ScalCore: Designing a Core for Voltage Scalability *

Bhargava Gopireddy, Choungki Song,† Josep Torrellas,
Nam Sung Kim, Aditya Agrawal,‡ and Asit Mishra♈

University of Illinois at Urbana-Champaign
†University of Wisconsin Madison
‡NVIDIA Corp.
♈Intel Corp.

gopired2,torrella,nskim@illinois.edu

ABSTRACT

Upcoming multicores need to provide increasingly stringent energy-efficient execution modes. Currently, energy efficiency is attained by lowering the voltage (V_{dd}) through DVFS. However, the effectiveness of DVFS is limited: designing cores for low V_{dd} results in energy inefficiency at nominal V_{dd}.

Our goal is to design a core for *Voltage Scalability*, i.e., one that can work in high-performance mode (HPMode) at nominal V_{dd}, and in a very energy-efficient mode (EEMode) at low V_{dd}. We call this core *ScalCore*. To operate energy-efficiently in EEMode, ScalCore introduces two ideas. First, since logic and storage structures scale differently with V_{dd}, ScalCore applies two low V_{dd}s to the pipeline: one to the logic stages (V_{logic}) and a higher one to storage-intensive stages. Secondly, ScalCore further increases the low V_{dd} of the storage-intensive stages (V_{op}), so that they are substantially faster than the logic ones. Then, it exploits the speed differential by either fusing storage-intensive pipeline stages or increasing the size of storage structures in the pipeline. Our simulations of 16 cores show that a design with ScalCores in EEMode is much more energy-efficient than one with conventional cores and aggressive DVFS: for approximately the same power, ScalCores reduce the average execution time of programs by 31%, the energy (E) consumed by 48%, and the *ED* product by 60%. In addition, dynamically switching between EEMode and HPMode based on program phases is very effective: it reduces the average execution time and *ED* product by a further 28% and 15%, respectively.

1. INTRODUCTION

Upcoming trends call for flexible processors. Users will continue to demand higher energy efficiency from computing devices in all domains, even as workloads become more dynamic. For example, sometimes all the cores in a large manycore can contribute to the application, but we can avoid dark silicon only if they all run in a most energy-efficient manner. At other times, only a few cores are active, executing a critical or serial section, and we want them to deliver high performance at any energy cost.

For homogeneous chips, some of this flexibility is currently attained with DVFS [3, 47]. A core operates at high voltage (V_{dd}) and frequency (f) when it needs to deliver high performance, and at low V_{dd} and f when it needs to run energy-efficiently. However, the effectiveness of DVFS is limited. Typically, its lowest V_{dd} is still "high" compared to the most energy-efficient regime.

It is well-known that the most energy-efficient operating point occurs at ultralow V_{dd}s [22]. However, this is a challenging environment, where circuits are affected by process variations. In particular, storage cells become failure-prone, since the V_{dd} is close to their minimum voltage for correct operation (V_{min}) [8, 40]. Intel has recently prototyped the experimental Claremont core [17, 38], which aggressively works all the way down to 0.28V.

The goal of this paper is to design a core for *Voltage Scalability*, meaning that it can flexibly work in high-performance mode (*HPMode*) at nominal V_{dd}, and in a very energy-efficient mode (*EEMode*) at low V_{dd} — a V_{dd} lower than can be attained with DVFS but not as low as Claremont's challenging levels. This is tricky because there is a fundamental design trade-off for a core: if it is designed to operate at very low V_{dd}, the resulting circuit overheads will cause it to consume higher power than needed at nominal V_{dd} — which in turn may trigger performance throttling at nominal V_{dd}. This is unacceptable, since we want our core to be competitive with the state of the art at nominal V_{dd}.

To address this problem, we make two observations. First, we note that the logic and the storage structures in a pipeline scale differently with lower V_{dd} [6]. If both share V_{dd} and f then, at low V_{dd}, the storage structures force the logic to work less energy-efficiently than it could. If, instead, storage cells are designed for low V_{min}, they have to use larger or more transistors, which consume additional power when operating at nominal V_{dd}. Hence, we propose to supply two different low V_{dd}s to the pipeline in EEMode: logic-intensive pipeline stages are powered at a lower V_{dd} (V_{logic}), while storage-intensive stages like those accessing registers and load/store queues are powered at a higher V_{dd}.

Secondly, at these low V_{dd} levels, tiny V_{dd} increases enable big f gains. Therefore, given that storage structures consume little dynamic energy, we propose to set the low V_{dd} of the storage-intensive stages in EEMode (V_{op}) to a value that is *higher* than we would need if we only took into account V_{min}. The result is storage structures that are substantially faster than logic ones. We propose to exploit this speed differential by either fusing storage-intensive pipeline stages or increasing the size of storage structures in the pipeline. Both changes increase IPC and reduce total energy. The result of our proposals is a *Voltage-Scalable* core, or *ScalCore*.

We describe the pipeline modifications to fuse stages and

* This work was supported in part by NSF under grants CCF-1012759 and CCF-1536795, and by DARPA under PERFECT Contract HR0011-12-2-0019. Kim has a financial interest in Samsung Semiconductor and AMD.

978-1-4673-9211-2/16/$31.00 ©2016 IEEE

to increase storage structure sizes. The resulting pipeline is kept relatively simple as it has a single *f*, and operates very energy-efficiently in EEMode. In HPMode, we disable the dual V_{dd}s, fused stages, and larger structures, recreating a plain out-of-order core optimized for high performance.

We use simulations of 16 cores to show that a design with ScalCores in EEMode is much more energy-efficient than one with conventional cores and aggressive DVFS: for approximately the same power, ScalCores reduce the average execution time of programs by 31%, the energy (*E*) consumed by 48%, and the *ED* product by 60%. In addition, dynamically switching between EEMode and HPMode based on program phases is very effective: it reduces the average execution time and *ED* product by a further 28% and 15%, respectively, over running in EEMode all the time.

The main contributions of this paper are: (1) the design of a voltage-scalable core based on our two observations, (2) pipeline changes to exploit the faster storage, and (3) evaluation of ScalCore and comparison to aggressive DVFS.

2. MOTIVATION

2.1 The Need for Voltage Scalability

In this paper, we consider a high-performance, power constrained large manycore of the future, e.g. targeting cloud servers and high-performance computing. In such environments, users will continue to demand increasing energy efficiency while, at the same time, requiring different execution modes. Sometimes, a program will be highly parallel. In this case, we will attain highest performance by enlisting all the cores in the manycore — as long as they run very energy-efficiently to avoid dark silicon. At other times, only a few cores will be able to run, as they execute mostly-serial sections. In this case, they will run with the highest performance, taking all the power budget of the chip.

Currently, there are two main approaches to address this conundrum: heterogeneity and DVFS. With heterogeneity, the chip contains some cores designed for energy efficiency and some for high performance. In the example above, the former would be used in the parallel section, while the latter in the mostly-serial one. The unused cores are power gated. An example of this approach is ARM's big.LITTLE [12].

While this approach is useful, it is suboptimal. First, the partition between the two types of cores in the chip is fixed, and may not be the best one for a given application. Second, there is always a fraction of the chip area that is unused — in a big-little pair, either the area of the big core or that of the little one (which can be ≈30% of the big core's area [44]). Finally, changing regimes involves migrating state, which has a performance overhead — e.g., ≈20μs [7].

DVFS [3, 47] uses cores of a single type and changes their V_{dd} and *f* values (and active core count) depending of the regime. However, this approach is also suboptimal. First, logic and storage structures scale differently with V_{dd} [6]. Hence, either at the high-V_{dd} or low-V_{dd} end, either the logic or the storage structures function suboptimaly. Note that this is not fully solved by providing one V_{dd} domain for the core and one for the caches: the core pipeline still has both logic and storage structures. The second reason for suboptimality is that the lowest V_{dd} with DVFS is still "high" compared to the most energy-efficient regime.

Such regime is at significantly-lower V_{dd}s [5, 8, 27]. In this regime, Intel has prototyped the Claremont processor, which supports V_{dd} scaling all the way to 0.28V [17, 38]. Core and caches are in two different V_{dd} domains. However, a core designed to operate at such ultralow V_{dd} needs to employ various circuit-level techniques that increase the area and, when the core operates at nominal V_{dd}, induce higher energy consumption.

This paper goes deeper into the general DVFS approach. Our goal is to design a core that scales V_{dd} to values lower than conventional DVFS — but not as low as Claremont, to minimize design complexity, area cost, and power overhead at nominal V_{dd}. Such power overhead is unacceptable because it may trigger performance throttling, and makes the core non-competitive in HPMode. Note that our goal is not to compare the heterogeneity and DVFS approaches. Comparing our design to a heterogeneous one is outside our scope.

2.2 Logic and Storage Structures in the Pipeline

A pipeline contains multiple storage structures, such as the register file, load/store queue, or ROB. In general, these structures are built with static cells, which become failure-prone as the V_{dd} goes below a value called V_{min} [8, 40]. There is a fundamental tradeoff between V_{min} and cell size: if we want a lower V_{min}, we need a cell with more transistors, with larger transistors, or FinFETs with more fins [23]. Hence, memory cells designed for lower V_{min} consume higher power and energy when operating at nominal V_{dd}.

This problem worsens for the storage structures used in the pipeline because they are heavily multi-ported. In this case, more transistors are connected to the cross-coupled inverters that form the core of the storage cell. The resulting higher loading effect on the cross-coupled inverters makes the cell more sensitive to process variations [13, 22, 49]. Consequently, we need to increase the cell size, which increases its consumption at nominal V_{dd}.

Since our goal is to keep the processor competitive at high V_{dd} operation, this is an unacceptable tradeoff. For example, Zhao et al. [49] show that going from a 1-fin 8T cell to a 2-fin 8T cell increases the leakage current by ≈20%. Moreover, our Spice simulations show that increasing the number of fins from 1 to 2 causes the 8T cell to consume 21% more power at nominal V_{dd}. This is shown in Figure 1, which plots the energy of 1-fin 8T and 2-fin 8T cells for different V_{dd}s normalized to 1-fin 8T at the nominal V_{dd} of 0.9V. The figure corresponds to 22nm and an activity factor of 1. Overall, since we want the storage cells in the pipeline storage structures to be competitive at high V_{dd}, we propose to use finFET-based cells with a single fin.

As we lower the V_{dd}, both logic and storage structures in the pipeline become slower. However, logic and storage structures scale differently [6]. This is shown in Figure 2, which we generate with Spice simulations of 22nm technology. The figure shows the increase in delay for a chain of FO4 inverters (*LogicDelay*) and for an 8T register-file bank (*SRAMDelay*) as V_{dd} decreases. The delay is the same and normalized to 1 at nominal V_{dd}. This plot includes the effect of random dopant fluctuation based on ITRS [14]. We see that storage structures become relatively slower. This is in

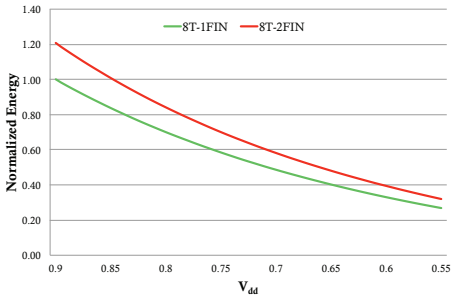


Figure 1: Effect of the number of fins in the FinFETs of the 8T cell on energy consumption for different V_{dd}s.

line with the observation made by [43].

In related work, Dreslinski et al. [9] characterized the energy consumption of cores and caches at near-threshold voltage, and found that the energy-optimal V_{dd} for caches makes them 2-4x faster than the cores. We go beyond and exploit the different behavior of logic and storage structures *within the pipeline*.

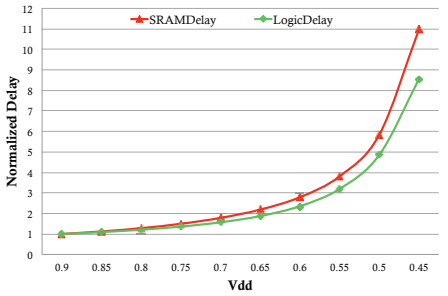


Figure 2: Increase in delay for logic and storage structures in the pipeline as we decrease V_{dd}.

3. ScalCore CONCEPT

3.1 Main Ideas

Our goal is to design a core for *Voltage Scalability*, which means that it can flexibly work in a high-performance mode (*HPMode*) at nominal V_{dd}, and in a very energy-efficient mode (*EEMode*) at low V_{dd}. The low V_{dd} is the lowest that can be sustained by the logic structures in the pipeline (but not the storage ones) before their performance becomes substantially degraded — and without requiring changes to basic circuit structures, or changes to transistor size or doping that can hurt the operation at nominal V_{dd}. To attain our goal, we rely on three ideas: (i) provide separate low V_{dd}s in the pipeline for logic and storage structures, (ii) further increase the low V_{dd} for the storage structures, and (iii) leverage the higher speed of the storage structures in the pipeline.

3.1.1 *Two Low V_{dd}s in Pipeline: Logic & Storage*

Designing storage cells to work at very low V_{dd} results in power inefficiency at nominal V_{dd}. Hence, we propose to modify the core to feed two V_{dd}s to the pipeline in EEMode: (1) logic structures are powered at a very low V_{dd} that still enables them to perform acceptably (V_{logic}); and (2) storage structures such as the register file and load/store queue are connected to a V_{dd} that is higher than V_{logic} (and at least as high as their V_{min}). With this design, the core operates with high energy-efficiency. Moreover, when we do not want to operate in EEMode, we apply a higher, equal V_{dd} level to both logic and storage structures.

To determine the V_{dd}s, we proceed as follows. We take a four-issue out-of-order core (more details in Section 7) and use McPAT's [25] high-performance process at 22nm to determine its f_{nom} at the nominal V_{dd} of 0.9V. Such f_{nom} is ≈3.5GHz. Starting from this point, we use our Spice simulations of Figure 2 to generate the V_{dd}-f scalability curves for logic and storage structures. We then adjust these curves with the effects of systematic process variations, using VARIUS [20] with the systematic variation values of EnergySmart [21]. The resulting V_{dd}-f curves are shown in Figure 3.

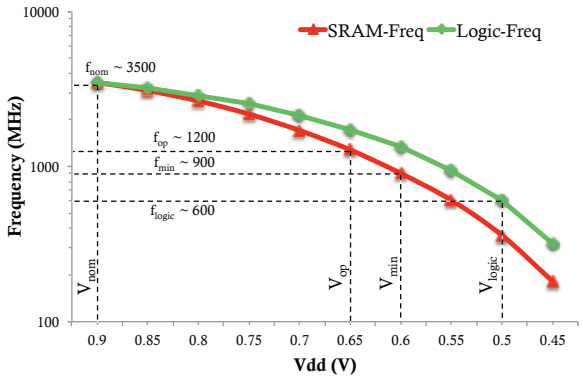


Figure 3: V_{dd}-f curves for logic and storage structures.

To pick V_{logic}, we observe that the logic delay curve in Figure 2 has a knee at V_{dd} ≈0.5V. Going below such value results in increasingly slower structures, which would cause the program to run substantially slower and consume substantially more leakage energy. This observation is consistent with the data in Kaul et al. [22]. Hence, in Figure 3, we set V_{logic}=0.5V, which corresponds to f_{logic} ≈600MHz.

To find the V_{min} of the storage structures, we argue that upcoming storage cells are likely to be aggressively designed for energy efficiency, and hence for low V_{min}. For example, Intel has attained SRAMs with V_{min}=0.6V in 22nm [18] and 14nm [19]. Hence, we set V_{min}=0.60V, which in Figure 3 corresponds to f_{min} ≈900MHz. This operating point {V_{dd}=0.60V, f=900MHz} is the *lowest point* that we assume can be reached with conventional DVFS with a single V_{dd} domain for the whole pipeline. While the V_{dd} looks aggressively low by today's standards, we think it is plausible, given the need for energy efficiency in upcoming designs.

3.1.2 *Further Increase V_{dd} for the Storage Structures*

We can improve the energy efficiency of the EEMode if we consider the following traits of the EEMode regime:

- While the storage structures need a higher V_{dd} than the logic ones for safe operation ($V_{min} > V_{logic}$), the storage structures at V_{min} can in fact operate faster than the logic structures at V_{logic} [6, 36]. This is seen in Figure 3.
- At this range of V_{dd}s, a small increase in V_{dd} provides a significant boost to the operating f.
- For storage structures at these low V_{dd}s, the dominant component of power consumption is leakage.

These observations suggest that, in EEMode, a small fur-

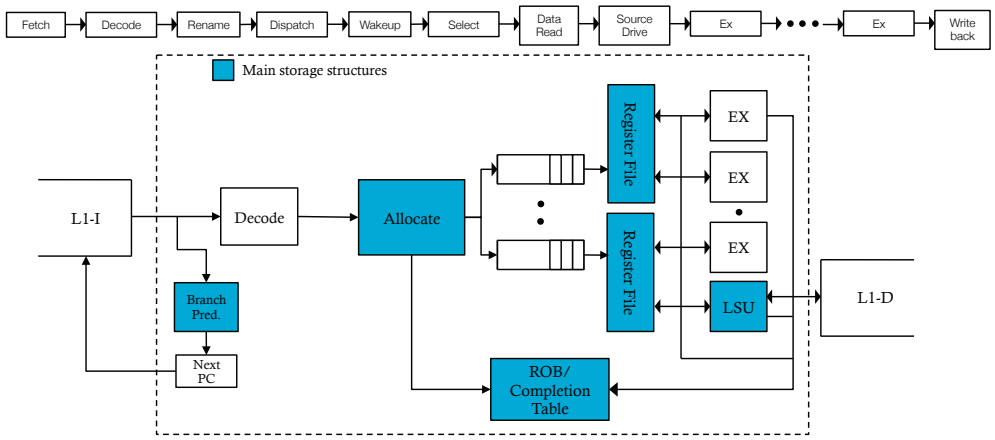


Figure 4: Pipeline of an out-of-order processor with the main storage structures.

ther increase in the V_{dd} of the storage structures can make them significantly faster, while consuming only a little more power. A higher V_{dd} increases the dynamic power comparatively more than the static power. However, since the dynamic power of storage structures in EEMode is small, the overall power will increase little.

Hence, we propose *ScalCore* as a core where, in EEMode, the V_{dd} of the storage structures is set to a voltage V_{op} that is higher than V_{min}. Specifically, we set $V_{op} > V_{min}$ such that storage structures can operate at 2x the f of the logic structures (which use V_{logic}). This is shown in Figure 3, where $V_{op} \approx$0.65V and $f_{op} \approx$1200MHz.

By setting the storage structures to V_{op} rather than V_{min}, we will improve the IPC of the cores. The resulting lower execution time of the applications in turn reduces the leakage energy — not only of the cores, but also of the caches. This reduction more than compensates the small increase in dynamic energy in the storage structures induced by going from V_{min} to V_{op}.

3.1.3 Leverage the Higher Speed of Storage

Finally, we exploit that storage structures are faster than logic ones in ScalCore's EEMode in one of two ways:

- Without changing the core's f, we fuse two consecutive pipeline stages that are dominated by storage structures into one. This reduces pipeline depth and improves the IPC.
- Without changing the core's f, we enable more entries in critical storage structures in the pipeline, consuming some of the available time slack. This increases the exploitable ILP and memory-level parallelism (MLP), and improves the IPC.

In either case, in EEMode, all the stages of the ScalCore pipeline cycle at the same f. This keeps the pipeline relatively simple. Outside EEMode, we disable the fusing of pipeline stages and the larger storage structures, and use a single V_{dd}. The core becomes a plain out-of-order core optimized for high performance. It can use conventional DVFS (with a single V_{dd} in the pipeline) to vary its operating point.

Overall, ScalCore can flexibly deliver high performance in HPMode and high energy efficiency in EEMode. However, we need to carefully select which pipeline stages to fuse and which structures to resize.

3.2 Analysis of Pipeline Stages

We analyze the pipeline stages that have storage structures to identify opportunities to improve EEMode operation. At low V_{dd}, Dreslinski et al. [9] showed that operating the L1 caches at a higher V_{dd} than the core delivers energy efficiency. This same approach was used in Claremont [17, 38]. Hence, all of our low-V_{dd} designs (including the baseline) operate the L1 caches at V_{op} by default. With ScalCore, we go beyond and use two V_{dd} domains in the pipeline.

Figure 4 shows the pipeline of an out-of-order processor and identifies the main storage structures. They are the register file, allocation structures, load/store unit (LSU), ROB, and branch predictor. We now analyze each of these structures for possible enhancements when we operate them at V_{op}. The enhancements can consist of either fusing stages or increasing the size of structures. The implementation is discussed in Section 4.

3.2.1 Register File

The critical path of a register file access in a pipelined design consists of two steps. In the first step, the operands are read from the data array into a buffer; in the second step, they are delivered to the execution units. This process takes at least two cycles in a typical high-frequency design [11]. By operating at V_{op} in EEMode, we can enhance the register file in one of two ways: either reducing its access latency in cycles or increasing its size.

In the first case, we operate the two steps of a register file access at twice the speed, and fuse them into a single cycle. In the second case, we increase the size of the physical register file without changing the two cycles of access time.

3.2.2 Allocation Structures

The allocation step primarily involves register renaming and instruction dispatch. A detailed analysis of the critical path in renaming is performed by Palacharla et al. [33]. The authors found that the critical path in rename consists of reading the current mappings of the source registers in the Register Alias Table (RAT), followed by updating the mappings of the destination registers in the RAT. The dependence check among the registers currently being renamed is not in the critical path, and is implemented using low-power transistors, which reduces the dynamic power. Also, the design of the RAT is similar to the register file. Hence, we operate the rename unit at V_{op} in EEMode.

The rename delay in an out-of-order core is proportional

to the core's width. In a very wide core, the rename operation may take more than one cycle. However, for a core width of four like ours, renaming in a baseline design can be completed in a single cycle. Hence, in ScalCore, we can combine the rename and dispatch stages and fuse them into one cycle operating at V_{op}. Since the dynamic power consumption of the dispatch unit is low [39], the increase in V_{dd} to V_{op} has only a small effect. We do not change the size of the rename unit because we find that it does not have a critical resource whose size can be increased to improve performance.

3.2.3 Load/Store Unit (LSU)

At a high level, the LSU consists of two stages, each taking one cycle. The first stage performs address generation, and the second one memory disambiguation. Also, in parallel to the disambiguation, store instructions write values to the store buffer. The load/store queue in the LSU contains in-flight memory instructions and maintains the ordering between instructions.

Since the LSU takes two cycles and also has a resource (load/store queue) whose size can be increased, as we operate the LSU at V_{op}, we can do one of two things: either we fuse the two stages to reduce the latency to one cycle, or we increase the load/store queue size. When we fuse the two stages, stores write to the store buffer in half of a cycle. When we increase the load/store queue size, we also increase the store buffer size. Note that increasing the size of the load-/store queue requires extra care, since it uses CAM structures to perform memory disambiguation (Section 4.1.2).

3.2.4 ROB

We consider a merged register file-based renaming scheme as described in [11]. The ROB only acts as a completion table, keeping track of the in-flight instructions. The entries in the ROB are reserved on instruction dispatch and are freed on commit. The commit process in HPMode takes only one cycle. Hence, there is no obvious opportunity to reduce the latency in EEMode. However, we modify the size of the ROB in EEmode to enable more in-flight instructions. This allows us to exploit more ILP or MLP, based on the application.

The dynamic power of the ROB itself is low [39]. Hence the increase of the V_{dd} to V_{op} causes a modest increase in dynamic energy.

3.2.5 Branch Predictor

Branch prediction in modern processors consists of a fast BTB to provide prediction in a single cycle, which is backed-up with a more complex and accurate branch predictor with a longer latency. In our baseline design, we have a tournament predictor with a total size of 48K entries and a BTB of 2K entries. Although the BTB is structurally similar to a register file or cache, there is no obvious opportunity to further reduce its latency. While it is possible to take advantage of the faster operation at V_{op} by using a fancier branch prediction, it is out of this paper's scope. Also, by simply increasing structure sizes, the expected improvement in IPC is small, considering the high baseline size [2, 10]. Hence, we do not optimize the branch predictor.

3.3 Summary

Based on the above analysis, Table 1 lists the pipeline structures that can be enhanced by ScalCore in EEMode, and the enhancement options. Note that a given structure cannot both run faster and be bigger at the same time. Outside EEMode, all structures are operated at the same V_{dd}, and at their baseline speed and size.

Structure	Faster	Bigger
Register File	✓	✓
Allocation Struc.	✓	✗
Load Store Unit	✓	✓
ROB	✗	✓
Branch Pred.	✗	✗

Table 1: Enhancements considered for different structures.

4. ScalCore DESIGN

To support the enhancements in Table 1, ScalCore requires some changes over conventional processors. We classify them into microarchitectural changes in the datapath and control-path, and circuit changes. We consider each in turn.

4.1 Datapath Microarchitectural Changes

4.1.1 Reduction in Latency

Fusing two consecutive pipeline stages in EEMode is accomplished by transforming the latch that sits in between them into a transparent (or flow-through) latch [16, 42]. This is done by ORing the clock to the latch with an Enable Flow-through signal. When the signal is set to logic one, the latch is transparent.

The logic design is shown in Figure 5a. In HPMode (upper chart), Stages 2a and 2b take one cycle each, and operate at nominal voltage (V_{nom}) like the other stages. The Enable Flow-through bit in the OR gate is set to logic zero. This causes the latch to be controlled by the clock.

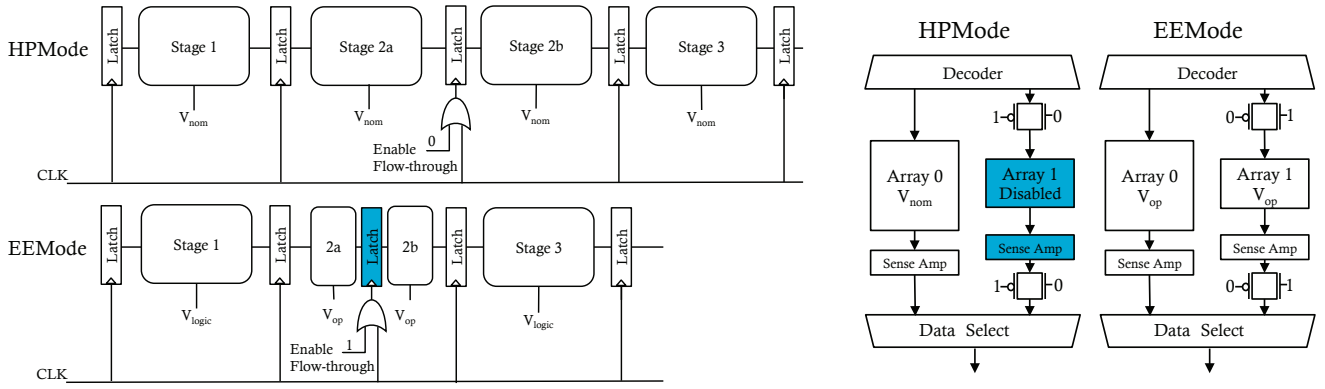
In EEMode (lower chart), Stages 2a and 2b are fused together to execute in a single cycle. The Enable Flow-through bit is set to logic one, which makes the latch transparent. In addition, Stages 2a and 2b operate at V_{op}, which is higher than the V_{logic} used in Stages 1 and 3.

This OR gate is added to the latches connecting the pairs of stages to be fused: (i) the two stages of a register file access, (ii) rename and dispatch, and (iii) the two stages in the LSU. Designs based on flip-flops can be modified in a similar manner, by providing a bypass path that is enabled in EEMode.

4.1.2 Bigger Structures

The size of a storage structure in ScalCore is increased by enabling an additional array with transmission gates. This general approach was proposed by Buyuktosunoglu et al. [4] for issue queues. By toggling the input to the transmission gates, we can easily enable or disable the new array. Figure 5b shows an example of a structure with an original array (Array 0), and an additional one (Array 1) connected with transmission gates. In HPMode (left chart), the transmission gates disable Array 1, while Array 0 operates at V_{nom}. In EEMode (right chart), the transmission gates enable Array 1. Both arrays are active and run at V_{op}.

With this design, the register file, load/store queue, store buffer, and ROB can use more entries in EEMode than in HP-Mode. CACTI analysis [30] shows that, at the higher V_{op}, we



(a) Fusing pipeline stages using a flow-through latch.

(b) Increasing the size with transmission gates.

Figure 5: Datapath changes in ScalCore. The shaded components are disabled.

could increase the structure sizes substantially and still meet the cycle time. However, very large structures incur area and leakage overheads, and are hard to use cost-effectively. As we will see, a possible design is to increase the size of these structures by 50% in EEMode.

We make special arrangements for the load/store queue, since it uses CAM structures to perform memory disambiguation. Specifically, to reduce complexity, ScalCore uses a segmented load/store queue [34] with two segments. In HPMode, only one segment is active and the other is power gated. Hence, the delay and power in HPMode are not impacted. In EEMode, both segments are active and are sequentially searched on a request. Since the segments operate at V_{op}, both segments are searched in a single cycle.

In EEMode, the dynamic power of the load/store queue increases only for the searches that overflow into the second segment. Also, techniques like low-swing, selective precharge of search line/matchline reduce the dynamic power of CAM by more than 50%, with no delay impact over a conventional design [32]. By using such techniques, the dynamic power component becomes small even in the baseline. Hence, the increase in load/store queue size causes only a small increase in the dynamic energy.

4.2 Controlpath Microarchitectural Changes

4.2.1 Reduction in Latency

Reducing the latency of some operations has an impact on the scheduling of various tasks in the pipeline. So, we need to modify the control logic responsible for those tasks. Specifically, in EEMode, by reducing the register file read latency (which includes the source drive of operands) from two to one cycle, the execution units receive operands in one cycle, and hence can begin execution one cycle earlier. In addition, as the execution finishes a cycle earlier, the dependent instructions can be woken up and scheduled earlier. Hence, the execution schedule time and the generation of the wakeup signal need to be updated based on the mode of operation. Similarly, to enable the speculative issue of load-dependent instructions, the issue logic should be updated with the new latency of the LSU.

Reducing the latency of the allocation operation does not have an obvious direct impact on the scheduling of tasks. The process of renaming and dispatching instructions in order can proceed in the same manner in both modes.

Therefore, the only controlpath modifications required are to ensure that the part of the issue-queue state machine responsible for generating ready signals accurately reflects the mode of operation. The issue queue already contains functionality to generate the wakeup signal at different times based on the latency of the functional unit (e.g., ADD vs. DIV) [11]. Hence, by modifying the counters used for this process, we ensure correct scheduling in both HPMode and EEMode.

4.2.2 Bigger Structures

The availability of a bigger resource should be conveyed to its corresponding resource manager to enable its usage. In EEMode, the sizes of the register file, load/store queue, store buffer, and ROB are higher. To benefit from the bigger physical register file, the list of free registers in the rename unit must be updated. The list of free registers is typically maintained as a circular buffer with head and tail pointers, which can be resized based on the mode of operation.

The dispatch unit is responsible for checking various execution resources and stall in case of unavailability. It maintains counters for the availability of each resource. To indicate the sizes of the current ROB, load/store queue, and store buffer, we just need to update these counters based on the mode of operation. Overall, therefore, the changes required in the controlpath for bigger structures involve reconfiguring the counters in the rename and dispatch stages.

Table 2 summarizes the microarchitectural changes.

Component	Reduced Latency	Bigger Size
Datapath	Gated clocks for transparent latches	Transmission gates to resize structures
Controlpath	Programmable counters for latency	Programmable counters for size

Table 2: Microarchitectural changes in ScalCore.

4.3 Circuit Changes

The circuit changes involve supporting dual voltage rails and level converters.

4.3.1 Dual Voltage Rails

In ScalCore, each pipeline stage is connected to one of two V_{dd} rails, as shown in Figure 6. All the storage-intensive stages are connected to one rail. They are the two stages of register file access, the rename stage, the dispatch stage,

the two stages of LSU access, and the commit stage. The other stages are connected to the other rail. Each rail is set to the required V_{dd} level, based on the mode of operation. In EEMode, the storage-intensive stages receive V_{op}, while the rest receive V_{logic}; outside EEMode, both rails supply the same voltage.

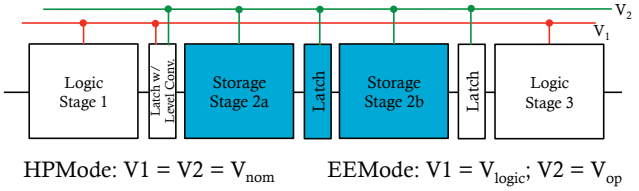


Figure 6: ScalCore pipeline with dual voltage rails.

These storage-intensive stages share the V_{dd} domain with the L1 caches which, as indicated in Section 3.2, also operate at V_{op} in EEMode. As a result, there is no need to add any additional voltage regulator in ScalCore over the baseline design. Note that the L1 caches are laid out together with the pipeline in current designs.

4.3.2 *Level Converters*

All the stages in ScalCore always operate at a common f. This simplifies the design by avoiding multiple clock trees and synchronization overhead across stages. However, consecutive pipeline stages may operate at different V_{dd} levels in EEMode. Hence, a level converter is required on the boundary between a stage at V_{logic} and one at V_{op}, to provide full swing input to the higher V_{op} domain. Note that the difference in V_{dd} levels is only 150mV.

Level converters can be designed as part of the latches or flip-flops that separate the stages (Figure 6). The design of a level converter is shown in Figure 7. It is based on [15], where pulsed half-latch level converting flip-flops are shown to be efficient, both in area and in energy-delay, compared to asynchronous level conversion in a dual-V_{dd} system.

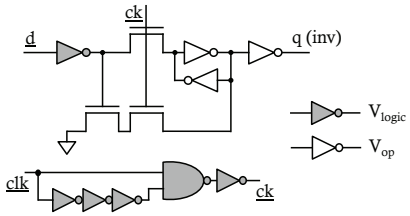


Figure 7: Level converter for up-conversion.

5. IMPLEMENTATION CONSIDERATIONS

This section considers three implementation issues: the transition between modes, a summary of ScalCore overheads, and a comparison to Intel Claremont.

5.1 Transition Between Modes

In highly-parallel sections, all the cores in a large many-core are powered-on and run in EEMode. In sections with little parallelism, only a few cores are powered-on and run in HPMode, using all the power budget of the chip. While in HPMode, ScalCore can also use DVFS.

ScalCore provides a simple way to reconfigure the pipeline between the two modes. First, pipeline reconfiguration can only occur when the pipeline is empty. Consequently, reconfiguration requires a pipeline flush. Such a flush can take a variable number of cycles — e.g., depending on whether there are pending writes in the store buffer (pending reads are squashed). However, the average flush is not likely to take more than several tens of cycles. Second, we need to change the f and V_{dd} of the core. Specifically, from a common V_{dd} in all stages in HPMode, we transition to V_{op} for the storage-intensive stages and V_{logic} for the others in EEMode (or vice-versa). This transition uses conventional DVFS mechanisms. Its overhead is likely to be modest in the future, as increasing DVFS speed is an active area of research and development. In our evaluation, we set this overhead of changing f and V_{dd} between modes to 1 μs. Note that pipeline stages *do not switch* V_{dd} rails; ScalCore merely changes the rails' voltage.

Overall, the overhead of transitioning from EEMode to HPMode and vice-versa is small. The reconfiguration can be triggered either in hardware by the power management unit, or in software by the operating system or program.

5.2 Summary of ScalCore Overheads

Table 3 summarizes the ScalCore overheads and their impact on HPMode. The first issue is dual V_{dd} rails. Their main overheads are the additional area they take and the need to customize their layout/routing, since automatic tools may not be able to handle them. One implementation of dual rails [31] estimates the area cost to be ≈5% of the core.

Issue	Type of Overhead	Impact on HPMode
Dual V_{dd} rails	1) Custom layout and routing. 2) Area increase	≈5% area [31]
Level converters	1) Carefully manage clock skew/timing. 2) Add gates in critical path	≈5% delay [15]
Fusing Pipeline Stages	1) OR gate added to the clock signal for each latch connecting fused stages. 2) Control logic for counters for latency	Tiny area and power
Increasing Array Sizes	1) Additional array. 2) Transmission gates to enable/disable additional array. 3) Segmented ld/st queue. 4) Control logic for counters for size	Area of power-gated array. Tiny power and delay [4]

Table 3: Summary of ScalCore overheads.

The second issue is level converters. They require carefully managing the clock skew and timing across domains. Moreover, they add a few gates to the pipeline stage. Based on Ishihara et al.'s [15] work, we estimate a delay impact in HPMode of ≈5%. The additional area and power of the level converters is negligible [15].

The third issue is fusing pipeline stages. It requires an OR gate added to the clock signal for each latch connecting fused stages, and control logic for counters for latency (Section 4.2.1). In HPMode, it adds a tiny area and power overhead but no delay in the critical path.

A fourth issue is increasing array sizes. It requires an additional array, transmission gates to enable/disable the additional array, a segmented load/store queue, and control logic for counters for size (Section 4.2.2). In HPMode, the additional array is power gated, but takes up area. The additional logic introduces only tiny power and delay overheads. Buyuktosunoglu et al. [4] show that the transmission gate delays are negligible. Their design is more involved than

ours in that they get multiple, dynamically variable latencies, which require careful synchronization with the rest of the pipeline. In our case, we only have two configurations, and the large one has ample timing slack. Similarly, Park et al.'s [34] segmented load/store queue is more involved because disambiguation can take a variable number of cycles, while ours always takes one cycle.

Finally, ScalCore introduces design complexity and verification costs, which are hard to quantify.

5.3 Comparison to Intel Claremont

Table 4 compares Intel's Claremont prototype [17, 38] to ScalCore. A main difference is that Claremont targets a very wide V_{dd} operating range (1.2V to 280mV) while ScalCore targets a more modest range (0.9V to 0.5V). As a result, Claremont uses more aggressive techniques, including manually prioritizing the placement of logic paths that have a high percentage of interconnect, or using two level-converters in series to bring the voltage to high V_{dd}.

Trait	Claremont	ScalCore
V_{dd} range	Very wide: 1.2V to 280mV	Wide: 0.9V to 0.5V
Focus	Circuit techniques: 1) Variation aware pruning & beefing-up of cells. 2) Circuits optimized for reliability at ultralow V_{dd}	Architectural techniques: 1) Separate V_{dd} for memory intensive pipe stages. 2) Fuse pipeline stages. 3) Increase array sizes
V_{dd} domains	1 for the pipeline + 1 for the L1	1 for the memory intensive pipe stages and L1 + 1 for the other pipe stages

Table 4: Comparing Intel Claremont to ScalCore.

Another difference is that Claremont's focus in on circuit techniques, while ScalCore's is on architectural techniques. Claremont includes variation-aware selective pruning and beefing-up of the standard cell library, and circuits optimized for reliability at ultralow V_{dd} — e.g., avoiding wide transmission-gate multiplexers and circuits with high transistor stacks. The result of needing to reach an ultralow V_{dd} is device area bloat.

ScalCore focuses on architectural techniques such as a separate V_{dd} for memory-intensive pipeline stages, fusing pipeline stages, and increasing array sizes.

Although the number of V_{dd} domains in both designs is the same, they are organized differently. Claremont has one domain for the pipeline and one for the L1; ScalCore has one for the memory-intensive pipeline stages plus the L1, and one for the rest of the pipeline stages.

6. IMPLICATIONS ON APPLICATIONS

6.1 Pipeline Stage Fusion

In EEMode, ScalCore can perform register access, rename-dispatch, and LSU handling in one cycle each, rather than in two cycles each. In total, the pipeline depth reduces by three cycles for load/store instructions and by two cycles for others. This reduction has a few implications, as noted by Tullsen et al. [46]. First, it reduces the branch misprediction penalty by 2 cycles. Second, it results in fewer instructions from the mispredicted path in the pipeline, which saves energy and resources. Third, registers are now held for a shorter duration, reducing the contention for physical registers during renaming. Finally, reducing the LSU latency also helps the optimistic issue of load-dependent instructions that assume a cache hit.

Overall, these effects result in an increase in the average IPC for a broad range of application types — especially for integer applications, which have more branches.

6.2 Bigger Structures

Increasing the sizes of the register file, load/store queue (LSQ), store buffer, and ROB enables an out-of-order core to extract more ILP/MLP and, therefore, boost performance. Traditionally, increasing LSQ size to exploit higher MLP results in additional pipeline stages, hurting the IPC. In ScalCore, by operating the LSQ at V_{op}, we are able to increase the size and still keep the number of stages constant.

These changes improve the IPC for most applications, especially those that stress the current structures — e.g., memory intensive codes constrained by LSQ size, or FP applications with sizable ILP.

7. EVALUATION SETUP

To evaluate ScalCore, we use the SESC architectural simulator [37], modeling up to 16 4-issue out-of-order cores. We use McPAT for detailed energy calculation [25]. For a more accurate evaluation of dynamic and leakage energies, we model the L1 and L2 caches with CACTI [30]. Table 5 shows the parameters of the simulated architecture.

Parameter	Value
Architecture	Up to 16 4-issue out-of-order cores at 22nm
Register file; ROB	128 regs; 128 entries
Issue queue	48 entries
Ld queue; St queue	48 entries; 32 entries
Branch prediction	Tournament predictor: bimodal (16K entry), gshare (16K entry) and selector (16K); 32-entry RAS; 4-way 2K-entry BTB
Functional units:	
2 integer ALU	4-cycle Mult/Div, 1-cycle for rest
2 LSU	2 cycles
2 FPU	1-cycle Add, 4-cycle Mult, 12-cycle Div
Private I-Cache	64KB; 2way; 64B line; Round-trip (RT): 2cycles (HPMode) or 1cycle (EEMode or any low-V_{dd} design)
Private D-Cache	64KB; 4way; writeback (WB); 64B line; RT: 2cycles (HPMode) or 1cycle (EEMode or any low-V_{dd} design)
Shared L2	Per core: 1MB; 8way; WB; 64B line; RT to local bank: 8cycles (HPMode) or 6cycles (EEmode or low-V_{dd} design)
DRAM latency	RT: 50ns
Network	2D mesh with MESI directory-based protocol
EEMode-HPMode change overhead	Pipeline flush + 1μs (for V_{dd} and f change)

Table 5: Parameters of the simulated architecture.

Table 6 shows the f and V_{dd} for HPMode and EEMode. They were justified in Section 3.1. We have penalized the f of our HPMode by 5%, due to the delay overhead of ScalCore in Table 3. Note that the cache latencies in cycles in Table 5 are higher in HPMode than in EEMode (and all of the low-V_{dd} designs that we will analyze). This is because the core's f is different. Finally, in EEMode and all of the low-V_{dd} designs, caches operate at the higher V_{op}=0.65V to improve efficiency, as suggested in [9].

HPMode	V_{dd}=V_{nom}=0.9V; f=3.3GHz /* 5% penalty as per Table 3 */
EEMode	V_{logic}=0.50V for logic and V_{op}=0.65V for storage; f=0.6GHz

Table 6: HPMode and EEMode design points.

7.1 Design Configurations

We evaluate several configurations operating at low V_{dd}, as shown in Table 7. First, *DVFS+* is the most energy-efficient voltage-frequency setting that we assume an aggressive DVFS can reach. It operates the whole pipeline at V_{dd}=V_{min}=0.6V, and f=0.9GHz. In addition, the caches operate at the more energy efficient V_{op}=0.65V (like all the other designs in the table) — hence, the suffix "+". This is the V_{dd} domain arrangement of Dreslinski et al. [9] and Claremont [17, 38], although the actual V_{dd} levels are different.

Config.	Parameters
DVFS+	Whole pipeline at V_{dd}=V_{min}=0.60V; f=0.9GHz. /* Uses the V_{dd} domain arrangement of Dreslinski et al. [9] and Claremont [17, 38] */
Pipe2Vdd	Pipe logic at V_{logic}=0.50V and pipe storage at V_{dd}=V_{min}=0.60V; f=0.6GHz. /* Storage not fast enough to exploit the speed difference */
SC:	Pipe logic at V_{logic}=0.50V and pipe storage at V_{dd}= V_{op}=0.65V; f=0.6GHz. /*ScalCore EEMode variations*/
SCsp1	Reg = 1 cycle latency
SCsp2	Reg, alloc, LSU = 1 cycle latency
SCsz	Reg, LSQ, store buff, ROB = 1.5x size
SCmx	Reg, alloc = 1 cycle lat; LSQ, store buff, ROB = 1.5x size

Table 7: Configurations explored in low V_{dd}.

Pipe2Vdd adds a dual V_{dd} domain in the pipeline: it sets V_{logic}=0.50V for the logic stages, and V_{dd}=V_{min}=0.60V for the storage stages. However, storage is not fast enough to exploit the speed difference with logic. f is 0.6GHz.

Next, *SC* creates ScalCore by keeping V_{logic}=0.50V for the logic stages but increasing V_{dd}=V_{op} =0.65V for the storage stages. f remains at 0.6GHz. We have four variations with different hardware support. First, *SCsp1* and *SCsp2* (for speed) reduce latencies by fusing pipeline stages. Specifically, SCsp1 fuses the two stages in the register file access into one. SCsp2 augments SCsp1 by also fusing the two stages in the allocation and the two in the LSU into one each.

SCsz (for size) keeps the pipeline unchanged but increases the sizes of the register file, LSQ, store buffer, and ROB. We have evaluated different amounts of size increases. Given our limited space, we present data for only one size, which delivers one of the best tradeoffs between energy and performance. The design is for 1.5x structure sizes. It can be shown that other sizes are less cost-effective or only marginally more cost-effective.

SCmx (for mixed) combines fusing stages and increasing structure sizes. Specifically, it fuses the two stages in the register file access into one, and the two in the allocation into one. It sets the sizes of the LSQ, store buffer, and ROB to 1.5x their original sizes. Note we can only increase either the speed or the size of a given unit at a time.

All of these low-V_{dd} designs have the same low cache latencies in cycles as ScalCore's EEMode. The values were shown in Table 5.

At V_{nom}, we evaluate *HPMode* (our ScalCore) and *HPRef* (state-of-the art, like *HPMode* but without the 5% f penalty).

7.2 Applications & Metrics

We evaluate the architectures with a variety of parallel applications. From SPLASH-2, we use Barnes (16K particles), Cholesky(tk29.O), FFT(2^{20}), FMM(16K), LU(512x512), Radiosity(batch), and Radix(2M keys). From PARSEC, we use Blackscholes (sim medium), Fluidanimate (sim small), and Streamcluster (sim small). From NAS, we use BT, LU, and SP (all W size). We run each application to completion.

Our metrics are execution time, energy consumption (E), and energy-delay product (ED). In our evaluation, we start by comparing the different ScalCore EEMode configurations to *HPRef*, *DVFS+*, and *Pipe2Vdd*. We do it in an environment with a fixed power, and one with no power constraints. In both cases, we run the SPLASH-2 and PARSEC codes and do not change configurations dynamically. Then, we consider the environment with the fixed power again, and allow changing the configurations dynamically. In this case, we run the NAS codes. In each NAS code, we prevent the parallelization of some parts so that the serial section takes ≈30% of the total execution time.

8. EVALUATION

8.1 Environment with a Fixed Power

We compare the different ScalCore EEMode configurations, *HPRef*, *DVFS+*, and *Pipe2Vdd* for a fixed amount of power. To do this, we note that the power consumption of a single *HPRef* core and its caches is ≈12W. Then, each of the other configurations can have as many cores as needed to get to ≈12W. Figure 8 shows, for each configuration, the power consumed and the number of cores that can execute. From the figure we see that, for our applications, for the power of one *HPRef* core (*1-HPRef*), we can run one *HPMode* core (*1-HPMode*), or eight *DVFS+* cores (*8-DVFS+*), or 16 *Pipe2Vdd* cores (*16-Pipe2Vdd*), or 16 cores under the various ScalCore EEMode configurations (*16-SC**).

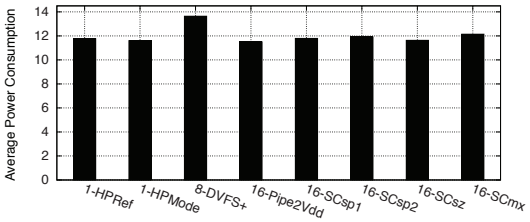


Figure 8: Configurations for a fixed power.

We now take these designs and compare their execution time, total E, and ED product running the applications. Figure 9 shows, for each design, the execution time of each application and the average. In each application, the bars are normalized to *16-Pipe2Vdd*.

We consider first the different ScalCore designs. We see that all of its variants reduce the execution time over *16-Pipe2Vdd*. The gains come from the increase in the V_{dd} of the storage structures in the pipeline, and the resulting pipeline reconfiguration. Most of the applications show higher gains due to a reduction in the latency (*16-SCsp1* and *16-SCsp2*) than due to an increase in the size of the structures (*16-SCsz*). This is because applications typically exhibit sensitivity to branch misprediction penalty and load-to-use delay, both of which are reduced in the *16-SCsp** designs. However, for these applications, the additional changes going from *16-SCsp1* to *16-SCsp2* have minimal impact, as they are hidden by other effects such as pipelining and inter-thread synchronization. In addition, applications that exhibit higher levels

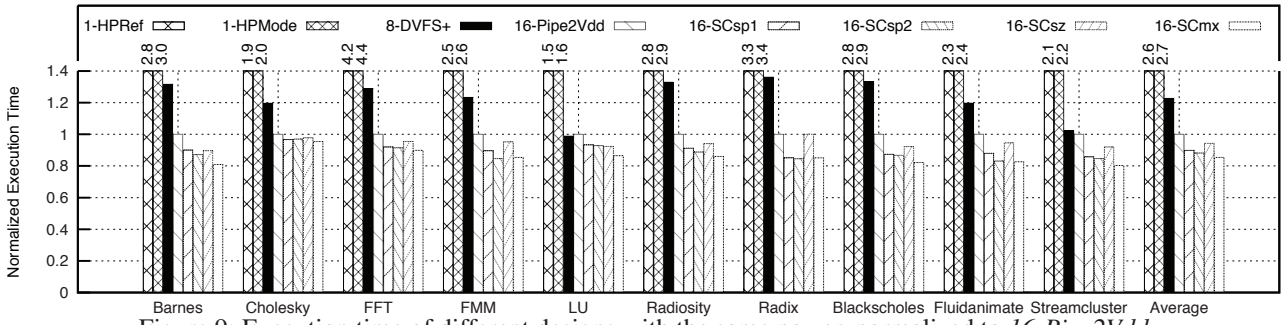


Figure 9: Execution time of different designs with the same power, normalized to *16-Pipe2Vdd*.

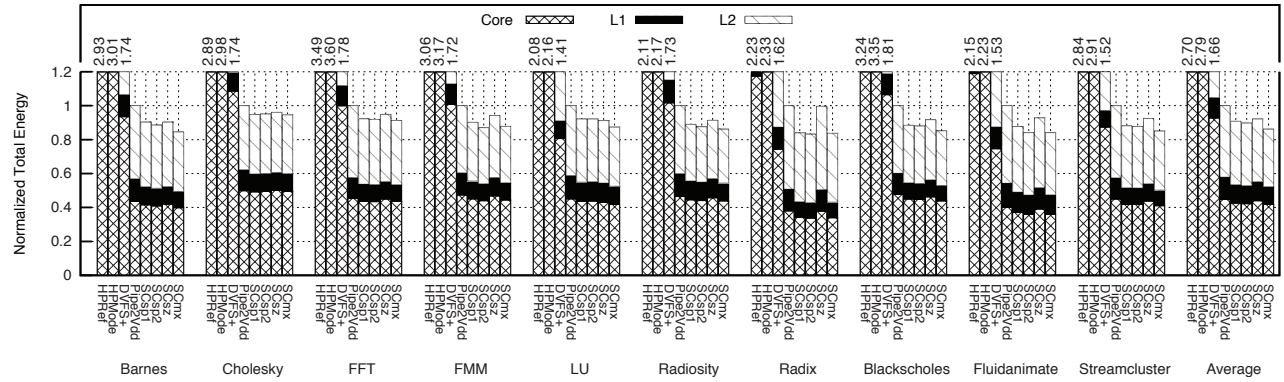


Figure 10: Energy consumption of different designs with the same power, normalized to *16-Pipe2Vdd*.

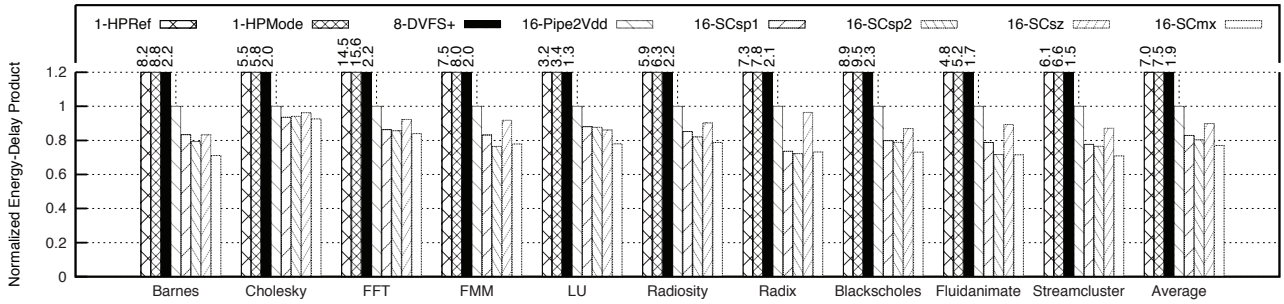


Figure 11: Energy-delay product of different designs with the same power normalized to *16-Pipe2Vdd*.

of MLP/ILP can also take advantage of bigger structures (*16-SCsz*). Overall, *16-SCmx* performs the best, as it provides a latency reduction of 2 cycles for all instructions, and also bigger structures for applications that can benefit from them.

Looking at the aggressive DVFS (*8-DVFS+*), we see that, despite having a frequency advantage, it performs much worse than *16-SCmx*. This is due to its lower number of cores. The high performance designs (*1-HPRef* and *1-HPMode*) perform even worse, and they are practically identical.

Overall, for our programs, which try to represent the load in large manycores, *16-SCmx* reduces the execution time by an average of 31% relative to *8-DVFS+*, and by 15% relative to *16-Pipe2Vdd*. These are substantial reductions.

Figure 10 shows the energy consumed by all the designs. The figure is organized as the previous one, except that the bars are broken down into the contributions of core, L1, and L2. Since this experiment is done at approximately constant power, the bars in Figure 10 roughly follow those of Figure 9. From the figure, we also see that, going from *Pipe2Vdd* to *SC**, there are good savings in L2 energy. This is mostly leakage eliminated by finishing the application earlier — even though the storage structures in the pipeline use a higher V_{dd} in *SC**.

Overall, the best design (*16-SCmx*) reduces the energy consumption by an average of 48% relative to *8-DVFS+*, and by 13% relative to *16-Pipe2Vdd*.

Finally, we consider the ED product. Figure 11 shows the ED product for all the designs. We see that most of the ScalCore EEMode designs provide substantial ED product reductions. *16-SCmx* reduces the ED product by an average of 60% relative to *8-DVFS+*. We can see that not even this aggressive DVFS level can get close to the efficiencies delivered by ScalCore. Moreover, *16-SCmx* reduces the ED product by an average of 23% relative to *16-Pipe2Vdd*. Note that *16-Pipe2Vdd* is already very energy efficient, with its separation of voltage rails for pipeline logic and storage structures. These results provide a strong motivation for ScalCore.

8.2 Environment without Power Constraints

We now repeat the previous experiments without imposing any power constraint. All the configurations run with 16 cores and, as a result, the power consumption is very different. Figure 12 shows the execution time organized like in Figure 9. The names of the designs do not include the core

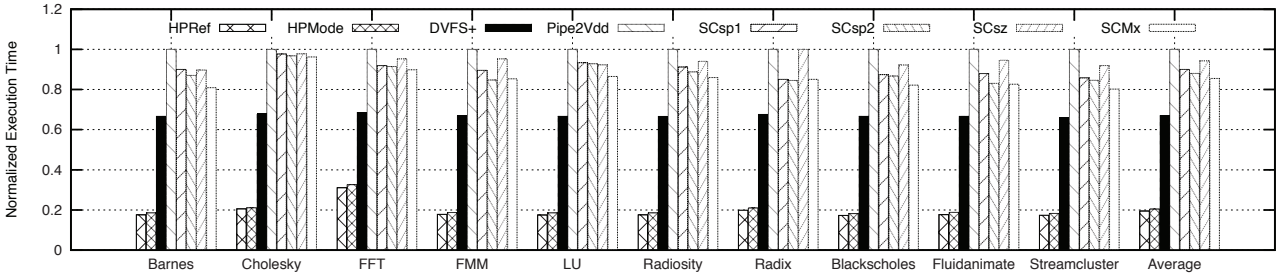


Figure 12: Execution time of different designs with 16 cores normalized to *Pipe2Vdd*.

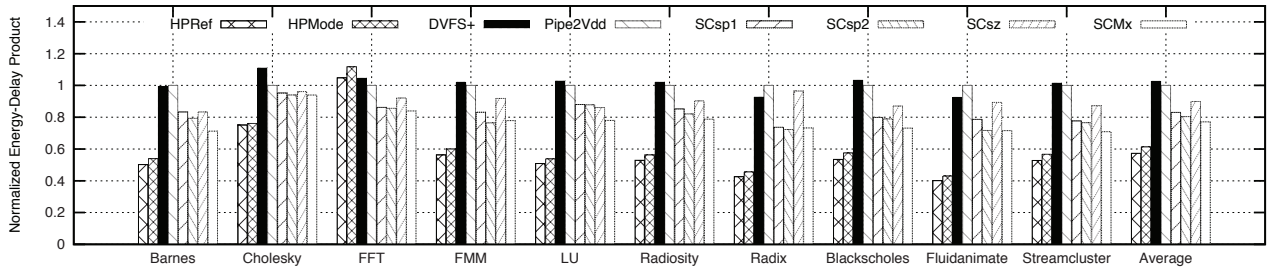


Figure 13: Energy-delay product of different designs with 16 cores normalized to *Pipe2Vdd*.

count anymore. Note that, in the figure, *DVFS+* and, especially, *HPRef* and *HPMode* are faster than *SC**. Also, Figure 13 shows the resulting ED products of the different designs. Thanks to the faster execution of *HPRef* and *HPMode*, their ED is now substantially lower than *SCmx*. However, the ED of *DVFS+* is still higher than *SCmx*.

Unfortunately, these execution time improvements of *HPRef*, *HPMode*, and *DVFS+* come at a *staggering power cost*. Indeed, extrapolating from Figure 8, we can see that each *HPRef* and *HPMode* core consumes ≈16x the power of a *SCmx* core. In addition, a *DVFS+* core consumes over 2x the power of a *SCmx* core. Consequently, in practice, this much power may not be available in the chip, and some of the cores under *HPRef*, *HPMode*, and *DVFS+* may be powered down, becoming dark silicon. In fact, *HPRef* and *HPMode* cores are most effective when executing mostly-serial codes, where a few cores can use all the chip power budget. This motivates the next section.

8.3 Dynamically Changing Configurations

To deliver the most energy-efficient execution, ScalCore can dynamically reconfigure. Hence, in this section, we impose the same power limit as in Section 8.1, but allow ScalCore to dynamically reconfigure between the *SCmx* EEMode and HPMode. Recall that HPMode is penalized with a 5% *f* reduction relative to *HPRef*. At any point, any unused cores are power gated. We call the design *Dyn-SC*.

We run the NAS applications, where an application executes a series of parallel loops and serial sections. We instrument the applications, so that ScalCore transitions to 16 *SCmx* cores before entering a loop, and transitions to one *HPMode* core after finishing the loop. This setup models support for program hints that directly invoke the power management unit. The analysis of more advanced interfaces to trigger the reconfiguration is beyond our scope.

We compare *Dyn-SC* to the best reconfigurable design that uses conventional processors. Such design dynamically switches between one *HPRef* core outside loops (a very high performance core without any *f* penalty) and 8 *DVFS+* cores in the loops (a very aggressive DVFS that sets V_{dd} to 0.6V). We call this design *Dyn-HPRef/8-DVFS+*. Such design is free of any ScalCore modifications.

As a reference, we also compare to three other designs. One is a single HPRef core all the time (*1-HPRef*). A second one is 16 SCmx cores all the time (*16-SCmx*). The third one is a reconfigurable environment that transitions between one HPMode core outside loops and 16 Pipe2Vdd cores in the loops (*Dyn-HPMode/16-Pipe2Vdd*). This is a design that includes some of the ideas of ScalCore, but not all. Specifically, it includes two V_{dd} domains in the pipeline and, hence, level converters. However, it does not include ScalCore's pipeline stage fusing or storage structure resizing.

Figure 14 shows the execution time, energy consumed, and ED product for the five designs. For each metric, the bars correspond to the average of the three NAS applications and are normalized to *16-SCmx*.

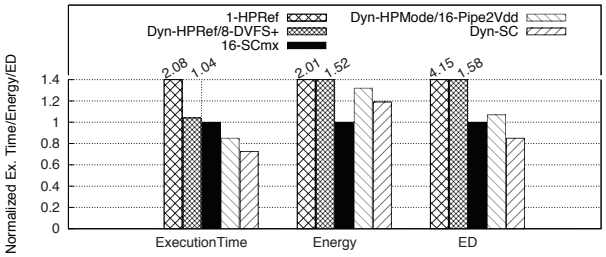


Figure 14: Dynamically reconfiguring ScalCore.

Dyn-SC provides the lowest execution time and the lowest ED product. It does not deliver the lowest energy because it runs a high-performance core during the serial sections. Instead, *16-SCmx* consumes less energy; however, it is slower. Compared to *16-SCmx*, *Dyn-SC* reduces the execution time and the ED by 28% and 15%, respectively, at the cost of 19% more energy. This is a very favorable tradeoff.

Dyn-SC is much better than *Dyn-HPRef/8-DVFS+* and substantially better than *Dyn-HPMode/16-Pipe2Vdd*. The comparison of *Dyn-SC* to *Dyn-HPMode/16-Pipe2Vdd* shows the impact of ScalCore's pipeline stage fusing and storage struc-

ture resizing, over simply having two V_{dd} domains in the pipeline. We can see that *Dyn-SC* reduces the execution time, energy, and ED of *Dyn-HPMode/16-Pipe2Vdd* by 14%, 10%, and 21%, respectively. Hence, both contributions of the ScalCore design, namely dual V_{dd} domains in the pipeline, and the resulting pipeline stage fusing and structure resizing, are beneficial.

The transition overhead between modes in *Dyn-SC* is negligible because of the relatively low frequency of transitions. Specifically, *Dyn-SC* performs 6-32 transitions in 130-700 million cycles. Overall, ScalCore with dynamic reconfiguration is a very attractive design for applications that change phases.

9. RELATED WORK

There are prior works on variable-latency pipelines. For example, when process variation makes a pipeline stage too slow, ReCycle [45] uses cycle-time stealing between stages, and ReVIVaL [26] makes a transparent latch opaque. On the other hand, Collapsible Pipelines [16] dynamically make a latch transparent when there is a bubble in the pipeline. Pipeline Stage Unification [42] combines every two or every four pipeline stages when the frequency is low. In this paper, we focus only on combining storage-intensive stages under dual-V_{dd} pipelines. Another variable latency pipeline technique is GALS [41, 48]. GALS operates different stages at different *f*, while we keep the same *f* for all stages.

Many works propose reconfigurable architectures that adapt to resource occupancy (e.g., [1, 24, 35]). The goal is to improve either performance or power efficiency. While these ideas are similar to ours, we apply the changes only to adapt to a low-V_{dd} mode, and have only two settings. In general, these techniques and our work are complementary.

There is past work on dual-V_{dd} architectures. Dreslinski et al. [9] apply a different V_{dd} to the core and to the caches. The same approach is followed in the Intel Claremont prototype [17, 38]. A detailed comparison to Claremont is provided in Section 5.3. Miller et al. [28, 29] provide two V_{dd} rails and allow a core to dynamically switch between them. In the presence of process variation, this technique hides speed heterogeneity and tolerates slow functional units. Our work is different in that different stages of the same pipeline have different V_{dd}s. In addition, pipeline stages do not switch between V_{dd} rails; ScalCore merely changes the rails' V_{dd}.

10. CONCLUSION

The goal of this paper was to design a voltage scalable core — namely, one that can work in high-performance mode (HPMode) at nominal V_{dd}, and in a very energy-efficient mode (EEMode) at low V_{dd}. ScalCore introduces two ideas to operate energy-efficiently in EEMode. First, it applies two low V_{dd}s to the pipeline: one to the logic stages (V_{logic}) and a higher one to the storage-intensive stages. Second, it increases the low V_{dd} of the storage-intensive stages (V_{op}) even further and exploits the speed differential to the logic ones by either fusing storage-intensive stages or increasing the size of storage structures in the pipeline. Simulations of 16 cores show that a design with ScalCores in EEMode is much more energy-efficient than one with conventional cores and aggressive DVFS: for approximately the same power consumption, ScalCores reduce the average execution time of programs by 31%, the energy consumed by 48%, and the *ED* product by 60%. In addition, dynamically switching between EEMode and HPMode is very effective: it reduces the average execution time and *ED* product by an additional 28% and 15%, respectively, over running in EEMode all the time.

11. REFERENCES

[1] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Dynamically Tuning Processor Resources with Adaptive Processing," *Computer*, December 2003.

[2] I. Burcea and A. Moshovos, "Phantom-BTB: A Virtualized Branch Target Buffer Design," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2009.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," in *International Solid-State Circuits Conference*, February 2000.

[4] A. Buyuktosunoglu, D. Albonesi, S. Schuster, D. Brooks, P. Bose, and P. Cook, "A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors," in *Great Lakes Symposium on VLSI*, March 2001.

[5] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch, "Practical Strategies for Power-Efficient Computing Technologies," *Proceedings of the IEEE*, February 2010.

[6] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-Driven Near-Threshold SRAM Design," *IEEE Transactions on VLSI Systems*, November 2010.

[7] H. D. Cho, K. Chung, and T. Kim, "Benefits of the big.LITTLE Architecture," February 2012, Samsung White Paper. [Online]. Available: http://www.arm.com/files/downloads/Benefits_of_the_big.LITTLE_architecture.pdf

[8] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, February 2010.

[9] R. Dreslinski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester, "An Energy Efficient Parallel Architecture Using Near Threshold Operation," in *International Conference on Parallel Architectures and Compilation Techniques*, September 2007.

[10] M. Evers, P.-Y. Chang, and Y. N. Patt, "Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches," in *International Symposium on Computer Architecture*, May 1996.

[11] A. Gonzalez, F. Latorre, and G. Magklis, "Processor Microarchitecture: An Implementation Perspective," *Synthesis Lectures on Computer Architecture*, December 2010.

[12] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," September 2011, ARM White Paper. [Online]. Available: http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf

[13] S. Hsu, A. Agarwal, M. Anders, H. Kaul, S. Mathew, F. Sheikh, R. Krishnamurthy, and S. Borkar, "A 2.8GHz 128-Entry 152b 3-Read/2-Write Multi-Precision Floating-Point Register File and Shuffler in 32nm CMOS," in *Symposium on VLSI Circuits*, June 2012.

[14] International Technology Roadmap for Semiconductors (ITRS), "ITRS 2012 Edition." [Online]. Available: http://www.itrs2.net.

[15] F. Ishihara, F. Sheikh, and B. Nikolic, "Level Conversion for Dual-Supply Systems," *IEEE Transactions on VLSI Systems*, February 2004.

[16] H. M. Jacobson, "Improved Clock-Gating Through Transparent Pipelining," in *International Symposium on Low Power Electronics and Design*, August 2004.

[17] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, A. Kumar, S. Gb, R. Ramanarayanan, V. Erraguntla, J. Howard, S. Vangal, S. Dighe, G. Ruhl, P. Aseron,

H. Wilson, N. Borkar, V. De, and S. Borkar, "A 280mV-to-1.2V wide-operating-range IA-32 processor in 32nm CMOS," in *International Solid-State Circuits Conference*, February 2012.

[18] C.-H. Jan, U. Bhattacharya, R. Brain, S.-J. Choi, G. Curello, G. Gupta, W. Hafez, M. Jang, M. Kang, K. Komeyli, T. Leo, N. Nidhi, L. Pan, J. Park, K. Phoa, A. Rahman, C. Staus, H. Tashiro, C. Tsai, P. Vandervoorn, L. Yang, J.-Y. Yeh, and P. Bai, "A 22nm SoC Platform Technology Featuring 3-D Tri-gate and High-k/Metal Gate, Optimized for Ultra Low Power, High Performance and High Density SoC Applications," in *International Electron Devices Meeting*, December 2012.

[19] E. Karl, Z. Guo, J. Conary, J. Miller, Y.-G. Ng, S. Nalam, D. Kim, J. Keane, U. Bhattacharya, and K. Zhang, "A 0.6V 1.5GHz 84Mb SRAM Design in 14nm FinFET CMOS Technology," in *International Solid-State Circuits Conference*, February 2015.

[20] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "VARIUS-NTV: A Microarchitectural Model to Capture the Increased Sensitivity of Manycores to Process Variations at Near-Threshold Voltages," in *International Conference on Dependable Systems and Networks*, June 2012.

[21] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "EnergySmart: Toward Energy-Efficient Manycores for Near-Threshold Computing," in *International Symposium on High Performance Computer Architecture*, February 2013.

[22] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "Near-Threshold Voltage (NTV) Design: Opportunities and Challenges," in *Design Automation Conference*, June 2012.

[23] N. S. Kim, S. Draper, S.-T. Zhou, S. Katariya, H. R. Ghasemi, and T. Park, "Analyzing the Impact of Joint Optimization of Cell Size, Redundancy, and ECC on Low-Voltage SRAM Array Total Area," in *IEEE Transactions on VLSI Systems*, December 2012.

[24] Y. Kora, K. Yamaguchi, and H. Ando, "MLP-aware Dynamic Instruction Window Resizing for Adaptively Exploiting Both ILP and MLP," in *International Symposium on Microarchitecture*, December 2013.

[25] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *International Symposium on Microarchitecture*, December 2009.

[26] X. Liang, G.-Y. Wei, and D. Brooks, "ReVIVaL: A Variation-Tolerant Architecture Using Voltage Interpolation and Variable Latency," in *International Symposium on Computer Architecture*, June 2008.

[27] D. Markovic, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey, "Ultralow-Power Design in Near-Threshold Region," *Proceedings of the IEEE*, February 2010.

[28] T. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips," in *International Symposium on High Performance Computer Architecture*, February 2012.

[29] T. Miller, R. Thomas, and R. Teodorescu, "Mitigating the Effects of Process Variation in Ultra-low Voltage Chip Multiprocessors using Dual Supply Voltages and Half-Speed Units," in *IEEE Computer Architecture Letters*, 2012.

[30] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Understand Large Caches," April 2009.

[31] K. U. Mutsunori, M. Igarashi, T. Ishikawa, M. Kanazawa, M. Takahashi, M. Hamada, H. Arakida, T. Terazawa, and T. Kuroda, "Design Methodology of Ultra Low-power MPEG4 Codec Core Exploiting Voltage Scaling Techniques," in *Design Automation Conference*, June 1998.

[32] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE Journal of Solid-State Circuits*, March 2006.

[33] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-Effective Superscalar Processors," in *International Symposium on Computer Architecture*, June 1997.

[34] I. Park, C. L. Ooi, and T. N. Vijaykumar, "Reducing Design Complexity of the Load/Store Queue," in *International Symposium on Microarchitecture*, December 2003.

[35] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker, "Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems," in *International Symposium on Computer Architecture*, June 2013.

[36] M. Qazi, M. Sinangil, and A. Chandrakasan, "Challenges and Directions for Low-Voltage SRAM," *IEEE Design Test of Computers*, January 2011.

[37] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC Simulator," January 2005.

[38] G. Ruhl, S. Dighe, S. Jain, S. Khare, and S. Vangal, "IA-32 Processor with a Wide-Voltage-Operating Range in 32-nm CMOS," *IEEE Micro*, March-April 2013.

[39] J. Sartori, B. Ahrens, and R. Kumar, "Power Balanced Pipelines," in *International Symposium on High Performance Computer Architecture*, February 2012.

[40] E. Seevinck, F. List, and J. Lohstroh, "Static-Noise Margin Analysis of MOS SRAM Cells," *IEEE Journal of Solid-State Circuits*, October 1987.

[41] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency scaling," in *International Symposium on High Performance Computer Architecture*, February 2002.

[42] H. Shimada, H. Ando, and T. Shimada, "Pipeline Stage Unification: A Low-Energy Consumption Technique for Future Mobile Processors," in *International Symposium on Low Power Electronics and Design*, August 2003.

[43] M. Sinangil, N. Verma, and A. Chandrakasan, "A Reconfigurable 8T Ultra-Dynamic Voltage Scalable (U-DVS) SRAM in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, November 2009.

[44] The Tech Report, "The Exynos 5433 SoC." [Online]. Available: http://techreport.com/review/27539/samsung-galaxy-note-4-with-the-exynos-5433-processor/2

[45] A. Tiwari, S. Sarangi, and J. Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation," in *International Symposium on Computer Architecture*, June 2007.

[46] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *International Symposium on Computer Architecture*, May 1996.

[47] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," in *Operating Systems Design and Implementation*, November 1994.

[48] 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*, October 2004.

[49] Y. Zhao, J. Li, and K. Mohanram, "Multi-Port FinFET SRAM Design," in *Great Lakes Symposium on VLSI*, May 2013.