

Exploiting Low Entropy to Reduce Wire Delay

Daniel Citron
IBM Haifa Labs
Haifa University Campus
Haifa 31905, Israel
citron@il.ibm.com

Abstract—Wires shrink less efficiently than transistors. Smaller dimensions increase relative delay and the probability of crosstalk. Solutions to this problem include adding additional latency with pipelining, using “fat wires” at higher metal levels, and advances in process and material technology.

We propose a stopgap solution to this problem by applying a decade old technique called *bus-expanding* to the problem. By exploiting low spatial and temporal entropy of data it is possible to transfer m bits of data over a n -bit wide bus in a single cycle ($m > n$). High entropy data will be routed directly over the bus while low entropy data will be compacted using small lookup tables. A table index will be transferred in the case of a successful lookup, otherwise the full value will be transferred in several cycles.

Reducing the number of wires per bus, enables the use of wider wires, which in turn reduces the wire delay. Examination of projected process technologies shows that by shrinking the number of bits in a bus (64 \rightarrow 48) instead of shrinking the individual wires maintains a constant wire delay. Tests on SPEC CPU2000 have shown that for the 64-bit buses leading from the L1 caches to the processor core it is possible to transfer all data types (addresses, integers, instructions and floating-points) using 40-bits per bus on the average.

I. INTRODUCTION

On-chip wires are designed as a hierarchy of metal levels deposited above the silicon gates. The main problem is that as wires shrink with the processor they incur higher resistance, capacitance, and inductance, which cause delays and crosstalk. The problem is emphasized in intermediate and global interconnects [4]. One solution is to use larger wire dimensions by migrating a bus to a higher metal level [11], however this has been shown to increase the number of metal levels [9]. The other widespread solution is to pipeline the affected buses [5] which adds latency.

Our solution reintroduces, in a different context, a decade old technique proposed by Farrens and Park [3] and expanded by Citron and Rudolph [2]. m bits of data are compacted into n bits (width of bus), transmitted, and restored to m bits at the receivers end. If compaction fails the data is transferred in multiple cycles. The key to successful compaction is identifying the levels of entropy in bit fields of the data. Fields with a low level of entropy can be compacted with a high degree of success, fields with a high level of entropy won't be compacted at all and sent directly over the bus.

Figure 1 shows how 64-bit instruction addresses can be transferred over a 32-bit bus by sending the 29 LSBs directly

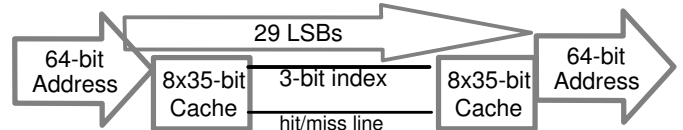


Fig. 1. Transmitting a 64-bit instruction address over a 32-bit bus.

over the bus and caching the 35 MSBs in a 8-entry direct-mapped table. An extra bit is added to the control lines in order to signal the receiving cache if the data is compacted or not (which then requires additional cycles). When an uncompacted address is transferred both caches are updated with this new value.

The original works focused on the junction between processor to off-chip buses in order to alleviate the pressure on pin connections or conform to existing narrow buses. Our work emphasizes a different constraint, the delay caused by narrower wires. In order to correctly gauge the impact of the technique we have to examine the following issues:

- 1) Establish that using less wires is beneficial (section II).
- 2) Measure the degrees of entropy of the data types transferred (section III).
- 3) Measure the compaction success rate of data transferred (section IV).
- 4) Analyze the cost (delay, power, area) of compacting low entropy data.
- 5) Determine the overall performance improvement/degradation (section V).

As this is a novel solution to the wire delay problem we will focus on the first three items in order to set the foundations for the technique.

II. WIRE SCALING: PROBLEMS AND METRICS

Wire attributes such as the κ -dielectric constant, aspect ratio (Height/Width), barrier thickness, sidewall capacitance, and others, influence the speed of wires. Nevertheless, the basic limitation of wires is the RC (Resistance \times Capacitance) delay. Given the resistance and capacitance per unit length of wire (R_w and C_w) and by using repeaters (a common practice), a first level approximation of the wire delay can be reduced to a linear dependency on wire length and be computed as:

$$\tau = 2.13\sqrt{R_w C_w FO1}$$

where the delay of a FO1 (an inverter driving one copy of itself, the repeater) is roughly a third of a FO4 (a “FanOut-of-4” inverter) delay.

Table I shows the projected RC delay for intermediate wires. The resistance and capacitance are computed by the following equations (based on [4]):

$$R_w = \frac{\rho}{(\text{thick} - \text{barrier})(\text{width} - 2\text{barrier})}$$

ρ is the material's resistance, *thick* and *width* are the wire's dimensions and *barrier* is the thickness of the barrier layer.

$$C_w = \epsilon_0(2K\epsilon_h \frac{\text{thick}}{\text{spacing}} + 2\epsilon_v \frac{\text{width}}{ILD_{\text{thick}}})$$

ϵ_0 is the permittivity of free space, ϵ_h and ϵ_v are the horizontal and vertical κ -dielectric constants, *spacing* is the distance between wires, and ILD_{thick} is the dielectric layer thickness. K is the "Miller Multiplication" constant and is set at 2. The data is based on the SIA's technology roadmap [7].

TABLE I

Projected RC delay for current and future process technologies.

Year	2λ nm	W. Pitch nm	R_w m $\Omega/\mu\text{m}$	C_w fF/ μm	FO4 ps/mm	Rpt. RC ps/mm
2003	130	320	139	237	47	48
2005	90	240	248	209	32	50
2007	70	195	353	193	25	51
2010	50	135	739	174	18	59
2013	35	95	1412	164	13	66
2016	25	65	2875	160	9	79

As we can see in table I the RC delay grows, in contrast to the gate delay (computed by 720λ for a FO4) which scales gracefully and decreases proportionately to the gate length. In order to keep the RC delay constant we must keep the wire dimensions constant in the face of a process technology shrink. Figure 2 shows the delay of three entities as a function of the process technology:

- 1) The projected delay of a 64-bit bus (using repeaters).
- 2) The projected delay of a FO4 inverter. The opposing directions of these two lines amplifies the "wire delay problem".
- 3) The wire dimensions are kept constant after a process shrink. The rate of delay is decelerated. In order to keep the total bus area constant less wires are implemented. The numbers shown at the junction points are the number of wires (NW) necessary in order to satisfy the equation:

$$64 \times \text{WirePitch}_{\text{gen}_n} = NW \times \text{WirePitch}_{\text{gen}_{n-1}}$$

The graph shows that by reducing the number of bits by 25-33% it is possible to maintain a constant RC delay after a process shrink. Section III shows that data transferred can suffer this bus narrowing.

Caveat Lector: The results presented in this section contain many assumptions regarding low-level process technology. It is totally possible that the omission of factors such as fringe effects, gate capacitance and resistance, signal propagation, etc., can skew our results by several bits. Even so, we believe that the basic idea behind the technique is still valid. Bus compaction is a micro-architectural scheme that complements process and material design yet transcends any particular technology.

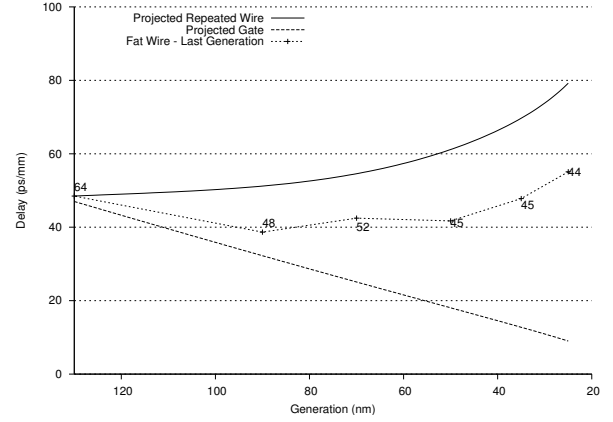


Fig. 2. The projected delay of wires, gates, and last generation fat-wires.

III. ENTROPY OF DATA

The test cases we have chosen are the busses that lead from the on-chip L1 caches to the processor's core. All types of data flow through them and they are prime candidates for bus compaction due to their relative distance from the datapath. Instructions, load/store data (integer and FP), and their addresses for the 64-bit POWER4 [10] architecture were collected from the complete SPEC CPU2000 suite using the MinneSPEC [6] lgred input sets. The C/C++ benchmarks were compiled on a POWER4 running AIX version 5.1 using the IBM compiler `xlc v6.0` with the flags: `-q64 -DSPEC_CPU2000_LP64 -O5`. The Fortran benchmarks were compiled using the `xlf v8.1` compiler with the flags: `-q64 -O5`.

$$E = - \sum_{k=1}^L p_k \log_2(p_k)$$

Entropy is computed using the above equation, L is the number of possible values each datum may represent and p_k is the probability of its appearance. A low entropy suggests that data can be represented with less bits. Figure 3 shows the CFP2000 byte entropy for instructions, data (combined and broken down into integers and FPs) and their addresses. The entropy data for CINT2000 was collected but isn't displayed. Analysis of the entropies suggests the following:

- **Addresses:** The 40 high order bits contain very little information. Address lines can be compacted with a very high degree of predictability.
- **Instructions:** The only borderline candidate for compaction is byte 1. This correlates with the register fields of an instruction.
- **Integer Values:** The entropies of byte 4 suggest that narrowing a bus to 32-bits won't be trivial.
- **FP values:** Only the upper bits of the exponent have a low entropy. Compacting these values will be a challenge.
- **Data Values:** The high entropy of the FP values dominates the total entropy, a solution that somehow differentiates the types must be applied.

The use of low entropy has been suggested before in other works: Brooks and Martonosi [1] represent the static approach: the fact that the high-order bits of integers are mainly all 0s

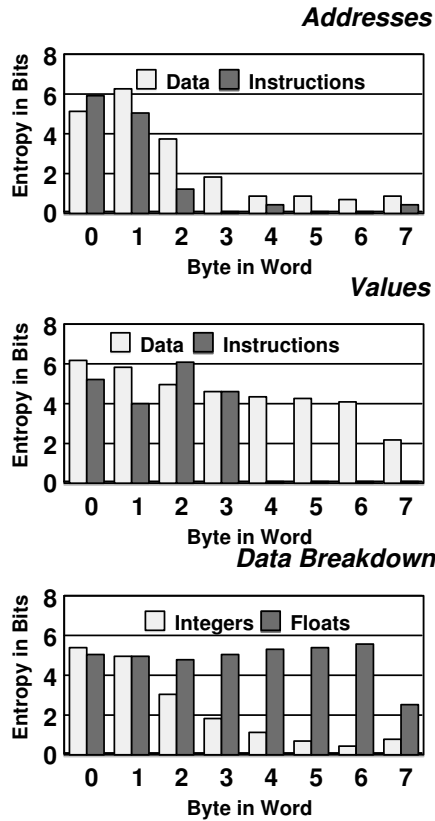


Fig. 3. CFP2000 entropy per byte for instruction and data address and values. The bottom chart breaks down the data into integers and FP values.

or 1s is used to create “narrow-width” pipelines that process only the lower bits, thus saving energy. In the next section we will compare bus-expanding to the narrow-width technique. The dynamic approach is represented by Yang, Zhang, and Gupta [12], who exploit the recurrence of data values to reduce storage requirements of on-chip caches. This technique can be used to augment our design and reduce its size and latency. Our scheme differs from *bus encoding* (of which [8] is a representative) where the polarity of bits is reserved in order to conserve power or reduce crosstalk. We look at bit-fields as a whole and compact them into less bits.

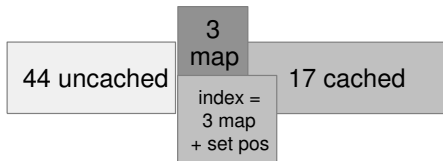


Fig. 4. 44 LSBs are uncached and sent directly over the bus, the next 3 bits are the map bits, and the 17 MSBs are the cached bits. In the case of a hit the index bits are the map + the set position bit.

IV. COMPACTION SUCCESS RATES

The next step in validating the technique is to measure the rate of successful compactions, the *compaction rate*. We chose to reduce a 64-bit bus to 48-bits, our calculations (exhibited in figure 2) show that this is the “sweet point” where keeping

the individual wire widths constant during a process shrink doesn’t widen the total bus.

We will name a compaction table as a *Bus-Expander* (the term used by [2]) and annotate it as a BE. Our first design choice is demonstrated on a 16-entry, 2-way set associative, BE. The 64-bit word is segmented into four fields (figure 4):

- 1) The *uncached* bits. The lower 44-bits are sent directly over the bus.
- 2) The *map* bits. The next 3 bits “map” the set in the BE.
- 3) The *cached* bits, the upper 17 bits that are stored in the BE.
- 4) The *index* bits. The bits that are sent over the bus and mark the exact entry of the cached data in the corresponding BE. They are the 3 map bits and the position bit of the entry in the set.

If the current upper 17 bits match a mapped cache entry a “hit” has occurred and the index bits are sent over the bus and point to the correct entry in the corresponding BE. If the data doesn’t match a “miss” has occurred and the upper 17 bits are sent in the consecutive cycle and the BEs are updated.

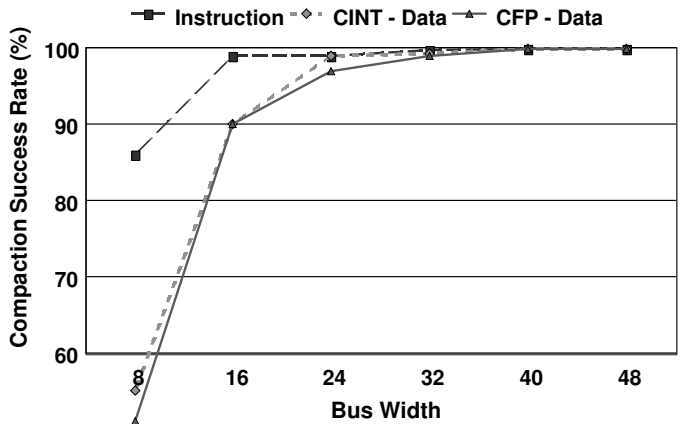


Fig. 5. Address compaction rates of instructions and data using an 8-entry, direct-mapped BE.

As expected addresses compact extremely well: The hit rate for addresses (instructions and data) is very close to 100%. It is possible to refine this and compact instruction addresses to 16 bits and data addresses to 24 bits with the same mechanism (figure 5). Figure 6 shows CFP data (combined, integer, FP) compaction rates of the upper 17 bits using a 16-entry, 2-way, BE. The 55% data compaction rate is a result of the 99% rate for integer values (great) and the 42% rate of the FP values (mediocre). The problem is that the FP values dominate the compaction rate.

TABLE II

Compaction rates compared with narrow-width detection rates. The BEs are 16-entry, 2-way set associative.

	Inst. Addr.	Data Addr.	Integers	Floats
Narrow	0.51	0.18	0.78	0.05
BE	0.99	0.99	0.99	0.40

Implementing narrow-width [1] detection in the BEs design hasn’t been beneficial. FP values and addresses are seldom

narrow-width and integer values compact at such a high rate that the difference isn't noticeable. Table II shows that no benefit is gained by augmenting the BEs with narrow width detection.

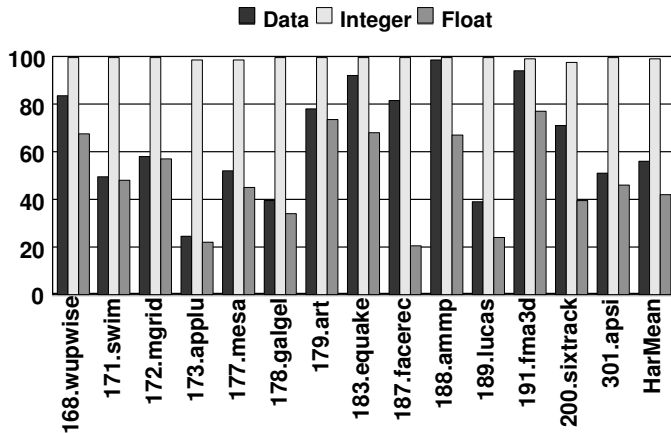


Fig. 6. CFP2000 compaction rates for data values. The 17 high order bits are stored in an 16-entry, 2-way, BE. Mapping is performed using bits 45-47.

The poor FP results can be blamed on several factors: size, associativity, and the number of bits to compact. Further exploration has shown that only when the top 10 bits are compacted do we achieve a relatively high rate of compaction (78% for all data on the CFP suite).

V. CONCLUSIONS

The wire metrics, entropy, and compaction reuse rate presented above lead to the following conclusions:

- 1) Instructions shouldn't be compacted at all.
- 2) Integer and FP data must use separate BEs. In the POWER architecture this differentiation is easily achieved by the opcode of the memory access instruction.
- 3) FP values can be compacted into 54 bits, the very low entropy of addresses enables compaction into 24 bits. This results in a bus width reduction from 128 to 78 bits with a high degree of predictability.
- 4) The instruction busses (address + data) can be similarly reduced to 80 bits by reducing the address bus to 16 bits and leaving the data bus unaltered.

However, several obstacles still have to be overcome. They include the BEs' latency, area, and power consumption, and detailed performance analysis. Preliminary tests have shown that the area required by the BEs is advantageous given the lengths of the busses involved¹. The access time required for compaction at the senders end and uncompression at the receivers end is $0.35 + 0.17 = 0.52ns$ which is larger than the $0.16ns$ bus delay (for a 4mm bus). However, an analysis of a typical L1 data cache and connected Load/Store Unit has shown that the BEs' latencies are overlapped by the cache's mechanisms.

We performed a first-order approximation of performance by comparing the CPU time ($InstructionCount \times CPI \times$

¹Although the area freed by the reduction in bits is used to implement fast wires and the area used by the BEs is implemented at the silicon level.

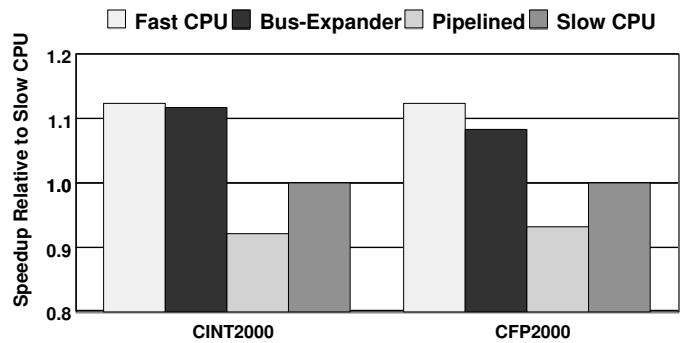


Fig. 7. Geometric means of the CPU time relative to the slow processor.

CycleTime) of several in-order, single-issue, processors. The first is unhindered by wire delay, the second pipelines L1 data cache access (to compensate for the wire delay), the third implements the BE scheme, and the last is slowed down to match the wire delay. It is assumed that all instructions take 1 cycle except memory loads which take 2 cycles in addition to any delay caused by pipelining or unsuccessful compaction (including on the address BEs). The base cycle time is 400ps with the slower processor having a cycle time of 450ps.

Figure 7 shows the geometric means relative to the slow processor of both CPU2000 suites: the BE scheme outperforms the two other limited schemes and is very close to the perfect processor. We are currently testing the technique on a wider range of BE configurations connecting various units, perfecting a power/performance model, and measuring the exact latency and area requirements of implementing bus-expanders.

REFERENCES

- [1] 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*, January 1999, pp. 13–22.
- [2] D. Citron and L. Rudolph, "Creating a Wider Bus using Caching Techniques," in *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, January 1995, pp. 90–99.
- [3] M. Farrens and A. Park, "Dynamic Base register Caching: A Technique for Reducing Address Bus Width," in *Proceedings of the 18th International Symposium on Computer Architecture*, May 1991, pp. 128–137.
- [4] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, April 2001.
- [5] S. Keckler, D. Burger, et al., "A Wire Delay Scalable Microprocessor Architecture for High Performance Systems," in *Proceedings of the 2003 IEEE International Solid State Circuit Conference*, February 2003.
- [6] A. KleinOswski and D. J. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *Computer Architecture Letters*, vol. 1, June 2002.
- [7] Semiconductor Industry Association, "National Technology Roadmap for Semiconductors," 2002.
- [8] M. R. Stan and W. P. Bursleson, "Bus-invert coding for low power i/o," *IEEE Transactions on VLSI*, vol. 3, no. 1, pp. 49–58, March 1995.
- [9] D. Sylvester and K. Keutzer, "Rethinking Deep-Submicron Circuit Design," *IEEE Computer*, vol. 32, no. 11, pp. 25–33, November 1999.
- [10] J. M. Tandler, J. S. Dodson, J. J. S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–26, 2002.
- [11] T. Theis, "The future of interconnection technology," *IBM Journal of Research and Development*, vol. 44, no. 3, pp. 379–390, May 2000.
- [12] J. Yang, Y. Zhang, and R. Gupta, "Frequent Value Compression in Data Caches," in *Proceedings of the 33rd International Symposium on Microarchitecture*, December 2000, pp. 258–265.