Exploiting Data-Dependent Slack Using Dynamic Multi-V_{DD} to Minimize Energy Consumption in Datapath Circuits*

Kaushal R. Gandhi and Nihar R. Mahapatra

Department of Electrical & Computer Engineering
Michigan State University, East Lansing, MI 48824-1226, U.S.A.
E-mail: {gandhika, nrm}@egr.msu.edu

Abstract

Modern microprocessors feature wide datapaths to support large on-chip memory and to enable computation on large-magnitude operands. With device scaling and rising clock frequencies, energy consumption and power density have become critical concerns, especially in datapath circuits. Datapaths are typically designed to optimize delay for worst-case operands. However, such operands rarely occur; the most frequently occurring input operand words (comprising long strings or subwords of 0's and 1's) present two major opportunities for energy optimization: (1) avoiding unnecessary computation involving such "special" input operand subword values and (2) exploiting timing slack in circuits (designed to accommodate worst-case inputs) arising due to such values. Previous techniques have exploited only one or the other of these factors, but not both simultaneously. Our new technique, dynamic multi-V_{DD}, which is capable of dynamically switching between supply voltages in hardware submodules, simultaneously exploits both factors. Using the computation bypass framework and multiple supply voltages, we estimate data-dependent slack based on submodules that will be bypassed and exploit this slack by operating active submodules at a lower supply voltage. Our analysis of SPEC CPU2K benchmarks shows energy savings of up to 55% (and 46.53% on average) in functional units with minimal performance overheads.

1. Introduction

The increasing use of wider datapaths in microprocessors has helped support growing memory requirements of programs and computation on large-magnitude operand values to improve processor performance. With unprecedented levels of device integration due to technology scaling and design and application trends moving towards multiple cores on a single chip, the number of components per chip, and hence total on-chip capacitance, has risen steadily. Further, increasing clock frequencies lead to higher switching activity in these components, especially in functional units, which as a result have power densities up to twenty times higher than that of large second-level caches [10, 2]. Datapath circuits consume up to one-third of the total power and constitute some of the most active components on-chip [15]. Static power has also been steadily increasing, both during sleep and active modes—sub-threshold and gate leakage are increasing due to decreasing device threshold voltage (V_T) and gate-oxide thickness, respectively. Thus, it has become necessary to curb energy consumption in datapath circuits.

Operand values that typically occur during a program run are not uniformly distributed across the broad range of values supported. Most of them consist of leading 1's or 0's or have subwords within them consisting of strings of 1's or 0's [4, 13, 12]. The use of wider datapaths thus presents opportunities to reduce energy consumption in datapath circuits by: (1) avoiding unnecessary computation involving such "special" input operand subword values and (2) exploiting *data-dependent timing slack* arising due to these values in circuits designed for worst-case inputs. Previous techniques have focused on only one or the other of these two factors, whereas we exploit both factors simultaneously to effectively curtail datapath energy consumption.

On the one hand, computation bypass techniques have focused on reducing switching activity in functional subunits by avoiding computation in them on inputs involving *narrow operands* (operands that can be represented in 32, 16, or fewer bits) [3, 8, 9, 1] or non-narrow operands [12]. Computation in a functional subunit involving such "exploitable" input subwords is bypassed by either clock gating its inputs and/or power/ground gating the subunit's constituent logic gates to save energy, while its output is determined through simpler, faster, lower-power means. These exploitable input subwords are encoded for easy identification and exploitation in hardware using additional encoding bits [12]. On the other hand, low-power techniques such as static multi-V_{DD}, dynamic voltage scaling (DVS), and Razor exploit circuit timing slack, but to different extents. Static multi-V_{DD} operates gates in a subset of the non-critical paths in a circuit at a lower supply voltage without affecting its critical path delay [16]. It only exploits slack available in the worst case determined during design time. DVS lowers both clock frequency and supply voltage during phases of low throughput operation in a system [6]. This technique relies on complicated control circuitry and voltage regulation to achieve energy savings. Both static multi-V_{DD} and DVS do not exploit data-dependent slack. In Razor DVS, only the supply voltage is lowered to save energy [11]. The technique is designed such that any error that occurs as a consequence of lowering the supply voltage is detected after the computation completes and a recovery mechanism is triggered. It thus exploits data-dependent slack. The supply voltage is regulated using a control system that employs a voltage regulator and maintains a tolerable error rate. However, any change in the supply voltage incurs a very long delay penalty ($\sim$ 10's of μs) [11]. Therefore, it cannot initiate a change in supply voltage on a per-operation basis.

*This research was supported by US National Science Foundation grant # 0102830.

3-9810801-0-6/DATE06 © 2006 EDAA

Our technique, *dynamic multi-V_{DD}*, which is capable of dynamically switching between supply voltages in functional subunits on a per-operation basis, uses the computation bypass framework and multiple supply voltages to *simultaneously* exploit both operand values and data-dependent slack for energy efficiency in datapath circuits. Since computation is bypassed in functional subunits, the delay originally incurred by these subunits contributes towards slack. This slack is exploited by operating active subunits at a lower supply voltage such that the critical path delay is unaffected. Two kinds of data-dependent slack are exploited: the slack that existed in the original circuit but that the computation bypass framework helps easily identify, and new slack created because of computation bypass; more details are in Sec. 4. We estimate data-dependent slack based on the number of bypassed functional subunits, and operate the active ones at a lower supply voltage if enough slack is available. Since we employ discrete supply voltages, which are available to each submodule, switching between supply voltages is possible on a per-operation basis. Combining both computation bypass and data-dependent slack exploitation yields energy savings of 55% maximum and 46.53% average in functional units, while our encoding scheme for operands provides 27.45% and 17% energy savings in the ALU result bus and pipeline registers, respectively.

In the remainder of the paper, we describe our simulation methodology in Section 2, followed by an overview of the computation bypass framework in Section 3. We then present our new design technique, dynamic multi-V_{DD}, and discuss how to optimize it in Section 4. Following that, we provide results and discuss related work in Section 5, and finally conclude in Section 6.

2. Simulation methodology

We used SimpleScalar to simulate a processor system similar to the Alpha 21264 [7]. The `sim-outorder` simulator was modified to analyze input operands just before they were operated on in the execute stage. Details of the simulation setup, architectural configuration, and inputs are presented in Table 1. The integer computational modules (ripple-carry adder, carry-lookahead tree adder, comparator, array multiplier, and bit-wise logical operation functional units) were designed as static CMOS circuits using Cadence design tools and simulated using the Spectre simulator. The designs were implemented in 0.18 μm technology.

For estimating energy savings, we used an energy model similar to the one presented in [12]. This model estimates energy consumption based on input patterns and also distinguishes between consumption at different bit positions of a module. We employ this model to estimate energy savings in bypassed functional subunits. For the remaining (active) functional subunits, we use a similar model but with modified functional subunits that have PMOS header transistors to select between different supply voltages (see Fig. 1(a)).

3. Computation bypass framework

We now discuss the framework necessary to facilitate operation bypass in functional units. Input operand words are partitioned into g contiguous subwords (SWs), numbered 0 through $g-1$ from least to most significant. SW values are characterized as either *special* (i.e., SW bits are either all 0 or all 1) or *regular* (all other values). Each SW is encoded using an extra bit called the *special value (SV) bit* to indicate whether all its bits are identical (SV = 1 for special) or not (SV = 0 for regular). The least significant bit (LSB) of a SW along with its SV bit can be used to determine a special valued SW.

Table 1: Processor configuration to trace operand values.

Category	Details
Benchmarks	All integer and floating point SPEC CPU2000 benchmarks
Inputs	Reference inputs for each benchmark
Instruction count	Simulation window of 100 million committed instructions based on single standard simulation points from SimPoint [14]
Processor core	RUU size: 16 instructions, LSQ size: 8 entries, Fetch queue size: 4 inst/cycle, Fetch width: 4 inst/cycle, Decode width: 4 inst/cycle, Issue width: 4 inst/cycle, Functional units: 4 integer ALUs, 1 integer multiply unit
Branch prediction	2048-entry bimodal predictor
L1 instruction cache	512 KB, 32 byte blocks, direct-mapped
L1 data cache	128 KB, 32 byte blocks, 4-way set associative
L2 cache	Unified, 1024 KB, 64 byte blocks, 4-way set associative

A functional unit (FU) is similarly partitioned into *functional subunits* (FSUs) that operate on the input operand SWs, and possibly outputs from other FSUs, to produce output SWs that can be combined to form the FU's output. When an FSU has certain combinations of special input SW values or special and regular input SW values (described later in this section), its operation can be either fully or mostly bypassed, i.e., the FSU operates in *bypass mode*; in this case, the FSU is put in a low power state (e.g., by clock gating or power and ground gating it) and its output is determined via some alternative, faster, lower-power means. For example, the output SW of a bypassed FSU can be formed by either extending its least significant bit output to the remaining output SW bits or by copying one of the input SWs to the output. Such input SW combinations for which the output can be generated in a simpler manner (i.e., for which an FSU is bypassed) are referred to as *exploitable*. Otherwise, the FSU operates normally (i.e., in *active mode*).

Our operand encoding scheme differs from that presented in [12] in the following way. A single additional *status bit* indicates whether an operand word is encoded or not. An operand word is encoded only if its most significant SW is special valued (97-99.9% of encountered values, see Fig. 1d). In the encoded format, the status bit takes a value of 1 and all the bits in the most significant SW other than its LSB are replaced by SV bits of other SWs. If the most significant SW is not special valued, the status bit will be 0 and all the operand SWs will be treated as regular SWs. Hence, the encoding overhead is minimized to 1-bit per operand word (1.56% for 64-bit operands for any number of partitions as long as all the SV bits of remaining SWs can be accommodated in the most significant SW), whereas previous approaches, which use one bit per SW, incur 12% overhead for byte-wise partitioning [8].

Exploitable SW combinations are similar to those identified in [12] and are classified as follows. Class I: combinations of special SW values with special SW values; Class II: class I combinations plus combinations of special SW

values with special or regular SW values but depending on input from adjacent FSU; Class III: class II combinations plus combinations of special SW values with special or regular SW values and independent of input from adjacent FSU; and Class IV: class III combinations plus combinations of sub-subwords (SSWs)—SWs are partitioned further into SSWs to increase bypass opportunities within a SW when a SW combination does not trigger a bypass mode from class III. Results for bypass opportunities in integer FUs are plotted in Fig. 1(d) for SPEC CPU2K benchmarks and all the four classes of SW combinations mentioned above. Without loss of generality, we perform our analysis on byte-wise partitions. It can be extended to any other kind of partitioning that a user may prefer. Next, we discuss our new technique and how it can be incorporated into the framework discussed above.

4. Dynamic multi-V_{DD}

Since computation bypass exhausts opportunities to further reduce energy in bypassed FSUs, we target active FSUs for further energy savings. Active FSUs are more likely to be found at low-order bit positions, where the per-bit energy consumption is also higher [12]. Dynamic multi-V_{DD} operates such active FSUs at a lower supply voltage whenever sufficient data-dependent slack is available to ensure critical path delay is not affected. At the same time, computation is bypassed in inactive FSUs. Two kinds of slack are exploited in dynamic multi-V_{DD}: (a) Slack that already existed in the original FU (without any low-power technique) but that the encoding bits of input SWs helps easily identify, e.g., when the input SWs to an FSU in two consecutive cycles of operation are identical exploitable SV combinations and thus its output SW remains unchanged. For instance, output changes in address arithmetic operations are restricted mostly to low-order bits. This existing slack was neither identified nor exploited by previous computation bypass techniques [3, 8, 9, 1, 12]. (b) New slack that is created because of computation bypass, e.g., when input SW combinations from classes II, III, and IV occur and that result in FSU output changes; in such cases, the slack occurs because an FSU is bypassed.

We primarily consider static CMOS circuits for the following reasons. First, static CMOS is well-suited for continued operation during supply voltage transitions in the same clock cycle, while this is not so for dynamic CMOS logic [5]. Second, due to technology scaling, dynamic logic circuits, which rely on the storage of charge on a precharged gate output capacitance, become increasingly susceptible to soft errors because of leakage, and hence the popularity of static CMOS circuits is expected to grow in the future.

4.1. Switching between supply voltages

There are two possible ways in which multiple supply voltages can be provided on-chip. They can be created on-chip using dc-dc converters. This method is not suitable for our approach since changing the voltage level incurs delays of up to 10 μs [11] and we require a change during the cycle of operation itself. Alternatively, a dual supply network can be used to provide fast switching between supply voltages. Header PMOS transistors with complementary control signals can be used to select between multiple supply voltages (see Fig. 1(a)). The control signals are determined before each cycle of computation commences. Modern ASICs and microprocessors already employ multiple on-chip supply networks and so we assume that multiple supply voltages are available. Note that, unlike DVS, clock frequency is not changed while operating at a lower voltage. This is because the available data-dependent slack is used to accommodate both the delay overheads due to switching between multiple supply voltages, which is a small portion, and the slower operation of FSUs due to this change.

4.2. Data-dependent slack

We implemented FUs as static CMOS circuits and considered different topologies (linear, array, and tree) because data-dependent slack depends upon hardware topology, and analysis of other similar topologies can be performed by a simple extension of our analysis. Computation in a linear topology proceeds in a sequential manner. For example, in a ripple-carry adder, the carry bit propagates from one full-adder to the next in increasing order of bit position. If computation in an FSU is bypassed, then the delay of that FSU, which it would have otherwise incurred, contributes towards slack. An array topology is very similar to a linear topology, the only difference being that the computation propagates along two dimensions. Therefore, the amount of slack available in both these topologies is directly proportional to the number of bypassed FSUs. But this is not so for a tree topology since FSUs at the same level in the tree operate in parallel. Here, the slack depends on the number of levels within the tree where computation can be bypassed.

We assume that active FSUs require their worst-case delays and thus estimate slack conservatively. For linear and array topologies, slack is estimated based on number of bypassed FSUs. For tree topologies, the number of bypassed FSUs and the number of levels within the tree where computation can be avoided is used to determine the amount of slack. In reality, we underestimate slack because an active FSU may take much less time to compute its output compared to its worst-case delay; we defer this analysis of estimating delay much more accurately to future work.

To study the variation of delay of FSUs w.r.t. hardware topology, we analyzed the three topologies under consideration (see Fig. 2(b)). Array topologies behave just like linear topologies; for brevity, we only show plots for linear and tree topologies. In a linear topology, the maximum bit-width of a submodule operating at a lower supply voltage that has a delay less than or equal to a 64-bit module operating at a higher supply voltage is 32 bits (compare curves V_DDH to V_DDL (linear) and V_DDH (linear) in Fig. 2(b)). Whereas, for a tree topology, it is only 8 bits (compare curves V_DDH to V_DDL (tree) and V_DDH (tree) in Fig. 2(b)). Therefore, when exploiting slack in a linear structure, we must ensure that the total bit-width of all bypassed FSUs is at least 32 bits. For a tree topology, we must ensure that all active FSUs complete their operation such that the total delay of computing the output is less than or equal to that of an 8-bit FSU operating at a lower supply voltage.

Fig. 2(a) shows the fraction of FSUs for 64-bit byte-wise partitioned FUs that can be bypassed based on the ex-

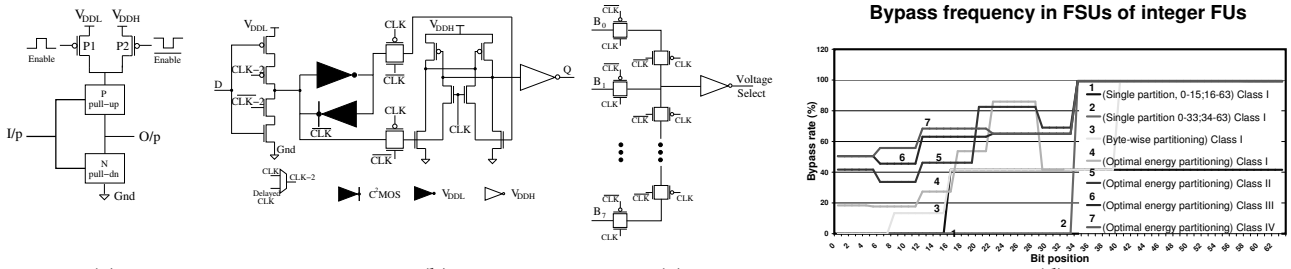

Figure 1: **(a) Multiple supply voltages:** PMOS header transistors P1 and P2 are used to select between two supply voltages. **(b) Level-converting register:** The master stage has the capability to sample outputs at two different times (using CLK and delayed CLK). The active clock is selected using a multiplexor and the slave stage performs level conversion. **(c) Voltage-level selection circuit:** Submodule bypass bits ($B_0, \ldots, B_7$) are selectively shorted through multiplexors to provide the input to an inverter. The charge stored on the input gate capacitance of the inverter is proportional to the number of bypassed modules and the output of the inverter is used as a control signal to select between supply voltages as shown in (a). **(d) Bypass opportunities:** Bypass frequencies for integer FUs based on various partitioning schemes and exploitable SW combinations.

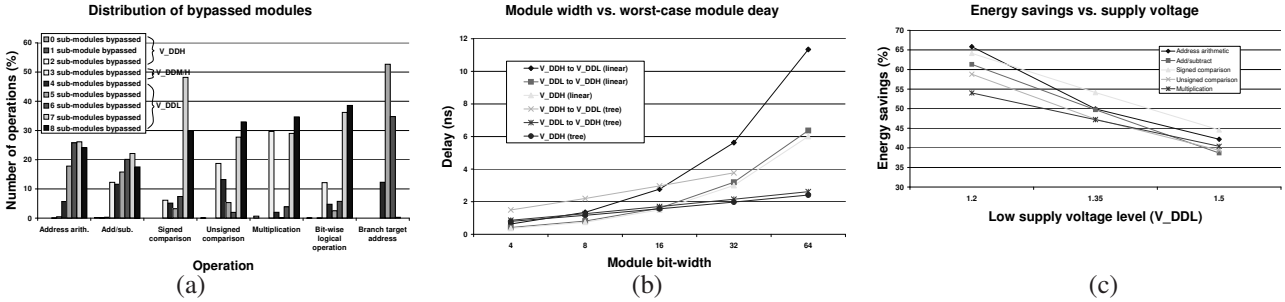

Figure 2: **(a) Availability of data-dependent slack:** Distribution of number of bypassed submodules per operation. **(b) Delay variation:** Worst case delay values for supply voltage transitions for linear (ripple-carry adder) topology and tree (carry-lookahead tree adder) topology modules of different bit-widths for 1.8 V (V_{DDH}) and 1.2 V (V_{DDL}).**(c) Energy savings versus supply voltage levels:** Energy savings in a computational module decrease with increasing supply voltage. A higher supply voltage provides smaller savings per operation although the opportunities to exploit slack may increase.

ploitable SW combinations discussed in Sec. 3. For address arithmetic, we find that four or more FSUs can be bypassed 99% of the time. Further, all eight FSUs can be bypassed 20% of the time because the base address has its least significant byte equal to 0, while the offset only has 8 significant bits. In contrast to Razor, where the slack that can be exploited is restricted to half a clock cycle, we can exploit slack close to the complete clock cycle. For other operations, such as comparison, multiplication, and bit-wise logical operations, four or more FSUs can be bypassed 68-97% of the time.

4.3. Microarchitectural design issues

Estimating data-dependent slack: Data-dependent slack is estimated based on the number of bypassed FSUs for linear and array topologies. For tree topologies (carry-lookahead tree adder), we estimate slack based on whether each byte-slice is capable of operating independently. Conditions governing parallel operation of byte-slices for other FUs implemented as tree topologies can also be identified by inspecting the input bits to each byte slice. The supply voltage level to an FSU must be chosen before computation commences. This decision must be made during the issue stage for data capture schedulers, i.e., schedulers that store data values, or after register file read following the issue stage in non-data capture schedulers. In non-data capture schedulers, registers are read during the first half of a clock cycle and written to during the second half of a clock cycle. Therefore, the delay of control circuitry to determine which FSUs can be bypassed and the voltage level to be selected can be masked. For designs that use data-capture schedulers, delay overhead due to the control circuitry can be offset by a portion of the available data-dependent slack.

Selecting FSU supply voltage levels: We use an analog counter circuit to determine the number of bypassed FSUs (see Fig. 1(c)). For tree topologies, it estimates whether the active FSUs can operate in parallel and complete their operation before the end of the clock cycle when operated at a lower supply voltage. The bypass bits for the FSUs, $B_0, \ldots, B_7$ (= 1 if an FSU is bypassed and 0, otherwise), initially charge/discharge the input capacitances of transmission gates. The bypass bits are then isolated from these capacitances, which are shorted and undergo charge sharing. The final voltage on these capacitances is input to an inverter. If sufficient number of FSUs are bypassed, the input to the inverter is "high" and its output is "low," which causes a low voltage level to be selected (see Fig. 1(a)). The PMOS and NMOS transistors of the inverter can be sized to set its switching threshold so that the lower supply voltage is selected only if a predetermined number of FSUs are bypassed, or equivalently, only if a predetermined amount of charge is stored on the input gate capacitance. The output of the inverter is then used to select the supply voltage for active FSUs (Fig. 1(a)). A digital counterpart would incur

several gate delays resulting in a longer delay penalty, and hence our choice of an analog circuit.

Level conversion of outputs: FSUs operating at a lower supply voltage require voltage conversion of their outputs for the next stage to perform correctly. For this, synchronous level converters embedded in the output pipeline registers are used. Note that no asynchronous level converters are required because, during any computation, all active FSUs operate at a single supply voltage. Also, there is an additional need for supporting sampling of outputs before the actual clock edge to mitigate performance overheads incurred for the worst-case scenario, wherein all FSUs are active and the supply voltage transitions from a low to a high level. The basic idea is to begin such a worst-case operation before its stipulated time, provided its operands are already available. However, it must be preceded by a computation that has data-dependent slack. This way, the low-latency operation is guaranteed to complete before its deadline and its output can be sampled at the output register before the active clock edge arrives. The long latency operation (which requires slightly more than a clock cycle) can begin earlier without affecting the output of the computation immediately preceding it and has enough time to complete. Fig. 1(b) illustrates a master-slave flip-flop, where a C^2MOS style logic is used. On the output side, the master stage has the ability to sample the output based on two clock signals (global clock and its delayed version), while the slave stage performs the level conversion. On the input side, the slave stage has the ability to sample at two different clock edges (global clock and its delayed version) to start the long-latency computation earlier.

Supply voltage levels: Two factors contribute towards energy savings while exploiting data-dependent slack: (1) the value of the lower supply voltage level and (2) how frequently FSUs operate at the lower supply voltage level. Operation at lower supply voltage levels requires greater slack, so that computation completes within the required time. Hence, the frequency with which the lower voltage modes can be triggered suffers. Alternatively, with higher values of the low supply voltage, less slack is required since computations would complete faster. Thus, there is a trade-off between how frequently the low voltage mode can be triggered and the value of the lower supply voltage. In order to determine the supply voltage, we obtain energy savings for different levels of the low supply voltage (see Fig. 2(c)). From this plot, we find that the most energy savings are obtained when the supply voltage is set to 1.2 V. Therefore, we use the nominal voltage of 1.8 V when data-dependent slack is not available and 1.2 V, otherwise.

4.4. Mitigating performance overheads

There are three sources of delay overheads: (i) header transistors used to switch between supply voltages during an operation add resistance to a charge path from the supply rail; (ii) the control circuitry used to determine the supply voltage level; and (iii) the pipeline registers used to convert voltage levels. We adopt the following approach to mitigate performance overheads. A computation without data-dependent slack (which is very rare) can be scheduled after one for which there is sufficient slack estimated to be available for exploitation (which is very often the case). The data-dependent slack in the latter computation is not exploited and can be passed to the former computation. Also, operands for a non-slack computation must be available much before the active clock edge (i.e., the clock edge at which new inputs are issued to an FU). This is not a problem in data-capture schedulers where operands are available in the issue queue itself. In non-data capture schedulers, operands are read from the register file during the first half of the clock cycle and therefore operands are available well before the active clock edge. Since the executing computation is the one where slack is available, its outputs are available before the end of the clock cycle and the output is buffered in the output pipeline register (see Fig. 1(b)). The master latch can sample data using one of two clock signals (the global clock and its delayed version) provided through a multiplexor. The slave latch provides the output to the next stage at the active clock edge since it is controlled by the global clock signal. In the worst case, if no other computation with slack is available, a single cycle penalty is incurred since the computation would have to extend to the next cycle; however, this will rarely be the case.

5. Results and discussions

Operand value characteristics: Our analysis of operand values shows that the three most significant bytes in all operations under consideration are found to have special values more than 97% of the time. Thus, computation in high-order FSUs is mostly bypassed and most of the switching activity is restricted to the low-order FSUs. This shows that computation bypass is possible in high-order FSUs for 97% of the operations; this also highlights the abundance of data-dependent slack that can be exploited by operating low-order FSUs at a lower supply voltage. Further, high-order FSUs are shut off for most operations and energy is primarily consumed in the low-order portion. Our technique thus targets the power-hungry portion of FUs, thereby providing significant energy savings.

Energy savings: We study energy savings for two design cases: (1) using two supply voltage levels: V_{DDH} = 1.8 V and V_{DDL} = 1.2 V and (2) using three supply voltage levels: V_{DDH} = 1.8 V, V_{DDM} = 1.5 V, and V_{DDL} = 1.2 V. We wanted to compare the energy savings for a dual supply voltage system and the additional benefit of using an intermediate supply voltage. It is only in tree topologies where we see improvements in energy savings by 5-10% because of the additional supply voltage level. In a tree topology, often the estimated slack is not sufficient to operate at V_{DDL}, and hence it loses out on opportunities when there are only two supply voltage levels. Net energy savings of up to 55%, and 46.53% on average, are possible in various FUs using two supply voltage levels. Using three supply voltage levels improves average savings to 48.3% (see Fig. 3). While computation bypass can provide energy savings close to 30%, further energy savings are only possible from active FSUs. Dynamic multi-V_{DD} is able to reduce energy consumption in active FSUs in addition to savings from computation bypass. Although the focus of the paper is on FUs, using our encoding scheme we also achieve energy savings in ALU buses and pipeline registers. Since special-valued SWs are represented

by SV bits (special-value bits), they are not written to the result bus or the pipeline latches, resulting in 21% and 27.45% self and coupling energy savings, respectively, in the integer ALU result bus, and 17% reduction in switching activity in pipeline registers.

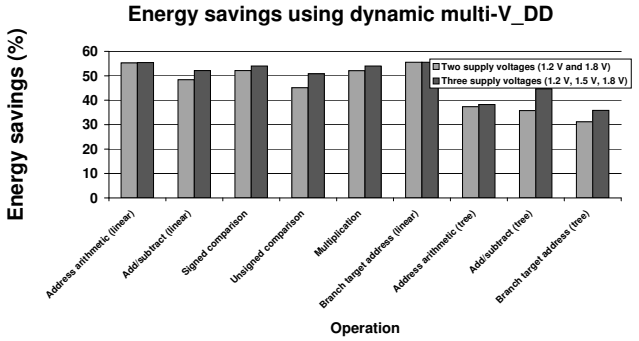

Figure 3: **Energy savings:** Energy savings in different FUs using dynamic multi-V_{DD} averaged across SPEC CPU2000 benchmarks.

Performance overhead: Recall that we only apply lower supply voltages to active FSUs when sufficient slack is available. However, as discussed in Sec. 4.4, the worst-case scenario can lead to performance overheads which can be mitigated by slack borrowing. To evaluate the performance overhead of our technique, we pessimistically assumed an additional cycle delay penalty for every computation without data-dependent slack. We find that, on average, a performance overhead of less than 1% reduction in instructions per cycle (IPC) is incurred. Note that, unlike Razor DVS, no error detection or correction measures are required and this helps us avoid expensive performance overheads.

Related work: Previously-proposed operand-value exploitation techniques focus solely on computation bypass to reduce switching activity in computational modules [3, 8, 9, 1, 12]. Our new encoding scheme for operand values has the least overhead compared to all other techniques and also helps reduce switching activity in the ALU result bus and pipeline registers. One of the most significant contributions is recognizing and availing of opportunities provided by operation bypass to exploit data-dependent slack in FUs. In contrast to static multi-V_{DD}, in which a subset of gates on non-critical paths always operate at a lower supply voltage and which exploits only the worst-case slack determined at design time, we are able to exploit data-dependent slack (which may change on a cycle-to-cycle basis) in both critical and non-critical paths. Other techniques, such as DVS, rely on frequency and voltage scaling to save energy during low throughput phases of an application. In contrast, dynamic multi-V_{DD} is not constrained by such requirements. Compared to Razor, which lowers supply voltage and then checks for error conditions, dynamic multi-V_{DD} relies on a priori estimation of data-dependent slack, thus eliminating the need for error detection and recovery. Thus our performance overheads are smaller: maximum of 1% compared to 3% incurred by Razor [11]. Further, Razor exploits slack up to only half a clock cycle, whereas our technique can exploit slack close to the complete clock cycle. Finally, Razor implicitly assumes the availability of voltage regulators to vary supply voltage and does not include the energy overheads incurred as we do while estimating net energy savings.

6. Conclusion

In this paper, we presented a design approach that utilizes the computation bypass framework to not only shutoff inactive FSUs to save energy, but that also simultaneously exploits data-dependent slack by operating active FSUs at a lower supply voltage. Net energy savings of up to 55% in FUs and 27.45-17% reduction in ALU result bus and pipeline register energy are achieved with minimal performance overheads. Datapaths in modern microprocessors contribute towards 30-33% of total chip power; therefore our technique could potentially reduce energy consumption of a chip by approximately 12% (average of 46.53% in the ALU and 27.5-17% in other datapath circuits). With designs moving towards multiple cores having more datapaths, benefits from our technique will be even greater. Compared to previous techniques, which either exploit only operand values through computation bypass or only timing characteristics, we are able to take advantage of both using dynamic multi-V_{DD}.

References

[1] A. Abddollahi, M. Pedram, F. Fallah, and I. Ghosh. Precomputation-based guarding for dynamic and leakage power reduction. In *Proc. 21st. IEEE International Conference on Computer Design (ICCD 2003), San Jose, CA*, pages 90–97, October 2003.

[2] S. Borkar. Gigascale integration - Challenges and opportunities. In *http://www.intel.com/cd/ids/developer/apac/zho/182440.htm*.

[3] D. Brooks and M. Martonosi. Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance. *ACM Transactions on Computer Systems*, 18(2):89–126, May 2000.

[4] D. Brooks, M. Martonosi, J.D. Wellman, and P. Bose. Power-performance modeling and tradeoff analysis for a high end microprocessor. In *Proceedings of the First International Workshop on Power-Aware Computer Systems (PACS'00) held with ASPLOS-IX, Cambridge, MA*, pages 126–136, November 2000.

[5] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of International symposium on Low Power Electronics and Design (ISLPED 2000), Rapallo, Italy*, pages 9–14, 2000.

[6] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences*, pages 288–297, 1995.

[7] D. Burger and T.M. Austin. The SimpleScalar Tool Set, version 2.0. *Computer Architecture News*, pages 13–25, June 1997.

[8] R. Canal, A. Gonzalez, and J. Smith. Very low power pipelines using significance compression. In *Proc. of Annual ACM/IEEE Symposium on Microarchitecture (MICRO 2000), Monterey, California,*.

[9] J. Choi, J. Jeon, and K. Choi. Power minimization of functional units by partially guarded computation. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED 2000), Rapallo, Italy*, pages 131–136, 2000.

[10] J. Deeney. Thermal modeling and measurement of large high-power silicon devices with asymmetric power distribution. In *Proc. International Symposium on Microelectronics*, 2002.

[11] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2003), San Diego, CA*, pages 7–18, 2003.

[12] K. R. Gandhi and N. R. Mahapatra. Dynamically exploiting frequent operand values for energy efficiency in integer functional units. In *Proceedings of the 18th. International Conference on VLSI Design (VLSID 2005),Kolkata, India*, pages 570–575, 2005.

[13] M. Lipasti, B. Mestan, and E. Gunadi. Physical register inlining. In *Proc. of International Symposium on Computer Architecture (ISCA 2004), Munich, Germany*, pages 325–337, 2004.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of Architectural Support for Programming Languages and Operating Systems, (ASPLOS 2002), San Jose, CA*, pages 45–57, 2002.

[15] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings of the 35th annual conference on Design automation conference (DAC 1998), San Francisco, CA*, pages 732–737, June 1998.

[16] K. Usami and M. Horowitz. Clustered voltage scaling technique for low-power design. In *International Symposium on Low Power Design*, pages 3–8, 1995.