Power Reduction in Superscalar Datapaths Through Dynamic Bit–Slice Activation *

Dmitry Ponomarev, Gurhan Kucuk, Kanad Ghose
Department of Computer Science
State University of New York, Binghamton, NY 13902–6000
e–mail:{dima, gurhan, ghose}@cs.binghamton.edu

Abstract

We show by simulating the execution of SPEC 95 benchmarks on a true hardware–level, cycle by cycle simulator for a superscalar CPU that about half of the bytes of operands flowing on the datapath, particularly the leading bytes, are all zeros. Furthermore, a significant number of the bits within the non–zero part of the data flowing on the various paths within the processor do not change from their prior value. We show how these two facts, attesting to the lack of a high level of entropy in the data streams, can be exploited to reduce power dissipation within all explicit and implicit storage components of a typical superscalar datapath such as register files, dispatch buffers, reorder buffers, as well as interconnections such as buses and direct links. Our simulation results and SPICE measurements from representative VLSI layouts show power savings of about 25% on the average over all SPEC 95 benchmarks.

1. Introduction

Contemporary superscalar datapath designs attempt to push the performance envelope by employing aggressive out–of–order instruction execution mechanisms, which entail the use of datapath artifacts such as dispatch buffers (DBs), large register files and reorder buffers (ROBs) or variants. In addition, multiple function units (FUs) and sizeable on–chip caches are frequently employed. Dispatch buffers and reorder buffers are generally implemented as multiported register files, with additional logic, such as associative addressing facilities. All of these components are implicit forms of storage within the datapath while the register files are an explicit form of storage (in addition to the on–chip caches). All of the implicit and explicit storage components in a modern superscalar datapath dissipate a considerable amount of energy [6, 11, 13].

While the absolute power requirements of high–end superscalar processors have gone up steadily over the years as increasingly higher clock rates and smaller circuit components are being used, the situation with areal energy density (i.e., energy dissipated per unit area of the die) has become the immediate, serious concern [9]. Unless energy dissipation is controlled through technology independent techniques, the areal energy density will soon become comparable to that of nuclear reactors, as shown in [9] leading to intermittent and permanent failures on the die and also creating serious challenges for the cooling facilities. Furthermore, the areal energy density distribution across a typical chip is highly skewed, being lower over the on–chip caches and significantly higher elsewhere. The non–uniform thermal stresses that result are also problematic. This paper introduces a technology independent, dynamic solution for reducing the energy dissipation in the implicit and explicit storage components in a manner that does not impede performance in any way. More pleasantly, it reduces energy dissipations over components that have higher areal energy densities, such as DBs, ROBs and register files. In addition, we introduce a related technique for reducing dissipations on the wires interconnecting these and other storage components.

Specifically, we exploit the presence of bytes containing all zeros, particularly in the higher–order bytes of operand values that are read out from physical registers, issued to function units, forwarded from function units or moved into the reorder and dispatch buffers. We avoid the activation of byte slices that contain all zeros along the interconnections and also within the implicit and explicit storage components of the processor to conserve power. The simulation results show that on the average about 50% of the bytes of operands are zeros. Furthermore, within the non–zero bytes, more than 65% of the bits are identical to what was driven immediately before on the data flow path. For the purpose of this paper, a *data stream* is a sequence of operand values from possibly different sources, that flow on an interconnection.

Exploiting the presence of bytes containing all zeros is not new. Zero bytes can be encoded to compact the data and instructions. It was suggested and used to reduce the power dissipation in a dispatch buffer in [5]. In [12], the same fact is used to reduce energy dissipations within the primary data and instruction caches for SPECint95 and other

* supported in part by DARPA through contract number FC 306020020525 under the PAC–C program, by the IEEC at SUNY–Binghamton and the NSF through award no. MIP 9504767 and EIA 9911099

integer–dominated benchmarks. In [2], the presence of leading zero bytes in a result produced by integer function units are exploited. Energy savings possible within the explicit and implicit storage components of the datapath and floating–point dominated benchmarks are not considered in [2]. In [4], a scheme for encoding significant zeros is exploited and investigated for power reduction in scalar pipelines; our study focuses on superscalar datapaths and considers many critical components like the dispatch buffer, multiple register files and reorder buffer that are not examined in [4]. In addition, compared to the scheme of [4] which requires a drastic redesign of the datapath, our power savings artifacts have a lower impact on the physical datapath/control design and as such they are more easily adaptable.

The rest of the paper is organized as follows. Section 2 describes two variations of superscalar datapaths considered in this study. Section 3 identifies the reasons for the low entropy in data streams. Relevant circuit components are shown in section 4. Section 5 discusses sources of power dissipation in modern superscalar datapaths. Our experimental methodology is presented in section 6 followed by the discussion of experimental results in section 7. Finally, we conclude in section 8.

2. Superscalar Datapath Configurations

Most contemporary microprocessors employ multiple instruction dispatching and multiple instruction issuing per cycle for performance. Instruction dispatching refers to the process of decoding instructions and noting data dependencies, while instruction issuing refers to the process of committing physical execution resources for the actual execution of the instruction once all input operands are available. Instruction dispatching and issuing are distinct steps in modern microprocessors. The dispatching step typically moves an instruction into a dispatch buffer (DB) irrespective of the availability of input operands and function units (FUs). As all input operands of an instruction waiting in the dispatch buffer become available along with a function unit of the required type, the instruction is issued to the function unit. Two variants of a superscalar datapath are in common use depending on where in the processing sequence the input operands of the instruction are accessed.

2.1. Superscalar Datapath A

Here, input registers that contain valid data are read out while the instruction is moved into the dispatch buffer (DB). As the register values required as an input by instructions waiting in the DB (and in the dispatch stage) are produced, they are forwarded through forwarding buses that run across the length of the DB [8]. The dispatch buffer entry for an instruction has one data field for each input operand, as well as an associated tag field that holds the address of the register whose value is required to fill the data field. When a function unit completes, it puts out the result produced along with the address of the destination register for this result on a forwarding bus. Comparators associated with each DB entry then match the tag values stored in the fields (for waited–on register values) against the destination register address floated on the forwarding bus [8]. On a tag match, the result floated on the bus is latched into the associated input field. Since multiple function units complete in a cycle, multiple forwarding buses are used; each input operand field within a DB entry thus uses a comparator for each forwarding bus. Examples of processors using this datapath style are the Intel Pentium Pro, Pentium II, IBM Power PC 604, 620 and the HAL SPARC 64 [7].

2.2. Superscalar Datapath B

In this datapath variation, even if input registers for an instruction contain valid data, these registers are not read out at the time of dispatch. Instead, when all the input operands of an instruction waiting in the DB are valid and a function unit of the required type is available, all of the input operands are read out from the register file (or as they are generated, using bypassing logic to forward data from latter pipeline stages) and the instruction is issued. In this case, the DB buffer entry for an instruction is considerably narrower compared to the DB entries for Datapath A, since entries do not have to hold input register values. Examples of processors using this datapath style are the MIPS 10000, 12000, the IBM Power 3, the HP PA 8000, 8500, and the DEC 21264 [1, 7].

For both Datapath A and Datapath B, an instruction can bypass the DB and can be directly issued to an available FU if the input operands are available. For our studies, we assume that a separate architectural register file is maintained for the two datapath variants; in many implementations the architectural register file and physical register file (PRF) can be merged. Furthermore, it is possible to do away with explicit physical registers and instead use ROB entries to act as placeholders for results – we have not reported results for these variants in this paper for the sake of brevity. We have also not considered variants where the DB, ROB and the PRF are all merged into integrated structures such as RUUs or DRIS (as used in the Lightning Metaflow processor, [10]), since the long wire delays of these structures make such schemes unattractive.

3. Exploiting the Lack of Data Stream Entropy

There are a variety of reasons that cause bytes throughout operands to consist of all–zeros. For integer 32–bit operands, leading bytes may be zeros due to the use

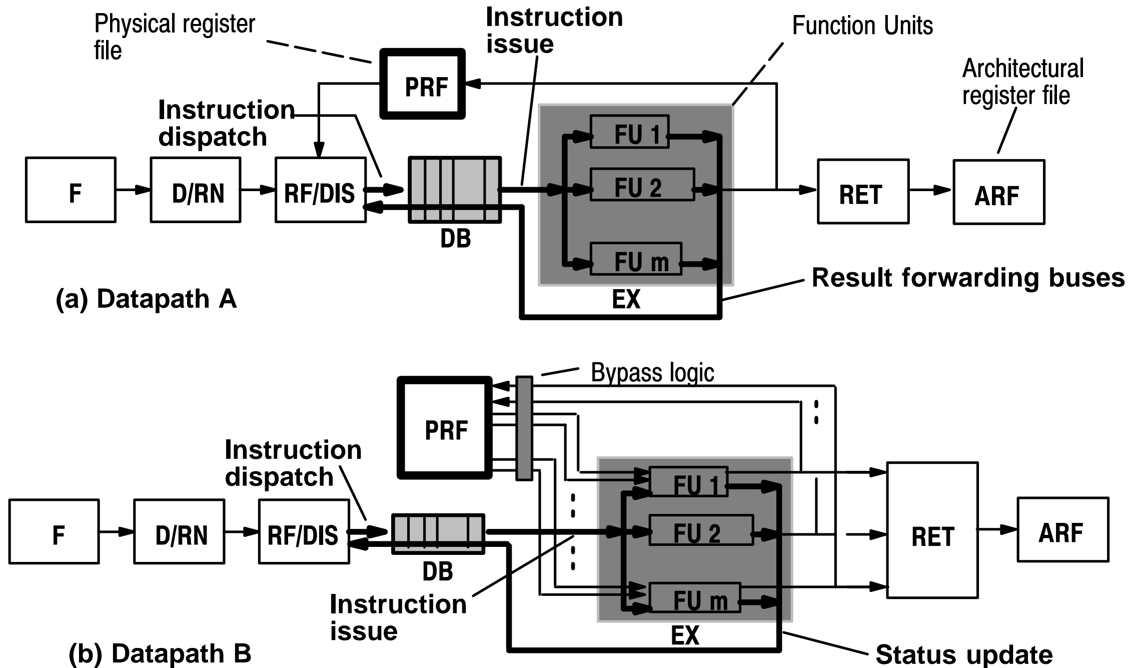


Figure 1. Two variants of a superscalar datapath

of predominantly (positive) small literal values that have only a few bits of significance. These are either literal operands for integer operations or small offsets used in branch instructions and load/store instructions. Zero bytes are also introduced in results when byte packing and unpacking operations are used or when bit or byte masks are used to isolate bits or bytes. For floating point operands in the IEEE format, small (or lower precision) floating point values may also contain zeros – but not necessarily in the leading part. Negative integer results with small absolute values can also have leading ones. Our results, however, show that the percentage of leading zero bytes or the percentage of zero bytes throughout 32–bit and 64–bit operands overwhelm the number of bytes with all ones. To simplify circuit requirements, we exploit only the bytes with all zeros. Byte–level encoding is preferred for two main reasons: it keeps encoding overhead (and decoding delays) to an acceptable level; it is also consistent with byte–addressing scheme.

This lack of entropy in data streams can be exploited in several ways. The first approach is to only drive byte slices that do not contain all zeros. Second, one can avoid driving the bit lines in transfer paths containing significant data (non–zero bytes) that were driven with the same value when the prior transfer was made on the transfer path. We have used the term *bit invariance* to allude to the fact that a bit line to be driven was driven with the same value when it was last used. Third, it is possible to avoid the reading and writing of byte slices that contain bytes with all zeros from/to explicit and implicit storage artifacts in the datapath such as register files, DB and the ROB.

4. Relevant Circuit Components

The encoding of bytes containing all zeros can be kept simple by associating a single bit (called the zero indicator, ZI) with every byte. This allows bytes with all zeros to be identified quickly with an acceptable storage overhead. Figure 2 depicts the circuitry needed to derive the ZI for a byte; such an encoder can be implemented within function units or within output latches/drivers. Two n–devices (*not shown*) are also used to discharge any charge accumulated on the gates on the two pulldown n–transistors to prevent false discharges of the output due to prior partial mismatches. Encoded data values can be moved by

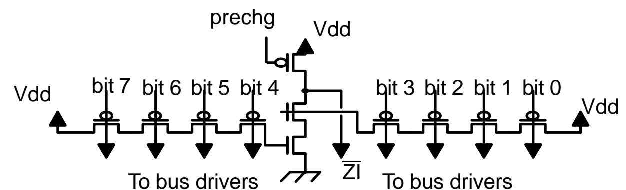


Figure 2. Encoding logic for all zero bytes

transferring the ZI bits for *every* byte within the data value, along with the bytes that do not contain all zeros. Bytes that contain all zeros are not transmitted. Values encoded as

described can be written directly to all storage components such as the register file, DB entries (for all operands for Datapath A and literal values for Datapath B), reorder buffers etc. All ZI bits are written, along with bytes that are not all zeros. The energy savings thus comes from not transferring or writing bytes that contain all zeros. To avoid reading out bytes that consist of all zeros, the physical words within register files and cache tag/data RAMs employ additional logic that uses a byte's associated ZI bit to disable the readout of bytes with all–zero, as shown in Figure 3. Obvious variations of the circuits exist when multiple ports are used. The sense amps used with the resulting RAMs and register files are also designed not to react to disabled columns by increasing the extent of the input dead zone. The power savings realized by not reading, writing or transferring bytes that contain all zeros come at a cost: additional ZI bits have to be driven, read and written. The ZI bits, by themselves, and additional metal lines increase the area of the register files and tag/data RAMs by about 12%. There is a slight increase in the cycle time of the register file because of the delay imposed by Q4 in the course of relaying the signal on the word select line to the "read" line; this increase does not jeopardize the processor's cycle time in any way for the layouts studied.

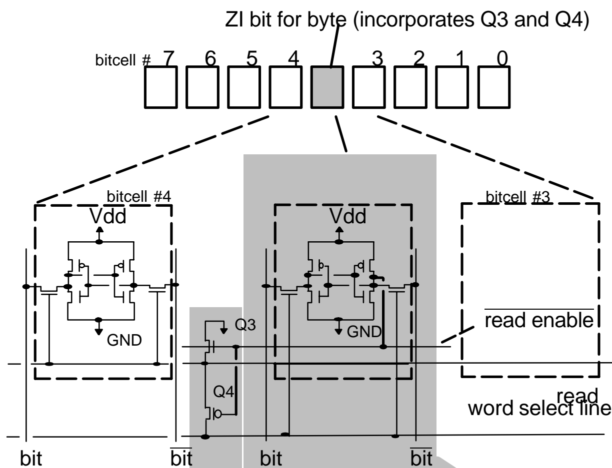


Figure 3. Bit–storage enhancements for avoiding the reading of all–zero bytes (for a single port)

Figure 4 shows a simplified equivalent circuit for driving only the bits within the non–zero bytes and ZI bits that have not changed from the prior value. The circuitry shown in Figure 4 is replicated for each bit that can be driven onto the bus/link. The latch PVj holds the prior value of the data on bit #j of the bus/link; the value of the current data to be driven on the same line is Dataj. The driver is enabled only when Dataj is not a bit within a byte field with all zeros (as indicated by a low–active ZI) and when the data bit to be driven is not the same as the prior driven value (as indicated by the output of the ex–or) and when the data has to be transferred (drive = 1). Keeper devices (not shown) are used to retain the past–driven value of bit lines till they are re–driven; such keeper devices are not shown in Figure 4. The power savings realized in this case is offset by the extra energy needed by the latches that hold the prior values and some of the associated logic. Additionally, power is expended in driving any ZI bits that have to be driven (i.e., that are different from their past driven values). Note that the circuit of Figure 4 is simplified for the purpose of exposition – Domino logic circuit designs (not shown here) are used to optimize delays introduced in the signal path that enables the driver, particularly in the case when the lines are driven from the same source (as in the case of dedicated links, such as the function unit–register port connections of Datapath B).

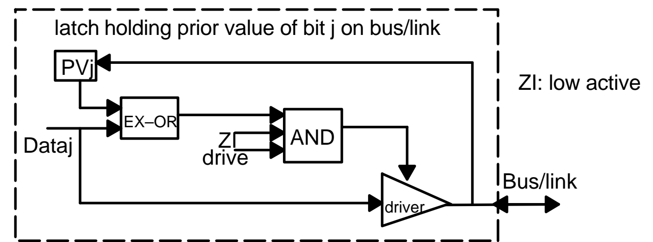


Figure 4. Circuit (equivalent) for driving only the bit lines that changed since the last transfer. Keeper transistors not shown

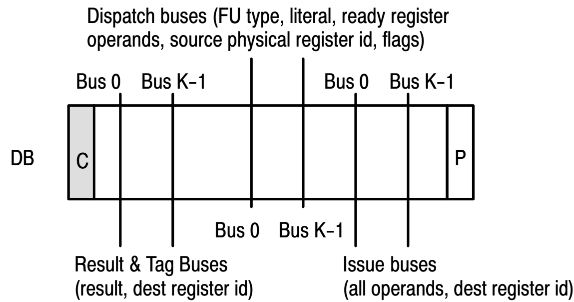
5. Sources of Energy Dissipation

To estimate the energy savings from zero byte encoding, we consider the main sources of energy dissipation within the dispatch buffer, the register files and the reorder buffer. In this paper, we do not focus on the energy savings due to zero byte encoding in other datapath components such as caches or function units, since they have been studied elsewhere [2, 12].

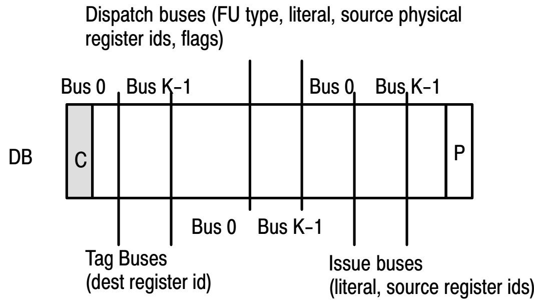
5.1. Energy dissipations within the DB

There are three main sources of energy dissipation in the dispatch buffer (DB). First, energy is dissipated in the DB in the process of establishing DB entries for dispatched instructions. Second, dissipation occurs when FUs complete and forward the results and/or status information to the DB entries. Third, energy is dissipated at the time of issuing instructions to the FUs.

To understand these components and the complexity of the connections, it is useful to refer to Figure 5. The basic structure of the DB is depicted in Figure 5(a), which shows the main *internal buses/wires* through the DB and the

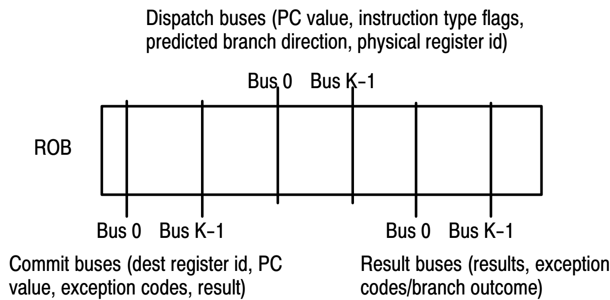


(a) Datapath A



(b) Datapath B

C: associative addressing logic for locating free entries
P: priority, function unit selection logic



(c) Datapath A and B

Figure 5. Dispatch buffer (DB) and Reorder buffer (ROB) structures and relevant *internal* buses/connections

information that flows on these buses. In a K–way superscalar CPU, there are K sets of internal buses, in general. At the time of dispatch, the DB is associatively addressed (using the associative addressing logic labeled as C in Figure 5(a)) to locate a free entry for an instruction being dispatched. With up to a maximum of K dispatches per cycle, K free dispatch buffer entries have to be located simultaneously in one cycle using this associative search. At the time of dispatch, an entry has to be set up for each dispatched instruction. For Datapath A this amounts to writing the following information into the entry: (a) the contents of the valid (input) operand registers that are read out from the Physical Register File (PRF), (b) tag values/flags to indicate that these data are valid, (c) the addresses of input registers whose contents were not ready, as tags used for forwarding, (d) literal operands and (e) the FU type needed and some bit flags (such as the predicted direction for a branch). For Datapath B, the following information has to be written into the DB entry at the time of dispatch are: (a) physical register ids for the operand registers, (b) literal operand, if any, and, (c) FU type needed and flags. The data on the relevant connections are shown in Figure 5(a) and 5(b). When results are generated by a FU, for both Datapaths A and B, energy dissipations occur within the DB in comparing tags (= destination register addresses) on each of the K tag buses. For Datapath A, additional dissipation occurs in the process of latching in results into operand value fields of matching entries. When all input operands of a DB entry are valid, the associated wakeup logic triggers the arbitration logic to compete with other ready–to–be–issued instructions for issue.

When instructions are issued from the DB, for both Datapaths A and B, dissipations occur in the DB in the process of generating wakeup signals and in arbitrating for a FU (as described in [8]). For Datapath A, additional energy dissipations occur within the DB as operand values are read out from the DB and driven on the issue buses to the selected FUs.

5.2. Dissipations in the Register Files

For Datapath A, the input operands are read in from the PRF at dispatch time, irrespective of whether they are valid or not – this causes energy dissipation in the PRF. For Datapath B, the operands are read out from the PRF at the time of issue. Some of the read out values can be bypassed. For both Datapaths A and B, energy is also dissipated in the PRF when the FUs complete and write their results into the PRF. When results are committed, the writes into the ARF cause energy dissipation as well.

5.3. Dissipations in the ROB

The internal buses used to establish, use and commit ROB entries are shown in Figure 5(c) and apply to both Datapaths A and B. Energy dissipations taking place within the ROB have the following components:

- Dissipations that occur in the ROB at the time of setting up the entry for dispatched instructions. Energy is dissipated at the time of writing the address of the instruction (PC value), flags indicating the instruction type (branch, store, register–to–register), the destination register id, and predicted branch direction (for a branch instruction) into the appropriate slot of the ROB. Overall, this dissipation is really in the form of

energy spent in writing to the register file that implements the ROB.

- Dissipations that take place when results are written to the ROB from the function units. This is again in the form of energy spent in writing to the register file that implements the ROB.
- Dissipations that take place when ROB entries are committed; this dissipation is the energy spent in reading the register file that implements the ROB.
- Dissipations that occur in clearing the ROB on mispredictions – this involves clearing the valid bit of all ROB entries. Compared to all of the other ROB dissipation components, this is quite small.

In variants of datapaths, where the destination value slots within ROB entries are used as physical registers for results, additional dissipations occur in the ROB due to the use of associative addressing to determine if any ROB entries exist for a register used as a source by a later instruction. These variants are not considered in this paper.

6. Experimental Methodology

The widely–used Simplescalar simulator [3] was significantly modified to implement *true hardware level, cycle–by–cycle* simulation models for such datapath components as dispatch buffers (for both types of datapaths), reorder buffers, register files, forwarding interconnections and dedicated transfer links. The Simplescalar simulator lumps ROBs, PRFs and the DB together into RUUs, making it impossible to directly assess switching activities within these components independently. It also assumes a single stage that performs instruction decoding, dispatching and register renaming. Such feats are difficult to accomplish entirely within a single, realistic pipeline stage. The front end of the Simplescalar pipeline was therefore modified to perform these steps over two pipeline stages.

To get reasonable overall simulation performance with all the instrumentation in place, we resorted to the use of multithreading. Specifically, we use a separate thread for the data stream analysis, as shown in Figure 6. The data acquired from basic instrumentation within the main simulation thread was buffered and fed into a separate thread where it was analyzed for the lack of entropy within significant byte slices and all byte slices within a data item as well as across consecutive data items within a data stream.

For estimating the energy/power for the key datapath components, the transition counts and event sequences gleaned from the multithreaded simulator were used, along with the energy dissipations for each type of event, as measured from the actual VLSI layouts using SPICE. CMOS layouts for the DB, PRF, ARF and DB in a 0.5 micron 4 metal layer CMOS process (HPCMOS–14TB) were used for the key datapath components to get an accurate idea of the energy dissipations for each type of transition. The results for the 0.5 micron layouts are quite representative, although greatly scaled down compared to what one would see with 0.18 micron implementations running at a faster clock rate. The exception to this are the wire dissipations *outside* these components. Dissipations on such wires at small feature sizes become relatively dominant.

The register files that implement the ROB and DB were carefully designed to optimize the dimensions and allow the use of a 300 MHz clock. A Vdd of 3.3 volts is assumed for all the measurements (The value of the clock rate was determined by the cache layouts for a two–stage pipelined cache in the same technology.) In particular, these register files feature differential sensing, limited bit line driving and pulsed word line driving to save power. Augmentations to

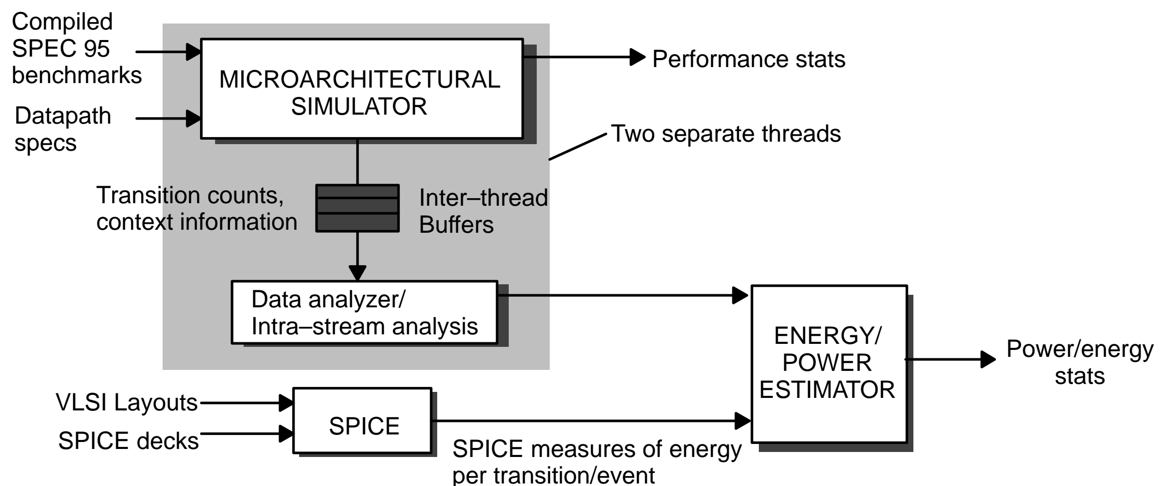


Figure 6. Experimental Methodology

the register file structures for the DB were also fully implemented; a pulldown comparator was used for associative data forwarding to entries within the DB and the device sizes were carefully optimized to strike a balance between the response speed and the energy dissipations. For each energy dissipating event, SPICE measurements were used to determine the dissipated energy. These measurements were used in conjunction with the transitions counted by the hardware–level, cycle–by–cycle simulator to estimate energy/power accurately.

Our methodology in computing the energy dissipations within these datapath components is to look at the traffic on each of the internal buses for the DB and the ROB and the traffic directed through register file ports and use these traffic measures to quantify the resulting energy requirements. We assumed that when a function unit produces a 32–bit result, it will drive at most 32 bits even if wider connections are available. We assume this to be true for the base cases as well. When zero byte encoding is in use, we also record the average number of zero bytes.

For the traffic directed to the DB from the FUs, we also record the number of DB entries that match the tag value floated on the result buses to estimate energy dissipations in the tag matching process and the energy spent in latching in result values (for Datapath A). A precharged comparator, similar to what has been described in [8] is used; mismatches cause energy dissipation by discharging the match lines.

The configuration of the system studied was as follows. The L1 I–cache and L1 D–cache were both 32 KBytes in capacity with a line size of 32 Bytes, with the former being direct–mapped and the latter being 4–way set–associative. A 4–way set–associative, integrated L2 cache with a capacity of 512 KBytes and a line size of 64 Bytes was assumed. The size of the dispatch buffer and the re–order buffer were kept at 60 entries and 100 entries respectively. The physical register file for integers and floats were 128 in number each. The function units are as follows: 4 integer units, one integer multiply/divide unit, 4 floating point multiply–add units, one floating point multiply/divide unit one load unit and one store unit. The following latencies of basic operations were assumed: integer addition – 1 cycle, floating point addition – 2 cycles, multiplication – 4 cycles, division – 12 cycles, load operation – at least 2 cycles excluding address computation. A 4–way dispatch, and a 4–way commitment were assumed. For Datapath A, a 4–way issue was assumed, and for Datapath B the issue width was limited by the number of FUs. For all of the SPEC 95 benchmarks, the results from the simulation of the first 200 million instructions were discarded and the results from the execution of the following 200 million instructions were used. Specified optimization levels and reference inputs were used for all the simulated benchmarks.

7. Experimental Results

The following figures show several representative results for both datapaths A and B. In Figure 7, the percentage of leading zero bytes and zero bytes throughout the operand, weighted and averaged over 32–bit and 64–bit operands, are shown for representative SPEC 95 benchmarks as well as for the averages across *all* integer and floating point benchmarks in the SPEC 95 suite and total averages (across all benchmarks – integers and floating points). For the integer benchmarks, which are

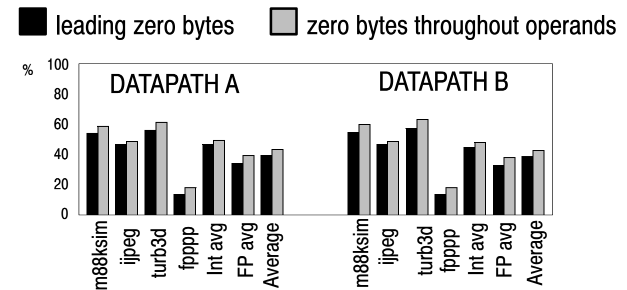


Figure 7. % of leading bytes with all zeros and total bytes with all zeros averaged over all operand sizes and all data streams

dominated by mostly 32–bit integer data streams, roughly 47% of the leading bytes are all zeros, while about 50% of the bytes contain all zeros (including all zero bytes in the leading bit positions). For the floating point benchmarks, which consist of 32–bit integers, 32–bit floats and 64–bit doubles, the weighted average shows that roughly 34% of the leading bytes are all zeros while 39% of the bytes in the data streams contain all zeros.

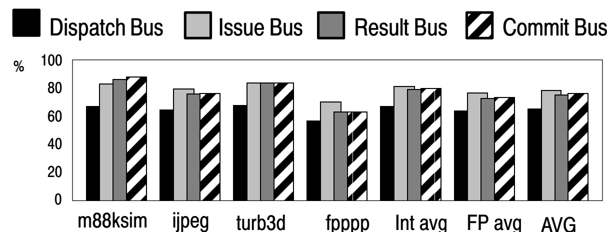


Figure 8. % of bit slices not driven using bit–slice invariance on top of zero byte encoding for Datapath A

In Figures 8 and 9, we show what happens when the zero valued bytes are not driven on transfer paths and, in addition, bit values within the non–zero valued bytes that do not change from their prior–driven values on the interconnection (driven from the same source or different sources) are also not driven. The combined amount of bit slices that do not have to be driven in this case is in the range of 65% to 90% with the average of 79%. This result shows

potential power savings in transferring data on the interconnections, particularly as wire capacitances begin to dominate in implementations with smaller feature sizes.

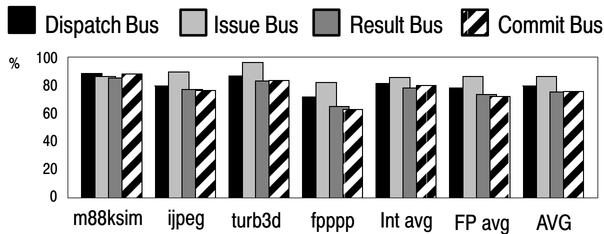


Figure 9. % of bit slices not driven using bit–slice invariance on top of zero byte encoding for Datapath B

The power measurements for the DB, the ROB, the register files are shown in Figures 10, 11, 12 and 13. We show the power dissipations for the original datapaths (i.e., base cases) along with the power that results when zero byte encoding is used. *Note that the absolute power values are low because of the use of 0.5 micron layouts*. In computing the power dissipations for the original datapaths without zero byte encoding, *we made a conservative assumption in favor of these base cases: even with 64–bit wide datapath connections available, all of that datapath would not be driven if the data length was 32–bits.*

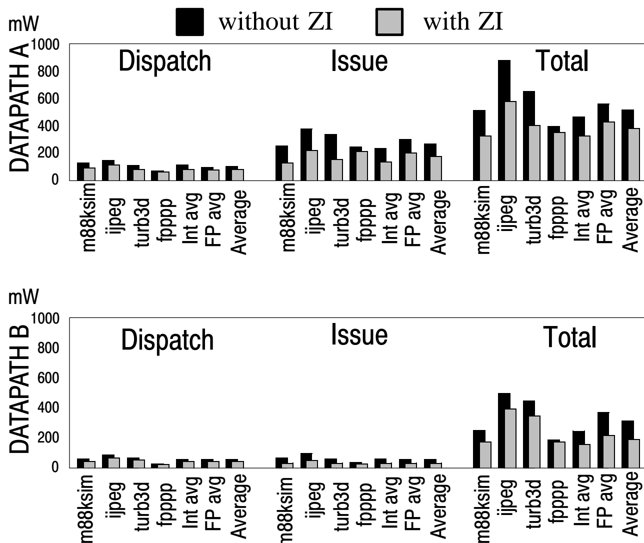


Figure 10. Power dissipations within the DB with and without zero byte encoding (ZI bits)

The results show that energy savings with zero encoding are higher for SPECint 95 and slightly lower for the SPECfp 95 applications: this is because the opportunities for zero byte encoding are higher in the SPECint 95 benchmarks. Notice also that the fraction of power savings is not as high as the relative number of all–zero bytes. This is a direct consequence of two things. First, additional bit flags (the ZI bits) are needed to encode bytes – both all–zero bytes and all other bytes. Storing, driving and reading these bit flags cause energy dissipations. Second, most of the energy dissipating events have flat dissipation components that do not depend on the number of bits driven, stored, read or written (such as word line drive power, tag matching, address decoder power etc.). The power savings realized for Datapaths A and B, as seen from Figures 10, 11, 12 and 13 are:

For Datapath A: In the DB, 26% of the dispatch component, 38% of the issue component, 12% of the forwarding component and total of about 29% over all SPEC 95 benchmarks. In the integer PRF, savings are 34%, and in the floating point PRF – 20%.

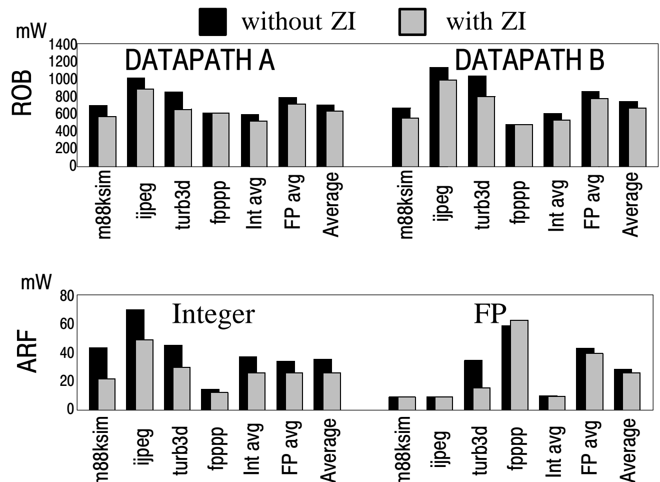


Figure 11. Power dissipations within the ROB and in ARFs with and without zero byte encoding (only Datapath A results are shown for ARF)

For Datapath B: In the DB, savings are 21% at the dispatch, 43% at the issue and 20% total. Power dissipation is reduced by 27% for the integer PRF and by 17% for the floating point PRF.

The energy dissipations in the ROBs for both datapaths are identical. This is because we assume a fixed number of connections (4 sets for the 4–way processors) to the ROB irrespective of datapath used. For Datapath B, this implies the use of buffering in–between the FUs and the ROB to buffer results that cannot be all written together to the ROB in a single cycle. The power requirements of the ARF are also identical for both datapaths as a consequence of this. For both datapaths, savings within the ROB are about 10%, within the integer ARF about 27% and within the floating point ARF about 9%.

Figures 11, 12 and 13 also indicate that zero–byte encoding does not always result in power savings. For instance, for the fppp benchmark, the dissipations in the register files and the ROB are actually higher with zero byte encoding. This is a consequence of the fact that the percentage of bytes containing zeros is very low in this

benchmark (Figure 7) and the 8 ZI bits that have to be driven, stored and read for 64 bit floats (which dominate fpppp) cause higher energy dissipations.

A comparison of two datapaths for the power savings realized for the DB shows that a higher fraction of power savings is realized for Datapath A. This is as expected – the DB entries for Datapath B do not contain fields for register operands; a large fraction of the DB power is due to the tag comparators. Our analysis show that out of 128 comparators within the DB for each forwarding bus, about 36 are waiting for the results at the time of a forwarding instance. With energy dissipations taking place in these comparators on a *mismatch,* savings from zero byte encoding are somewhat defeated. Power dissipation in physical register files is higher for Datapath B in the base case, since every register source of every instruction is read out from the register file at the time of issue and, in addition, the register files have more read and write ports in this datapath.

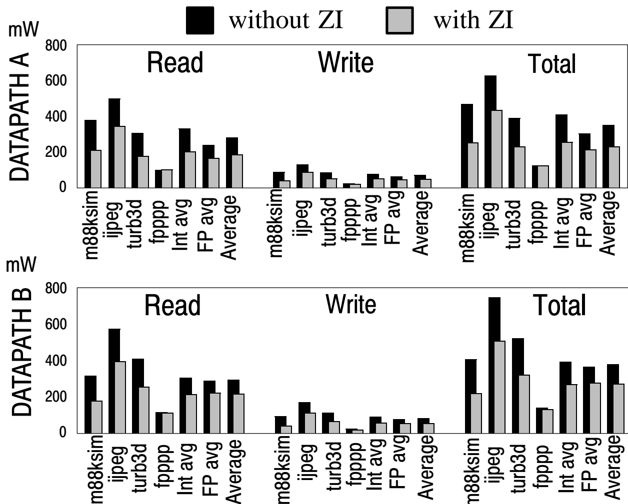


Figure 12. Power dissipations within the integer physical register file

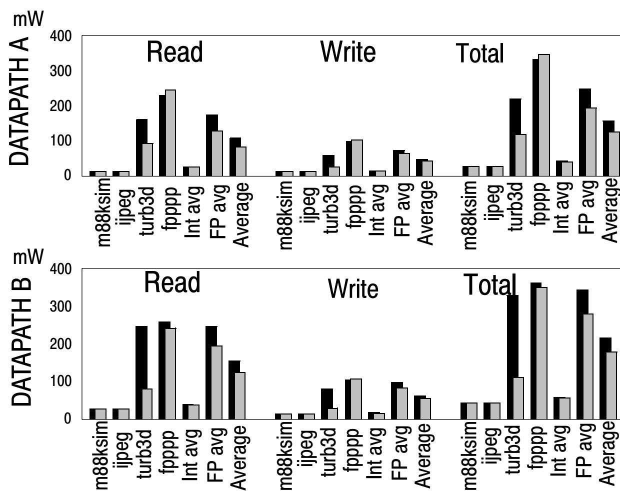


Figure 13. Power dissipations within the floating point physical register file

8. Conclusions

By encoding zero bytes within data items flowing within the datapath, power requirements within the register files, dispatch buffers, reorder buffers can be cut down by 10% to 34%. Additional power savings are realizable within the various interconnections if only the ZI bits and the unchanged bits within the non–zero bytes are driven. The savings on the interconnections are in the range of 65% to 90% with the average of 79%, if zero byte encoding is used and only unchanged bits within the non–zero bytes (and the ZI bits) are driven. These power savings are realized without any increase in the cycle time and only with a modest increase (11%) in the layout area of the on–chip storage components.

9. References

[1] Bhandarkar, D., "Alpha Implementations and Architecture – Complete Reference and Guide", Digital Press, 1996.

[2] Brooks, D. and Martonosi, M., "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", Proc. HPCA, 1999.

[3] Burger, D., and Austin, T. M., "The SimpleScalar tool set: Version 2.0", Tech. Report, Dept. of CS, Univ. of Wisconsin–Madison, June 1997.

[4] Canal R., Gonzales A., and Smith J., "Very Low Power Pipelines using Significance Compression", Proc. Micro33, 2000.

[5] Ghose, K., "Reducing Energy Requirements for Instruction Issue and Dispatch in Superscalar Microprocessors," Proc. ISLPED, 2000, pp.231–234.

[6] Gonzalez, R., and Horowitz, M., "Energy dissipation in general purpose microprocessors", IEEE Journal of Solid State Circuits, 31(9): September 1996, pp 1277–1284.

[7] Microprocessor Report, various issues, 1996–1999.

[8] Palacharla, S., Jouppi, N. P. and Smith, J.E., "Quantifying the complexity of superscalar processors", Technical report CS–TR–96–1308, Dept. of CS, Univ. of Wisconsin, 1996.

[9] Pollack, F., "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies", Keynote Presentation, Micro–32, November 1999.

[10] Popescu V., et.al. "The Metaflow architecture", IEEE Micro, June 1991, pp. 10–13, 63–73.

[11] Tiwari, V. et al, "Reducing power in high–performance microprocessors", in 35th Design Automation Conference, 1998.

[12] Villa, L., Zhang, M. and Asanovic, K., "Dynamic Zero Compression for Cache Energy Reduction", Micro–33, Dec. 2000.

[13] Zyuban, V. and Kogge, P., "Optimization of High–Performance Superscalar Architectures for Energy Efficiency", Proc. ISLPED, 2000, pp. 84–89.