Independent Front-end and Back-end Dynamic Voltage Scaling for a GALS Microarchitecture

Grigorios Magklis, Pedro Chaparro, José González, Antonio González

Intel Barcelona Research Center, Intel Labs – UPC

{grigorios.magklis, pedro.chaparro.monferrer, pepe.gonzalez, antonio.gonzalez}@intel.com

ABSTRACT

In recent years, Globally Asynchronous Locally Synchronous (GALS) designs and dynamic voltage scaling (DVS) have emerged as some of the most popular approaches to address the ever increasing microprocessor energy consumption. In this work, we propose two on-line algorithms for adjusting dynamically, and independently, the voltage and frequency of the front-end and back-end domains of a novel two-domain microprocessor. We evaluate our mechanisms for both internal and external voltage regulators, and we present optimal dynamic voltage scaling results for the proposed microarchitecture. Our schemes achieve average improvement of 12% of the energy-delay^2 metric, when using internal voltage regulators.

Categories and Subject Descriptors

C.1.0 [**Processor Architectures**]: General

General Terms

Algorithms, Design.

Keywords

DVS, GALS, MCD, energy efficiency, microarchitecture

1. INTRODUCTION

Microprocessor power consumption has increased significantly in recent years, so much so that *energy efficiency* (or power efficiency) has become one of the main targets of microprocessor architects. Energy efficiency can be addressed statically by changing the microarchitecture design, or dynamically by employing run-time mechanisms to adapt the hardware to applications. One static technique is employing a Globally Asynchronous Locally Synchronous design [4]. GALS achieves better energy efficiency by reducing the complexity and power dissipation of the clock distribution, which constitutes a large part of the total processor power [10][11]. One of the most successful run-time techniques for improving energy efficiency is dynamic voltage and frequency scaling (DVS for short) [1]. GALS systems have the unique ability to operate each domain at different frequency and voltage, which allows applying DVS independently to different parts of the processor. It has been shown that per-domain adaptation is significantly more energy efficient compared to global adaptation [20].

In this work, we study the effect of fine-grain DVS on a clustered GALS microprocessor. Our microarchitecture utilizes clustering on both the front-end and back-end of the microprocessor, resulting in a very efficient baseline [3]. We propose to further increase the efficiency of the system by separating the core into two clock domains (front-end and back-end) and by allowing independent DVS for each domain. Our work is the first to propose a front-end/back-end split of the microprocessor and to propose DVS control for *all* domains of a GALS microprocessor. Moreover, we perform a more complete study than previous works, by including the energy costs of internal voltage regulators, and leakage energy in our results. Third, we use a significantly different microarchitecture than previous MCD-like studies.

We have evaluated a previous state-of-the-art work based on queue utilization [20] for our microarchitecture and it achieves near zero energy-delay^2 improvement. The reason is the difference in the microarchitectures studied. Queue utilization is not a sufficiently good indicator of performance for our microarchitecture. Firstly, because our pipeline is much wider and has a far greater degree of out-of-order execution compared to previous proposals. Secondly, we have more queues in the microarchitecture than MCD-like proposals (due to synchronization FIFOs and clustering), which requires a significantly more complex mapping of queue utilization to performance.

2. GALS MICROARCHITECTURE

Figure 1 shows our GALS microarchitecture. The processor consists of three clock domains, shaded grey: front-end (FE), back-end (BE) and memory (MEM). We consider the memory domain external to the core of the microprocessor (but still on-chip). The processor follows a clustered design.

The FE contains a branch predictor, a trace cache, and an IA32 decoder, the ROB, the dispatch and the commit logic. The FE is divided into two clusters [3]. Instruction fetch, decode and steering is centralized. After the steering logic decides the destination BE cluster [8], it directs the instruction to the corresponding FE cluster. Each FE cluster has its own rename table and ROB, which are simpler and smaller than in a monolithic design. Renaming and allocation proceeds independently and in parallel. The commit logic must maintain order between the two ROBs, which increases its latency, but it is not on the critical path of the execution [3].

The BE is also divided into two clusters. Each cluster consists of an integer and a floating-point out-of-order execution engines. The load-store queue and the first-level data cache are centralized, and shared among the clusters. Addresses are calculated at the execution clusters and then are passed to the LSQ for resolution and issuing to the cache. Special *copy* instructions explicitly communicate register values between the clusters using point-to-point links [14].

Table 1. Voltage-frequency levels.

Level	mV	GHz	Level	mV	GHz
0	700	3400	7	875	5000
1	725	3700	8	900	5200
2	750	3900	9	925	5400
3	775	4200	10	950	5600
4	800	4400	11	975	5800
5	825	4600	12	1000	6000
6	850	4800			

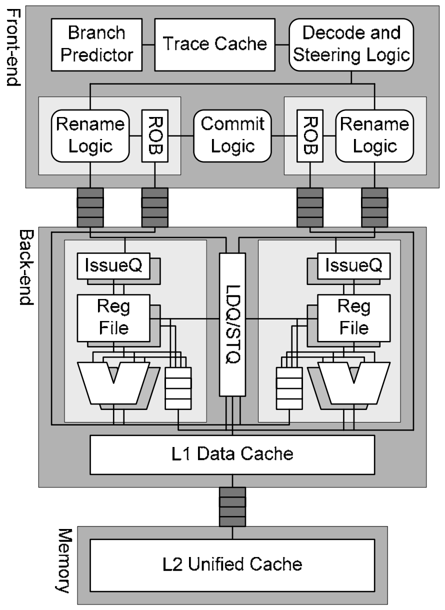

Figure 1. Clustered GALS microarchitecture.

Each domain has its own local clock network that distributes a reference clock signal to the domain. We assume that the skew between the domain reference clocks can be arbitrary. This allows to run each domain at a different frequency, and to apply DVS to each domain independently of the others but inter-domain communication must be synchronized correctly to avoid meta-stability We assume synchronizing FIFOs for inter-domain communication, similar to previous studies [5][16][17]. We chose this domain separation so as to minimize the performance loss due to synchronization delays. We keep the schedule–execute–bypass loop inside a domain so we can perform back to back scheduling of dependent instructions. Moreover, we do not put each cluster in a different domain because that would not allow back-to-back execution of copy- and load-dependent instructions.

3. DYNAMIC VOLTAGE SCALING

Domains can execute through voltage changes, similar to previous studies [11][12][16][20]. Each domain includes an on-chip digital clock multiplier connected to a single, shared, external PLL [6][13]. This limits our DVS choices (Table 1), but allows frequency changes without stopping the domain. Each domain includes an internal voltage regulator [9]. This allows for extremely fast voltage switching (in just a few nsec). Hazucha *et al.* report 94% efficiency (output power divided by input power) for their design. We are more conservative and assume only 90% efficiency. Our power model accounts for power losses due to the regulator inefficiency (independently for each regulator).

The goal of our proposed mechanisms is to improve energy efficiency of the baseline microarchitecture by adapting the voltage and frequency of the front-end and the back-end domains of the processor. There are several metrics for energy efficiency. In this study, we adopt the *energy-delay2* (ED^2) metric proposed by Brooks *et al.* [1]. Thus, we need a mechanism—at the microarchitecture level—to estimate ED^2. The ED^2 metric, for an interval, is calculated as $(E_{dynamic}+E_{static})\cdot N^2$, where N is the value of the cycle counter for the interval and $E_{dynamic}$ and E_{static} are the dynamic and static energy consumption for this interval.

We propose to use the mechanism devised in [7] to measure dynamic energy. The microprocessor uses performance counters to measure the activity on the various units. The total activity, during a time interval, is multiplied by the energy-per-access register associated with the unit (fixed at design time), to calculate the total energy of the unit. In this work, we estimate the static energy of a circuit, using the formula by Zhang *et al.* [21]; we assume that there exists a mechanism to estimate static energy consumption at run-time.

For a specific execution interval, if we plot the ED^2 metric over all voltage and frequency (V-F) settings, we will observe a roughly U-shaped graph. The lowest point in the graph is the optimal ED^2 for this execution interval. All points to the left of the optimal (*i.e.*, those with lower V-F) have higher ED^2 because the application loses too much performance compared to the energy reduction achieved. Correspondingly, to the right of the optimal, ED^2 is higher because the application consumes too much energy for the performance benefit realized.

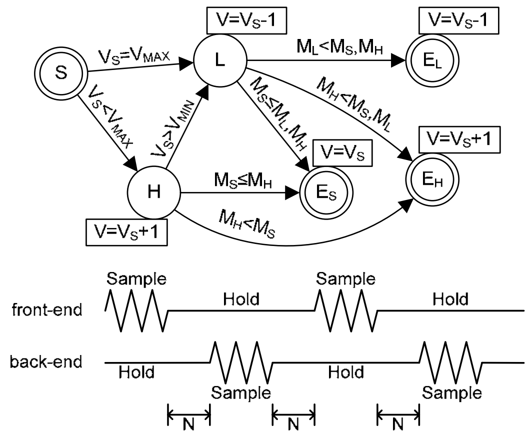

Figure 2. *Best-neighbor* state machine and meta-control.

3.1. Best-Neighbor Adaptation

For *best-neighbor* adaptation, the controller operates in two distinct phases: *sample* and *hold*. In the *sample* phase, the system calculates the ED^2 for the different V-F settings in the neighborhood of the current V-F level by trying each configuration for an interval. At the end of the *sample* phase, the controller decides the V-F setting with the minimum ED^2. This setting is then applied and the controller moves to the *hold* phase. In this phase, the controller maintains the chosen V-F setting for several intervals.

Figure 2 shows the state machine of the domain controller, for the *sample* phase. The controller starts at state S, with voltage V_S. After one interval, it moves to state H, to sample the ED^2 at a higher V-F setting (if not already running at maximum V-F). Next interval, it moves to state L, to sample ED^2 at a lower V-F setting. In the figure, M_S, M_L, and M_H denote the ED^2 of the corresponding states. Then, the controller chooses the best V-F setting of the

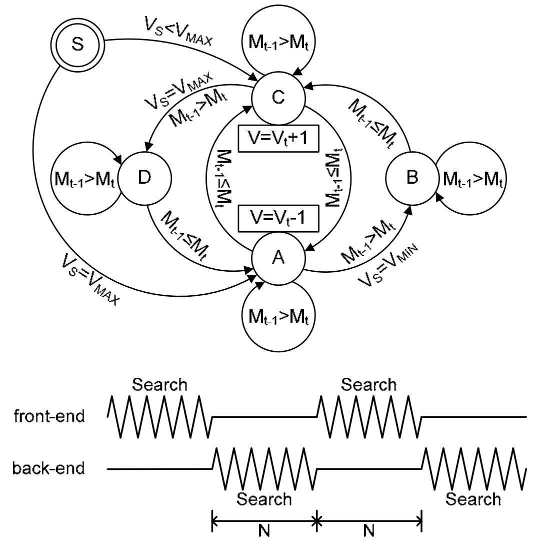

Figure 3. *Greedy-search* state machine and meta-control.

three for the *hold* phase. States E_S, E_L and E_H correspond to the terminal states of the controller. Upon entering one of the *E* states, the controller sets the V-F to the appropriate values, resets to the *S* state, and waits until the *hold* phase ends.

Imagine the situation where we are trying to make a decision on the front-end V-F setting. If during the *sample* phase, we do not hold the back-end V-F constant then we do not know if the benefit in ED^2 that we see during an interval is because of the front-end or the back-end V-F changes. To avoid this situation there is a meta-controller to force the domain controllers to operate in an interleaved fashion. The meta-controller ensures that when one of the domains is in the *sample* phase the other one will be in the *hold* phase and vice versa. Figure 2 shows the meta-controller operation graphically. The parameter *N* defines how many intervals should pass between the *sample* phases of the two domains.

3.2. Greedy-Search Adaptation

The goal of *greedy-search* is to approximate the optimal ED^2 by following an imaginary U-shaped curve. Figure 3 shows the state machine of the domain controller. The controller starts at state *S* and after one interval it moves to either state *A* or *C*, depending on the starting V-F (the V-F changes accordingly). After this point, the controller uses the history of the last two intervals *t* and *t*-1 to make decisions for the upcoming interval *t*+1 (M_t and M_{t-1} denote the ED^2 of the corresponding intervals). If the ED^2 was reduced, then the last V-F change was correct and we continue changing in the same direction. Otherwise, if the ED^2 increased, then we reverse direction.

The assumption is that if the last V-F change was beneficial then we are in the right direction to reach the valley of the ED^2 U-curve. If the last change was not beneficial then we are going in the wrong direction (either we started wrong or we reached the bottom of the U-curve and started climbing the other way) and we should change. Ideally, this mechanism will force the system to stay close to the lowest point of the U-curve.

There is also a meta-controller to interleave the two domain controllers. The meta-controller periodically (every *N* intervals) disables the currently active controller and enables the other one (Figure 3). When a controller is de-activated, the voltage and frequency for the corresponding domain remain at the last setting chosen until the controller is re-activated. Every time a controller is re-activated, it starts from the *S* state.

Table 2. Microarchitectural parameters.

Front-end	
Fetch	24K inst. trace cache, 6 inst./cycle, 5 cycle fetch-to-dispatch
Decode, rename and steer	3+3 inst./cycle, 1 cycle latency, plus 1 cycle wire delay to synch, FIFOs
ROB	256+256 entries, commit 3+3 inst./cycle
Back-end (configuration shown per cluster)	
Synch. FIFOs	1 FIFO per issue queue, 24 entries each
Issue queues	48-entry INT, 2 inst./cycle and 48-entry FP, 2 inst./cycle and 24-entry COPY, 1 inst./cycle
Register file	256-entry INT, 256-entry FP
Inter-cluster	1 cycle latency, 1 copy/cycle
L1 data	32KB, 4-way, 3 cycle hit, 2 read ports, 1 write port, 256-entry LSQ
Memory	
L2 unified	2MB, 16-way, 13 cycle hit, ≥ 500 cycle miss, 1 read port, 1 write port

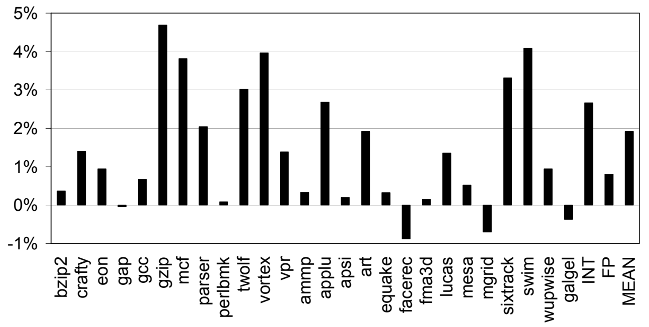

Figure 4. GALS performance loss due to inter-domain synchronization.

4. PERFORMANCE RESULTS

We use a cycle accurate simulator that executes traces of IA32 binaries, including OS code. The simulator also includes a power estimation module, based on an enhanced version of CACTI [18], utilizing activity counters (similar to Wattch [2]) and a leakage module utilizing the formula by Zhang *et al*. [21]. The assumed technology is 45nm. We initialize domain clocks at random values at the beginning of simulation time to account for clock skew. Table 2 shows the main microarchitectural parameters of the processor. For our experiments, we use the twenty-six applications of the SPEC CPU2000 benchmark suite, compiled with the latest Intel® compiler with full optimizations, and run with the reference input set. We simulate 200M IA32 instructions from the middle of the application.

We also employ off-line simulation to calculate the optimal V-F scheduling for a given execution. For the off-line analysis and all simulations that compare with the off-line, we simulate only 20M instructions, due to simulation time constraints. First, we simulate an application for all combinations of FE and BE V-F settings, and we collect statistics at fixed instruction intervals. Then we construct a timeline, *i.e.*, a sequence of intervals, for each application. To emulate a control mechanism *C*, we choose a V-F combination *A*, according to the control policy of *C*, for an interval of execution. We then add the statistics of *A* for this interval to the

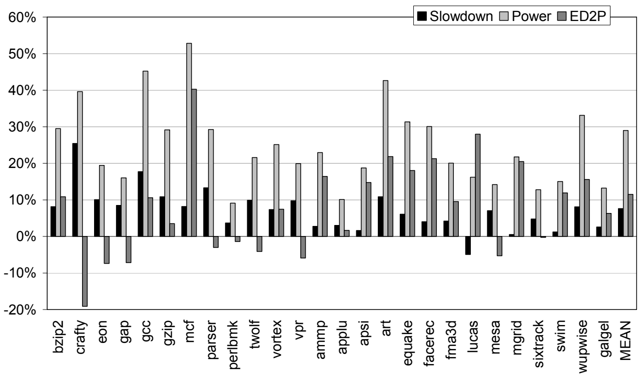

Figure 5. *Best-neighbor* slowdown, power reduction, and ED^2 improvement.

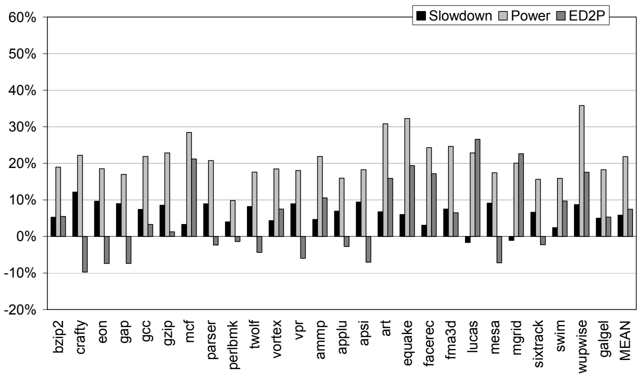

Figure 6. *Greedy-search* slowdown, power reduction, and ED^2 improvement.

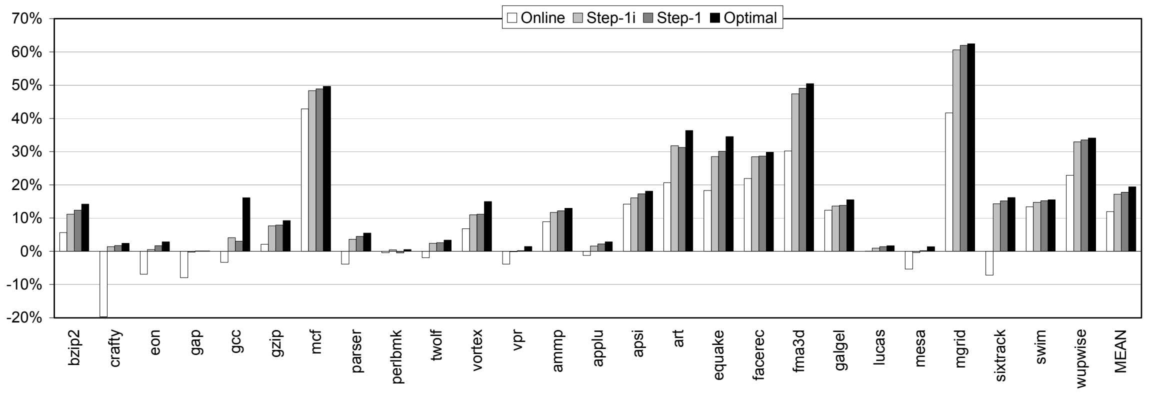

Figure 7. ED^2 improvement of *best-neighbor* (*Online*) compared to various off-line mechanisms.

off-line total statistics, and repeat this process for the next interval. We stop when we reach the end of the timeline. The average error of an off-line vs. a dynamic run is about 0.45%.

4.1. GALS Performance

In this section, we evaluate the performance loss due to inter-domain synchronization whereas in following sections we evaluate the proposed DVS schemes. Figure 4 shows the slowdown of our GALS microprocessor running at maximum frequency, compared to an identical, fully synchronous one. All of the performance loss shown in this figure is due to inter-domain synchronization penalties. The worst-case slowdown is 4.7% for *gzip* with an average of less than 2% over all benchmarks.

There are two sources of performance loss in the microprocessor: the branch misprediction loop and the cache miss loop. Branch misprediction involves two synchronizations: (a) from the FE to the BE, when we decode a branch and before the branch is resolved, and (b) from the BE to the FE, to communicate the misprediction after the branch is resolved. L1 misses also involve two synchronizations: (a) from the BE to the L2, to notify for the miss and request the data, and (b) from the L2 to the BE, when the request is serviced. Integer benchmarks exhibit higher slowdown on average compared to floating-point ones (2.6% vs. 0.8%), mainly due to higher branch misprediction rate.

4.2. Fine-grain DVS

Figure 5 shows the performance loss, power reduction and ED^2 improvement of *best-neighbor*. The comparison is performed against the clustered GALS processor running all domains at the highest V-F. The interval is 10K instructions and *N*=1. This translates to 50K instructions for the *hold* phase on average. Figure 6 shows the same results for *greedy-search*. The interval length is similarly set, and the meta-controller interleaves the domains every 90K instructions (*N*=9). For both mechanisms, *N* was chosen after an extensive search process, utilizing the off-line simulator. For many values of *N* close to the ones chosen, the results do not vary significantly.

Summarizing the two figures, the average ED^2 improvement of *best-neighbor* is about 11.5% while *greedy-search* achieves roughly 7.5%. The average slowdown of *greedy-search* is lower than *best-neighbor*, mainly due to its ability to perform large frequency changes faster: *best-neighbor* performs only a single-step change per turn, while *greedy-search* is allowed up to nine changes (*N*=9). This is beneficial when the application suffers abrupt changes in demand. *Greedy-search* reacts faster and reaches the new stable state earlier. *Best-neighbor* has the benefit of being more stable. If the domain is already running at the best V-F then the *sample* phase will choose not to change; *greedy-search* will constantly move around the best setting.

4.3. Optimal Fine-grain DVS

In order to evaluate the effectiveness of our schemes we calculate the best ED^2 for each application, utilizing off-line simulation. Due to space restrictions, we show results only for *best-neighbor*. Since the *hold* phase is roughly 50K instructions, we use this number as the interval length for all off-line mechanisms. We compare our schemes against three different upper bounds.

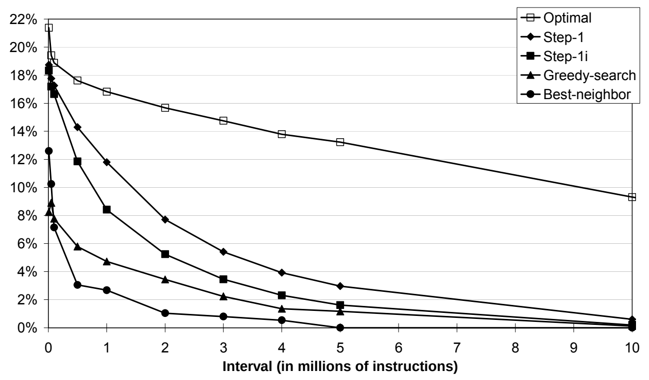

Figure 8. ED^2 improvement for various interval sizes.

The first scheme is called *Optimal*. At each interval, it searches all combinations of V-F for the two domains and chooses the best one. This is the best we can do with any kind of dynamic mechanism. Our DVS mechanisms are restricted to single-step V-F changes. The second upper bound scheme, called *Step-1*, is used to discover if this restriction has an inherent inefficiency. At each interval, the V-F of each domain is changed by one level, towards the direction of the optimal V-F. Our third off-line mechanism, called *Step-1i*, accounts for the meta-controller by interleaving the two domains. This mechanism also accounts for single-step V-F changes, similarly to *Step-1*. This represents a tighter upper bound for our DVS mechanisms.

Figure 7 shows the ED^2 achieved with the four schemes. The results for *best-neighbor*, labeled *Online*, are not exactly the same as those in Figure 5 but very similar, because now we simulate only 20M instructions, while in Section 4.2 we simulated 200M instructions. *Online* achieves an average ED^2 improvement of 12% over all benchmarks, while *Step-1i*, *Step-1* and *Optimal* achieve 17.2%, 17.8% and 19.4% respectively. This means that we achieve about 70% of the efficiency of *Step-1i*, which is the best we could do given the two restrictions of our mechanisms: single-step V-F changes and domain interleaving.

Comparing *Step-1i* and *Step-1*, we can see that interleaving has minimal impact on the efficiency of the mechanisms (*Step-1i* achieves 97% of *Step-1*). This means that we can develop simpler control systems without loosing efficiency, which is exactly what we propose: simple per-domain control that ignores interactions and side effects from the rest of the domains, and simple meta-control to co-ordinate the domain controllers. Comparing *Step-1* and *Optimal*, we can see that single-step V-F changes have bigger impact than interleaving (*Step-1* achieves 92% of *Optimal*) due to the slow change rate in the face of sharp changes in program behavior. When the control interval is short, as in this case, the time it takes to stabilize to the new V-F is short. When the interval is long, the stabilization delay may incur significant inefficiency.

Figure 8 shows the sensitivity to the interval size. As expected, power efficiency gains are smaller when moving to longer intervals. This is because the mechanisms—including *Optimal*—cannot take advantage of short application phases; they must adjust V-F to account for the average behavior over long periods. The efficiency loss due to single-step V-F changes becomes more pronounced as the interval length increases. Interleaving the domains also affects the efficiency of the control, as can be seen by comparing *Step-1i* to *Step-1*, but not so much as single-level V-F changes. We can also see that for intervals of 500K instructions and above, *greedy-search* outperforms *best-neighbor*. When the interval length becomes significant compared to the application phase length, the faster V-F changes of *greedy-search* pay off. For short intervals, the stability of *best-neighbor* wins.

The determining factor of the interval length is the speed of the voltage regulator (VR). The left side of Figure 8 corresponds to fast internal VRs (voltage changes in nsec), while the right side corresponds to slow external VRs (voltage changes in msec). The middle could be either internal or external VRs (voltage changes in μsec). We conclude that *best-neighbor* is best for fast VRs, where *greedy-search* is best otherwise.

We have also evaluated the best known previous work, based on queue utilization [20]. In that work, the utilization of the queues that connect the fetch and the execution domains is tried to be maintained inside some pre-defined bounds.. The ED^2 improvement for this mechanism was close to zero, for all benchmarks (we did not put it in our figures, because it does not show).

This is because of the great differences in the MCD-like microarchitecture of [20] and our proposal. In [20] the back-end is split into domains according to operation type, and most importantly, the instruction queues are the synchronizers. We have special synchronization FIFOs before the instruction queues, and we follow a clustered design. A simple queue model (for mixed-clock FIFO, instruction queue or combination of both), like the one assumed in [20] does not work for our microarchitecture.

Moreover, in our simulations we include the energy consumption of the internal VRs, and leakage energy. Both of these factors influence the improvement we see with DVS, and have been ignored from all previous studies (including [20]), as far as we know. Finally, we claim that our baseline microarchitecture is highly energy efficient, much more than previous proposals. This is due to extensive use of clustering both at the front-end and at the back-end. MCD-like microprocessors may be energy efficient, but we think are very hard to design, due to the non-deterministic latencies of domain crossings.

5. RELATED WORK

Chapiro [4] was the first to introduce the idea of GALS systems. Since then there have been several published works on GALS and DVS. Iyer and Marculescu [11] propose a microprocessor with five domains: fetch, decode and rename, integer pipeline, floating-point pipeline, and memory pipeline (includes first level cache). They conclude that the performance loss due to domain synchronization was significant but that GALS could be more efficient than fully synchronous designs using fine-grain adaptation mechanisms.

Semeraro *et al.* [16][17] propose a Multiple Clock Domain (MCD) processor, with four domains: front-end (fetch and dispatch), integer, floating-point, and memory (with first and second level cache). This separation results in minimal slowdown compared to a globally synchronous design. Moreover, they describe an off-line mechanism to obtain almost optimal DVS. Semeraro *et al.* [15] propose an interval-based hardware control mechanism for the domains of the MCD (all but the front-end), called the *Attack/Decay* [17]. The *Attack/Decay* uses the rate of change of the occupancy of the issue queues of the MCD to decide if the frequency should increase or decrease for the next interval. They report energy-delay product improvement of about 85% of what was achieved with their off-line mechanism.

The authors in [12] combine clustering with MCD into a Clustered Multiple Clock Domain (CMCD) design. The CMCD consists of four back-end clusters (each with a local first level cache), a shared front-end, and a shared second level cache each in a separate domain. They also propose a mathematical model that relates the fetch queue utilization, the branch prediction accuracy, the front-end frequency and the application performance. They use the model in a control mechanism to adapt the voltage and frequency of only the front-end domain, achieving close to optimal results.

Wu *et al.* [19] model the MCD domains as queue systems and propose a feedback control DVS system based on a Proportional-Integral (PI) controller. The controller uses the occupancy of the domain input queue over some interval of time and responds with a frequency for the upcoming interval. The goal is to maintain occupancy close to a pre-defined nominal value. Wu *et al.* [20] propose a DVS mechanism for the MCD that reacts to workload changes instead of making decisions at fixed time intervals. The controller utilizes both the queue occupancy and the rate of change of the occupancy. When a metric consistently exceeds a predefined threshold for a consecutive number of cycles an action is taken.

6. CONCLUSIONS

We have presented and evaluated a novel GALS microarchitecture with minimal IPC degradation (less than 2%). We have also described how to enable independent voltage and frequency control for the domains of the microprocessor, and how to measure energy consumption, including leakage, at run-time. Finally, we have proposed and evaluated two mechanisms for dynamically adapting the frequency and voltage of the domains.

To our knowledge, our proposed dynamic adaptation mechanisms are the first ones to control all domains of a GALS microprocessor. We perform an extensive evaluation of our mechanisms, comparing with optimal control and under different voltage regulator speeds. We conclude that our *best-neighbor* mechanism performs best for internal (fast) voltage regulators while our *greedy-search* mechanism performs best in all other cases. When using internal voltage regulators *best-neighbor* achieves about 12% ED^2 improvement, or 70% of an optimal algorithm with similar characteristics.

7. REFERENCES

[1] D. M. Brooks *et al.* Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6), Nov./Dec. 2000.

[2] D. Brooks, V. Tiwari and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimization. In *International Symposium on Computer Architecture*, June 2000.

[3] P. Chaparro, G. Magklis, J. González and A. González. Distributing the Frontend for Temperature Reduction. In *International Symposium on High-Performance Computer Architecture*, Feb. 2005.

[4] D. M. Chapiro. *Globally Asynchronous Locally Synchronous Systems*. PhD thesis, Stanford University, 1984.

[5] T. Chelcea and S. M. Nowick. Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols. In *Design Automation Conference*, June 2001.

[6] T. Fischer *et al.* A 90-nm Variable Frequency Clock System for a Power-Managed Itanium Architecture Processor. *IEEE Journal of Solid State Circuits*, 41(1), Jan. 2006.

[7] J. González and A. González. Dynamic Cluster Resizing. In *International Conference on Computer Design*, Oct. 2003.

[8] J. González, F. Latorre and A. González. Cache Organizations for Clustered Microarchitectures. In *Workshop on Memory Performance Issues*, June 2004.

[9] P. Hazucha *et al.* Area-Efficient Linear Regulator with Ultra-Fast Load Regulation. *IEEE Journal of Solid-State Circuits*, 40(4), April 2005.

[10] A. Hemani *et al*. Lowering Power Consumption in Clock by Using Globally Synchronous Locally Synchronous Design Style. In *Conference on Design Automation*, June 1999.

[11] A. Iyer and D. Marculescu. Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors. In *International Symposium on Computer Architecture*, May 2002.

[12] G. Magklis, J. González and A. González. Frontend Frequency-Voltage Adaptation for Optimal Energy-Delay^2. In *International Conference on Computer Design*, Oct. 2004.

[13] T. Olsson *et al.* A Digitally Controlled Low-Power Clock Multiplier for Globally Asynchronous Locally Synchronous Designs. In *International Symposium on Circuits and Systems*, May 2000.

[14] J. M. Parcerisa, J. Sahuquillo, A. González and J. Duato. Efficient Interconnects for Clustered Microarchitectures. In *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2002.

[15] G. Semeraro *et al*. Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture. In *International Symposium on Microarchitecture*, Nov. 2002.

[16] G. Semeraro *et al*. Hiding Synchronization Delays in a GALS Processor Microarchitecture. In *International Symposium on Asynchronous Circuits and Systems*, April 2004.

[17] G. Semeraro *et al*. Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *International Symposium on High-Performance Computer Architecture*, Feb. 2002.

[18] P. Shivakumar and N. P. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. WRL Research Report 2001/2, Aug. 2001.

[19] Q. Wu, P. Juang, M. Martonosi and D. W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[20] Q. Wu, P. Juang, M. Martonosi and D. W. Clark. Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors. In *International Symposium on High-Performance Computer Architecture*, Feb. 2005.

[21] Y. Zhang *et al.* HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. Technical Report CS-2003-05, Dept. of Computer Science, University of Virginia, Mar. 2003.