Power Efficient Resource Scaling in Partitioned Architectures through Dynamic Heterogeneity

Naveen Muralimanohar, Karthik Ramani, Rajeev Balasubramonian
School of Computing, University of Utah *
{naveen, karthikr, rajeev}@cs.utah.edu

Abstract

The ever increasing demand for high clock speeds and the desire to exploit abundant transistor budgets have resulted in alarming increases in processor power dissipation. Partitioned (or clustered) architectures have been proposed in recent years to address scalability concerns in future billion-transistor microprocessors. Our analysis shows that increasing processor resources in a clustered architecture results in a linear increase in power consumption, while providing diminishing improvements in single-thread performance. To preserve high performance to power ratios, we claim that the power consumption of additional resources should be in proportion to the performance improvements they yield. Hence, in this paper, we propose the implementation of heterogeneous clusters that have varying delay and power characteristics. A cluster's performance and power characteristic is tuned by scaling its frequency and novel policies dynamically assign frequencies to clusters, while attempting to either meet a fixed power budget or minimize a metric such as $Energy \times Delay^2$ *(*ED^2*). By increasing resources in a power-efficient manner, we observe a 11% improvement in* ED^2 *and a 22.4% average reduction in peak temperature, when compared to a processor with homogeneous units. Our proposed processor model also provides strategies to handle thermal emergencies that have a relatively low impact on performance.*

Keywords: *partitioned (clustered) architectures,* $Energy \times Delay^2$*, temperature, dynamic frequency scaling.*

1 Introduction

Recent technology trends have led to abundant transistor budgets, high clock speeds, and high power densities in modern microprocessors. Simultaneously, latencies of on-chip and off-chip storage structures (caches, memories) have increased relative to logic delays. If architects make no attempt to hide these long latencies, clock speed improvements do not translate into significant performance improvements. Long latencies can be tolerated effectively via a number of strategies, such as out-of-order execution with a large in-flight instruction window, data prefetching, etc. However, the quest for instruction-level parallelism (ILP) often succumbs to the law of diminishing returns. Once the low-hanging fruit has been picked, transistors allocated for ILP yield marginal improvements, but expend significant energy. This paper investigates if abundant transistor budgets can be exploited without causing inordinate increases in power density.

Partitioned architectures [2, 6, 7, 12, 20] have been proposed in recent years to allow processor resources to scale up without impacting clock speed or design complexity. A partitioned architecture employs small processing cores (also referred to as clusters) with an interconnect fabric, and distributes instructions of a single application across the processing cores. The small size of each core enables low design complexity and fast clock speeds, and independent dependence chains executing on separate clusters enable high parallelism. Of course, most applications cannot be decomposed into perfectly independent dependence chains. This results in significant amounts of data being communicated between clusters.

Even though a partitioned architecture is more scalable than a monolithic architecture, the addition of more resources will eventually yield marginal improvements in ILP (if at all). On the other hand, the addition of resources inevitably leads to a linear increase in power consumption. Figure 1 shows the improvement in instructions per cycle (IPC) achieved by adding more clusters. It also shows the power overhead incurred by adding more clusters (the simulated processor model and the workload are described in detail in Section 4). Improvement in IPC is not commensurate with the increase in processor power, and aggressive designs with numerous resources can have very poor $Energy \times Delay$ and $Energy \times Delay^2$ (ED^2) characteristics. In this paper, we argue that if additional resources are likely to yield marginal IPC improvements, they must also incur marginal power overheads. To achieve this goal, we propose the design of heterogeneous clusters, where differ-

*This work was supported in part by NSF grant CCF-0430063.

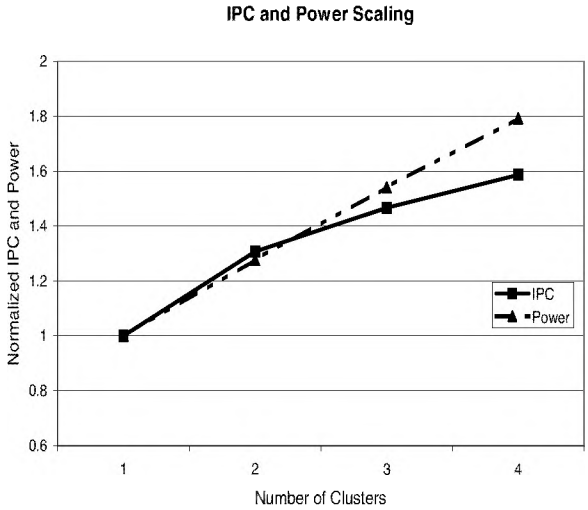


Figure 1. IPC and Power as number of clusters increases, normalized to the 1-cluster system

ent clusters are optimized for either performance or power. Such an approach can entail significant design complexity in a conventional monolithic superscalar architecture. For example, accommodating different register implementations within a single register file can pose circuit timing issues and pipeline scheduling difficulties in a monolithic superscalar. A partitioned architecture, on the other hand, is modular enough that the properties of one cluster do not influence the design of another cluster.

Further, we advocate dynamic heterogeneity – frequency scaling allows a cluster to dynamically serve as either a high-performance or low-power cluster. We propose a dynamic adaptation policy that takes advantage of program metrics that estimate the performance potential of additional resources and selects a cluster configuration with minimum ED^2. We observe ED^2 improvements of 11%, compared to a system comprised of homogeneous clusters. If the processor must meet a fixed power budget, we observe that a heterogeneous configuration is often able to maximize performance while meeting this power budget. By allowing a cluster to alternate between high and low-power mode, operating temperature on a chip is reduced, leading to lower leakage power dissipation and fewer thermal emergencies. Heterogeneous clusters also present the option of handling a thermal emergency in a manner that minimally impacts performance.

In summary, this paper presents low-complexity novel proposals (heterogeneous clusters and dynamic resource allocation) and detailed evaluations that demonstrate significant ED^2 and temperature benefits. These innovations are especially important when single-threaded workloads execute on aggressive microprocessor designs that yield diminishing IPC improvements. The paper is organized as follows. Section 2 reviews our base architecture model. Section 3 discusses the design of heterogeneous clusters and our novel dynamic adaptation policy. Section 4 evaluates the impact of our proposed techniques on ED^2 and temperature. Finally, Section 5 discusses related work and we conclude in Section 6.

2 The Base Clustered Processor

A variety of architectures have been proposed to exploit large transistor budgets on a chip [2, 6, 7, 8, 12, 18, 20, 28, 29, 33, 35, 39]. Most proposals partition the architecture into multiple execution units and allocate instructions across these clusters, either statically or dynamically. For our evaluations in this study, we employ a dynamically scheduled clustered architecture. This model has been shown to work well for many classes of applications with little or no compiler support.

2.1 The Centralized Front End

As shown in Figure 2, instruction fetch, decode, and dispatch (register rename) are centralized in our processor model. During register rename, instructions are assigned to one of four clusters. The instruction steering heuristic is based on Canal *et al.*'s ARMBS algorithm [13] and attempts to minimize load imbalance and inter-cluster communication. For every instruction, we assign weights to each cluster to determine the cluster that is most likely to minimize communication and issue-related stalls. Weights are assigned to a cluster if it produces input operands for the instruction. Additional weights are assigned if that producer has been on the critical path in the past [44]. A cluster also receives weights depending on the number of free issue queue entries within the cluster. When dispatching load instructions, more weights are assigned to clusters that are closest to the data cache. Each instruction is assigned to the cluster that has the highest weight according to the above calculations. If that cluster has no free register and issue queue resources, the instruction is assigned to a neighboring cluster with available resources.

2.2 The Execution Units

Our clustered architecture employs small computation units (clusters) that can be easily replicated on the die. Each cluster consists of a small issue queue, physical register file, and a limited number of functional units with a single cycle bypass network among them. The clock speed and design complexity benefits stem from the small sizes of structures within each cluster. Dependence chains can execute quickly if they only access values within a cluster. If an instruction sources an operand that resides in a remote register file, the register rename stage inserts a "copy instruction" [13] in the producing cluster so that the value is moved to the consumer's register file as soon as it is produced. These register value communications happen over longer global wires and can take up a few cycles. Aggarwal and Franklin [2] show that a crossbar interconnect performs the best when

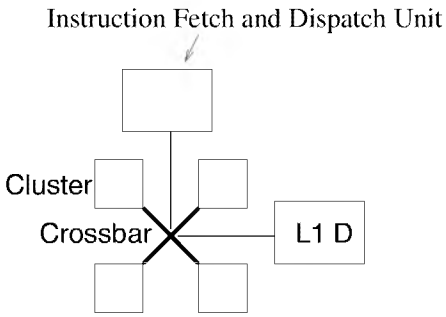


Figure 2. A partitioned architecture model with 4 clusters and a centralized frontend.

connecting a small number of clusters (up to four), while a hierarchical interconnect performs better for a large number of clusters. Even though future aggressive systems are likely to employ many clusters, we expect that the clusters will be partitioned among multiple threads. Only a subset of programs show significant IPC benefits by employing more than four clusters and even this improvement may not warrant taking resources away from other threads. In the common case, we expect that most single-threaded programs will execute on four clusters and we therefore restrict ourselves to systems with fewer than four clusters that employ a crossbar interconnect.

Earlier studies have shown that the performance improvement from a distributed cache may not warrant the implementation complexity [5, 25, 38]. We implement a centralized cache and load/store queue (LSQ) that are also accessible through the crossbar router (Figure 2). Load and store instructions are assigned to clusters, where effective address computation happens. The effective addresses are sent to the centralized LSQ and cache. The LSQ checks for memory dependences before issuing a load and returning the word back to the requesting cluster.

3 Heterogeneous Clusters

Partitioned or clustered architectures have favorable scalability properties. As transistor budgets increase, we can increase the number of clusters while only impacting the complexity of the dispatch stage. A higher number of clusters enables larger in-flight instruction windows, thereby potentially boosting ILP. However, as is true with most ILP techniques, this strategy quickly yields marginal returns. As Figure 1 showed us, the IPC jump in moving from one to two clusters is 31%, but the jump in moving from three to four clusters is only 8%. Additional resources either remain under-utilized or are busy executing instructions along mis-predicted paths. Unfortunately, the power overhead of each additional cluster is roughly constant. Moreover, disproportionate utilization rates of different processor structures lead to localized hotspots that severely degrade chip reliability and trigger thermal emergencies.

3.1 High-Performance (HPC) and Low-Power Clusters (LPC)

In order to better match the IPC and power curves, we propose adding resources in a power-efficient manner. Each resource can toggle between high-performance and low-power mode, thereby also improving thermal properties. We employ two different classes of clusters. The first class of clusters, named the high-performance and high-power clusters (HPCs), are designed to deliver high performance while consuming significant power. The second class of clusters, the low-power and low-performance clusters (LPCs), are frequency scaled versions of the HPCs. They operate at a frequency that is lower than that of the HPCs by a factor of four. In some cases (thermal emergencies or high inter-cluster communication), we even completely turn a cluster off by gating its supply voltage. An important attribute of partitioned architectures is low design complexity because each cluster can be created by replicating a single cluster's design. By incorporating heterogeneity, we are not compromising this advantage. Each cluster has an identical design and heterogeneity is provided through frequency scaling.

We allow each cluster's clock to be independently controlled, resulting in at least four distinct clock domains. The clocks may or may not originate from the same source. Either way, the interface between clock domains does not impose a significant performance or power overhead [40, 42]. This interface is implemented at the crossbar router. Each link operates at the frequency of the cluster it is connected to. The buffers in the router can be read and written at edges belonging to different clocks. Stall cycles are introduced if the clock edge that wrote the result is not separated from the clock edge that attempts to read the result by a minimum synchronization delay. Stalls usually happen only if the buffer is nearly empty or nearly full. If the clocks originate from a single source and are integral multiples of each other (as is the case in our simulations), these synchronization delays are rare. If the processor is forced to employ multiple clock sources due to clock distribution problems, these synchronization delays are introduced in the base processor as well. Even though one of the links may be draining a buffer only once every four (fast) cycles, the buffer size is bounded. Each instruction in a cluster can generate no more than three input data communications and the number of instructions in a cluster is bounded by its window size.

Dynamic frequency scaling (DFS) is a well-established low-overhead technique that is commonly employed in modern-day microprocessors [17, 21]. Most implementations allow the processor to continue executing instructions even as the clock speed is being altered. In our case, we only toggle between two frequencies (5 GHz and 1.25 GHz) and a change happens once every few million instructions, on average. This amounts to a negligible overhead even if we

assume that the processor is stalled for as many as thousand cycles during a frequency change. Note that some stalls are introduced because in-flight packets must be routed before the frequency change can happen. Not doing this could result in register tags arriving at the wrong time, thereby leading to scheduling problems. Further, if a cluster is being turned off, architectural registers in that cluster will have to be copied elsewhere. We will assume that turning a cluster on and off with power-gating [1] can also happen within the assumed thousand cycle overhead. In tandem with frequency scaling, voltages can be scaled down as well, allowing significant power savings. However, voltage scaling has higher overheads and we will restrict ourselves to the more complexity-effective approach of dynamic frequency scaling for all of our results. With DFS, each cluster can independently switch from being an HPC to an LPC, or vice versa. Next, we will describe how various metrics are considered in determining the frequency for each cluster.

3.2 Dynamic Adaptation Mechanism

Before devising a resource allocation mechanism that balances performance benefits and power overheads, we must define the metric we are attempting to optimize. $Energy \times Delay^2$ or ED^2 is commonly acknowledged to be the most reliable metric while describing both performance and power [9, 31]. Not only does it reflect marketing goals (performance is slightly more important than energy), it also has favorable theoretical properties. The ED^2 equation can be expressed as follows (α is activity factor, C is capacitance, f is frequency, and V is voltage):

$$ED^2 = Power \times D^3 = \alpha C f V^2 D^3$$

If we assume a linear relationship between the speed and voltage of a circuit, we see that the ED^2 term is an invariant when voltage and frequency scaling is applied to the processor. Put differently, $(ED^2)^{\frac{1}{3}}$ or $(PD^3)^{\frac{1}{3}}$ is a measure of performance for different processors that are all voltage-and-frequency scaled so as to consume an equal amount of power. For example, if V is reduced to $0.9V$, then f is reduced to $0.9f$ and D is increased to $D/0.9$, resulting in the cancellation of the constant factors. A different metric, such as ED, may not be robust under such optimizations. With the ED metric, any processor (A) operating well above the threshold voltage can be shown to be "better" than any other processor (B) if it (A) is sufficiently voltage and frequency scaled. Similarly, ED^2 is not susceptible to frequency scaling tricks – ED^2 is minimized when the processor operates at its peak frequency. For all these reasons, our algorithm attempts to optimize the ED^2 metric. The linear relationship between circuit speed and voltage does not always hold in practice. Therefore, our results also present IPC and power values, allowing the interested reader to compute other relevant metrics.

Since each cluster can operate as an HPC, an LPC, or be completely turned off, the processor can be configured in many interesting ways. Figure 3 shows the ED^2 value for each benchmark program under various fixed configurations. For example, the bars denoted by *"3f"* refer to a processor model where three clusters operate at peak frequency while one cluster is turned off, and the bars denoted by *"3f1s"* refer to a processor model with three clusters at peak frequency (fast) and one cluster at $1/4^{th}$ the peak frequency (slow). For brevity, many of the configurations that yield uninteresting ED^2 values are left out. It is clear that a single statically chosen design point benefits only a subset of the applications while adversely affecting the performance/power characteristics of other applications. For example, the processor model with two fast clusters (*2f*) delivers low ED^2 while executing benchmarks such as gcc and twolf, but degrades performance significantly for galgel.

Benchmark	Distance between branch mispredicts	L2 miss rate	L1D miss rate	Nearly optimal ED^2 models	Base Case IPC (*4f*)
eon	57	0.14	0.72	*3f*	1.47
crafty	72	0.7	1.68	*2f 3f*	1.12
bzip2	53	2.5	3.5	*2f*	0.82
gap	65	11.77	0.67	*2f*	0.90
gcc	103	2.25	3.38	*2f*	0.75
gzip	129	0.21	6.56	*2f*	0.73
parser	80	4.35	3.83	*2f*	0.78
twolf	60	0.31	7.28	*2f*	0.77
ammp	289	0.79	6.63	*4f*	1.74
apsi	360	34.8	2.77	*4f*	1.85
vortex	183	3.521	1.44	*4f*	1.78
wupwise	170	20.61	2.22	*4f*	1.92
galgel	356	1.24	7.19	*3f, 4f*	2.63
mesa	204	5.73	0.63	*3f 4f*	2.35
vpr	161	11.74	5.67	*2f 3f*	0.51
mgrid	18430	12.29	6.62	*4f 4s*	1.43
fma3d	544	22.28	6.48	*4s*	1.45
equake	506	16.44	17.02	*4s*	0.6
lucas	1628292	16.67	20.52	*4s*	0.5
swim	33895	13.76	20.19	*4s*	1.29
art	97	7.8	41.39	*3f1s*	0.79
applu	55	23.03	9.76	*3f, 4f*	1.04
mcf	57	43.25	37.61	*4s, 1f3s*	0.17

Table 1. Average distance between branch mispredicts, L2 and L1D cache miss rates, configurations that yield ED^2 within 5% of the optimal ED^2, and raw IPC for baseline model with 4 HPCs.

Table 1 shows the average L1 data and L2 cache miss rates, average distance between branch mispredicts, and the configurations that yield ED^2 within 5% of the lowest ED^2 observed among all the statically fixed configurations (referred to as the optimal ED^2 for each benchmark). From this data, we take note of the following trends. (i) Programs (lucas, equake, etc.) with high data cache miss rates ($\geq$ 8% L1D miss rate and $\geq$ 12% L2 miss rate) benefit from using more clusters. More clusters imply more registers and issue queue entries, which in turn imply large in-flight windows that increase the chances of finding useful work while wait-

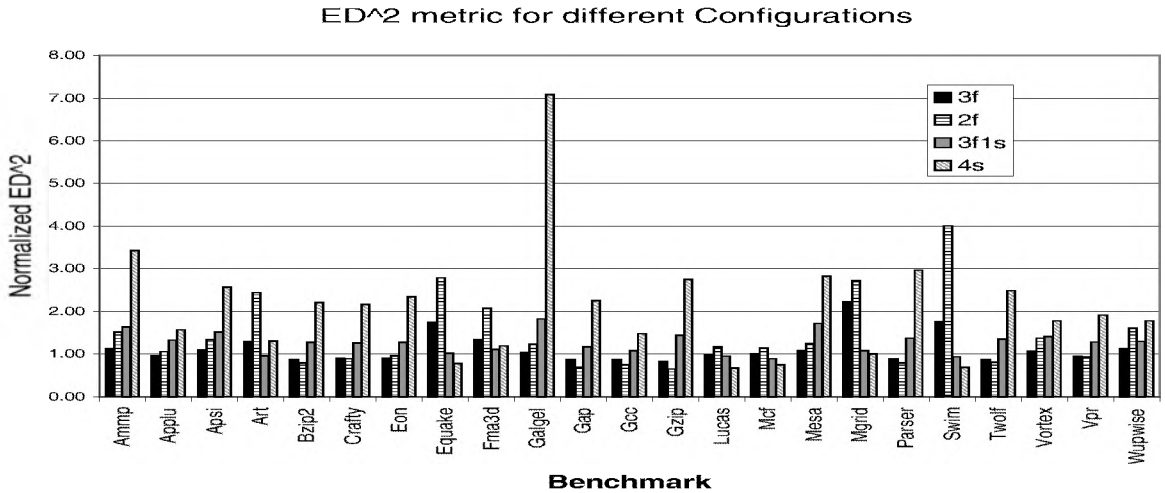


Figure 3. ED^2 **metric for different static configurations normalized to baseline system with 4 HPCs.**

ing for cache misses. There is usually not enough work to keep the processor busy for the entire duration of a cache miss, which implies that the execution units can execute at low frequencies and still complete their work before the cache miss returns. (ii) Programs (gcc, bzip2, etc.) with high branch misprediction rates (average distance between mispredicts being less than 100 instructions) benefit very little from large in-flight windows as most instructions in the window are likely to be on mispredicted paths. Executing instructions at a low frequency can severely degrade performance because it increases the time taken to detect a branch misprediction.

Clearly, branch mispredictions and cache misses are the biggest bottlenecks to performance and simply observing these metrics guides us in detecting the most "efficient" processor configuration. Based on the observations above, we identify the following relationships:

- *Low cache miss rates and high branch misprediction rates:* Both factors suggest using small in-flight windows with high frequencies. Minimum ED^2 was observed for configurations that employed only two clusters at peak frequency (*2f*).

- *Low cache miss rates and low branch misprediction rates:* Both factors indicate that the program has high ILP and could be severely degraded if we either reduced window size or frequency. The base case organization with four clusters operating at peak frequency (*4f*) yielded minimum ED^2.

- *High cache miss rates and low branch misprediction rates:* Both factors suggest the use of large windows and the first factor suggests the use of frequency-scaled clusters. A configuration such as *1f3s* or *4s* often yields the lowest ED^2 (*4s* was selected for our simulations).

- *High cache miss rates and high branch misprediction rates:* The two factors make conflicting suggestions for window size and frequency. Empirically, we observed that the performance penalty of low frequency and small window size was too high and the lowest ED^2 was observed for configurations that struck a balance, such as *3f1s*.

A small subset of processor configurations are sufficient to deliver optimal ED^2 values for most of the benchmarks. Of the many different configurations, our dynamic algorithm employs the following four configurations: *4s, 2f, 4f, 3f1s*. The reduction in the number of configuration choices available to the dynamic algorithm helps reduce its complexity. Only for four of the 23 benchmarks (eon, vpr, applu, mcf) does our heuristic select a configuration with ED^2 not within 5% of the optimal ED^2.

Many recent studies [4, 6, 19, 22, 23, 26, 37, 45] have examined algorithms for dynamic processor adaptation. Some of these algorithms make decisions at subroutine boundaries, some make decisions at specific events (cache misses, full buffers, *etc.*), while some make decisions at periodic intervals. We found that for our experiments, interval-based techniques entailed little complexity and reacted appropriately to phase changes. An interval is defined as the time taken to commit one million instructions (we also evaluate other interval sizes). At the end of every interval, hardware counters that track L1 and L2 cache miss rates and the number of branch mispredictions are examined. Depending on whether these counters exceed certain thresholds, one of the four configurations described above is selected and employed for the next interval. Since these counters directly indicate the optimal configuration, there is no need for an exploration process that profiles many candidate configurations and selects one. Since there is no exploration process, there is also no need to detect when the program moves to a new phase with radically different behavior.

Our algorithm is based on the premise that behavior in the last interval will repeat in the next interval. In the worst case, a program may exhibit dramatically different behavior

in every successive interval. This can be easily detected as it would result in configuration changes in every interval. The operating system can then choose to increase the interval length or shut off the dynamic adaptation mechanism. It must be noted that our adaptation algorithm simply selects the configuration that is likely to yield minimum ED^2 with little regard to the actual performance loss. If significant performance losses cannot be tolerated, the adaptation algorithm will have to be altered and this is discussed in a later section.

Once an appropriate configuration has been chosen, say *3f1s*, the following questions arise: (i) which cluster is assigned the low frequency, and (ii) what instructions are assigned to the slow cluster? The answer to the first question lies in the thermal properties of the processor and is discussed in the next sub-section. To address the second question, we attempted a number of steering mechanisms that take instruction criticality into account. We modeled per-instruction and branch-confidence based criticality predictors to identify instructions that are off the critical path and dispatch them to slow clusters. However, these steering policies inevitably increased the degree of inter-cluster communication, often resulting in contention cycles for other data transfers that were on the program critical path. Load balance was also affected as critical and non-critical instructions are rarely in the same ratio as the fast and slow clusters (3:1 in the case of *3f1s*). In spite of our best efforts, even complex alterations to the steering policy resulted in minor IPC improvements (if at all) over the base steering policy that only attempts to balance communication and load. We will only consider the base steering policy for the rest of the paper, which implies that some critical instructions may be executed on the slower clusters.

3.3 Thermal Emergencies

If each cluster operates at peak frequency, the register files in each cluster often emerge as the hottest spots on the chip. By allowing each cluster to occasionally operate at a lower frequency (or be turned off), the operating temperature at each register file is greatly reduced. If a configuration such as *3f1s* is chosen, a round-robin policy selects the cluster that is frequency-scaled every million cycles. There are two advantages to a lower operating temperature: (i) The number of thermal emergencies are reduced. On a thermal emergency, the processor must take steps to reduce power dissipation at the hot-spot, often leading to a performance penalty. (ii) The leakage power dissipation on a chip is an exponential function of the operating temperature. Even if thermal emergencies are not triggered, by reducing operating temperature, we can significantly reduce leakage power.

It is noteworthy that heterogeneous clusters provide interesting options in the event of a thermal emergency. If all four clusters are operating at peak frequency and a thermal emergency is triggered, a different configuration that yields a slightly higher ED^2 can be invoked. Since this configuration periodically shuts each cluster off or lowers its frequency, it can mitigate the thermal emergency without imposing a significant penalty in performance or ED^2. In Section 4, we show that such a thermal emergency strategy performs better than approaches for traditional superscalars, such as global frequency scaling.

3.4 Achieving Maximum Performance for a Fixed Power Budget

In some microprocessor domains, designers are most concerned with maximizing performance while staying within a fixed maximum power budget. Heterogeneous clusters allow low-overhead run-time flexibility that allows a processor to meet power budgets for each program. At the start of program execution, all four clusters operate at peak frequency. At the end of every million cycle interval, hardware counters that track average power dissipation or activity are examined. If the average power dissipation is greater than the power budget, some of the clusters will have to be either slowed down or turned off. Likewise, if the average power dissipation is well below the power budget, clusters can be either activated or have their frequency scaled up. The correlation between cache miss rates, branch mispredict rates, and optimal ED^2 configurations guides our adaptation policy again. For programs with frequent branch mispredicts that exceed the power budget, we choose to turn clusters off in steps until the power dissipation drops to acceptable levels. For all other programs, we choose to scale down the frequencies of clusters in steps until the power constraint is met.

Our mechanism, by using branch mispredict rates, tries to seek out configurations that are likely to have competitive ED^2 and meets the power constraint through low-overhead techniques that scale down frequency or turn a cluster off. Note that $(ED^2)^{\frac{1}{3}}$ or $(PD^3)^{\frac{1}{3}}$ is a measure of performance for different processors that are all voltage-and-frequency scaled so as to consume an equal amount of power. By employing configurations that have near-optimal ED^2, the adaptation mechanism achieves near-optimal performance at the fixed power budget and does so without employing the high overhead technique of voltage scaling.

4 Results

4.1 Methodology

Our simulator is based on SimpleScalar-3.0 [11] for the Alpha ISA. The cycle accurate simulator has been extended with a power model based on Wattch [10] to compute power for the entire processor, including clusters, interconnect, caches, and the centralized front-end. Leakage power for the caches and the register files follows a methodology similar to HotLeakage [46]. HotLeakage extends the Butts-Sohi leakage power model based on the BSIM3 [34] data and calculates leakage currents dynamically from temperature and voltage changes due to operating conditions. For

Fetch queue size	64	Fetch width	8 (across up to 2 basic blocks)
Branch predictor	comb. of bimodal and 2-level	Bimodal predictor size	16K
Level 1 predictor	16K entries, history 12	Level 2 predictor	16K entries
BTB size	16K sets, 2-way	Branch mispredict penalty	at least 12 cycles
Issue queue size	30 per cluster (int and fp, each)	Register file size	64 per cluster (int and fp, each)
Integer ALUs/mult-div	1/1 per cluster	FP ALUs/mult-div	1/1 per cluster
L1 I-cache	32KB 2-way	Memory latency	300 cycles for the first block
L1 D-cache	32KB 4-way set-associative, 6 cycles, 4-way word-interleaved	L2 unified cache	8MB 8-way, 30 cycles
		I and D TLB	128 entries, 8KB page size
Frequency	5 GHz	Initial Temperature	70 C
Convection resistance	0.8 K/W	Heatsink thickness	6.9 mm
Maximum temperature	85 C	Vdd	1.1 V
Load/Store queue	240 entries		

Table 2. Simulator parameters.

all other functional units, leakage power is calculated based on the formula derived by curve fitting with ITRS data [3].

Separate integer and floating point issue queues and register files are modeled for each cluster. Each of the clusters is assumed to have 64 registers (Int and FP, each), 30 issue queue entries (Int and FP, each) and one functional unit of each kind. Contention on the interconnects and for memory hierarchy resources (ports, banks, buffers, etc.) are modeled in detail. Inter-cluster communication through the crossbar incurs a two cycle delay (in addition to contention cycles) when all links are operating at peak frequency. This delay grows to five cycles if one of the clusters is operating as an LPC and to eight cycles if both clusters are operating as LPCs. For all processor models in this study, when frequency scaling is applied to clusters, the processor front-end continues to execute at peak frequency.

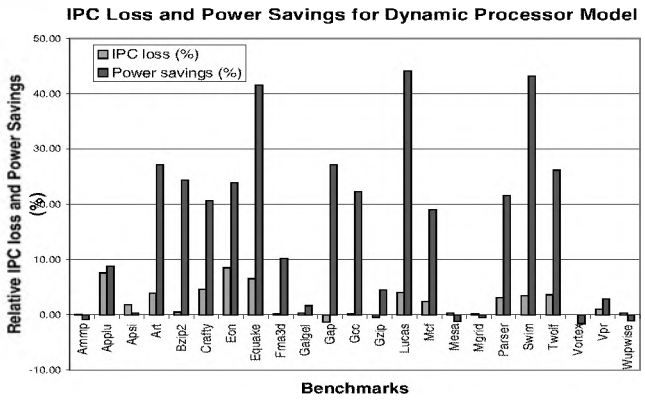
We use the Hotspot-2.0 [43] model to extend our Wattch-based simulator for sensing temperature of various units at 100K cycle intervals. This model exploits the duality between heat and electricity to model the temperature of the processor at a functional unit granularity. The average power of each of the units over recent cycles is supplied to the model to calculate the current temperature of the functional units from a known initial temperature. Heat removal is done via airflow by convection using an equivalent thermal resistance of 0.8K/W and an ambient temperature of 40 C is assumed. The floorplan layout of the processor models the four clusters along with the front end and follows the procedure indicated in [43]. At the beginning of the simulation, we assume that the processor has been operating for a long time, dissipating nominal dynamic and leakage power at an operating temperature of 70 C.

All important simulation parameters are shown in Table 2. For the 4-cluster system operating at peak frequency, our power models show an average processor power dissipation of 113W at 90nm technology, with each cluster accounting for approximately 17W, and leakage power accounting for 20W over the entire processor.

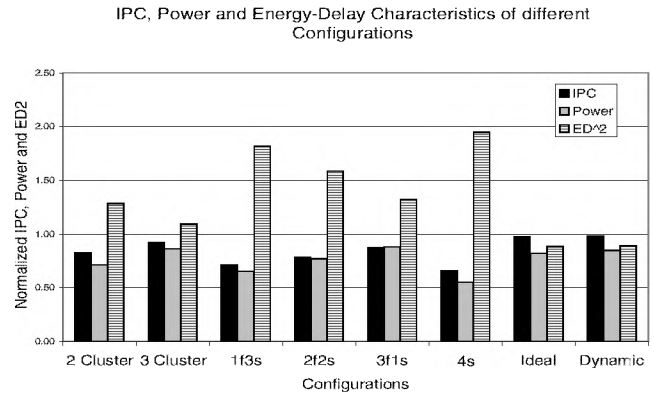
We use 23 of the 26 SPEC-2k programs with reference inputs as a benchmark set (the remaining three benchmark programs were not compatible with our simulator). Each program was simulated for 100 million instructions over simulation windows identified by the SimPoint toolset [41]. The raw IPCs for each program for the base case are listed in Table 1.

4.2 ED^2 Analysis

We begin by showing ED^2 improvements that can be achieved by employing the dynamic adaptation mechanism that tunes processor resources to suit application needs. The base case is a high performance model that employs four HPCs. The dynamic processor model examines L1 and L2 cache miss and branch misprediction counters in every interval to select the configuration for the next interval. Section 3.2 has already discussed the strong correlation between these metrics and the ED^2-optimal configuration. For our experiments, we assume an interval size of one million instructions. The thresholds for L1 and L2 cache miss rates is 8% and 12%, respectively, while the threshold for the branch misprediction counter is 1 in every 100 committed instructions. Figure 4a shows the IPC degradation and the power savings obtained with the dynamic model as compared to the base processor model. Overall, we observe a power saving of 15% and a performance degradation of only 2%. In certain cases, as in gap, employing two clusters provides a slight performance improvement (less than 2%) that can be attributed to a reduction in inter-cluster communication and contention cycles. Most programs (17 of the 23 benchmarks) encountered fewer than five configuration transitions over the course of the execution of the program. The other programs contain many different program phases during their execution as a result of which the cache miss rates and the branch misprediction rates vary significantly. For example, art incurs as many as 91 phase transitions over the execution of the program. These frequent transistions cause the ED^2 to be slightly sub-optimal, but the overall ED^2 is much lower than that for the base case for most of these programs. As pointed out in Section 3.2, applu is one of the programs that is not handled by our heuristic and it yields an ED^2 value that is higher than that of the base case. Overall, the ED^2 values achieved by the dynamic mechanism are very similar to that of the optimal configurations shown for each benchmark in Figure 3.



a. IPC loss and power savings for dynamic model compared to baseline model with 4 HPCs.



b. IPC, Power and ED^2 for different configurations, each normalized with respect to the baseline processor model with 4 HPCs.

Figure 4. Performance and power characteristics.

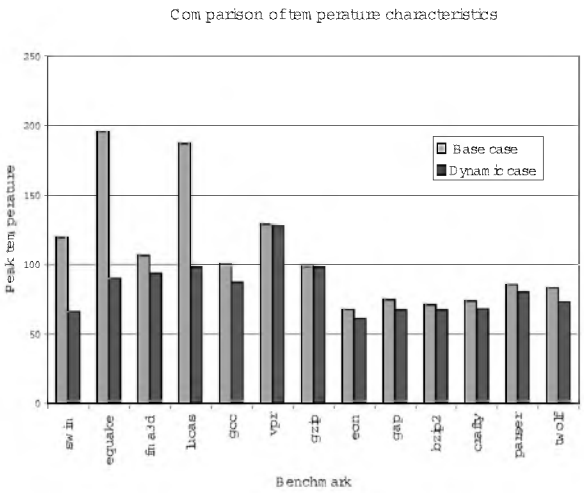


Figure 5. Peak operating temperatures reached by the base processor model and the dynamically adaptive processor model

Figure 4b summarizes performance, power, and ED^2 for different configurations, averaged across the benchmark set (averages are taken such that every program executes for an equal number of cycles [27]) and normalized with respect to the base processor model with four fast clusters. None of the configurations exceed the performance of the base processor. Similarly, all of the studied configurations consume much lower power than the base case. The model with four slow clusters (*4s*) consumes the least power but also incurs a performance penalty of nearly 40%. All static configurations have a higher ED^2 than the base case. The bars for the ideal processor model show results if the ED^2-optimal configuration for each benchmark is picked by an oracle from a large set of configurations. The dynamic mechanism's behavior is very close to this ideal scenario (ED^2 improvement of 11%, compared to 12% for the ideal case).

4.3 Temperature Analysis

The dynamic mechanism selects a configuration that is likely to optimize ED^2. Even though temperature reduction was not a priority in this mechanism, a configuration optimized for ED^2 can also reduce operating temperature, compared to the base processor. Figure 5 compares the peak temperatures reached by our dynamically adaptive processor model and the base processor model (no thermal emergency threshold was set). The graph leaves out programs that had the same peak temperature in both cases (note that the dynamic mechanism executes nearly half the programs with four HPCs). Overall, across these programs, we observe a 22.4% reduction in peak operating temperature. For benchmarks that employ four LPCs, this reduction was 48%. For bechmarks that employ two HPCs, clusters alternate between executing at peak frequency and being shut off. By allowing each cluster to periodically cool down, we observed an average reduction of 7.5% in peak temperature for these programs.

A subset of the benchmarks reach peak temperatures that are high enough to trigger thermal emergencies. The peak temperature corresponds to the temperature of the hottest block on the micro-architectural floorplan. The use of heterogeneous clusters on a thermal emergency allows the program to have tolerable performance penalties while lowering the operating temperature. For example, the program vortex reaches a peak temperature of 93 C without any thermal management in both the base model and the dynamic processor model (that employs four HPCs for minimum ED^2). While a baseline model would scale the frequency of the entire processor, heterogeneity allows us to operate with three HPCs and alternate the clusters between executing at peak frequency and being switched off. This strategy incurs a performance loss of 5.6%, when compared against the baseline processor model employing no thermal management. When the base case employs thermal management by scaling the frequency of all four clusters, we observe a per-

Benchmark	Configuration with best performance for power $\leq$ 80 W	Benchmark	Configuration with best performance for power $\leq$ 80 W
ammp	*3s1f*	applu	*2c*
art	*2f2s*	bzip2	*3s1f*
crafty	*3s1f*	eon	*3s1f*
equake	*2f2s*	fma3d	*2f2s*
galgel	*3s1f*	gap	*3s1f*
gcc	*2c*	gzip	*2c*
lucas	*2f2s*	mcf	*2f2s*
mesa	*3s1f*	mgrid	*2f2s*
parser	*3s1f*	swim	*2f2s*
twolf	*2c*	vortex	*3s1f*
vpr	*2c*	wupwise	*3s1f*
apsi	*2c*		

Table 3. Best static configurations for a fixed power budget.

formance degradation of 8.5% (with respect to the base case with no thermal management). On an average, for benchmarks that reach thermal emergencies, our proposed processor model performs 7.5% better than the base processor model, when both employ thermal emergency management strategies.

4.4 Fixed Power Budget

If a processor is to operate within a fixed power budget, we can scale its voltage and frequency until the power constraint is met. Voltage scaling has its limitations in future technologies as overheads increase and acceptable voltage margins shrink. If we restrict ourselves to frequency scaling or turning off individual clusters, we observe that heterogeneity often yields the optimal performance for a given power budget. Table 3 shows the static configuration that delivers the best performance for a fixed power budget of 80W. Similar results are observed for different power budgets. Clearly, a single static configuration is not sufficient to deliver the best performance for all benchmarks. Compared to a configuration that employs frequency scaling for all four clusters (*4s*), the dynamic algorithm described in Section 3.4 provides 23% improved performance.

4.5 Sensitivity Analysis

All through this study, we have assumed that the LPCs execute at one-fourth the frequency of the HPCs. Figure 6 tracks the IPC, power, and ED^2 characteristics for processor models with varying LPC frequencies, all normalized with respect to the base case. In all configurations, a dynamic processor configuration provides ED^2 improvements, with the highest improvement seen when the frequency of the LPC is one-fourth the HPC frequency. As we increase the frequency of the LPCs, we obtain performance improvements and corresponding increases in power dissipation. By tuning the frequency of the LPCs, we can arrive at different points in the power-performance curve, while still achieving ED^2 benefits. We also found that the use of voltage scaling further increases our overall ED^2 improvements, but it has not been employed for any of the presented

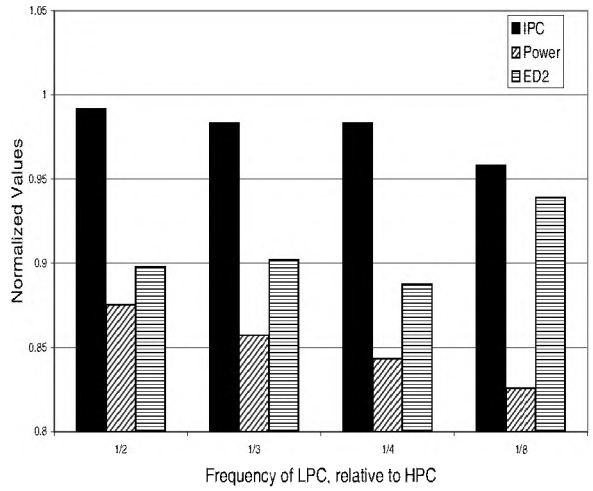


Figure 6. Normalized IPC, Power, and ED^2 for the dynamic mechanism for different LPC frequencies

results because of the associated non-trivial overheads.

In Figure 4a, we had seen that the proposed dynamic algorithm incurs a maximum performance loss of around 8% for eon and 7% for applu. In some cases, designers may want to impose the constraint that ED^2 be improved while limiting performance loss. Tuning the LPC frequency (above) is one strategy to limit performance loss. We can also tune the algorithm to limit the maximum performance loss to a certain value by choosing conservative configurations at the cost of relatively higher power dissipation. For example, for benchmarks with high branch misprediction rates, we can choose to employ three HPCs instead of two HPCs. Likewise, for benchmarks that have a high cache miss frequency, we can employ LPCs that have frequencies that are one-half the HPC frequency. With these new policies, the maximum performance loss for any program was 2.5% (applu). The overall performance loss was less than 1%, but the ED^2 saving was only 7%. The above analysis demonstrates the robustness of the dynamic mechanism in adapting itself to varying performance and power require-

ments.

We also examined the sensitivity of our results as a function of interval length (for making configuration decisions). If frequency change overheads were ignored, we found that ED^2 improvements were approximately constant even if the interval size was reduced to 10K instructions. For an interval size of 1K instructions, the cache miss and branch misprediction measurements are noisy, leading to frequent selections of sub-optimal configurations. While a smaller interval length may allow us to capture the behavior of smaller phases, we found this benefit to be minimal. A large interval length helps amortize the overheads of changing the frequency of a cluster or turning a cluster off.

5 Related Work

Commercial manufacturers have developed processors capable of voltage and frequency scaling [17, 21]. Intel's XScale [17] processor employs this dynamic scaling technique in applications with real-time constraints, thereby improving battery life. Frequency and voltage scaling has also been employed within specific domains of a processor, such as in the MCD processor proposed by Semeraro *et al.* [40]. Interval-based strategies for global frequency scaling for real-time and multimedia applications have been proposed by Pering *et al.* [36]. Childers *et al.* [16] propose trading IPC for clock frequency. In their study, high IPC programs run at low frequencies and low IPC programs run at high frequencies.

The primary goal of our study is efficient utilization of transistor budgets, where ED^2 is the metric for efficiency. The temperature side-effects of our proposal are similar in spirit to the "cluster hopping" proposal of Chaparro *et al.* [14], where parts of the back end are shut down to reduce peak temperatures. That study does not explore the effect of operating clusters at lower frequencies. Chen *et al.* [15] propose a clustered architecture that combines a high-ILP low-frequency core with a low-ILP high-frequency core. The authors demonstrate that such an approach can better match the ILP needs of programs and improve performance. Our hypothesis is that identical cores combined with frequency scaling have lower design complexity, provide significant ED^2 improvements, and reduce operating temperatures. Morad *et al.* [32] employ an asymmetric chip multiprocessor in which different threads of a program are assigned to cores of varying complexity. Kumar *et al.* [30] design a chip multi-processor with heterogeneous cores to execute multi-programmed workloads. Programs are assigned to cores of varying complexity based on their ILP requirements. Ghiasi *et al.* [24] implement a scheduler that examines the execution characteristics of a program and assigns it to a processor that matches its needs. The above studies demonstrate that heterogeneity has favorable energy and energy-delay properties when executing independent programs on multi-processor systems.

6 Conclusions

Future billion transistor architectures are capable of high performance, but suffer from extremely high power densities. This paper advocates power aware resource scaling in an effort to achieve high performance while reducing power and operating temperature. The paper makes the following key contributions:

- A heterogeneous architecture that leverages the modularity of a partitioned architecture and preserves the low design complexity of a partitioned architecture by employing frequency scaling to provide heterogeneity
- A dynamic adaptation policy that exploits information on cache miss rates and branch prediction accuracy to predict an ED^2-optimal configuration
- A mechanism that switches the frequency of a cluster at periodic intervals to lower operating temperatures and a thermal emergency handling mechanism that imposes minimal performance penalties

Based on our evaluations, we draw the following major quantitative conclusions:

- Heterogeneity, combined with our dynamic adaptation policy, reduces ED^2 by 11% (power saving of 15% and performance penalty of 2%).
- Benchmark programs that employ four LPCs show an average 48% reduction in operating temperature. The same number for programs that employ two HPCs is 7.5%.
- By scaling individual clusters on a thermal emergency, our dynamic algorithm performs 7.5% better than the base case that scales the operating frequency of all clusters on a thermal emergency.

The proposed innovations introduce very little design complexity. Our results demonstrate that heterogeneity has the potential to improve single-thread performance in a power-efficient manner. While this paper focuses on single-thread behavior on a clustered architecture, the general ideas can also apply to multi-threaded workloads on chip multi-processors (CMPs). Multiple threads of a single application that execute on different cores of a CMP are analogous to dependence chains executing on clusters. For future work, we will explore the potential of intelligent thread assignment to cores within an asymmetric CMP. We will also examine other techniques to introduce heterogeneity within clusters or cores, such as the use of simple in-order pipelines.

References

[1] A. Abdollahi, F. Fallah, and M. Pedram. An Effective Power Mode Transition Technique in MTCMOS Circuits. In *Proceedings of DAC-42*, 2005.

[2] A. Aggarwal and M. Franklin. An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors. In *Proceedings of ISPASS*, 2001.

[3] S. I. Association. International Technology Roadmap for Semiconductors 2001. http://public.itrs.net/Files/2001ITRS/Home.htm.

[4] R. I. Bahar and S. Manne. Power and Energy Reduction Via Pipeline Balancing. In *Proceedings of ISCA-28*, pages 218–229, July 2001.

[5] R. Balasubramonian. Cluster Prefetch: Tolerating On-Chip Wire Delays in Clustered Microarchitectures. In *Proceedings of ICS-18*, June 2004.

[6] R. Balasubramonian, S. Dwarkadas, and D. Albonesi. Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors. In *Proceedings of ISCA-30*, pages 275–286, June 2003.

[7] A. Baniasadi and A. Moshovos. Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors. In *Proceedings of MICRO-33*, pages 337–347, December 2000.

[8] R. Barua, W. Lee, S. Amarasinghe, and A. Agarwal. Maps: A Compiler-Managed Memory System for Raw Machines. In *Proceedings of ISCA-26*, May 1999.

[9] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, November/December 2000.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of ISCA-27*, pages 83–94, June 2000.

[11] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[12] R. Canal, J. M. Parcerisa, and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. In *Proceedings of HPCA-6*, pages 132–142, January 2000.

[13] R. Canal, J. M. Parcerisa, and A. Gonzalez. Dynamic Code Partitioning for Clustered Architectures. *International Journal of Parallel Programming*, 29(1):59–79, 2001.

[14] P. Chaparro, J. Gonzalez, and A. Gonzalez. Thermal-effective Clustered Micro-architectures. In *Proceedings of the 1st Workshop on Temperature Aware Computer Systems, held in conjunction with ISCA-31*, June 2004.

[15] L. Chen, D. Albonesi, and S. Drophso. Dynamically Matching ILP Characteristics Via a Heterogeneous Clustered Microarchitecture. In *IBM Watson Conference on the Interaction between Architecture, Circuits and Compilers*, pages 136–143, October 2004.

[16] B. Childers, H. Tang, and R. Melhem. Adapting Processor Supply Voltage to Instruction Level Parallelism. In *Proceedings of the Kool Chips Workshop, in conjunction with MICRO-33*, December 2000.

[17] L. Clark. Circuit Design of XScale Microprocessors. In *Symposium on VLSI Circuits (Short Course on Physical Design for Low-Power and High-Performance Microprocessor Circuits)*, June 2001.

[18] J. Collins and D. Tullsen. Clustered Multithreaded Architectures – Pursuing Both IPC and Cycle Time. In *Proceedings of the 18th IPDPS*, April 2004.

[19] A. Dhodapkar and J. E. Smith. Managing Multi-Configurable Hardware via Dynamic Working Set Analysis. In *Proceedings of ISCA-29*, pages 233–244, May 2002.

[20] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. The Multicluster Architecture: Reducing Cycle Time through Partitioning. In *Proceedings of MICRO-30*, pages 149–159, December 1997.

[21] M. Fleischmann. Longrun Power Management. Technical report, Transmeta Corporation, January 2001.

[22] D. Folegnani and A. Gonzalez. Reducing Power Consumption of the Issue Logic. In *Workshop on Complexity-Effective Design (WCED2000, held in conjunction with ISCA-27)*, June 2000.

[23] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC Variations in Workloads with Externally Specified Rates to Reduce Power Consumption. In *Workshop on Complexity Effective Design (WCED2000, held in conjunction with ISCA-27)*, June 2000.

[24] S. Ghiasi, T. Keller, and F. Dawson. Scheduling for Heterogeneous Processors in Server Systems. In *Proceedings of the Second Conference on Computing Frontiers*, pages 199–210, 2005.

[25] E. Gibert, J. Sanchez, and A. Gonzalez. Flexible Compiler-Managed L0 Buffers for Clustered VLIW Processors. In *Proceedings of MICRO-36*, December 2003.

[26] M. Huang, J. Renau, and J. Torrellas. Positional Adaptation of Processors: Applications to Energy Reduction. In *Proceedings of ISCA-30*, pages 157–168, June 2003.

[27] L. John. More on Finding a Single Number to Indicate Overall Performance of a Benchmark Suite. *ACM Computer Architecture News*, 32(1), March 2004.

[28] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany. The Imagine Stream Processor. In *Proceedings of ICCD*, September 2002.

[29] H.-S. Kim and J. Smith. An Instruction Set and Microarchitecture for Instruction Level Distributed Processing. In *Proceedings of ISCA-29*, May 2002.

[30] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proceedings of the 36th International Symposium on Micro-Architecture*, December 2003.

[31] A. Martin, M. Nystrom, and P. Penzes. ET2: A Metric for Time and Energy Efficiency of Computation. Technical Report CaltechCSTR:2001.007, Caltech Computer Science, 2001.

[32] T. Morad, U. Weiser, A. Kolodny, M. Valero, and E. Ayguade. Performance, Power Efficiency, and Scalability of Asymmetric Cluster Chip Multiprocessors. Technical Report CCIT Technical Report #514, January 2005.

[33] R. Nagarajan, K. Sankaralingam, D. Burger, and S. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *Proceedings of MICRO-34*, pages 40–51, December 2001.

[34] U. of California Berkeley. BSIM3v3.3 Berkeley Spice Device Models. http://www-device.eecs.berkeley.edu/ bsim3/.

[35] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The Case for a Single-Chip Multiprocessor. In *Proceedings of ASPLOS-VII*, October 1996.

[36] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, August 1998.

[37] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources. In *Proceedings of MICRO-34*, pages 90–101, December 2001.

[38] P. Racunas and Y. Patt. Partitioned First-Level Cache Design for Clustered Microarchitectures. In *Proceedings of ICS-17*, June 2003.

[39] J. Sanchez and A. Gonzalez. Modulo Scheduling for a Fully-Distributed Clustered VLIW Architecture. In *Proceedings of MICRO-33*, pages 124–133, December 2000.

[40] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott. Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *Proceedings of HPCA-8*, pages 29–40, February 2002.

[41] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of ASPLOS-X*, October 2002.

[42] A. Sjogren and C. Myers. Interfacing SYnchronous and Asynchronous Modules within a High-Speed Pipeline. In *Proceedings of 17th Conference on Advanced Research in VLSI*, September 1997.

[43] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of ISCA-30*, June 2003.

[44] E. Tune, D. Liang, D. Tullsen, and B. Calder. Dynamic Prediction of Critical Path Instructions. In *Proceedings of HPCA-7*, pages 185–196, January 2001.

[45] S. Yang, M. Powell, B. Falsafi, K. Roy, and T. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep Submicron High-Performance I-Caches. In *Proceedings of HPCA-7*, pages 147–158, January 2001.

[46] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. Technical Report CS-2003-05, Univ. of Virginia, March 2003.