidth and cache capacity. This increases not only core-to-core latency, power consumption and manufacturing cost, but also design and verification cost. In general, the major problem with on-chip networks is that they simply do not scale very well.

Nevertheless, the issue of concern here is not cost per se (as it is natural for greater performance to entail higher cost), but rather wastefulness. It is wasteful to dedicate further resources to any portion of the design before it has been fully utilized. However, a large portion of workloads tend to underutilize the cache and interconnect. The fundamental reason for this is that workloads that place high demand on these resources are more dominant in influencing what entails a well provisioned design – as insufficient provisioning dramatically degrades their performance (and consequently overall performance).

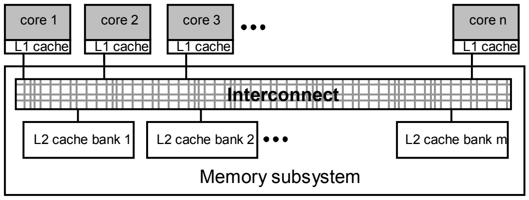


Figure 3. Complexity of the interconnection in a conventional design.

Adding cores to a system without increasing the provisioned interconnection resources can alternatively be achieved by splitting the existing resources between the cores (i.e. splitting the channel bit-width in crossbar-based interconnect, and the bisection-width in tile-based). However, if doing so yields better overall performance it indicates that the interconnect was originally over-provisioned anyway. Moreover, the overhead of this form of increasing concurrency permanently impacts the system – such that applications that are inherently less scalable permanently observe performance degradation.

All in all, we believe that an objective should be to enhance the instruction-level performance of workloads that underutilize the existing provisions in cache and interconnection resources, before provisioning more of such resources.

E. Related Work

The *Conjoined-core* [7] approach to Chip Multiprocessor design has been proposed as a solution to efficiently increase concurrency. The approach is a tradeoff between simultaneous multithreading [8] and single-chip multiprocessing [9]. Multiple homogeneous cores are added to the processing nodes of a CMP, and the cores time-share resources such as the floating-point unit, instruction cache, data cache and crossbar ports. In this manner, concurrency can be increased with fewer resources. The CASH architecture [10] is a similar approach. L1 cache port sharing has also been employed in Sun's recent Rock processor [28].

In the implementation of core-selectability, resources are also shared between cores. However, the objective is to enhance performance through dynamic core customization, rather than concurrency. In fact, while the sharing of resources is part of the objective in the conjoined-core and CASH techniques (and Rock architecture), it is more of an imposition in core-selectability.

Core-fusion [4] and *Composable Lightweight Processors* [3] are similar techniques to enable reconfiguration of the cores in a chip multiprocessor by combining smaller cores to form larger ones. This enables the cores to vary and become more suitable for the application at hand. However, these techniques are not aimed at providing performance benefit to scalable multithreaded applications, as maximizing the number of available cores provides the best overall performance. More importantly, the reconfiguration overhead of implementing both techniques manifests itself in the microarchitectural units that are most critical to ILP extraction, i.e., the issue-queue, ROB and LSQ. Salverda and Zilles show that there are fundamental obstacles to achieving good performance through core-fusion with in-order cores [23].

Previous studies have demonstrated the performance and power benefit of heterogeneity over homogeneity in CMPs designed for multi-programming environments [11] [12] [13] [30]. Heterogeneity entails the employment of differently designed cores for the execution of tasks with different workload behavior. However, it has also been shown that heterogeneity can degrade the performance predictability and scalability of multi-threaded applications [14].

In panel talks, Patt [27] has suggested the abstract notion of employing large specialized units that can be powered down when not needed (the "Refrigerator" analogy), as a power-efficient approach to putting the growing availability of transistors to use. However, to the best of our knowledge, no prior work has looked at the option of employing complete processing cores that are differently designed with the intent to employ only one at a time.

Integral to our evaluation methodology is the circuit-level modeling of the propagation delay of different microarchitectural units in the design of processing cores. The Illinois Verilog Model [16] is functionally the closest to our design, although it is not fully synthesizable. Other freely available HDL processor models [17] [18] do not represent the complexity of an out-of-order superscalar microarchitecture.

III. Implementation of Core-Selectability

The strength of core-selectability lies in its simplicity and the absence of microarchitectural invasiveness. Basically, the design adjustments needed for implementing core-selectability are only in the back-end and front-end of the core designs. This is essential to minimizing design and verification effort. Moreover, the main mechanism required to implement core-selectability (i.e., port sharing) has been proposed and employed elsewhere. Thus, the novelty of core-selectability lies in the purpose for which such mechanisms are employed, rather than the mechanisms themselves.

In the front-end, the cores can share a port to the instruction cache. However, it is important that each core possess a dedicated fetch engine, as this unit is closely tied to the functional units that determine branch outcomes. In the back-end, the data-paths of differently designed cores need to share a port to the data cache. Only the core that is selected for active employment will have access to these ports.

The cores can be made selectable at the L1 cache level or the port to the shared L2 cache. Implementing selectability at the L1 level has the advantage of better utilization of die area. Selectability at the L2 level has the advantage of enabling L1 cache customization, and places no overhead on the more critical L1 cache accesses. In this study we focus on implementing core-selectability at the L1 level. The design of the Rock processor [28] ensures that, at the very least, port-sharing is physically implementable at this level.

Figure 4 illustrates the basic schematic of a two-way core-selectable design. Within each node, the active core takes over the port to the L1 data cache. Note that this figure only shows the multiplexing of the address signals to the L1 cache, not how the data paths are directed to both cores.

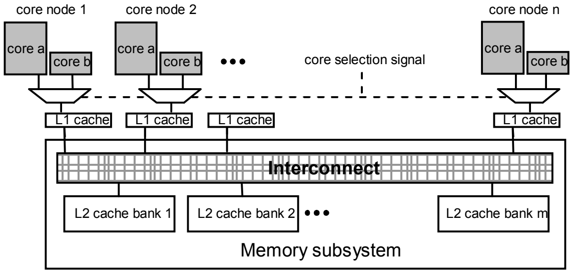


Figure 4. Complexity of the interconnection in a core-selectable design.

Another option in the implementation of core-selectability is whether or not to allow the different cores to have different clock frequencies. Customizing the clock frequency allows for the pipeline depth of the core designs to be customized to the characteristics of the workload, and can considerably increase the viable design space of the processing cores. When cores with different clock domains are employed, preventing pipeline slack requires either adaptable caches at the L1 level, or asynchronous buffering – which results in a form of Globally-Asynchronous Locally-Synchronous (GALS) design [34].

A. *The Propagation Overhead of Core-Selectability*

As shown in Figure 5, the multiplexing of access to the caches does introduce extra logic to the system. In addition, the cores may end up being more physically distant from the shared port to the L1 cache than a dedicated core would have been.

Using SPICE analysis with the 45nm FreePDK library [19], a wire with a length in the range of 1mm (the diameter of a typical core) in the L3 metal layer is estimated to have a capacitance no larger than 100fF. In this technology, a multiplexer designed with pass-transistor logic and optimized for up to 100fF input capacitance, was found to display 26 picoseconds propagation delay. Therefore, port sharing should result in added propagation delay no larger than 26 picoseconds. This small propagation overhead is made possible by the fact that the sizing of the gate need only be optimized for minimal propagation delay, rather than minimal switching delay. Nevertheless, the main issue is not the small propagation overhead, but the fact that the changeability is not implemented within tightly-coupled microarchitectural pipeline stages.

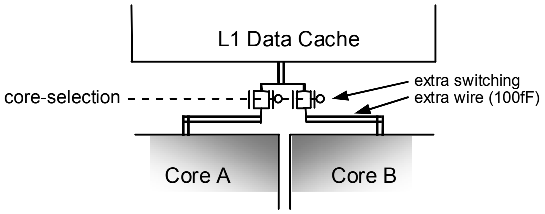


Figure 5. The extra switching and wire propagation delay of port-sharing.

B. *Transferring Execution*

All user-level and system-level instructions execute on the currently active core within a node. When the operating system scheduler chooses to schedule a task on a core that is different from the currently active core, it first finishes up what it is doing on the currently active core and then executes a final instruction on the currently active core, that simultaneously configures the currently active core to be inactive and asserts an external interrupt signal of the core to be activated. This implies that (1) a core can assert the activation interrupt signal of any other core and (2) a core's external interrupt unit is always active whether or not the core is active. When an inactive core receives an activation interrupt, it vectors its program counter to an interrupt handler that starts the task.

IV. Methodology

A. *Customizing the Core Design*

The goal of core customization is to find a global design optimum that captures the interplay between workload characteristics, the microarchitecture, and the physical implementation. Thus, propagation delays of microarchitectural units are fundamental to this exercise. To this end, we developed a synthesizable Verilog model of an out-of-order superscalar processor. Details of this model are available in a preliminary report [24]. Major components or features are either parametrically configurable (*e.g.*, structure sizes) or different configurations for them have been explicitly designed (*e.g.*, number of superscalar ways in each pipeline stage). Different designs were synthesized with Synopsis Design Compiler V2005.09-SP3 and placed-and-routed with Cadence SoC Encounter V7.1, using the FreePDK OpenAccess 45nm Standard Cell Library [19].

Since a superscalar processor makes use of many specialized and highly-ported RAMs (*e.g.*, rename map table, architectural map table, shadow map tables, free-list, active-list, physical register file, etc.), we also developed a register file compiler. It uses custom layouts of multi-ported bit-cells and peripheral circuits to generate RAMs and characterize their access times (SPICE model extraction).

B. *Multicore Simulation Setup*

We explore the core design space and evaluate the effect of core-selectability with full-system simulation using the

Virtutech's Simics simulator [25] extended with the Wisconsin GEMS and OPAL [26] simulators. The GEMS simulator provides a detailed memory system timing model, and the OPAL simulator provides a detailed microarchitectural timing model of a processor with the Sparc ISA. The cache and interconnection characteristics considered in all studies are shown in Table II.

TABLE II. CACHE AND INTERCONNECTION CHARACTERISTICS

NETWORK TOPOLOGY	HIERARCHICAL SWITCH
COHERENCE PROTOCOL	MOESI
DATA BLOCK BYTES	64
L1 CACHE ASSOC	2
L1 CACHE NUM SETS BITS	9
L2 CACHE ASSOC	4
L2 CACHE NUM SETS BITS	12

A diverse set of multithreaded benchmarks from the Splash-2, Java-grande and SpecJbb benchmark suites, and the Blast biometric benchmark, are accounted for. The benchmarks and the employed input parameters are listed in Table III. The benchmarks were compiled using the PARMACS [40] library from UPC.

TABLE III. BENCHMARKS WITH INPUT PARAMETERS

Suite	Benchmark + input parameters
Splash2	barnes 8192 123 0.025 0.05 1.0 2.0 5.0 0.075 0.25 4
	cholesky -p4 -B128 -C16384 < tk29.o
	fft -m22 -p4 -n65536 -l4
	fmm two cluster plummer 8192 1e-6 4 5 .025 0.0 cost zones
	lu -p4 -n2048 -b64
	ocean –n258 -p4 -e1e-07 -r20000 -t28800
	radiosity -p 4 -room –batch
	radix -p4 -n2621440 -r2048 -m524288
	raytrace -a8 -p4 teapot.env
	volrend 4 head
	water_spatial < input.p4
Java-Grande	java -cp .:/RayTracer/jg JGFRayTracerBenchSizeA 4
	java -cp .:/MoldDyn/jg JGFMolDynBenchSizeA 4
	java -cp .:/MonteCarlo/jg JGFMonteCarloBenchSizeA 4
Specjbb	java -classpath -propfile specjbb.props
Blast	blastall -p blastn -d ecoli_nt -a 4 < alu.n

C. *The Core Designs*

A major factor in the evaluation of core-selectability is the design of the cores employed in the system. The different microarchitectural parameters were explored under the constraint that the propagation delays of different units remain within a certain number of clock cycles. The performance of each design solution was evaluated for the benchmarks detailed in Table III.

The benchmarks were executed on the multithreaded simulation setup detailed in the previous subsection for 10 million instructions. This was preceded by skipping the initialization phase of each benchmark to arrive at the main execution loop, and warming up the caches for 10 million instructions.

The microarchitectural attributes of the best core design found for average execution time across all the benchmarks are listed under the label *Core-U* in Table IV. Two other core designs were also extracted that, compared to Core-U, provide notably higher performance on different subsets of the benchmarks. The microarchitectural attributes of each of these two core designs, which we will refer to as *Core-A* and *Core-B*, are also listed in Table IV. Each core design attains higher performance on a subset of benchmarks at the cost of lower overall performance across all benchmarks. In choosing these core designs, care was also taken to limit the design space to a fixed clock period, equal to that of Core-U (0.6 nanoseconds). This was necessary to preserve lucidity in the difference between the designs, and prevent the need for asynchronous buffering or adaptable caches.

TABLE IV. CONFIGURATION OF CORES

	Core-U	Core-A	Core-B
FETCH STAGES	4	3	5
DECODE STAGES	1	1	1
RETIRE STAGES	2	2	2
ISSUE WIDTH	3	2	5
ROB SIZE	512	1024	512
IWINDOW SIZE	64	128	32
Clock period	.6ns	.6ns	.6ns

Core-A provides higher performance to applications that have hard-to-access ILP. It has a large issue-queue and yet has limited issue width. This allows for the propagation delay of the wakeup-select logic to be focused on looking further ahead in the dynamic instruction stream to find the limited ILP. Core-B, on the other hand, provides higher performance to applications that have easier accessible local ILP. It has a smaller issue-queue, yet it has wider issue width. This allows for the propagation delay of the wakeup-select logic to be focused on issuing more instructions per cycle. Figure 6 shows the average execution time of Core-A and Core-B across all the benchmarks, normalized to that of Core-U.

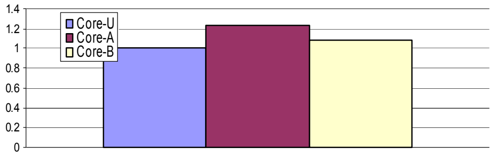


Figure 6. Average execution time across all benchmarks for different core designs normalized to that of Core-U.

Note that Core-A and Core-B are not truly the best core designs for core-selectability. However, they do present a scenario in which the source of performance difference between the cores is more lucidly discernable. Moreover, the manner in which the different benchmarks display preference towards being executed on these cores provides a convenient scenario to explain a few factors that need to be considered in the incorporated core designs.

V. RESULTS: MULTITHREADED

Figure 7 shows the execution time of individual benchmarks on Core-A and Core-B normalized to the execution time on Core-U. These results show that while Core-U displays the best overall performance across all benchmarks, it can display considerably suboptimal performance under individual benchmarks. The results also show that for all the benchmarks, other than RayTracer, either Core-A or Core-B performs better than Core-U. In addition, for all benchmarks, other than the biometric benchmark (Blast), either Core-A or Core-B performs worse than Core-U.

Core-selectability allows for the user to dynamically pick and choose the employed core design. Thus, a two-way core-selectable design that employs Core-A and Core-B, will be able to perform better than Core-U across almost all benchmarks. However, for the benchmark RayTracer, this solution will perform 20% worse than Core-U alone. This highlights the importance of good design space exploration for the employed cores. Ideally, the core designs should provide higher performance than the best single design to *collectively exhaustive* subsets of the workload space. In addition, the more *mutually exclusive* the subsets are, the more potential there will be for performance gain.

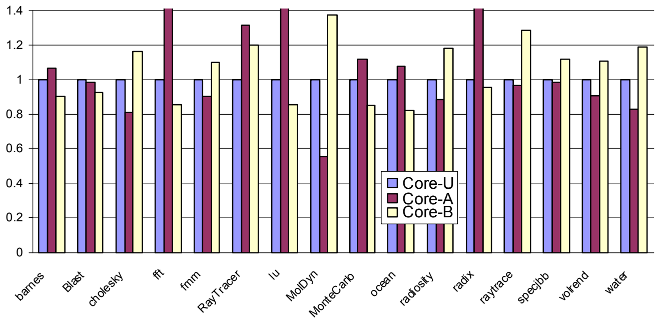


Figure 7. Execution time of each benchmark on each core design normalized to the execution time on Core-U.

The benchmarks for which Core-A and Core-B provide higher performance than Core-U are almost mutually exclusive and almost collectively exhaustive. Nevertheless, it may be of importance that absolutely none of the benchmarks display lower performance than the baseline design (even if average performance is improved). If that is the case, the core design space needs to be explored more rigorously. One option is to increase the number of selectable cores, and employ one that is customized for RayTracer. A more straightforward solution is to employ Core-U as one of the selectable core designs.

Figure 8 shows the average execution time for three combinations of two-way core-selectable designs normalized to the execution time of the baseline design, Core-U. These results show that the best overall performance for two-way core-selectability is attained with Core-A and Core-B, showing an average performance enhancement greater than 10%. Core-selectability between Core-U and Core-B, will not display performance lower than the baseline design on any benchmark. But it attains a smaller overall performance enhancement, of close to 7%, compared to Core-U alone.

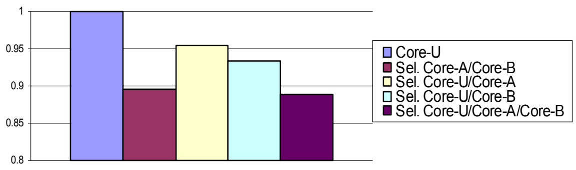


Figure 8. Execution time of core-selectability with different combinations of cores, normalized to that of Core-U.

Another option is core-selectability between all three core designs (Core-A, Core-B and Core-U). The average decrease in execution time for this scenario (also shown in Figure 8) is over 11%, while not displaying performance less than the baseline design under any of the benchmarks. However, increasing the number of selectable cores will increase the overhead in propagation delay.

A. *The Overhead of Selectability*

As discussed in the implementation section, core-selectability can introduce minor overheads to the front-end and back-end of the core designs.

Accounting for the multiplexing of address signals to the L1 data cache and the extra wire load, the propagation delay in the back-end was found to not be increased by more than 26ps. This delay is conveniently less than the slack observed in the L1 cache access time for all core designs. Although this can not be generalized to an arbitrary design, cache access time does vary in steps (with the associativity or number of sets) and the optimal core clock period is dependent on many microarchitectural parameters.

We do not repeat such an evaluation here, as the effect of increased propagation delay in the front-end has been studied elsewhere, and the core designs considered here are not the very best for core-selectability anyway.

B. *The Source of Performance*

We investigated the source of performance enhancement by looking at the code of the benchmark that displays the largest performance gain from a customized core design, Moldyn (from Java-Grande). This benchmark simulates molecular dynamics. The main program loop of the application performs force calculations between only particles that are within a certain distance of one another. There is little local ILP within each iteration of the main loop of the application. But with a high enough window it is possible to reach future iterations [38], which are mostly independent. It is for this reason that the application gains such large performance through increasing the issue-queue size at the cost of narrower issue width.

One may argue that this form of distant parallelism may be better extractable with simultaneous threads [8]. But we argue that the performance gain of core-selectability is orthogonal to that of simultaneous multithreading. This is because, even with simultaneous multithreading (SMT), upsizing the issue-queue will yield better performance with such a benchmark (compared to the best design across all benchmarks), as it will enable the extraction of parallelism within individual threads. Of course, this is assuming that the scalability of the application is not such that SMT just happens to finish off all remaining parallelism. However, due to the lack of simultaneous multithreading in the OPAL simulator, we were unable to quantitatively verify this.

Although there is thread-level parallelism in this application, it has been shown that it is difficult to extract without speculation support [38]. Core-selectability enables extraction of this parallelism without burdening the design components that affect general performance.

VI. RESULTS: MULTIPROGRAMMED

At a higher level, core-selectability can also be viewed as providing selectability between a homogeneous or heterogeneous multi-core design.

As pointed out in the related work section, it has been shown that a heterogeneous multiprocessor design can provide better throughput in a multiprogrammed environment. But, it has also been shown that heterogeneity can degrade the performance of multithreaded workloads. Therefore, a design that can transform between heterogeneity and homogeneity will have a degree of robustness in performance that is unachievable by other design solutions. In this section we investigate the potential performance benefit of such robustness.

Note that, from a stochastic standpoint, what renders multithreaded applications unsuitable for heterogeneity is the fact that they cause tasks (i.e., threads) with the same workload behavior to arrive at the system in bursts. This is opposed to the more normal distribution of task arrival in a multiprogrammed environment (see [35] for a more detailed study of the importance of accurately accounting for the task arrival pattern).

A. *Methodology*

In order to stochastically model a multiprogrammed environment, we simulate the queuing and occupation of different processing cores for different workload types. Tasks of different workload types are generated according to a random process with a normal distribution.

Tasks are primarily placed in the dedicated task queue of the core with the most suitable microarchitecture for the task's workload type. If the most suitable core is occupied, the task is directed to the next best core. If all cores are in use, the task waits for the most suitable core to become available. When there are cores with the same architectural configuration in the system (homogeneous), tasks are randomly assigned to them based on availability. Once the core is free, the task at the head of the task queue is consumed by the core for a given amount of time.

The amount of time it takes each task to be executed by a specific core depends on the design of the core and the task workload behavior. In this analysis, the workload behavior that a task may display is limited to that of the Simpoints [41] of the integer SPEC2000 suite of benchmarks. The considered core designs are the same three derived in Section IV (Core-U, Core-A and Core-B), but simulated with *sim-mase* from the Simplescalar V4.0 toolset [36].

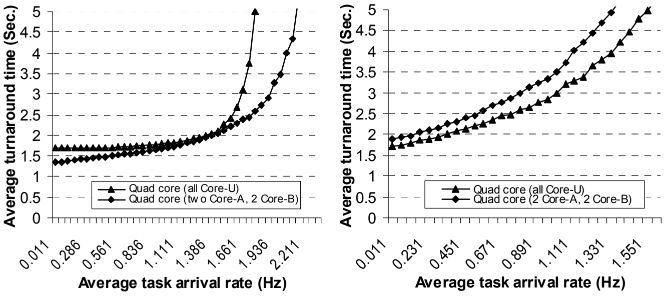
All tasks are considered to consist of 3.2 billion instructions no matter what the workload type. The amount of time a core is occupied by a task is determined by the rate with which the task's workload type is executed on the microarchitectural configuration of that core. As an example, a task that executes on a specific core design at a rate of β instructions per nanosecond is executed on that core in 3.2÷β seconds.

B. *Evaluation Results*

In order to evaluate the potential of core-selectability under different task arrival patterns, we compare the task turnaround time of systems with different combinations of core designs. Figure 9.a displays the average turnaround time of tasks submitted to two such quad-core systems. One system is homogeneous, and consists of four cores of the Core-U design. The other is heterogeneous, and consists of two cores of the Core-A design and two cores of the Core-B design. Results are presented across the spectrum of the task arrival rate, from low-contention to saturation. Tasks of different workload types arrive independently of each other. Thus, this task arrival pattern can be considered to be more representative of a multiprogramming environment.

These results show that the heterogeneous design results in around 25% lower task turnaround time in low arrival rates compared to the homogeneous design. Moreover, it displays roughly 14% higher execution bandwidth (the task arrival rate at which task turnaround time increases unboundedly). Therefore, employing 2-way core-selectability can provide such performance enhancement to multiprogramming environments. This is while, contrary to a fixed heterogeneous design, core-selectability does not degrade the performance of multithreaded applications, as it can switch back to a homogeneous design (and even provide higher performance than a fixed homogeneous design, as observed in the prior section).

For comparison, Figure 9.b displays the average turnaround time of tasks submitted to the same two differently designed quad-core systems. Here, however, tasks of different workload types arrive in bursts of four tasks of the same workload type. Thus, this task arrival pattern can be considered to be more representative of a multithreaded environment. These results show that under this task arrival pattern the heterogeneous design results in 10% higher task turnaround time in low arrival rates, and lower execution bandwidth, compared to the homogenous design. A core-selectable design enables the system to transform into the best quad-core solution for the task arrival pattern at hand.



(a) (b)

Figure 9. Average task turnaround time for (a) normal traffic, and (b) bursty traffic.

VII. Discussion

A. *The Limitations of this Study*

The evaluation results presented here are by no means conclusive enough to place a solid verdict on the notion of core-selectability, as the results may potentially be biased by some subtle circuit-level details. Nevertheless, we believe that in conjunction with the qualitative merits outlined here, a compelling enough argument emerges to warrant further investigation of this technique.

It is also important to note that the results presented here do not represent an upper bound on the performance enhancement attainable from core-selectability. For instance, the customized core designs we consider display limited diversity (for demonstration purposes and limitations in simulation). In actual implementation, not only can the cores differ in their microarchitectural parameters, but also in fundamental functionality and even ISA.

A chief benefit of core-selectability is the fact that it allows for the instruction-level performance enhancement of different types of workload behavior to be separated from each other. This is an aspect of core-selectability that we do not quantitatively evaluate here, as it pertains to design and verification effort, which is challenging to quantify.

We do not present results for a head-on performance comparison of core-selectability with the option of using the die area of the added cores for other purposes, e.g., more cache. Although such comparison has become commonplace in the academic arena, we believe that the assumption that it is die area that limits the sizing of structures is out-dated. If added cache were to be of performance benefit, it would already be provisioned in a well-designed baseline system. If it is not, it is because the increase in access latency would outweigh the benefit. Note that in the evaluations caches equivalent to those of modern processors are employed.

What does need to be evaluated in future work is the potential of core-selectability in the presence of simultaneous multithreading for multithreaded workloads. Although core-selectability is aimed at enhancing the extraction of ILP, a portion of this parallelism may be extractable through fine-grain threading.

An intriguing twist, that is not investigated here, is to employ core-selectability in conjunction with adaptable caches and cores with different clock domains. This can allow for much broader diversity in the design of the selectable core designs without introducing slack to the cache accesses.

B. *Future Trends*

The value of core-selectability is more evident when taking into consideration a number of technology trends.

Core-selectability exploits the increasing trend of transistors available on dies. It also does not exacerbate the trend of increasing power consumption and verification effort that alternative approaches to performance enhancement entail. This is because it enables the separation of the circuitry and design complexity necessary for dealing with different types of workload behavior, without drastically adding to the interconnection complexity.

Figure 10 shows the decreasing trend in the portion of die area consumed by the actual processing core(s) in chip multiprocessors over the past years. With the continuation of this trend, the viability of employing core-selectability will increase. A recent study by Rogers et al. [37] also forecasts a shrinking trend in the aggregate die area consumed by the cores in chip multiprocessors.

C. *Potential Drawbacks*

There are two main potential drawbacks to the implementation of core-selectability.

One is the added cost of engineering multiple core designs. The effort of designing a single core that has been tweaked to attain high performance across a wide range of applications may turn out to be less than that of designing multiple cores that are customized to specific workload behaviour, although not tweaked as much.

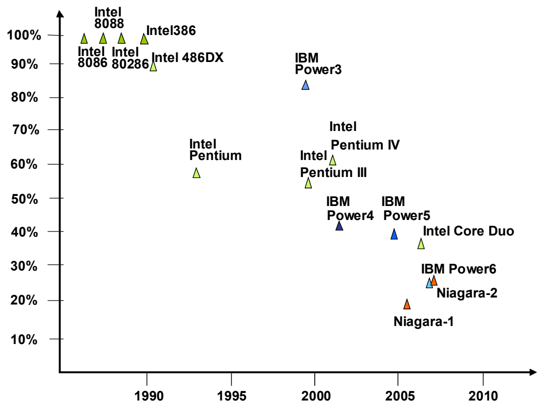


Figure 10. Percentage of die area consumed by the processing cores in commercial microprocessors over the past years*.

The other potential drawback is the added propagation delay at the shared ports to the system. Although with the technology library and the detailed baseline architecture employed in the evaluations of this study, the extra propagation delay was of negligible impact, it is not out of the question that it may be problematic in other settings. Whether the performance gain of core-selectability exceeds the potential loss due to increased propagation delay at the shared ports is a question that needs to be addressed in the context of the characteristics of the technology and development setup in which it is to be implemented.

VIII. Conclusion

This study investigates a potential solution to increasing the utilization of existing provisioning in the cache and interconnection resources of a chip multiprocessor. The approach is to place a number of differently designed cores (with different ILP-extracting units) within each node, and provide the ability to select which core to use depending on the characteristics of the applications at hand.

With the technology library considered in this study, it is shown that employing this technique with two core designs that focus on different ILP behavior, can result in better performance across a wide range of parallel applications, compared to a conventional design employing the best core design for overall performance across the same benchmarks. It is also shown that this design solution can provide greater throughput under multi-programmed workloads, by enabling the system to transform into a heterogeneous design when needed, i.e., providing selectability between homogeneity and heterogeneity.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. Muawya Al-Otoom and Greg Byrd also provided important feedback to this study. Wim Heir-

[*] These estimates may be somewhat subjective as they were extracted by looking at the die photographs of the different processors, distinguishing the actual cores from the non-core (including all levels of cache), and then measuring the proportional area.

man of Ghent University provided much needed assistance in running the multithreaded benchmarks on Simics.

This research was supported in part by NSF grant No. CCF-0811707, and funding from Intel and IBM. 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] S. Hauck, A. DeHon, “Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing,” *Morgan Kaufman*, 2008.

[2] D. Albonesi. “Dynamic IPC/clock rate optimization,” In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA)*, 1998.

[3] C. Kim, S. Sethumadhavan, M.S. Govindan, N. Ranganathan, D. Gulati, D. Burger and S.W. Keckler, “Composable Lightweight Processors”, In *Proceedings of the Int’l Symposium on Microarchitecture (MICRO)*, 2007.

[4] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, “Core Fusion: Accommodating software diversity in chip multiprocessors”, In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA)*, 2007.

[5] A. Lungu, D. J. Sorin, “Verification-Aware Microprocessor Design”, In *Proceedings of the Int’l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2007.

[6] U Gajanan, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, D. Greenhill, “Implementation of an 8-core, 64-Thread, Power-Efficient SPARC Server on a Chip”, *IEEE Journal of Solid-State Circuits, Vol. 43, No. 1*, 2008.

[7] Rakesh Kumar, Norman P. Jouppi, Dean M. Tullsen, “Conjoined-core Chip Multiprocessing” In *Proceedings of Int’l Symposium on Microarchitecture (MICRO)*, 2004.

[8] D. Tullsen, S. Eggers, and H. Levy, “Simultaneous multithreading: Maximizing on-chip parallelism”, In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA)*, 1995.

[9] L. Hammond, B. A. Nayfeh, and K. Olukotun, “A single-chip multiprocessor”, In *Computer, volume* 30*, no. 9,* Sept. 1997.

[10] R. Dolbeau and A. Seznec, “CASH: Revisiting hardware sharing in single-chip parallel processor”, In *Journal of Instruction-Level Parallelism (JILP), vol. 6,* April 2004. (www.jilp.org/vol6).

[11] R. Kumar, D. M. Tullsen, P. Ranganathan, N. Jouppi, K. I. Farkas, "Single-ISA Heterogeneous Multicore Architectures for Multithreaded Workload Performance", In *Proceedings of the Int’l Symp. on Computer Architecture (ISCA)*, 2004.

[12] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, D. M. Tullsen, “Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction”, In *Proceedings of the Int’l Symposium on Microarchitecture (MICRO)*, 2003.

[13] H. H. Najaf-abadi and E. Rotenberg, “Architectural Contesting”, In *Proceedings of the Int’l Symposium on High-Performance Computer Architecture (HPCA)*, 2009.

[14] Saisanthosh Balakrishnan , Ravi Rajwar , Mike Upton , Konrad Lai, “The Impact of Performance Asymmetry in Emerging Multicore Architectures”, In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA)*, 2005.

[15] R. E. Kessler, “The Alpha 21264 Microprocessor”, In *IEEE Micro, v.19 n.2*, March 1999.

[16] http://www.crhc.illinois.edu/ACS/tools/ivm/about.html

[17] http://www.opencores.org

[18] http://www.opensparc.net

[19] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, R. Jenkal, "FreePDK: An Open-Source Variation-Aware Design Kit", In *Proceedings of the Int’l Conference on Microelectronic Systems Education (MSE'07)*, 2007.

[20] M. Frank, W. Lee, S. Amarasinghe, “A software framework for supporting general purpose applications on Raw computation fabric”, *MIT-LCS Technical Memo MIT-LCS-TM-619*, 2001.

[21] J. Koppanalil, P. Ramrakhyani, S. Desai, A. Vaidyanathan, and E. Rotenberg. “A Case for Dynamic Pipeline Scaling”, In *Proceedings of the Int’l Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02)*, 2002.

[22] R. Berridge et. al. “IBM POWER6 microprocessor physical design and design methodology”, In *IBM Journal of Research and Development, Volume 51, Issue 6*, Nov. 2007.

[23] P. Salverda and C. Zilles. “Fundamental performance constraints in horizontal fusion of in-order cores”, In *Proceedings of the Int’l Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[24] N. Choudhary et al., "FabScalar", In *the Workshop on Architecture Research Prototyping (WARP)*, 2009.

[25] http://www.virtutech.com

[26] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood, “Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset”, In *SIGARCH Computer Architecture News*, v.33 n.4, 2005.

[27] J. Emer et al., “Single-threaded vs. Multithreaded: Where should we focus?”, *IEEE Micro, vol. 27, no. 6,* Nov./Dec. 2007.

[28] S. Chaudhry et al., “Rock: A high-performance SPARC CMT processor”, *IEEE Micro, vol. 29, no. 2,* Mar./Apr. 2009.

[29] M. A. Suleman, O. Mutlu, M. K. Qureshi, Y. N. Patt, “Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures”, In *Proceedings of the Int’l Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, 2009.

[30] R. Kumar, D. M. Tullsen, N. P. Jouppi “Core architecture optimization for heterogeneous chip multiprocessors”, In *proceedings of the Int’l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[31] H. H. Najaf-abadi, E. Rotenberg, “Configurational Workload Characterization”, In *Proceedings of the Int’l Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2008.

[32] R. Balasubramonian and D. Albonesi, “Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures”, In *Proceedings of the Int’l Symposium on Microarchitecture (MICRO)*, 1999.

[33] R. Hum, “How to Boost Verification Productivity”, *EETimes*, January, 2005.

[34] S. G. Dropsho, G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, “Dynamically Trading Frequency for Complexity in a GALS Microprocessor”, In *Proceedings of the Int’l Symposium on Microarchitecture (MICRO)*, 2004.

[35] H. H. Najaf-abadi, E. Rotenberg, “The Importance of Accurate Task Arrival Characterization in the Design of Processing Cores”, In *Proceedings of the IEEE Int’l Symposium on Workload Characterization (IISWC)*, 2009.

[36] E. Larson, S. Chatterjee, T. Austin, “The MASE Microarchitecture Simulation Environment”, In *Proceedings of the Int’l Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2001.

[37] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, Y. Solihin, “Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling”, In *Proceedings of the Int’l Symposium on Computer Architecture (ISCA)*, 2009.

[38] M. Chen, K. Olukotun, “TEST: A Tracer for Extracting Speculative Threads”, In *Proceedings of the Int’l Symposium on Code Generation and Optimization (CGO)*, 2003.

[39] M. Frank, C. A. Moritz, B. Greenwald, S. Amarasinghe, A. Agarwal, “SUDS: Primitive Mechanisms for Memory Dependence Speculation”, *MIT/LCS Technical Memo MIT-LCS-TM-591*, 1999.

[40] R. Calkin, R. Hempel, H. Hoppe, P. Wypior, “Portable programming with the PARMACS Message-Passing Library”, In *Proceedings of the Parallel Comput., Special Issue on Message-Passing Interfaces*, 1994.

[41] T. Sherwood, E. Perelman, G. Hamerly, B. Calder, “Automatically Characterizing Large Scale Program Behavior,” In *proceedings of the Int’l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.