Dynamic Base Register Caching: A Technique for Reducing Address Bus Width

Matthew Farrens

Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-9678
fax: (916) 752-4767
email: farrens@cs.eecs.ucdavis.edu

Arvin Park

Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-5183
fax: (916) 752-4767
email: park@cs.eecs.ucdavis.edu

Abstract

When address reference streams exhibit high degrees of spatial and temporal locality, many of the higher order address lines carry redundant information. By caching the higher order portions of address references in a set of dynamically allocated base registers, it becomes possible to transmit small register indices between the processor and memory instead of the high order address bits themselves. Trace driven simulations indicate that this technique can significantly reduce processor-to-memory address bus width without an appreciable loss in performance, thereby increasing available processor bandwidth. Our results imply that as much as 25% of the available I/O bandwidth of a processor is used less than 1% of the time.

1. Introduction

Advances in VLSI technology have increased both the amount of circuitry that can be integrated onto a chip and the speed of this circuitry. However, these increases have not been matched by a proportionate increase in the number of pins that connect a VLSI chip to external circuitry. As the speed and number of transistors per pin has increased, the ratio of computational power to number of I/O pins has grown dramatically. Available bandwidth, and not computational speed, will be the main constraint to increased processor performance in the future.

One way to reduce the impact of bandwidth constraints on processor performance is to increase the amount of *useful* information transferred per I/O pin per unit time. By recoding the information to be transferred, it is often possible to eliminate redundant and/or useless information. For example, most addresses are associated with instruction accesses, which tend to be sequential. These sequential references generally differ only in the low order bits, with the higher order bits remaining unchanged; the high order bits contain essentially *redundant* information. Furthermore, as address spaces and their corresponding address words have grown in size from 16 to 24 to 32 bits, so has the percentage of a given address word containing redundant information. Recoding addresses to minimize this information redundancy has the potential to significantly reduce the bandwidth constraint problem.

A recoding scheme that reduces the number of address lines necessary to transfer addresses between a processor and memory can improve computer system performance in a number of ways. For example, assuming a fixed number of I/O pins, such a reduction may make it possible to place more than one processor on a single chip. In addition, eliminating unnecessary address lines makes it possible to increase the number of data lines between processor and memory, allowing more data to be moved between processor and memory in a single transfer operation. Decreasing the number of address lines also makes it possible to transmit both data and instruction addresses in parallel. Instead of a single large address bus for both instruction and data addresses, two smaller busses, one for reduced instruction addresses and one for reduced data addresses, can operate in parallel potentially increasing throughput between processor and memory.

In order to reduce the address line count, we developed a technique we refer to as *Dynamic Base Register Caching*. This paper will present a detailed description of the effectiveness of this technique, starting in section 2 with a brief background of related work. Section 3 describes Dynamic Base Register Caching in more detail, and in section 4 the traces used in this study and the simulations performed are described. Section 5 contains a discussion of the results of the simulations, which is followed in section 6 by the development of a metric called Address Line Utilization.

2. Background

A study by Hammerstrom and Davidson [HaDa77] suggests that the information content of address reference

---

This work was supported by the National Science Foundation under Grants CCR-90-11535, CCR-90-122800, and by the University of California under MICRO Grant No. 89-165.

streams tends to be very low. They derive estimates from traces of numerical IBM 360 programs that indicate the average information content of an address word is often less than two bits. (The information content of address words varies considerably from program to program, obviously; a program that generates completely random references distributed uniformly across a large address space conveys a great deal of information in each address word, while a program that generates a simple deterministic pattern of address references conveys very little.)

Hammerstrom and Davidson proposed the use of a set of base registers to increase the information content of address words. Under this scheme a *compacted* address word consists of a pairing of an index to a base register and an offset from the register. A complete address is formed by adding the base register to the offset. Base registers are managed with special instructions, which are used to update base registers as a processor's working set changes over time. Hammerstrom and Davidson show that a significant reduction in the number of address lines needed is possible using this scheme.

A similar study was performed by Pleszkun, Rau and Davidson [PlRD81]. They propose an *address prediction stack*, which features a stack of commonly referenced blocks of sequential addresses. If an address is contained in this stack, a small stack location descriptor can be sent from processor to memory instead of the entire address. Like the Hammerstrom and Davidson study, the Pleszkun, Rau and Davidson study analyzes the effectiveness of the address prediction stack on traces of numerical FORTRAN codes running on an IBM 360 architecture. The authors explored a variety of different stack configurations, and evaluated the effectiveness of each. No attempt was made to comment on how difficult each of the schemes would be to implement, however.

3. Dynamic Base Register Caching

Dynamic Base Register Caching is a technique to reduce the number of lines necessary to transfer addresses to main memory. In this scheme, hardware divides each address word into two components, a high order component (a Base) and a low order component (an offset). The low order component of the address is transmitted directly from processor to memory, while commonly referenced high order components are stored in matching sets of base registers located in both the processor and the memory. The name derives from the fact that unlike the Hammerstrom and Davidson scheme, where the program explicitly manipulates base registers with special purpose instructions, in this scheme base register assignment is *dynamic* and requires no code to specifically manipulate base registers.

A detailed description of the Dynamic Base Register Caching scheme is as follows: Each time a new address is generated by the processor, the set of dynamically allocated base registers in the processor is searched for a register whose contents match the high order portion of the address. In order to facilitate a fast search for a new index, as well as exploit the high degrees of spatial and temporal locality exhibited by most address reference streams, the set of base registers *in the processor* is configured as a cache. If the search is successful, the location of the register in the cache is transmitted to the memory, instead of the high order portion itself. The memory uses this index to extract the proper high order component of the address from its base register file. (Note that the set of registers at the memory are configured as a register file, **not** as a cache). The proper address is then regenerated by concatenating the transmitted low order component with the extracted high order component. A diagram of the scheme appears in Figure 1.

Since the base register index is typically much smaller than the high order component it replaces, a significant reduction in the number of address lines between processor and memory is possible. We refer to the index and the low order component as the *reduced* address.

If the high order component of an address is not found in one of the processor base registers, a fault occurs. One of the base registers at the processor is immediately overwritten with the new high order component, and memory is informed of this fault by the transmission of a reserved index bit pattern. After receiving notification of a fault, the corresponding base register at the memory is

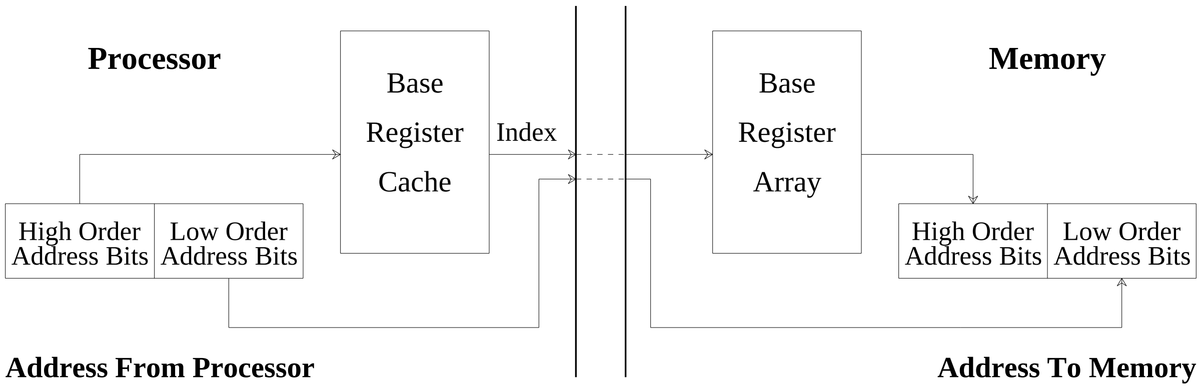


Figure 1

loaded by the processor across the address or data lines between processor and memory.

Dynamic Base Register Caching can be viewed as a form of address segmentation. However, Dynamic Base Register Caching differs from classic segmentation in several ways: (1) Classical segmentation is specified in machine instructions, while Base Register Caching occurs dynamically and is transparent to the programmer. (2) Classical segmentation provides protection and allows an architecture with a limited data word width to access a larger address space, whereas dynamic base register caching only reduces the number of address lines between processor and memory. (3) Since there is no bit overlap between base registers and low order components, base registers are simply concatenated with low order components to form a complete addresses. Because of overlap, segment registers are typically added to low order components to produce complete addresses.

Dynamic Base Register Caching can be viewed as a special case of Pleszkun's address prediction stack where the size of each code block is one. Unlike the address prediction stack, Dynamic Base Register Caching is simpler and more practical to implement. It requires adding only a few extra registers and some address comparison logic to existing processor and memory systems.

3.1. Implementation Issues

Dynamic Base Register Caching is attractive because of its simplicity. However, a number of implementation issues must be resolved before it can be efficiently implemented. A Dynamic Base Register Cache configuration and replacement scheme must be selected; the processor hardware must also be able to transfer actual high order components instead of indices to the memory after Dynamic Base Register Cache faults.

This can be accomplished most efficiently in one of two ways. New high order components can be transferred from processor to memory over the reduced address lines in multiple cycles, or in a single additional cycle over the data lines. Although sending the high order component over the data lines seems preferable, mixing address and data lines may introduce propagation delays in the critical path of the processor and thereby increase the cycle time on all operations. Sending the high order component in multiple cycles over the reduced address lines also requires additional gating circuitry. However, address lines tend to be near each other, and the propagation delays involved in switching address lines may be less than delays involved in switching between address and data lines. The tradeoff between using address or data lines to transfer the high order component between processor and memory must be studied in greater detail for specific technologies (e.g. VLSI, TTL, ECL).

It is worth repeating that the base register hardware at the memory can be simpler than at the processor, because cache hit detection and cache replacement is only necessary at the processor. The hardware at the memory passively accepts an index to the appropriate base register if there was a hit in the DBRC of the processor, and receives a new high order component with replacement instructions if there was a DBRC miss. Therefore, it is expected that such a scheme will have little impact on memory system performance.

4. The Simulation Environment

In order to test the effectiveness of the Dynamic Base Register Caching scheme, a wide range of trace driven simulations were performed using traces taken from a set of C codes for both VAX and SPUR architectures. (We were not limited to using numerical FORTRAN codes for an IBM 360 system as Hammerstrom and Davidson were.) The goal of the simulations was to determine the magnitude of the performance degradation that occurs using Dynamic Base Register Caching as well as the optimal design parameters for its implementation. In order to meet this goal, a set of representative traces were selected, and realistic Dynamic Base Register Cache (DBRC) and system configurations were defined. The following sections detail the traces we chose and configurations we elected to simulate.

4.1. The Simulations

The simulations performed measured DBRC hit rate as a function of the number of base registers and the reduced address size (which is a combination of both the index and low order component). The number of base registers in a given configuration is determined by the number of index bits; for example, since one index bit pattern is reserved to indicate a DBRC miss, k index bits can address up to 2^k-1 base registers.

Because of the difficulty in performing large n-way associative searches, two different DBRC configurations were simulated. One assumes a fully associative DBRC using an LRU replacement scheme, and the other assumes a direct mapped DBRC. For the fully associative scheme, systems with 2 to 6 index bits and reduced address sizes that varied from 4 to 28 bits were simulated. (No more than 6 index bits were simulated because of the difficulty of implementing a fully associative search over more than 2^6-1 registers.) For the direct mapped DBRC configuration, the number of index bits ranged from 3 to 10, and the reduced address size varied from 4 to 30 bits.

Since the recent trend in single-chip processor design has been to include larger and larger on-chip caches [BCDF87, KoMa89], simulations were also performed assuming that a cache was situated between the address generation logic and the external memory. This cache size varied from 256 to 32,768 bytes, and both a direct mapped and a fully associative cache were simulated. DBRC lookups were done in parallel with cache lookups, so that whenever a cache miss was detected the DBRC logic was ready with the correct reduced address. These simulations were performed with the expectation that the caches would capture most available locality, decreasing the effectiveness of the DBRC.

4.2. The Traces

The address stream traces used in this study were taken from several different sources. Five of the traces were taken from 3 LISP programs compiled for the SPUR architecture by Hill [Hill87]. Hill selected these samples to exhibit different but somewhat pessimistic locality profiles. These programs were Slc (the SPUR LISP compiler), Rsim (a circuit simulator simulating a counter), and Weaver (a production system for VLSI chip routing).

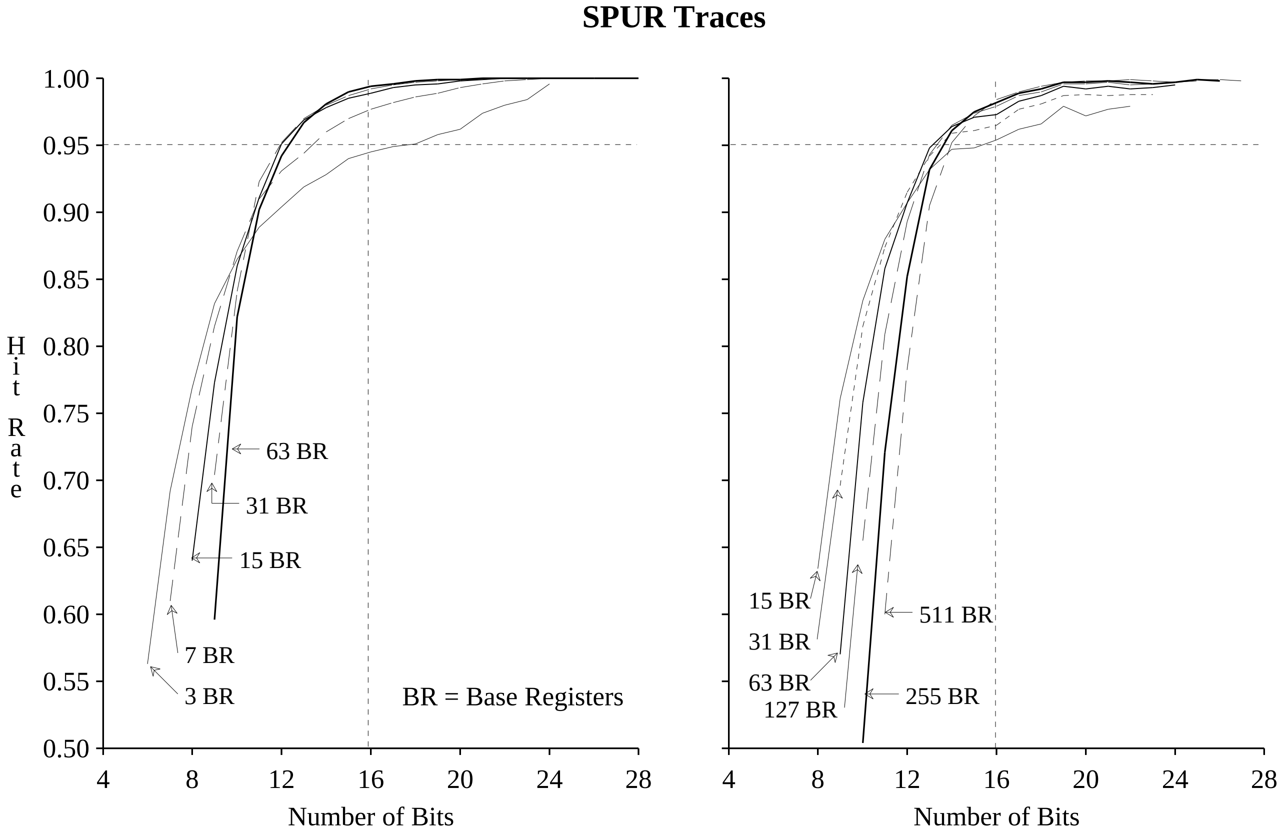
Two 500,000 instruction trace data samples were taken from each of the runs of the Slc and Weaver programs, and one 500,000 instruction trace was used of the Rsim program. The 5 traces were concatenated together, and treated as one large composite trace file. A total of 3,100,330 memory references (both data and instruction) are generated by this composite trace.

In addition to the SPUR traces, the ATUM address traces (gathered using microcode patches on a VAX 8200 [AgSH86]) were also used. The ATUM traces consist of snapshots of address referencing activity from each of nine different programs running on Ultrix and VMS operating systems. These traces are of interest primarily because they include operating system references. To examine the effects of system references on Dynamic Base Register Cache performance, we simulated Dynamic Base Register Caching using ATUM traces with and without system calls. The ATUM traces contain a total of 8,397,122 references (6,23,280 references when the operating system references are removed).

5. Discussion of Simulation Results

As stated in the previous section, the main goal of this study was to attempt to ascertain the effectiveness of the Dynamic Base Register Caching scheme. An exhaustive presentation of the results of the hundreds of simulations performed is well beyond the scope of this paper. Instead, this section will present a discussion of the most interesting results, and indicate areas that merit further investigation.

The results of the simulation runs on the different composite traces appear in figures 2 through 4. In each of these figures, a range of fully associative Dynamic Base Register Cache (DBRC) sizes using an LRU replacement scheme is compared to a number of direct mapped DBRC configurations. For the fully associative DBRC configuration, the DBRC sizes presented range from 3 to 63 base registers; for the direct mapped DBRC configuration, the DBRC sizes presented range from 15 to 511 registers. Figure 2 shows the results of the simulation runs of the SPUR composite trace, figure 3 contains the results of the ATUM composite trace simulations with the operating system calls intact, and figure 4 presents the simulation results of the ATUM composite trace with all operating system references removed. In these simulations



(a) Fully Associative BRC **(b) Direct Mapped BRC**

Figure 2

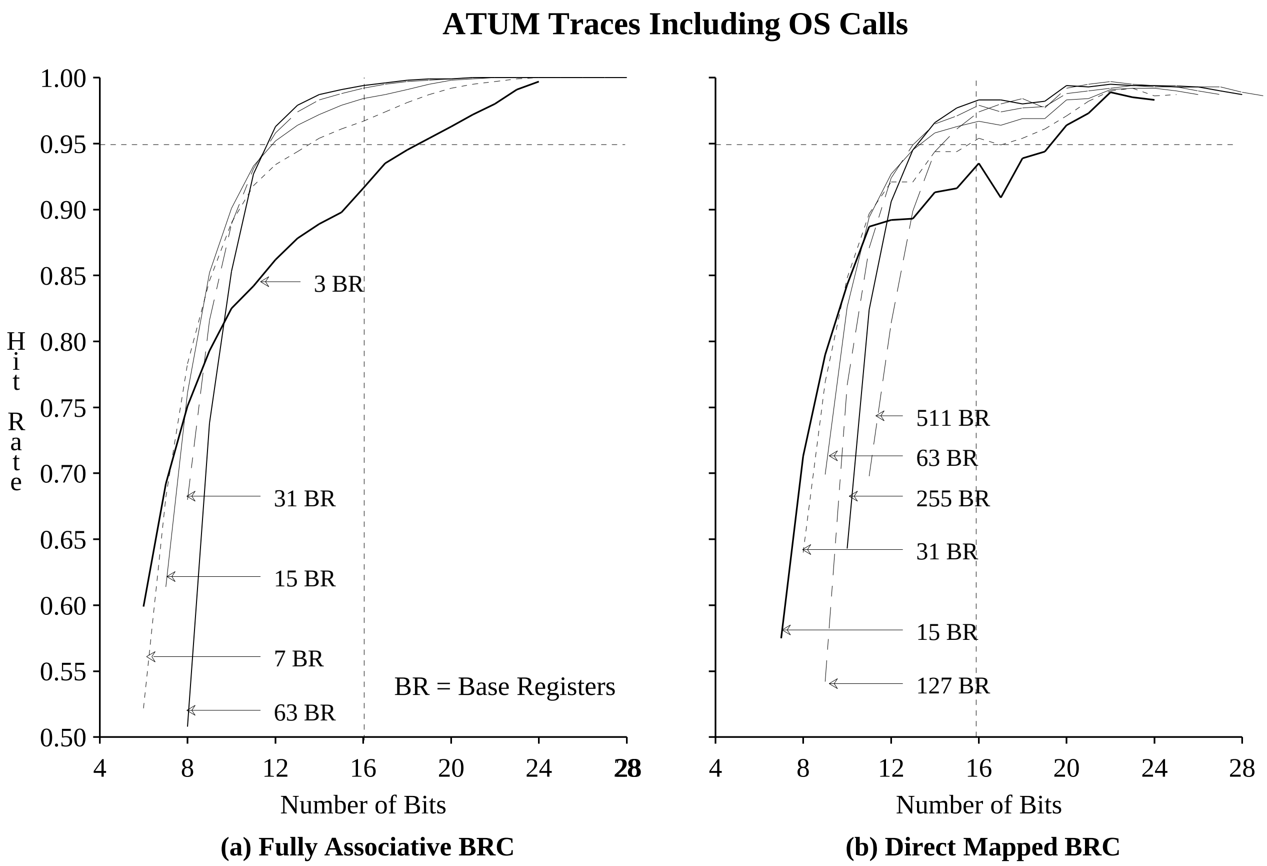


Figure 3

no cache was assumed to lie between the address generation logic and external memory.

Examining these three figures, we see that the basic shape of the majority of the curves remains constant in all cases (the exception being the worst case performers). The similarities between the SPUR (figure 2) and ATUM traces with OS calls included (figure 3) are remarkable, especially when one considers that the ATUM traces originate from a markedly different architecture and programming language than the SPUR Lisp traces (in addition to including OS references.) There are slight differences between the figures for smaller reduced address sizes, but once the reduced address size exceeds a certain point the graphs converge to within a couple of percent. This implies that the effectiveness of the DBRC is not dependent on any specific architectural feature.

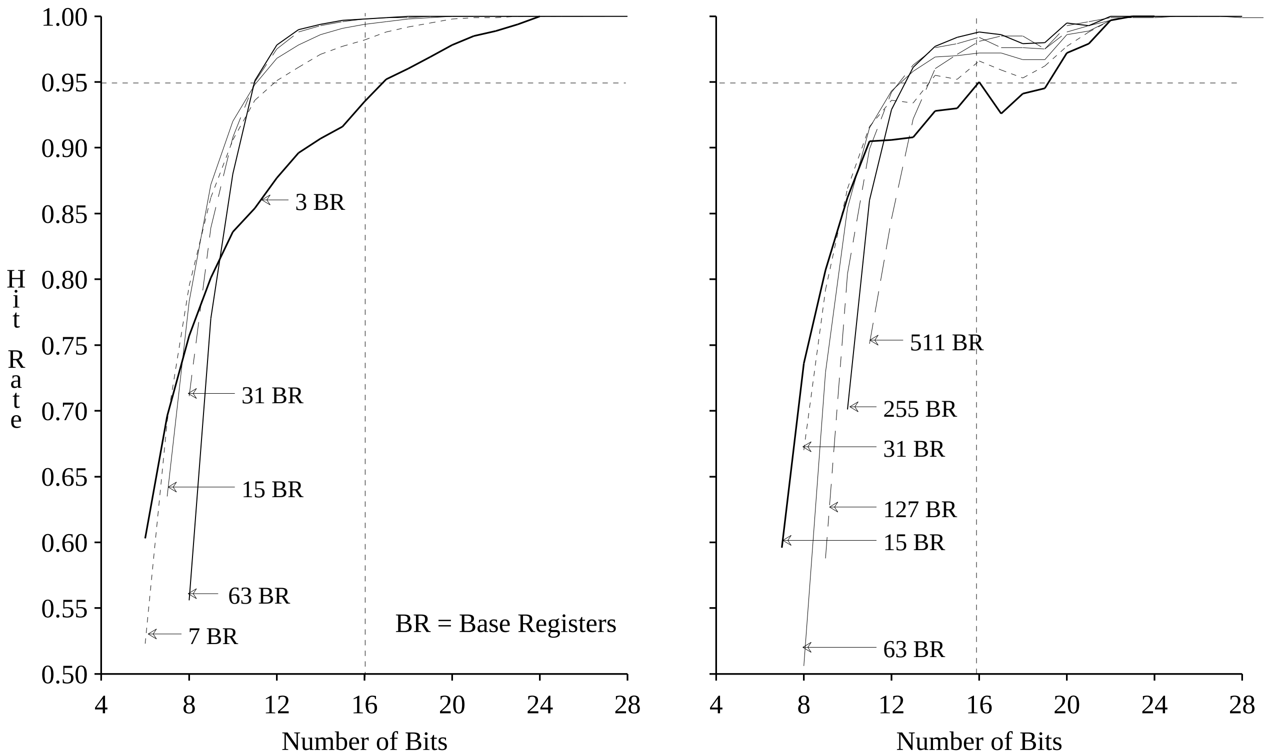
In each of the three figures we also see that when the reduced address size is small, the fully associative DBRC outperforms the corresponding direct mapped DBRC for most DBRC sizes. This is not unexpected, since a fully associative cache generally has a higher hit rate than a direct mapped cache. However, by the time the reduced address size has grown to approximately 16, the differences between the two configurations have significantly decreased. In the case of the SPUR trace (figure 2), when the reduced address size is 16 the difference between the two configurations for most DBRC sizes is less than 2%. This is important, since for both speed and ease of implementation reasons, a direct mapped DBRC may be preferable to an n-way associative DBRC.

Based on an analysis of these three figures, and keeping in mind that both the VAX and the SPUR architectures have a wordsize of 32 bits, we can make the following observations: For reduced address sizes below 16 bits, a fully associative DBRC can be significantly more effective than a direct mapped DBRC. However, above 16 bits, there is very little difference between the fully associative and direct mapped configurations. Therefore, a reduced address size of 16 bits appears to be a good choice, assuming at least 15 registers in a fully associative DBRC and at least 63 registers in a direct mapped DBRC.

Extending this analysis a bit further, we note that with a reduced address size of 16 bits and either a fully associative DBRC with 15 or more registers or a direct mapped DBRC with 63 or more registers, 98% of the time an address request is handled without generating a fault. Only 2% of the time more than 16 address lines are necessary to satisfy a memory request. This also means that 50% of the existing address bandwidth of this machine (25% of the total bandwidth) exists to support only 2% of the memory reference operations.

There is an obvious problem with these results, however. For a processor without an on-chip cache, every instruction must come from off-chip, and a lookup into the DBRC must be performed before an instruction address can be generated. Therefore, the DBRC will almost

ATUM Traces With OS Calls Removed



(a) Fully Associative BRC **(b) Direct Mapped BRC**

Figure 4

undoubtedly lie in the worst case path. This means that this technique may not as appealing to designers of cacheless processors (unless I/O pin limitations are the overriding system bottleneck), even though for a given DBRC configuration the hit rate can be as high as 99.9%.

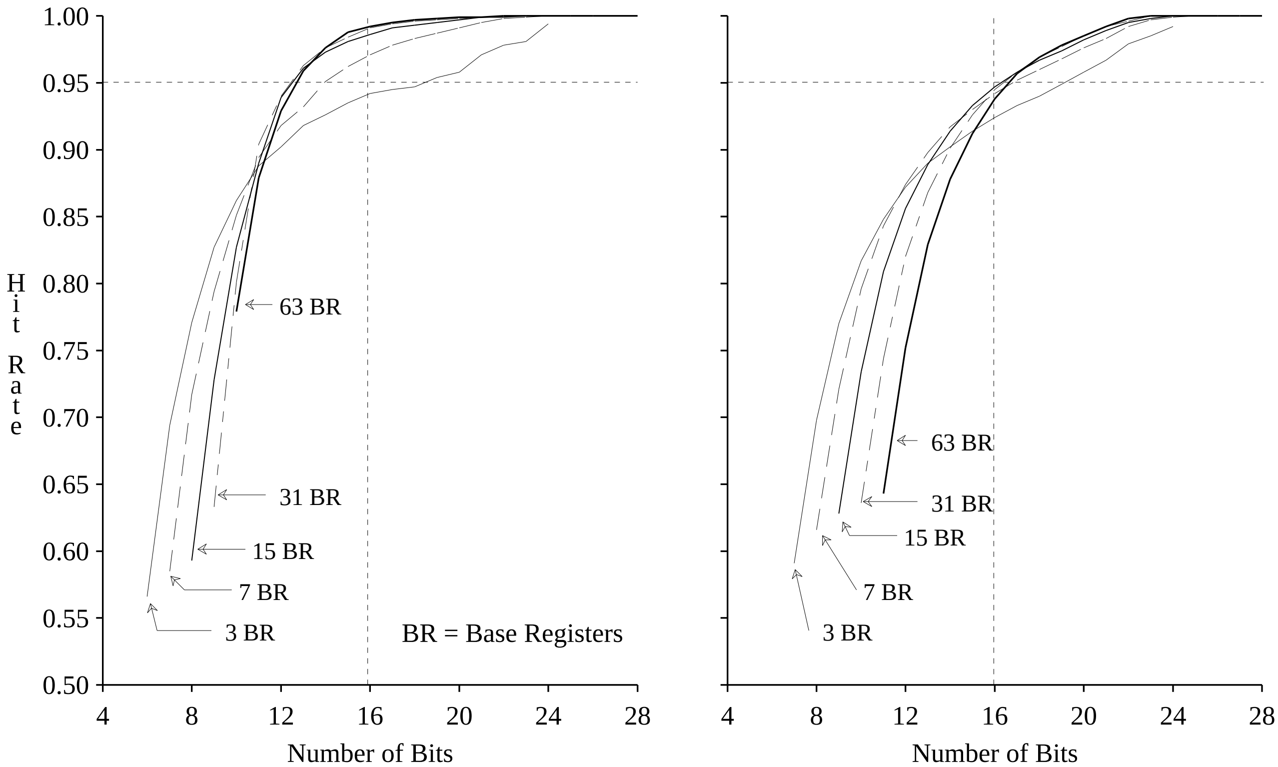
However, the presence of an on-chip cache makes it possible to hide the time cost of a DBRC lookup. The DBRC lookup can be executed in parallel with the cache lookup, so that by the time a miss in the on-chip cache has been detected the proper address index will already have been calculated. Intuitively, one would expect the performance of the DBRC scheme to degrade when used in conjunction with a cache, since the cache captures most of the locality in a reference stream that the DBRC was designed to exploit. In order to estimate the magnitude of the performance degradation of the DBRC scheme in this case, another set of simulations were performed. Interestingly, we did not find the degradation to be as significant as expected, as the following three figures will demonstrate.

Figure 5 shows the results of the simulation runs on the SPUR composite trace after it has been passed through 2 different cache filters. In figure 5a, the trace was filtered through a 256 byte cache, while in figure 5b the cache size was 32,768 bytes. In both figures, a fully associative DBRC configuration was used.

As can be seen in the figure, as the size of the on-chip cache increases, the DBRC scheme performance degrades noticeably (for smaller reduced address sizes). Looking at the graphs, it appears that for a large on-chip cache, the optimum reduced address size is much closer to 20 bits than to 16. It is important to remember, however, that the DBRC is servicing far fewer requests. For the SPUR trace the hit rate in the 32K byte on-chip cache is almost 95%, so the fact that a DBRC configuration with 16 bit reduced addresses successfully handles "only" 95% of the requests made to it also means that only **.25%** of the total address requests generated by the processor require the full 32 address lines. Sixteen address lines are enough to handle **99.75%** of all requests generated by the processor!

Figure 5 demonstrated that a DBRC still works well when the maximum amount of locality has been removed. Figure 6 shows how the presence of a large direct-mapped on-chip cache affects the performance of the different DBRC configurations. (The on-chip cache is assumed to be direct mapped for reasons given in [Hill87, Joup89]) The on-chip cache hit rate in this figure is 88.9%. In figure 6a, a fully associative DBRC is simulated, while in figure 6b the DBRC is direct mapped. In this figure we see that the hit rate of the direct mapped DBRC is noticeably worse (4%-5% worse) than the fully associative DBRC for a reduced address size of 16 bits. However, because the on-chip cache is capturing so many of the off-chip references, even when the hit rate in the DBRC falls to near 90%, 16 address lines are sufficient to handle over 99% of all address references. Again, 50% of the address lines are being effectively used less than 1% of the time.

SPUR Traces: Fully Associative BRC



(a) 256 Byte On-Chip Cache **(b) 32768 Byte On-Chip Cache**

Figure 5

It is also interesting to note in figure 6a how well a small fully associative DBRC works. With only seven elements, the DBRC hit ratio is almost 94%. This effect is similar to that pointed out by Jouppi in [Joup90], where he found that a small (2-5 element) associative cache located behind a large direct mapped cache was very effective at removing many conflict misses. In this case, the DBRC functions in a similar manner (although the DBRC contains only addresses and no data.)

The SPUR traces are of interest because the SPUR processor has a RISC architecture. However, the ATUM traces are also of great interest because they include operating system calls. Figure 7 is a repeat of figure 6, with the exception that the trace used in the simulation was the ATUM trace with operating system calls included. In figure 7a, a fully associative DBRC is simulated, while in figure 7b the DBRC is direct mapped. (Interestingly enough, the on-chip cache hit rate is higher (92.2%) for the ATUM traces with operating system calls included than for the SPUR traces.)

In figure 7, we see a definite decrease in the effectiveness of the DBRC for both configurations. For most direct-mapped DBRC configurations, the hit rate has fallen into the upper 80's. As was the case in the previous two figures, though, this lower hit rate coupled with the higher on-chip cache hit rate means that a reduced address size of 16 is still sufficient to handle over 99% of all requests. It is also interesting to note that the effectiveness of a small associative DBRC is reduced in this case. This is probably due to the presence of the operating system calls.

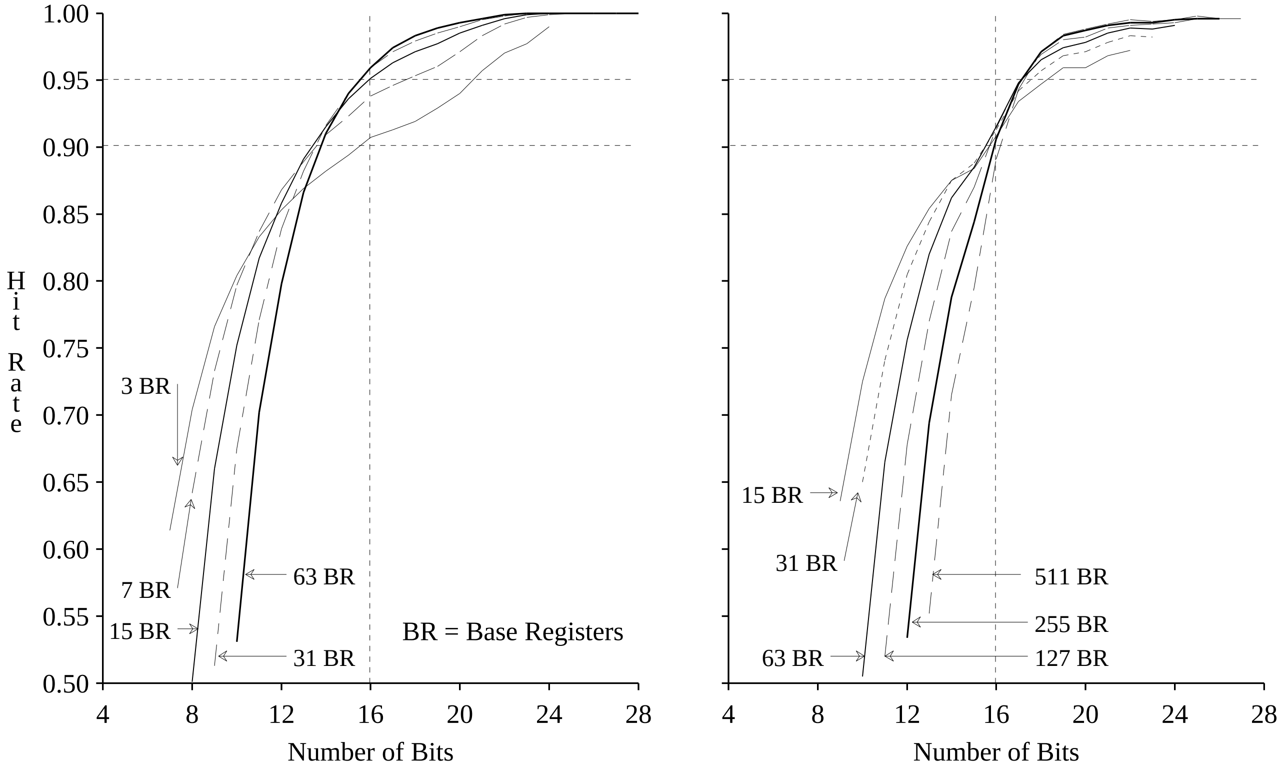
What must be stressed when evaluating the last three figures is that the DBRC lookup time can now be hidden, by performing the DBRC lookup in parallel with the cache lookup. This fact makes the DBRC scheme extremely viable, and brings into question the wisdom of devoting 50% of a processors address bandwidth (25% of all I/O bandwidth) to supporting events that occur less than 1% of the time.

6. Address Line Utilization

Dynamic Base Register Caching trades a performance loss due to Dynamic Base Register Cache faults for a reduction in the number of address lines between processor and memory. It is clearly advisable to incur a small performance loss to achieve a large reduction in address lines. However, as the number of address lines is reduced, a greater amount of performance is lost for each additional line that is eliminated. At some point it becomes unclear whether it is advisable to eliminate an additional line.

One possible way to make this decision is to optimize *Address Line Utilization.* We define Address Line Utilization as the average number of uncompressed address bits that are transferred across each reduced address line per cycle. If bandwidth is the main constraint on system performance, maximizing Address Line Utilization also maximizes system performance. We define

SPUR Traces: 32768 Byte On-Chip Direct-Mapped Cache



(a) Fully Associative BRC **(b) Direct Mapped BRC**

Figure 6

Address Line Utilization more formally as:

$$\textit{Address Line Utilization} = U = \frac{A}{RC}$$

where U = Address Line Utilization
A = Size of address word
R = Size of reduced address
C = the average number of cycles to transfer each addresss word

As the size of the reduced address R decreases, Address Line Utilization increases. This is offset by a reduction in the hit rate which causes more faults and thereby increases the average number of cycles per transfer C.

The average number of cycles per transfer is determined by the following formula:

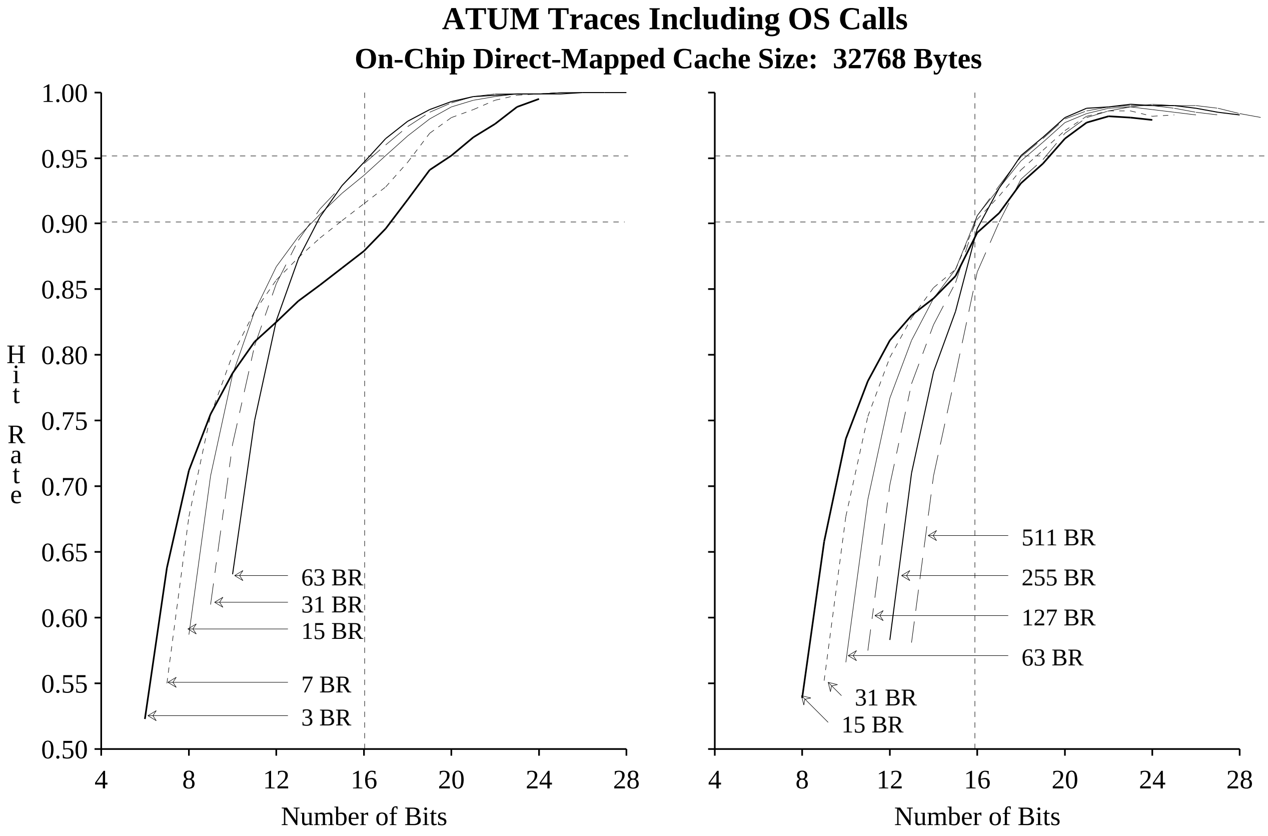
$$C = 1 + (1 - h) \times (M\ cycles)$$

where h = hit rate
M = penalty for a Dynamic Base Register Cache miss

The size of the miss penalty M depends upon how fault recovery is implemented in Dynamic Base Register Caching.

A number of actions take place during a fault. A miss is detected at the processor when a new high order component does not match any of the high order components contained in the processor's Dynamic Base Register Cache. The new high order component must be moved into the base registers at both the processor and the memory. The base register at the processor can be loaded directly, while the base register at the memory must be loaded across the existing address or data lines between processor and memory. To accomplish this, the reserved bit pattern is first transferred across the index lines to indicate to the memory that a miss has occurred. The new high order component is then transferred from the processor to the base register array at the memory. The new high order component can either be transferred across the reduced address lines, across the data lines, or across both reduced address and data lines. The high order component is typically larger than the reduced address, so multiple cycles will be required to transfer the high order component across the reduced address lines.

Only one cycle is required to transfer the high order component across the data lines because the data word is at least as large as the high order component. However, there are costs for transferring address signals on data lines. It is hard to switch address and data information onto the same lines because address and data paths tend to reside in disjoint portions of a processor. Long wires or traces are required to bring address and data signals together, and these wires can introduce delay in the critical path of the processor which may increase the processor cycle time for all operations. It is preferable to increase the time taken on faults (which occur infrequently) rather than



(a) Fully Associative BRC

(b) Direct Mapped BRC

Figure 7

increasing the cycle time for all operations. Gating address signals onto data lines can also interfere with normal operation of the data lines.

Consider the miss penalty that results from transferring the high order components over the reduced address lines. Multiple cycles are required to do this. The miss penalty becomes:

$$Miss\ Penalty = M = \frac{H}{R}$$

where	H	=	size of high order component
	R	=	Size of reduced address

The miss penalty M is the number of cycles required to transfer a high order component of size H over R reduced address lines.

Figure 8 shows Address Line Utilization as a function of reduced address size. The miss rates used to compute Address Line Utilization are taken from simulations using the composite ATUM trace with system references. For each reduced address size we use the number of base registers which yields the highest hit rate. This graph shows that utilization is maximized when 11 reduced address bits are used. This maximum utilization is approximately 2.5 which translates into a 60% reduction in processor-to-memory address bandwidth. Of course optimizing Address Line Utilization lengthens the average time required to transmit addresses because smaller reduced address words result in more Dynamic Base Register Cache faults. Increasing the size of the reduced address

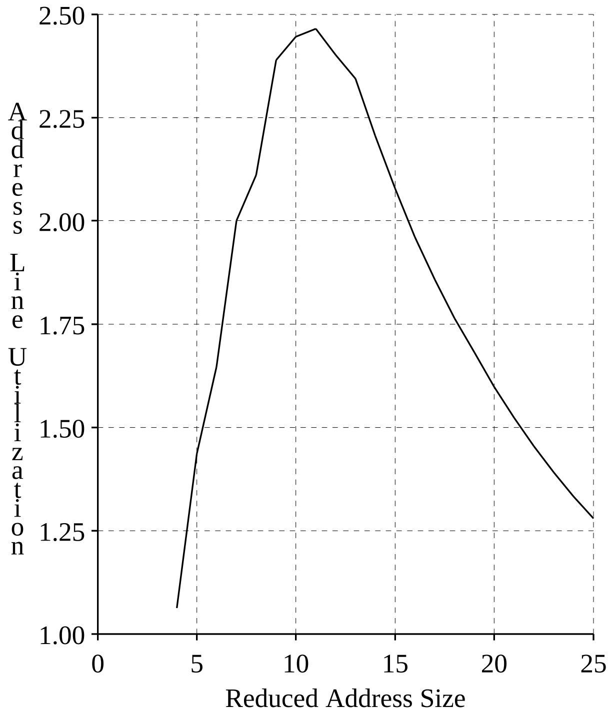


Figure 8

word, for example, to a larger size such as 16 bits, decreases average address transmission time but results in a slightly lower Address Line Utilization. The proper tradeoff point depends on overall system performance constraints.

Address Line Utilization is only one of a large number of possible metrics one can use to optimize Dynamic Base Register Caching. Individual applications have specific bandwidth and processing performance requirements. In some applications bandwidth is the main constraint to performance, and bandwidth must be maximized at all costs. In other applications processor performance must be maximized, even at the cost of increasing bandwidth requirements. We intend to explore a wide range of metrics to optimize Dynamic Base Register Cache performance.

7. Conclusions

In this paper, we have presented a technique to reduce processor-to-memory address bus width by dynamically dividing each address word into two components, a high order component and a low order component. By caching the higher order portions of address references in a set of dynamically allocated base registers and providing a similar set of registers in the memory system, it becomes possible to transmit small register indices between the processor and memory instead of the high order address bits themselves. This is important because of the growing disparity between processor speeds and I/O bandwidth.

A major advantage of Dynamic Base Register Caching is its simplicity. This makes it easy to implement, and since it is transparent to the architecture it is straightforward to incorporate into existing systems. We have demonstrated using trace driven simulations that for two very different architectures, the incorporation of a Dynamic Base Register Cache can effectively support a significantly reduced processor-to-memory address bus width. For 32 bit processors, several different Dynamic Base Register Cache configurations can provide a hit rate of over 98% using 16 address lines to transmit the low order component-register index pair.

A Dynamic Base Register Cache may introduce extra delay in a processors critical path, however. Fortunately, if the processor incorporates an on-chip cache, the DBRC lookup time can be effectively hidden. Trace-driven simulations of a processor with an on-chip cache indicate that over 99% of all address references can be handled with only 16 address lines. This implies that 50% of a processors address bandwidth (25% of the total bandwidth) is used less than 1% of the time.

8. Acknowledgements

We would like to thank Andrew Pleszkun for his seminal ideas on this subject, and Jeffrey Becker and various reviewers for their suggested revisions of this paper.

References

[AgSH86] A. Agarwal, R. L. Sites and M. Horowitz, ‘‘ATUM: A New Technique for Capturing Address Traces Using Microcode’’, *Proceedings of the Thirteenth Annual International Symposium on Computer Architecture*, Tokyo, Japan (June 2-5, 1986), pp. 119-127.

[BCDF87] A. D. Berenbaum, B. W. Colbry, D. R. Ditzel, R. D. Freeman, H. R. McLellan, K. J. O’Connor and M. Shoji, ‘‘CRISP: A Pipelined 32-bit Microprocessor with 13-kbit of Cache Memory’’, *IEEEJSS*, vol. SC-22 (October 1987), pp. 776-782.

[HaDa77] D. Hammerstrom and E. Davidson, ‘‘Information Content of CPU Memory Referencing Behavior’’, *Proceedings of the Fourth Annual Symposium on Computer Architecture*(March 1977), pp. 184-192.

[Hill87] M. D. Hill, *Aspects of Cache Memory and Instruction Buffer Performance*, Ph.D. Thesis, Department of Computer Sciences,, Berkeley, California, (November 1987).

[Joup89] N. Jouppi, ‘‘Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU’’, *Proceedings of the Sixteenth Annual International Symposium on Computer Architecture*, vol. 17, no. 3 (June 1989), pp. 281-289.

[Joup90] N. Jouppi, ‘‘Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers’’, *Proceedings of the Seventeenth Annual International Symposium on Computer Architecture*, vol. 18, no. 2 (May 1990), pp. 364-373.

[KoMa89] L. Kohn and N. Margulis, ‘‘The i860 64-bit Supercomputing Microprocessor’’, *Proceedings of Supercomputing 1989*, Reno, Nevada (November 13-17, 1989), pp. 450-456.

[PlRD81] A. R. Pleszkun, B. R. Rau and E. S. Davidson, ‘‘An Address Prediction Mechanism for Reducing Processor-Memory Address Bandwidth’’, *Proceedings of the 1981 IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*(Nov. 11, 1981), pp. 141-148.