Open access • Journal Article • DOI:10.1109/TVLSI.2005.844305

Toward a multiple clock/voltage island design style for power-aware processors

Emil Talpes, Diana Marculescu

Institutions: Carnegie Mellon University

Published on: 01 May 2005 - IEEE Transactions on Very Large Scale Integration Systems (IEEE)

Topics: Dynamic voltage scaling, Frequency scaling, Low-power electronics, Clock signal and Design methods

Toward a Multiple Clock/Voltage Island Design Style for Power-Aware Processors

Emil Talpes and Diana Marculescu, *Member, IEEE*

***Abstract*—Enabled by the continuous advancement in fabrication technology, present-day synchronous microprocessors include more than 100 million transistors and have clock speeds well in excess of the 1-GHz mark. Distributing a low-skew clock signal in this frequency range to all areas of a large chip is a task of growing complexity. As a solution to this problem, designers have recently suggested the use of frequency islands that are locally clocked and externally communicate with each other using mixed clock communication schemes. Such a design style fits nicely with the recently proposed concept of voltage islands that, in addition, can potentially enable fine-grain dynamic power management by simultaneous voltage and frequency scaling. This paper proposes a design exploration framework for *application-adaptive multiple-clock processors* which provides the means for analyzing and identifying the right interdomain communication scheme and the proper granularity for the choice of voltage/frequency islands in case of superscalar, out-of-order processors. In addition, the presented design exploration framework allows for comparative analysis of newly proposed or already published application-driven dynamic power management strategies. Such a design exploration framework and accompanying results can help designers and computer architects in choosing the right design strategy for achieving better power–performance tradeoffs in multiple-clock high-end processors.**

***Index Terms*—Clocking strategies, low-power design, superscalar processor designs, simulation.**

I. Introduction

THE LAST few decades have been dominated by Moore's Law, with performance being the primary driving force in processor design. This trend has lead to a vast increase in the number of transistors used in modern microprocessors, while pushing the clock frequencies to higher values. Unfortunately, such a trend has also resulted in a huge increase in power dissipation as well, with current processors already dissipating more than 100 W. The increase in power dissipation directly impacts the CPU lifetime, the system power delivery costs and the system thermal design needed for keeping the operating temperature under specified limits.

In addition, one of the major design bottlenecks in today's high-performance VLSI systems is the clock distribution network. Large clock nets perform like very long signal paths, making it hard for designers to keep the clock skew within tolerable limits. With clock frequencies in the multigigahertz range, the long wires needed for driving the clock signal onto increasingly large processor dies become a significant bottleneck, and more complex clocking and synchronization schemes are needed [1]. Furthermore, the clock skew is getting worse with each technology shrink due to the increasingly high process and system parameter variation (up to 100% for a 100-nm technology).

As recent case studies show [2], [3], the circuitry needed for driving such frequencies is already a dominant source for power consumption. Adding more and more transistors on a single die tends to worsen the problem—larger clock nets require more and more power.

To address these problems, two approaches are possible. The first option is to use fully asynchronous designs. While this has been tried successfully in isolated cases [3]–[5], design methodologies for asynchronous design are far from being mature and thus, far from widespread acceptance. Asynchronous designs are usually hard to test and validate, since it is almost impossible to generate any possible timing combinations. Furthermore, it would be impossible to reproduce the signal timings that cause a particular error, so debugging such designs might be very complicated.

Another alternative is to use globally asynchronous locally synchronous (GALS) architectures [6]–[11], which attempt to combine the benefits of both fully synchronous and asynchronous systems. A GALS architecture is composed of synchronous blocks that communicate with each other only on demand, using an asynchronous or mixed-clock communication scheme. Through the use of a locally generated clock signal within each individual domain, such architectures make it possible to take advantage of the industry-standard synchronous design methodology. Not requiring a global clock distribution network and deskewing circuitry, such systems have important advantages when compared to their fully synchronous counterparts. However, the overhead introduced by communicating data across clock domain boundaries may become a fundamental drawback, limiting the performance of these systems. Thus, the choice of granularity for these synchronous blocks or islands must be very carefully done in order to prevent the interdomain communication from becoming a significant bottleneck. At the same time, the choice of the interdomain communication scheme, as well as of the on-the-fly mechanisms for per-domain dynamic speed/voltage scaling (DVS) become critical when analyzing overall power-performance trends.

The contribution of the work proposed in this paper is twofold.

Manuscript received April 25, 2004; revised August 9, 2004. This work was supported in part by IBM Corporation under SUR Grant 4901B10170 and under SRC Grant 2001-HJ-898.

The authors are with the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: etalpes@ece.cmu.edu; dianam@ece.cmu.edu).

Digital Object Identifier 10.1109/TVLSI.2005.844305

- First, our approach allows for design space exploration for application-adaptive multiple-clock high-end processors by analyzing the available power/performance tradeoffs. Parameters under consideration include: choice of microarchitecture-level granularity for frequency islands; choice of mixed-clock communication scheme; choice of the application-adaptive mechanisms for fine-grain power management.
- Second, this paper introduces a new type of dynamic control strategy for application-adaptive multiple-clock processors that more closely matches the voltage/speed for various frequency/voltage islands to the ones required by the application workload. The newly proposed dynamic control mechanism relies on better matching front-end and back-end throughputs and using **dependency information** across clock domains to take better global decisions.

Our analysis shows that baseline GALS processors are not necessarily better in terms of *power consumption* than fully synchronous designs. Moreover, they are characterized by a decrease in performance due to the asynchronous communication overhead; this can also have adverse effects on overall energy consumption. The energy savings obtained by eliminating the global clock are, in many cases, offset by the additional power consumed due to longer execution times. However, the use of dynamic voltage scaling in GALS microprocessors makes them more appealing for many applications by providing additional power savings.

The paper is organized as follows. Section II details the motivation that lies behind our investigation of minimally clocked processors. In Sections III–V, we present the design exploration framework, the microarchitecture under consideration, and the various design choices that we can make in designing a GALS microprocessor. Section VI presents the theoretical aspects of scaling the frequency and the clock voltage, as well as the algorithms that we use. Section VII details our simulation framework and the experimental setup, while the results of all the tests are presented in Section VIII. Section IX presents previous work related to the aspects addressed in this paper, including the global clock distribution, asynchronous communication, and dynamic control strategies. Finally, we conclude with an analysis of the trends observed in the experimental results, together with our conclusions and directions for future research.

II. Motivation

As the transistor minimum feature is scaled down below 130 nm and the clock speed enters the gigahertz range, more problems related to distributing clock signals across an increasingly complex die tend to surface. In [2], the designers of the Alpha 21264 microprocessor present the problems encountered when migrating the design to a 180-nm copper process and how the nonscalability of wire delays is affecting this migration process.

While the long wires required for driving logical signals across the chip are affecting performance and must be shortened as much as possible to allow for further frequency increases, they also have a negative impact on the ability of driving global clock signals. An example of a very complex clock design can be found in another commercial processor, the Intel Pentium IV. For delivering a clock signal in excess of 2 GHz to the entire processor core, Intel has developed special circuitry for programmable clock deskewing [1].

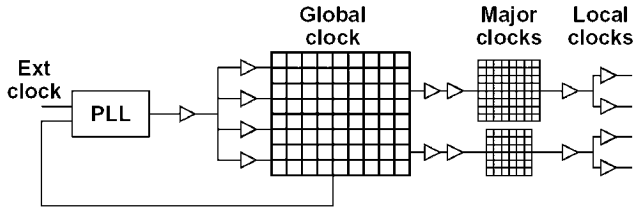

Fig. 1. Example of a hierarchical clock distribution network.

A. Clock Distribution

Generating a high-frequency clock signal and distributing it across a large die with low skew is a challenging task demanding a lot of design effort, die area, and power. In most cases, a phase-lock loop (PLL) generates a high-frequency clock signal from a slower external clock. A combination of metal grids and a tree of buffers is used to distribute the clock throughout the chip. Trees have low latency, dissipate less power, and use less wiring; but they need to be rerouted whenever the logic is modified even slightly, and in a custom-designed processor, this requires a lot of effort. Trees work well if the clock load is uniform across the chip area; unfortunately, most microprocessors have widely varying clock loads. Metal grids provide a regular structure to facilitate the early design and characterization of the clock network. They also minimize local skew by providing more direct interconnections between clock pins.

Moreover, clocking in most processors today is hierarchical. Fig. 1 shows an example of a hierarchical distribution network; several major clocks are derived from a global clock grid, and local clocks are in turn derived from the major clocks. This approach serves to modularize the overall design and to minimize the local skew inside a block. It also has the advantage that clock drivers for each functional block can be customized to the skew and drive requirements of that block; thus the drive on the global clock grid need not be designed for the worst-case clock loading.

With high-clock frequencies driving an ever-increasing number of transistor available on-chip, the power burned in the clock distribution network starts to become another limitation. For instance, the Alpha 21264 had a clock distribution network which consumed 24 W out of the processor's total power budget of 72 W. High power is not only an issue for mobile devices, but also for desktops or even servers; for instance in the case of the first generation Alpha processor, the single-line driver of the clock grid led to thermal management problems since the continuous switching at the driver led to a very high temperature at the chip's center and hence a high-temperature gradient [12]. Power has thus become a first-class constraint in clock system design.

Apart from the power burned in large distribution networks, they are also a problem for the quality of the clock signal. Restle *et al.* have argued in [13] that clock skew arises mainly due to process variations in the tree of buffers driving the clock. Since

device geometries will continue to shrink and clock frequencies and die sizes will continue to increase, global clock skew induced by such process variations can only get worse. Hence, we argue that we may not be far from the point where clock skew will become a significant proportion of the cycle time and thus, will directly affect performance. For instance, the first release of Intel's Itanium core had a clock skew that was 9% of the cycle time using traditional clock distribution techniques; using a network of active deskewing elements [14], the clock skew was reduced to 2% of the cycle time. While techniques like active deskewing help to push the envelope for clocked systems further, they come at additional cost in terms of die area and power dissipation. At some point, pushing the limits of global clock distribution networks will lead to diminishing marginal returns.

Another aspect that makes local clocks more promising is the increasing variation in process and operating parameters across various regions of the core. All these variations affect the maximum sustainable clock speed. As Skadron *et al.* show in [26], there can be variations of up to 15% in temperature between different microarchitecture components. While process variations are difficult to quantify at the microarchitectural level, it is obvious that some parts of the core will support a lower maximum frequency than others. Thus, a GALS microarchitecture will be able to take advantage of these variations, clocking some of the domains faster than the fully synchronous counterpart.

B. Fine-Grain Dynamic Voltage Scaling

Most applications running on core processors (be it high-end, embedded, or application specific) exhibit a wide range of run-time profiles, both within and across applications. This is mainly manifested via nonuniform resource usage, as well as bursty communication patterns among various parts of the pipeline. One such example is the fetch stage during which the I-cache is accessed and instructions are brought into the fetch queue. While an I-cache miss is being resolved, the issue and execution stages of the pipeline may proceed at their own pace until no more instructions advance in the pipeline due to pending dependencies. A similar situation may appear in case of nonblocking D-cache misses. In this case, multiple outstanding D-cache misses are resolved, while instructions may proceed normally through the pipeline. In addition, if there are instructions noncritical to the overall performance (e.g., infrequent floating-point operations in integer applications), their execution may proceed at a lower speed without significantly affecting performance. While the use of multiple-clock domains offers this flexibility, it also comes with additional potential for power savings. Locally synchronous blocks, whose speed may be gracefully scaled down, can also run at a lower voltage, providing additional power savings.

III. Design Exploration Framework

In this paper, we start with a fairly typical out-of-order, superscalar architecture and analyze the impact of various microarchitecture design decisions on the power-performance tradeoffs available in a multiple-clock processor. To this end, we assume a 16-stage pipeline that implements a four-way superscalar, out-of-order processor. While this pipeline is significantly longer than the ones studied before [8]–[11], we feel that this increased length resembles more accurately the pipelines projected for the next generation of commercial processors.

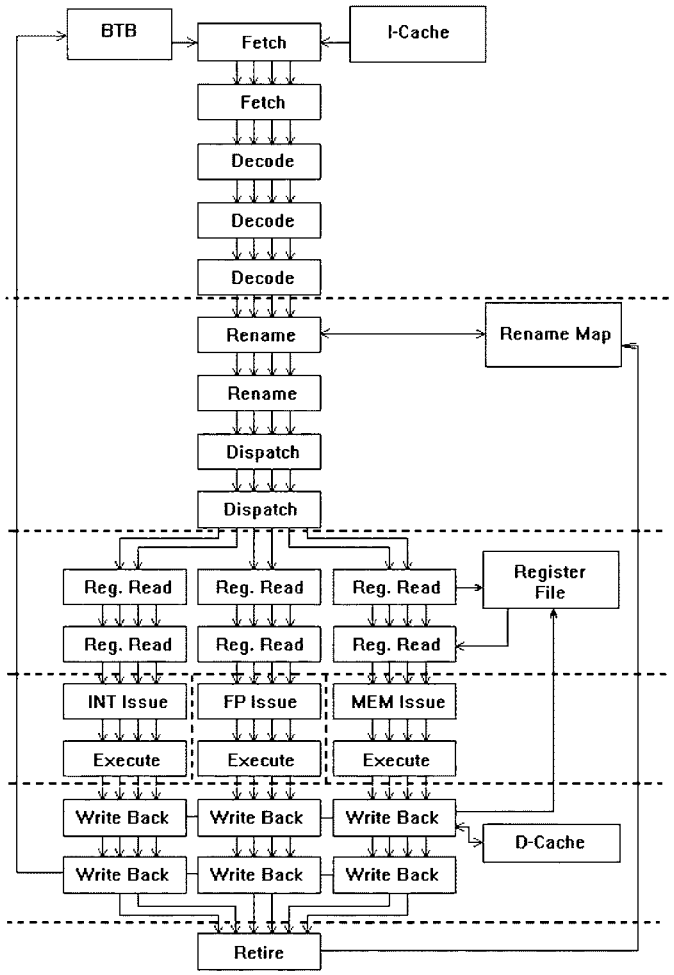

Fig. 2. Baseline microarchitecture.

The underlying microarchitecture organization is shown in Fig. 2. Groups of up to four aligned instructions are brought from the Level 1 Instruction Cache in the **Fetch** stages at the current program counter (PC) address, while the next PC is predicted using a G-share branch predictor [15]. The instructions are then decoded in the next three pipeline stages (named here **Decode**) while registers are renamed in the **Rename** stages. After the **Dispatch** stages, instructions are steered according to their type, toward the Integer, Floating Point, or Memory Clusters of the pipeline. The ordering information that needs to be preserved for in-order retirement is also added here. In **Register Read**, the read operation completes and the source operand values are sent to the execution core together with the instruction opcode.

Instructions are placed in a distributed Issue Buffer (similar to the one used by Alpha 21264) and reordered according to their data dependencies. Independent instructions are sent in parallel to the out-of-order execution core. The execution can take one or more clock cycles (depending on the type of functional unit that

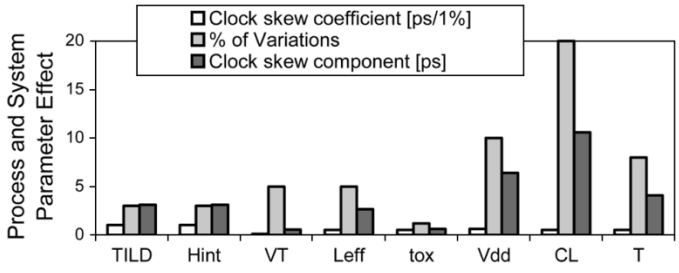

Fig. 3. Total clock skew for Alpha 21264 as function of interconnect parameters T_{ILD} (interlevel dielectric thickness), H_{int} (interconnect thickness), technology parameters V_t (threshold voltage), L_{eff} (transistor channel length), T_{ox} (oxide thickness) and system parameters V_{dd} (supply voltage), C_L (load capacitance), and T (temperature) [19].

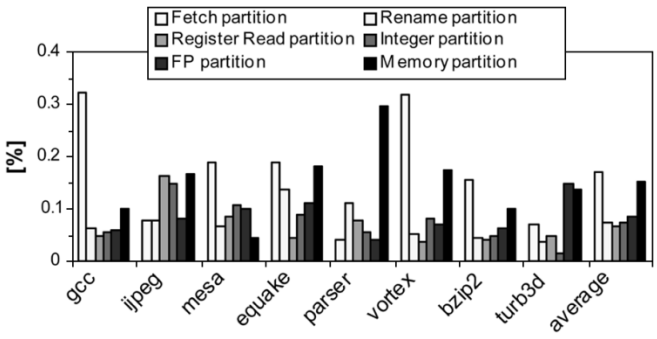

Fig. 4. Performance increase for 1% per-domain speedup.

executes the instruction) and the results are written back to the register file in the **Write Back** stages. Finally, the instructions are reordered for in-order retirement, according to the tags received during **Dispatch**. Branches are resolved in **Write Back**, thus providing a minimum mispredict penalty of 14 cycles.

Of extreme importance for our design exploration is the choice of various design knobs that impact the overall power-performance tradeoffs in GALS processors. Since our primary focus is at the microarchitecture level, we chose to omit several lower-level issues in our study. Some areas which have been dealt with in detail elsewhere are as follows.

- **Local clock generation:** Each clock domain in a GALS system needs its own local clock generator; ring oscillators have been proposed as a viable clock generation scheme [16], [17]. We assume that we can use ring oscillators in each synchronous block in the GALS processor.
- **Failure modeling:** A system with multiple-clock domains is prone to synchronization failures; we do not attempt to model these since their probabilities are rather small for the communication mechanisms considered (but nonzero) [17], [29] and our work does not target mission-critical systems.

Instead, we are focusing on the following microarchitecture design knobs:

- the choice of the **communication** scheme among frequency islands;
- the granularity chosen for the **frequency islands**;
- the **dynamic control strategy** for adjusting voltage/speed of clock domains so as to achieve better power efficiency.

We detail in the following how each of these microarchitecture-level design decisions may potentially impact the overall power efficiency or performance profile for the GALS architecture.

IV. Choice of Clock Domain Granularity

To assess the impact of introducing a mixed-clock interface on the overall performance of our baseline pipeline, we assume that the pipeline is broken into several synchronous blocks. The natural approach—minimize the communication over the synchronous blocks boundaries—does not necessarily work here. An instruction must pass through all the pipeline stages in order to be completed. Thus, we have to find other criteria for determining these synchronous islands.

One possible criterion for placing the asynchronous interfaces is to minimize clock skew, thus allowing for faster local clocks. In [19], the authors propose a model for the skew of the on-chip clock signal. By applying the model to an Alpha 21264 microprocessor, one can evaluate the contribution of different microarchitectural and physical elements in increasing the skew and thus in limiting the clock frequency.

As it can be seen in Fig. 3, the main components affecting clock skew are system parameter variations (supply voltage V_{dd}, load capacitance C_L, and local temperature T), especially the variations in C_L. Since the microarchitecture described in Fig. 2 exhibits a large variation in the number of pipeline registers clocked, a possible placement of the asynchronous interfaces is shown dotted. To evaluate which partitions would enable higher performance if sped up independently, we show in Fig. 4 the "speedup coefficient" that characterizes the amount of overall speedup achieved by each domain when its local clock frequency is increased by 1%. As it can be seen, across our set of benchmarks, the most significant speedup can be achieved by increasing the clock speed in the Fetch or Memory, followed by Integer and FP partitions. Thus, these modules should be placed in separate clock domains if possible, since individually they could provide significant performance increase if sped up.

To break the execution core, we have used the same partitioning scheme that was proposed before [9], [11]. By starting with a processor with separate clusters for integer, floating-point and memory execution units (much like the Alpha 21264 design) we can naturally separate these clusters in three synchronous modules. The drawback of this scheme is that it increases significantly the latency of forwarding a result across the asynchronous interface toward another functional unit. This effect can be seen mainly in load-use operations that are executed in separate clock domains, imposing a significant penalty on the overall performance in some programs.

To limit the latency of reading/writing data from the registers, the Register Read and the Write Back stages must be placed together, in the same synchronous partition as the Register File. Following the same rationale, the Rename and Retire stages need both to access the Rename Table so they must be placed in the same partition.

Following these design choices, we can now split the pipeline into at least four clock regions. The first one is composed of the Fetch stage, together with all the branch prediction and instruction cache logic. The two Decode stages can be included either

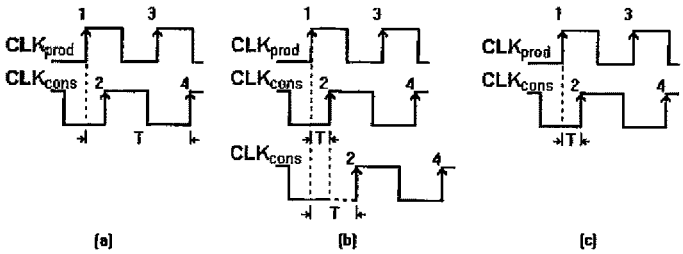

Fig. 5. Timing diagram. (a) Synchronizers. (b) Plausible clocks. (c) Arbiters.

in the first clocking region or in the second one—all the instructions that pass from Fetch to Decode will be passed down the pipeline to Rename.

Given that we intend to limit the clock skew as much as possible by limiting load capacitance variations and considering the bitwidth increase of the pipeline after Decode, we introduce here an asynchronous boundary. The second clocking region will be organized around the Renaming mechanism, and it will also contain the Reorder Buffer and the Retire logic. Given the variation in the register width for the rest of the pipeline, an asynchronous boundary can be also introduced after Dispatch. The third clocking region must be organized around the Register File, including the Register Read and Write Back stages. Finally, the out-of-order part of the pipeline (the Issue logic and the execution units) is split into separate clusters that amount for three different clock regions. The forwarding paths can thus be internal—toward a unit with the same type and placed in the same clock region—or external—toward other clock regions.

V. Choice of Interdomain Communication Scheme

One of the most important aspects of implementing a GALS microprocessor is choosing an asynchronous communication protocol. For high-performance processors, the bandwidth and latency of the internal communication are both important and a tradeoff is harder to identify. Several mechanisms have been proposed for asynchronous data communication between synchronous modules in a larger design [17].

The conventional scheme to tackle such problems is the extensive use of synchronizers—a double latching mechanism that conservatively delays a potential read, waiting for data signals to stabilize as shown in Fig. 5(a). Even though data are produced before time step 2, the synchronizer enforces its availability at the consumer only at time step 4. This makes classical synchronizers rather unattractive, as their use decreases performance and the probability of failure for the whole system rises with the number of synchronized signals.

Pausable clocks [Fig. 5(b)] have been proposed as a scheme that relies on stretching the clock periods on the two communicating blocks until the data are available or the receiver is ready to accept it [7]. If T is greater than an arbitrary threshold, then the read can proceed; otherwise the active edge 2 of the consumer clock is delayed. While the latency is better in this approach, it assumes that asynchronous communication is infrequent. Stretching the clock is reflected in the performance of each synchronous block, and thus it is most effective when the two blocks use a similar clock frequency. It can also be an effective mechanism when the whole block must wait anyway until data are available.

Another approach is to use arbiters for detecting any timing violation condition—Fig. 5(c). In this case, data produced at time step 1 may be available at time step 2 if T is larger than a certain threshold. While the mechanism is conceptually similar to that of synchronizers, it offers a smaller latency.

Asynchronous FIFO queues were proposed [20], using either synchronizers or arbiters. Such an approach works well under the assumption that the FIFO is neither completely full, nor completely empty. The scheme retains the extra latency introduced by the use of synchronizers, but improves the bandwidth through pipelining. For the nominal operation of this structure (when the FIFO is neither empty, nor full), a potential read is serviced using a different cell than the one handling the next write, so both can be performed without synchronization.

All these mechanisms reduce the error probability to very low levels, but they cannot ensure that metastability will never occur. However, as Ginosar showed recently [29], the error rate can be reduced as much as it is desired. Typically, the mean time to failure is on the order of hundreds of years, at least an order of magnitude higher than the time between soft error occurrences [31] or the expected life of the product.

VI. Choice of Dynamic Control Strategy

One of the main advantages offered by the GALS approach is the ability to run each synchronous module at a different clock frequency. If the original pipeline stages are not perfectly balanced, the synchronous blocks that we obtain after the partitioning can naturally be clocked at different frequencies. For example, if the longest signal path belongs to the Register Renaming module, in the GALS approach we could potentially run the Execution Core at a higher clock frequency than the fully synchronous design.

Furthermore, even if we start with a perfectly balanced design (or we resize transistors in order to speed up the longer signal paths), we can slow down synchronous blocks that are off the critical path while keeping the others running at nominal speed. The slower clock domains could also operate at a lower supply voltage, thus producing additional power savings. Since energy consumption is quadratically dependent on $V_{\rm dd}$, reducing it can lead to significant energy benefits, while latency (D) is increased accordingly

$$D \propto \frac{V_{\rm dd}}{(V_{\rm dd} - V_t)^{\alpha}} \tag{1}$$

where α is a technology-dependent factor, which is 1.2 to 1.6 for current technologies [22] and V_t is the threshold voltage.

To this end, we provide some theoretical results on the efficiency of using fine-grained dynamic voltage scaling in multiple-clock dynamic voltage cores. We assume that the following hold.

- We consider the case of pipelined cores in which each clock domain i has n_i pipe stages, with L_i total load on the critical path of the clock domain.

- The switched capacitance for each clock domain is C_i. The voltage and clock speed associated with clock domain i are $V_{\mathrm{dd},i}$ and f_i

$$f_i \propto \frac{(V_{\mathrm{dd},i} - V_t)^\alpha}{V_{\mathrm{dd},i}}.$$

Lemma 1: Assuming a linear pipeline organization (without feedback paths), a synchronous design achieves better energy savings than a corresponding multiclock design under the same slowdown factor per computation, **if** the switched capacitance per clock domain C_i is proportional to the total load on the critical path L_i.

Proof: We assume that the base pipeline is run at a voltage V_{dd} such that the latency per computation is the same as the DVS-enabled, GALS pipeline with k clock domains, each running at voltage $V_{\mathrm{dd},i}$ $(i = 1, 2, \ldots, k)$. We also assume that between clock domains $(i, i+1)$ there exists a load overhead of l_i due to synchronization and voltage level conversion. To achieve the same latency, the following has to hold:

$$(L_1 + l_1)\frac{V_{\mathrm{dd},1}}{(V_{\mathrm{dd},1} - V_t)^\alpha} + (L_2 + l_2)\frac{V_{\mathrm{dd},2}}{(V_{\mathrm{dd},2} - V_t)^\alpha} + \cdots + L_k\frac{V_{\mathrm{dd},k}}{(V_{\mathrm{dd},k} - V_t)^\alpha} = (L_1 + \cdots + L_k)\frac{V_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_t)^\alpha}$$

or

$$\sum_{i=1}^{k} \frac{L_i V_{\mathrm{dd},i}}{(V_{\mathrm{dd},i} - V_t)^\alpha} < \left(\sum_{i=1}^{k} L_i\right) \frac{V_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_t)^\alpha}. \quad (2)$$

Consider the function

$$f(x) = \frac{\sqrt{x}}{(\sqrt{x} - V_t)^\alpha}$$

which is convex. Thus

$$\sum_{i=1}^{k} L_i f\left(V_{\mathrm{dd},i}^2\right) \geq \left(\sum_{i=1}^{k} L_i\right) f\left(\frac{\sum_{i=1}^{k} L_i V_{\mathrm{dd},i}^2}{\sum_{i=1}^{k} L_i}\right). \quad (3)$$

Function f is also monotonically decreasing; thus, from (2) and (3) and from the proportionality of C_j with L_j for all clock domains $j = 1, 2, \ldots, k$, i.e.,

$$\frac{C_j}{\sum_{i=1}^{k} C_i} = \frac{L_j}{\sum_{i=1}^{k} L_i}$$

we get

$$\sum_{i=1}^{k} C_i V_{\mathrm{dd},i}^2 \geq \left(\sum_{i=1}^{k} C_i\right) V_{\mathrm{dd}}^2$$

which shows that the synchronous pipeline performs better energy-wise than the GALS version for the same slowdown factor (or performance penalty) under the assumptions of Lemma 1. □

This result is in fact a generalization of the optimal voltage-scheduling problem for applications with hard real-time constraints. Ishihara and Yasuura showed [23] that using a single voltage level in a dynamic voltage-scheduling environment achieves the best energy savings under given performance constraints.

The general problem is, however, far from being that simple. In fact, in most designs, there is no relationship between the load on the critical path and the corresponding switched capacitance for that clock domain. We show in the following a necessary condition for fine-grained dynamic voltage assignment to different clock domains for achieving better energy savings for GALS when compared to synchronous pipelines, in the special case of a two-clock domain implementation.

Lemma 2: In the case of a two-clock domain pipeline, if the ratio of the switched capacitance per clock domain C_i and the load on the critical path L_i is different in each clock domain, the GALS pipeline cannot achieve better energy savings than the synchronous counterpart for the same slowdown factor, **unless** the lower voltage is applied to the stage which satisfies the relation

$$\frac{C_1}{C_1 + C_2} > \frac{L_1}{L_1 + L_2}.$$

Proof: As in Lemma 1, we have

$$L_1 V_{\mathrm{dd},1}^2 + L_2 V_{\mathrm{dd},2}^2 < (L_1 + L_2) V_{\mathrm{dd}}^2$$

if (2) is satisfied for $k = 2$. For GALS to be better in terms of energy than the synchronous counterpart, we need

$$\frac{L_1 V_{\mathrm{dd},1}^2 + L_2 V_{\mathrm{dd},2}^2}{L_1 + L_2} \geq V_{\mathrm{dd}}^2 \geq \frac{C_1 V_{\mathrm{dd},1}^2 + C_2 V_{\mathrm{dd},2}^2}{C_1 + C_2}.$$

From the above, we get

$$V_{\mathrm{dd},1} \leq V_{\mathrm{dd},2}. \quad \square$$

This result confirms the intuition that in order to achieve energy savings in GALS versus a DVS-enabled synchronous design for the same slowdown factor, lower voltages should be applied to the more power consuming clock domains if they are also contributing the least to the end-to-end latency per computation.

The above results can also be extended to linear pipelines with feedback paths, by lumping strongly connected components in the component graph into single nodes and applying the above results on single, linear pipelines [30]. Based on the above theoretical results, if multiple computations are executed in parallel (as in the case of high-end processors), computations off the critical path can be executed at a lower voltage, without affecting the overall performance, for a given slowdown factor.

To exploit nonuniform program profiles and noncriticality of various workloads, different schemes have been previously proposed for selecting the optimal frequency and voltage supply in a GALS processor. In [9], a simple threshold-based algorithm is used for selecting the best operating point for modules that have a normal and a low-power mode. The algorithm monitors the average occupancy of each issue window and can decide to switch the module to a low-power mode when this occupancy drops below a predefined threshold, or ramp the voltage up when a high-threshold is exceeded. For each issue window (integer, floating point, and memory), the algorithm is as follows.

```
1. if (occupancy > MODULE_UP_THRESHOLD) &&
   (module_speed == LOW_SPEED)
2.    module_speed = HIGH_SPEED;
3. if (occupancy < MODULE_DOWN_THRESHOLD) &&
   (module_speed == HIGH_SPEED)
```

```
4.    module_speed = LOW_SPEED;
```

A more complex model is proposed in [11]. Here, an attack–decay algorithm is assumed for selecting the best operating point for processors that offer a wide range of frequencies and supply voltages. The algorithm monitors the instruction window occupancy and, based on its *variation*, decides whether the frequency should be increased or decreased. Any significant variation triggers a rapid change of the clock frequency in order to counter it. For small or no variations, the clock frequency is decayed continuously, while monitoring performance.

```
1.   if ((prev_occupancy − occupancy > THRESHOLD)
          && (old_IPC−IPC < THRESHOLD))
2.     module_speed− = ATTACK;
3.   else
4.   if (occupancy−prev_occupancy > THRESHOLD)
5.     module_speed+ = ATTACK;
6.   else
7.   if (module_speed == HIGH_SPEED) &&
      (counter > MAX_LIMIT)
8.    module_speed− = ATTACK;
9.   else
10. if (module_speed == LOW_SPEED) &&
      (counter > MAX_LIMIT)
11.    module_speed+ = ATTACK;
12. else
13. module_speed− = DECAY;
14. if (module_speed <= LOW_SPEED) {
15.    module_speed = LOW_SPEED;
16.     counter++;
17. }
18. if (module_speed >= HIGH_SPEED) {
19.     module_speed = HIGH_SPEED;
20.     counter++;
21.  }
22. prev_occupancy = occupancy
```

Looking at the superscalar, out-of-order microarchitecture, we note that the instruction window occupancy is not the only significant aspect that can be considered for deciding a switch. Even though an instruction window could have high occupancy, this could be due to a bottleneck in another cluster. If load operations are delayed, it is very likely that instructions will accumulate in the integer cluster as well. However, speeding up the clock in the integer domain will not improve performance. In this case, taking decisions based only on local issue queue occupancy will not help and the number of *interdomain data dependencies* (i.e., the number of pending dependencies to or from another clock domain) may be more significant than the issue window occupancy.

Furthermore, both [9] and [11] allow dynamic voltage scaling just for the execution core, assuming that the clock speed of the front-end is critical for the overall performance, and thus should not be reduced. However, there are large portions of the core where the usable parallelism [defined here in terms of instructions committed per clock cycle (IPC)] is significantly smaller than the theoretical pipeline throughput. In these cases, it makes sense to reduce the speed of the front-end since it produces more instructions that can be processed by the back-end.

In order to study the efficiency of these observations, we propose to modify the previously described methods to include both information about the number of *interdomain dependencies* and dynamic voltage-scaling algorithm for the *front-end* of the pipeline.

Applying our modification on the threshold-based algorithm for the execution modules, we obtain the following.

```
1. if ((inter_domain_dependencies+occupancy)
       > MODULE_UP_THRESHOLD) &&
       (module_speed == LOW_SPEED)
2.    module_speed = HIGH_SPEED;
3. if ((inter_domain_dependencies+occupancy)
       < (MODULE_DOWN_THRESHOLD) &&
       (module_speed == HIGH_SPEED)
4.    module_speed = LOW_SPEED;
```

For the front-end clock domain, the algorithm is as follows:

```
1. if (front_end_throughput/
            back_end_throughput)
        < FRONT_END_UP_THRESHOLD
2.    module_speed = HIGH_SPEED;
3. if (front_end_throughput/
            back_end_throughput
        > FRONT_END_DOWN_THRESHOLD)
4.    module_speed = LOW_SPEED;
```

A similarly modified algorithm was derived from the attack–decay approach, using the same combined metric and allowing for variations in the front-end frequency. In this study we compare both the methods proposed in [9] and [11] and their modified counterparts that take into consideration the interdomain dependencies.

VII. Simulation Framework

To measure the impact on performance, as well as on the power required by our GALS microarchitecture, we have implemented a cycle-accurate simulation model of the original pipeline (presented in Fig. 2). Our simulator is based on SimpleScalar [21], but reflects the target pipeline more accurately. As opposed to SimpleScalar, it uses normal pipeline registers, separate Instruction Windows for each of the three execution clusters and a Retire Buffer. The register renaming mechanism chosen is similar to the one used by the MIPS R10000 processor. We have also moved the execution from Decode (as it is done in SimpleScalar) to the Execute stage, to better reflect the behavior of the pipeline.

In order to model a GALS environment without any global synchronization point, we have developed an event-driven simulation engine. Events associated with each frequency island are synchronized with a given clock signal, randomly started at the

TABLE I
TEMPERATURE AND CLOCK SPEED VARIATIONS

Domain	Temperature (C)	Relative clock speed
Fetch	75.7	1.11
Rename / Dispatch	74.2	1.13
Register Read / Write	84.5	1.00
Integer	77.3	1.09
Floating point	72.7	1.15
Memory	79.6	1.06

beginning of the simulation. This event-driven simulation engine allows for any mixture of clocks running at different speeds and with different starting phases.

We have used the Wattch framework [18] to include power models in our design exploration framework. These power models (including the ones for the asynchronous communication) are integrated in our baseline and GALS simulator versions to provide energy statistics. To this end, we have assumed that each component not used can be clock-gated, on a per-cycle basis, with an overhead given by the leakage current. The leakage power is obtained using the methodology proposed by Butts and Sohi [27]. This model is based on the estimation of the total number of devices for the entire processor, as well as their type. The normalized leakage current per device was estimated as in [28].

In addition to modeling the switching capacitance of memories and buses inside the processor, we have also modeled the switched capacitance of the clock grids. We have assumed a clock distribution hierarchy resembling the one used by the Alpha 21264 processor. We have modeled one global clock grid and local clock grids corresponding to each of the synchronous domains. The area and metal density for each clock grid are the ones published for the Alpha 21264 processor.

Since it is very difficult to model process variations at the microarchitectural level, we only model the effect of the die temperature on the maximum clock frequency. For this, we use the results from [26], which show a temperature variation of up to 15% across different modules (Table I). For computing the relative speeds of the various clock domains, we have used the model proposed in [32]

$$f = \frac{1}{T \cdot L_d}$$

where T is the temperature and L_d is the logic depth (that we assume identical across the different pipeline stages (given that the original was a balanced pipelined implementation).

While these are not the only effects that can impact the maximum clock speed, they provide a good idea about the usefulness of having independent clock grids for each domain.

The parameters for the microarchitecture under consideration are presented in Table II. In our experiments, we have used integer and floating-point benchmarks from both SPEC95 and SPEC2000 suites. For all experiments, we have skipped the first 500 million instructions and then continued simulation for another 50 million instructions. For the GALS case, since the local clock signals are randomly staggered, simulations in this case were run three times, averaging the results. To compare the various dynamic control strategies, we have used an arbiter-based

TABLE II
MICROARCHITECTURE PARAMETERS

Parameter	Value
Pipeline	16 stages, 4 way out-of-order
Instruction Window	64 entries – 32 Int, 16 FP, 16 Mem
Load / Store Queue	32 entries
I-Cache	32k, 2 way set-associative, 1 cycle hit time, LRU replacement
D-Cache	32k, 4 way set-associative, 2 cycles hit time, LRU replacement
L2 Cache	Unified, 256k, 4 way set-associative, LRU replacement
L2 access time	10 cycles
Memory access time	100 cycles
Functional Units	4 Integer ALUs, 2 Integer MUL/DIV 2 Memory ports 2 FP Adders, 1 FP MUL/DIV
Branch Prediction	G-share, 11 bits history, 2048 entries
Technology	0.13 um (high speed STMicro) technology
V_{dd}	1.8V
V_t	0.2V
Normalized leakage current per device [27]	80nA
Clock Speed / V_{dd}	250MHz – 1000MHz, 0.7V – 1.8V
Thresholds	Integer – 9, 12 Memory – 9, 12 FP – 6, 9 Front-end: Twice as fast as the commit Attack-decay: 10% occupancy/IPC variation
Frequency levels for the threshold – based algorithm	Integer – High 1GHz, Low 750MHz FP – High 1GHz, Low 250MHz Memory – High 1GHz, Low 500MHz
Frequency / Voltage levels for attack-decay	12, with V_{dd} levels equally spaced between 0.7V and 1.8V computed according to equation (1)

asynchronous FIFO communication. Similar to the synchronous pipeline, we assume that the active clock edge in the producer signals the moment when data are available for reading (a consumer cycle can start). Thus, a subsequent active edge in the consumer can be accepted as a valid request, the setup time being already observed during the producer cycle.

VIII. EXPERIMENTAL RESULTS

Assuming that all modules will be clocked at the same frequency as the synchronous design, we have analyzed the impact of the granularity and asynchronous communication choice, as well as the impact of the dynamic control strategy on the overall performance and power. Using the arbiter-based asynchronous FIFOs for interdomain communication, in the six-clock domain case a performance penalty of up to 18% is observed. As shown in Fig. 6, the performance hit increases with the number of asynchronous interfaces—from an average of 5% for four clock domains case to almost 10% for six-clock domains.

In terms of power consumption, the GALS processor is more efficient due to its lack of global clock grid. However, due to an increased execution time, even though the power per cycle is significantly improved, the total energy per task required by the GALS processor is only slightly decreased (Fig. 7). Since the six-clock domain microarchitecture would only allow an independent speedup in the register file clock domain (that, according to Fig. 4, does not have a significant impact in overall

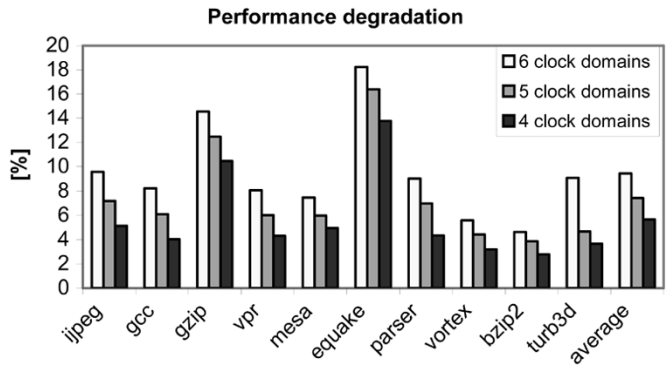

Fig. 6. Performance degradation for a GALS microarchitecture. In all cases, the baseline is the fully synchronous microarchitecture, using the same clock frequency as the GALS domains.

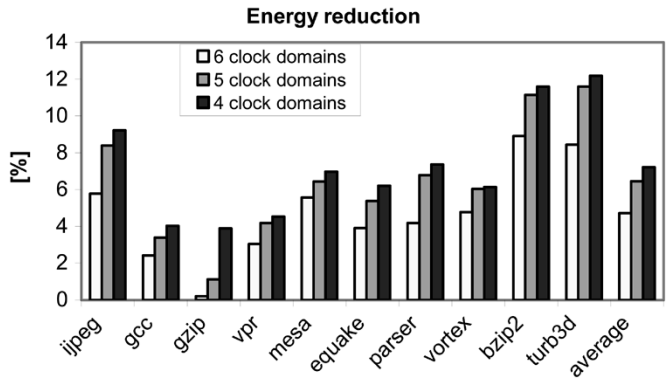

Fig. 7. Energy reduction for a GALS design for 4–6 partitions.

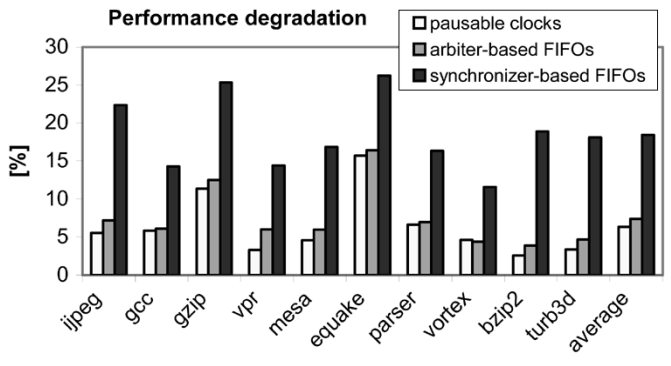

Fig. 8. Performance degradation using different mechanisms for the asynchronous communication. In all cases, the baseline is the fully synchronous microarchitecture, using the same clock frequency as the GALS domains.

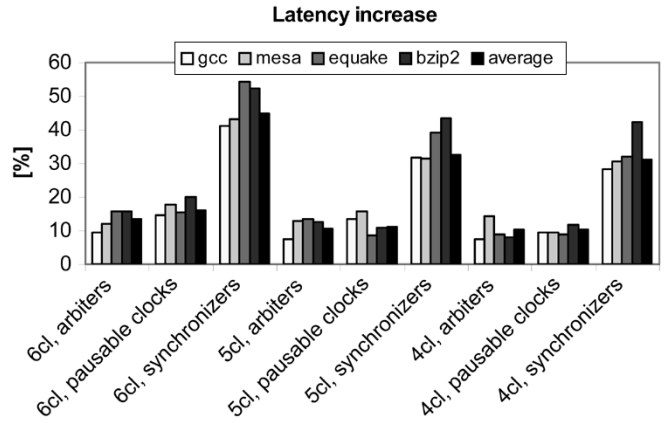

Fig. 9. End-to-end latency increase for various GALS configurations.

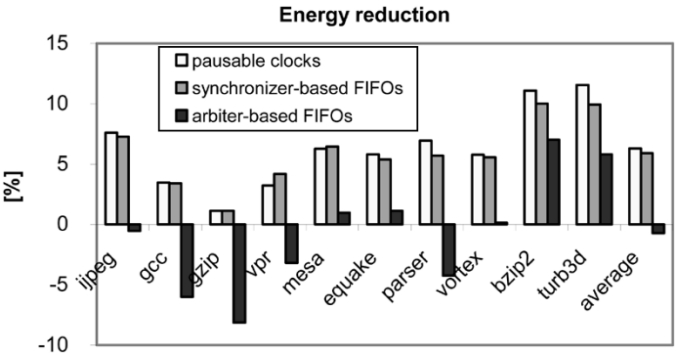

Fig. 10. Energy reduction for a GALS design using different asynchronous communication mechanisms.

performance), it seems that the best choice is a five clock domain design which would allow the Fetch and Execution cores to run at possibly different speeds. In this setup, the GALS processor requires 7% less energy for completing the same task.

In order to evaluate the effectiveness of each asynchronous communication scheme, we have considered arbiter-based and synchronizer-based FIFOs, as well as pausable clocks.

Since the pausable clocks approach effectively delays an active clock edge when the synchronization cannot be done, the effective producer-to-consumer latency of this approach is minimal. For arbiters, however, a failed attempt to read data must be followed by a normal consumer cycle whose active edge is completely asynchronous with respect to the producer clock. This introduces an additional average delay of 0.5 cycles. Using the same reasoning, the one cycle latency associated with the synchronizers is actually observed as 1.5 cycles on average when coupled with random starting phases for the producer and consumer clocks.

As expected, the largest drop in performance is observed when using synchronizer-based FIFOs. In this case, the performance decrease observed by a five clock domain design can be up to 26%, with an average of 18.4% (Fig. 8). The smallest hit in performance is achieved when using pausable clocks, with an average of 6.3%.

For a better understanding of these results, we looked at how end-to-end instruction latency varies with the number of asynchronous interfaces and with the communication mechanism. While this latency increases by around 12% for both pausable clocks and arbiters, instructions can spend up to 50% more time between Fetch and Retire when the communication is based on synchronizers (Fig. 9). This additional latency increases both the mispredict penalty and the bypass latency, affecting the overall performance.

In terms of the overall energy requirement, none of the three mechanisms brings a very significant improvement. While a small improvement can be noticed for pausable clocks and arbiters (6% and 6.5% on average), the case of synchronizer-based FIFOs leads to an increase of 1% in the energy demands. The energy results for our experiments are presented in Fig. 10.

As expected, our results show that the mechanisms introducing the smallest additional latency (pausable clocks and

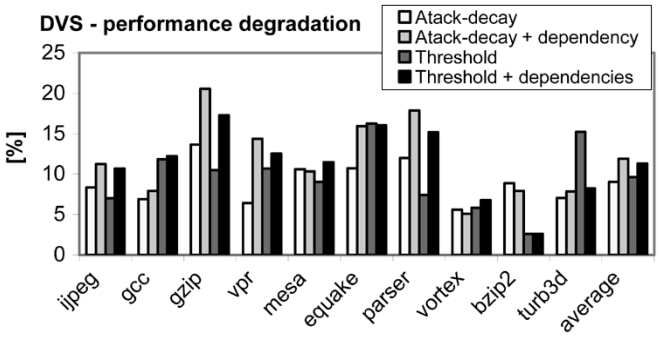

Fig. 11. Performance of the DVS-enabled GALS design (five clock domains, arbiter-based FIFOs). In all cases, the baseline is the fully synchronous microarchitecture, using the maximum clock frequency supported by the GALS modules.

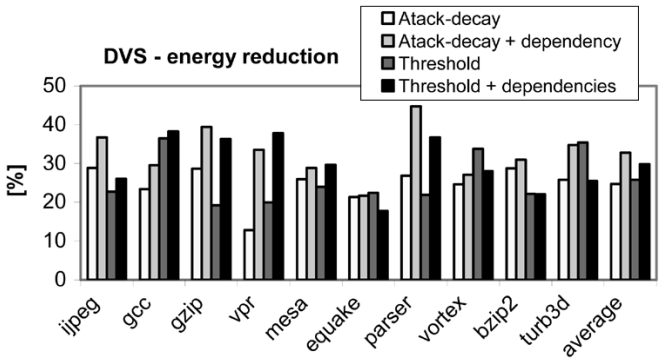

Fig. 12. Energy consumption of the DVS-enabled design (five clock domains, arbiter-based FIFOs).

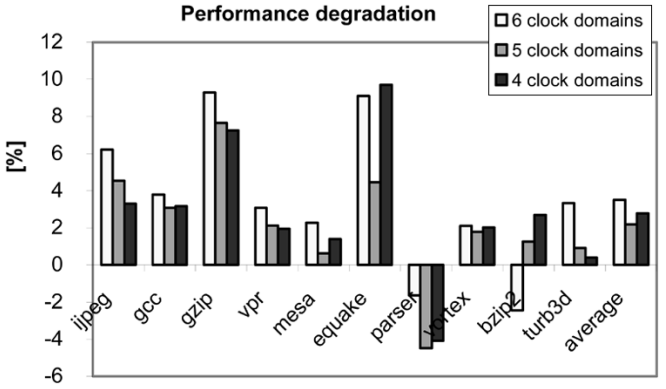

Fig. 13. Performance degradation for the GALS microarchitecture. In all cases, the GALS domains use a faster clock speed (Table II), and the baseline is the fully synchronous microarchitecture.

synchronizer-based FIFOs) are performing very well, both in terms of performance and power consumption. The arbiter-based FIFO behaves very close, as it offers much better bandwidth at the expense of a slightly increased latency. All the FIFO-based mechanisms introduce an additional load on the clock generation network, by increasing the number of interstage latches. However, the effect is very limited, our tests showing an average of 8% increase in clock power, which translates into less than 2% increase for the entire core.

By essentially blocking the clock signal in the consumer to observe the synchronization latency, pausable clocks make it more difficult to use of different speeds across domains. Thus, for implementing dynamic control of local speeds/voltages we chose to use arbiter-based FIFOs.

The first important advantages of a GALS microarchitecture is its ability to independently scale the voltage and clock frequency (DVS) for each of its synchronous partitions. We evaluate both the performance and the power requirements of such a processor using two previously proposed algorithms: the threshold based one that can select the best out of two operating points [9] and the attack–decay algorithm that assumes a much larger set of operating points [10]. For both of them, we test the efficiency of focusing on the average Instruction Window occupancy (threshold and attack–decay) or on the number of interdomain dependency (Threshold + dependency and attack-decay + dependency).

All the dynamic control mechanisms evaluated introduce an average drop in performance of 9% to 12% when compared to the synchronous baseline architecture (Fig. 11). A very interesting aspect is that introducing the interdomain dependency information does not automatically improve the performance. In both cases, performance was slightly worse when using dependency information. This behavior was mostly caused by our decision to allow dynamic voltage scaling in the front-end as well. However, this additional drop in performance (up to 4%) is offset by better energy efficiency (Fig. 12).

In terms of energy requirement, the DVS-enabled GALS design saves between 8% and almost 45% when compared to the baseline synchronous case. On average, the four DVS algorithms that we study offer energy savings between 25% and 33%, at the expense of about 10% drop in overall performance.

A second advantage of a GALS microarchitecture is that each clock domain comes with its independent clock distribution network, being able to take advantage of higher frequencies than the fully synchronous processor. A GALS microarchitecture can take advantage of any slight unbalances in the design, but it can also adapt to different operating conditions.

To evaluate the response of a GALS microarchitecture to such conditions, we consider on-die temperature variation [26] as a rationale for correlating clock speeds with operating temperatures. As shown before [26] (Table I), there exists a significant temperature variation across different modules, variation that translates into different maximum clock frequencies. This is probably a pessimistic assumption, since it does not take into consideration potential process variation as well as other system parameter factors that may affect local clocking speeds. However, it offers a good idea about the effects of the faster clock domains.

When allowing each clock domain to run at its own maximum frequency (as in Table I), the overall performance increases significantly. The effect of the asynchronous communication is offset and the GALS microarchitecture performs within 2%–3% of the fully synchronous baseline. In some cases (e.g., *parser, bzip2*), the performance of the GALS microarchitecture can actually be better (Fig. 13).

Another interesting aspect is that the results do not show the same regular pattern any more. In some cases, a version with more clock domains can be faster than one with less. This effect

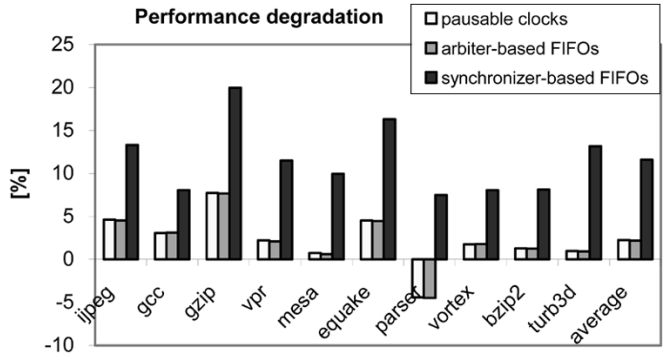

Fig. 14. Performance degradation for a GALS microarchitecture. In all cases, the GALS domains use a faster clock speed (Table II), and the baseline is the fully synchronous microarchitecture.

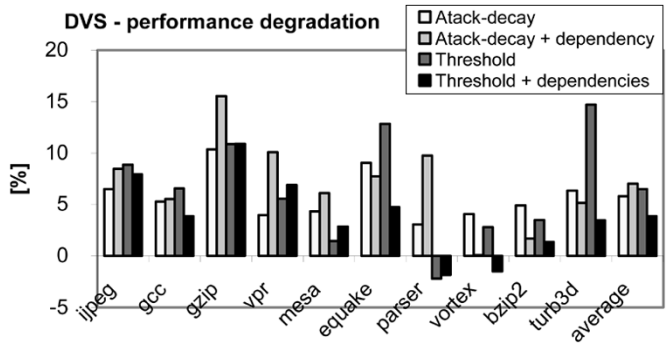

Fig. 15. Performance of the DVS-enabled GALS design (five clock domains, arbiter-based FIFOs). In all cases, the GALS domains use a faster clock speed (Table II), and the baseline is the fully synchronous microarchitecture.

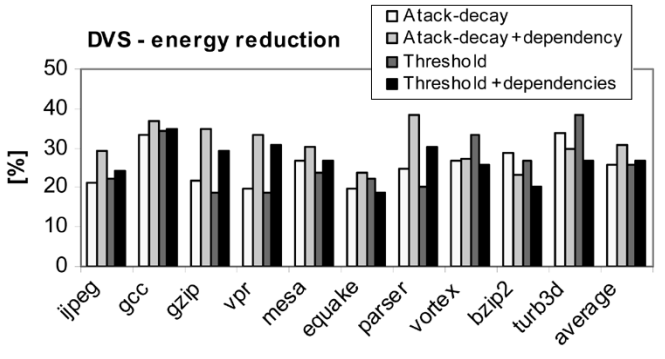

Fig. 16. Energy consumption of the DVS-enabled design (five clock domains, arbiter-based FIFOs). In all cases, the GALS domains use a faster clock speed (Table II), and the baseline is the fully synchronous microarchitecture.

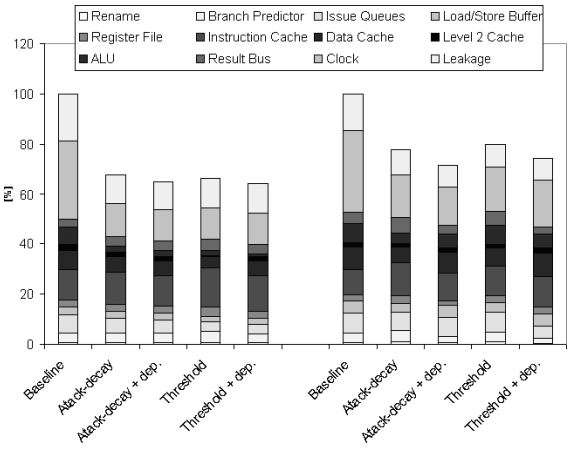

Fig. 17. Energy breakdown for the DVS-enabled design, with five clock domains and arbiter-based FIFOs. Benchmarks included are (left) *gcc* and (right) *mesa*. The breakdown starts with the energy values for Rename and Branch predictor (bottom of each vertical bar) and ends with the values for Clock energy and Leakage (top of the bars).

is caused by the fact that when two clock domains are combined, the resulting one has to work at the speed of the slowest one. For example, for obtaining the five-clock domain version we combine the Rename/Dispatch with the Register File, and the resulting domain works at the speed of the Register File (the hottest and thus, the slowest one in our tests).

We see a similar picture when looking at different communication mechanisms. The performance reduction is much smaller, but it maintains a similar trend, the lower latency mechanisms performing better. These results are presented in Fig. 14.

When enabling dynamic voltage scaling, we assumed that all clock frequency levels are scaled by the same ratio as in Table I. Thus, the performance gap narrows, the DVS-enabled GALS microarchitecture performing within 4%–7% of the fully synchronous baseline (Fig. 15).

In our tests, we assume that clock speed improvements for any domain can be obtained without resorting to increases in the corresponding voltage level. Thus, the energy level remains fairly similar, with reductions within the 25%–31% range. These results are presented in Fig. 16.

The final analysis includes the breakdown of the overall energy budget for both the baseline and the DVS-enabled GALS cores. As it can be seen in Fig. 17, significant savings in the clock distribution network lead to better energy efficiency. The second most important source of energy reduction is the leakage, which is linearly dependent of the voltage supply. Additional savings can be obtained in the Issue Queues and the ALUs if one of the execution clusters remains unused or the application does not offer enough parallelism (*gcc*). In the case of *mesa* (right bars in Fig. 13), the processor utilizes better both the integer and the floating-point execution clusters. As the front-end clock speed is only very infrequently scaled down, the energy savings in other modules are only marginal.

IX. Related Work

To address the problems introduced by the need of global clocking in a large, fast circuit, two approaches have been proposed in recent years. The first option is to use fully asynchronous designs. This has been tried successfully in isolated cases [3]–[5], but the design methodology for asynchronous design is far from widespread acceptance. While asynchronous systems eliminate the clock altogether, industry is not yet ready to switch to a completely asynchronous design style, mostly because design tools in this area are not as mature as those for synchronous design.

Another approach that exploits the trend toward making functional blocks more autonomous while maintaining the synchronous design methodology is the *Globally Asynchronous, Locally Synchronous* clocking style [6]. Several case studies that have been shown to benefit from such a technique were

proposed [24], [25]. All of them are based on the observation that in these specific cases the communication performance across different building blocks is not critical. This technique is mentioned in some studies as a possible solution for dealing with the global clock problems in future high-performance processors [16].

Other studies have focused on assessing the viability of a GALS clocking strategy to a superscalar, out-of-order processor. The performance and power consumption of such a processor are evaluated in [8] and [9]. While performance is worse than in the fully synchronous case by an average of 10%, the paper identifies the ability of the GALS processor to use different clock frequencies and supply voltages for each of the synchronous islands. By using a simple threshold-based algorithm that can select the best out of two possible voltage levels, the paper shows that a GALS processor can actually provide power benefits at the expense of some performance loss.

The same idea of scaling the clock frequency and the supply voltage on a per-block basis is studied in [10] and [11]. While acknowledging a drop in performance, the authors find that such processors can be more power efficient than their fully synchronous counterparts. They propose a more complicated algorithm (based on the attack–decay strategy) that can select an optimum voltage and clock frequency out of a large number of possible levels.

Another related area of research addresses mixed-clock and asynchronous interface design for robust, speed-independent communication among frequency islands. A number of mechanisms for avoiding potential race conditions are evaluated in [17]: asynchronous wrappers, stretchable clock generators, demand ports, poll ports. While these designs can ensure a proper behavior of the system, they provide worse communication characteristics than their synchronous counterparts. To address this issue, asynchronous queues were proposed in [20]. This mechanism does not improve the latency, but it increases the communication bandwidth, allowing data transfers during each clock cycle for both clock domains.

X. Conclusion

In this paper, we propose a simulation framework that allows for rapid evaluation of the different design choices available when implementing a GALS processor microarchitecture. By using this framework, we can evaluate the power and performance achieved by several GALS implementations of a superscalar, out-of-order processor. Our results show that asynchronous interfaces introduced between the several synchronous modules can have a very significant effect, varying from 5% when using pausable clocks to almost 18% when using synchronizers. However, even though pausable clocks seem to be a better choice, they do not allow communicating modules to run at different clock frequencies.

The GALS design becomes more attractive when we take into consideration process and system parameter variations. It makes sense to assume that at least some of the resulting partitions can work faster than the original, fully synchronous pipeline, and our results show that the GALS microarchitecture can take advantage of this. Even when considering the temperature variations alone, the performance drop can be significantly limited.

In terms of power consumption, the GALS design paradigm does not offer a significant benefit when dynamic voltage scaling is not implemented. The reduced clock power can be offset in some cases by the additional runtime needed to finish the same computation.

The significant advantage offered by the GALS methodology is that it allows the frequency and voltage levels to be changed independently for each module. When using DVS, an average energy reduction of up to 30% can be achieved at the expense of around 10% reduction in performance. This performance gap can be significantly narrowed by independently setting the maximum speed for each domain, without significant effects for the power consumption. In this case, our results show a reduction of 25%–30% at the expense of 5%–7% reduction in performance.

Acknowledgment

The authors would like to thank Anoop Iyer for his contribution to the initial version of the GALS simulation environment.

References

[1] N. Kurd, J. Barkatullah, R. Dizon, T. Fletcher, and P. Madland, "Multi-GHz clocking scheme for Intel Pentium 4 microprocessor," presented at the *Int. Solid-State and Circuits Conf.*, Feb. 2001.

[2] R. Hokinson, B. Benschneider, M. Ameborn, and D. Clay, "Design and migration challenges for an alpha microprocessor in a 0.18 um copper process," presented at the *Int. Solid-State and Circuits Conf.*, Feb. 2001.

[3] S. Furber, J. Garside, and D. Gilbert, "AMULET3: A high-performance self-timed ARM microprocessor," presented at the *Int. Conf. Computer Design*, Sep. 2000.

[4] K. Stevens, S. Rotem, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, and M. Roncken, "An asynchronous instruction length decoder," presented at the *5th Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999.

[5] A. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, "The design of an asynchronous MIPS R3000 microprocessor," presented at the *17th Conf. Advanced Research in VLSI*, Sep. 1997.

[6] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, D. Lindqvist, H. Tenhunen, and A. Postula, "Evaluating benefits of globally asynchronous locally synchronous VLSI architecture," presented at the *16th Norchip Conf.*, Nov. 1998.

[7] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally asynchronous locally synchronous architectures to simplify the design of on-chip systems," presented at the *12th IEEE Int. ASIC/SOC Conf.*, Sep. 1999.

[8] A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors," presented at the *Int. Symp. Computer Architecture*, May 2002.

[9] ——, "Power efficiency of multiple clock, multiple voltagecores," presented at the *IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, Nov. 2002.

[10] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," presented at the *35th Int. Symp. Microarchitecture*, Nov. 2002.

[11] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," presented at the *Symp. High Performance Computer Architecture*, Feb. 2002.

[12] P. E. Gronowski, W. J. Bowhill, and R. P. Preston, "High-performance microprocessor design," *IEEE J. Solid-State Circuits*, no. 5, pp. 676–686, May 1998.

[13] P. J. Restle, "A clock distribution network for microprocessors," *IEEE J. Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.

[14] S. Tam, S. Rusu, U. N. Desai, R. Kim, J. Zhang, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.
[15] S. McFarling, "Combining branch predictors," DEC Western Res. Lab., Palo Alto, CA, Tech. Rep. DEC WRL Tech. Note TN-36, 1993.
[16] R. Ronen, A. Mendelson, K. Lai, S.-L. Lu, F. Pollack, and J. P. Shen, "Coming challenges in microarchitecture and architecture," *Proc. IEEE*, vol. 89, no. 3, pp. 325–340, Mar. 2001.
[17] J. Muttersbach, T. Vilinger, and W. Fichtner, "Practical design of globally-asynchronous locally synchronous systems," presented at the *6th Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 2000.
[18] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," presented at the *Int. Symp. Computer Architecture*, Jun. 2000.
[19] P. Zarkesh-Ha, T. Mule', and J. D. Meindl, "Characterization and modeling of clock skew with process variations," in *Proc. IEEE Custom Integrated Circuit Conf.*, May 1999, pp. 441–444.
[20] T. Chelcea and S. Nowick, "Robust interfaces for mixed systems with application to latency-insensitive protocols," presented at the *Design Automation Conf.*, Jun. 2001.
[21] D. Burger, T. Austin, and S. Bennet, "Evaluating future microprocessors: The SimpleScalar tool set," Univ. Wisconsin, Madison, Tech. Rep. 1308, Jul. 1996.
[22] K. Chen and C. Hu, "Performance and $V_{\rm dd}$ scaling in deep submicrometer CMOS," *IEEE J. Solid-State Circuits*, vol. 33, no. 10, pp. 1586–1589, Oct. 1998.
[23] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," presented at the *Int. Symp. Low Power Electronics and Design (ISLPED)*, 1998.
[24] C. Jesshope and A. Shafarenko, "Asynchrony in parallel computing—A question of scale," presented at the *Int. Conf. Massively Parallel Computing Systems*, Apr. 1998.
[25] L. Bengtsson and B. Svensson, "A globally asynchronous, locally synchronous SIMD processor," presented at the *Int. Conf. Massively Parallel Computing Systems*, Apr. 1998.
[26] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," presented at the *30th Int. Symp. Computer Architecture*, Jun. 2003.
[27] J. A. Butts and G. S. Sohi, "A static power model for architects," in *Proc. Int. Symp. Microarchitecture*, Dec. 2000, pp. 191–201.
[28] E. Acar, A. Devgan, R. Rao, Y. Liu, H. Su, S. Nassif, and J. Burns, "Leakage and leakage sensitivity computation for combinational circuits," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 2003, pp. 96–99.
[29] R. Ginosar, "Fourteen ways to fool your synchronizer," presented at the *Int. Symp. Asynchronous Circuits and Systems*, 2003.
[30] K. Niyogi and D. Marculescu, "Speed and voltage selection for GALS systems based on voltage/frequency islands," presented at the *ACM/IEEE Asian-South Pacific Design Automation Conf. (ASPDAC)*, Shanghai, China, Jan. 2005.
[31] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," presented at the *Int. Conf. Dependable Systems and Networks*, 2002.
[32] A. Basu, S. Lin, V. Wason, A. Mehrotra, and K. Banerjee, "Simultaneous optimization of supply and threshold voltages for low-power and high-performance circuits in the leakage dominant era," presented at the *Design Automation Conf.*, Jun. 2004.

Emil Talpes received the M.S. degree in computer science from "Politehnica" University of Bucharest, Bucharest, Romania, and the Ph.D. degree from Carnegie Mellon University (CMU), Pittsburgh, PA, in 2000 and 2004, respectively.

He is currently a Postdoctoral Researcher at CMU. His research interests include computer architecture and energy-aware computing.

Diana Marculescu (S'94–M'98) received the M.S. degree in computer science from "Politehnica" University of Bucharest, Bucharest, Romania, in 1991, and the Ph.D. degree in computer engineering from University of Southern California, Los Angeles, in 1998.

She is currently an Assistant Professor of electrical and computer engineering at Carnegie Mellon University, Pittsburgh, PA. Her research interests include energy-aware computing, CAD tools for low-power systems, and emerging technologies (such as electronic textiles or ambient intelligent systems)

Dr. Marculescu is the recipient of a National Science Foundation Faculty Career Award (2000–2004) and of an ACM-SIGDA Technical Leadership Award (2003). She is an IEEE Circuits and Systems Society Distinguished Lecturer (2004–2005) and a member of the Executive Board of the ACM Special Interest Group on Design Automation (SIGDA).