Exploring Memory Management Strategies in Catamount

Kurt Ferreira Kevin T. Pedretti Michael Levenhagen
Ron Brightwell
Sandia National Laboratories *
{kbferre,ktpedre,mjleven,rbbrigh}@sandia.gov

Presented at the Cray User Group Conference
Helsinki, Finland
May 2008

Abstract

In this paper, we describe how the mapping of virtual to physical memory that is set up by the operating system can have a significant and unexpected impact on the performance of STREAM [6] and a sparse solver "mini-application" that exhibits STREAM-like behavior. The Cray Performance Analysis Toolkit (CrayPAT) [2] is utilized to identify the cause of the performance degradation to be row buffer conflicts in the memory system. This is shown to be due to the internal architecture of modern commodity DRAM chips. To mitigate the performance impact of row buffer conflicts, we explore alternative strategies for constructing a process's virtual to physical memory mapping. Ultimately, this work points out another level of locality in the memory system, i.e., DRAM row buffers, that few system software and application developers are aware of and that can have a significant impact on performance.

1 Introduction

Many HPC applications are memory bound, meaning that their performance is dependent on the performance of the memory system. The gap between memory latency and processor performance is growing with each successive processor generation. Additionally, multi-core processors are putting more strain on the memory system, often resulting in reduced available memory bandwidth per core. This requires application developers to invest a great deal of effort to exploit the memory hierarchy to its full potential. Techniques such as tiling,

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

blocking, data structure re-arrangement, and careful orchestration of data access patterns are often employed to optimize memory system performance.

The mapping of virtual (logical) to physical memory is controlled by the operating system and is outside the control of application developers, but it can also have a significant impact on memory system performance. The CPU uses the map (e.g., a page table) to determine the physical address of each memory reference and then the memory controller maps the resulting physical address to a channel, rank, row, column, and bank address in memory. Streams of contiguous accesses to successive physical addresses should result in the highest possible performance, assuming a well-designed memory controller. However, accessing multiple streams simultaneously (e.g., accessing two arrays element-wise in a loop) provides potential for bank conflicts, which reduce performance.

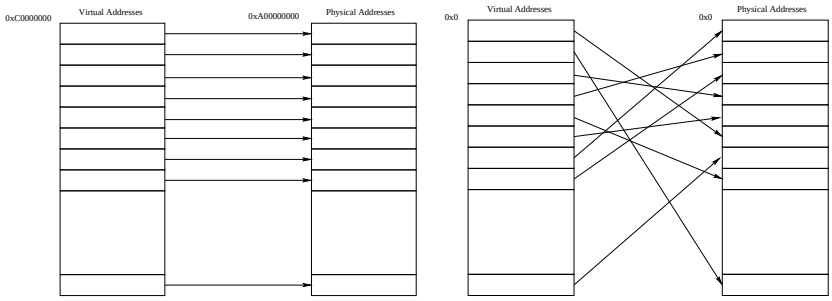
In this study, we show how the operating system's policy for mapping virtual to physical memory can work against a programmer's efforts to achieve optimal memory performance. The STREAM micro-benchmark is used to represent a best-case application that has been optimized for maximum memory bandwidth. Two memory mapping strategies are compared: the static, contiguous mapped strategy of Catamount, and the demand-paged strategy of Linux. Large swings in performance are observed depending on the mapping strategy and the relative alignment of the arrays accessed by STREAM. The results point to the importance of a level of cache in the memory hierarchy unknown to most application and system developers, the DRAM row buffer. We outline techniques to manage this cache and optimize locality at the system level.

The rest of this paper is organized as follows. The next section provides background on two virtual memory mapping strategies and a brief introduction to modern DRAM architecture. The details of our test platform and the micro-benchmarks from which we gathered performance data are described in Section 3. Performance results and analysis are provided in Section 4. Conclusions are summarized in Section 5 and future work is discussed in Section 6.

2 Background

In modern operating systems, each process executes in its own private virtual address space. This gives a process the illusion that it has a large and contiguous address space when in fact it may be scattered all around physical memory or not even loaded in memory at all (e.g., paged to disk). The operating system is in control of how the mapping of virtual to physical memory is performed. Most CPU architectures employ some form of table that must be initialized in order to specify the mapping (e.g., a 4-level page table on 64-bit x86 CPUs).

Catamount is a lightweight kernel (LWK) operating system that runs on the Red Storm system at Sandia National Laboratories. Catamount provides a limited but sufficient set of system services required for scalable scientific computing applications. On application load, Catamount creates a static virtual-to-physical mapping for the application. This maps all of physical memory (or the process's portion of physical memory) in one contiguous chunk. The im-



(a) Catamount (b) Compute Node Linux

Figure 1: Example Virtual to Physical Page-Table Mapping

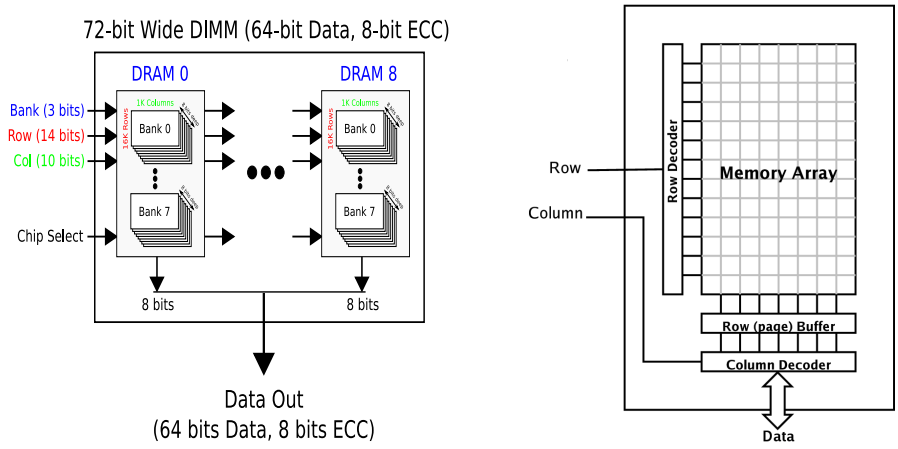
plication is that two contiguous locations in virtual memory will be physically continuous even if those addresses cross a page boundary. Figure 1(a) shows a representation of this mapping strategy. The contiguous mapping strategy greatly simplifies the network stack by allowing memory regions to be validated using a simple base address and length calculation. Additionally, the contiguous mapping strategy allows for more efficient pipelining of large sequential memory accesses and therefore increases overall memory bandwidth.

Compute Node Linux (CNL) is Cray's next-generation operating system for Cray XT systems, replacing Catamount. CNL is a Linux kernel that has been modified to be lighter-weight than standard Linux kernels. Since it is based on Linux, it uses a demand-paged memory mapping strategy where a process's mapping to physical memory is determined on-the-fly as the process executes. This results in an application's memory being scattered throughout the physical address space in a more-or-less random way. Figure 1(b) shows a representation of this mapping strategy.

2.1 DRAM Architecture

Figure 2 shows the internal organization of an example commodity DIMM (Dual-inline Memory Module) and DRAM bank. Groups of DRAM chips are arranged in parallel sets, called a rank. DIMM modules package one or more ranks together. The memory controller uses the chip select lines to specify which rank should be accessed. Only one of a DIMM's ranks may be accessed at a time.

All of the DRAM chips within a rank are addressed by the same bank, row, and column address wires. In Figure 2(a), one rank is shown consisting of 8 DRAM chips for data and 1 DRAM chip for ECC (Error Correcting Code). This is a common arrangement for one gigabyte DDR2 DIMMs. Each (bank, row, column) address specified by the memory controller identifies 64 data bits (8 bits per DRAM chip) and 8 ECC bits. DDR2 DRAM devices mandate burst accesses of length four or eight, so each access from the memory controller will



(a) DIMM

(b) DRAM Bank

Figure 2: Internal organization of an example DIMM and DRAM bank

return a minimum of $64 * 4$ data bits, or 32 bytes.

Internally, each DRAM chip is divided into multiple banks. Memory controllers typically seek to interleave contiguous physical address regions across the available banks, therefore increasing the ability to pipeline requests. The interleaving typically occurs on DRAM row boundaries since accessing data within the same row (i.e., an open row) is faster than switching banks. In the example shown, each DRAM bank has a row size of 1024 bytes. Each rank has an aggregate row size of $1024 * 8$ data bytes.

Figure 2(b) illustrates a conceptual view of the internal structure of a bank. When a (row, column) request arrives at the bank, one of two things happens. If the request is to the same row as the previous request, the item will already be present in the row buffer and the appropriate column can simply be read out and returned. Things are much more complicated if the access is to a different row (i.e. a closed row). The open row must first be written back to memory (PRECHARGE) and then the desired row read into the row buffer (ACTIVATE). In modern DRAM chips, the PRECHARGE and ACTIVATE operations take anywhere from three to ten memory clock cycles (e.g., 9 to 30 ns for DDR2-667 memory) and can therefore dramatically reduce memory bandwidth. The row buffer is another level of cache in the memory system and it is advantageous to exploit as much row buffer locality as possible.

3 Test Environment

This section provides an overview of the test hardware platform, micro-benchmarks, and performance monitoring tools used for this study.

3.1 Hardware Platform

All testing was performed using a single cage Cray XT4 development system with 28 compute nodes. Each compute node contained a 2.4 GHz dual-core AMD Opteron processor and 4 GB of memory, comprised of two 2 GB dual-rank DDR2-667 DIMMs (each rank providing 1 GB). The system was booted into either Catamount or a recent Compute Node Linux release (2.0.35) for testing.

The dual-core Opteron's memory controller supports a 128-bit wide memory bus, interleaved across two DDR2 memory channels. Each DIMM was connected to a separate memory channel. The aggregate DRAM row width of the two channels is 16 KB with each channel providing 8 KB. Each rank consists of 8 banks, resulting in 16 banks total for the two rank memory configuration. The physical address space is interleaved first among the banks within a rank. For example, the first 16 KB of the address space is stored in the first row of bank 0 on rank 0, the next 16 KB on the first row of bank 1 on rank 0, and so on until all eight ranks are visited. Then, rather than switch back to bank 0 on rank 0, interleaving was performed across ranks so that the next 16 KB is stored in the first row of bank 0 on rank 1. This strategy allows more rows to be kept open than if the memory supplied by one rank is fully used before switching to the next rank. Note that adding additional DIMMs to each channel would result in more ranks and banks, potentially improving performance.

Both bank and rank interleaving are configurable parameters of the Opteron's memory controller [1] and are setup at system boot time by the system BIOS, or Cray's Coldstart in the case of the XT systems (Coldstart is a BIOS replacement). The bank and rank interleaving configurations that were used for this study are typical for dual-core Opteron systems.

3.2 Micro-benchmarks

The STREAM micro-benchmark is used to examine memory bandwidth. STREAM utilizes four kernels to characterize the memory bandwidth performance of a system: COPY, ADD, TRIAD, and SCALE. Each of these kernels operates on two to three arrays. The COPY kernel copies the contents of one array sequentially into another. The ADD operation sequentially performs a vector addition of two arrays into a third array. Similarly, TRIAD performs a vector addition of two arrays where one of those arrays is multiplied by a scalar value with the result stored into a third array. Lastly, the SCALE operation sequentially multiples one array by a scalar value and stores that result into a second array.

In addition to STREAM, the HPCCG [3] mini-application is used to investigate whether bank conflicts affect more than just the STREAM micro-benchmark. HPCCG is a simple sparse conjugate gradient solver designed to capture an important component of Sandia's production workload, and therefore be useful for performance evaluation. The majority of its runtime is spent performing sparse matrix-vector multiplies, where the sparse matrix is encoded in compressed row storage format. It therefore has similar memory access patterns

to STREAM.

3.3 Performance Analysis

The CrayPAT toolkit [2] is used to monitor the Opteron CPU and memory controller performance counters. CrayPAT internally uses the PAPI [5] API, which is provided on both Catamount and Compute Node Linux, to gather performance counter data. PAPI defines a standardized set of performance counters that are portable across different platforms as well as a native interface to access performance counters that are specific to a particular platform. Since neither CrayPAT or PAPI provide standardized memory controller performance counters, we use the more complicated PAPI native interface.

4 Results

4.1 STREAM on Catamount

Figure 3 shows the STREAM COPY performance as a function of array OFFSET on Catamount. OFFSET is used to control the relative spacing in memory of the arrays accessed by the benchmark.[1] The default value of OFFSET is 0, resulting in the A, B, and C arrays being directly adjacent to one another in memory. Non-zero values of OFFSET add padding between each array. For example, a value of 1000 would add 1000 doubles (8000 bytes) of padding between the A and B arrays and the between B and C arrays. The STREAM documentation states that modifying the value of OFFSET is necessary on some systems to obtain optimal performance, but the common practice today is to leave it set to the default value of 0. Note that changing the array sizes used by STREAM (the value of N) has the same effect as modifying OFFSET.

The large dips at offsets 120000 and 250000 bytes in Figure 3 represent nearly a 60% decrease in memory bandwidth and are spaced approximately 128 KB apart. Figure 4 shows similar but less regular behavior for the TRIAD kernel. In fact, all of the STREAM kernels show the behavior but we limit our discussion to COPY to simplify the analysis. Since COPY involves only two arrays it produces more regular patterns than the other kernels that access three arrays, but our analysis is relevant to all of the kernels.

As can be seen in Figure 5, the dips do not occur on CNL but the maximum memory bandwidth is lower for CNL. This was very puzzling to us since the original hypothesis was that the sawtooth pattern was due to caching or TLB effects. We spent considerable time investigating whether something about Catamount was resulting in the Opteron's memory controller begin setup incorrectly. This proved not to be the case.

Eventually we stumbled upon the fact that the Opteron's memory controller provides a set of performance counters. After employing a "try-them-all" approach, it was determined that the cause of the performance dips was due to

[1] OFFSET is a pre-processor define specified at the beginning of the STREAM source code.

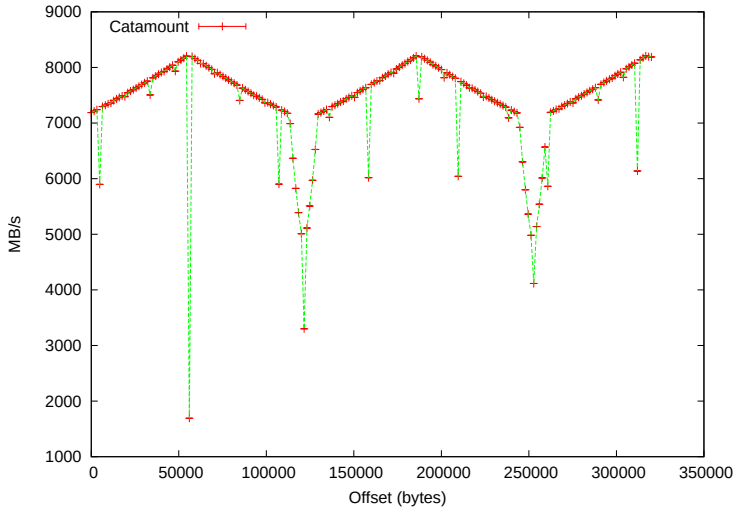


Figure 3: STREAM COPY performance on Catamount as a function of array OFFSET.

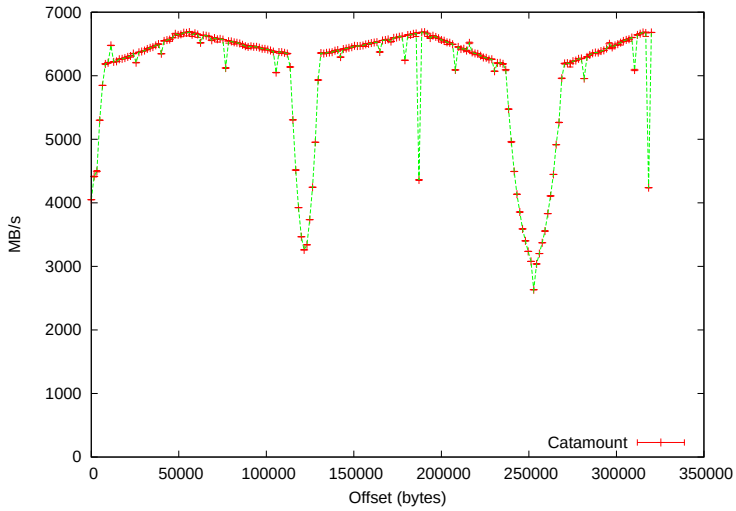


Figure 4: STREAM TRIAD performance on Catamount as a function of array OFFSET.

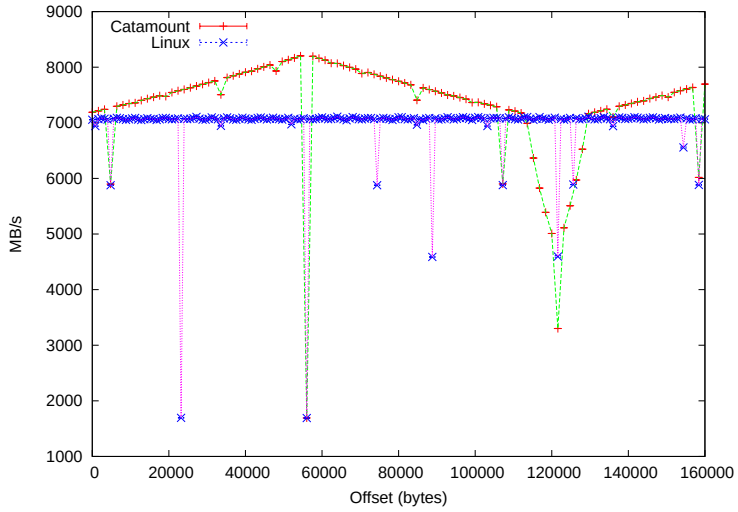


Figure 5: STREAM COPY comparison of Catamount and CNL.

DRAM row buffer conflicts (also known as DRAM page conflicts), shown in Figure 6. Recall from Figure 2(b) that the row buffer is effectively a large cache for a DRAM bank. A DRAM row buffer conflict occurs when a sequence of memory requests correspond to different rows of the same memory bank. These conflicts affect performance because the currently open row must be written back to memory (PRECHARGE) and the next row must be read into the row buffer (ACTIVATE). These operations cannot be pipelined.

For STREAM COPY, the worst case is when the source and destination arrays are situated in physical memory such that each iteration of the copy loop causes a row buffer conflict. The 128 KB spacing of the worst-case dips corresponds to our test system's number of DRAM banks (16) multiplied by the aggregate DRAM row buffer size (16 KB) divided by two. The divide by two factor is because the STREAM copy kernel actually uses the A and C arrays, so the impact of the OFFSET value is doubled. In the worst case, A[i] and C[i] conflict for all values of i.

A great deal of work has gone into trying to mitigate row buffer conflicts at the hardware level [4, 7, 8, 9, 10, 11]. Well-designed memory controllers attempt to schedule memory accesses to maximize hits to open rows and take advantage of as much row buffer locality as possible. Many memory controllers support multiple configurable strategies for mapping the physical address space to the DRAM chips (i.e., memory interleaving and bank swizzle mode on the AMD Opteron). While these methods may reduce the chance of row buffer conflicts, they are at a level invisible to the operating system and applications.

In contrast to hardware-based approaches, we attempted to mitigate row buffer conflicts by utilizing a software-based approach that takes advantage of the operating system's control over the virtual to physical memory map. Cata-

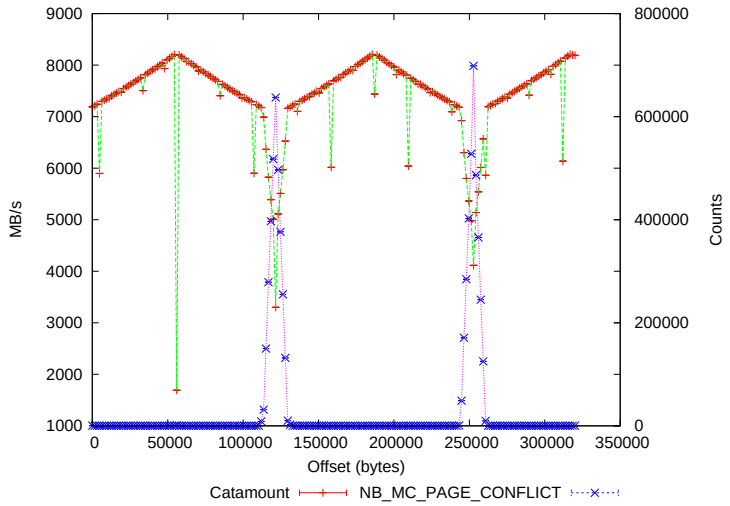


Figure 6: STREAM COPY performance on Catamount with corresponding `NB_MC_PAGE_CONFLICT` performance counter data.

mount was modified to monitor the row buffer conflicts of a running application and then shuffle the memory mapping if the number of page conflicts exceeded a threshold per unit time. This type of memory adaptation is fairly straightforward to do in Catamount due to its simple memory management scheme.

Figure 7 shows the results of running STREAM using this adaption technique. We see the number and impact of row buffer conflicts is reduced. It is important to note that this technique worked well for STREAM because it performs many trials and only reports the best result. This specific adaptation approach may not work well for more complicated applications, but we believe that the general approach of having the OS monitor performance counters and adapt as necessary is promising. Compile-time analysis is another technique that may be beneficial.

4.2 STREAM on CNL

Figure 5 shows that Catamount and CNL demonstrate significantly different behavior. This is due to the virtual to physical memory mapping strategies employed by the two operating systems. Catamount uses a one-to-one mapping between virtual and physical memory. For the STREAM COPY kernel, this means that once a row buffer conflict is encountered it is likely that subsequent iterations will also conflict, even if a DRAM row boundary is crossed. In contrast, Linux employs a demand-paged strategy where the virtual to physical mapping is setup on-the-fly as a program executes. This results in a more-or-less randomized mapping. On our test system, the mapping is at a 4 KB granularity since that is the memory management page size used by the Linux kernel on

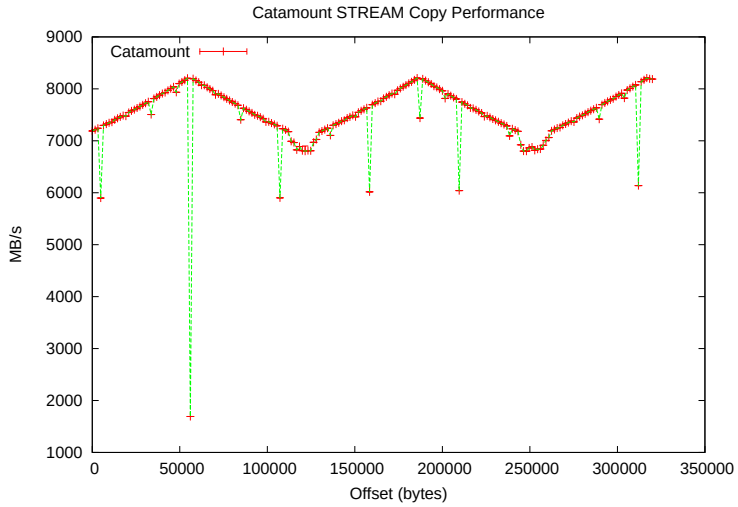


Figure 7: STREAM COPY performance on Catamount with adaptive row buffer conflict mitigation.

x86-64 systems. This means that runs of row buffer conflicts will usually be limited to at most 4 KB, since it is highly unlikely that the location of adjacent 4 KB memory management pages will map to different rows of the same DRAM bank.

The STREAM results in Figure 8 are from a freshly booted CNL node. Since Linux uses a demand-paged strategy, performance can vary from run to run depending on which physical pages are available for allocation and how fragmented they are. To investigate possible worst-case page frame allocations, a series of applications was run in order to fragment memory. Figure 9 shows the increased variance that results after doing this compared to the results of the freshly booted system. In both figures, the low-performance outliers are due to both page conflicts and cache conflicts.

4.3 HPCCG on Catamount

In this section we look at the effect memory mapping has on the HPCCG mini-application. The majority of this application's runtime is spent performing sparse matrix vector multiplications. A pair of arrays are used to represent the sparse matrix, one storing the non-zero values and another storing the column index of each non-zero. The HPCCG code was instrumented to allow for variable offsets between these two arrays in order to investigate the effect that memory placement has on performance.

Figure 10 shows the performance of HPCCG on 28 nodes for a variety of offsets. The offset value on the x-axis represents the number of bytes between the two arrays. As with STREAM, certain offsets lead to significantly degraded

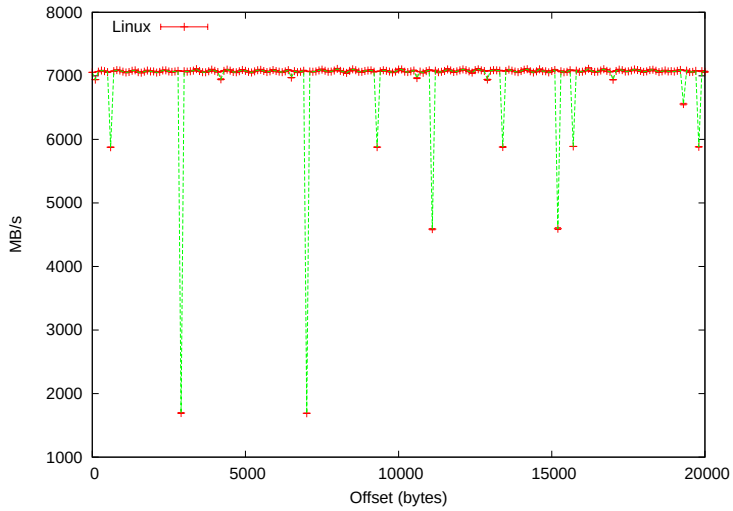


Figure 8: STREAM COPY performance on a freshly booted CNL node.

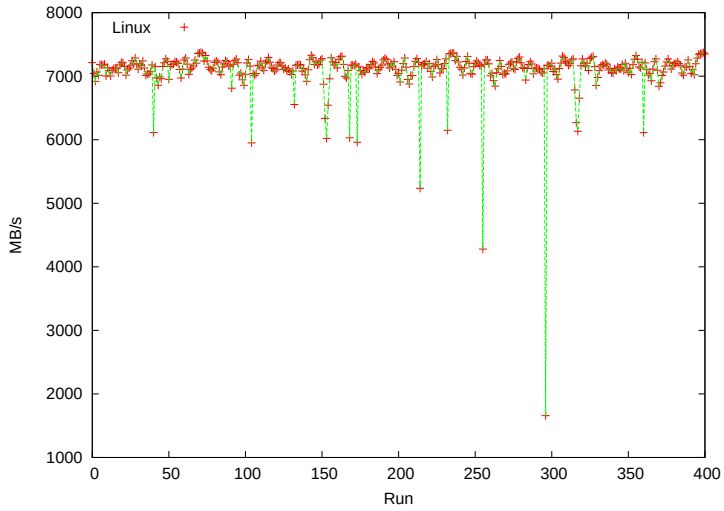


Figure 9: STREAM COPY performance of CNL long after boot. Memory becomes more fragmented over time, leading to increased performance variability.

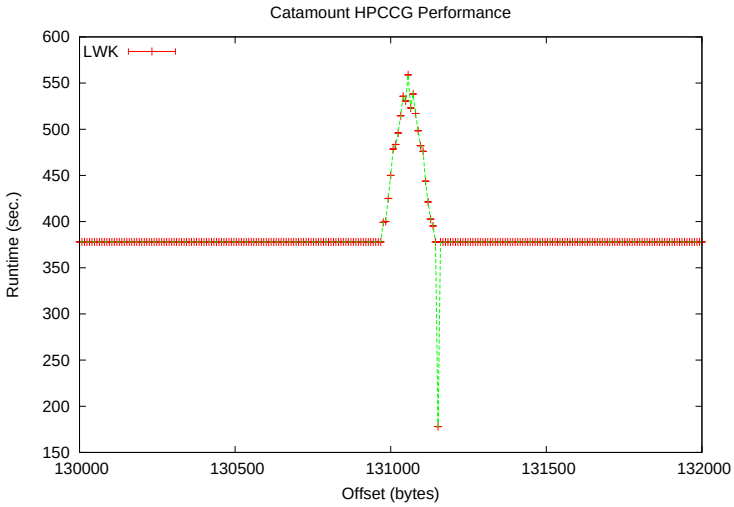


Figure 10: HPCCG runtime performance on Catamount for various array offsets.

performance due to DRAM row buffer conflicts (lower is better in this graph).

5 Conclusion

We have described how the mapping of virtual to physical memory that is setup by the operating system can have a significant and unexpected impact on the performance of memory bandwidth bound applications. This has been shown to be due to row buffer conflicts caused by the internal architecture of commodity DRAM chips. The memory mapping strategies of two operating systems—Catamount and CNL—have been analyzed. The simple one-to-one mapping strategy used by Catamount results in slightly better best-case performance, but rarely results in very poor performance for certain data alignments. The demand-paged layout used by CNL reduces the chance of hitting the rare worst case at the cost of sacrificing 10% to 20% of peak memory bandwidth. To mitigate the worst-case behavior observed with Catamount, we developed a simple operating system controlled adaptive mapping strategy.

6 Future Work

We plan to further investigate the effect of DRAM row buffer conflicts on the performance of real applications. This may be the cause of non-IO related performance differences between Catamount and CNL. We also plan to explore more closely the impact of multi-core processors on row buffer conflicts.

References

[1] AMD. *BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors*, 2007.

[2] Cray Inc. *Using Cray Performance Analysis Tools*, December 2007. `http://docs.cray.com/books/S-2376-41/S-2376-41.pdf`.

[3] Mike Heroux. HPCCG MicroApp. `http://www.cs.sandia.gov/~maherou/HPCCG-0.3.tar.gz`, July 2007.

[4] Bruce Jacob. A Case for Studying DRAM Issues at the System Level. *IEEE Micro*, 23(4):44–56, 2003.

[5] Innovative Computing Laboratory. PAPI homepage. `http://icl.cs.utk.edu/papi/index.html`.

[6] John D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`.

[7] B. Ramakrishna Rau. Pseudo-Randomly Interleaved Memory. *SIGARCH Comput. Archit. News*, 19(3):74–83, 1991.

[8] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter R. Mattson, and John D. Owens. Memory Access Scheduling. In *ISCA*, pages 128–138, 2000.

[9] Jun Shao and Brian T. Davis. The Bit-reversal SDRAM Address Mapping. In *Proceedings of the 9th International Workshop on Software and Compilers for Embedded Systems (SCOPES05)*, pages 62–71, Sept. 29 - Oct. 1 2005.

[10] Jun Shao and Brian T. Davis. A Burst Scheduling Access Reordering Mechanism. In *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 285–294, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *International Symposium on Microarchitecture*, pages 32–41, 2000.