Exploiting eDRAM bandwidth with data prefetching: simulation and measurements

Valentina Salapura, José R. Brunheroto, Fernando Redígolo, Alan Gara

IBM Thomas J. Watson Research Center
Yorktown Heights, NY

Abstract

Compared to conventional SRAM, embedded DRAM (eDRAM) offers power, bandwidth and density advantages for large on-chip cache memories. However, eDRAM suffers from comparatively slower access times than conventional SRAM arrays. To hide eDRAM access latencies, the Blue Gene/L® supercomputer implements small private prefetch caches.

We present an exploration of design trade-offs for the prefetch D-cache for eDRAM. We use full system simulation to consider operating system impact. We validate our modeling environment by comparing our simulation results to measurements on actual Blue Gene systems. Actual execution times also include any system effects not modeled in our performance simulator, and confirm the selection of simulation parameters included in the model.

Our experiments show that even small prefetch caches with wide lines efficiently capture spatial locality in many applications. Our 2kB private prefetch caches reduce execution time by 10% on average, effectively hiding the latency of the eDRAM-based memory system.

1 Introduction

Future microprocessor designs will require new design trade-offs to address new constraints on architectures. The increasing compute power available per chip from the use of chip multiprocessors is not matched by a commensurate increase memory bandwidth via off-chip I/O. This may lead to a potentially unbalanced and inefficient design.

SRAM arrays are conventionally used as on-chip cache memories to obtain a significant reduction in I/O bandwidth requirements. However, the use of SRAM arrays is limited by the comparatively low density, and high power dissipation. SRAM memories are also suffering from manufacturability constraints limiting future access speeds due to device variation limiting the ability to accurately match FET devices of storage cells [8, 12].

A promising solution to these multiple constraints is the adoption of embedded DRAM (eDRAM) for high-capacity, high-density on-chip caches. Embedded DRAM merges DRAM and logic fabrication technologies to build the familiar 1T DRAM cell into a logic chip, and offers a significant increase in memory capacity per given unit area over SRAM, as well as low power operation and very wide data ports [13]. However, eDRAM typically will have a higher access latency than an SRAM-based solution. This could be alleviated by deploying a prefetch scheme to decouple application access latency from eDRAM access latency, and using the available eDRAM bandwidth to hide latency.

The Blue Gene/L system is the first high performance computing (HPC) system that delivers on the promise of on-chip eDRAM for increased performance at lower cost. The Blue Gene/L compute chip [5, 19, 20] has two cores, each with private L1 instruction and D-caches. Misses at the L1 level are given to a small private prefetch cache that acts as the L2. Each of the two L2 prefetch caches communicates with the L3 on-chip 4MB eDRAM cache, which is shared between the two processors.

The work described here evaluates the architecture for the small private prefetch caches with respect to hiding the access latency to eDRAM in a multiprocessor environment. Many previous studies focus on application traces only and may neglect the impact of the interaction between application software and operating system. In contrast, we study prefetch behavior for a set of compute-intensive workloads using full system simulation. We present an initial evaluation of the prefetch algorithms in prior work [6]. Here we compare our simulation results with measurements on actual Blue Gene systems to evaluate simulation accuracy and decisions made in the design process.

The contributions of this paper are: (1) an extensive simulation-based design space exploration of prefetch cachees for an on-chip eDRAM cache, (2) an analysis of operating-system impact on prefetch effectiveness, and (3) a validation of simulation results with hardware measurements on a Blue Gene/L system.

2 Blue Gene Memory Subsystem

The Blue Gene/L Compute chip (BLC chip) contains two PowerPC 440 processor cores, each with a SIMD floating point unit, as illustrated in Figure 1. Each PowerPC

1-4244-1258-7/07/$25.00 ©2007 IEEE

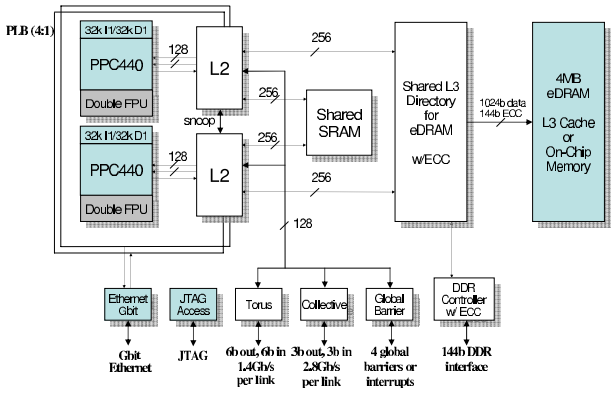

Figure 1. Blue Gene/L compute chip.

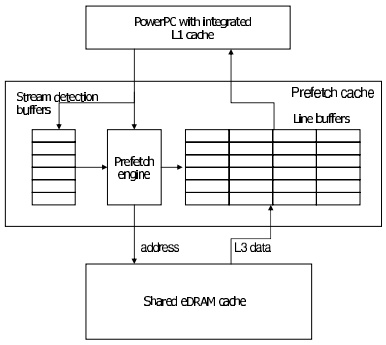

Figure 2. Prefetch cache architecture.

440 core contains 32kB private L1 I- and D-caches with 32B lines, and interfaces to a private prefetch L2 cache with 128B buffer line size. The L2 caches communicate with a shared 4MB L3 on-chip eDRAM cache with 128B lines. The eDRAM is configured as two 2MB interleaved banks [16]. The prefetch L2 cache decouples the number and timing of requests generated by the core from requests to the L3. The L2 cache stores wide L3 cache lines, and satisfies multiple narrower L1 requests. This reduces latencies for L1 requests along with traffic to the L3. Comparing the Blue Gene/L memory system design to a traditional memory system design with an SRAM-based L2 cache in terms of complexity, area, and power/performance, the present design offers several advantages. There are several different ways to make this comparison:

- Remove the prefetch cache and use the eDRAM as L2: cache size remains the same, but latency increases. We simulate this configuration, which we heretofore refer to as the "no prefetch cache".
- Keep the L3 eDRAM cache, and add a standard L2 SRAM-based cache, having the same size (2kB) as our prefetch cache. Such a small L2 cache would be ineffective. A larger L2 unacceptably increases chip area.
- Remove the eDRAM L3, and use a standard SRAM L2. To maintain chip area, the 4MB eDRAM can be replaced by a 1MB SRAM. In addition, the eDRAM solution requires about four times less power than a $1/4$ sized SRAM solution. Power considerations are especially important for embedded and large-scale systems such as Blue Gene.

It was a Blue Gene/L project requirement to reuse an unmodified PowerPC core available as a hard macro. Any core changes would have incurred significant cost and would have delayed introduction of the system. The PowerPC 440 L1 cache is tightly integrated with the load-store unit, and any changes to the L1 cache would have required changes to the processor. Such changes affect the entire core design, requiring expensive re-timing and re-validation. By deploying an external prefetch cache, the pre-tuned PowerPC 440 hard macro could be placed unmodified and could achieve peak clock frequency without design rework.

By prefetching into a dedicated prefetch cache, and keeping prefetched data out of the L1 D-cache until referenced by the application, we avoid pollution from prematurely fetching data that could potentially displace those still needed by the processor. This is particularly important for carefully tuned algorithms that size their working sets to efficiently exploit the memory subsystem. Extraneous prefetching in this environment interferes with delicate tuning for peak performance.

2.1 Prefetch Cache Architecture

Figure 2 illustrates the prefetch cache architecture explored in this work and deployed as the L2 D-cache in the Blue Gene/L compute chip. The prefetch cache consists of line buffers storing demand-fetched and prefetched lines from eDRAM, a prefetch engine that initiates prefetches and manages line buffer replacement, and a stream detector unit that detects data streams.

On each L1 D-cache miss, the prefetch cache directory is checked. If the requested data are available in the prefetch cache, they are forwarded to the L1 D-cache. If the request misses in the prefetch L2 cache, only $1/4$ of the addressed L3 line (the portion corresponding to the requested L1 line) is fetched from the L3 and is buffered in the L2 (in a portion of a dedicated line buffer) before it is forwarded to the L1. Line buffers are fully associative. Once prefetched, lines reside in the prefetch cache until other requests evict them.

To detect streams, we use an N-deep history queue for storing prefetch cache address tags [17]. We refer to this as the *stream detection buffer*. When the processor requests data that miss in the L1 D-cache, the prefetch unit records the corresponding L2 address in the stream detection buffer. If the requested address matches an L2 address recorded in the stream detection buffers, but the tag does not match, the requested L3 cache line is fetched and stored in a line buffer, and a stream is established. The first subsequent access to stream data resident in the line buffer triggers a prefetch request that loads one L3 cache line (corresponding to four L1 lines) to the prefetch cache.

An alternative approach avoids stream detection buffers, but instead issues fetch requests for each new L2 D-cache request not satisfied in the prefetch cache along with a prefetch request for the next line. This approach automatically prefetches a data stream based on only one request. We refer to this as *optimistic prefetch stream detection*. Advantages of this approach include using prefetch address

NAS	Instructions	L1 Misses	Misses per 1000 instructions
BT	547,414,050	30,788,712	56.24
CG	349,304,498	19,824,670	56.75
FT	645,116,212	37,248,944	57.74
IS	30,697,133	564,715	18.40
LU	238,891,062	10,934,076	45.77
MG	56,399,797	2,897,583	51.38
SP	273,988,939	20,660,969	75.41

Table 1. NAS Benchmarks Characteristics

tags associated with each line buffer as the address tracking method for identifying streams. This reduces the number of state bits that must be maintained. Optimistic prefetching is more aggressive, and thus issues more prefetches to the L3.

3 Methodology

We use full system simulation and two different operating systems to explore the effectiveness of stream prefetching for supercomputer applications. We use BGLsim [7], a full system simulator for the Blue Gene/L system based on the Mambo PowerPC simulator [4]. BGLsim is architecturally accurate at the instruction-set level. BGLsim exposes all architected features of the hardware, including processors, floating-point units, caches, memory, interconnection, and other supporting devices. The simulator runs unmodified system and user software as used on actual Blue Gene hardware. The simulator includes interaction mechanisms for inspecting the entire internal machine state, allowing for more flexible and detailed instrumentation than that possible with real hardware.

In our experiments, we use the L1 address miss sequence containing both application and operating system references for a variety of numerically intensive applications. We opt for a multi-model simulation environment comprised of two modules: one is the full system simulator (with pseudo cycle accuracy) that takes binary code as input, and the other is based on traces. The latter is faster, since it abstracts out many details, and is useful for coarse design space exploration, yielding design parameters that we then evaluate in full detail using the first model.

The full system simulator can simulate a multi-node system, but here we use it for simulation of a single Blue Gene/L node. We opt for a single processor simulation, since having multiple processors on a chip does not affect the prefetch hit rate. The reason for this is that prefetch cache is private to each processor, and there is no interprocessor interaction for prefetch caches. Our simulation model assumes that the subsequent request to the L3 is satisfied with a constant L3 latency, but in the actual hardware, the L3 latency depends on several factors (e.g., page already open, or number of pending load requests).

Splash	Instructions	L1 Misses	Misses per 1000 instructions
LU	57,687,452	343,118	5.95
FFT	60,373,803	712,177	11.80
Radix	87,116,807	582,659	6.69
Ocean	30,005,066	1,843,293	61.43

Table 2. Splash-2 Benchmarks Characteristics

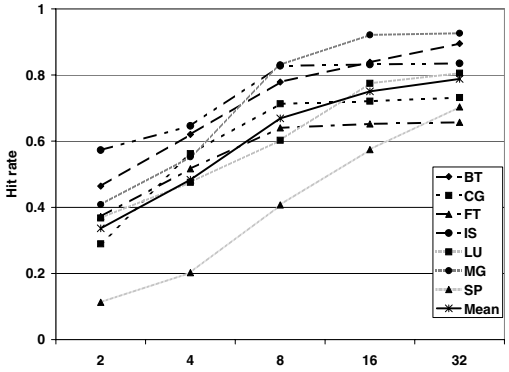

Figure 3. Varying stream detector size for the NAS benchmark.

To characterize prefetch cache performance we use prefetch cache hit rate, prefetch cache miss rate, and execution time (as predicted by the Blue Gene/L Pseudo Accurate Timing Model) [1]. The prefetch hit rate is the fraction of L1 D-cache misses that hit in the prefetch cache.

In our experiments, we use applications from the NAS [2] and Splash-2 [22] suites. We opt to use these publicly available applications, as they are representative of a wide range of scientific applications. We concentrate our efforts on scientific-computing intensive applications, since these are the target workloads for Blue Gene. Here, we report on all NAS class S benchmarks and Splash-2 kernel applications (LU, Radix, FFT), along with the ocean application. Tables 1 and 2 show benchmarks used, number of instructions executed, number of absolute L1 D-cache misses, and number of L1 D-cache misses per 1000 instructions which represent the total prefetch opportunity.

4 Experiments and Simulation Results

We model the prefetch cache to study the impact of design parameters, and we examine two operating systems with respect to prefetch cache hit rate and execution time. We first vary the size of the stream detector buffers to determine the minimum size yielding good prefetch hit rates. Then we vary the number of line buffers, and explore the impact of using various replacement policies for them. We evaluate the impact of supporting bi-directional stream detection, and analyze the impact of the operating system used. To fully understand the impact of prefetching on the overall memory subsystem, we determine the impact of prefetching on memory bandwidth to the shared L3 cache, and on application execution time. We also evaluate the impact of the prefetching, as opposed to using line buffers without prefetch support.

4.1 Stream Detector Buffers

When detecting data streams via the N-deep history queue, a new stream is started only if an L2 request hits

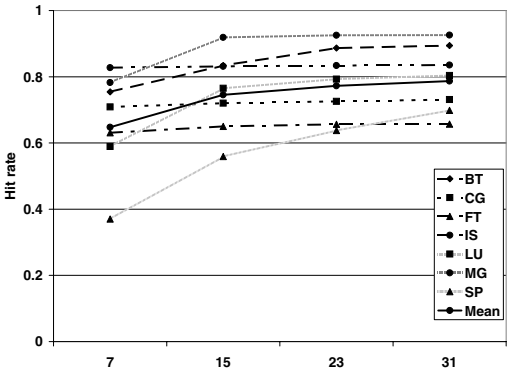

Figure 4. Varying the number of line buffers in the prefetch cache for the NAS benchmark.

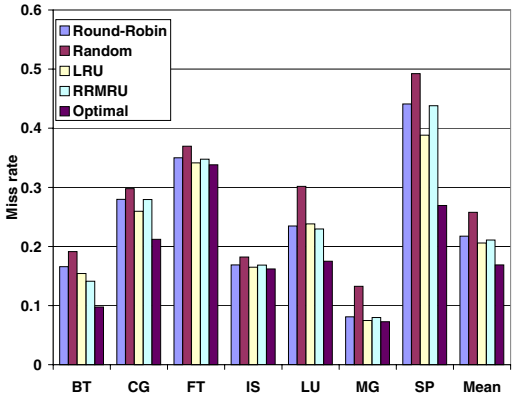

Figure 5. Prefetch cache miss rate for various line buffer replacement policies for the NAS benchmarks.

in the address history queue, requiring two requests to establish a data stream. Figure 3 shows behavior of a prefetch cache with stream detection. We simulate both NAS and Splash-2 benchmarks, but space limitations require that we list only results for NAS. For this simulation, we use a prefetch cache large enough not to limit the number of streams that can be tracked.

We vary the number of stream detector buffers from 2-32. Results indicate that using stream detector sizes above 16 does not significantly improve hit rates, except for SP. For SP, adding more stream buffers continues to increase the prefetch hit rate, as more of the distinct data streams can reside in the prefetch cache. For most applications, 16 stream detection buffers are sufficient to detect all data streams. Splash-2 benchmarks show similar trends.

4.2 Prefetch Cache Size

To determine the optimal number of prefetch line buffers, we vary their number from 7-31 while fixing the stream detector size at 16. We change numbers of line buffers in multiples of eight. One line buffer is used for data returned from L3 demand fetches not buffered in L2 (e.g., L2 requests without an established stream), hence the odd number of line buffers available for stream prefetches. Results are illustrated in figure 4.

The effect of increasing the prefetch cache size on hit rate is non-linear. Choosing a prefetch cache size of seven lines clearly fails to exploit the full prefetch potential: 15 lines yields significantly better performance. For the NAS benchmarks, selecting 23 line buffers increases hit rates across all benchmarks on average by 2.7%, with the biggest benefit for SP, with a hit rate increase of 7%. Using 31 line buffers only increases the hit rate for SP. Splash-2 simulations confirm that selecting more than 15 line buffers delivers only incremental performance gains at significant area cost.

4.3 Prefetch Cache Replacement Policy

The replacement policy determines how streams are aged out of the prefetch cache to make room for new data. We model several replacement policies: round-robin, random, least recently used (LRU), and round-robin skipping the three most recently used (RRMRU). We also include optimal replacement, one that relies on future knowledge (and thus cannot be implemented in hardware), to show theoretical upper bounds for stream detection.

Figure 5 presents effects of varying replacement policies on miss rates for NAS. For all applications, the various replacement policies are positioned between optimal replacement and random replacement. As expected, LRU is the best choice for most applications, but is the most complex policy to implement in hardware. Both round-robin and RRMRU (round-robin with skipping the three most recently used lines) offer comparable performance to LRU. RRMRU offers better performance than round robin replacement. The RRMRU policy is as simple to implement in hardware as round-robin, requiring only the addition of two latches per line buffer to record the MRU status for the last three requests.

4.4 Support for Bidirectional Streams

All results so far assume streams with ascending addresses. We also explore whether bidirectional stream support (i.e., detecting and prefetching streams with positive and negative strides) helps performance. To implement bidi-

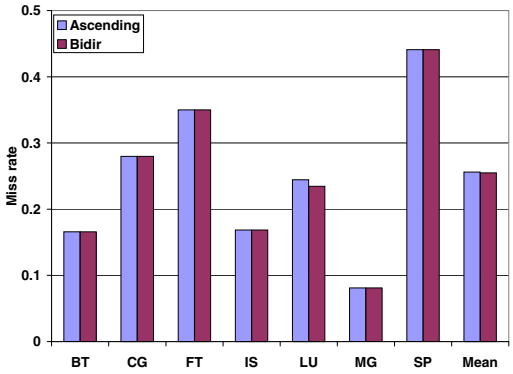

Figure 6. Bidirectional stream support using NAS benchmarks.

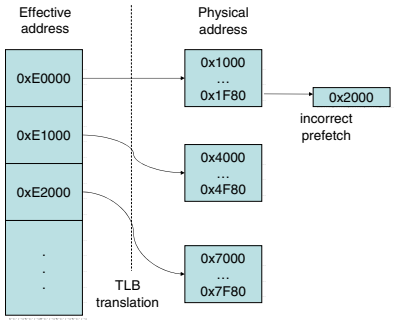

Figure 7. Linux page translation and prefetching.

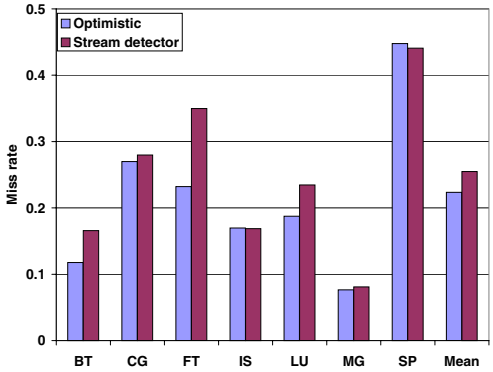

Figure 9. Miss rate for the optimistic and stream detector buffers prefetching for the NAS benchmark.

rectional stream support, each line buffer stores two extra bits recording the L1 line address of the first request. For subsequent requests to prefetch cache lines, the address of the new request is compared to that saved, and we determine whether the new address is descending or ascending compared to the previous request. Based on this, the next prefetch request is issued to access the ascending or descending address.

Figure 6 shows the effect of changing from ascending stream detection to bidirectional detectionr. There is no significant benefit from the bidirectional stream detector, indicating that there are no significant access patterns with negative strides present in these benchmarks. Also, scientific workloads, in general, do not show negative-stride streams.

4.5 Operating System Impact

We also explore the impact of using different operating systems on prefetch cache performance. We compare two models representing a full-fledged multithreaded UNIX operating system (Linux) and a streamlined single-threaded kernel solution (the compute node kernel CNK employed in Blue Gene/L [15]). CNK implements static mapping of virtual addresses to physical addresses. Linear mapping ensures that an application's access patterns in virtual address space are reflected in the physical address space of the memory subsystem. In comparison, a standard Linux kernel uses a 4kB page. Establishing page translations in response to demand paging causes the kernel to map continuous virtual address spaces to discontinuous physical 4kB pages, as illustrated in Figure 7. At page boundaries, the prefetch engine continues to prefetch from the contiguous physical address, which may not match the virtual address access pattern. Thus, streams must be re-established, and bandwidth and access efficiency is lost at page transitions. Figure 8 compares the impact of memory allocation policies in Linux and CNK on prefetch cache hit rates.

With small pages, the PowerPC440 processor's 64-entry TLB cannot contain the entire address space for memory and I/O devices of a Blue Gene node. Additional degradation is introduced when TLB entries must be reloaded. This is particularly expensive in environments without hardware-managed TLBs, where each miss causes an exception. This effect is somewhat mitigated in more recent Linux versions that introduce support for large pages.

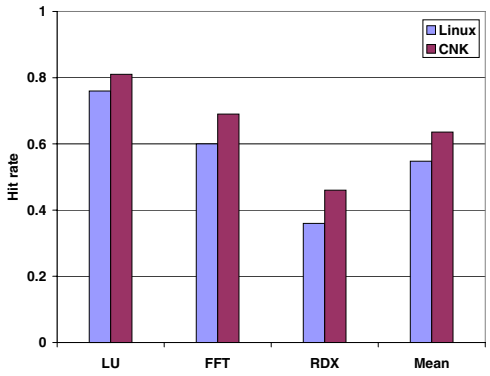

Figure 8. Splash-2 benchmarks on CNK and Linux.

4.6 Optimistic vs. Stream Detector Buffers

To evaluate the efficiency of the stream detection buffer, we compare it against optimistic prefetching (described in 2.1). Figure 9 gives miss rates for optimistic and stream detector prefetching schemes. For some benchmarks (BT, FT and LU), the optimistic approach yields lower miss rates, but for the others, both approaches yield roughly the

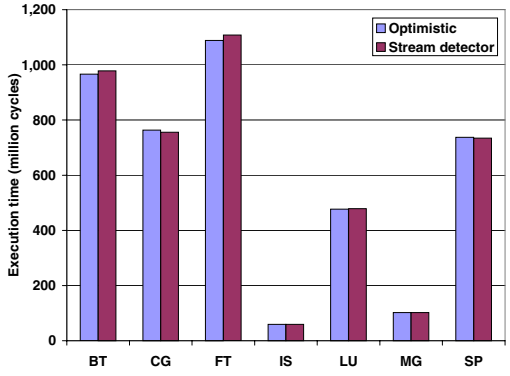

Figure 10. Execution time for the optimistic and stream detector buffers prefetch comparison for the NAS benchmark.

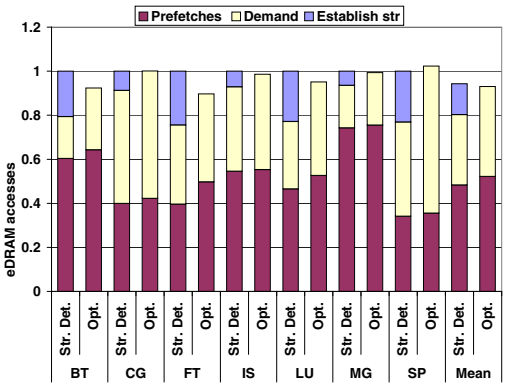

Figure 11. Normalized breakdown of eDRAM accesses for the optimistic vs. stream detector prefetching for the NAS benchmarks.

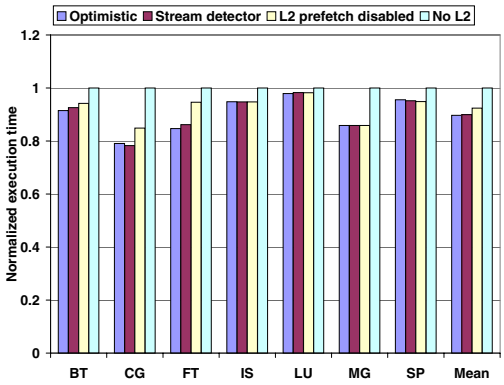

Figure 12. Normalized execution time for the two prefetch schemes, L2 with disabled prefetching, and without the prefetch cache.

same miss rates. We compare execution times for the two prefetching schemes, illustrated in Figure 10. Execution times for both approaches are remarkably similar for all NAS benchmarks, with the largest difference being 1.8%. The optimistic approach has shorter execution times for BT and FT, and the stream detection buffer shows better execution time for CG.

Optimistic prefetching is more aggressive and issues more prefetch requests to the L3, thus we expected that L3 bandwidth requirements would increase. However, the optimistic prefetcher produces fewer accesses for most workloads (as exemplified by the FT benchmark), while the stream detector produces fewer accesses for other workloads (as exemplified by SP), as illustrated in Figure 11. The breakdown of L3 accesses into categories gives more insight: we classify the number of L3 accesses into two broad request categories, demand requests and prefetch requests. For our stream detection buffers, we further classify demand fetches into demand requests and stream-establishing demand requests (i.e., a demand request hitting in the stream detection buffers and thereby causing a stream to be identified).

As expected, optimistic prefetching initiates a higher number of L3 prefetch accesses relative to the stream detector. However, the number of demand accesses is larger for stream detector prefetching, resulting in a larger number of total accesses. The breakdown of demand fetches for the stream detector shows the cause of those demand accesses: when a stream has not been detected, no buffer is allocated to store a wide L3 line for future accesses. As a result, two subsequent demand accesses (a demand access to an L3 line and a second, stream-establishing demand access to the same L3 line) are performed before establishing a stream. In comparison, optimistic prefetching immediately assigns a line buffer and retains the entire L3 cache line for future accesses, thereby obviating the need for performing more accesses to the same line.

4.7 Prefetch Cache Performance Characteristics

To evaluate the efficiency of the prefetch cache, we compare the two prefetch schemes — stream detector buffers and optimistic prefetching — with application performance results obtained when prefetching is disabled, but L2 line buffers are used. We also compare these schemes with the configuration without the L2 prefetch cache. Figure 12 shows normalized execution times for the four approaches. The prefetch cache reduces execution time for both prefetch methods by 12%, on average. The biggest improvement is achieved for the CG benchmark (22%), whereas for the LU benchmark the performance improvement is only 2%. Simulation results show a performance advantage when using prefetching versus just exploiting line buffers without prefetching. While line buffers reduce execution time by about 10%, on average, prefetching improves performance by an additional 2%–5% for all applications studied.

The second advantage of the prefetch cache is reduction in number of accesses to the L3, which reduces contention for the L3 cache port among the two L2 prefetch caches, the network interface, and the memory controller. Figure 13 shows normalized breakdown of L3 accesses for both prefetch schemes and without the L2 prefetch cache for the NAS benchmarks. The prefetch cache reduces L3 accesses by 60%, on average, for all NAS benchmarks. This is due to the streaming accesses exhibited by most scientific application, and thus buffering the eDRAM data in wide 128B prefetch cache lines dramatically reduces the num-

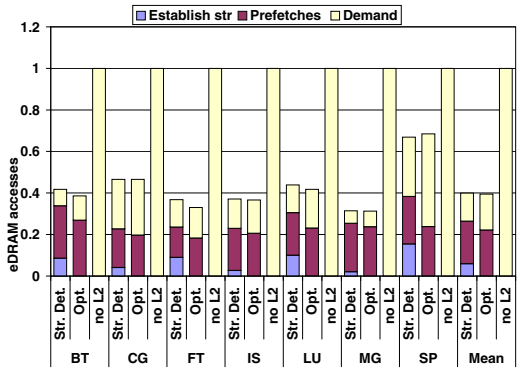

Figure 13. Normalized breakdown of eDRAM accesses for the two prefetch schemes, and for configuration without the prefetch cache.

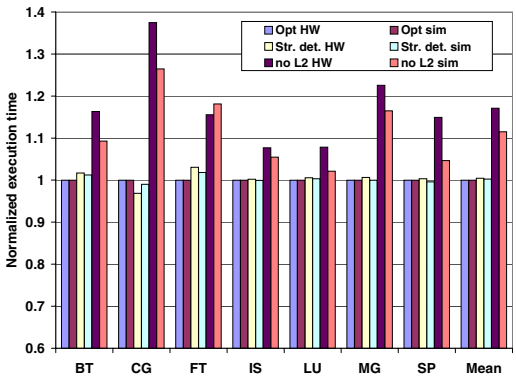

Figure 14. Hardware measurement execution time and simulated execution time (normalized) for the NAS benchmark.

ber of requests needed. The two prefetch schemes show remarkably similar characteristics in terms of execution time and eDRAM accesses.

5 Hardware Measurements

To verify our simulation results, we perform extensive empirical performance analysis. Here, we report hardware measurement results for NAS benchmarks. Figure 14 shows normalized execution times for three configurations implemented in hardware, and compares the measured execution times to simulated execution times for each benchmark. The three implemented hardware configurations are the two prefetch schemes and L2 disabled (which bypasses L2). Hardware measurements confirm trends shown by simulations: significant improvements in performance are due to the prefetch cache. Both hardware and simulation results are normalized results (e.g., hardware results are normalized using hardware optimistic prefetch results, and simulation results are normalized using optimistic prefetch simulation results) to eliminate systematic deviations between simulator and hardware measurements.

Figure 15 plots simulation error expressed as a difference of simulated execution time in cycles and measured hardware execution time over the corresponding measured

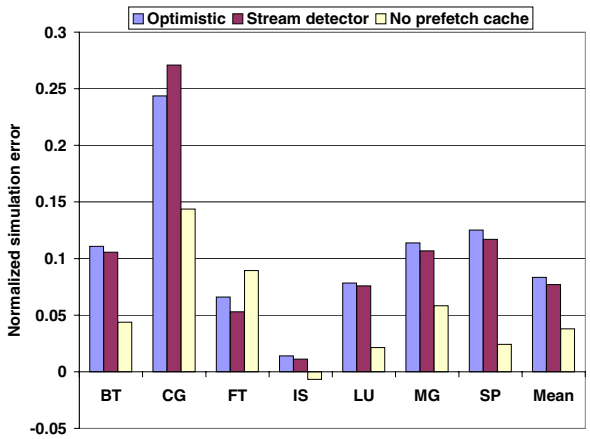

Figure 15. Simulation error against hardware measurements for the NAS benchmark.

hardware execution time. Simulation results are typically within 10-20% of actual hardware measurements, including all system effects and operating system interactions. Simulation results are *conservative* in projecting both baseline performance, and even more so, in modeled improvements. Simulation error compares favorably to the only other work published on correlating simulated results against actual hardware measurements for the FLASH system [10]. This confirms the quality of our simulation environment, and validates the decision to go with a full system simulator for the Blue Gene system.

6 Related Work

Data prefetching has been widely explored. Ideally, only needed data are prefetched, so the data are ready when the processor needs them. However, prefetching unneeded data reduces available memory bandwidth for other participants on the memory bus. Moreover, the data cache is polluted by prefetched data displacing useful data.

All prefetch schemes can be grouped into hardware prefetching, software prefetching, and hybrid techniques. Hardware prefetching needs no modification of existing executables, and can be implemented with relatively simple hardware. Software prefetching is generally based on application properties obtained at compilation time and/or run time. While it requires no hardware support, the application suffers additional overhead (e.g., code expansion), runtime cycles for prefetching instructions, and increased register usage.

Early work on cache prefetching includes the one-block-lookahead (OBL) scheme by Smith [21]. This approach initiates a prefetch for $(i + 1)$-th block into the D-cache when the i-th block is accessed. Jouppi [14] extends this idea by introducing external stream buffers to hold prefetched data. Palacharla [17] proposes several improvements to stream buffers. They limit the number of unnecessary prefetches by using a history buffer to detect data streams, and prefetch only for detected streams.

Gschwind [11] prefetches into stream buffers under program control. Prefetch streams are identified by prefetch register FIFOs, and software can specify arbitrary strides. The PowerPC architecture supports data stream prefetching into L1 cache with appropriate data-stream touch instructions. However, these approaches require significant programmer investment (or appropriate compiler support) to specify streams. Lee et al. [24] evaluate performance of several prefetching cache architectures for multimedia applications. Puzak et al. [18] discuss prefetching metrics and analyze potential for prefetching in SPECcpu and OLTP workloads. McKee [23] combines a stride-based reference prediction table to prefetch L2 cache lines, and a memory controller to dynamically schedules accesses, delivering good speedups for scientic applications.

Sequential prefetching techniques perform poorly for sequences of irregular access patterns, as in pointer chasing, where the code follows a serial chain of loads. The approach described in Collins et al. [9] uses a pointer cache to assist prefetching for pointer load sequences.

7 Conclusion

Large capacity eDRAM caches make high bandwidth access to high capacity on-chip storage possible by offering both wide data paths and high on-chip transfer speeds. In conjunction with chip-multiprocessor solutions, we can deliver increased performance at low power, with reduced bandwidth requirements to off-chip memory. eDRAM is characterized by low power, high density, and high bandwidth, but higher latency. Prefetching fundamentally trades bandwidth to hide access latency. This study presents an exhaustive analysis of design options for a prefetch cache designed specifically to interface to a large L3 cache implemented with embedded DRAM. We concentrate on performance of supercomputer applications, and consider operating system impact.We use full system simulation to generate representative cache miss behavior, including OS interaction, and use full L1 miss sequences to explore the prefetch cache design space.

In architectures where prefetching is implemented within a cache, careful prefetching is important, so as not to pollute the cache. This equation has changed for the architectures where prefetch cache lines are outside of the L1. Using wide prefetch cache lines captures spatial locality present in many applications (and, in particular, many HPC workloads), exploiting wide L3 lines of our eDRAM cache efficiently. High bandwidth of the shared eDRAM cache can sustain both processors' memory requests. This enables efficient data stream prefetching, reducing execution time. For the Blue Gene/L architecture, the prefetch cache's size is only 2kB per processor, yet it reduces execution time across many workloads by 10%, on average. While prefetching is not a complete solution to memory latency issues, we find that prefetching combined with high density on-chip eDRAM-based caches is a successful solution for the systems and applications we study.

8 Acknowledgments

The Blue Gene/L system resulted from the dedicated work of a large team, and we thank all members of that team. We acknowledge, in particular, contributions by Dirk Hoenicke, who was responsible for the design of the stream prefetching logic. We thank Sally A. McKee, John-David Wellman and Michael Gschwind for many useful discussions, and for their help in the preparation of this paper, and we thank Thomas Puzak and Ruud Haring for their suggestions during the preparation of this manuscript. The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratories on behalf of the United States Department of Energy, under Lawrence Livermore National Laboratories Subcontract No. B517552.

References

[1] L. R. Bachega et al. The BlueGene/L pseudo cycle-accurate simulator. In *2004 IEEE International Symposium on Perfomance Analysis of Systems and Software*, Austin, TX, March 2004.

[2] D. Bailey et al. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-929, NASA Ames Research Center, December 1995.

[3] M. Blumrich et al. A holistic approach to system reliability in Blue Gene. In *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*. IEEE Computer Society Press, 2006.

[4] P. Bohrer et al. Mambo – a full system simulator for the powerpc architecture. *ACM SIGMETRICS Performance Evaluation Review*, 31(4), March 2004.

[5] A. A. Bright et al. Creating the BlueGene/L supercomputer from low power SoC ASICs. In *Digest of Technical Papers, 2005 IEEE International Solid-State Circuits Conference*, pages 188–189, 2005.

[6] J. Brunheroto et al. Data cache prefetching design space exploration for Blue Gene/L supercomputer. In *Proc. of SBAC-PAD*, October 2005.

[7] L. Ceze et al. Full circle: Simulating Linux clusters on Linux clusters. In *4th LCI International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, CA, June 2003.

[8] L. Chang et al. Stable SRAM cell design for the 32 nm node and beyond. In *VLSI Symposium on Technology*, Kyoto, Japan, June 2005.

[9] J. Collins et al. Pointer cache assisted prefetching. In *35th Annual IEEE/ACM International Symposium on Microarchitecture MICRO-35*, 2002.

[10] J. Gibson et al. FLASH vs. (simulated) FLASH: closing the simulation loop. In *9th International conference on Architectural support for programming languages and operating systems*, Cambridge, MA, 2000. ACM Press.

[11] M. Gschwind et al. Vector prefetching. *ACM Computer Architecture News*, 23(5):1–7, December 1995.

[12] M. Gschwind et al. Exploiting fine-grained memory locality with predictive dispatch. IBM Research Report RC23633, IBM TJ Watson Research Center, Yorktown Heights, NY, 2004.

[13] S. S. Iyer et al. Embedded DRAM: Technology platform for the Blue Gene/L chip. *IBM Journal of Research and Development*, 49(2/3), 2005.

[14] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *17st International Symposium on Computer Architecture*, pages 364–373, May 1990.

[15] J. E. Moreira et al. BlueGene/L programming and operating environment. *IBM Journal of Research and Development*, 49(2/3), 2005.

[16] M. Ohmacht et al. Blue Gene/L compute chip: Memory and Ethernet subsystem. *IBM Journal of Research and Development*, 49(2/3), 2005.

[17] S. Palacharla et al. Evaluating stream buffers as a secondary cache replacement. In *21st International Symposium on Computer Architecture*, April 1994.

[18] T. Puzak et al. When prefetching improves/degrades performance. In *ACM Computing Frontiers 2005*, Ischia, Italy, 2005. ACM Press.

[19] V. Salapura et al. Power and performance optimization at the system level. In *ACM Computing Frontiers 2005*, Ischia, Italy, May 2005. ACM Press.

[20] V. Salapura et al. Exploiting workload parallelism for performance and power optimization in Blue Gene. *IEEE Micro*, 26(5), September 2006.

[21] A. Smith. Cache memories. *ACM Computer Surveys*, 14:473–530, September 1982.

[22] S. C. Woo et al. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, June 1995.

[23] C. Zhang and S. McKee. Hardware-only stream prefetching and dynamic access ordering. In *14th ACM International Conference on Supercomputing*, pages 167–175, Santa Fe, NM, May 2000.

[24] D. F. Zucker et al. Hardware and software cache prefetching techinques for MPEG benchmarks. *IEEE Transactions for Circuits and Systems for Video Technology*, 10(5):782–796, August 2000.