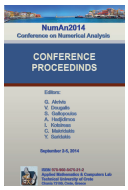
OpenAccess

Proceedings of the 6th International Conference on Numerical Analysis, pp 97-102

Contents lists available at AMCL's Digital Library

NumAn2014 Conference Proceedings

Digital Library Triton : http://lib.amcl.tuc.gr



Exploring the Performance of Out-of-Core Linear Algebra Algorithms in Flash based Storage

Athanasios Fevgas, Panagiota Tsompanopoulou, and Panayiotis Bozanis

Department of Electrical and Computer Engineering
University of Thessaly
Volos, Greece
{fevgas,yota,pbozanis}@inf.uth.gr

Abstract. In the recent years, flash memory has been widely utilized as storage medium to mobile and embedded systems, laptops and servers. However, due to its idiosyncrasies (asymmetric read/write speeds, erase before update, wear out) it introduces new challenges for researchers. Many studies, especially in the field of databases, investigate for algorithms and data structures optimized for flash. The outstanding efficiency of this new storage medium, motivated us to study the performance of out-of-core, sparse, linear system solvers in flash SSDs. Two solvers PARDISO and MUMPS with out-of-core functionality were evaluated both on SSD and HDD. Experimental results show that PARDISO execution time is improved by a factor of two to three in most of the test cases. Performance gain for MUMPS exists but it is less.

1 Introduction

Flash memory is a non-volatile electronic storage that can be electrically erased and reprogrammed. There are two types of flash, NOR and NAND with the later utilized as mass storage medium. In the rest of this document the term flash denotes the NAND flash. Storages based on flash lack mechanical and moving parts, providing low power consumption, shock resistance and high read/write performance. Flash consists of cells which store one (SLC) or more bits (MLC). Cells are organized to pages and pages to blocks, with typical page sizes to vary from 512 bytes to 16Kb and block sizes from 32Kb to 512Kb. Reads and writes are performed at page level, while erases at block level. Write operations are slower than reads and erases are even slower. Moreover, pages have to be erased before are re-written and flash endurance is limited by a finite number of write/erase cycles (wear out).

Solid state drives (SSDs) are block devices compatible with traditional hard disk drives (HDDs) relying in flash. The main components of an SSD are the flash memory chips and a controller which emulates the block interface using FTL (Flash Translation Layer). FTL remaps logical addresses, used by the upper layers, to physical addresses in flash chips. It incorporates out-of-place-updates, wear leveling and garbage collection mechanisms aiming to improve write performance and prevent wear-out. Specifically, out-of-place-update mechanism redirects update requests to clean (already erased) pages invalidating the old ones. Wear leveling distributes writes across medium

ISBN: 978-960-8475-22-9 *http://lib.amcl.tuc.gr/handle/triton/36*

to prevent wear out of certain cells and garbage collection algorithms take care of reclaiming invalidate pages.

All the above make clear that data structures and algorithms designed for hard disks do not perform well in flash. Thus, recent studies in databases endeavor to design flash efficient indexes and buffer managers, reducing expensive write operations and exploiting fast random reads.

In several cases log-based data structures are utilized for updating indexes instead of performing expensive page rewrites. The delta records are accumulated to a main memory buffer and persisted to flash in batches when the buffer is full in [1]. A node translation table is maintained, for mapping the tree-nodes with the flash pages that store their log records. All delta records for a particular tree-node, are aggregated to continuous flash pages if a spread limit is reached. In LCR-tree [2] all the deltas for an R-tree node are stored to the same flash page assuring one additional read for each node access. Other researchers propose lazy updating of B+tree nodes [3] using flash resident cascading buffers that store updates for a node and its children. Updates are propagated in batch to the buffers of the next level sub-tree. This approach is more memory efficient compared to the previous ones. In the same context, an overflow buffer technique for R-tree is proposed [4] deferring spit operations in the future in order to reduce random writes. Researchers in [5] experimentally found that read and write operations in flash may benefit by SSD internal parallelism. Thus, they proposed an efficient method to perform parallel reads and writes using asynchronous I/O and applied it for B+tree optimization.

Traditional buffer replacement algorithms focus on maximizing hit ratio, but do not pay any attention for reducing writes, since reads and writes cost the same in HDDs. Therefore, new algorithms have been proposed that try to reduce writes in flash resident databases. The eviction of clean pages from a window of the w least recently used pages in the buffer have been introduced in [6]. An improved algorithm which takes into account not only the recency but and the frequency of references as well, was proposed in [7].

The development of efficient external memory algorithms for solving linear equations systems or calculating eigenvalues of large matrices has been a popular research topic. Several algorithms have been proposed pursuing to accelerate calculations by efficiently partitioning and managing large disk-resident datasets into main memory blocks. TAUCS [8] is a representative example of a C library of sparse linear solvers for external memory. More recent studies [9, 10] deal, among others, with matrix partitioning, prefetching and overlapping I/O and computations. Alternative in-core (IC) approaches require clusters with distributed memory, large enough for the entire dataset, and high bandwidth interconnections.

Nowadays, the emergence of multicore and GPU accelerated computers provides high processing power at low cost. On the other hand, non-volatile memories are capable to accelerate the storage layer. The outstanding performance of enterprise-level flash-based storage systems motivated authors of [11, 12] to address the problems of sparse matrix vector multiplication (SpMV) and large scale eigenvalues calculation using an SSD equipped cluster. A high level abstraction and a middleware for distributed out-of-core (OOC) computations were introduced providing a simple application in-

terface for linear algebra operations. Several experiments were conducted in order to evaluate the performance of out-of-core implementations.

The performance of three different out-of-core solvers (MUMPS, Intel MKL PARDISO, and HSL_MA78) was studied in [13]. The solvers were evaluated for the solution of three dimensional Navier-Stokes finite elements formulations. In-core and out-of-core CPU time and memory requirements were measured for each solver and PARDISO was classified as the best.

Considering the benefits of flash based storages, we study the performance of out-of-core versions of Intel MKL PARDISO (Parallel Direct Sparse Solver) and MUMPS (MUltifrontal Massively Parallel sparse direct Solver) in solid state disks.

2 Out -of-Core Sparse Solvers

PARDISO [14, 15] is a shared memory multiprocessing parallel direct solver for large sparse linear systems of equations. It supports real, complex, symmetric, structurally symmetric, unsymmetric, positive definite, indefinite and Hermitian systems. PARDISO conducts the solution of a linear system in three phases: a) analysis and symbolic factorization, b) numerical factorization and c) forward and backward substitution including iterative refinement. The Intel MKL version of PARDISO supports out-or-core functionality, utilizing external memory to retain matrix factors. PARDISO might be called either by FORTRAN or C/C++ programs.

MUMPS [16, 17] is a parallel direct solver for sparse linear equations and like PARDISO it provides out of core functionality. It supports unsymmetric, symmetric positive definite, or general symmetric systems. The solving procedure involves three steps: a) analysis, b) factorization and c) solution. MUMPS relies on MPI for parallelization, a host processor distributes the matrix and aggregates the results. The host processor, performs the most tasks of the analysis stage. MUMPS is developed in FORTRAN but provides C, Matlab, Octave and SciLab programming interfaces as well.

3 Experiments

In this paper we aim to evaluate the OOC functionality of two state of the art sparse direct solvers, Intel MKL PARDISO and MUMPS. Our evaluation focus on the performance of each solver in terms of memory usage and execution time for HDD and SSD drives. The experiments were performed on two DELL Precision T3500 workstations equipped with 8GB of DDR3 RAM and a 4-core Intel Xeon W3550 3.06GHz CPU each. The first workstation is configured with an Intel 520 SSD 240GB (connected to SATA-III interface) as booting device (INTEL in Tables 2, 3) and an OCZ Revodrive 350 PCIe 480GB as additional storage device (OCZ in Tables 2, 3). The Intel 520 drive is able to deliver 550 MB/s in sequential reads operations and 540 MB/s in sequential writes. The OCZ SSD succeeds up to 1800MB/s and 1700MB/s in sequential reads and writes, respectively. The second workstation utilizes a Seagate Barracuda 7200rpm 1TB hard drive with max supported data rate up to 210MB/s (HDD in Tables 2, 3). Both workstations run Centos 6.5 with kernel 2.6.32-431.el6.x86_64 operating system and are configured with 8GB of swap space. Intel MKL version 11.1.2 and Intel ICC

Table 1. Test Matrices

Matrix	Type	Equations	Non-zeros	Description
INLINE_1	R, S, PD	503,712	18,660,027	Parasol project
ASTER_PERF002C	R, S, I	1,008,012	37,926,024	Structural engineering
ASTER_PERF011A	R, S, I	853,632	71,098,992	Structural engineering
ATMOSMODL	R, U, I	1,489,752	10,319,760	Atmospheric modeling
AUDIKW_1	R, S, I	943,695	39,297,771	Automotive model
NICE20MC	R, S, I	715,923	28,066,527	Earthquake dynamic analysis

14.0.2 compiler were used for PARDISO and GCC 4.4.7 compiler and OpenMPI (4 parallel processes) were utilized for MUMPS experiments. PARDISO uses METIS [18] algorithm for reordering while MUMPS utilizes PORD [19].

Test data were derived from Sparse Matrix Collection[1] of the University of Florida and GRID-TLSE[2] project website. Real, symmetric and unsymmetric linear systems of equations were examined. A detailed description of the test sets is given in Table 1. All tests were repeated for three times and the average values are presented in the following. The wall-clock time of the out-of-core runs for each of the two SSDs and the HDD is given, along with in-core times for comparison purposes. The in-core experiments were executed on the SSD enabled workstation. Memory requirements for both out-of-core and in-core executions were also recorded.

Table 2. Performance of PARDISO

MATRIX	**OOC PARDISO**					**IC PARDISO**	
	Wall-clock time (sec)			**Memory (GB)**		**Wall-clock time (sec)**	**Memory (GB)**
	OCZ	**INTEL**	**HDD**	**IC**	**OOC**		
INLINE_1	39.33	38.33	43.33	0.67	1.45	22	1.80
ASTER_PERF002C	160	221.33	485	1.38	7	179.67	7.28
ASTER_PERF011A	582.33	757	1,620.67	2.31	14.9	-	14.72
ATMOSMODL	654.50	796	1,379.67	1.21	15	-	15.65
AUDIKW_1	367.33	441	758	1.51	10.05	817	10.57
NICE20MC	274.00	327	549	1.08	8.59	553.67	9.06

Out-of-core PARDISO solver runs 2 to 3 times faster to the SSD enabled workstation, for the five out of six datasets. The gain is less for the INLINE_1 dataset, probably because the amount of I/O is less, thus have no significant affect to the overall execution time of the OOC algorithm. The IC execution is faster than OOC for small data sets, but when memory requirements increase and swapping is involved then IC performs worse.

[1] http://www.cise.ufl.edu/research/sparse/matrices
[2] http://tlse.enseeiht.fr

Table 3. Performance of MUMPS

MATRIX	OOC MUMPS					IC MUMPS	
	Wall-clock time (sec)			Memory (GB)		Wall-clock time (sec)	Memory (GB)
	OCZ	INTEL	HDD	IC	OOC		
INLINE_1	42.7	42.3	43.3	0.6	1.4	42.67	2.8
ASTER_PERF002C	485.7	503.7	556.3	2.1	6.9	583	13.6
ASTER_PERF011A	3,906	3,961.5	4,428	9	14.8	-	37.7
ATMOSMODL	2,349.3	2,384	2,480.5	5.3	13.9	-	27.8
AUDIKW_1	1,362	1,392.7	1,495.3	4.7	9.4	-	19.3
NICE20MC	1,388.7	1,561	1,629	4.6	9.1	-	20.68

Table 3 shows the performance of the MUMPS solver. A performance gain for the SSDs still exists, but in this case is much less. Moreover, the overall performance of MUMPS is lower compared to PARDISO and main memory requirements are quite longer for both OOC and IC execution. It is noticeable that four of the linear systems is not possible to be solved in-core.

4 Conclusions

Flash is a new type of non-volatile memory with outstanding efficiency compared to hard drives. However, its specific characteristics make algorithms designed for rotational disks not perform well. We tried to describe the characteristics of this new storage medium and highlight the possible benefits of use in numerical linear algebra. Therefore, we evaluated the performance of OOC versions of Intel MKL PARDISO and MUMPS sparse, direct solvers. Experiments show that PARDISO benefits significantly from SSD storage, while, performance gain in MUMPS is less but exists. Experiments unveil that out-of-core execution outperforms in-core when the swapping is used.

Relying to the above we argue that flash enabled storages are able to lead to important savings of computational time in numerical linear algebra, but more research is needed in order to exploit their full advantages. Our future work aims to develop flash efficient algorithms for out of core execution for linear algebra methods.

Acknowledgement

The present research work has been co- nanced by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS . Investing in knownledge society through the European Social Fund (MIS 379416).

References

1. Wu, C.H., Kuo, T.W., Chang, L.P.: An efficient b-tree layer implementation for flash-memory storage systems. ACM Transactions on Embedded Computing Systems (TECS) **6** (2007) 19
2. Lv, Y., Li, J., Cui, B., Chen, X.: Log-compact r-tree: an efficient spatial index for ssd. In: Database Systems for Adanced Applications. Springer (2011) 202–213
3. Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y., Singh, S.: Lazy-adaptive tree: An optimized index structure for flash devices. Proceedings of the VLDB Endowment **2** (2009) 361–372
4. Wang, N., Jin, P., Wan, S., Zhang, Y., Yue, L.: Or-tree: An optimized spatial tree index for flash-memory storage systems. In: Data and Knowledge Engineering. Springer (2012) 1–14
5. Roh, H., Park, S., Kim, S., Shin, M., Lee, S.W.: B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives. Proceedings of the VLDB Endowment **5** (2011) 286–297
6. Park, S.y., Jung, D., Kang, J.u., Kim, J.s., Lee, J.: Cflru: a replacement algorithm for flash memory. In: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, ACM (2006) 234–241
7. Jin, P., Ou, Y., Harder, T., Li, Z.: Ad-lru: An efficient buffer replacement algorithm for flash-based databases. Data & Knowledge Engineering **72** (2012) 83–102
8. Toledo, S., Chen, D., Rotkin, V.: Taucs: A library of sparse linear solvers (2003)
9. Castellanos, J., Larrazabal, G.: A cholesky out-of-core factorization. Mathematical and Computer Modelling **57** (2013) 2207–2222
10. IGUAL, F., MARQUES, M., VAN DE GEIJN, R.A.: (A run-time system for programming out-of-core matrix algorithms-by-tiles on multithreaded architectures)
11. Zhou, Z., Saule, E., Aktulga, H.M., Yang, C., Ng, E.G., Maris, P., Vary, J.P., Catalyurek, U.V.: An out-of-core dataflow middleware to reduce the cost of large scale iterative solvers. In: Parallel Processing Workshops (ICPPW), 2012 41st International Conference on, IEEE (2012) 71–80
12. Zhou, Z., Saule, E., Aktulga, H.M., Yang, C., Ng, E.G., Maris, P., Vary, J.P., Catalyurek, U.V.: An out-of-core eigensolver on ssd-equipped clusters. In: Cluster Computing (CLUSTER), 2012 IEEE International Conference on, IEEE (2012) 248–256
13. Raju, M., Khaitan, S.: High performance computing using out-of-core sparse direct solvers. International Journal of Mathematical, Physical and Engineering Sciences **3** (2009) 377–383
14. Schenk, O.: Scalable parallel sparse lu factorization methods on shared memory multiprocessors. (2000)
15. Schenk, O., Gartner, K.: Solving unsymmetric sparse systems of linear equations with pardiso. Future Generation Computer Systems **20** (2004) 475–487
16. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications **23** (2001) 15–41
17. Amestoy, P.R., Guermouche, A., L Excellent, J.Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Computing **32** (2006) 136–156
18. Karypis, G., Kumar, V.: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN (1998)
19. Schulze, J.: Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. BIT Numerical Mathematics **41** (2001) 800–841