Fast, Performance-Optimized Partial Match Address Compression for Low-Latency On-Chip Address Buses

Jiangjiang Liu
Dept. of Comp. Sci.
Lamar University
Beaumont, TX 77710, U. S. A.
E-mail: liu@cs.lamar.edu

Krishnan Sundaresan
Dept. of Elect. & Comp. Eng.
Michigan State University
East Lansing, MI 48824, U. S. A.
E-mail: sundare2@egr.msu.edu

Nihar R. Mahapatra
Dept. of Elect. & Comp. Eng.
Michigan State University
East Lansing, MI 48824, U. S. A.
E-mail: nrm@egr.msu.edu

Abstract— **The influence of interconnects on processor performance and cost is becoming increasingly pronounced with technology scaling. In this paper, we present a fast compression scheme that exploits the spatial and temporal locality of addresses to dynamically compress them to different extents depending upon the extent to which they match the higher-order portions of recently-occurring addresses saved in a very small "compression cache" of capacity less than 500 bits. When a maximal match occurs, the address is compressed to the maximum extent and is transmitted on a narrow bus in one cycle. When a partial match occurs, one or more extra cycles are required for address transmission depending upon the extent of the partial match. To minimize this** *transmission cycle penalty (TCP)***, we use an efficient algorithm to determine the optimal set of partial matches to be supported in our** *partial match compression (PMC) scheme*— **we refer to this scheme as** *performance-optimized PMC (PO-PMC)***. A previously-proposed scheme called bus expander (BE) supports only a single, fixed-size match for compression. We show that all addresses that result in (maximal) matches in BE also result in the same in PMC, but the remaining addresses that are considered "no matches" in BE frequently result in partial matches in PMC, thus helping curtail the latter's TCP significantly. Across many SPEC CPU2000 integer and floating-point benchmarks, we find that average program performance improves by 3% when using PO-PMC compared to that when using BE. Further, we investigate how area slack arising from compression can be exploited for bus latency improvement by increasing inter-wire spacing. We find that, on the average, it can reduce bus latency by up to 84.63% and thereby improve program performance by about 16%.**

I. Introduction

Since on-chip wiring complexity grows exponentially with gate count, chip design is becoming increasingly interconnect-centric with technology scaling. Trends also show that IC pin count is growing at 8% to 11% per year, necessitating more and more on-chip I/O pads, which makes die area pad-limited [1]. Further, wire delays are severely limiting the use of higher clock frequencies in high-performance designs. For instance, in the Pentium 4 microprocessor, designers found that inter-wire capacitive coupling in global wires limited the clock frequency by 50 MHz from the maximum possible for that technology (0.18 μm) [2]. Also, in the Pentium 4 microarchitecture, two out of twenty pipeline stages are devoted exclusively to carrying signals on buses [3]. Thus, wires pose major performance and cost problems as technology scales. This will be more so in the future.

Various approaches have been adopted at different levels of abstraction to keep problems due to wire delay in check. Some of them include: new materials such as copper interconnect and low-K dielectric insulators; high-aspect ratio (tall and thin) wires at the process level; use of timing- instead of area-driven routing algorithms at the layout level; and repeater and flip-flop insertion and simultaneous wire and driver sizing at the circuit level. But, in spite of applying these techniques, wires still pose a major problem for current designs [4]. An effective solution to alleviate these problems may be to consider microarchitecture-level techniques which provide more room for optimization. This paper explores such an approach in which addresses are dynamically compressed and transmitted on narrower buses and area slack arising from compression is exploited to improve bus latency by increasing inter-wire spacing. Consequently, interconnect cost and interconnect and program performance can simultaneously improve.

The organization of the rest of this paper is as follows. We review related work and outline our contributions in Sec. II. Next, in Sec. III, we discuss our proposed technique and its optimization. Then, in Sec. IV, we describe our simulation environment and methodology. In Sec. V, we present results and discussions. Following that, we briefly discuss compression and decompression latency issues in Sec. VI. Finally, we conclude in Sec. VII.

II. Related Work

Address buses have been studied widely in previous work and schemes have been proposed to reduce their energy and/or area/cost. The dynamic compression schemes dynamic base register caching (DBRC) and bus expander (BE) were first proposed in [5], [6]. They use small compression caches at the sending end and register files at the receiving end of a processor-to-memory address bus and were meant to reduce off-chip bus widths and pin counts. They were subsequently used for compressing instruction and data buses in [7].

A recent attempt at using compression to reduce on-chip wire delay was presented in [8]. However, in that work, wire delay reduction was treated in an ad hoc manner and performance impact was estimated with first-order approximation and not using simulations [8]. Most recently, a comprehensive analysis and comparison results for DBRC and BE, from the perspective of optimizing performance, energy, and cost, were presented in [9]. Also, a number of techniques that can be used with BE for address buses to improve energy efficiency when transmitting compressed addresses over narrower-width buses were presented in [10].

A. Contributions

In this paper, we present a fast compression scheme that exploits the spatial and temporal locality of addresses to dynamically compress them to different extents depending upon the extent to which they match the higher-order portions of recently-occurring addresses saved in a very small "compression cache" of capacity less than 500 bits. To minimize the *transmission cycle penalty (TCP)* that results when a maximal match does not occur, we use an efficient algorithm to determine the optimal set of partial matches to be supported in our *partial match compression (PMC) scheme*—we refer to this scheme as *performance-optimized PMC (PO-PMC)*. We show that all addresses that result in (maximal) matches in BE also result in the same in PMC, but the remaining addresses that are considered "no matches" in BE frequently result in partial matches in PMC, thus helping curtail the latter's TCP significantly. Further, we investigate how area slack arising from compression can be exploited for bus latency improvement by increasing inter-wire spacing. Thus by using a combination of compression and wire spacing, interconnect cost and interconnect and program performance can be simultaneously improved. Our work is perhaps the first to examine the impact of compression and wire spacing on overall program performance using a realistic simulator and real-world benchmark programs.

III. Partial Match Compression and Its Performance Optimization

In the next subsection, we first describe how bus compression works and discuss the shortcomings of existing schemes in order to motivate our new techniques which will be discussed after that. The details of previous bus compression techniques can be found in [5], [6], [9], [10].

A. Limitations of Previous Bus Compression Schemes

The structure and working of a dynamic bus compression technique, such as BE, is explained next. As shown in Fig. 1(a), the higher-order portion of an address is compressed by the compressor which is implemented as a small compression cache. The lower-order portion, which is not very compressible due to its highly-varying nature (*U field* in Fig. 1(b)), is transmitted as is on the compressed bus. The width of the compressed bus is equal to the width of the compressed address. At the receiving end, the original address is retrieved by looking up a register file present in the decompresser hardware. The small compression cache stores the higher-order portion, called *tag field* (*T field* in Fig. 1(b)), of recently transmitted addresses in its tag-RAM.

To compress addresses, a set of bits from the incoming address, called the *index* (*I field* in Fig. 1(b)), is used to search the compression cache. Note that, in this paper, we always refer to tag and index fields for the compression cache, and this is not related to the tag and index fields of conventional (L1 or L2) caches in the memory system. If the compression cache is a-way set-associative, then one of a tags stored in the set indexed by the I field can potentially fully match the tag from the incoming address, and provide the *way bits* (*W field* in Fig. 1(b)).

In the case of a hit, the I and W fields, along with the *hit control bit* (the 1-bit C_H *field* in Fig. 1(b)), which is 1 for a hit, and the U field will form a compressed address with a combined bit width less than the original. The compressed bus width is equal to the sum of the widths of the C_H, I, W, and U fields. So in the hit case, there is no performance penalty due to the transmission of the compressed address. In case of a miss at the sending end in the compression cache, the tag corresponding to the least recently used (LRU) entry in the set indexed by the I field is replaced by the tag of the new address. The C_H field, which is 0 for a miss, and the entire address (starting from the higher-order to lower-order bits), which is wider than the compressed bus width, is transmitted in more than one cycle. The decompresser will be updated at the receiving end accordingly. The miss in the compression cache causes transmission cycle penalty for transmission because extra cycles will be needed for transferring the control bits and also the entire address.

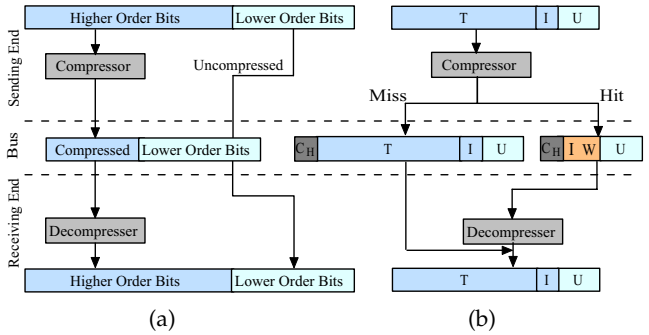

Fig. 1. Bus Compression Scheme: (a) General organization of a bus compression scheme. (b) Figure depicting how compressed addresses are formed.

In our earlier work, we applied BE to the L1→L2 address bus with the simulation setup described in Sec. IV

and reported the optimal compression cache parameters and proposed a number of transmission techniques to improve the original BE address compression scheme by considering trade-offs between performance, on-chip and off-chip energy consumption, and cost [9], [10]. We shall refer to this as *optimized BE (O-BE)* and will use it as the basis for comparison. The L1→L2 address bus considered for compression has an uncompressed width of 38 bits. Results for optimized BE are summarized in Table I. Results for three categories of bus widths were reported: *narrow* (compressed buses with 8, 10, or 12 lines), *medium* (compressed buses with 14 to 16 lines), and *wide* (compressed buses with 19 to 32 lines). The same compression cache parameters will be used in our experiments to collect comparison results.

B. Partial Match Address Compression

In this subsection, we describe the partial match compression (PMC) scheme [11]. As the name suggests, partial matches are allowed in the compression cache. Thus, the longest match between the tag portion stored in the compression cache and the tag portion of the incoming address is used for compression. Since there is more redundancy in the higher-order portion of the address, k possible groups of bits, ending at the most significant bit (MSB) of the incoming address, are considered as candidates for partial match (shown in Fig. 2 as partition 1, partition 2, and partition 3, for $k = 3$). Note that partition 0 corresponds to complete hit (or the hit case in BE) and partition $k+1$ corresponds to complete miss (or the miss case in BE). There are three main hit/miss cases in PMC as explained next.

1) **Complete hit:** If the tag matches fully, PMC performs exactly the same as the BE scheme studied in prior work [7], [9], [10]. The hit control bit C_H is set 1 to indicate a complete hit. If not, it is set 0.
2) **Partial hit:** If a partial match in any of the k groups occurs, the partial match control bit pattern corresponding to the longest match is transmitted to indicate a partial hit. For example, when $k = 3$, partial match control bit pattern C_1C_0 will be 00, 01, or 10 for partition 1 hit, partition 2 hit, or partition 3 hit, respectively. The remaining portion of the higher-order part of the address (that did not match the tag), T_{miss}, is sent in uncompressed form as is the lower-order portion of the address. In this case, less bits are transmitted over the compressed bus than would have been in optimized BE; this reduces the transmission cycle penalty.
3) **Complete miss:** In case of a complete miss in PMC compression cache, when none of the partial matches succeed, the entire address is sent along with the control bits C_H and $C_1C_0 = 11$. Even though more bits need to be transferred in PMC than in the case of optimized BE—due to the partial match control bits for indicating a complete miss—the performance will not be affected much because the number of complete misses will be much less in PMC than in the case of optimized BE.

It is to be noted that PMC cache's tag field is updated whenever complete matches do not occur, which is the same as in the BE compression cache. So the tags stored in PMC and optimized BE compression caches are identical when running the same program. Hence, the compression cache power consumption and latencies are also likely to be identical for similar-sized units.

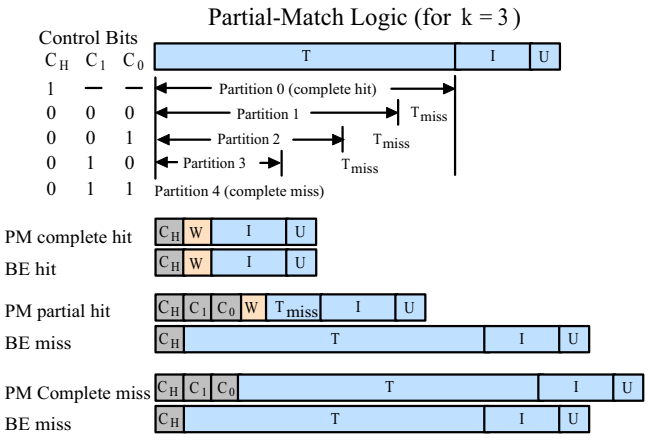

Fig. 2. Partial Match Address Compression.

C. Optimizing Performance

In this subsection, we discuss how we partition the tag field to obtain the best performance. For performance optimized partial match compression (PO-PMC), less cycles required for transmitting an address will lead to less performance penalty. Therefore, in PO-PMC, we choose a combination of partitions that minimizes the *average transmission cycle penalty (ATCP)* across n benchmark programs, which is given by:

$$\text{ATCP} = \frac{\sum_{i=1}^{n} \text{CTCP}_i}{n};$$

here, $CTCP_i$ is the transmission cycle penalty for a combination of partitions for benchmark program i. For a combination of k partitions for any benchmark:

$$\text{CTCP} = \sum_{j=1}^{k+1} (\text{TCP}_j \times \text{MF}_j),$$

where TCP_j is the transmission cycle penalty for the jth partition (i.e., the number of extra cycles required for transmission due to partial hit or complete miss, including the unmatched tag portion and control bits) and MF_j is the match frequency for the j^{th} partition—partitions are numbered from 0 (corresponding to the complete hit case and the largest partition with width

TABLE I
COMPRESSION CACHES AND COMPRESSED BUSES FOR ADDRESS COMPRESSION USING BE SCHEME. ALL COMPRESSION CACHES ARE 2-WAY SET-ASSOCIATIVE.

	Compressed Bus Width (Original Address Width: 38 bits)								
	Narrow			Medium		Wide			
	8	**10**	**12**	**14**	**16**	**20**	**24**	**28**	**32**
Index (I field) (bits)	1	2	3	2	2	1	1	1	1
Tag (T field) (bits)	32	30	28	26	24	20	16	12	8
Miss Rate	0.39	0.28	0.21	0.15	0.10	0.00	0.00	0.00	0.00
Miss Penalty (cycles)	4	3	3	2	2	1	1	1	1
Perf. Penalty(%)	4.98	2.34	1.62	0.83	0.46	0.07	0.00	0.0	0.0
Comp. Cache Size (bits)	156	256	480	224	208	88	72	56	40

equal to the tag field width) to $k+1$ (corresponding to the complete miss case and the smallest partition with width equal to 0) as shown in Fig. 2. In order to determine the optimal combination of partitions to use in PO-PMC for a given compressed bus width, we consider all possible values for the number of partitions k. For each value of k, the TCP value for any partition can be easily determined based on the number of extra bits to be transmitted in case of a partial match at that partition—these bits are the unmatched tag bits and the extra control bits. The only other set of values needed to determine the combination of partitions to use in PO-PMC are the partition MF values; the way we determine these is discussed next.

Let the jth partition's least significant bit (LSB) position within the tag be denoted by LSB_j; in this discussion, LSB positions of partitions are given relative to the LSB of the tag field, which mean $\text{LSB}_0 = 0$. It is clear that MF_j depends upon both LSB_j and LSB_{j-1}, since only those PMC cache references that do not result in a match in the $(j-1)$th or earlier partition may result in a match at the jth partition. To determine the MF values of partitions in a general implementation of PMC, we consider and simulate a special "maximal" partitioning scenario in PMC in which there are as many partitions as possible, i.e., partition j is one bit wider than partition $j+1$ and the number of partitions (including complete hit and complete miss cases) is equal to one more than the width of the tag field; this version of PMC is denoted as *MAX-PMC*. From this simulation, we obtain the match frequencies of all possible partitions and denote the match frequency—referred to as individual match frequency in this special maximal partitioning scenario—of the jth partition as IMF_j. Once the IMF values are obtained from MAX-PMC, the MF value of the jth partition in a general implementation of PMC can be determined from:

$$\text{MF}_j = \sum_{l=\text{LSB}_{j-1}+1}^{\text{LSB}_j} \text{IMF}_l,$$

where IMF_l is the individual match frequency of the partition starting at the lth bit and ending at MSB of the tag in MAX-PMC. If the entire tag field in the current address hits in the PMC cache, it is a complete hit. If the matching length is greater than 0 and less than the tag field width, it is a partial hit. In the partial hit case, out of all entries in a set pointed to by the address index, the entry that has the longest matching length is the hit entry. If the matching length is 0, it is a complete miss. For example, Fig. 3 lists the IMFs and TCPs of all possible partitions for an 8-bit compressed bus for MAX-PMC.

We can see why PO-PMC can achieve lower miss rate and transmission cycle penalty compared to O-BE. In a scheme like DBRC, BE, or O-BE, there are only two outcomes for a compression cache lookup: hit and miss. So even if the tag field from the incoming address and the stored tag differ in one bit, a miss occurs and the entire tag field will be transmitted over the compressed bus in multiple cycles. In PO-PMC, only the unmatched tag bits need to be transferred if a partial hit occurs during the lookup, and this reduces the number of extra bits needed to be transferred. For example, for the 8-bit compressed bus, in which the higher-order 32 bits are used as tag for compression, the complete hit rate (column labeled 0^{th} in Fig. 3) is 68.15% for both O-BE and PO-PMC, which means the complete miss rate for O-BE is 31.85%. However, in PMC, if $\text{LSB}_1 = 1$, 9% of the misses in O-BE can be eliminated by partial hit at the first partition ($\text{IMF}_1 = 2.83\%$ in column labeled 1^{st} in Fig. 3), and if $\text{LSB}_1 = 5$, 30% of the misses can be eliminated by partial hit at the first partition ($\sum_{l=1}^{5} \text{IMF}_l = 2.83\% + 1.95\% + 1.39\% + 1.56\% + 1.98\% = 9.71\%$ in columns labeled $1^{st}, 2^{nd}, 3^{rd}, 4^{th}$, and 5^{th} in Fig. 3). As is evident from Fig. 3, a complete miss will be a very unlikely event when partial hits are considered. Therefore, if PMC is applied with $\text{LSB}_1 = 5$, just 5 tag bits and control bits need to be transferred for 30% of the complete misses in O-BE, which is much fewer than the original tag field width of 32 bits.

We devised an efficient algorithm to determine the optimal combination of partitions to minimize ATCP in PO-PMC. Table II lists partitions, their LSBs, and corresponding transmission cycle penalties for each compressed bus width for which we obtained performance-optimized designs.

TABLE II

PARTITION POINTS FOR PO-PMC: LSB IS THE PERFORMANCE-OPTIMIZED PARTITION LSB. TCP IS THE TRANSMISSION CYCLE PENALTY THAT IS INCURRED IN THE CASE OF PARTIAL HIT/MISS.

Compressed bus width (bits) [Bus cost savings] [Tag width] (bits) Original bus width: 38 bits									
8 [79%] [32]		10 [74%] [30]		12 [68%] [28]		14 [63%] [26]		16 [58%] [24]	
LSB	TCP	LSB	TCP	LSB	TCP	LSB	TCP	LSB	TCP
6^{th}	2	9^{th}	2	11^{th}	2	13^{th}	2	15^{th}	2
14^{th}	3	18^{th}	3	28^{th}	4	26^{th}	3	24^{th}	3
22^{nd}	4	30^{th}	5	-	-	-	-	-	-
32^{nd}	6	-	-	-	-	-	-	-	-
18 [53%] [22]		19 [50%] [21]		20 [47%] [20]		24 [37%] [16]		32 [16%] [8]	
LSB	TCP	LSB	TCP	LSB	TCP	LSB	TCP	LSB	TCP
17^{th}	2	18^{th}	2	20^{th}	2	16^{th}	2	8^{th}	2
22^{nd}	3	21^{st}	3	-	-	-	-	-	-

32-bit Tag Field (Compressed Bus Width: 8 bits; Bus Cost Savings: 79%)									
LSB	**0th**	**1st**	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**
IMF	68.15%	2.83%	1.95%	1.39%	1.56%	1.98%	2.42%	2.95%	2.56%
TCP	0	1	1	1	1	1	1	1	1
LSB	**9th**	**10th**	**11th**	**12th**	**13th**	**14th**	**15th**	**16th**	**17th**
IMF	4.05%	6.23%	1.95%	1.11%	0.43%	0.00%	0.00%	0.19%	0.00%
TCP	2	2	2	2	22	2	2	2	3
LSB	**18th**	**19th**	**20th**	**21st**	**22nd**	**23rd**	**24th**	**25th**	**26th**
IMF	0.00%	0.00%	0.00%	0.25%	0.00%	0.00%	0.00%	0.00%	0.00%
TCP	3	3	3	3	3	3	3	4	4
LSB	**27th**	**28th**	**29th**	**30th**	**31st**	**32nd**			
IMF	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%			
TCP	4	4	4	4	4	5			

Fig. 3. Individual Match Frequencies and Transmission Cycle Penalties for All Partitions in MAX-PMC. LSB refers to the position of the LSB of a partition. In MAX-PMC, LSB of the jth partition is j. TCP refers to the transmission cycle penalty for a partition.

IV. METHODOLOGY

We used *sim-alpha*, the validated Alpha 21264 simulator based on the SimpleScalar tool, as the platform for our experiments [12] and benchmarks from the SPEC CPU2000 suite. We ran the benchmark programs using reference input sets provided with the SPEC2000 suite and to limit the execution times of our simulations we used a methodology similar to the one described by Skadron, et al. [13]. Their research shows that accurate simulation results can be obtained by avoiding unrepresentative behavior at the beginning of a benchmark program's execution and by using a single, short simulation window of 50 million instructions. In our experiments, we simulate (but do not collect results for) instructions before the representative segment (warm-up window) and use a sampling window of 50 million instructions to collect our results. The sizes of the warm-up windows are also different for different SPEC programs [13]. We simulated address compression and transmission and collected performance and bus energy results for 50 million committed instructions for 7 integer benchmarks and 7 floating-point benchmarks randomly chosen from the SPEC CPU2000 suite: *gcc*, *gzip*, *parser*, *vpr*, *twolf*, *mcf*, *crafty*, *applu*, *swim*, *wupwise*, *lucas*, *art*, *ammp*, and *equake*.

We implemented our bus compression scheme for the 38-bit address bus between the L1 and L2 caches, in a memory hierarchy with two levels of caches and a main memory. In our simulated system, the L1→L2 address bus carries physical addresses and both instruction and data addresses are carried on the same bus. In order to compare our design with optimized BE, we used the optimal compression cache parameters from our earlier work [9], summarized in Table I, for different compressed buses ranging from 8 to 32 bits.

The execution time of a benchmark program running on the target system was collected in terms of processor cycles. Since we used an execution-driven simulator, our calculation of performance included any latencies due to pipeline stalls too and not just the latencies due to address transmission. We report the *performance improvement (PI)* for n benchmarks which is defined as follows:

$$PI = \frac{\sum_{i=1}^{n}(t_{i,O-BE} - t_{i,PO-PMC})}{\sum_{i=1}^{n} t_{i,O-BE}},$$

where $t_{i,PO-PMC}$ is the total time (processor cycles) for execution of program i on the modified target system with PO-PMC and $t_{i,O-BE}$ is the total time for execution of the same program on the target system with O-BE.

A. Delay Model

Next, we describe the wire delay model used in this work. Delay characteristics of buses depend on self and coupling transitions occurring in individual wires and pairs of adjacent wires, respectively, constituting the bus. Self transitions are of two types: charge ($0 \rightarrow 1$) and discharge ($1 \rightarrow 0$), and coupling transitions in a pair of adjacent wires are of three types: *coupling charge*

transitions ($00 \rightarrow 01$[1], $00 \rightarrow 10$, $01 \rightarrow 11$, and $10 \rightarrow 11$), *coupling discharge transitions* ($01 \rightarrow 00$, $10 \rightarrow 00$, $11 \rightarrow 01$, and $11 \rightarrow 10$), and *toggle* transitions ($01 \rightarrow 10$ and $10 \rightarrow 01$).

Due to inter-wire capacitive coupling, the propagation delay of the k^{th}-wire in a bus, which is a function of transitions in its neighboring wires $k-1$ and $k+1$, can be expressed as follows [1]: $t_{p,k} = g \cdot C_w \cdot (0.38R_w + 0.69R_D)$, where C_w is the self capacitance (capacitance to ground) and R_w and R_D are the resistances of the wire and driver, respectively; g is the *delay correction factor* due to crosstalk between adjacent wires separated by the minimum spacing distance and is a function of the *capacitance ratio* $r = C_c/C_w$ and adjacent wire activity (see Table III), where C_c is the coupling capacitance between two adjacent wires.

TABLE III
WIRE DELAY CORRECTION FACTORS.

$k-1$,k,$k+1$	Delay factor (g)
↑,↑,↑	1
↑,↑,–	1+r
↑,↑,↓	1+2r
–,↑,–	1+2r
–,↑,↓	1+3r
↓,↑,↓	1+4r

Note that the value of r depends on technology and the layer of metal being considered. For our simulations, we used $r \approx 5$ which was obtained for topmost metal layer (M6) in TSMC 0.18 μm technology considering the substrate layer as the ground plane. The effect of non-standard wire spacings can be captured in the above model by introducing a *spacing correction factor* for r in the above equations and Table III. Thus, if v_0 is the minimum allowable spacing between two wires and v is the new spacing ($v \geq v_0$), then the new capacitance ratio will be: $r \cdot (\frac{v_0}{v})$. In this case, the worst-case delay of a wire (which occurs when there are toggles on both adjacent coupling capacitances of the wire–bottommost row in Table III) will improve by a factor of $\frac{1+4r}{1+4r \cdot (\frac{v_0}{v})}$.

V. RESULTS AND DISCUSSION

In this section we present results showing how PO-PMC performs compared to bus expander proposed in prior work.

A. Performance Comparison of PO-PMC and O-BE

We report the transmission cycle penalties incurred during O-BE and PO-PMC address compression to facilitate a side-by-side comparison. Since there are several partitions in PO-PMC, as listed in Table II, the PO-PMC transmission cycle penalty is calculated as the weighted average of the cycle penalties of all partitions (the weights depending upon the frequencies of different partial hits) for a given bus width. As shown in Fig. 4(a), compared to O-BE, the transmission cycle penalty can be reduced by 50%-66% with PO-PMC, especially for narrower compressed buses. Next, in Fig. 4(b), we show the average percentage of extra cycles taken to complete program execution for O-BE and PO-PMC compared to that for the original default system without compression for different bus widths. Again, as we see, the performance overhead due to compression reduces significantly (0.13% – 3.13%) by using PO-PMC instead of O-BE, especially at narrower bus widths. Since, there was not much improvement for wider compressed buses, these results are not shown. This is expected since PO-PMC is designed for improving compression cache performance for narrower compressed buses for which O-BE's miss rates are significant.

B. Bus Latency vs. Bus Area

Using compressed buses grants an extra degree of freedom while performing global wire routing for high-performance designs. Common optimizations like *net shielding* (inserting power or ground wires on both sides to protect a wire on the critical path from inter-wire couplings) and *soft spacing* (a technique that automatically maximizes spacing between tightly packed wires within given area constraints) can be greatly facilitated by using compressed buses. Such optimizations go a long way in achieving signal integrity and timing closure in current nanometer designs [4]. Although these techniques have been used in the VLSI design community for a long time, our work is the first to examine their implications in the context of compressed buses. In the simplest application of wire spacing in bus compression, the individual wires can be spaced further apart while maintaining the same area footprint as the original bus. Due to the increased inter-wire distance, coupling capacitances will have lower values. Also, wire spacing involves no additional cost.

The reduction in peak crosstalk delay (bus latency) with wire spacing applied is plotted for different compressed bus widths in Fig. 5. The bus delay model described in Sec. IV-A was used. In Fig. 5, the vertical bars denote wire delay ratios for the compressed bus using the proposed PO-PMC compression scheme and the x-axis labels (20%, 30%, . . ., 100%) denote how much of the area footprint of the original bus is used for the compressed bus as a whole (100% means that the original and the compressed buses occupy the same total routing area). Since the delay ratio for the compressed bus with minimum spacing is unity, it is not reported in this figure. The results for all compressed buses we considered, 8-bit, 10-bit, 12-bit, 14-bit, 16-bit, 19-bit, 20-bit, 24-bit, and 32-bit, show that bus delay can be reduced, on the average, when the original bus area is used to do wire spacing, by about 84.63% for narrow

[1]For two lines i and j, this notation represents the transition: $V_i^{initial} V_j^{initial} \rightarrow V_i^{final} V_j^{final}$.

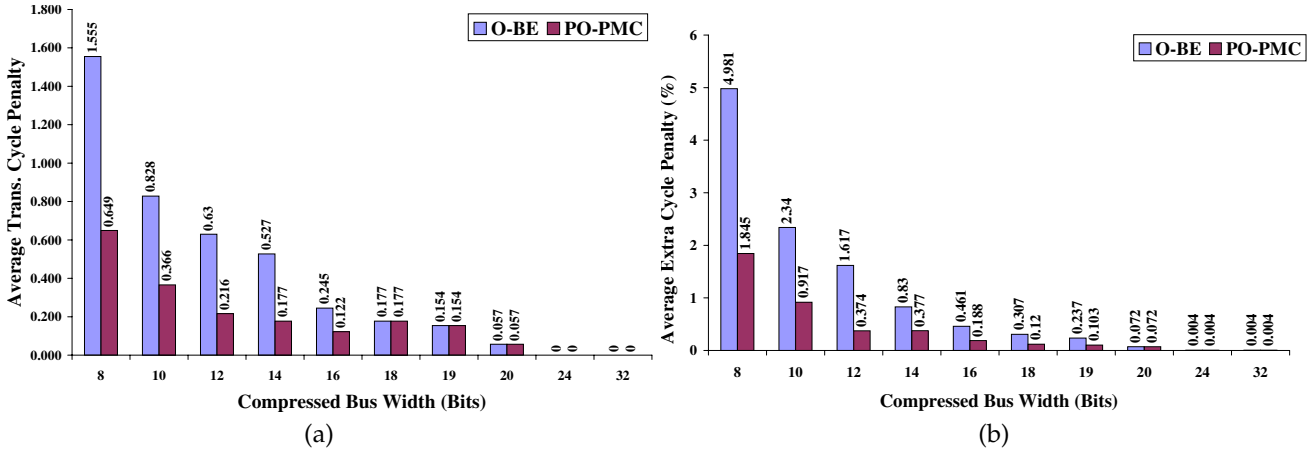

Fig. 4. Performance Comparison of PO-PMC and O-BE Address Compression Across Different Bus Widths: (a) Average transmission cycle penalty. (b) Average extra cycle penalty.

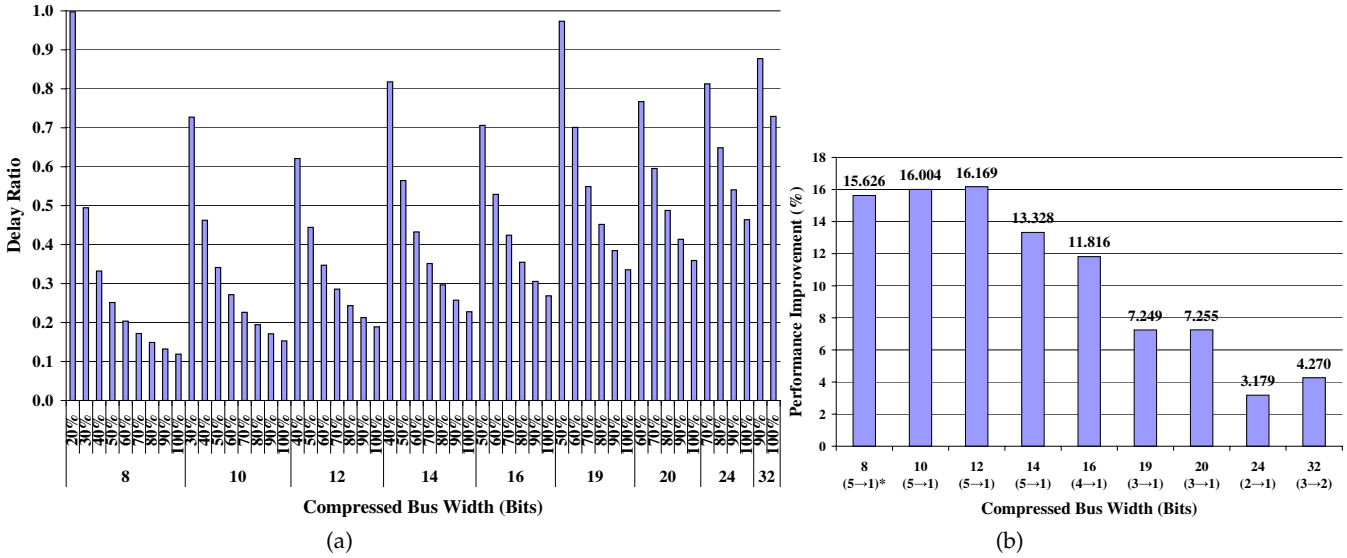

Fig. 5. (a) Delay Variation with Wire Spacing Across Compressed Bus Widths for Performance-Optimized Partial Match Compression. (b) Performance Improvement with Wire Spacing Across Compressed Buses for PO-PMC. (*($i \rightarrow j$): The L1→L2 address bus is assumed to have a latency of i CPU cycles in the default system without compression and j CPU cycles for the compressed bus in the target system with PO-PMC based on the wire delay ratio.)

(8 to 12 bits) buses, 75.18% for medium (14 to 16 bits) buses, and by about 52.80% for wide (19 to 32 bits) compressed buses using PO-PMC with wire spacing. A maximum of 88% bus delay reduction can be obtained for an 8-bit bus. Fig. 5(b) shows the performance improvement that is obtained when PO-PMC with wire spacing (corresponding to "100%" in Fig. 5(a) is used. If the entire area footprint of the original bus is used to increase wire spacing, on the average, performance can be improved by up to 16% compared to the default system without any compression (e.g., for compressed bus width equal to 12 in Fig. 5(b)). Here, the L1→L2 address bus is assumed to have a latency of i CPU cycles in the default system without compression and j CPU cycles for the compressed bus based on the delay ratio for the bus width in Fig. 5 when PO-PMC is used.

VI. Compression and Decompression Latencies

When compression/decompression schemes are used in buses, a major concern is the impact on latency. We explain next why compression/decompression latencies will be minimal and may even be lower than bus encoding/decoding latencies in actual design. We use the simplest low power encoding scheme, bus-invert (BI) [14], as a representative scheme to justify this. In BI, encoding consists of three steps. First, the Hamming distance is computed. This step requires a constant time operation for bitwise XOR and $O(m)$ to $O(\log m)$ time

for counting transitions depending upon the counter structure used, where m is the number of bits in the address. In the second step, the Hamming distance is compared with $\frac{m}{2}$ to check which is greater; this can be completed in $O(m)$ to $O(\log m)$ time, again depending on the hardware structure used. Finally, the current pattern is inverted or sent as-is and this takes constant time. Thus, BI encoding takes at least $O(\log m)$ time. In contrast, our optimized address compression scheme, which uses a 480-bit compression cache, has an access latency of well under one cycle according to CACTI estimates [15]. Hence, compression schemes are more performance-efficient compared to encoding.

VII. Conclusion and Future Work

In this work, we presented techniques to help reduce the growing impact of interconnects on wire delay and cost of systems. By using a new performance-optimized partial-match compression (PO-PMC) technique that uses a small cache of size not bigger than 500 bits at the sending end and registers at the receiving end, we compress addresses dynamically. This helps reduce the number of on-chip address-carrying lines significantly. In order to achieve best performance, we devised an efficient partitioning algorithm for our compression scheme, called Min-ATCP, which takes all possible partial match cases into account and provides the best partial match partition solution for performance optimization. We showed that compression cache miss rate, wire delay, and costs can also be reduced significantly using our technique. Compared to BE, average program performance can be improved by up to 3% when addresses are compressed with our technique. We also investigated wire spacing to exploit the area slack created due to compression, and found that, on the average, it can reduce bus latencies by up to 84.63% and consequently improve program performance by about 16%.

The PO-PMC scheme presented in this work is somewhat tailored to address buses. However, the scheme can be modified and applied to instruction and data buses, too. We chose to apply our scheme only to address buses in this work to demonstrate its effectiveness and the utility of our methodology. Even otherwise, schemes proposed in this work are applicable to instruction and data buses. Also, even better results can be obtained if PO-PMC can be tailored to the characteristics of the information streams that it operates on. In future work, we intend to develop such schemes for instruction and data buses.

Acknowledgment

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

References

[1] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, Second edition*. Prentice-Hall, 2003.

[2] P. Green, "A GHz IA-32 Architecture Microprocessor Implemented on 0.18 μm Technology with Aluminum Interconnect," in *Digest of Technical Papers, IEEE International Solid-State Circuits Conference*, Feb. 2000, pp. 98–99.

[3] R. Kumar, "Interconnect and Noise Immunity Design for the Pentium 4 Processor," *Intel Technology Journal, 1^{st} Quarter 2001*, 2001.

[4] L. Lev and P. Chao, "Down to the Wire: Requirements for Nanometer Design Implementation," White Paper, Cadence Design Systems Inc., 2002.

[5] A. Park and M. Farrens, "Address Compression through Base Register Caching," in *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, Nov. 1990, pp. 193–199.

[6] M. Farrens and A. Park, "Dynamic Base Register Caching: A Technique for Reducing Address Bus Width," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, May 1991.

[7] D. Citron and L. Rudolph, "Creating a Wider Bus Using Caching Techniques," in *Proceedings of International Symposium on High Performance Computer Architecture*, Jan. 1995, pp. 90–99.

[8] D. Citron, "Exploiting Low Entropy to Reduce Wire Delay," *Computer Architecture Letters*, vol. 3, Jan. 2004.

[9] J. Liu, K. Sundaresan, and N. R. Mahapatra, "Dynamic Address Compression Schemes: A Performance, Energy, and Cost Study," in *Proceedings of International Conference on Computer Design*, Oct. 2004, pp. 458–463.

[10] ——, "Energy-Efficient Compressed Address Transmission," in *Proceedings of the 18^{th} International Conference on VLSI Design*, Jan. 2005, pp. 592–597.

[11] ——, "A Fast Dynamic Compress Scheme for Low-latency On-Chip Address Buses," in *Proceedings of the Fourth Annual Workshop on Memory Performance Issues*, Feb. 2006.

[12] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator," The University of Texas at Austin, Department of Computer Sciences, Tech. Rep. TR-01-23, 2001.

[13] K. Skadron, P. Ahuja, M. Martonosi, and D. Clark, "Selecting a Single, Representative Sample for Accurate Simulation of SPECint Benchmarks," *IEEE Transactions on Computers*, vol. 48, no. 11, pp. 1260–1281, Nov. 1999.

[14] M. Stan and W. Burleson, "Bus-Invert Coding for Low-power I/O," *IEEE Transactions on VLSI Systems*, vol. 3, pp. 49–58, Mar. 1995.

[15] P. Shivakumar and N. Jouppi, "CACTI 3.0: An Integrated Cache Cycle Timing, Power, and Area Model," Compaq Western Research Laboratory, Tech. Rep. WRL Research Report 2001/2, Aug. 2001.