Efficient Power Gating of SIMD Accelerators through Dynamic Selective Devectorization in a HW/SW Co-designed Environment

RAKESH KUMAR, Universitat Politècnica de Catalunya, Barcelona, Spain
ALEJANDRO MARTÍNEZ, Intel Barcelona Research Center, Intel Labs
ANTONIO GONZÁLEZ, Intel Barcelona Research Center, Intel Labs - UPC

Leakage energy is a growing concern in current and future microprocessors. Functional units of microprocessors are responsible for a major fraction of this energy. Therefore, reducing functional unit leakage has received much attention in the recent years. Power gating is one of the most widely used techniques to minimize leakage energy. Power gating turns off the functional units during the idle periods to reduce the leakage. Therefore, the amount of leakage energy savings is directly proportional to the idle time duration. This paper focuses on increasing the idle interval for the higher SIMD lanes. The applications are profiled dynamically, in a Hardware/Software co-designed environment, to find the higher SIMD lanes usage pattern. If the higher lanes need to be turned-on for small time periods, the corresponding portion of the code is devectorized to keep the higher lanes off. The devectorized code is executed on the lowest SIMD lane. Our experimental results show that the average energy savings of the proposed mechanism are 15%, 12% and 71% greater than power gating, for SPECFP2006, Physicsbench and Eigen benchmark suites respectively. Moreover, the slowdown caused due to devectorization is negligible.

Categories and Subject Descriptors: **C.1.2 [Computer System Organization]**: Multiprocessor-SIMD; **D.3.4 [Software]**: Processors-Optimization

General Terms: Algorithms, Performance, Experimentation

Additional Key Words and Phrases: Hardware/Software Co-designed Processors, Devectorization, Power Gating, Leakage

1. INTRODUCTION

Modern microprocessors need to meet the high performance/throughput requirements of the increasingly complex applications. In addition, they have to provide such high performance under a very stringent power envelope. Moreover, the increase in leakage power at sub-100-nanometer technologies has put further constraints on the power budget. Therefore, it is of prime importance for computer architects to achieve a balance between the energy consumption and performance.

Single Instruction Multiple Data (SIMD) accelerators are incorporated in the processors, from different computing domains, to improve performance, especially for compute intensive data parallel applications [Intel Software Developer´s Manual; D´Arcy et al. 1999; Baron 2005; Diefendorffet al. 2000; Kahle et al. 2005; Lee 1996;

This article extends an earlier version, Dynamic Selective Devectorization for Efficient Power Gating of SIMD units in a HW/SW Co-designed Environment [Kumar et al. 2013], that appeared in the 25th IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2013).

Authors' addresses: Rakesh Kumar (corresponding author), Department of Computer Architecture, Universitat Politècnica de Catalunya, Barcelona, Spain; Alejandro Martínez, Intel Barcelona Research Center, Barcelona, Spain; Antonio González, Intel Barcelona Research Center, Barcelona, Spain and Department of Computer Architecture, Universitat Politècnica de Catalunya, Barcelona, Spain. email: rkumar@ac.upc.edu

DOI:http://dx.doi.org/10.1145/0000000.0000000

Sporny et al. 2002]. However, due to their wider datapaths, they become main source of leakage energy for applications lacking data level parallelism. Therefore, it is crucial to control the leakage of these accelerators when they are not being utilized.

Many leakage control techniques have been studied [Hu et al. 2004; Kim et al. 2010; Tschanz et al. 2003; Ye et al. 1998], power gating being one of the most prominent ones. Power gating cuts the supply voltage to the idle functional units, resulting in leakage energy savings. The amount of leakage energy saved is directly proportional to the length of time interval for which the circuit remains idle. The longer the idle time interval, the more is the leakage energy saving. Therefore, it is desirable to have longer idle time intervals to save maximum leakage energy. However, power gating has an energy and performance overhead associated with it. Certain amount of energy is required to turn a functional unit off and then on again, resulting in energy overhead. Moreover, a certain number of cycles are required before the functional unit can be used after starting the turn on procedure, resulting in performance penalty.

It is important to consider two special cases in power gating context:

(1) Small idle intervals during periods of high utilization
(2) Small busy intervals during otherwise idle interval

In the first case, a functional unit is awakened too early after turning it off. In this case, power gating energy overhead might not be offset by the leakage energy savings and power gating will result in net energy loss. Due to their obvious adverse effects on the net energy savings, several mechanisms have been studied to avoid such cases [Lungu et al. 2009; Youssef et al. 2006]. In the second case, the functional unit is awakened only for a small period of time before it is tuned off again. Power gating benefits can be increased if, somehow, the functional units can be kept off during these intervals. The gain here is twofold:

(1) Since the functional unit is not turned on and then off again, there is no energy overhead.
(2) Avoiding to turn on the functional unit also saves the performance overhead of power gating.

However, an alternate functional unit is required to avoid turning on the power gated (turned off) unit. The work presented in the paper focuses on reducing these cases to improve the net energy savings.

SIMD accelerators have duplicated functional units/lanes to perform several independent operations in parallel. Lowest SIMD lane executes scalar/unvectorized code, whereas, the higher SIMD lanes come into the action when the application code is vectorized. In the cases when the higher SIMD lanes are power gated and need to be tuned on only for smaller periods of time, the corresponding portion of the code can be devectorized and executed on the lowest lane. Thus, the energy and performance overhead of power gating the higher SIMD lanes can be saved, resulting in increased net energy savings. However, the portions of the application to be devectorized should be chosen cautiously, as aggressive devectorization might also result in significant slowdown. Furthermore, the slowdown might result in a net energy loss due to extra leakage energy incurred in the entire core.

One of the ways of choosing devectorizable portions of the application is to profile the application offline and then guiding the compile time vectorizer to vectorize only the specific portions of the application. This method, however, has two major

drawbacks. First, the execution profile of applications might change with the input. Thus, when an application is executed with an input other than the one with which it was profiled, the profile guided optimizations will not help. It might even result in slowdown if the frequently executed portions with the current input are not vectorized. Secondly, the existing code has to be recompiled to get benefits of the new techniques. Another alternative is to profile the applications dynamically and choose the devectorizable portions of the code at runtime, for the current input. Dynamic Binary Translators/Optimizers (DBTO) and Hardware/Software (HW/SW) co-designed processors both provide this kind of runtime profiling and optimization opportunities. HW/SW co-designed processors also provide additional advantage of being able to incorporate new hardware features transparently to the software stack. Therefore, we choose HW/SW co-designed processor over DBTOs even though we don´t rely heavily on this feature.

We propose to extract maximum vectorization opportunities at compile time. Then, at run-time, profile the application dynamically to find out the candidates for devectorization. Therefore, dynamic selective devectorization discovers and devectorizes only the portions of code that help improving the power gating efficiency without having a significant effect on the performance. The main contributions of this work can be summarized as:

(1) Proposes a mechanism to increase power gating efficiency by increasing the idle interval duration.
(2) Proposes a dynamic selective devectorization algorithm to keep the higher SIMD lanes idle for long time duration without significant effect on performance.
(3) A dynamic profiling technique to discover devectorizable portions of the code.
(4) Evaluation of the proposals and comparison with power gating. The energy savings of the proposed technique are 15%, 12% and 71% greater than power gating for SPECFP2006, Physicsbench and Eigen benchmarks respectively.
(5) A sensitivity study of effects of breakeven threshold and wakeup delay variations on the energy savings of proposed mechanism.

For the rest of the paper, Section 2 provides a background and related work on HW/SW Co-designed processors and power gating. Section 3 provides the motivation for the work presented in this paper. Section 4 describes the proposals of dynamic profiling and devectorization. Evaluation of the proposals using SPECFP2006, Physicsbench and Eigen benchmarks is presented in Section 5. Finally, Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

HW/SW Co-designed processors [Dehnert et al. 2003; Ebcioğlu et al. 1997; Sathaye et al. 1999] have enticed researchers for more than a decade. Moreover, there is a renewed interest in them in both industry and academia [Lupon et al. 2014; Branković et al 2014; Wang et al. 2013; Pavlou et al 2012; Neelakantam et al 2010]. These processors employ a software layer that resides between the hardware and the operating system. This software layer allows host and guest ISAs to be completely different, by translating the guest ISA instructions to the host ISA dynamically. The host ISA is the ISA which is implemented in the hardware, whereas, guest ISA is the one for which applications are compiled. The basic idea behind these processors is to have a simple host ISA to reduce power consumption and complexity.

The software layer translates the guest ISA instructions to the host ISA in multiple phases. Generally, in the first phase, guest ISA instructions are interpreted. In the rest of the phases, guest code in translated and stored in a code cache, after applying several dynamic optimizations, for faster execution. The number of translation phases and optimizations in each phase are implementation dependent.

As leakage is becoming a growing concern in the current microprocessor designs, several leakage control mechanisms have been studied [Hu et al. 2004; Kim et al. 2010; Tschanz et al. 2003; Ye et al. 1998]. All these mechanisms try to reduce leakage when the circuit is in idle state. Power gating [Hu et al. 2004] consists of shutting down parts of the circuit by cutting their power supply by means of high threshold header or footer transistors, called sleep transistors. Supply Switching with Ground Collapse (SSGC) [Kim et al. 2010] is similar to power gating as this technique also cuts the power supply to the circuit. However, it is more effective than power gating in reducing leakage in data retention circuits. Input vector activation [Ye et al. 1998] changes the input of the circuit to keep the maximum number of transistors in the off state. As the number of off transistors between power supply and ground increases the leakage reduces. Adoptive body biasing techniques [Tschanz et al. 2003; Ananthan et al. 2004] increase transistor threshold voltage by applying a reverse bias at transistor body. The increased threshold voltage reduces the sub-threshold and gate leakages.

Among all these leakage control mechanisms power gating has received maximum attention. Several in-depth studies have been carried out to improve the efficiency of power gating. Hu [Hu et al. 2004] showed several key intervals in power gating, three of the most important being: idle detect interval, breakeven threshold and wakeup delay. Idle detect interval is the amount of time needed to decide when to shut down a unit. At the end of idle detect interval a sleep signal is generated to shut down the functional unit. Breakeven threshold is the amount of time a unit must remain shut down to offset the power gating energy overhead. Waking up a unit before this threshold, results in net energy loss. Finally, wakeup delay is the amount of time needed before the unit can be used after turning it on. Therefore, a higher wakeup delay translates to a higher performance penalty.

Hu also proposed a branch prediction based and a counter based technique to generate sleep signal. In branch prediction based technique the unit is shut down after a branch misprediction is detected whereas, the counter based technique generates the sleep signal after the unit has been idle for a fixed number of cycles. As noted before, if the power gated unit needs to be awakened before crossing the breakeven threshold, power gating suffers a net energy loss. Several techniques has been proposed to minimize this energy loss [Agarwal et al. 2006; Lungu et al. 2009; Youssef et al. 2006]. A. Youssef [Youssef et al. 2006] proposed to change idle detect interval dynamically. Their proposal increases the idle interval during the period of high utilization, when the functional units are being used frequently. Since the probability of a unit being awakened before crossing the breakeven threshold is high during these periods, increasing the idle detect interval reduces the number of power gating instances and hence the likelihood of energy loss. On the contrary, they reduce the idle detect interval during the phases of low activity to increase the number of powered off cycles and hence the energy savings. A. Lungu [Lungu et al. 2009] proposed to use success monitors to measure the success of power gating during a certain time interval. If power gating saves energy it is applied in the next interval as well, if possible. Otherwise, power gating would be deactivated in the next time interval even if a possibility existed. K. Agarwal [Agarwal et al. 2006] proposed to have multiple sleep modes in power gating. Each mode has different wakeup delay

and energy savings. By trading-off these two parameters during periods of different activity they achieve higher energy savings.

All of these techniques focus on improving the power gating efficiency by improving the decision of when to shut down a unit. On the other hand, our work focuses on how to keep a unit shut down for longer time intervals once it is already power gated. Even though we target SIMD accelerators to show the potential of the proposal, it can be applied the any functional units with multiple instances. To increase the length of the idle periods, the higher SIMD lanes usage is profiled dynamically. Then the portions of the code corresponding to the low utilization periods of higher lanes are located. This piece of code is then devectorized and executed on the lowest SIMD lane. Furthermore, our technique is applicable as long as there is some support for dynamic profiling and optimizations, be it HW/SW co-designed processors or DBTOs.

3. MOTIVATION

Power gating net energy savings depend on the leakage energy saved by putting the functional units in sleep mode and the energy overhead of doing power gating itself. The energy overhead comes due to the fact that the sleep signal needs to be generated and distributed to the appropriate functional units. Moreover, turning the sleep transistor on and off also requires energy. Therefore the net energy saving of power gating can be computed as:

$$\text{Net Energy Savings} = E_L * \sum_{k=0}^{n} off\_cycles[k] - (n * E_{overhead})$$

Where E_L is the leakage energy per cycle, $E_{overhead}$ is power gating energy overhead per power gating instance and n is the number of power gating instances. Thus, having large *off_cycles* with minimum number of power gating instances (n) results in maximum energy savings. Furthermore, a functional unit cannot be used immediately after putting the power supply back on, resulting in performance loss. Therefore, to get maximum leakage savings at minimum performance penalty, a functional unit needs to be kept shut down for longer time intervals, with minimum number of power gating instances.

Reducing the number of power gating instances not only reduces the power gating energy overhead but also has a secondary energy saving effect. Imagine a situation where a power gated functional unit needs to be awakened to execute just one instruction. First, we need to disable the sleep signal and wait for "wakeup latency" number of cycles; then the instruction is executed. Next, we have to wait for "idle detect" number of cycles before enabling the sleep signal; then after waiting for another "breakeven threshold" number of cycles the leakage energy savings of power gating begin. However, if we execute this instruction on a different functional unit which is already ON, we can save "wakeup delay + number of cycles required for instruction execution + idle detect + breakeven threshold" cycles of leakage energy in addition to saving the power gating energy overhead.

Functional unit usage profile of an application changes during its execution. During the low utilization period the function unit is used scarcely. Therefore, power gating targets these periods for leakage savings. However, every time the functional unit is needed, it needs to be awakened from the power gated state and needs to be shut down afterwards. The wakeup and shutting down energy overhead reduces overall leakage energy savings. If the functional unit is kept turned off and the corresponding code is executed on some other functional unit (which are already on);

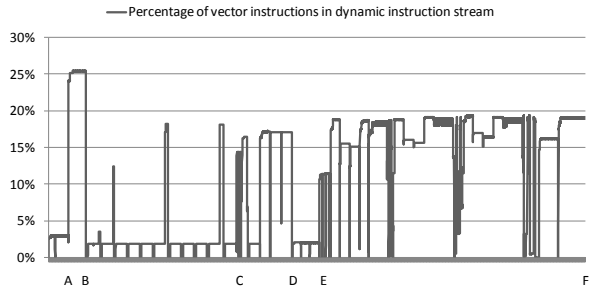


Figure 1 Percentage of vector instruction (excluding memory instructions) in the dynamic instruction stream over the time (4 billion instructions).

the effectiveness of power gating in saving leakage energy can be increased. Specifically for SIMD accelerators, higher SIMD lanes can be switched off during sporadic usage period and the corresponding code can be executed on lowest lane after devectorization.

We profiled SPECFP2006 using a dynamic profiling technique described in the next section, to discover the higher SIMD lanes usage pattern. Figure 1 shows the percentage of vector instructions (higher lanes usage profile) in the dynamic instruction stream over the execution time for 434.zeusmp. The higher lanes usage profile shown in the figure is for 4 billion instruction executed starting from the most frequently executed function/routine. Moreover, the shown vector instruction profile does not include memory instructions since they do not use SIMD functional units. As can be seen in the figure, higher lane usage profile changes during the execution. During the time intervals A-B, C-D and E-F around 20% of the dynamic instructions are vector instructions and utilize higher SIMD lanes. Therefore, higher SIMD lanes need be activated during these intervals. On the other hand, during the time intervals 0-A, B-C and D-E only less than 3% of the dynamic instructions are vector instructions. During these intervals power gating will wakeup SIMD lanes from sleep state, for short durations of time, to execute these vector instructions.

We propose to devectorize the portion of code corresponding to the time intervals 0-A, B-C and D-E, if it does not affect the percentage of vectorized code in the other time intervals. Devectorizing this piece of code results in fewer (in some cases none) vectorized instructions during these time intervals. Therefore the number of power gating instances also reduces during these intervals. As a result, the power gating energy overhead diminishes and the net leakage savings increase. However, the dynamic energy consumption of the lowest lane increases, as it has to execute more instructions now. Nevertheless, as will be shown in the performance evaluation section, this increase is relatively small compared to the reduction in the leakage energy.

4. PROFILING AND DEVECTORIZATION

This section provides the details of the dynamic profiling and devectorization schemes. We start with a brief motivation for dynamic profiling. It is followed by a brief description of the software layer of the modeled HW/SW co-designed processor leading to profiling and devectorization details.

Profiling is necessary to discover the code segments that can be devectorized to keep the higher SIMD lanes power gated without affecting the performance. These code segments must not be performance critical, as devectorizing performance critical code will result in excessive slowdown. Moreover, due to the slowdown caused by devectorization overall energy consumption will increase. It is important to note that the performance critical code segments of an application might change with the

input. Therefore, profiling the applications offline with a particular set of inputs will not help in deciding which code segments to devectorize. For that reason, we choose to profile the applications dynamically at runtime. Dynamic profiling discovers non-performance critical devectorization candidates and pass this information to the runtime devectorizer. The selected code segments are then devectorized, resulting in effective power gating of SIMD units.

The software layer of our HW/SW co-designed processor is called Translation Optimization Layer (TOL). It operates in three translation modes for generating host code from the guest x86 code: Interpretation Mode (IM), Basic Block Translation Mode (BBM) and Superblock Translation Mode (SBM). We collect the profiling information for the basic blocks in BBM. This information is then used in SBM during superblock optimization phase to decide whether or not to devectorize the given superblock.

4.1 Profiling and Superblock Creation

TOL starts by interpreting guest x86 instruction stream in IM. When a basic block is executed more than a predetermined number of times, TOL switches to BBM. In this mode, the whole basic block is translated and stored in the code cache and the rest of the executions of this basic block are done from the code cache. Moreover, branch profiling information for direction and target of branches is also collected. Once the execution of a basic block exceeds another predetermined threshold, TOL creates a bigger optimization region, called superblock, using the branch profiling information collected during BBM.

A superblock generally includes multiple basic blocks following the biased direction of branches. Moreover, branches inside the superblocks are converted to "asserts" so that a superblock can be treated as a single-entry, single-exit sequence of instructions. This enable more aggressive optimizations. "Asserts" are similar to branches in the sense that both checks a condition. Branches determine the next instruction to be executed based on the condition, however asserts have no such effect. If the condition is true assert does nothing. However, if the condition evaluates to false, the assert "fails" and the execution is restarted from a previously saved checkpoint in IM. Dynamic selective devectorization is done only on superblocks.

The applications are profiled in BBM to get the following information

(1) Execution and Branch profiling information:

Software counters are used to count the number of times a basic block has been executed in BBM. Besides, software counters are also employed to get the biased direction of branches. As mentioned earlier, this information is used to create superblocks in SBM. Furthermore, we also profile the higher SIMD lanes usage pattern that helps us in deciding which superblocks to devectorize.

(2) Higher SIMD lanes usage pattern:

To monitor the usage of higher SIMD lanes an N-bit shift register is employed. Before executing an instruction, the content of this register are shifted by one and the new position is set to 1 if the current instruction is a vector instruction, otherwise it is reset to zero. Therefore, the number of ones in the shift register gives the number of vector instructions executed in the last N instructions.

Each basic block in BBM has a software "devec" counter associated with it. Every time a basic block, having at least one vector instruction, is executed in BBM, the contents of the shift register are read. If the number of ones in the shift register are

less than a threshold (DV_{th}), it would be desirable to devectorize the basic block, if it is included in a superblock. The devectorization is desirable in this case because having less number of vector instructions indicate low usage of higher SIMD lanes. Therefore, devectorizing this code will help improving power gating efficiency without a significant impact on the overall performance. To increase the devectorization likelihood of this basic block the devec counter is incremented. However, if during the next execution of the same basic block the number of ones in the shift register is more that DV_{th}, the devec counter is decremented. It indicates that devectorization is not favored due to more utilization of higher SIMD lanes. Therefore, the final decision of whether to devectorize the basic block or not depends on the shift register values during all the executions of the basic block in BBM. This helps in devectorizing only the basic blocks which are executing during the low usage phase of higher SIMD lanes like B-C in Figure 1.

While creating a superblock devec counters of all the basic blocks included in the superblock are examined. If all the counters are greater than a predetermined threshold, the superblock is devectorized. Otherwise, the superblock is kept in the vectorized form. This selective devectorization of superblocks improves leakage energy savings through power gating while maintaining the performance.

4.2 Optimizations

In this phase, several standard optimizations are applied dynamically. First of all, the superblock is converted into Static Single Assignment (SSA) form to remove anti and output dependences. Then, the optimizations Constant Propagation, Copy Propagation, Constant Folding, Common Sub-expression Elimination and Dead Code Elimination are applied. The next step is to generate the Data Dependence Graph (DDG). During DDG creation, Redundant Load Elimination and Store Forwarding are also applied to improve the quality of the generated code.

4.3 Devectorization

Once a superblock has been identified for devectorization through profiling, it goes through a devectorization phase. The devectorization pass simply replaces vector instructions by their corresponding scalar instructions and generates permutation instructions if required. Moreover, vector memory instructions are not devectorized since they do not use SIMD functional units.

As shown in Algorithm 1a, "***devect***" is the top level routine that receives the superblock "SB" to be devectorized. The routine goes over all the instructions in the superblock in the program order. All the vector instructions (excluding memory access instructions) are candidates for devectorization. The first step in devectorization is to find devectorization length (*get_devec_len*). It is the number of scalar instructions to be generated corresponding to the vector instruction. Then the scalar opcode for the scalar instructions to be generated is obtained (*get_scalar_opcode*). Next, the "***get_scalar_in_reg***" routine of Algorithm 1b checks if the input vector registers of the current instruction have already been mapped to scalar registers or not. If the producers of the current instruction have already been devectorized, the corresponding input registers are already mapped to the output scalar registers of the scalar producers. However, if the producers cannot be devectorized (producers being vector memory loads or live-in of the superblock), an Unpack instruction is generated (*generate_Unpack_insn*). This Unpack instruction distributes the contents of the input vector register to a set of scalar registers depending on the devect length. Once all the input vector registers have been mapped to scalar registers, new output scalar registers are allocated (*allocate_reg*)

ALGORITHM 1A. Top Level Dynamic Devectorization Routine

```
devect(SB):
    for each instruction s in SB:
        if s is devectorizable:
            devec_len ← get_devec_len(s)
            scalar_op ← get_scalar_opcode(s)
            scalar_in_regs ← get_scalar_in_reg (s)

            scalar_out_reg ← ø
            for i ← 0 to devec_len do:
                scalar_out_reg ← scalar_out_reg ∪ allocate_reg()

            for i ← 0 to devec_len do:
                generate_insn(scalar_op, scalar_in_reg, scalar_out_reg)

            add_to_mapped_reg(org_out_reg)

            if org_out_reg is architectural_reg or vectorized_consumer:
                generate_Pack_insn(scalar_out_reg)
```

ALGORITHM 1B. Vector to Scalar Register Mapping

```
get_scalar_in_reg (s)
    scalar_in_regs ← ø

    for each input_register ireg of s:
        if ireg in mapped_regs:
            scalar_in_regs ← scalar_in_regs ∪ get_mapped_reg(ireg)
    else
            generate_Unpack_insn(ireg)
            scalar_in_regs ← scalar_in_regs ∪ get_mapped_reg(ireg)

    return scalar_in_regs
```

for new scalar instructions to be generated. In the next step, the scalar instructions are generated (*generate_insn*) using scalar input and output registers collected during the earlier steps. The vector output register of the current instruction is mapped to the new scalar output registers allocated (*add_to_mapped_reg*). Finally, if the output register is an architecture register or the consumers of the current instruction cannot be devectorized (vector memory stores), a Pack instruction is generated (*generate_Pack_insn*). The Pack instruction collects the values from the scalar output registers and packs them in a new vector register so that it can be used by the vectorized consumers.

As the devectorization proceeds the producer-consumer relations keep changing. Thus it is important to update the predecessor/successors chains. However, it is not shown in the Algorithm 1 for the sake of simplicity.

4.4 Reducing Devectorization Slowdown

Dynamic selective devectorization serializes the parallel portions of code to save energy at small performance cost. To reduce the effect of this serialization on the

performance, we do partial devectorization whenever possible. To better understand partial devectorization, consider a 128-bit wide SIMD accelerator with two 64-bit wide lanes. Each lane can execute either one 64-bit double-precision floating-point operation or two 32-bit single-precision floating-point operations. Devectorized code is executed on the lower lane, so that the higher lane could be switched off.

In general, a single-precision floating-point vector instruction would be devectorized into four single-precision scalar instructions. However, partial devectorization generates only two single-precision "half-vector" instructions. A "half-vector" instruction combines two scalar instructions that can be executed in parallel. The rationale behind partial devectorization is to utilize the whole 64-bit wide vector lane. Since one vector lane can execute two single-precision operations, it is better to partially devectorized the code instead of full devectorization. As a result, the effect of devectorization on performance is reduced while still saving energy by power gating the higher lane. We propose to have "half-vector" instructions in the host processor ISA. However, these instructions are transparent to the compiler/user and are generated dynamically by the runtime devectorizer. The co-designed nature of the host processor allows including new instructions without any change in the guest ISA or compiler/recompiling.

5. PERFORMANCE EVALUATION

DARCO [Pavlou et al. 2011], which is an infrastructure for evaluating HW/SW co-designed virtual machines, is used to evaluate the proposals. DARCO executes guest x86 binary on a PowerPC-like RISC host architecture. The proposed profiling and devectorization algorithm are implemented in TOL. Furthermore, for energy consumption analysis McPAT [Li et al. 2009] is integrated with DARCO. Moreover, we consider only the floating point instructions for devectorization because they are the main target of SIMD accelerators. In our experiments, we assume that the host architecture consists of a 128-bit wide SIMD accelerator. Moreover, we consider that the SIMD accelerator is composed of two 64-bit wide lanes.

From power gating point of view SIMD accelerator can be viewed as a single unit or two separate lanes. In other words, both the lanes of the SIMD accelerator can be powered together or separately. If both the lanes are power gated together, we call it combined power gating (CPG). CPG, however, is not efficient, since higher lane, generally, is used lesser than the lower lane. Therefore, power gating the higher lane, even though the lower lane is functional, would result in more power savings. We call this configuration Split Power Gating (SPG). We compare our results with both the configurations. Moreover, the optimizations of Section 4.2 are activated in all three cases: CPG, SPG and with Dynamic Selective Devectorization (DSD). Furthermore, the results presented are for the modeled host processors and include profiling and translation overheads if not mentioned otherwise.

To measure the success of the proposals we use applications from SPECFP2006 [Standard Performance Evaluation Corporation], Physicsbench [Yeh et al. 2007] and Eigen [Eigen] benchmarks suites. For SPECFP2006 we instrument the benchmarks, using PIN [Luk et al. 2005], to find the most frequently executed routines. Then we simulate four billion instructions starting from these routines. The benchmarks in Physicsbench are executed till completion. For Eigen benchmarks also we execute only four billion instructions. The benchmarks are compiled with Intel ICC version 12.1.4, optimization flags "-O3" and vectorization flag "-xSSE3". Only floating point benchmarks in SPEC2006 are considered for evaluation since the floating point code is the main target of our proposals.

Table I. Processor Microarchitectural and McPAT Parameters.

Parameter	Value
Processor Microarchitectural Parameters	
L1 I-cache	64KB, 4-way set associative, 64-byte line, 1 cycle hit, LRU
L1 D-cache	64KB, 4-way set associative, 64-byte line, 1 cycle hit, LRU
Unified L2 cache	512KB, 8-way set associative, 64-byte line, 6 cycle hit, LRU
Scalar Functional Units (latency)	2 simple int(1), 2 intmul/div (3/10) 2 simple FP(2), 2 FP mul/div (4/20)
Vector/SIMD Functional Units (latency)	1 simple int(1), 1 intmul/div (3/10) 1 simple FP(2), 1 FP mul/div (4/20)
Registers	128-Integer, 128-Vector, 32-FP
Main memory Lat	128 Cycles
McPAT Parameters	
Technology	65nm
Clock Rate	1.5 GHz
Temperature	350 K
Device Type	High Performance

5.1 Models and Parameters

To measure the success of the proposals, we refer to the energy model proposed by Hu [Hu et al. 2004]. However, we changed some of model input values. Their breakeven threshold value is between 9 and 24 cycles. However, as A. Youssef [Youssef et al. 2006] explained, the breakeven threshold value in the real implementations can be more than 100 cycles. We use the breakeven threshold of 150 cycles. The wakeup latency of the functional units is considered to be 10 cycles. Moreover, later we show a sensitivity study for breakeven threshold and wakeup delay variations.

A. Lungu [Lungu et al. 2009] proposed a success monitor based improvement to the time-based power gating mechanism of [Hu et al. 2004]. They use success counters to monitor whether power gating has been successful (saved energy) or harmful (consumed more energy) during a monitoring interval. Power gating in the next monitoring interval is disabled if it has been harmful in the current interval, otherwise it is enabled. This power gating scheme with success monitors serves as the baseline for our proposals. A. Lungu [Lungu et al. 2009] have a fixed idle detect interval in their proposal. However, this interval is varied dynamically in our baseline, depending on the utilization of the functional units (SIMD lanes), as proposed by A. Youssef [Youssef et al. 2006].

We model a simple in-order processor, in congruence with the simple hardware design philosophy of HW/SW co-designed processors, with issue width of two. Microarchitectural parameters for the modeled processor are given in Table I. The table also shows key McPAT parameters used to get the energy consumption of the modeled processor.

5.2 Higher SIMD Lane Usage Profile

The dynamic selective devectorization (DSD) tries to minimize the usage of higher SIMD lane during the low utilization period. As shown in Figure 1 in Section 3, 434.zesump has several time intervals during which the higher SIMD lane usage could be minimized. Minimizing the higher lane usages during these intervals minimizes the number of power gating instances and hence the energy overhead of power gating.

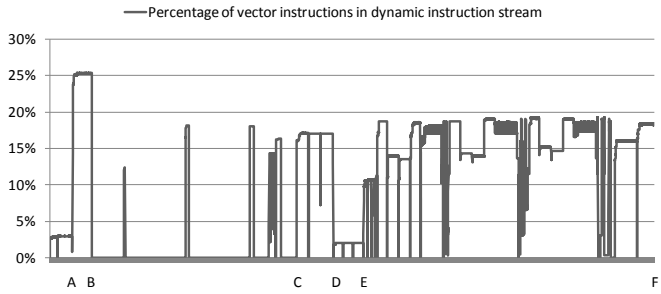


Figure 2 Percentage of vector instruction in the dynamic instruction stream after dynamic selective devectorization for 434.zesump.

Figure 2 shows the vector instruction profile for the same benchmark after dynamic selective devectorization. As the figure shows, the dynamic selective devectorization has been able to reduce the higher SIMD lane usage significantly during the time interval B-C. Therefore, the leakage energy savings by power gating during this interval will be improved. However, the vector code corresponding to the low usage periods 0-A and D-E is not devectorized. This piece of code is executed during the high usage periods also and its devectorization would result in significant performance loss. Therefore, this code is always executed in the vectorized version. Moreover, it is also important to note that the number of vector instructions during the high usage periods A-B, C-D and E-F is the same as before devectorization. Therefore, the effect of devectorization on the performance is going to be negligible.

5.3 SIMD Accelerator Energy Savings

The proposed mechanism reduces the number of higher SIMD lane power gating instances to reduce power gating energy overhead and in turn, the leakage energy of the SIMD accelerator. However, dynamic selective devectorization has an energy and performance overhead associated with it. The energy overhead of DSD includes the following components:

(1) **Lower SIMD Lane Dynamic energy**: The dynamic energy consumption of the lower SIMD lane increases, since it has to execute more instructions.
(2) **Rest of the core Energy:** The rest of the core includes all the components of the core except for the SIMD accelerator. The overall energy of the rest of the core may increase due to:
    a. Dynamic energy consumption increases due to profiling and devectorization of selected superblocks.
    b. Leakage energy of the rest of the core might increase due to the possible slowdown because of devectorization.

Figure 3 and 4 show the SIMD accelerator overall (dynamic + leakage) energy savings for Combined power gating (CPG), Split power gating (SPG) and DSD, without and with DSD overheads respectively. There as several important points to note in these two figures. First of all, the energy overhead of DSD is minimal as most of the benchmarks show similar overall energy savings with and without considering DSD energy overhead. The only exceptions are 410.bwaves and FFT; the reason behind it is explained in Section 5.5 while discussing the performance results. Since the energy savings are similar with and without considering the energy overhead of

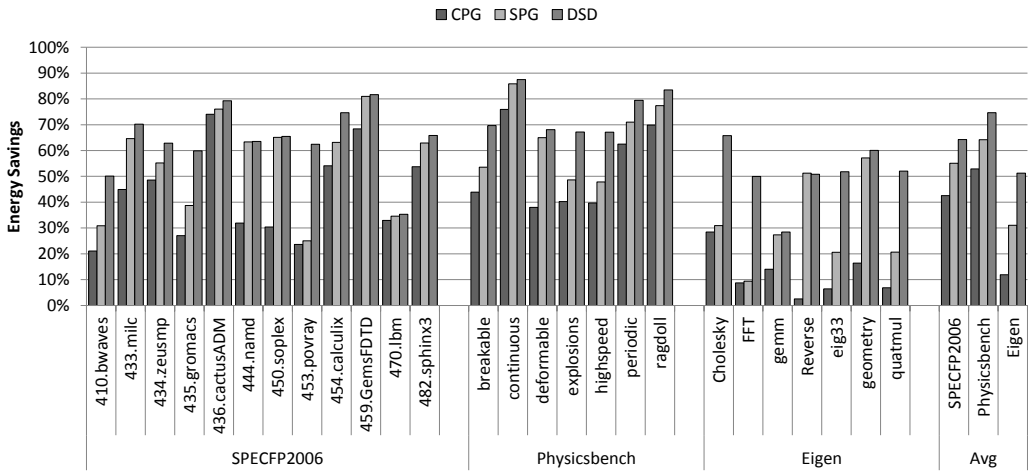


Figure 3 SIMD accelerator overall (dynamic + leakage) energy savings for CPG, SPG and DSD **without including** DSD energy overhead.

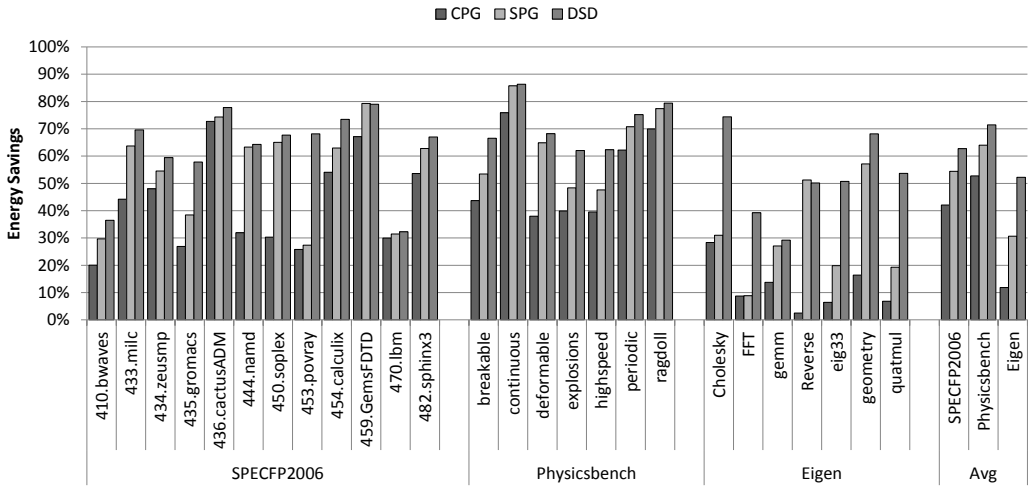


Figure 4 SIMD accelerator overall (dynamic + leakage) energy savings for CPG, SPG and DSD **including** DSD energy overhead.

DSD, the rest of this section focuses on results with overhead, the Figure 4. As this figure shows, DSD outperforms both CPG and SPG significantly. The overall energy savings of the proposed technique are 49%, 35% and 340% greater than CPG and 15%, 12% and 71% greater than SPG for SPECFP2006, Physicsbench and Eigen respectively. In absolute energy savings terms, DSD saves 63%, 72% and 52% overall energy, SPG saves 54%, 64% and 31% overall energy whereas, CPG saves 42%, 53% and 12% overall energy for SPECFP2006, Physicsbench and Eigen respectively. CPG performs worse than SPG because it treats the whole SIMD accelerator as a single unit. Therefore, either both lanes are powered or neither of them. On the other hand, SPG can turn higher lane off even if the lower lane is in use. Therefore, SPG saves more energy than CPG. DSD goes one step ahead and keeps the higher lane powered off (because of devectorized code) for longer periods and outperforms SPG as well.

The benchmarks in Figure 4 can be divided into three categories depending on their energy saving pattern:

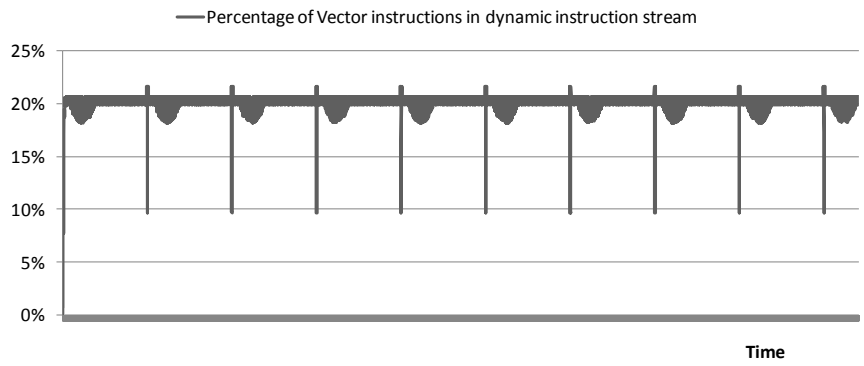


Figure 5 Percentage of vector instruction (excluding memory instructions) in the dynamic instruction stream for 470.lbm.

1) Moderately Vectorizable Benchmarks:

The benchmarks in this category include 410.bwaves, 434.zeusmp, 435.gromacs, 454.calulix, 482.sphinx3, eig33, quatmul and most of the Physicsbench benchmarks. These are the benchmarks for which compilers are able to extract enough vector parallelism, however they are not completely vectorized. Therefore, during the periods of high lower lane usage and idle higher lane, SPG achieves energy savings over CPG by power gating only the higher lane. Moreover, these benchmarks have periods of low higher lane activity, as shown in Figure 1 for 434.zeusmp. The proposed mechanism devectorizes the code corresponding to these intervals and achieve even more energy savings.

2) Highly Vectorizable Benchmarks:

The benchmarks in the category are 436.cactusADM and 470.lbm. These benchmarks are completely vectorizable. In other words, the vectorized code uses either both the vector lanes or none of them. As a result SPG does not provide any additional benefits over CPG. Moreover, the higher lane utilization in these benchmarks is uniform over the execution time as shown in Figure 5 for 470.lbm. Any attempt of devectorization would result in significant performance loss. Therefore, these benchmarks are executed in the vectorized form and no additional leakage energy savings are achieved by DSD either. Furthermore, the overall energy savings for 436.cactus are much more compared to 470.lbm for all the three techniques. The energy savings depend on how long the SIMD accelerator is used during the execution time of the application. Even though both the benchmarks use both the SIMD lanes together, the overall usage of SIMD accelerator is less in 436.cactus, hence power gating provides more energy savings in this benchmark.

3) Unvectorizable Benchmarks:

The benchmarks in this category include 444.namd, 450.soplex, gemm, reverse etc. Since compilers do not find enough vectorization opportunities in these benchmarks, the higher SIMD lane is idle for most of the time. As a result, SPG is able to attain significant energy savings over CPG by power gating the higher SIMD lane alone. However, the proposed mechanism does not have enough opportunities to devectorize because compilers do not vectorize the code. Therefore, DSD does not provide much energy savings over SPG.

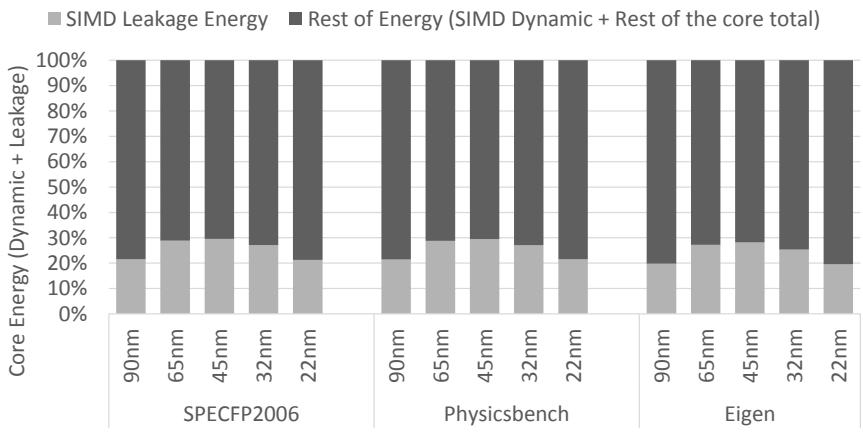


Figure 6 Core overall energy distribution at different technologies.

It is important to note that the most of the benchmarks fall in the first category "Moderately Vectorizable Benchmarks" which is targeted by DSD to achieve additional power savings over power gating. Another interesting point to note in Figure 4 is that in the cases where DSD is not able to reduce leakage, e.g. 470.lbm, the energy overhead of DSD is negligible. Hence, DSD has insignificant energy penalty when it fails to provide leakage benefits.

5.4 Core Energy Savings

This section first presents the ratio of SIMD accelerator leakage energy to the rest of energy (SIMD accelerator dynamic energy + rest of the core overall energy) and then presents core level overall energy savings by CPG, SPG and DSD. Figure 6 shows the energy distribution for five different technologies: 90nm, 65nm, 45nm, 32nm and 22nm. As the figure shows SIMD accelerator leakage energy accounts for 20% to 30% of overall core energy at various technologies. It is also interesting to note that the SIMD leakage energy increases as we move from 90nm to 45nm however, it reduces as the technology is further scaled down to 22nm. The leakage reduction comes from the enhancement in fabrication process below 45nm. Nonetheless, SIMD leakage energy still forms a significant portion of the overall core energy.

C. Bira [Bira et al. 2013] showed that according to Zedboard documentation a dual-core ARM CPU consumes a maximum of 1.25 Watts. They also reported that according to Xilinx power estimation tools the SIMD accelerator consumes 600 mW. This translates to SIMD accelerator being responsible for consuming approximately half of the CPU power. Assuming leakage being responsible for 40-50% of total power, SIMD accelerator leakage is responsible for 20%-25% of total CPU power. This estimation is in coherence with the results of Figure 6.

Figure 7 shows the overall energy savings of the whole core by CPG, SPG and DSD. Since, we consider power gating only the SIMD accelerator and no other functional unit, absolute overall energy savings are not as high as for the SIMD accelerator alone. However, DSD still outperforms both SPG and CPG. DSD energy savings are 48%, 35% and 330% greater than CPG and 15%, 12% and 71% greater than SPG for SPECFP2006, Physicsbench and Eigen respectively. In absolute energy savings terms, DSD saves approximately 19%, 22% and 16% overall energy, SPG saves 16%, 19% and 9% overall energy while CPG saves 13%, 16% and 4% overall energy for SPECFP2006, Physicsbench and Eigen respectively. As the results show, DSD is able to save comparatively more overall core energy than CPG and SPG even when SIMD accelerator is the only power gated functional unit.

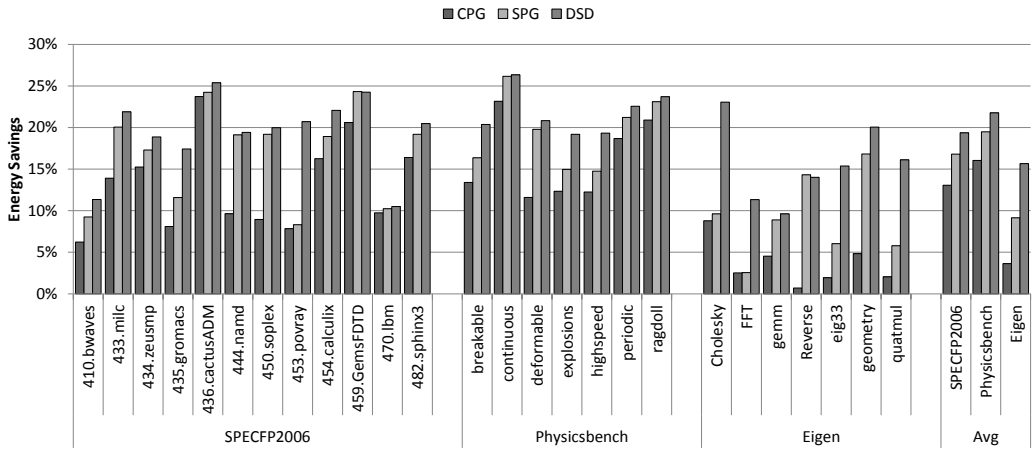


Figure 7 Core overall (dynamic + leakage) energy savings for CPG, SPG and DSD.

5.5 Performance

As mentioned earlier, power gating has both energy and performance overhead associated with it. The performance overhead arises because the functional unit cannot be used immediately after sending the wakeup signal. Moreover, the performance penalty has to be paid every time the functional unit is awakened from the power gated state.

Reducing the number of power gating instances, using DSD, reduces both the energy and performance overhead of power gating. However, DSD also has its own performance overhead. This overhead arises because the lower SIMD lane has to execute more scalar instructions. Furthermore, profiling and devectorization of the selected superblocks also diminish performance.

In summary, DSD, on one hand, reduces power gating performance overhead. However, on the other hand, it adds its own overhead. Therefore the overall performance depends on the following factors:

(1) Speedup, due to fewer power gating instances.
(2) Slowdown, due to more scalar instructions.
(3) Slowdown, due to profiling and devectorization overhead.

Figure 8 shows the performance results after considering all these factors. The results are normalized to SPG performance. As the figure shows, on average DSD experiences a slowdown of less than 1% for Physicsbench. Moreover, for SPECFP2006 the performance is very similar to SPG performance. It is also interesting to note that there are benchmarks like 433.milc, 450.soplex, 453.povray, 482.shpinx3 and Eigen benchmarks that experience a small speedup. The speedup comes due to fewer power gating instances and hence lesser performance overhead of power gating. The performance increase also translates to reduction in the leakage energy in the core because it is now ON for less time. On the other hand, 410.bwaves and FFT suffer slowdown of 6% and 4% respectively, due to excessive devectorization. Figure 9a and 9b show vector instruction profiles before and after devectorization respectively, for 410.bwaves. The excessive devectorization not only affects the performance but the overall energy savings also. Due to the slowdown, the leakage energy in the rest of the core increases, and hence net energy savings reduce.

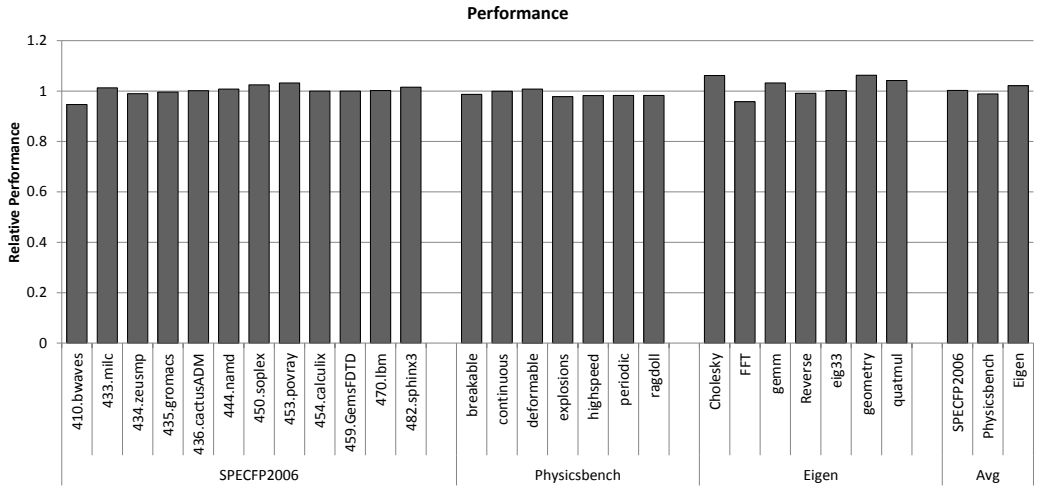
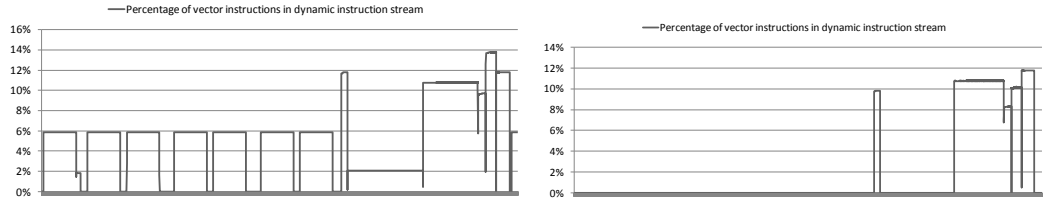


Figure 8 Overall Performance after DSD normalized to SPG (Higher is better).

The overall energy savings for 410.bwaves are approximately 50% without the energy overheads of DSD as shown in Figure 3, however after considering the energy overheads they fall down to 38% as shown in Figure 4. Therefore, DSD provides a trade-off between performance and energy.



a) Before DSD. b) After DSD.

Figure 9 Percentage of vector instructions (excluding memory instructions) in the dynamic instruction stream for 410.bwaves before and after DSD.

5.6 DSD Energy Overhead Analysis

As mentioned earlier DSD energy overhead includes two components:

1) Lower SIMD Lane Energy Overhead:

Figure 10 shows lower SIMD lane dynamic energy for DSD normalized to SPG. As the figure shows the dynamic energy for the lower lane increases by 12% for Eigen and 5% for SPECFP2006 and Physicsbench, on average. This increment in the dynamic energy consumption comes from the additional scalar code executed by the lower SIMD lane after devectorization. 410.bwaves and FFT show significant increase in lower SIMD lane dynamic energy due to excessive devectorization. The overall (dynamic + leakage) lower SIMD lane energy is shown in Figure 11. On average, the overall lower SIMD lane energy reduces for Eigen by 1% and increases only by 1% and 2.5% for SPECFP2006 and Physicsbench respectively. This energy increment is lower than the dynamic energy increase of Figure 10. There are two reasons behind it:

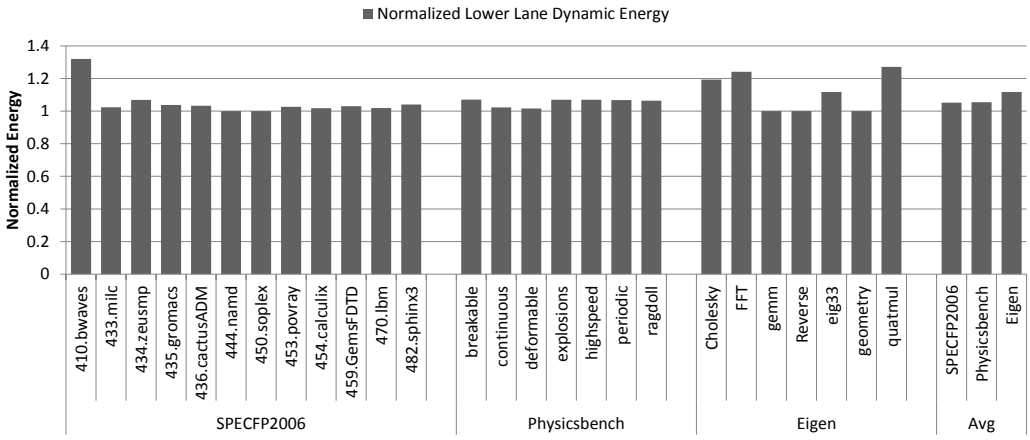


Figure 10 Lower SIMD lane **dynamic** energy for DSD normalized to SPG.

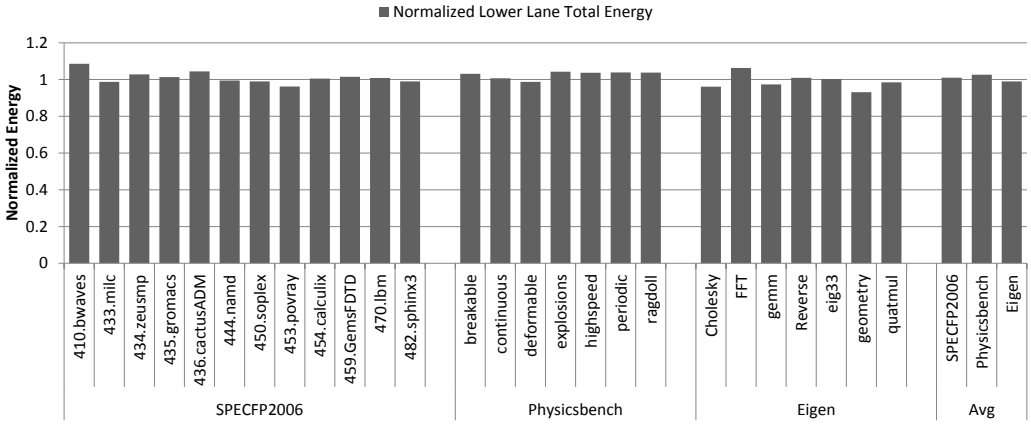


Figure 11 Lower SIMD lane **overall** energy for DSD normalized to SPG.

a) As shown in Figure 8, the performance for some benchmarks increases after DSD over SPG. As a result, the leakage energy of lower SIMD lane reduces and this compensate for the increase in dynamic energy.
b) The lower SIMD lane leakage energy accounts for 80% of its overall energy consumption. Hence, the reduction in leakage energy has more weight over the increase in dynamic energy.

These two factors make the overall lower SIMD lane energy overhead to be less than the increase in dynamic energy. Moreover, as shown in Figure 11 the lower SIMD lane energy overhead of DSD is not significant.

2) Rest of the Core Energy Overhead:

The rest of the core energy includes the overall energy of all the components except for the SIMD accelerator. The dynamic energy component here increases due to extra energy spent in profiling and devectorization. However, the leakage energy component may increase or decrease depending upon slowdown or speedup experienced after DSD. Figure 12 shows the increase in dynamic energy. On average, the dynamic energy increases by 1.2%, 1.5% and 1% for SPECFP2006, Physicsbench

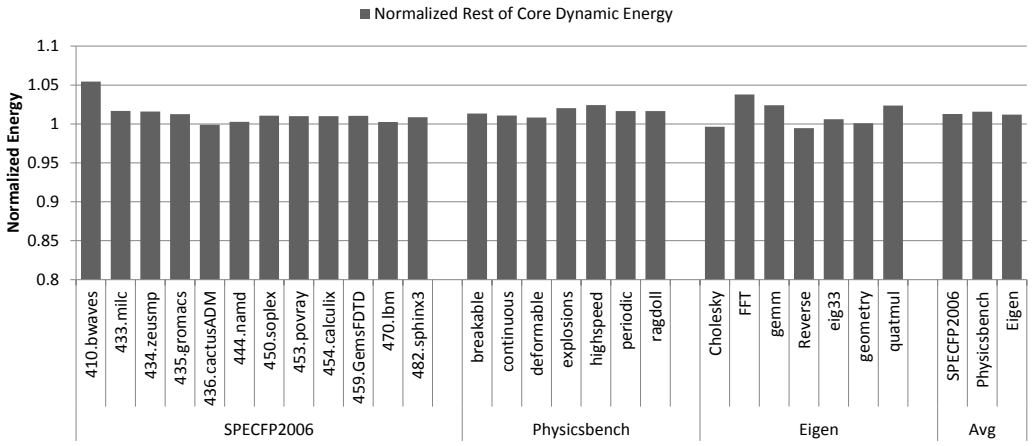


Figure 12 Rest of the core **dynamic** energy for DSD normalized to SPG.

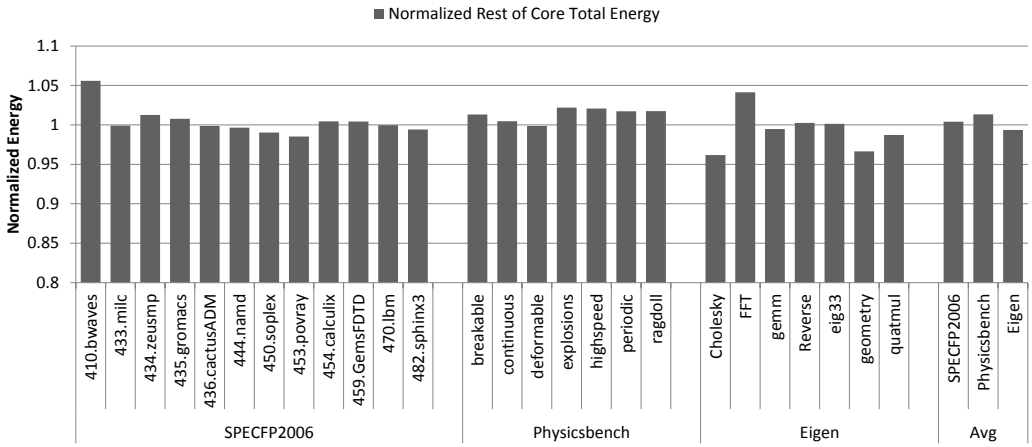


Figure 13 Rest of the core **overall** energy for DSD normalized to SPG.

and Eigen respectively. However, the overall energy increases only by 0.4% and 1.3% for SPECFP2006 and Physicsbench. On average, Eigen benchmarks show overall energy reduction of 1% as shown in Figure 13. Therefore, the rest of the core energy overhead of DSD is also negligible.

5.7 Sensitivity Analysis

As mentioned in Section 5.1, we assumed a breakeven threshold of 150 cycles and wakeup delay of 10 cycles in our experiments. These two parameters are technology dependent and to discover the effect of variations in their values we do a sensitivity study. For this study, first we vary the breakeven threshold from 20 cycles to 300 cycles while keeping the wakeup delay at 10 cycles. Next we vary the wakeup delay from 5 to 35 cycles while keeping the breakeven threshold at 150 cycles.

1) Breakeven threshold sensitivity study:

Figure 14 shows the results for the effect of breakeven threshold variations on the overall energy savings of DSD over SPG. As the figure shows, the overall energy savings of DSD, across different breakeven thresholds, are similar. However, as mentioned in Section 3, one of the components of DSD energy savings is directly proportional to the breakeven threshold. Therefore, one would expect the more

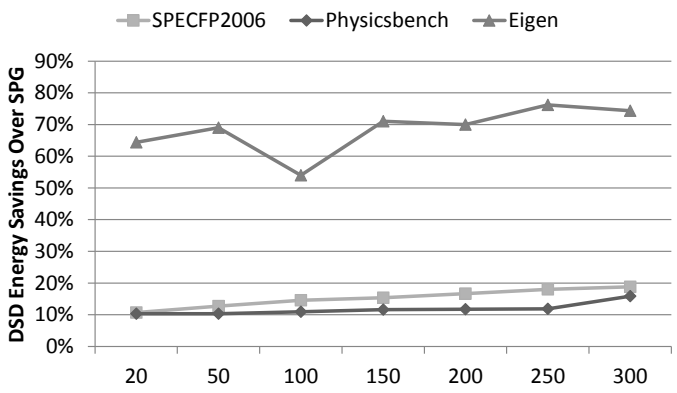


Figure 14 Effect of breakeven threshold variation on DSD overall (dynamic + leakage) energy savings over SPG with a fixed wakeup latency of 10 cycles.

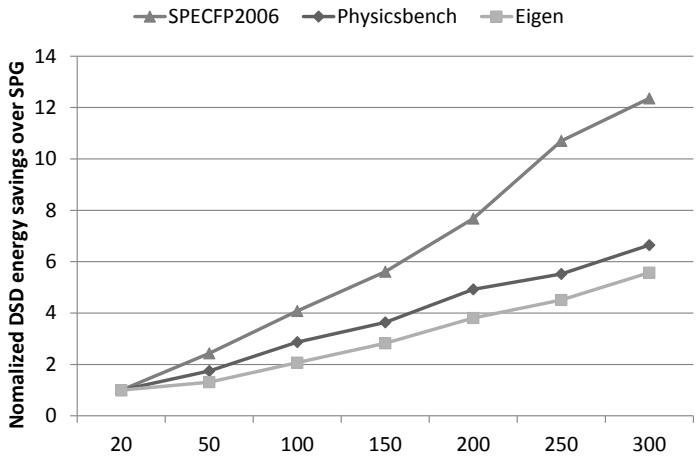


Figure 15 Effect of breakeven threshold variation on DSD overall (dynamic + leakage) energy savings over SPG normalized to breakeven threshold of 20 cycles, with a fixed wakeup latency of 10 cycles (no success monitors, no dynamic idle detect interval).

energy savings as the breakeven threshold is increased. The reason for no (or minimal) improvement in the overall energy saving is the use of success monitors and dynamic idle detect interval. If we disable these two improvements, we get more energy saving as breakeven threshold in increased as shown in Figure 15. The figure shows energy benefits of DSD over SPG normalized to the savings corresponding to breakeven threshold of 20 cycles. As the figures shows the energy savings of DSD increases over SPG as the breakeven threshold increases from 20 cycles to higher values.

2) Wakeup Delay sensitivity study:

Figure 16 shows the effect of wakeup delay variation on the overall energy savings of DSD over SPG. As with breakeven threshold variation results, these results are consistent over the range of wakeup delay values. Furthermore, these results are with success monitors and dynamic idle detect interval enabled. Disabling these two features will show improvement in DSD overall energy savings as wakeup delay increases.

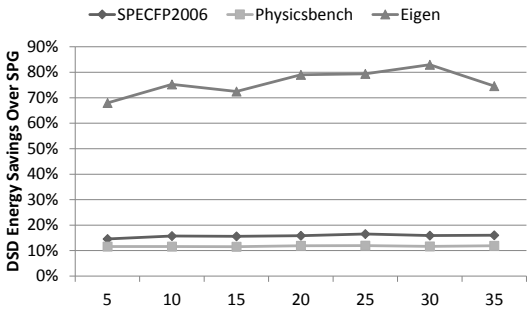


Figure 16 Effect of wakeup delay variation on DSD overall (dynamic + leakage) energy savings over SPG with a fixed breakeven threshold of 150 cycles.

Discussion:

In our experiments, we consider power gating as the leakage control mechanism implemented in the hardware. However, our proposal of dynamic selective devectorization does not restrict the choice of leakage control mechanism to power gating. DSD will work with any other leakage control mechanism equally well. The basic idea of DSD is to increase the idle intervals of the functional units independent of the leakage control mechanism.

We presented a mechanism to increase the idle period of higher SIMD lanes to save more leakage energy. DSD devectorizes certain portions of the code to reduce the higher SIMD lanes utilization during low usage periods. Even though the work in this paper focuses on higher SIMD lanes, the basic concept can be extended to any functional unit. The only requirement is to have more than one instance of the functional unit. For example, if we have two integer units, the idle interval of the second one could be increased by executing more code on the first one. This, however, is helpful only during the low utilization period of the second unit, to reduce the performance penalty of serialization. In case of SIMD accelerator, a dynamic profiler guides the devectorizer to decide which segments of code to serialize. However, in the case of integer units, the dynamic profiler needs to guide the instruction scheduler to make serialization decisions.

Moreover, even though we considered a HW/SW co-designed environment in our experiments, our proposals are general enough to be extended to other environments as long as some support for dynamic profiling and optimizations is provided like in DBTOs.

Furthermore, in Performance Evaluation section we considered only single threaded applications for the evaluation of our proposals. However, our proposals are applicable to multithreaded applications and multi-programmed environment as well. For multithreaded applications, we can profile each thread for higher SIMD lane utilization and devectorize the code corresponding to the low utilization periods. Later, threads should be scheduled such that all the threads being executed enter the high/low SIMD lane utilization periods together. The reason for scheduling threads in this manner is following:

1) **High utilization period for "Higher SIMD lanes":** If all the threads have significant vector instructions, whole SIMD accelerator usage will be high and it should not be power gated. The proposed mechanism does not target this case.
2) **Low utilization period for "Higher SIMD lanes":** If there are only a few vector instructions, higher SIMD lanes utilization will be low and power gating

will shut down these lanes. However, every time a vector instruction is encountered (which needs higher SIMD lanes) the higher SIMD lanes need to be awakened. This results in energy overhead. The proposed mechanism will devectorize this code and as a result, the higher SIMD lanes can be kept switched off for longer time duration. Therefore, resulting in additional energy savings.

Thread scheduling should also take into account resource conflict for SIMD accelerator.

6. CONCLUSIONS

This paper proposed to increase the leakage energy savings by increasing the idle interval of the higher SIMD lanes. To increase the idle interval, we proposed a dynamic profiling based dynamic selective devectorization scheme. The dynamic profiler monitors higher SIMD lanes usage and discover the code corresponding to the low utilization period. A dynamic devectorizer then selectively devectorizes the code based upon the inputs from the profiler. The dynamic selective devectorization increases the idle interval during the low utilization period of the higher lanes. Increase in the idle period helps the leakage control mechanism to save more energy. The proposed mechanism can work with any leakage control mechanism like power gating, SSGC [Kim et al. 2010] etc. Moreover the idea of increasing idle period is general enough to be extended to other functional units as well.

Our experimental results show on average, our proposed technique´s overall energy saving are 15%, 12% and 71% greater than power gating, for SPECFP2006, Physicsbench and Eigen benchmarks respectively. Moreover the slowdown caused due to devectorization is negligible. Furthermore, our sensitivity study results show that DSD energy savings hold across different breakeven threshold and different wakeup delay values.

REFERENCES

Eigen. http://eigen.tuxfamily.org/index.php?title=Main_Page

Intel Corporation, Intel® 64 and IA-32 Architectures Software Developer´s Manual, Volume 1-3.

Standard Performance Evaluation Corporation. SPEC CPU2006 Benchmarks. URL http://www.spec.org/cpu2006/.

Kanak Agarwal, Kevin Nowka, Harmander Deogun, and Dennis Sylvester. 2006. Power Gating with Multiple Sleep Modes. In *Proceedings of the 7th International Symposium on Quality Electronic Design* (ISQED '06). IEEE Computer Society, Washington, DC, USA, 633-637.

Hari Ananthan, Chris H. Kim, and Kaushik Roy. 2004. Larger-than-vdd forward body bias in sub-0.5V nanoscale CMOS. In *Proceedings of the 2004 international symposium on Low power electronics and design* (ISLPED '04). ACM, New York, NY, USA, 8-13.

Baron, M. 2005. Cortex-A8: High speed, low power. M*icroprocessor Report*,11(14):1–6, 2005.

Calin Bira, Liviu Gugu, Radu Hobincu, Valeriu Codreanu, Lucian Petrica, and Sorin Cotofana. 2013. An Energy Effective SIMD Accelerator for Visual Pattern Matching. *In Proceedings of the Fourth International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies.* Edinburgh, Scotland, 13-14 June 2013.

Aleksandar Branković, Kyriakos Stavrou, Enric Gibert, and Antonio González. 2014. Warm-Up Simulation Methodology for HW/SW Co-Designed Processors. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization* (CGO '14). ACM, New York, NY, USA, Pages 284, 11 pages.

James C. Dehnert, Brian K. Grant, John P. Banning, Richard Johnson, Thomas Kistler, Alexander Klaiber, and Jim Mattson. 2003. The Transmeta Code Morphing™ Software: using speculation, recovery, and adaptive retranslation to address real-life challenges. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization* (CGO '03). IEEE Computer Society, Washington, DC, USA, 15-24.

Paul D´Arcy and Scott Beach. 1999, StarCore SC140: A New DSP Architecture for Portable Devices. In *Wireless Symposium*. Motorola, September 1999.

Keith Diefendorff, Pradeep K. Dubey, Ron Hochsprung, and Hunter Scales. 2000. AltiVec Extension to PowerPC Accelerates Media Processing. *IEEE Micro* 20, 2 (March 2000), 85-95.

Kemal Ebcioğlu and Erik R. Altman. 1997. DAISY: dynamic compilation for 100% architectural compatibility. In *Proceedings of the 24th annual international symposium on Computer architecture* (ISCA '97). ACM, New York, NY, USA, 26-37.

Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. 2004. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 international symposium on Low power electronics and design* (ISLPED '04). ACM, New York, NY, USA, 32-37.

J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. 2005. Introduction to the cell multiprocessor. *IBM J. Res. Dev.* 49, 4/5 (July 2005), 589-604.

Hyung-Ock Kim, Bong Hyun Lee, Jong-Tae Kim, Jung Yun Choi, Kyu-Myung Choi, and Youngsoo Shin. 2010. Supply switching with ground collapse for low-leakage register files in 65-nm CMOS. *IEEE Trans. Very Large Scale Integr. Syst.* 18, 3 (March 2010), 505-509.

Rakesh Kumar, Alejandro Martínez, and Antonio González. "Dynamic Selective Devectorization for Efficient Power Gating of SIMD Units in a HW/SW Co-Designed Environment," *25th International Symposium onComputer Architecture and High Performance Computing (SBAC-PAD), 2013*, vol., no., pp.81,88, 23-26 Oct. 2013.

Ruby B. Lee. 1996. Subword Parallelism with MAX-2. *IEEE Micro* 16, 4 (August 1996), 51-59.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO 42). ACM, New York, NY, USA, 469-480.

Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation* (PLDI '05). ACM, New York, NY, USA, 190-200.

Anita Lungu, Pradip Bose, Alper Buyuktosunoglu, and Daniel J. Sorin. 2009. Dynamic power gating with

quality guarantees. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design* (ISLPED '09). ACM, New York, NY, USA, 377-382.

Marc Lupon, Enric Gibert, Grigorios Magklis, Sridhar Samudrala, Raúl Martínez, Kyriakos Stavrou, and David R. Ditzel. 2014. Speculative hardware/software co-designed floating-point multiply-add fusion. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems* (ASPLOS '14). ACM, New York, NY, USA, 623-638.

Naveen Neelakantam, David R. Ditzel, and Craig Zilles. 2010. A real system evaluation of hardware atomicity for software speculation. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems* (ASPLOS XV). ACM, New York, NY, USA, 29-38.

Demos Pavlou, Aleksandar Brankovic, Rakesh Kumar, Maria Gregori, Kyriakos Stavrou, Enric Gibert, and Antonio Gonzalez. 2011. DARCO: Infrastructure for Research on HW/SW co-designed Virtual Machines. In Proceedings of the 4th Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT'11), held in conjuction with the 38th International Symposium on Computer Architecture (ISCA-38), San Jose, California, USA, June 4, 2011.http://arco.e.ac.upc.edu/wiki/images/d/df/Pavlou_amasbt11.pdf

Demos Pavlou, Enric Gibert, Fernando Latorre, and Antonio Gonzalez. 2012. DDGacc: boosting dynamic DDG-based binary optimizations through specialized hardware support. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments* (VEE '12). ACM, New York, NY, USA, 159-168.

Sumedh Sathaye, Paul Ledak, Jay Leblanc, Stephen Kosonocky, Michael Gschwind, Jason Fritts, Arthur Bright, Erik Altman, and Craig Agricola. BOA: Targeting multi-gigahertz with binary translation. In *Proc. of the 1999 Workshop on Binary Translation, IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pages 2–11, 1999.

Manu Sporny, Gray Carper, and Jonathan Turner. 2002. The Playstation 2 Linux Kit Handbook.

James W. Tschanz, Siva G. Narendra, Yibin Ye, Bradley A. Bloechel, Shekhar Borkar, and Vivek De. "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *IEEE Journal of Solid-State Circuits,* vol.38, no.11, pp.1838,1845, Nov. 2003.

Cheng Wang, Marcelo Cintra, and Youfeng Wu. 2013. Acceldroid: Co-designed acceleration of Android bytecode. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (CGO '13). IEEE Computer Society, Washington, DC, USA, 1-10.

Yibin Ye, Shekhar Borkar, and Vivek De. "A new technique for standby leakage reduction in high-performance circuits," *1998 Symposium on VLSI Circuits, 1998. Digest of Technical Papers*, vol., no., pp.40,41, 11-13 June 1998

Thomas Y. Yeh, Petros Faloutsos, Sanjay J. Patel, and Glenn Reinman. 2007. ParallAX: an architecture for real-time physics. In *Proceedings of the 34th annual international symposium on Computer architecture* (ISCA '07). ACM, New York, NY, USA, 232-243.

Ahmed Youssef, Mohab Anis, and Mohamed Elmasry. 2006. Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO 39). IEEE Computer Society, Washington, DC, USA, 371-384.