EXPLOITING BIT–SLICE INACTIVITIES FOR REDUCING ENERGY REQUIREMENTS OF SUPERSCALAR PROCESSORS*

Kanad Ghose, Dmitry Ponomarev, Gurhan Kucuk, Andrew Flinders
Dept. of Computer Science, State University of New York, Binghamton, NY 13902
email:{ghose. dima, gurhan, afilnders}@cs.binghamton.edu

Peter M. Kogge
Dept. of Computer Sci. and Engg, University of Notre Dame, Notre Dame, Indiana IN 46556
email: kogge@cse.nd.edu

and

Nikzad (Benny) Toomarian
CISM, Jet Propulsion Lab, Pasadena, CA 91109
email: benny@cism.jpl.nasa.gov

Abstract

We show by simulating the execution of SPEC 95 benchmarks on a detailed register–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. 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 a typical superscalar datapath. Power savings are achieved within all explicit and implicit storage components such as caches, register files, instruction dispatch buffers, re–order buffers, as well as interconnections such as buses and direct links. Relevant circuit components for encoding zero bytes within storage components and interconnections and avoiding the driving of bit lines that do not change in value are also presented. Preliminary results showing power savings in representative datpath components are quite encouraging.

1. Introduction

Modern superscalar processors incorporate explicit and implicit components such as on–chip caches, large physical register files, an instruction dispatch buffer and a reorder buffer. The later two are essentially implemented as physical register files with associative addressing and additional logic.

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 to save power. Specifically, we avoid the activation of byte slices that contain all zeros along the interconnections and storage components of the processor to conserve power. The simulated execution of the SPEC 95 benchmarks show that on the average about 50% or more of the bytes of operands are zero. 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. We can thus curtail bit–slice activities quite drastically by exploiting these two facts and achieve serious power savings. This study focuses on the exploitation of bit slice inactivities in reducing the power associated with instruction dispatching, issuing and forwarding in a superscalar microprocessor.

Exploiting the presence of bytes containing all zeros is not new. It was suggested and used to reduce the power dissipation in dispatch buffers in [1]. In [2], the same fact is used to reduce energy dissipations within the caches. In [3], the presence of leading zero bytes in a result produced by an

* 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

integer function unit is exploited to reduce the power dissipated by the integer function unit and within the interconnections; energy savings possible within the explicit and implicit storage components of the datapath are not considered in [3]. This study demonstrates that the presence of bytes with all zeros in data streams – integers, floating point operands and others – can be exploited to reduce energy requirements all across the datapath of contemporary superscalar CPUs, including explicit and implicit storage components and interconnections. (For the purpose of this paper, a *data stream* is a sequence of operand values from possibly different sources, that flow on an interconnection during program execution.) This study also identifies an additional source for energy savings in driving data on the flow paths – a technique that exploits the correlationship among consecutive data flowing on a datapath bus or interconnection. Energy savings on buses due to correlationships among the past and immediate values of bits have also been explored [4, 5]. However, the energy savings possible from just the exploitation of such bit correlationships is not very great. The use of encodings for zero–valued bytes, together with the use of bit–value invariances, as presented in this paper offer the potential of more significant energy savings in the data flow paths.

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 resolving 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. These are as follows:

- **Datapath A** (Figure 1 (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 [6]. 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 [6]. 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.

- **Datapath B** (Figure 1 (b)): Here, 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 written to the register file, using bypassing logic) 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. The dispatch/issue logic can be implemented using a global scoreboard that keeps track of instructions and register/FU availability. Alternatively, an associative logic similar to that of Datapath A can be used to update the status of input registers for instructions waiting within the DB (as shown in Figure 1(b)).

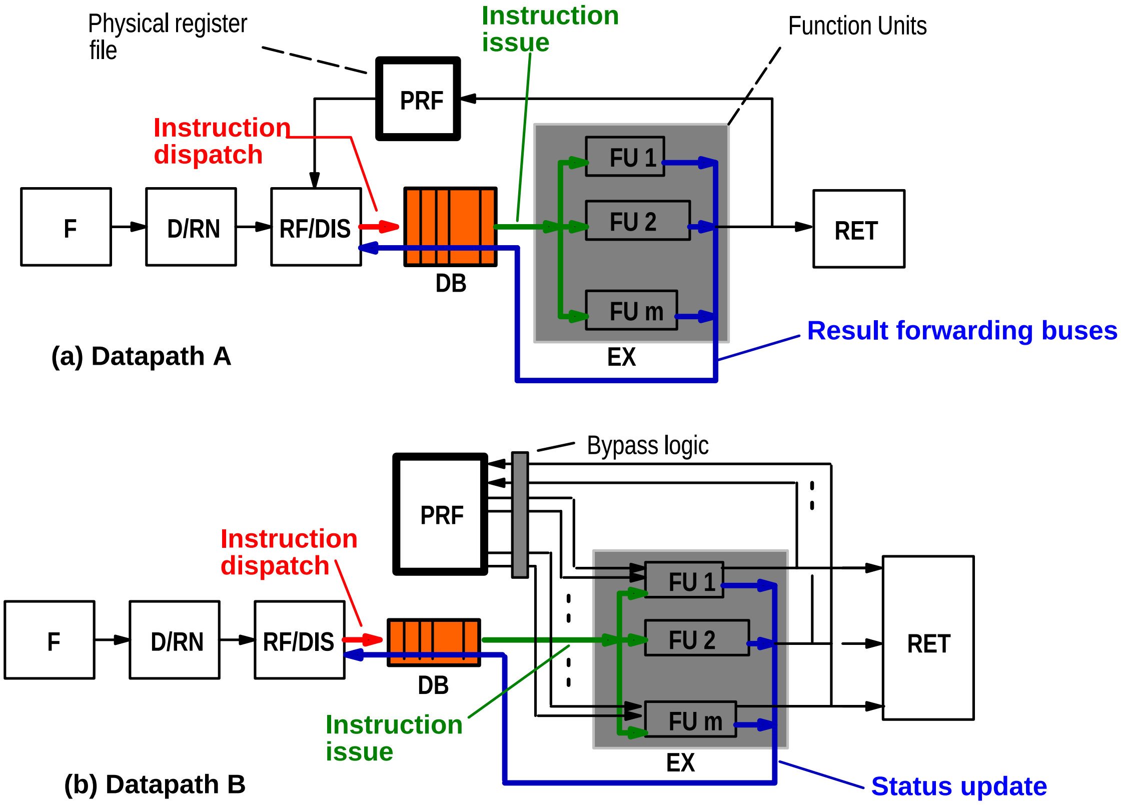


Figure 1. Two variants of a superscalar datapath

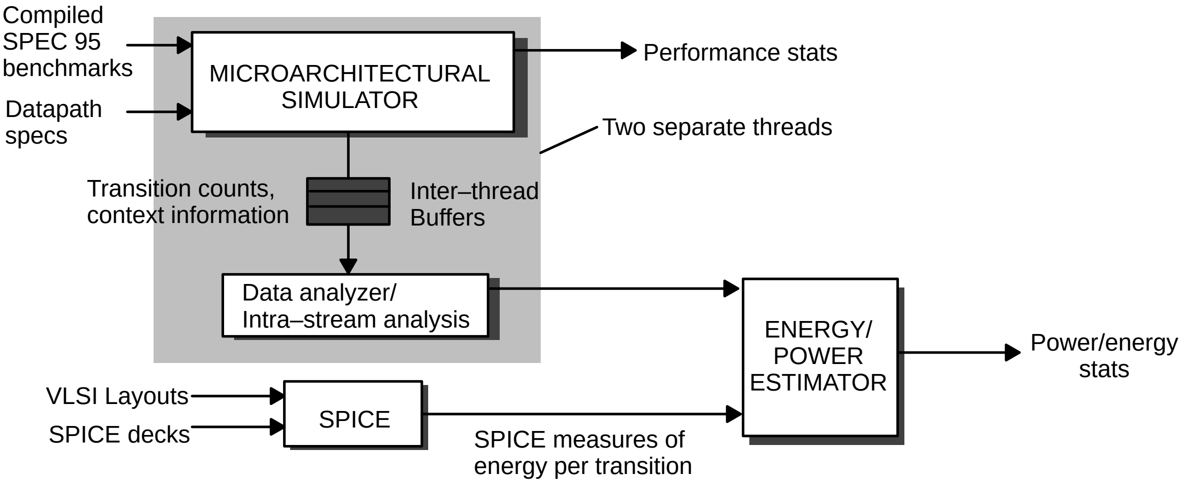

Compiled SPEC 95 benchmarks
Datapath specs
MICROARCHITECTURAL SIMULATOR
Performance stats
Two separate threads
Transition counts, context information
Inter–thread Buffers
Data analyzer/ Intra–stream analysis
ENERGY/ POWER ESTIMATOR
Power/energy stats
VLSI Layouts
SPICE decks
SPICE
SPICE measures of energy per transition

Figure 2. Evaluation Methodology

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.

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 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–lvel 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 schemes.

Irrespective of the type of datapath used, a considerable amount of energy dissipation occurs in the process of instruction dispatching, result/status forwarding, instruction issuing, result writeback and instruction retirement. The goal of this work is to exploit the lack of entropy in data streams along *all* data flow paths (and data stored within storage devices) to save energy by:

1. Only driving byte slices that do not contain all zeros.
2. Avoiding the driving of 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.
3. Avoiding 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, dispatch buffers and the re–order buffer.
4. Avoiding or minimizing energy dissipations within function unit byte or bit slices that do not contain significant data bits/bytes.

We thus exploit the lack of entropy in data streams as exhibited by the presence of zero bytes and invariance in the values of bits driven for the non–zero bytes and the encoding bits on a bus from one transfer to the next. In the next section, we describe our methodology for identifying the lack of entropy in data streams and present relevant experimental results.

4. Experimental Methodology

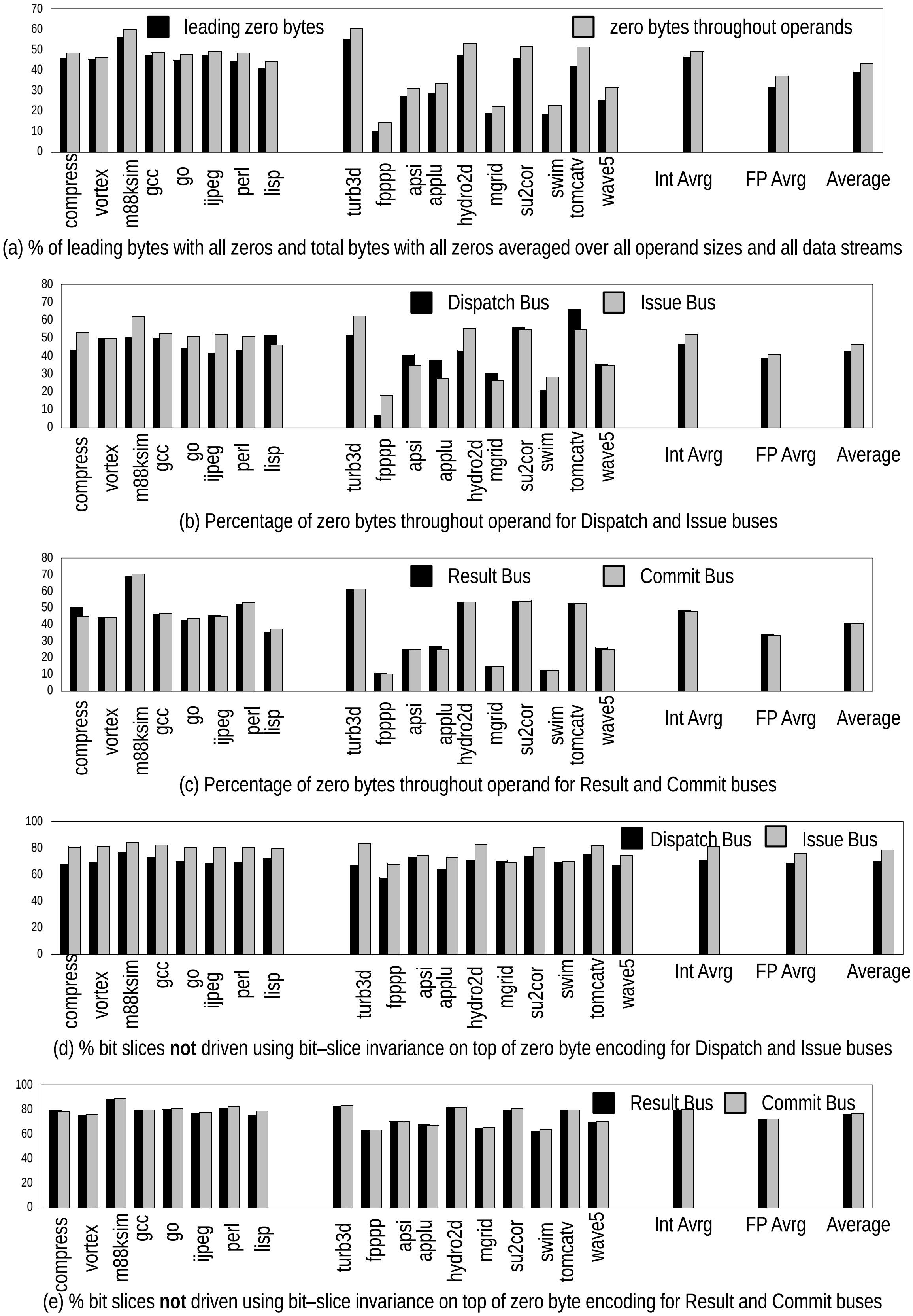
Our methodology for identifying the lack of entropy in the data streams and the energy savings achieved using the techniques described above is shown in Figure 2. The widely–used Simplescalar simulator was significantly modified to implement *true hardware level, cycle–by–cycle* simulation models for such datapath components as dispatch buffers (for both types of datapaths), re–order buffers, forwarding interconnections, dedicated transfer links and caches. (The cycle–by–cycle Simplescalar simulator simulates cycle level events without modeling specific implementation details of these components; the implementation details are crucial to the accurate estimation of energy dissipations.) Instrumentation to keep track of data values and transition counts were also put into place. The data acquired from the instrumentation 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 across consecutive data items within a data stream. The use of a separate analyzer thread allowed the overall simulation speed to be kept at an acceptable level. With a single thread implementing all of the simulation, instrumentation and analysis, the overall simulation speed was reduced by as much as 40% compared to the original Simplescalar simulation without any modification and instrumentation. With both threads in place as shown, and with the use of inter–thread buffers of an optimized size, the overall simulation time achieved was often significantly better on a SMP compared to the single–threaded implementation. The performance of the dual–threaded version was also about the same as that of the original Simplescalar simulator without any of the enhancements and the instrumentation. The transition counts gleaned from the simulator are finally used along with capacitive coefficients obtained from SPICE simulations of actual VLSI layouts to evaluate the energy savings. Our current measurements are based on layouts of register files, dispatch buffers, re–order buffers and caches in a 0.5 micron 4–metal layer CMOS process. This allows for a 300 MHz. clock, as determined by the access time of the L1 cache tag/data arrays. We are in the process of migrating these layouts to a 0.18 micron process to bring our measurements to the realm of current practices. The results for the 0.5 micron layouts are, however, quite representative even if the layouts used use a coarser feature size.

The configuration of the system studied were 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 128 KBytes and a line size of 64 Bytes was assumed. The size of the dispatch buffer and the re–order buffer were kept at 64 entries and 100 entries respectively. The function unit assumed for the two datapaths (Datapath A, Datapath B) were as follows: 4 integer units, one integer multiply/divide unit, 4 floating point multiply–add units, one floating point multiply/divide units and one load and a load/store unit. For Datapath B, the physical register files were assumed to have the requisite number of ports to permit concurrent register file accesses by function units. The latencies were as used in the original Simplescalar simulator, with the exception of the floating point multiply–add unit, which was assumed to have a latency of 4 cycles. All function units with a multi cycle latency were assumed to be pipelined. A 4–way dispatch, a 4–way issue and a 4–way commitment was assumed. 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 800 million instructions were used. Specified optimization levels and reference inputs were used for all the simulated benchmarks.

Figure 3 shows several representative results for Datapath A. In Figure 3 (a), 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 individual SPEC 95 benchmarks. For the integer benchmarks, which are dominated by mostly 32–bit integer data streams, roughly 47% of the leading bytes are all zeros, while about 49% 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 32% of the leading bytes are all zeros while 37% of the bytes in the data streams contain all zeros. The floating point numbers account for the higher difference between the number of leading zero bytes and the number of zero bytes throughout the operands. Figures 3(b) and 3(c) show the distribution of zero bytes throughout the operands for some specific data flow paths (the dispatch, issue, result and the commit buses.) Considerable energy savings will be achieved within all datapath components such as caches, register files, the dispatch buffer, the re–order buffer and function units if the presence of leading bytes with all zeros and bytes containing zeros throughout the operands are exploited.

In Figures 3 (d) and 3(e), 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 a 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 81% on the aforesaid buses. This result suggests that a considerable amount of power can be saved in transferring data on these interconnections, particularly as wire capacitances begin to dominate in implementations that have smaller feature sizes.

The results for Datapath B are similar to that for Datapath A and are shown in Figure 4. Here again, by exploiting bit–slice invariance and the presence of bytes with all zeros, considerable energy savings will be achieved in all datapath components, buses and dedicated links (in–between function units and



(a) % of leading bytes with all zeros and total bytes with all zeros averaged over all operand sizes and all data streams

(b) Percentage of zero bytes throughout operand for Dispatch and Issue buses

(c) Percentage of zero bytes throughout operand for Result and Commit buses

(d) % bit slices **not** driven using bit–slice invariance on top of zero byte encoding for Dispatch and Issue buses

(e) % bit slices **not** driven using bit–slice invariance on top of zero byte encoding for Result and Commit buses

Figure 3. Representative results for Datapath A

register files, in–between cache levels, and between load/store function units and the L1 caches).

5. Circuit Components

We now present the circuit components necessary for encoding and exploiting all zero valued bytes and for exploiting invariance of bit–slice values. These are depicted in Figure 5.

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 5 (a) 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. Encoded data values can be transferred by 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), re–order 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 5 (b). 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 bits (namely the ZI bits) have to be driven, read and written. The ZI bits, by themselves, increase the area of the register files and tag/data RAMs by about 12%.

Figure 5 (c) 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 5(c) 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 5(c). 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 5(c) is simplified for the purpose of exposition – clever 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).

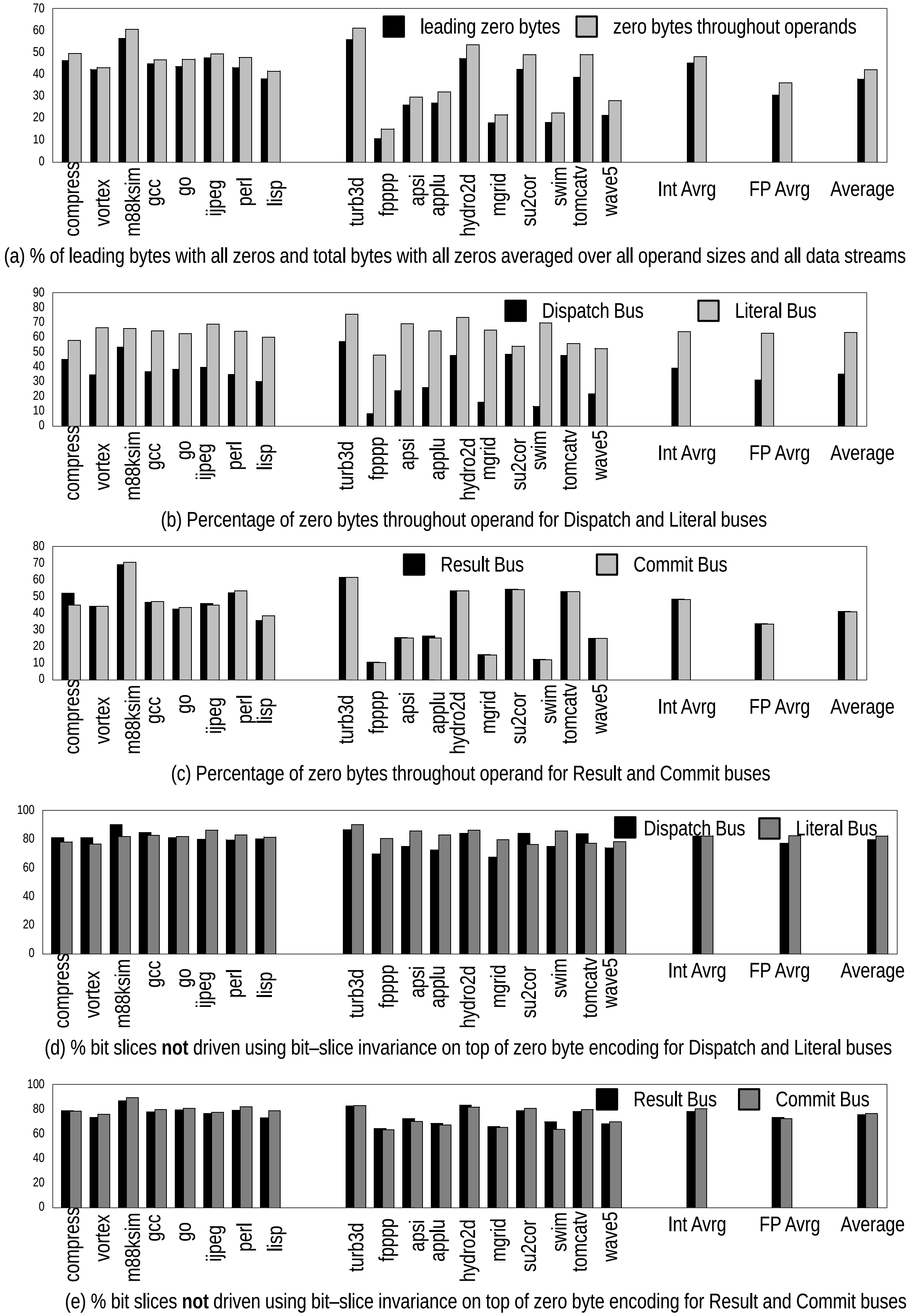
6. Preliminary Results and Conclusions

For the 0.5 micron layouts used, *preliminary* computations of the energy savings for some of the datapath components, averaged over all of the SPEC 95 benchmarks are shown in Table I. We are in the process of computing energy savings for the caches and the interconnections (based on the estimated wire lengths).

	Dispatch Buffer	Register File	Re–order Buffer
Datapath A	43.8%	37%	38.4 %
Datapath B	30.4%	29.6%	38.4%

Table 1. Preliminary estimates of energy savings achieved with proposed techniques for some datapath components

The results from Table I suggests that the proposed techniques achieve higher *percentage* of savings for dispatch buffers and register files for Datapath B than for Datapath A. There are several reasons for this: first, the dispatch buffer entries for Datapath B are narrower compared to the width of entries for Datapath A. Thus, for Datapath B, energy savings in the dispatch buffer result only in driving the non–zero bytes of literal operands and only those bits that are different from the prior value on the



(a) % of leading bytes with all zeros and total bytes with all zeros averaged over all operand sizes and all data streams

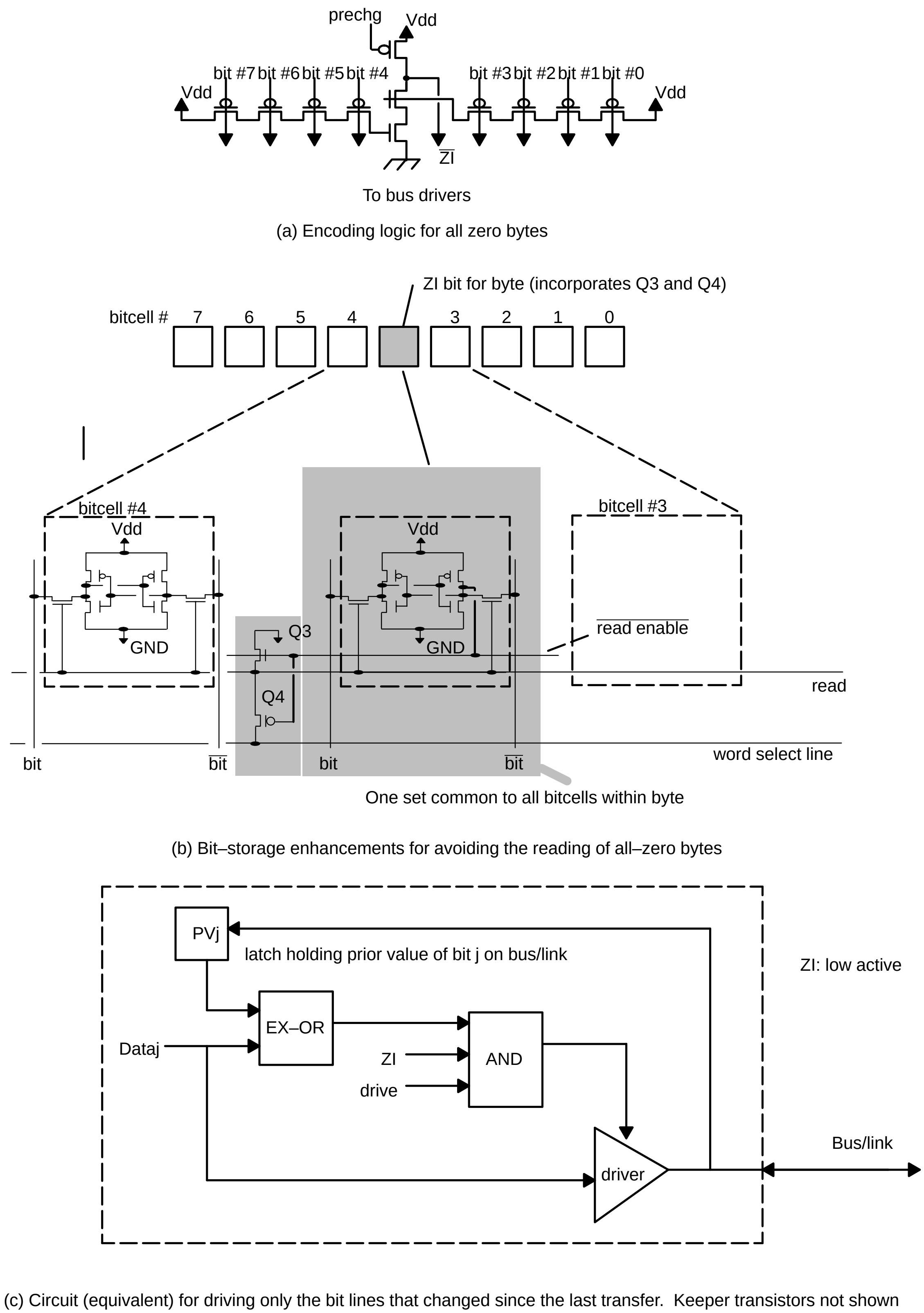
(b) Percentage of zero bytes throughout operand for Dispatch and Literal buses

(c) Percentage of zero bytes throughout operand for Result and Commit buses

(d) % bit slices **not** driven using bit–slice invariance on top of zero byte encoding for Dispatch and Literal buses

(e) % bit slices **not** driven using bit–slice invariance on top of zero byte encoding for Result and Commit buses

Figure 4. Representative results for Datapath B



(a) Encoding logic for all zero bytes

(b) Bit–storage enhancements for avoiding the reading of all–zero bytes

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

Figure 5. Examples of circuit components for exploiting the presence of bytes with all zeros and bit–slice invariance within the non–zero bytes corresponding bit line, on the narrow literal bus. For Datapath A, significant energy reductions are achieved within the dispatch buffer by not driving zero bytes of operands and bits in the non–zero bytes that remain unchanged for the literals, valid register operands (at the time of dispatch) and results (which are forwarded to waiting DB entries). For Datapath B, the energy savings in the register file is smaller compared to that for Datapath A for a completely different reason: the large number of ports required on the register file for Datapath B introduce an overhead that scales disproportionately with the number of bits driven (because of sizing requirements for pre–chargers and sense amps of heavily loaded bit lines). The re–order buffer structures for both Datapaths are identical, and so are the energy savings. The results given in Table I are only preliminary: additional quantification is needed for the other datapath components and the interconnections to gauge the overall energy savings possible from the exploitation of the lack of entropy in data streams. Nevertheless, the results of Table I do suggest that serious power reductions are possible through the use of the techniques proposed in this paper.

References

1. Ghose, K., “Reducing Energy Requirements for Instruction Issue and Dispatch in Superscalar Microprocessors”, in Proc. ISLPED 2000 (July 2000), pp.231–234.
2. Villa, L., Zhang, M. and Asanovic, K., “Dynamic Zero Compression for Cache Energy Reduction”, to appear in Micro–33, Dec. 2000.
3. Brooks, D. and Martonosi, M., “Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance”, Proc. HPCA, 1999.
4. Stan, M. and Burleson, W.P., “Bus–Invert Coding for Low–Power I/O”, IEEE Trans. on VLSI, 3(1), 1995, pp. 49–58.
5. Su, C.L., Tsui, C.Y. and Despain, A.M., “Saving Power in the Control Path of Embedded Processors”, IEEE Design and Test of Computers, 11(4), 1994, pp. 24–30.
6. 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.