A Highly Efficient I/O-based Out-of-Core Stencil Algorithm with Globally Optimized Temporal Blocking

Hiroko Midorikawa, Hideyuki Tan
Department of Computer and Information Science
Seikei University, JST CREST
Tokyo, Japan
midori@st.seikei.ac.jp

Abstract— **This paper proposes the most efficient I/O-based out-of-core stencil algorithm for large-capacity type of non-volatile memory (NVM), such as flash. The paper evaluates the performances of various out-of-core stencil algorithms and implementations designed for flash. The algorithms for flash are very different from existing algorithms designed for memory-and-cache, host-and-GPU, and local-and-remote nodes, in their schemes, data structures used in stencil computations, and the way of using blocking technique to increase data access locality for accelerating performance. The proposed algorithm achieves 80% of the performance of in-core computing using sufficient capacity of the main memory, even if available memory capacity is limited to 6.3% of the data size required in the stencil computation problem. In other words, the algorithm degrades performance within 20% for the stencil computation problem that requires 2TiB of data by using only 128GiB of main memory and flash SSDs whose access latency is much larger than that of DRAM.**

Keywords—Non-volatile memory; flash memory; temporal blocking; stencil; algorithm; out-of-core; asynchronous I/O; access locality; auto-tuning;

I. Introduction

Stencil computation is one of the most important computation kernels in various scientific and engineering simulations. Stencil computations often require significant amounts of memory for addressing large-scale problems and/or for higher resolution data analysis. However, there is a limit to how much DRAM can be increased in main memory because of the number of memory slots on server boards, power consumption constraints, and other resource limitations. One of the common solutions used to satisfy this requirement is using cost-effective and large-capacity non-volatile memory (NVM).

Today, various types of new memory devices including non-volatile memories, such as ReRAM, MRAM, PCM, 3D-Xpoint and Z-NAND [5], are being actively investigated. Nevertheless, NAND-flash memory is the most ubiquitous to end users at present and it is still determined as a cost-effective, power-efficient, and large-capacity type of memory behind the DRAM layer in the memory hierarchy. The three-dimensional (3D)-structured NAND-flash has been improving in performance and capacity, but its access latency is still much longer, by a factor of 1000, than DRAM access latency. The latency gap between DRAM and flash is much larger than that between cache and DRAM.

We have been investigating out-of-core stencil computation algorithms designed for large-capacity type of NVMs, such as NAND flash [1-4]. This paper proposes the most efficient I/O-based out-of-core stencil algorithm and implementation as well as the evaluation of its performance. The algorithm is available for not only one-node server but also SSD-equipped clusters for local-node calculations [4]. The algorithm for flash is very different from existing algorithms designed for memory-and-cache, host-and-GPU, and local-and-remote nodes, in their implementations, data layout used in stencil computations, and the way of using blocking technique to increase data access locality for accelerating performance. The algorithm uses highly parallel asynchronous I/O (AIO) and a temporal blocking technique without redundant calculations to overcome the latency divide between flash and DRAM. It achieves 80% of the performance of in-core computing using sufficient capacity of the main memory, even if available memory capacity is limited to 6.3% of the data size required in the stencil computation problem. In other words, the algorithm degrades performance within 20% for a stencil problem that requires 2 TiB of data by using only 128 GiB of main memory and flash SSDs.

According to our earlier studies [2], I/O-based algorithms gains not only higher performance but also more stable behavior, compared to mmap-based and swap-based algorithms with memory-semantic access. The processes using mmap large capacity of memory are often killed unexpectedly by the operating system (OS), i.e. the out of memory (OOM) killer, when the remaining available memory size becomes lower than the predefined threshold by the OS kernel parameter. It is difficult to control and prevent OOM killer for individual platform and problem. Moreover, the performance of mmap-based algorithms depends on the amount of available unused main memory of DRAM at runtime, because the remaining main memory is used by the OS as page cache area for memory-mapped file accessing. When the size of the remaining unused memory is limited at runtime, mmap-based algorithms exhibit lower performance.

Another advantage of I/O-based algorithms is that using explicit I/O provides benefits in calculating globally optimal blocking sizes in the temporal blocking technique. To achieve the maximum performance, the algorithms use appropriate spatial and temporal blocking parameters that minimizes the amount of data transferred between the flash device and the DRAM, which is a dominant factor affecting the performance of out-of-core computing. By a just-in-time auto-tuning system, such as Blk-Tune [3], the I/O-based algorithm allows users to

978-1-5386-0700-8/17/$31.00 ©2017 IEEE

maximize the performance easily for particular platforms and application settings.

The algorithms proposed here are not limited to flash but are also available to other I/O-based NVMs and read/write-based memories. With the use of the algorithm, large-scale stencil problems can be solved with a limited size of main memory.

II. TEMPORAL BLOCKING STENCIL ALGORITHMS

Stencil computation is considered a memory-bound type of computation, and it has been studied in numerous works to accelerate its execution. One typical technique is increasing data access locality, which typically causes higher cache hit, by introducing blocking techniques in spatial and temporal spaces in stencil computations [6–8].

A. Algorithms With/Without Redundant Calculations

Temporal blocking algorithms are categorized into two types: one with redundant calculations, AL-R in Fig. 1, and the other without them, AL-NR in Fig. 2(a). Domain data buffer arrays are divided into sub-blocks, $bx \times by$, and each block is updated in bt steps locally and written back to the destination buffer array. In this sutdy, bt is the temporal blocking size and (bx, by, bz) for a 3D data domain is the spatial blocking size. h is a size parameter in the stencil computation kernel, e.g., $h = 1$ for a typical seven-point stencil for a 3D data domain.

In AL-R, one block is read from a domain source (src) *buffer array* and an updated destination (dst) *block* is written back to a dst *buffer array*. The calculation area in the *block arrays* shrinks according to the current update step progress, as shown in Fig. 1. The final result area updated bt times is smaller than the area initially read from the src buffer array. This difference in area corresponds to the amount of redundant calculations.

In AL-NR, the calculation areas in the block arrays are shifted according to the current update step progress, as shown in Fig. 2(a). Unlike AL-R, AL-NR reads two block arrays from both the src and dst *buffer arrays*, and updated results in both the src and dst *block arrays* are written back to the *buffer arrays* after bt-step updates. This is necessary because each block array still includes an uncompleted calculation area, which is used and

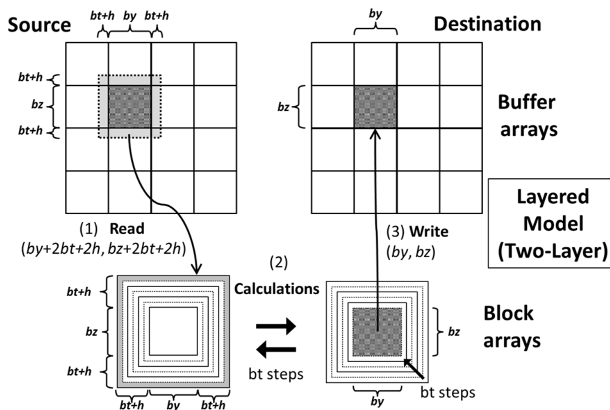

Fig. 1 Calculation of *bt* steps in block arrays and read/write from/to domain arrays in the AL-R algorithm.

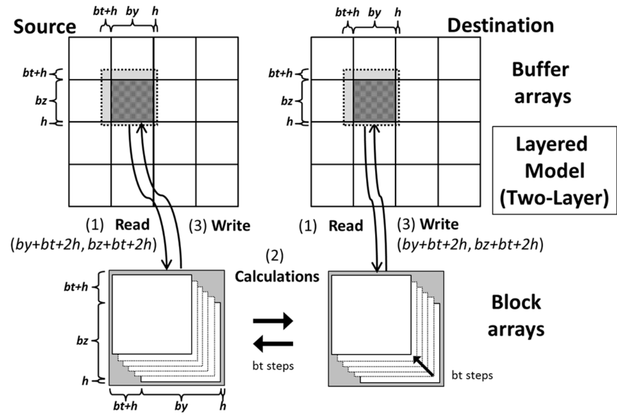

Fig. 2 (a) Calculation of *bt* steps in block arrays and read/write from/to domain buffer arrays in the AL-NR algorithm.

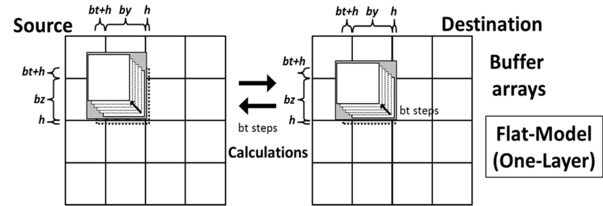

Fig. 2 (b) Flat-model available to the AL-NR algorithm. The *bt*-step update calculations are performed directly on buffer arrays.

completed in the next neighbor block update procedures. This is an essential mechanism to eliminate the redundant calculations in AL-NR. Thus, AL-NR decreases the amount of calculations compared to AL-R, but it increases the amount of read/write data from/to buffer arrays. This becomes an important factor in the performance of out-of-core algorithms using flash for storing buffer arrays. Read/write of buffer arrays corresponds to flash I/O. The performance impact is very different from that in existing memory-based algorithms. The most of the existing theoretical studies of stencil computations focus on memory systems, whose access latency is much smaller than that of the flash SSD.

B. Data Structures : Flat Model and Layered Model

AL-R is required to use two block arrays in addition to the two domain data buffer arrays, because it updates a larger calculation area than the final result-block area. Thus AL-R uses a layered data model shown in Fig. 1. On the other hand, AL-NR can use a flat-model using only buffer arrays in Fig 2(b), as well as the layered model in Fig. 2 (a). The selection of these models are more important in the flash-based out-of-core algorithms than the algorithms designed for memory systems.

III. THE FLASH-BASED OUT-OF-CORE ALGORITHMS

The out-of-core algorithms evaluated here employ a hierarchical blocking scheme corresponding to a memory hierarchy as shown in Fig. 3(a). Flash in the bottom stores

problem domain data as buffer arrays. Main memory in the middle contains block arrays for temporal blocking. Cache in the top includes iblock arrays for spatial blocking for multi-thread computations in the z-dimension as shown in Fig. 3(b).

A. *Three Methods to Use Flash SSDs*

Temporal blocking was first applied to a flash and main memory tier for an out-of-core algorithm in our earlier work [1], where flash was used as a swap device under the revised swap kernel fast-swap (OpenNVM) [9] in Linux. Moreover, we evaluated three methods, namely, 1) swap device, 2) file mmap, and 3) asynchronous I/O (AIO), that use a flash device for out-of-core stencil computations [2].

In the swap method (1), buffer and block arrays are allocated by the malloc() in applications. The block arrays are locked onto main memory by the mlock() to prevent them being swapped out. This method is transparent to application programs. However, the swap daemon was originally designed to rescue a process in the emergent situation with lack of memory. Thus, the performance of program execution is very low and unstable. It is not an adequate method for ordinary out-of-core computations.

In the mmap method (2), the block arrays are allocated by malloc() and the buffer arrays are represented as files that are memory mapped by the mmap(). In this method, normal in-core programs are available to out-of-core computing with a little modification. The performance is better than that in the swap method, but it depends on available unused memory capacity for page cache. It is also difficult to prevent the OOM killer, as mentioned in Section I.

In the aio method, Linux kernel asynchronous input/output (AIO) library functions, such as io_submit() and io_getevents(), are used. The block arrays are allocated by malloc() and the buffer arrays are represented as consecutive blocks on a flash block device. A flash SSD is opened with O_DIRECT, which eliminates file-system-layer overhead and kernel-managed page cache. The recent improvement of block storage stacks in Linux [10], specifically, multiple I/O request queues for multi-core, provides higher performance to AIO. It makes it possible to issue 64K or more I/O operations simultaneously by multiple threads in an asynchronous fashion. The aio method gains higher and more stable performance compared to other two methods.

B. *Block Array Memory Layout for the AIO method*

When using the aio method, AIO requires block-size-aligned data access for block devices. Thus, block arrays in Fig. 3a are implemented with a one-dimensional pointer array for the z-dimension and multiple xy-planes pointed to by the pointers in the array for the z-dimension, as shown in Fig. 3(c). The start address and the size of each xy-plane are aligned on the device block size boundary of the flash SSD, which was 4KB in our case. Each xy-plane is a single I/O unit of AIOs by multiple threads in parallel. Highly-parallel, large-size, and block-aligned I/O is a key to boost the performance of flash-based algorithms.

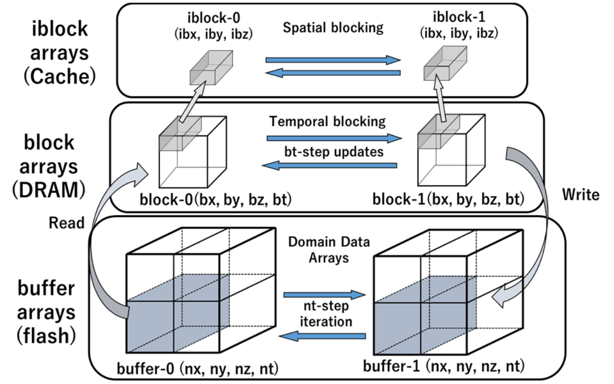

Fig. 3 (a) Three data arrays in memory layers*: buffer arrays* in flash SSD, *block arrays* in DRAM, and *iblock arrays* in cache for locality extraction.

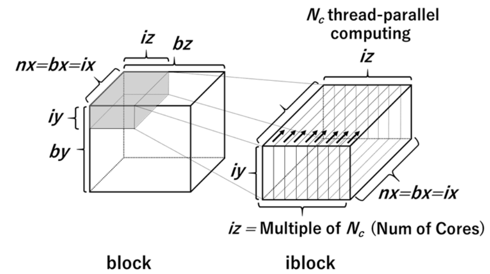

Fig. 3 (b) A *block array* is divided into *iblock arrays*. The *iblock arrays* in cache are updated by multiple threads in parallel for the z-dimension.

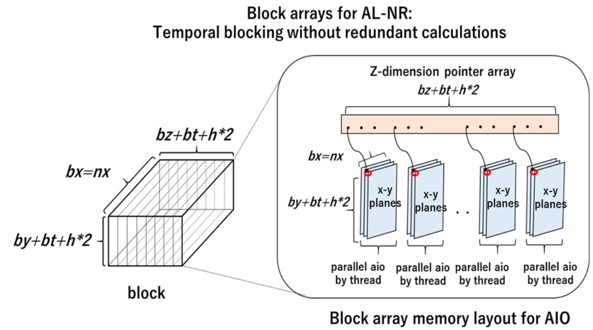

Fig. 3 (c) A *block array* memory layout for AIO.

IV. The Evaluation of Flash-Based Stencil Algorithms

Fig. 4 shows the comparison of eleven algorithms with different methods, swap, mmap, and aio, and different data models, flat (1-layer) and layered (2-layer) models, in relative execution times of 3D-stencil computations (seven-point stencil, problem size: 64GiB). It also shows the difference in times between the case using sufficient 128GiB-memory (in-core computing) and the case using insufficient 32GiB-memory (out-of-core computing). The baseline is the fastest in-core computing case, mmap1 (1-layer model with optimal layout) with AL-NR in column (col.) 6b.

The leftmost case, noTB with the swap method in col. 1, shows the time when using only spatial blocking. Without temporal blocking, out-of-core computing time grows 66.29 in contrast to the in-core time that grows only 1.02 in col. 1b. In cols. 2 to 4, times using temporal blocking with redundant calculations, AL-R, are shown for three methods. In AL-R, aio in col. 4 is the fastest for out-of-core computing. For in-core computing, AL-R in col. 2b is not efficient compared to the one without temporal blocking in col. 1b. The temporal blocking size used here, bt (=128), is inadequate to the DRAM-cache tier for in-core computing, but adequate to the DRAM-flash tier for out-

of-core computing. In AL-R, larger *bt* size accelerates performance by increasing temporal access locality and degrades performance by increasing redundant calculations. Appropriate bt size depends on this tradeoff. Generally, optimal bt size for a DRAM-cache tier is smaller than that for a DRAM-Flash tier.

In cols. 5 to 11, AL-NR, temporal blocking with no redundant calculations, is used with the mmap and aio methods. The *mmap1* algorithm in cols. 5 and 5b corresponds to the original version AL-NR designed only for main memory and cache without flash devices. The *mmap1+opt* in col. 6 is an optimized version of *mmap1* by introducing element padding to block arrays, a typical technique to reduce cache line and TLB entry conflicts by multiple threads. Both algorithms employ a flat model (1-layer) in Fig. 2 (b) because their original design is based on cache-DRAM maintenance being implicitly performed by hardware. When employing the *mmap1* algorithm for a flash-based out-of-core computation with the mmap method, its DRAM-flash maintenance is implicitly carried out as page cache maintenance by the OS. For in-core computing, the 1-layer-opt model with AL-NR in cols. 6b gains the highest performance. In contrast, for out-of-core computing using flash, the 1-layer AL-NR in cols. 5 and 6 exhibit worse performance compared to that of the 2-layer AL-R in col. 3. The *mmap2* in col. 7, a modified version of *mmap1-opt* that introduces an intermediate layer, improves its performance considerably. Thus, the 2-layer model is more effective for out-of-core computing using flash, and the 1-layer model is only effective for in-core computing. This is attributed to the implicit page caching in mmap that causes inefficient data transfer to the flash device, as a result of using only buffer arrays, without an intermediate block array layer.

Algorithms in cols. 8 to 11 are advanced versions of AL-NR with the 2-layer model, which are optimized to reduce the amount of read/write traffic between buffer arrays and block arrays as shown in Fig. 2(a). The first group are *mmap3y* and *aio3y* in cols. 8 and 10, and the second group are *mmap5y* and *aio5y* in cols. 9 and 11. Details of these algorithms are described in the next section. According to the evaluation of 11 algorithms in Fig. 5, the most efficient out-of-core algorithm is aio5y in col. 11. Its performance degradation compared to the best of in-core algorithm, *mmap1-opt*, is only 24%, when computing the 64GiB-size problem by using only a half-size of 32GiB-DRAM.

In contrast to AL-R, AL-NR is free from the redundant calculation overhead, and there is only the tradeoff in how the spatial and temporal blocking sizes share the fixed DRAM capacity. The globally optimal blocking size combination of spatial and temporal sizes can be determined by the search technique used in Blk-Tune. It is designed for reducing the the amount of read/write traffic between buffer arrays and block arrays, or in this case, the amount of I/O traffic between DRAM and flash.

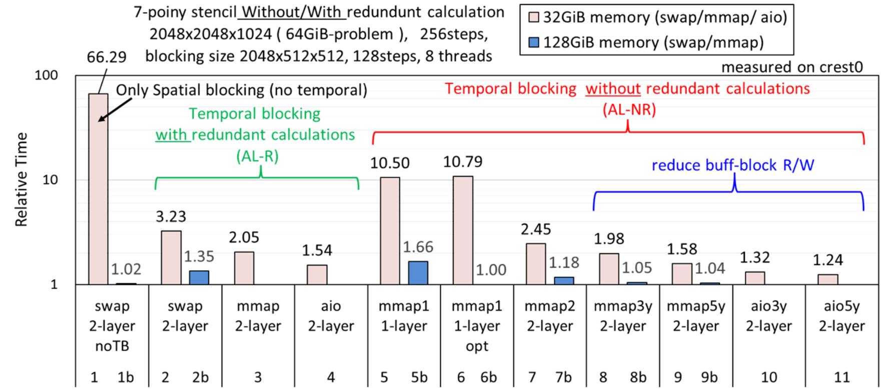

Fig. 4 Comparison of out-of-core algorithms. Relative execution times of seven-point stencil computation

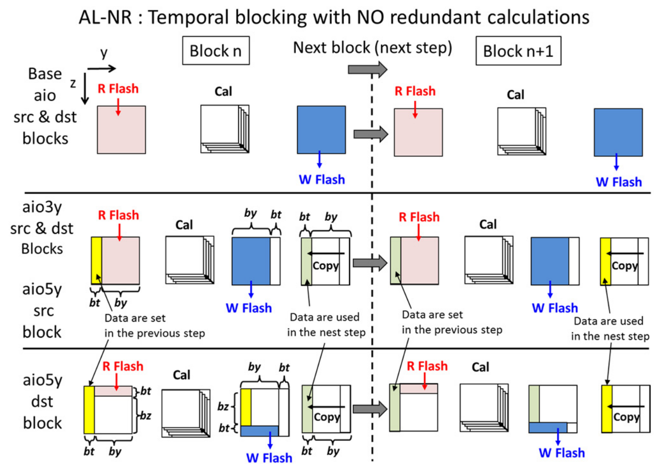

Fig. 5 AL-NR: read/write patterns read/write between buffer arrays and block arrays in three algorithms, aio, aio3y, aio5y

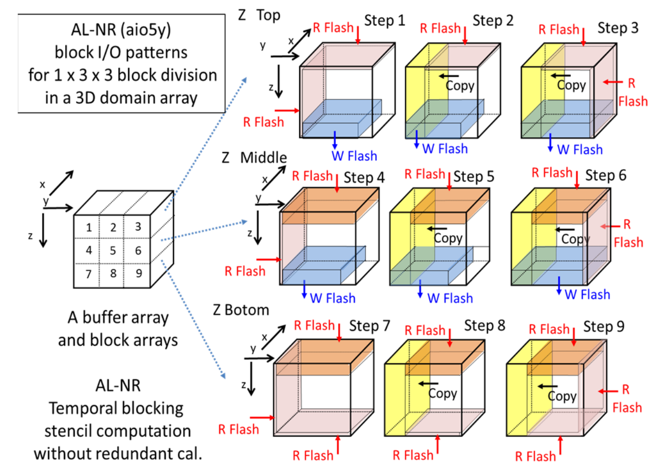

Fig. 6 Examples of read/write patterns aio5y for 3D block arrays

V. I/O-Based Optimized Algorithms

In this section, the most efficient I/O-based AL-NR algorithms, *aio3y* and *aio5y*, are described in detail. Fig. 6 shows how one block calculations proceed from block n to block n+1 in the next position in the y-dimension in this case. It also shows read/write areas in block arrays in one-block updates for three algorithms, (1) aio in col. 4, (2) aio3y in col. 10, and (3) aio5y in col. 11 in Fig. 5. They have different read/write patterns.

In *aio* and *aio3y*, the top and the second row of Fig. 6, both src and dst block arrays have the same manner of read/write from/to the buffer arrays, src and dst. In the basic algorithm, aio, Step1: each block array reads data from each buffer array, Step2: update block n in *bt* steps using the src and dst blocks , Step3: each block writes its data back to each buffer array. To reduce the amount of read/write traffic, *aio3y* introduces an additional step, Step4: the rightmost part of a block array is copied in the leftmost part in the same block, for reuse by the next block update in block n+1 in the right hand part of Fig. 6. Typically, the amount of reduced read/write is double that of of the copied area, $(bt + h) \times (bt + bz + 2h)$.

In *aio5y*, the src block array has the same manner as *aio3y*, but the dst block array has a different read/write pattern as shown in the bottom of Fig. 6. In the first update step using block arrays, the most part of the dst block array is updated before read. Thus, it is possible to reduce the read areas from the dst buffer array to the dst block array. For the same reason, the write area in the dst block array to the buffer array can be reduced, because only a small area as shown in the bottom part of the dst array is used by the neighboring block in the z-dimension. Fig. 7 shows the read/write patterns of the dst block array in *aio5y* for 3D stencil computations. The read/write patterns are different for the position in the buffer array and block division scheme employed. The boundary conditions in the buffer arrays also cause different patterns. The central block in Fig. 6 is the typical case in 3D computation that corresponds to the 2D case in the bottom of Fig. 5.

VI. Performance Evaluation

In this section, the most effective I/O-based out-of-core stencil algorithm, aio5y and aio3y are evaluated for 3D-stencil problems ranging from 128 GiB to 2 TiB in data size by using 128GiB main memory in the platforms, crest6 and crest10 in Table I. Flash devices used here are the latest cost-effective M.2 NVMe flash, Samsung 950Pro (512GB) and 960Pro (2TB). They are configured as 1TB and 4TB devices for crest6 and crest10 individually by using software RAID0 to double their performance. Read/write bandwidths of various flash devices are shown in Fig. 7. They are measured by parallel AIO of actual sizes used in the 64GiB-stencil-problem in our algorithms.

In this experiment, the best combinations of spatial and temporal blocking sizes for each stencil problem and platform are determined by Blk-Tune [3], as shown in Table II for crest10.

Table I Platforms

server	L1 cache (KiB)	L2 cache (KiB)	L3 cache (MiB)	Phys Mem (GiB)	Flash Mem (TiB)	CPU Xeon E5, (GiHz)	Total cores	socket	cores/ socket
crest0	32	256	20	32	1.2	2650, (2)	8	1	8
crest4	32	256	20	64	0.785	2687W, (3.1)	16	2	8
crest6	32	256	25	128	1	2687W v3,(3.1)	20	2	10
cres10	32	256	30	128	4	2687W v4,(3.0)	24	2	12

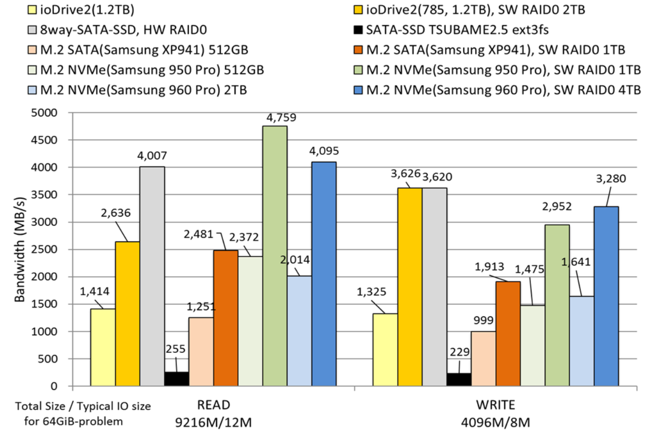

Fig. 7 Read/write performance of the data sizes used in the 64GiB-stencil-problem for various flash devices, (AIO, measured in crest4)

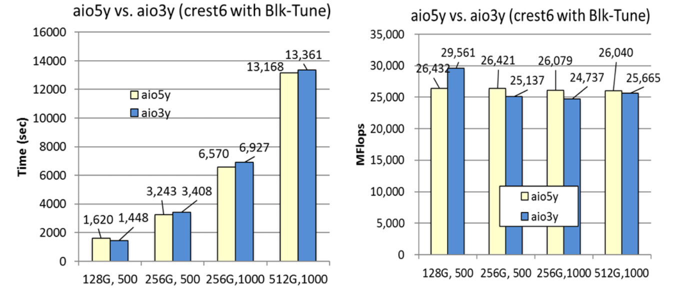

Fig. 8 Execution times and performances in aio3y and aio5y

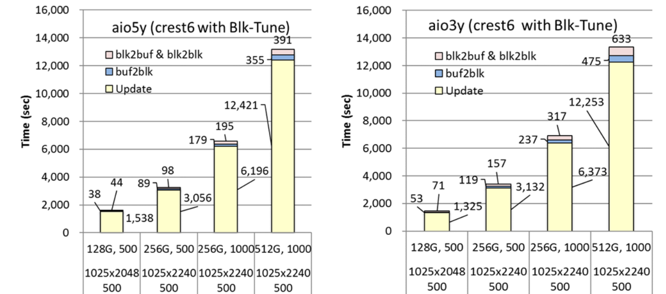

Fig. 9 Time components in aio3y and aio5y

Table II Temporal & spatial Block sizes for crest10 by Blk-Tune

problem size	512 GiB	1 TiB	2 TiB
nx, ny, nz	2048 x 4096 x 4096	4096 x 4096 x 4096	4096 x 4096 x 8192
nt	1000	1000	1000
bx, by, bz	2048 x 1025 x 2304	4096 x 683 x 1536	4096 x 683 x 1440
bt	500	334	334
ix, iy, iz	2048 x 1 x 192	4096 x 1 x 192	4096 x 1 x 192
block-division	1 x 4 x 2	1 x 6 x 3	1 x 6 x 6
Iteration	2	3	3
total num of blocks processed	16	54	108
max size of blocks	107.3 GiB	116.2 GiB	110.3 GiB

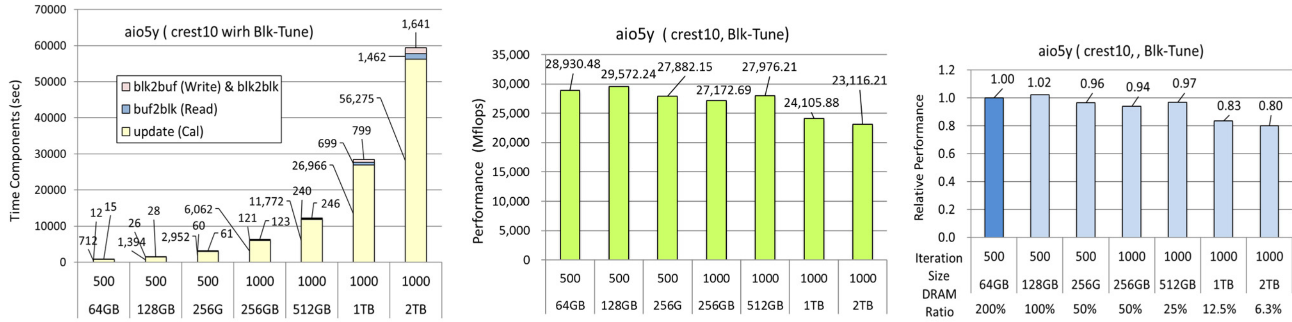

Fig. 10 Execution times, performances (Mflops) and relative performances of aio5y for large-size stencil problems

Blk-Tune retrieves platform hardware information and selects the best blocking parameters for a given problem and platform at runtime. It minimizes the amount of data transferred between buffer arrays (typically in the flash device) and block arrays (in the DRAM), which is a dominant factor affecting the performance of out-of-core algorithms using flash.

A. Comparison of aio3y and aio5y

The Fig. 8 shows the execution times and performance (Mflops) on crest6 for four problems (128 GiB, 500-step), (256 GiB ,500-step), (256 GiB, 1000-step) , and (512 GiB, 1000-step) using aio3y and aio5y. The aio5y algorithm gains better performance than aio3y, except in the case of the 128 GiB problem that exists on the border between in-core and is out-of-core computing. The overhead come from the algorithm complexity in aio5y becomes larger than that in aio3y for in-core computing. For the 512 GiB-problem, performance degradations in both algorithms are small. Fig.9 shows the time components of *aio3y* and *aio5y* for the same problems. The read (buf2blk) and write (blk2buf & blk2blk) time components are small in *aio5y* in compared to those in *aio3y*.

B. The Performance of aio5y for Large-Size Problems

By using 4-TiB flash, large-size stencil problems are evaluated by *aio5y*. Fig. 10 shows execution times, performances (Mflops), and relative performances for seven problems of 64 GiB – 2 TiB sizes by *aio5y* using 128-GiB main memory in crest10. All *aio5y* executions including the 2-TiB problem using 128-GiB memory are stable, unlike mmap5y, which is difficult to use for the execution of large size problems.

According to the relative performance in Fig. 11, the performance in the 512-GiB problem, where the DRAM ratio (the main memory size divided by the problem size) is 25%, achives 97% of the performance gained by the in-core computing in 64GiB-problem. Even in the 2-TiB problem case, where the DRAM ratio is only 6.25%, the performance maintains 80% of that in-core computing. The *aio5y* realizes sufficiently high performances.

VII. CONCLUSIONS

This paper proposes the most efficient I/O-based out-of-core stencil algorithm for large-capacity type of NVM, such as flash. The paper also evaluates the advantages and disadvantages of various out-of-core stencil algorithms and implementations designed for flash, by using state-of-the-art flash devices. We believe it is the first review of flash-based out-of-core stencil computations. Among these, the most efficient I/O-based out-of-core stencil algorithms, *aio3y* and *aio5y,* are described in detail and their performance is evaluated. They achieve 97% and 80% of in-core computing performance with sufficient memory, for out-of-core computing of 512-GiB and 2-TiB problems with only 128 GiB-DRAM and a cost-effective flash NVM. The I/O-based algorithms show not only high performance but also highly stable behavior. Moreover, they can use globally optimal blocking sizes which can be calculated precisely and achieve the best performance for a given platform and problem. With the use of the algorithms, large-scale stencil problems can be solved with limited main memory size, such as that in a personal work-station.

REFERENCES

[1] H. Midorikawa, H. Tan, and T. Endo, "An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations," Proc. 2014 Int. Conf. on High Performance Computing and Simulation IEEE-HPCS2014, Jul. 2014, pp. 268-277.

[2] H. Midorikawa and H. Tan, "Locality-Aware Stencil Computations Using Flash SSDs as Main Memory Extension," Proc. IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing CCGrid2015, May 2015, pp. 1163-1168.

[3] H. Midorikawa: "Blk-Tune: Blocking Parameter Auto-Tuning to Minimize Input-Output Traffic for Flash-Based Out-of-Core Stencil Computations", The 11th Int. workshop on Automatic Performance Tuning, iWAPT2016, IEEE Int. Parallel and Distributed Process. Symp. Workshops (DOI 10.1109/IPDPSW.2016.48), pp.1516-1526, May 2016

[4] H.Midorikawa, H.Tan: "Evaluation of Flash-based Out-of-core Stencil Computation Algorithms for SSD-Equipped Clusters", The 22nd IEEE International Conference on Parallel and Distributed Systems ICPADS2016 , pp.1031-1040, DOI: 10.1109/ICPADS.2016.0137

[5] Proc. of Flash Memory Summit 2016, Aug. 9-11, 2016, Santa Clara.
http://www.flashmemorysummit.com/English/Collaterals/Documents/FlashMemorySummit_Preview_Program.pdf

[6] F. Bassetti, et.al., "Optimizing Transformations of Stencil Operations for Parallel Object-Oriented Scientific Frameworks on Cache-Based Architectures", ISCOPE'98, LNCS 1505, pp107-118, 1998.

[7] L. Renganaranaya, et al., "Toward Optimal Multi-level Tiling for Stencil Computations", IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007).

[8] D. Wonnacott, "Using Time Skewing to Eliminate Idle Time Due to Memory Bandwidth and Network Limitations", International Parallel and Distributed Processing Symposium (IPDPS 2000).

[9] OpenNVM, FusionIO, http://opennvm.github.io.

[10] M Bjørling, J Axboe, D Nellans, P Bonnet, "Linux Block IO: Introducing Multi-Queue SSD Access on Multi-Core Systems", Proc. of the 6th International Systems and Storage Conference (SYSTOR '13)