

Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth

Gabriel H. Loh
Yale University
Dept. of Computer Science
gabriel.loh@yale.edu

Abstract

In a 64-bit processor, many of the data values actually used in computations require much narrower data-widths. In this study, we demonstrate that instruction data-widths exhibit very strong temporal locality and describe mechanisms to accurately predict data-widths.

To exploit the predictability of data-widths, we propose a Multi-Bit-Width (MBW) microarchitecture which, when the opportunity arises, takes the wires normally used to route the operands and bypass the result of a 64-bit instruction, and instead uses them for multiple narrow-width instructions. This technique increases the effective issue width without adding many additional wires by reusing already existing datapaths.

Compared to a traditional four-wide superscalar processor, our best MBW configuration with a peak issue rate of eight IPC achieves a 7.1% speedup on the simulated SPECint2000 benchmarks, which performs very well when compared to a 7.9% speedup attainable by a processor with a perfect data-width predictor.

1 Introduction

The critical nature of data bypassing and the trend toward deeper pipeline depths make increasing the issue-width of superscalar processors very challenging. Palacharla et al’s study concludes that for a traditional 8-issue superscalar processor in a 0.18 μ m process, the result bypassing delay limits the processor cycle time [24, 25]. The number of bypass paths is also proportional to the number of pipeline stages. The pipeline depth of the Intel NetBurst microarchitecture is twenty stages [11], and it is likely that future processors will use even deeper pipelines [10, 30]. Increasing the issue width of a superscalar processor requires more area for additional functional units, additional wires to route the extra operands to the functional units, and more wires to bypass the results to dependent instructions. This increase in area can easily translate into longer clock cycle times and therefore lower overall performance.

In this paper, we propose a new approach that increases the effective issue rate by reusing already existing datapaths. We believe our technique does not have much effect on the chip-area required by the execution core, and therefore would have little impact on clock speed. A 64-bit processor needs 64 wires to route the data bits of a single operand or result of an instruction. If this particular data value is only 32 bits wide, that is the upper or most significant 32 bits are all zero, then only half of the 64 wires are really needed. An independent instruction could potentially use the remaining 32 wires to route another 32-bit value.

Brooks and Martonosi published a significant study that has many similarities to our work, and it is important to distinguish how these studies differ [3]. In their work, Brooks and Martonosi assume that operand data-widths are available at the time an instruction is scheduled to a functional unit. Current processors already require multiple cycles for scheduling [11], and so we feel that this assumption is no longer valid. To handle this, we introduce data-width prediction, which relies on the fact that instruction data-widths do not vary greatly throughout the execution of a program. We call this data-width locality. In their work, they observed that data-widths sometimes change, but only used this to argue against purely static (compiler or profile) approaches to estimating instruction data-widths. To increase processor performance, they describe a method to perform a dynamic SIMD packing that, for example, collects four add instructions with 16-bit operands and merges them into a single vector addition. Our proposed scheme can be viewed as a more general MIMD approach (MIMD in the sense that the different instructions do not all have to be ADDs for example; not MIMD in the multithreaded programming context).

There are three primary contributions of this study. First in Section 2, we introduce the concept of data-width locality and we show how to accurately predict the size of an instruction’s operands and result. Second in Section 3, we describe how to modify a traditional superscalar microarchitecture to make use of data-width locality. Third, we evaluate the performance of our proposed technique in com-

parison to a traditional microarchitecture in Section 4. The remainder of the paper surveys related work in Section 5 and concludes in Section 6.

2 Data-Width Locality

Integer instructions in a 64-bit architecture typically operate on 64-bit values. The largest value of an unsigned 64-bit integer is about 18×10^{18} . One would expect that most programs do not actually manipulate data values of such large magnitude. In this section, we explore the data-width characteristics of the SPECint2000 benchmarks to discover their distribution and predictability.

We start with a definition for an instruction’s *data-width*. The data-width of an instruction is the maximum of its input operand widths and its output value width. The width of a data value is the position of the first zero bit such that all bits in more significant positions are also zero. We will only consider simple one-cycle *icomp* or integer computation instructions, such as addition/subtraction or bit-wise logic, that consume register values or immediate constants and create register results. Brooks and Martonosi showed that there are many instructions with data-widths less than 64 bits for the SPECint95 and MediaBench benchmarks. We repeat a similar experiment for the SPECint2000 benchmarks, and then show that the widths of instructions are very predictable. In Section 3, we will describe one method to take advantage of these observations to build a super-scalar processor with a larger effective issue width.

2.1 Data-Width Profiling Environment

We used the SimpleScalar [5] simulation toolset for the Alpha architecture to collect statistics about the data-widths of instructions in eight of the SPECint2000 benchmarks. The benchmarks were compiled with `cc` ver. 5.9 on an Alpha 21264 with optimizations enabled (`-arch ev6 -non_shared -fast -O4`). We do not consider the floating point applications because our definition of data-widths does not extend naturally to floating point values. For all benchmarks, we chose simulation windows that do not include start-up code. We also placed the simulation window to capture different phases of the program execution [28]. For example, our sampling windows cover both compression and decompression phases of the compressing benchmarks (`gzip` and `bzip2`). We simulate all benchmarks for 200 million instructions. The benchmark names, input sets, and number of initial instructions skipped are all listed in Table 1.

| Benchmark Name | Input Set | Input File | Instructions Skipped |
|----------------------|-----------|------------|----------------------|
| <code>bzip2</code> | ref | source | 50.2B |
| <code>gap</code> | ref | — | 30B |
| <code>gcc</code> | ref | 200.i | 20B |
| <code>gzip</code> | ref | random | 31.39B |
| <code>parser</code> | ref | — | 18.9B |
| <code>perlbmk</code> | ref | splitmail | 100M |
| <code>vortex</code> | ref | lendian2 | 13.6B |
| <code>vpr</code> | ref | place | 200M |

Table 1. The benchmarks from the SPECint2000 suite used in our study, along with the individual input files and the number of instructions skipped before the start of simulation.

2.2 Data-Width Patterns

The purpose of our first experiment is to determine the instruction data-width distribution of the integer computation, or *icomp*, instructions in the SPECint2000 benchmarks. In particular, we only consider instructions that would operate on a “simple” integer ALU, which excludes integer multiplication for example. For every dynamic *icomp* instruction, we classify it as *short*-sized if its data-width is 16 bits or less, *addr*-sized (address) for 17-33 bits, or *qword*-sized (quad-word) for 34-64 bits. For this section, we treat all data as unsigned values, so negative values are automatically classified as *qword*-sized. The data-width distributions, locality and predictability do not change by much if we include small magnitude negative numbers. We use a 33-bit cutoff instead of the more natural boundary of 32 bits because for our binaries the compiler has laid out the memory using 33-bit wide addresses.

The distribution of *icomp* instructions consists mostly of *qword*-sized instructions, nearly 50% on average, as illustrated by Figure 1. Our data-width distributions differ from those reported by Brooks and Martonosi [3] for several reasons: we use a more recent benchmark suite, they do not consider an instruction’s output size in determining data-width, and we do not include some instructions such as multiplies.

Having shown that the integer instructions in SPECint2000 exhibit different data-widths, our second experiment measures the *data-width locality* of the *icomp* instructions. A static *icomp* instruction is said to exhibit high data-width locality when the data-width of a dynamic instance is very likely to be the same as recent instances. For every instruction, we record the data-width of the instruction for the last n instances. We then measure the fraction of the dynamic *icomp* instructions where the data-width is identical to the last n instances. For this

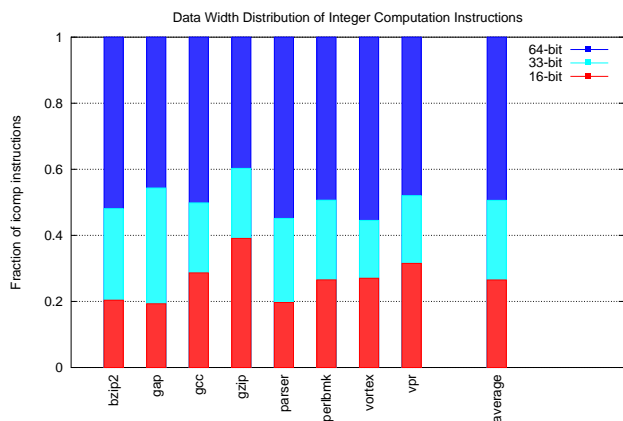


Figure 1. The distribution of integer computation instruction data-widths by benchmark and on average.

experiment, we consider instructions to have the same data-width based on the the quantized short-, addr- and qword-size classifications.

We performed this experiment using values 1, 3 and 7 for n . We chose these values because they correspond to the maximum values of 1-, 2- and 3-bit counters, respectively. Figure 2 shows the results for each benchmark as well as the average. We found that over 94% of dynamic icomp instructions have the same data-width as the previous instance, just under 90% are the same as all previous 3 instances, and 86% as all previous 7 instances. Brooks and Martonosi observed that there exist instructions whose data-widths vary during the execution of a program, but the strong data-width locality of instructions suggests that these data-width transitions do not occur too frequently.

2.3 Data-Width Predictability

Having demonstrated that the data-widths of icomp instructions do exhibit strong data-width locality, our next task is to show how we can accurately predict the data-width of an instruction. Data-width prediction has similarities to branch prediction [26, 29, 32] and data-value prediction [17, 18] in that all three techniques attempt to predict something about an instruction. Data-width prediction is a novel concept because instead of predicting the value of an instruction (e.g. the branch outcome or the output value), we are making a prediction about a *property* of the value.

We propose two simple data-width predictors. The first is similar to the load value predictor [18]. The second is an extension of the bimodal saturating counter branch predictor [29].

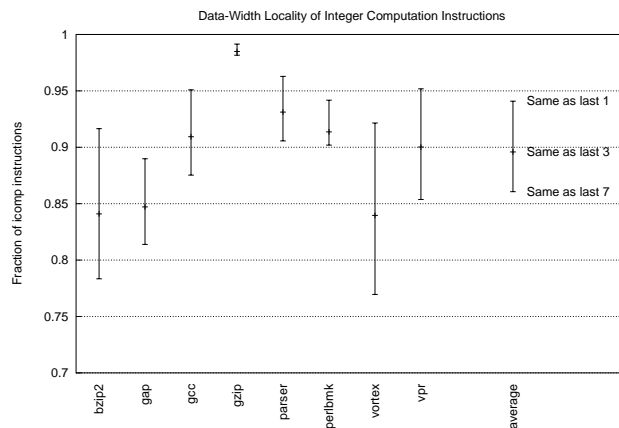


Figure 2. The fraction of icomp instructions that exhibited the same data-width as the previous 1, 3 and 7 instances of the same instruction.

The first predictor is the *resetting* counter. We maintain two pieces of information for each instruction. First is the most recent data-width of the instruction. The second is a k -bit confidence counter that indicates how many consecutive times the stored data-width has been repeated, as shown in Figure 3a. The first phase of the prediction is the lookup phase. If the confidence counter is less than the maximum value of $2^k - 1$, then the predictor makes a conservative prediction that the instruction is qword-sized. Otherwise, the prediction is made according to the stored value of the most recent data-width. The update step proceeds as follows. If there was a data-width misprediction, or if the actual instruction data-width is qword sized, then the counter is reset to zero. Otherwise, if the outcome agreed with the stored data-width, then the counter is incremented, saturating at the maximum value of $2^k - 1$. Note that the stored width never needs to represent qword-sized instructions, and so a single bit can cover the cases of short- or addr-sized predictions.

There are three possible predictions (short-, addr-, or qword-sized), and so our second predictor is an extension of the bimodal saturating counter that we call a *trimodal* counter. This FSM adds some hysteresis to the predictor such that more than one misprediction may be necessary to change the outcome of the next prediction. Figure 3b shows the state transition diagram of the trimodal predictor. There are six states: strong short (SS), weak short (WS), strong addr (SA), weak addr (WA), strong qword (SQ) and weak qword (WQ). The predictor in a state $*X$ predicts that the instruction will have a data-width of X . For example, $*A$ (states SA and WA) result in an addr-sized prediction. Dur-

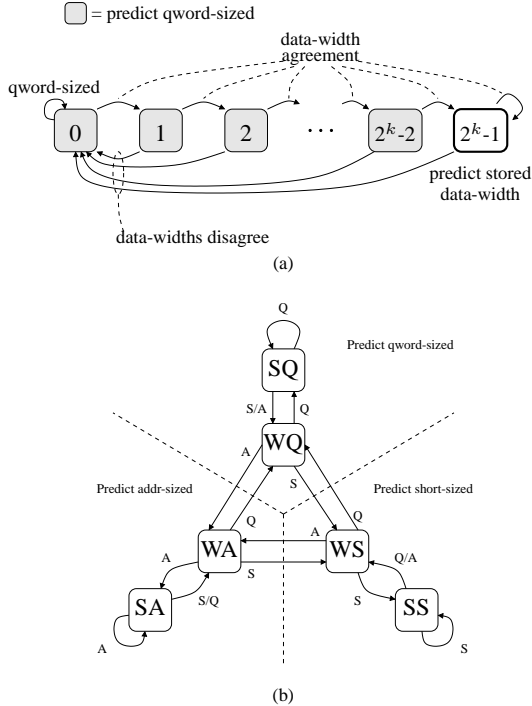


Figure 3. (a) The resetting counter data-width predictor, and (b) the trimodal data-width predictor.

ing the update phase, a misprediction in a weak state causes the FSM to enter the weak state of the most recent outcome (e.g. the FSM in the WS state transitions to WA if the outcome was addr-sized, and WQ if the outcome was qword-sized). A correct prediction in a weak state causes the FSM to enter the corresponding strong state. A misprediction in a strong state causes the FSM to drop back into a weak state.

For either the resetting predictor or the trimodal predictor, a PC-indexed table of these state comprise the overall data-width predictor. This is the same structure as in simple branch predictors [29].

Depending on the prediction and the actual data-width, there are two kinds of mispredictions that we consider: *aggressive* and *conservative* mispredictions. An aggressive data-width misprediction occurs when the prediction is smaller than the actual data-width. This type of mis-speculation may lead to scheduler replay traps because the processor only reads the lower bits of an operand when the computation requires all 64 bits. A conservative data-width misprediction occurs when the prediction is larger than the actual data-width. Conservative data-width mispredictions are not harmful in that they do not cause scheduler replays, but they still represent missed opportunities for data-width-optimizations.

We simulated data-width predictors that use an infinite

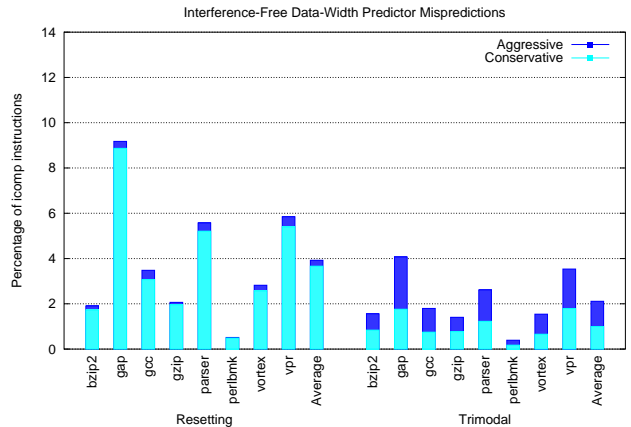


Figure 4. The per-benchmark and average misprediction rates for interference-free resetting and trimodal data-width predictors.

number of entries, such that there is no interference between different icomp instructions. Figure 4 shows the distribution of aggressively and conservatively mispredicted icomp instructions for both types of data-width predictors for each benchmark and the average over all benchmarks. The results for the resetting predictor are for a configuration with $k=2$ bits. We also experimented with $k=1$ and $k=3$ bits, but these configurations did not perform as well. The trimodal predictor achieves a lower overall misprediction rate of just over 2%, but the fraction of aggressive mispredictions is larger than that of the resetting predictor.

We also simulated the same data-width predictors with finite-sized tables. Figure 5 shows the misprediction rates (averaged across the benchmarks) for predictor tables ranging from 1024 to 64K entries. We also include the misprediction rates of the interference-free configurations for comparison. Interference between unrelated instructions in the predictor tables severely impact prediction accuracy at smaller table sizes. After the table sizes reach 16K entries, the predictor accuracy levels out and is very close to the ideal case of an interference-free predictor. For either type of predictor, each table entry stores three bits, and so a 16K table requires 6KB of state.

3 Implementation

In the previous section, we observed that programs exhibit strong data-width locality and predictability, but we have not explained how one could use this information to improve the performance of a superscalar processor. In this section, we describe one way to modify a traditional superscalar microarchitecture to increase the effective issue

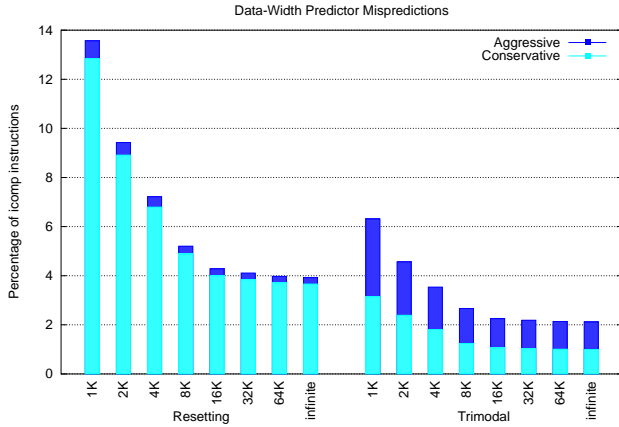


Figure 5. The average misprediction rates for different sized predictor tables for the resetting and trimodal data-width predictors.

bandwidth of the processor.

In a traditional microarchitecture, increasing the issue width requires a quadratic increase in the delay of the critical result bypassing logic [24, 25]. Larger issue widths also require additional wires to route the values from the reservation stations to the functional units. We propose the *Multiple Bit-Width* (MBW) microarchitecture that makes use of the existing operand routing and result bypassing wiring to issue a single 64-bit instruction or four independent 16-bit instructions, for example. For simplicity, we initially describe how multiple short-sized instructions can be issued on a single 64-bit datapath; addr-sized instructions can be dealt with similarly, and we will return to them later.

3.1 The Multi-Bit-Width Core

A typical superscalar core consists of multiple components. A reorder buffer (ROB) tracks the status of in-flight instructions that may be waiting for register operands, currently executing, or already completed but are waiting to commit their results. The issue queue consists of multiple reservation stations (RS) that contain some state and possibly data for instructions that are waiting to issue. For this discussion, we will assume a base processor configuration with two integer ALUs.

3.1.1 Making the Data-Width Prediction

Starting at the beginning of the processor pipeline, the data-width prediction lookup can begin as soon as the instruction’s PC is known. Note that we will not use the data-width prediction until the instruction scheduling stage, and there-

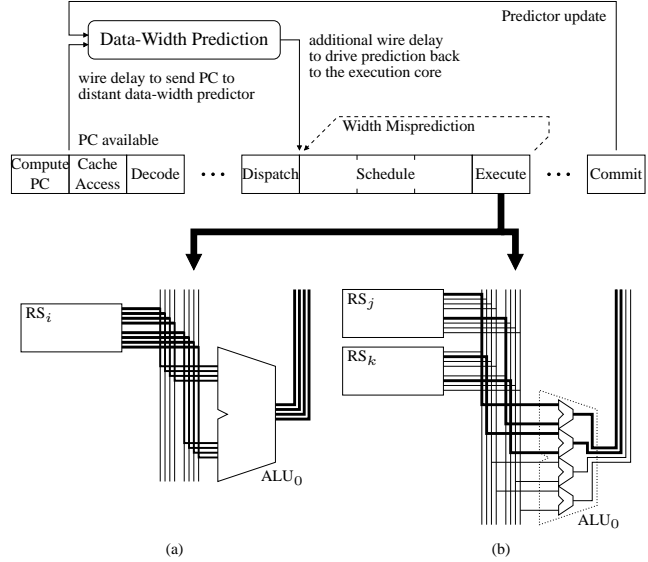


Figure 6. (top) An overview of the Multi-Bit-Width (MBW) pipeline, and its interaction with a data-width predictor, (a) data path for a single qword-sized instruction, and (b) data-path for two short-sized instructions sharing a single 64-bit datapath.

fore we have many cycles to perform the lookup as illustrated in Figure 6. This is important because the data-width prediction tables can be placed in a location on the chip that is distant from the critical paths of the execution core.

3.1.2 MBW Execution

At any point in time, the reservation stations may contain (predicted) short-width or qword-sized instructions. In our MBW microarchitecture, the exact same datapath can be repartitioned into multiple short-width datapaths. Figure 6a shows the dataflow of a single qword-sized instruction from a reservation station to a functional unit (ALU₀) and out through the result bypass path (the data are represented by bold lines and each wire represents 16 bits). This is how the data might flow in a traditional superscalar processor where the operands and the result each require 64 bits of wiring. Figure 6b shows two short-width instructions issuing to two short-width ALUs. The two instructions share the 64-bit datapath, including the result bypass path. This allows up to four short-width instructions to issue per cycle without adding any additional wires to the critical result bypass paths. Depending on what instructions are ready to issue on each cycle, the datapath to ALU₀ may vary in function, acting as a single regular 64-bit datapath on one cycle

and as four 16-bit datapaths on a different cycle.

Our scheduler uses an oldest first policy, but it is also aware of the different data-widths. In a four-issue processor with two integer ALUs, our proposed MBW scheme would allow ALU_0 to switch between acting as a regular ALU or four short-width ALUs, while ALU_1 always executes qword-sized instructions. When scheduling a qword-sized instruction, the scheduler will attempt to assign the instruction to ALU_1 (the fulltime qword-sized ALU). This leaves the MBW ALU_0 to either accept multiple short-width instructions or another single qword-sized instruction.

The reservation stations need additional logic and wiring. When a reservation station issues a qword-sized instruction, the reservation station asserts the operands on all 64 bits of both operand buses. When the reservation station issues a short-width instruction, the 16-bit operands must be aligned to the proper short-width datapath. This data-shifting requires a few extra multiplexers in the reservation station. Brooks and Martonosi argue that these multiplexers may already exist in processors that support multimedia instruction sets and therefore do not incur any “additional” cost [4].

Although the short-width ALUs shown in Figure 6b are conceptually four distinct functional units, the original 64-bit ALU could be used instead. Processors supporting multimedia instruction sets already have ALUs that can exploit SIMD parallelism by executing the same instruction on multiple data within the same register. The only modification to the ALU for our MBW microarchitecture is that each subword operation would require its own independent opcode.

After a short-width computation completes, the result may be bypassed to another short-width instruction or to a qword-sized instruction. In the first case, the short-width ALU input multiplexers must be widened to select from any of the four short-width ALU outputs. Palacharla et al argue that the majority of the bypassing delay is due to wire delay, and so we feel that this slight increase in the result selection logic should not have a serious impact on the cycle time [24, 25]. In the second case of a short-width instruction feeding a dependent qword-sized instruction, the consuming ALU must first right shift the 16-bit result from the corresponding 16-bit bypass path and then pad the upper 48 bits with zeros.

3.1.3 Misprediction Detection, Recovery and Update

To detect a data-width misprediction, the short-width ALUs can simply use their overflow detection logic. In processors supporting multimedia instruction sets, saturating operations are commonly available where a result that overflows gets pinned to the largest possible value for the data’s width. Upon detecting a data-width misprediction, the pro-

cessor must reexecute the instruction on an appropriately sized functional unit. This is called a replay trap. In our simulations in Section 4, a replay trap causes the processor to flush the entire scheduling pipeline. Any instructions that have not issued must restart from the beginning of the scheduler pipeline. Note that this is more conservative than the selective reissue policy used by the Intel NetBurst microarchitecture for cache miss replays [11].

Finally after an instruction commits, the processor reports the actual data-width of the instruction to the data-width predictor so that it can update its prediction tables. Although speculative updates are very important for predicting branch outcomes [9], we found that the data-width locality is sufficiently strong that the prediction rates are not greatly affected by delaying predictor updates until the commit stage.

3.1.4 Other Hardware Considerations

Although our MBW technique may increase the effective issue-width of the processor, this can also lead to an increase in the demand for register file (RF) ports. For reading values from the register file, the processor must support up to six additional 16-bit reads. One possibility is to replicate only the least significant 16 bits of the register file and have the narrow width instructions read their data from this second RF. This still increases the area needed for the register file, but this solution should have significantly less impact than extending the register file to support three additional sets of qword-width ports.

The instruction wakeup logic will become more complex since three additional instructions may issue each cycle. To prevent the wakeup logic from becoming critical, it may be necessary to use additional techniques such as segmenting the instruction window [12] or using tag elimination [7]. Furthermore, there are relatively few cycles where all four short-sized issue slots are busy. An alternative approach would be to limit the number of short-sized instructions that can issue on the same datapath to three or even two instructions. This in turn can reduce the complexity of the wakeup and select logic, as well as reduce the number of ports for the register file.

3.2 MBW Extensions

The MBW microarchitecture described in the previous subsection is but one specific configuration that makes use of data-width locality. We now briefly describe three extensions that may be possible with our MBW processor: double pumping, an address-width MBW configuration, and handling negative values.

3.2.1 Double Pumping the ALUs

The latency of a 16-bit integer ALU operation may be significantly lower than that of a 64-bit computation. Depending on the implementation of the functional units, it may be possible to execute two dependent short-sized instructions in the same cycle. Such an approach has similarities to the “double-pumped” ALUs in the NetBurst microarchitecture, but achieves the effect by entirely different means [11]. Our approach requires the two dependent instructions to both be short-sized (and correctly predicted as such), while the NetBurst double-pumping is a circuit level technique that works for any sized data.

The number of short-sized datapaths remains limited to four, regardless of whether the instructions are double-pumped. Therefore, even though the four short-sized integer ALUs may be capable of executing a total of eight 16-bit operations in a single cycle, our processor microarchitecture still only has the datapath bandwidth to issue four short-sized instructions.

3.2.2 Handling addr-sized Instructions

Our data-width distribution results from Section 2 show that there are a significant number of instructions that operate on address-sized (33-bit) values. Two 33-bit operations can not get packed onto a single 64-bit datapath. For the small cost of increasing an ALU’s operand bus and bypass path widths to 66 bits, we could support two independent address-sized instructions on a single full-width datapath. Increasing the address processing issue width may be very helpful since the address computation is on the critical path of a load instruction. This technique does not scale well as programs and compilers use larger addresses. An alternative approach would be to pack a single 16-bit operation with a calculation that is at most 48 bits wide.

An address-width MBW configuration could be combined with the short-width MBW configuration described in the previous subsection. The datapath associated with ALU_0 could be time multiplexed between a single 64-bit datapath and four 16-bit datapaths. Independently, if we increase the datapath associated with ALU_1 to 66 bits, then it could be switched between a full-width path and two 33-bit datapaths. This would turn a traditional four-issue processor into a MBW processor with a peak issue rate of eight.

3.2.3 Supporting Negative Values

The data-width distribution profiling of Section 2 treated all values as unsigned integers, which automatically forced negative numbers into the qword-size category. We repeated the experiment, this time considering the absolute value of input operands and results, thus allowing small

magnitude negative numbers to be classified as either short-sized or addr-sized. This resulted in only a very minor change in the data-width distributions. Approximately 3.8% of instructions formerly in the qword-sized category (or 1.9% of all icomp instructions) were reclassified as either short- or addr-sized.

The MBW microarchitecture requires a few modifications to support negative values. First, the data-width misprediction logic must account for the signs of the inputs and output when identifying overflow situations. Second, instead of zero-extending short-sized (or addr-sized) values back into a 64-bit value, the processor must perform some form of sign-extension. Simply extending the most significant bit does not work. For example, adding the two short-sized values 0x7fff and 0x0001 results in 0x8000. Blindly sign-extending from the most significant bit results in the incorrect value of *negative* 32768. We must either allocate an extra wire for the sign, or limit what we consider short-sized to only include values with a magnitude that fits in 15 bits.

4 Performance

In this section, we present the results of our performance study to quantify the potential performance benefits of our MBW technique. We describe our simulation environment, and present the performance results of a limit study and for a variety of MBW processor configurations.

4.1 Methodology

Our performance results are based on SimpleScalar simulations [5]. In particular, we used the MASE cycle-level simulator [14] and modified the simulator to support our MBW issue policy and our data-width predictors. We only update the data-width predictors at the time an instruction retires. The MASE processor model simulates a reorder buffer (ROB) that is separate from the reservation stations (RS). The simulated benchmarks and simulation windows are identical to those listed in Table 1 for measuring data-width locality and predictability.

The base processor configuration for our timing simulations is listed in Table 2. Similar to the Intel NetBurst microarchitecture [11], we assume a 3-cycle pipelined instruction scheduler. The processor optimistically predicts that load instructions will hit in the level one data cache, and must replay if the load latency prediction was incorrect.

4.2 Results

This section presents the results of our limit study and our evaluation of MBW processors based on real predic-

| | |
|---------------------|------------------------|
| Fetch Width | 8 |
| Issue Width | 4 |
| Commit Width | 8 |
| Int ALUs | 2 |
| Int Multiplier | 1 |
| Memory Ports | 1 |
| ROB size | 128 insts |
| RS size | 32 insts |
| LSQ size | 64 insts |
| IFQ size | 64 insts |
| IFQ delay | 4 cycles |
| Scheduler Latency | 3 cycles |
| IL1 cache | 8KB, 8-way, 1 cycle |
| DL1 cache | 8KB, 4-way, 2 cycle |
| DL2 cache (unified) | 512KB, 8-way, 12 cycle |
| DL3 cache (unified) | 2MB, 16-way, 48 cycle |
| main memory latency | 250 cycles |

Table 2. The basic parameters for the simulated processors.

tors (i.e. non-perfect). We first present the performance of our baseline four-wide processor that we will compare our results to. Figure 7 shows the instruction level parallelism (ILP) achieved by the base processor (**Base-4**) for a processor with a 16K entry enhanced gskewed branch predictor [21] (dark bars) and blind memory speculation, and with perfect branch prediction and perfect memory disambiguation (light bars). The figure provides the individual IPC measurements for each benchmark, as well as the harmonic mean over all benchmarks.

Our first study is a limit study to determine the performance potential of our proposed technique. For this experiment, we use a processor configuration with perfect branch prediction, perfect memory disambiguation, and perfect data-width prediction.

Figure 8 shows the relative IPC speedup for different MBW configurations. The first, labeled **short**, is a MBW microarchitecture that allows one ALU datapath to switch between four short-sized paths and a single qword-sized path. The second, labeled **addr**, allows one ALU path to switch between two addr-sized paths and one qword-sized path. The third, labeled **both**, combines the **short** and **addr** configurations, allowing the first datapath to switch between a qword-sized path and four short-sized paths, *and* the second datapath to switch between a qword-sized path and two addr-sized paths. The fourth, labeled **both+DP**, is the **both** configuration where the short-sized ALUs may be double-pumped. The fifth, labeled **both+DP_{neg}** adds the ability to handle negative numbers. Finally, the last configuration labeled **base-8** is an upper limit where up to six integer instructions may issue every cycle, independent of data-width sizes. All of the performance gains reported in Figure 8

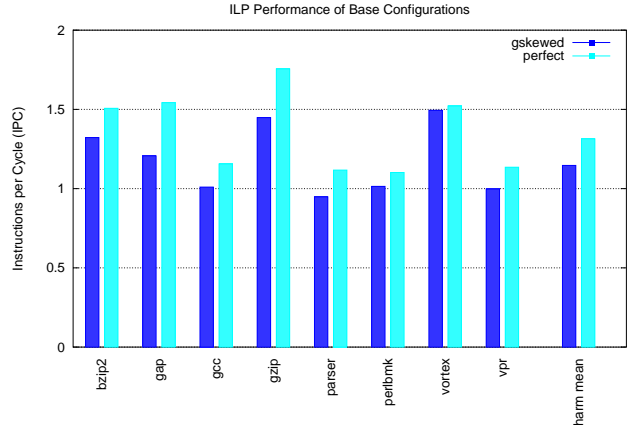


Figure 7. The performance of the baseline processors before augmenting with the MBW scheme.

are relative to the baseline results of Figure 7 where perfect prediction is used.

The benefit of adding more short-sized execution bandwidth or more addr-sized execution bandwidth varies depending on the benchmark. For instance, `gcc` and `gzip` both show greater performance gains for the **short** MBW configuration, while others such as `parser` perform better in the **addr** case. On average, increasing the short-sized execution bandwidth provides more benefit. Further extending the processor to the **both** configuration increases the performance gain for all benchmarks, with a speedup in harmonic mean ILP of 8.7%. Allowing the short-sized ALUs to double pump further increases the speedup to 9.3%. Supporting negative short- and addr-sized values yields an improvement of 9.7% over the base configuration. This is a strong improvement especially when compared to the idealized **base-8** configuration, which achieves a 13.9% speedup.

Having shown that the MBW approach could potentially provide reasonable performance improvements, we now explore the actual speedups achievable with more realistic assumptions about the various predictors. For this experiment, we use the **both+DP_{neg}** MBW configuration with a 16K entry gskewed predictor, blind speculation on load instructions, and the data-width predictors described in Section 2. In particular, we use a 16K entry resetting predictor and a 16K entry trimodal predictor. We compare our results to a configuration that uses a perfect data-width predictor (but still realistic branch and memory dependence prediction) to measure the impact of data-width mispredictions. Similar to the limit study, we also provide performance results for an eight-wide processor without any data-width constraints.

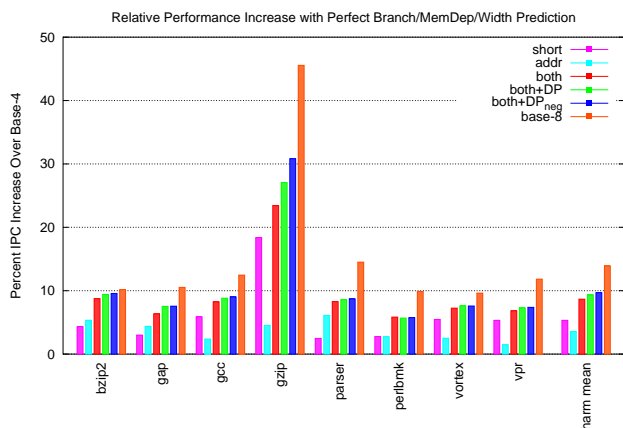


Figure 8. The relative performance increase over the baseline configuration with perfect prediction.

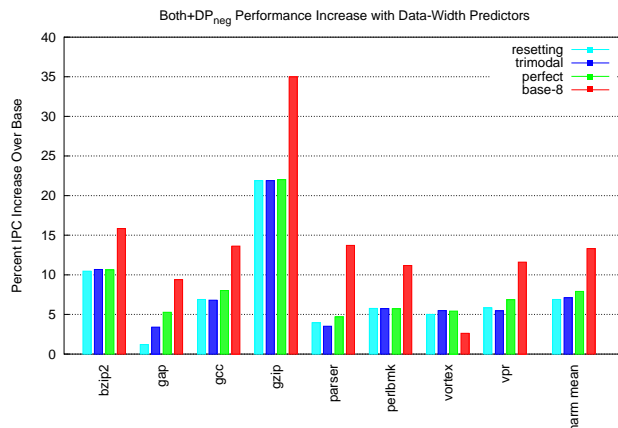


Figure 9. The relative performance increase of the both+DP_{neg} configuration with realistic data-width predictors.

Figure 9 shows the performance gains of the both+DP_{neg} MBW processor over the four-wide baseline. Even with the gskewed branch predictor, blind memory speculation and realistic data-width predictors, the both+DP_{neg} configuration can still achieve a 7.1% speedup. For the harmonic mean IPC, the configuration with the trimodal predictor performs slightly better than the resetting predictor. For some benchmarks, the resetting predictor yields a greater increase in ILP, and for other benchmarks the trimodal predictor is superior. A hybrid data-width predictor similar to those in the branch prediction literature [20] could perhaps yield even better performance. In any case, the impact of data-width mispredictions is not too large. The 7.1% speedup for the realistic predictors is respectable when compared to the 7.9% speedup using perfect data-width predictors.

The base-8 configuration for the vortex benchmark surprisingly performs worse than the MBW configurations. We attribute this to the cache prefetching effects of executing wrong-path instructions. We came to this conclusion from examining the execution statistics where we found that the total number of data cache accesses for the MBW configurations are much greater than for base-8.

While the ideal base-8 configuration exposes higher levels of ILP, we believe that the impact on the clock cycle makes our MBW approach more attractive. In Palacharla et al’s study, they found that an eight-wide processor in a 0.18 μ m process has a bypass delay of 1056.4ps. Using their same equations and constants, we estimated that the both+DP_{neg} configuration’s bypass delay is only 750.5ps with the conservative assumption that the register file can support eight qword-sized instructions. If we consider a

register file where only the lesser significant bits are replicated for the additional short- and addr-sized issue bandwidth, then the bypass delay drops to 539.0ps. At this point, bypassing is no longer the critical delay; Palacharla et al’s study indicates that the wakeup and scheduling logic for an 8-wide processor is 724.0ps (in the same technology). In Section 3, we briefly mentioned the possibility of limiting ALU₀’s shared datapath to only two per cycle to reduce the complexity of the wakeup and scheduling logic. This may be very desirable since the impact on IPC is less than 0.3%. In any case, the additional ILP exposed by the base-8 configuration (+13.3% over base-4) can not compensate for this increase in the critical bypass delay. We believe that in 0.13 μ m and smaller processes, the difference would be even more dramatic.

An argument could be made that the solution to overcome long result bypass delays is to partition the processor core into clusters, such as in the Alpha 21264 [13]. We feel that the MBW technique is still relevant in a clustered microarchitecture since it could be used to increase the per-cluster execution bandwidth. In Baniasadi and Moshovos’ study of instruction distribution heuristics for quad-clustered processors, they found that increasing the per-cluster issue width from two instructions per cycle to four per cycle results in a speedup of about 15-20% [2]. The MBW approach is one way to boost the per-cluster issue width, although we have not yet studied the exact performance impact of the MBW scheme in a clustered microarchitecture.

5 Related Work

There is much past work that relates to our proposed MBW microarchitecture in various ways. The most similar study was conducted by Brooks and Martonosi [3], which we have discussed throughout this paper in the relevant sections. Besides performance improvements, they proposed to reduce power consumption by clock gating the upper bits of the datapath when processing small-width values. In [4], Brooks and Martonosi also explored some speculative techniques to exploit the limited bit-widths of instructions. In their studies, they used blind speculation to either reduce power or increase performance. Our proposed data-width predictors could be useful for speculation control to reduce the number of replay traps incurred by blind speculation, thus further reducing power or increasing performance.

Multimedia instruction set extensions also allow the processor to increase the effective execution bandwidth by allowing a program to perform a single operation on multiple sets of input data in parallel [16, 27]. In the Intel MMX extensions, the input data may consist of several independent bytes, shorts (16-bit operands) or longs (32-bit operands) all packed into a single 64-bit value. With these SIMD instruction set extensions, the programmer or compiler must statically choose a data-width to use and make sure that the chosen width is sufficient for the task.

Our proposed data-width prediction follows from a large body of prior research in other forms of speculation. Branch prediction [15, 20, 21, 26, 29, 32–34], memory dependence prediction [6, 22, 23] and data value prediction [17, 18] are but a few of the more well-known areas. The structure of our data-width predictors has been heavily influenced by the branch prediction and value prediction literature. Our data-width speculation also has similarities to Liu and Lu’s circuit level speculation, which employs circuits that *usually* provide the correct result which allows for faster implementations of the circuits [19].

The Dynamic Zero Compression (DZC) cache reduces energy consumption by using a single bit to indicate that a full byte is zero [31]. Both the DZC cache and our MBW scheme take advantage of the fact that individual bytes of a larger value are often zero. The PowerPC 603 also took advantage of smaller-width data values to reduce the latency of multi-cycle instructions [8].

The function of the wires of the MBW datapath may vary from cycle to cycle. This bears some resemblance to *virtual wires* [1]. Virtual wires implement multiple logical wires by time-multiplexing a single physical wire. In a sense, our MBW scheme effectively time multiplexes the wires between a single 64-bit datapath and multiple narrow-width datapaths.

6 Conclusion and Future Work

Instruction operands and output values vary in size, and this can be used to increase the effective issue width of a superscalar processor without adding very many additional wires. We have shown that instruction data-widths exhibit strong locality, which makes them predictable. Using these predictions, a processor can decide to partition the existing wire resources among multiple instructions of smaller widths. Being able to use the wires as either a single 64-bit datapath or four independent 16-bit datapath increases the peak issue rate of a 4-wide processor to seven instructions per cycle. Slightly increasing the datapath width of the second ALU to 66 bits and allowing the ALU to act as two 33-bit functional units further increases the peak issue rate to eight IPC. We have discussed how supporting multiple bit-widths impacts the microarchitecture of the superscalar core, and have also demonstrated that this technique can potentially provide performance improvements across many of the SPECint2000 benchmarks.

Our concept of data-width locality is really a specific case of *data-attribute* locality. Other characteristics of data values may be predictable, and we are looking for some that may be useful to further increase processor performance. Our data-width predictors are not highly optimized either. Given the large latency that the predictors can tolerate, it should be possible to employ many of the techniques used in branch predictor design to improve the accuracy of the data-width predictors (such as a Bi-Mode data-width predictor [15]).

Acknowledgments

This research was supported by NSF Grants CCR-9702281 and CCR-9985304. We thank the anonymous reviewers for their helpful comments and criticisms that strengthened this paper. Rahul Sami, Daniel Friendly and Arvind Krishnamurthy of Yale University also provided several useful suggestions.

References

- [1] J. Babb, R. Tessier, and A. Agarwal. Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 142–151, Los Alamitos, CA, USA, 1993.
- [2] A. Baniasadi and A. Moshovos. Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 337–347, Monterey, CA, USA, 2000.

- [3] D. Brooks and M. Martonosi. Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance. In *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, pages 13–22, Orlando, FL, USA, January 1999.
- [4] 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.
- [5] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, University of Wisconsin, June 1997.
- [6] G. Z. Chrysos and J. S. Emer. Memory Dependence Prediction Using Store Sets. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 142–153, Barcelona, Spain, June 1998.
- [7] D. Ernst and T. Austin. Efficient Dynamic Scheduling Through Tag Elimination. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 37–45, Anchorage, Alaska, May 2002.
- [8] G. G. et al. A 2.2 W, 80 Mhz Superscalar RISC Microprocessor. *IEEE Journal of Solid-State Circuits*, 29(12):1440–1454, December 1994.
- [9] E. Hao, P.-Y. Chand, and Y. N. Patt. The Effect of Speculatively Updating Branch History on Branch Prediction Accuracy, Revisited. In *Proceedings of the 27th International Symposium on Microarchitecture*, pages 228–232, San Jose, CA, USA, November 1994.
- [10] A. Hartstein and T. R. Puzak. The Optimum Pipeline Depth for a Microprocessor. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 7–13, Anchorage, Alaska, May 2002.
- [11] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Karmean, A. Kyler, and P. Roussel. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal*, Q1 2001.
- [12] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar. The Optimum Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 14–24, Anchorage, Alaska, May 2002.
- [13] R. E. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro Magazine*, 19(2):24–36, March–April 1999.
- [14] E. Larson, S. Chatterjee, and T. Austin. MASE: A Novel Infrastructure for Detailed Microarchitectural Modeling. In *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, Tucson, AZ, USA, November 2001.
- [15] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge. The Bi-Mode Branch Predictor. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 4–13, Research Triangle Park, NC, USA, December 1997.
- [16] R. Lee. Accelerating Multimedia with Enhanced Microprocessors. *IEEE Micro Magazine*, 15(2):22–32, April 1995.
- [17] M. H. Lipasti and J. P. Shen. Exceeding the Dataflow Limit via Value Prediction. In *Proceedings of the 29th International Symposium on Microarchitecture*, pages 226–237, Paris, France, December 1996.
- [18] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value Locality and Load Value Prediction. In *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, Cambridge, MA, USA, October 1996.
- [19] T. Liu and S.-L. Lu. Performance Improvement with Circuit-Level Speculation. In *Proceedings of the 33rd International Symposium on Microarchitecture*, Monterey, CA, USA, December 2000.
- [20] S. McFarling. Combining Branch Predictors. TN 36, Compaq Computer Corporation Western Research Laboratory, June 1993.
- [21] P. Michaud, A. Seznev, and R. Uhlig. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 292–303, Boulder, CO, USA, June 1997.
- [22] A. Moshovos, S. E. Breach, T. N. Vijaykumar, and G. S. Sohi. Dynamic Speculation and Synchronization of Data Dependences. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 181–193, Boulder, CO, USA, June 1997.
- [23] A. Moshovos and G. S. Sohi. Streamlining Inter-operation Memory Communication via Data Dependence Prediction. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 235–245, Research Triangle Park, NC, USA, December 1997.
- [24] S. Palacharla, N. P. Jouppi, and J. E. Smith. Quantifying the Complexity of Superscalar Processors. CS-TR 1996-1328, University of Wisconsin at Madison, November 1996.
- [25] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 206–218, Boulder, CO, USA, June 1997.
- [26] S. T. Pan, K. So, and J. T. Rahmeh. Improving the Accuracy of Dynamic Branch Prediction using Branch Correlation. In *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 12–15, Boston, MA, USA, October 1992.
- [27] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro Magazine*, 16(4):51–59, August 1996.
- [28] K. Skadron, M. Martonosi, and D. W. Clark. Selecting a Single, Representative Sample for Accurate Simulation of SPECint Benchmarks. TR 595-99, Princeton University Department of Computer Science, January 1999.
- [29] J. E. Smith. A Study of Branch Prediction Strategies. In *Proceedings of the 8th International Symposium on Computer Architecture*, pages 135–148, Minneapolis, MN, USA, May 1981.
- [30] E. Sprangle and D. Carmean. Increasing Processor Performance by Implementing Deeper Pipelines. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 25–34, Anchorage, Alaska, May 2002.
- [31] L. Villa, M. Zhang, and K. Asanović. Dynamic Zero Compression for Cache Energy Reduction. In *Proceedings of the 33rd International Symposium on Microarchitecture*, Monterey, CA, USA, December 2000.

- [32] T.-Y. Yeh and Y. N. Patt. Two-Level Adaptive Branch Prediction. In *Proceedings of the 24th International Symposium on Microarchitecture*, pages 51–61, Albuquerque, NM, USA, November 1991.
- [33] T.-Y. Yeh and Y. N. Patt. Alternative Implementations of Two-Level Adaptive Branch Prediction. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 124–134, Gold Coast, Australia, May 1992.
- [34] T.-Y. Yeh and Y. N. Patt. A Comparison of Dynamic Branch Predictors That Use Two Levels of Branch History. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 257–266, 1993.