Exploiting Content Similarity to Improve Memory Performance in Large-Scale High-Performance Computing Systems

Scott Levy
4th year Ph.D. student / Advised by Patrick G. Bridges
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico
slevy@cs.unm.edu

I. Motivation

As we consider building the next generation of extreme-scale systems, many of the biggest challenges are related to memory characteristics. In particular, overcoming challenges related to resilience and memory bandwidth will require innovative strategies for improving the performance of main memory.

DRAM ECC failures are one of the most frequently observed sources of node failure in large-scale distributed systems [1]. Because the rate at which failures occur increases proportionally to the number of processors, larger, more powerful systems will experience more frequent failures [2]. As a result, traditional approaches to fault tolerance (e.g., coordinated checkpoint/restart) may no longer be sufficient to efficiently recover from these errors [3]. Moreover, power concerns may exacerbate this problem as we consider deploying low voltage memory chips that are more prone to error [4].

Memory bandwidth limitations will restrict our ability to fully exploit the increasingly powerful multicore processors that will compose future systems [5]. Because processor clock rates have plateaued, growth in total computational power has been achieved by increasing the number of cores per processor. However, the rate at which the number of cores is growing is outstripping the rate at which memory access speeds are increasing. As a result, the effective memory bandwidth available to each core is decreasing over time.

II. Proposed Approach

In this paper, we propose to exploit memory content similarity to improve memory performance. We begin by presenting several novel strategies that leverage memory content similarity to improve system resilience and effective memory bandwidth. Additionally, we seek to understand the source of similarity in the memory of HPC applications.

A. Resilience

1) DRAM ECC Errors: When an uncorrectable ECC error is detected in an x86 system, the memory controller raises a Machine Check Exception (MCE) in the processor. The consequences of raising an MCE vary by operating system. Recent versions of Linux attempt to minimize the impact of a MCE by adopting simple recovery strategies. In the event that none of its recovery strategies is successful, Linux poisons the hardware page and kills all of the processes that had the faulted page mapped into their address space [6]. In other operating systems (e.g., the Kitten lightweight kernel [7], older versions of Linux), raising a MCE simply crashes the node.

We can prevent some ECC DRAM errors from leading to node failure by exploiting similarities in main memory. The basic idea is that when a memory error occurs on a page that is similar to one or more other pages in the address space of an application, we can use the contents of the similar page to reconstruct the contents of the faulted page without needing to terminate the affected application.

2) Silent Data Corruption: Not all memory errors can be corrected (or even detected) by ECC. Although soft errors (e.g., those caused by cosmic rays) only rarely result in multiple bit errors within a single block, the consequences of silent data corruption are severe. For many undetected errors, the only observable effect is that the application produces an incorrect result [8]. Moreover, ECC requires additional DRAM, which increases system cost and energy consumption.

We can detect and potentially correct silent data corruption by exploiting memory content similarity. For example, if we know that two pages are similar to each other, we can periodically compare the contents of the pages to ensure that the difference between the two remains the same. When we detect a change in the difference between the two pages without a modifying write to either page, then we know that one of the two pages has been corrupted. If we know the difference between three or more pages (e.g., three duplicate pages), we can also potentially correct the error.

B. Memory Bandwidth

1) NUMA: In NUMA architectures, processors can access not only their local memory but also the local memory of other processors. However, accessing remote memory is comparatively slow. We propose to leverage memory content similarity to improve effective memory bandwidth by replacing accesses to slow, remote memory with accesses to similar fast, local memory.

2) Cache: One of the most established ways of reducing the impact of limited memory bandwidth is caching. Storing frequently-used data in close, high-speed memory conceals memory bandwidth limitations from the processor. We propose to use memory content similarity to improve the effective memory bandwidth of a processor by increasing the cache hit rate. For example, if a memory request would result in a cache miss but a cache line from a similar page is resident in cache, we may be able to leverage memory content similarity to satisfy the request from the cache instead of going to memory.

C. Source of Memory Similarity

A growing body of evidence suggests that significant similarity exists in the memory of HPC applications [9], [10], [11]. Section III contains a brief summary of the data that we have collected in support of this proposition. We propose to examine the relationship in HPC applications between regions of similarity in memory and application structure. To this end, we have begun to collaborate with experts on key HPC applications to correlate the data that we have collected with the structure of their applications.

III. Preliminary Results

Our initial inquiry was to examine the viability of this approach by evaluating: (a) the prevalence of memory content similarity in several important HPC applications; and (b) the cost of exploiting similarity in these applications. We present a synopsis of our data here; a more detailed discussion of all of the data that we collected is available in [11].

A. Data Collection

We collected snapshots of the applications' memory by linking `libmemstate` against each of the target applications. `libmemstate` is a library that uses the MPI Profiling layer to interpose itself in MPI calls made by the application. It begins by intercepting the call to `MPI_Init` and setting a timed signal (`SIGALRM`). Each time the signal expires, `libmemstate` reads the `/proc/<pid>/maps` file to gather information about the application's address space. Based on the information it gathers, `libmemstate` writes a copy of the address space to stable storage. Each snapshot includes all of the application's heap, stack and anonymous memory.

B. Memory Content Overview

We examined the memory of the eight HPC applications in Table I to ascertain the extent of memory content similarity in HPC applications. We generated the data presented in this paper by running each application using 64 MPI ranks equally distributed across 8 nodes of a Cray XE6 supercomputer. We began our analysis by placing each page in the address space of an application into one of four categories:

- DUPLICATE PAGES : pages whose contents exactly match one or more other pages and include at least one non-zero byte.
- ZERO PAGES : pages whose contents are entirely zero.
- SIMILAR PAGES : pages that (a) are not duplicate or zero pages; and (b) can be paired with at least one other page in application memory such that the difference between the two can be represented by a `cx_bsdiff` [21] patch that is smaller than a predetermined threshold.
- UNIQUE PAGES : pages that do not fall into any of the preceding three categories.

Given this system of page categorization, Figure 1 presents the fraction of each application's address space that falls into each of these four categories. We produced the data in this figure by increasing the patch size threshold for each application until the patches occupied just less than 5% of the application's total memory.

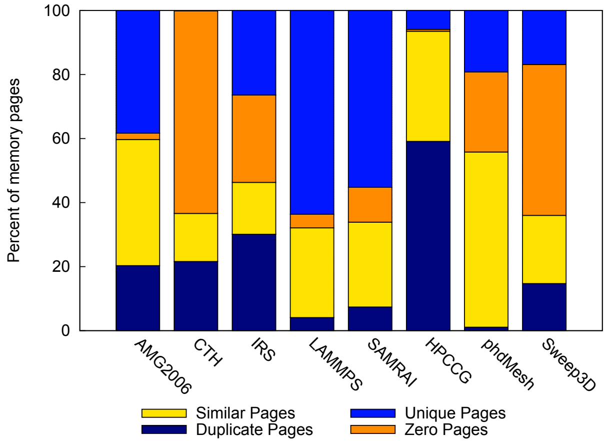

Figure 1. Page categorization within Rank 0 for each application. Each bar represents the page categorization for the memory snapshot that contained the smallest fraction of similar and duplicate pages. The total size of all of the patches for each application is less than 5% of application memory.

The key observation we make is that the memory of each of the applications is comprised of at least 32% similar and duplicate pages. Moreover, the memory of four of the applications (AMG, IRS, HPCCG and phdMesh) contain more than 45% similar and duplicate pages.

C. Overhead

The overhead of exploiting memory content similarity comes primarily from two sources: (a) memory to store metadata; and (b) time to maintain metadata. The bulk of the metadata that must be stored is patch data. For the data presented in this paper, the patch data would occupy less than 5% of application memory.

ASC Sequoia Marquee Performance Codes [12]	AMG	A parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids [13].
	IRS	Implicit Radiation Solver. Solves the radiation transport equation by the flux-limited diffusion approximation using an implicit matrix solution [14].
DOE Production Applications	CTH	A multi-material, large deformation, strong shock wave, solid mechanics code [15]
	LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator. A classical molecular dynamics simulator [16].
Mantevo Mini-Applications [17], [18]	HPCCG	Designed to mimic the finite element generation, assembly and solution for an unstructured grid problem.
	phdMesh	Parallel Heterogeneous Dynamic Mesh. An application designed to mimic the contact search applications in an explicit finite element application.
Miscellaneous Applications	SAMRAI	Structured Adaptive Mesh Refinement Application Infrastructure. Designed to enable the application of structured adaptive mesh refinement to large-scale multi-physics problems [19].
	Sweep3D	Solves a 1-group time-independent discrete ordinates (Sn) 3D cartesian (XYZ) geometry neutron transport problem. [20]

Table I
A BRIEF SUMMARY OF HPC APPLICATIONS USED

Application	Changed 1+ Times	Changed 1 Time	Changed 2 Times	Changed 3 Times	Changed 4+ Times
AMG2006	20.2 %	12.0 %	3.8 %	3.0 %	1.5 %
CTH	39.1 %	6.5 %	3.5 %	14.1 %	15.0 %
IRS	31.7 %	17.2 %	0.1 %	0.0 %	14.4 %
LAMMPS	11.0 %	0.2 %	0.1 %	0.0 %	10.7 %
SAMRAI	82.2 %	16.0 %	6.9 %	35.9 %	23.4 %
HPCCG	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
phdMesh	6.9 %	1.7 %	0.4 %	0.3 %	4.4 %
Sweep3D	2.5 %	1.4 %	0.6 %	0.0 %	0.5 %

Table II
MODIFICATION BEHAVIOR OF THE PAGES IN THE MEMORY OF RANK 0 THAT ARE EVER CATEGORIZED AS SIMILAR OR DUPLICATE.

The magnitude of the temporal overhead will depend largely on how frequently similar and duplicate pages are modified. Each time a page that has been classified as similar or duplicate is modified, we no longer know whether the page is similar or duplicate. As a result, we need to update our metadata to account for this change. The more rapdily that similar and duplicate pages change, the higher the temporal overhead of managing metadata will be.

To get a sense of how frequently similar and duplicate pages change, we compared the memory contents across the sequence of snapshots we collected for each application. By hashing each page, we were able to determine whether a given page in the application's virtual address space changed from one snapshot to the next.

This table shows the modification behavior for all of the pages in application memory that are *ever* classified as duplicate or similar given a 128-byte patch size threshold. The data in this table suggest that for most applications, a substantial majority of the similar and duplicate pages are either read-only/read-mostly or are written to without being modified [22]. For six of the eight applications (AMG, IRS, LAMMPS, HPCCG, phdMesh and Sweep3D), more than 85% of the similar and duplicate pages are modified either once or not at all.

IV. RELATED WORK

The technique of de-duplication has been used in virtualization, [23], [24], [25], HPC systems [9], [10], and in storage/backup applications [26]. However, to our knowledge, our proposed application of memory content similarlity to problems that are not directly related to data storage requirements is novel. Moveover, we are the first to consider similar memory in HPC applications.

In Linux, the machine check exception handler attempts to absorb faults that occur in memory that is not owned by a running process or can be read from a backing store [6]. However, we are aware of no work that allows a system to withstand an ECC DRAM error without re-launching the affected applications.

Recent work on improving memory bandwidth demands has largely focused either compiler techniques for efficient cache reuse [27] or data compression techniques that allow more application data to be delivered to the processor in fewer cache lines [28]. In [29] the authors propose a caching strategy for reusing duplicate cache lines in the context of multi-execution. We propose to exploit similarity as well as duplication to improve cache behavior.

Although significant data has been collected on deduplication, there has been little that explores the source of the similarity. In [30], the authors examine the source of page sharing for several virtualization workloads. A similar evaluation has not been performed for HPC applications.

REFERENCES

[1] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature

of DRAM errors and the implications for system design," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '12. New York, NY, USA: ACM, 2012, pp. 111–122. [Online]. Available: http://doi.acm.org/10.1145/2150976.2150989

[2] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN2006)*, Jun. 2006. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/stray/dsn06_abs.html

[3] K. Ferreira, R. Riesen, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, P. Bridges, D. Arnold, and R. Brightwell, "Evaluating the viability of process replication reliability for exascale systems," in *Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage, and Analysis, (SC'11)*, Nov 2011.

[4] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," in *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*. IEEE, 2008, pp. 114–122.

[5] B. Rogers, A. Krishna, G. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for CMP scaling," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 371–382.

[6] A. Kleen, "mcelog: memory error handling in user space," in *Proceedings of Linux Kongress 2010*, Nuremburg, Germany, September 2010.

[7] Sandia National Laboratory, "Kitten lightweight kernel," https://software.sandia.gov/trac/kitten, March 10 2012.

[8] C. Lu and D. Reed, "Assessing fault sensitivity in MPI applications," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2004, p. 37.

[9] S. Biswas, B. R. d. Supinski, M. Schulz, D. Franklin, T. Sherwood, and F. T. Chong, "Exploiting data similarity to reduce memory footprints," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 152–163. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2011.24

[10] L. Xia and P. A. Dinda, "A case for tracking and exploiting inter-node and intra-node memory content sharing in virtualized large-scale parallel systems," in *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing*, ser. VTDC '12. New York, NY, USA: ACM, 2012, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/2287056.2287061

[11] S. Levy, K. B. Ferreira, P. G. Bridges, A. P. Thompson, and C. Trott, "An Examination of Content Similarity within the Memory of HPC Applications," Sandia National Laboratory, Tech. Rep. SAND2013-0055, 2013.

[12] Lawrence Livermore National Laboratories, "ASC Sequoia Benchmark Codes," https://asc.llnl.gov/sequoia/benchmarks, August 2009.

[13] V. Henson and U. Yang, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[14] Lawrence Livermore National Laboratories, "IRS: Implicit Radiation Solver 1.4 Build Notes," https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/irs/irs.readme.html.

[15] J. McGlaun, S. Thompson, and M. Elrick, "CTH: a three-dimensional shock wave physics code," *International Journal of Impact Engineering*, vol. 10, no. 1, pp. 351–360, 1990.

[16] Sandia National Laboratories, "The LAMMPS molecular dynamics simulator," http://lammps.sandia.gov, April 2010.

[17] ——, "Mantevo," http://software.sandia.gov/mantevo.

[18] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratory, Tech. Rep. SAND2009-5574, 2009.

[19] Lawrence Livermore National Laboratories, "SAMRAI," https://computation.llnl.gov/casc/SAMRAI/index.html.

[20] Los Alamos National Laboratories, "Sweep3d," http://www.c3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html, 1999.

[21] A. Tuininga, "cx_bsdiff," http://starship.python.net/crew/atuining/cx_bsdiff/index.html, February 2006.

[22] K. Ferreira, R. Riesen, R. Brighwell, P. Bridges, and D. Arnold, "libhashckpt: hash-based incremental checkpointing using GPUs," *Recent Advances in the Message Passing Interface*, pp. 272–281, 2011.

[23] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum, "Disco: running commodity operating systems on scalable multiprocessors," *ACM Trans. Comput. Syst.*, vol. 15, no. 4, pp. 412–447, Nov. 1997. [Online]. Available: http://doi.acm.org/10.1145/265924.265930

[24] C. A. Waldspurger, "Memory resource management in VMware ESX server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844146

[25] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference Engine: Harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1831407.1831429

[26] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 18:1–18:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364813.1364831

[27] C. Ding and K. Kennedy, "Improving effective bandwidth through compiler enhancement of global cache reuse," *J. Parallel Distrib. Comput.*, vol. 64, no. 1, pp. 108–134, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2003.09.005

[28] J. Willcock and A. Lumsdaine, "Accelerating sparse matrix computations via data compression," in *Proceedings of the 20th annual international conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 307–316. [Online]. Available: http://doi.acm.org/10.1145/1183401.1183444

[29] S. Biswas, D. Franklin, A. Savage, R. Dixon, T. Sherwood, and F. Chong, "Multi-execution: multicore caching for data-similar executions," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 164–173.

[30] S. Barker, T. Wood, P. Shenoy, and R. Sitaraman, "An empirical study of memory sharing in virtual machines," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference*. USENIX Association, 2012, pp. 25–25.