DOI 10.12694/scpe.v19i1.1389
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 1, pp. 1–14. http://www.scpe.org

A SOLUTION TO IMAGE PROCESSING WITH PARALLEL MPI I/O AND DISTRIBUTED NVRAM CACHE

ARTUR MALINOWSKI AND PAWEŁ CZARNUL*

Abstract. The paper presents a new approach to parallel image processing using byte addressable, non-volatile memory (NVRAM). We show that our custom built MPI I/O implementation of selected functions that use a distributed cache that incorporates NVRAMs located in cluster nodes can be used for efficient processing of large images. We demonstrate performance benefits of such a solution compared to a traditional implementation without NVRAM for various sizes of buffers used to read image parts, process and write back to storage. We also show that our implementation benefits from overlapping reading subsequent images while processing already loaded ones. We present results obtained in a cluster environment for three parallel implementation of blur, multipass blur and Sobel filters, for various NVRAM parameters such as latencies and bandwidth values.

Key words: image processing, high performance computing, NVRAM, distributed cache, Sobel, blur filter

AMS subject classifications. 68U10, 68W10

1. Introduction. For many customers the number of recorded megapixels is a key factor when choosing digital cameras. Assuming that engineers properly matched the size of an image sensor to its resolution, it is completely justified – each pixel contains additional information that could be used in order to improve quality of an image. The first affordable, commercially available digital cameras started from the resolution of about one megapixel, like Kodak DCS or NASA's Nikon F4 that was used during Space Shuttle missions [26]. The megapixel race led to about 20 megapixel sensors in modern smartphones and more than 50 megapixel sensors in DSLR equipment for professional photographers. Some digital cameras take things even a step further, like the recently announced Hasselblad device with effective resolution of 400 megapixels [11]. The scale grows even bigger for specialized devices, such as Hawaii telescopes of the project Pan-STARRS that are equipped with sensors of a resolution more than one gigapixel [13].

Apart from better sensors, the final image resolution could be obtained by combining multiple smaller parts. The sharpest view of the Andromeda Galaxy, created by NASA/ESA using data from Hubble telescope, contains 1.5 billion pixels [27]. This technique is useful not only in scientific research – in 2016 Bentley Motors created a 53 gigapixel photo only for demonstration of the companys commitment to technological innovation [38].

Large images are more difficult during processing. The size of a file could exceed the amount of RAM installed within a single node. Moreover, more demanding algorithms involve complex computations. It is possible to offload some processing to GPU (compatible with GPGPU technologies, e.g. NVIDIA CUDA, OpenCL) or computational accelerators (e.g. Intel® Xeon Phi®), but it may require many round trips between a host memory and a device due to limited memory on such compute devices.

An alternative solution for high data volume (especially with multiple images) and demanding computations is changing the processing application from running on a single node to the one distributed among several nodes. With such an approach, an application designer can include all of the techniques and methods of High Performance Computing (HPC) in the application. It should be especially convenient for processing used in scientific applications, where developers are familiar with HPC tools and technologies.

Within this paper we propose a distributed architecture for a large image processing application based on HPC tools. The solution is created using Message Passing Interface (MPI), image file access is accelerated by a byte-addressable non-volatile RAM (NVRAM) distributed cache. Initially, the framework consists of three exemplary image filters, but it can be easily extended further. A set of experiments proved that the architecture is capable to process large images (single or multiple) in reasonable time.

2. Related work and motivation. Firstly, parallel image processing, as a way to decrease execution times of image filtering, important in many fields, has been analyzed and used for many years already.

*Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland, artur.malinowski@pg.edu.pl, pczarnul@eti.pg.edu.pl

2.1. Parallel image processing. Parallel programming with Message Passing Interface (MPI) traditionally allows utilization of clusters but MPI-based programs can be also run successfully on powerful workstations with multi- (such as Intel Xeon) and many-core processors (such as Intel Xeon Phi x200) or many-core coprocessors (such as Intel Xeon Phi x100). Parallel image processing with MPI has been analyzed in many works. Paper [29] presents design and results of a C+MPI framework for low-level image processing, run on a cluster of up to 64 nodes connected with Myrinet. An example of a multi-baseline stereo vision application is used for which speed-ups up to around 10 have been obtained on 32 nodes. In work [28] authors demonstrated parallel extensions, also using MPI, to the Delft Image Processing LIBrary (DIPLIB) library. For geometric mean filters and larger window sizes and image sizes (15x15 for an 256x256 and 9x9 and 15x15 for 1024x1024 images) the authors have obtained linear speed-ups up to 24 machines. Paper [33] introduced Parallel Image Processing Toolkit (PIPT) that uses MPI, with load balancing schemes in the framework for transparent distribution of computations. Paper [2] deals with parallel image processing and considers data distribution for heterogeneous machines with scalability results for an active contour algorithm. Website [5] provides a C/C++ with MPI implementation of several operations on images: contrast an image, filtering: smooth, blur, sharpen, mean_removal, emboss as well as computing image entropy.

Other parallel programming APIs allowing execution on clusters have also been used for image processing. In paper [6] the authors extended the well known image processing tool GIMP with a possibility to use a cluster based system for pipelined image processing. Paper [32] presents parallel processing of pressure-sensitive paint images on a multiprocessor machine or a cluster with multiple nodes, however implemented with forks/pipes and TCP/IP.

Recently, parallel image processing with GPUs has been explored deeper in many fields, for example: plant growth analysis [30], medical applications such as cancer research [31] object recognition [35], embedded systems [4]. Several frameworks for image processing with GPUs have been developed [1, 16] along with a possibility to perform GPU image processing from higher level systems such as MATLAB [10]. However, GPUs have limitations in terms of maximum memory capacity – currently up to 16GB in the latest and expensive cards such as NVIDIA V100 or 12GB in NVIDIA Titan X. Consumer grade cards offer up to 8GB of memory. In view of this, processing of very large images might require several communications over PCI Express and the overall performance will suffer. Because of this, we explore the possibility of using NVRAM in parallel image processing, especially in a cluster environment, in which NVRAMs from various nodes might offer even higher capacities.

2.2. NVRAM applications. Non-volatile byte-addressable memory has several potential advantages that make it an interesting solution in increasing performance of demanding applications [19]. These include byte-addressability, sizes larger than RAM and persistence. There are several examples of applications analyzed in the literature.

One is low overhead checkpointing. Paper [14] proposes NVM-checkpoints for storing checkpoints locally and remotely. Authors propose checkpointing based on a hybrid memory model. An application uses an NVM interface for provision of information regarding checkpointed data. While data remains in RAM, it can be copied to NVM. NVM is used for local and less frequent remote checkpoints. Apart from an NVM kernel manager, the provided NVM user library for handling checkpointed data: allocation, moving data from DRAM to NVM and restart. Checkpointing using NVRAM was also proposed by us in [7] where wrappers to MPI functions for use with NVRAM were proposed. We demonstrated that for expected performance characteristics of actual NVRAM devices (latencies and bandwidth) NVRAM based checkpointing performs considerably better than the traditional disk based approach for applications such as the HPCCG benchmark and the PageRank algorithm.

In terms of storage oriented solutions, several contributions have been made. Paper [17] presents NVWAL that uses NVRAM for maintaining a write-ahead log that benefits from byte addressability of NVRAM, provides a transaction-aware persistency and shows that NVWAL provides considerably better performance for SQLite compared to using flash memory. In paper [39] authors present Mojim which is a two-tier system where the first one contains a mirrored pair of nodes and the second tier encompasses secondary backup nodes with weakly consistent data copies. The system provides reliability and availability. The paper demonstrates that a solution with replication with non-volatile memory provides similar or better performance than a version with non-volatile memory without replication. In paper [9] authors propose Phoenix (PHX) which is an NVRAM-

bandwidth aware object store for persistent objects. What is interesting is the fact that it can use NVRAM and DRAM simultaneously as well as incorporate information such as bandwidths of devices, distances and energy costs. The authors have presented that their solution reduced checkpoint/restart times for three tested HPC benchmarks such as GTC, CM1 and S3D HPC compared to NVRAM only solutions.

Work [18] demonstrates how NVRAM can be used in order to improve performance of a browser. This includes making use of NVRAM for placement of files for startup. Furthermore, perceived performance is increased through caching of web resources in NVRAM.

Another application that can benefit from NVRAM is online transaction processing. Paper [12] presents how NVRAM can be used for a logging subsystem and justifies that it provides better performance to cost ratio than replacing the whole storage with NVRAM.

Paper [8], on the other hand, provides a study of impact of NVRAM on a Breadth-First Search (BFS) graph traversal algorithm. An NVRAM simulator called PerMA has been used which allows to model latencies and bandwidths of memory types from flash to RAM. The authors came to the conclusion that with sufficient concurrency, with NVRAM the analyzed algorithm will be able to approach the performance of an in-memory algorithm.

In paper [15] authors investigate benefits from using NVRAM for large scale data intensive (I/O) applications and demonstrated gains such as 3.85x I/O throughput and 1.6x for ort based data post processing over a disk only approach.

Paper [36] demonstrates benefits through an average of 2.7x performance improvement of the map phase of Map Reduce from using NVRAM. Benchmarks and workloads included those from Intel HiBench and PUMA. Low level optimization of processing using NVRAM is analyzed in work [20] in which authors presents a software cache in which lines to be flushed are buffered first and flushed later. Cache size is adapted at run time.

So far image processing with NVRAM has been addressed to a certain degree in the literature in terms of energy efficiency. For instance, paper [25] presents that power consumption for parallel image processing can be reduced greatly with the use of non-volatile memory. Work [37] presents energy efficient in memory machine learning for image processing and corresponding benefits when using non-volatile memory.

2.3. Motivation and goal. Taking into account the aforementioned contributions, in this work we propose to integrate usage of NVRAM for parallel image processing, especially large images, and demonstrate benefits of this approach. With expected adoption of NVRAM in the nearest future, we believe that it will be an asset applicable to a variety of fields and users.

3. Proposed solution.

3.1. Assumptions. From the HPC perspective, image processing could be regarded as performing a set of operations on a two-dimensional matrix in which each element corresponds to a pixel of a selected color. In the most straightforward approach an application performs the following steps:

1. The application opens an image file and reads an image as a matrix.
2. The matrix is split into submatrices.
3. Submatrices are assigned to processes (one to one or many to one depending on the processing paradigm)
4. Each process reads required data, performs its computations on the assigned part and writes output.
5. The application closes the file.

We believe that these steps make the application as simple as possible from a developer point of view. Our solution sticks to this in order to make it easy to implement own image processing algorithms. On the other hand, our solution uses NVRAM in an intermediate layer between files and the MPI I/O functions invoked within the application.

Such an approach should be efficient with images of a size limited to the sum of all RAM capacities in a cluster. In such case a file is read once at the beginning of processing and written back after computations have been completed. The situation changes when the size of an image increases. The greater the size of a file, the smaller image part can be processed at once and the application requires more read/write requests. Finally, the application that used to be computationally bound becomes data intensive and I/O operations appear to be a bottleneck.

Taking everything into account, the task is to design the application as simple as possible, keeping in mind that in order to provide good application performance, it is required to perform I/O operations efficiently.

3.2. Design of the solution. As a base for our application we used the C programming language and MPI which is the de facto standard for message passing parallel programming allowing to run applications on clusters but also within nodes with multi-core CPUs. Choosing MPI ensures a wide knowledge of a platform among HPC application programmers – e.g. MVAPICH, one of most popular MPI implementation, declares being used by more than 2,750 organizations[1]. Many MPI implementations also offer additional advantages like built-in Infiniband integration, efficient process managers, several levels of threads support or dynamic process management routines.

In order to provide file access for all nodes within a cluster, in a typical HPC environment a parallel file system (PFS) is used. Most popular PFSes provide POSIX support, but applications based on MPI can use MPI I/O – a set of functions that allow accessing a file in a way convenient for a programmer. Popular MPI I/O implementations also include many different optimizations, e.g. data sieving and two-phase I/O in ROMIO [34].

One of the conditions for comfortable file usage is the possibility to read and write a data chunk efficiently independently from its location and size. As we will show in experiments, performance of specific operations in regular MPI I/O and PFS could be significantly improved using our byte-addressable NVRAM distributed cache [23].

The NVRAM cache, previously proposed by us, is a transparent component placed between an application and MPI I/O. Its transparency is achieved through reimplementing selected MPI I/O API, so no additional effort is needed from a developer. The most important requirements of this extension are installation of a NVRAM device in each node of a cluster and the size of a file limited to the sum of all NVRAM capacities. As justified in related work, NVRAM capacities are expected to far exceed RAM properties, so it should not be a problem in the nearest future. The main distinguishing features of the cache are fully decentralized management, prefetching the whole file during opening, synchronizing the whole file during closing and keeping minimal metadata. A set of tests with synthetic benchmarks and real-life applications like map searching, crowd simulation or graph processing proved I/O improved performance, especially for long running applications [21, 22, 23]. An additional feature of the cache that naturally benefits from NVRAM persistence is safety of the data during processing [24]. The cache can also run using volatile RAM as its storage. Such a configuration forces reducing RAM available for the application and does not offer any fail-safe mechanisms, but can be successfully applied in order to enhance the efficiency of I/O.

Figure 3.1 presents the most important architecture components and their dependencies. As previously stated, image files are served by a PFS. An application accesses it using MPI I/O API. The NVRAM cache is a transparent component that improves efficiency of file access. Optionally, an application can use computational accelerators.

3.3. Performance optimization. According to the description of the NVRAM based cache, overhead for opening and closing a file may have a significant impact on application execution time. In a typical HPC case this issue is negligible – in long running applications the gain from faster data access fully compensates the initialization and deinitialization phases. However, omitting this overhead for fast and simple image processing algorithms would be noticeable. The proposed application is prepared to be used as a service that is able to process requested images one by one. The common idea in HPC used for avoidance of waiting for a data is overlapping communication and computation. In our application we implemented a similar approach – image processing overlaps with opening/closing a file.

Another important task was tuning the PFS. Our cluster was equipped both with Infiniband and 10Gb/s Ethernet, so we were able to separate the application and the PFS traffic. Internal MPI communication was based on Infiniband, while our cache was connected to PFS using Ethernet. The OrangeFS setting that has the most impact on significant reduction of application execution time was turning off TroveSyncData option. It allowed to omit costly data synchronization after each operation at the cost of increased risk of loosing data. Instead of protection offered by OrangeFS we can use the fail-safe mode built into the NVRAM cache mechanism or take the risk – in the worst case the application will process a lost image once more.

[1] `http://mvapich.cse.ohio-state.edu/`

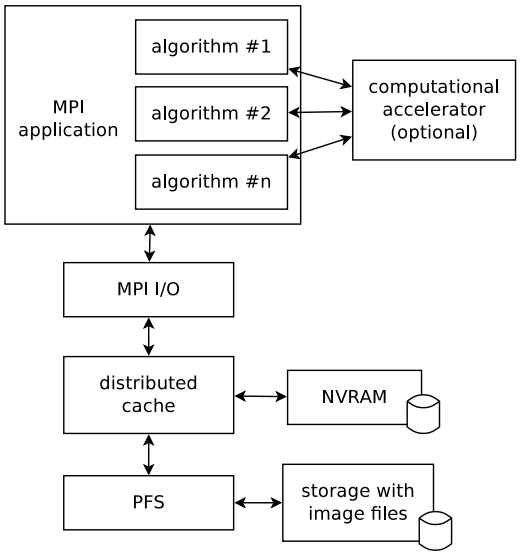


FIG. 3.1. *Architecture of the application for large image processing*

Application optimizations included also, among others, tuning buffer sizes, reducing number of round-trips between PFS and NVRAM cache or proper usage of MPI I/O flags like `MPI_MODE_READONLY`.

3.4. Exemplary filters. Our implementation contains code for parallel execution of three different filters. Figure 3.2 presents exemples of an original image and processed results. The initial version of application contains the following filters:

1. Blur – the idea is to blur an image by averaging values of neighboring pixels (to each pixel).
2. Multi-pass blur – similarly to the standard blur but the value of a considered pixel is also considered in the average. Additionally, several passes through an image are executed.
3. Sobel – in this case, application of the filter relies on conversion of each pixel to grayscale (using a luminosity approach) and application of a 3x3 operator on each pixel. This allows to compute minimum and maximum values across a domain and then normalize final values based on these minimum and maximum values. It should be noted that these minimum and maximum values need to be propagated across all processes (realized using `MPI_Allreduce()`).

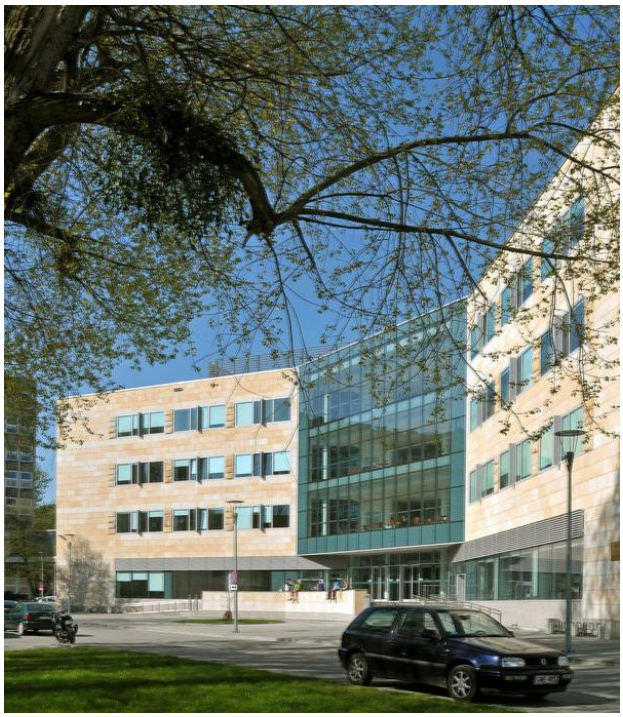
4. Experiments. The main goal of experiments was not only to show that the application processes images in reasonable time, but also to present a comparison of an architecture with and without the byte-addressable NVRAM distributed cache. As parameters of expected NVRAM devices are not yet published, the last three experiments were dedicated to different settings of the simulation platform.

4.1. Testbed environment. The extension was tested on cluster named Lap06, its technical specifications are included in Table 4.1 and Table 4.2. As the actual NVRAM devices are not available on the market yet, we used a hardware simulation platform. The platform internally uses RAM but increases its access latency and modifies the bandwidth. Unless otherwise noted, the platform was set according to Table 4.3.

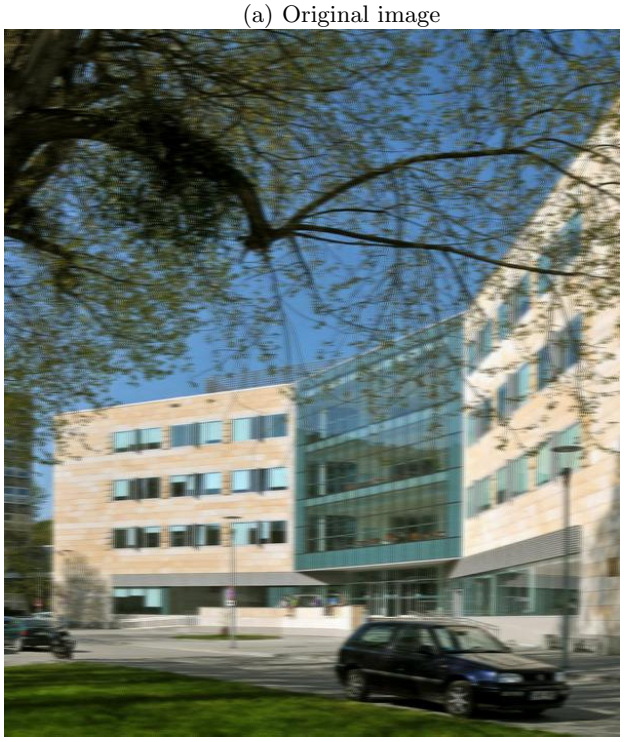
4.2. Results of NVRAM extension vs regular MPI I/O.

4.2.1. Various image sizes. The first set of experiments verified the possibility of processing large images efficiently. In this scenario we processed a single image using 6 computing nodes, 7 processes per each. As presented in Fig. 4.1 and 4.2, in one minute the application was able to apply blur filter on an 800 megapixel image and blur multi-pass filter (10 passes) on a 500 megapixel image. As expected, processing time increases linearly with increasing the size of an image. Comparison of unmodified MPI I/O and the one supported by NVRAM cache clearly shows that extended version performed much better. For the blur filter execution time was about 30% lower for small images (100 megapixels) up to more than 40% lower for images larger that 500

(a) Original image



(b) Blur filter



(c) Blur-multipass filter

(d) Sobel filter

FIG. 3.2. *Original image and images processed using three different filters (fragment of a photo of Gdansk University of Technology, author: Krzysztof Krzempek)*

TABLE 4.1
Hardware used in performance tests

Number of computing nodes	6
Number of PFS nodes	2
CPU	2 x Intel® Xeon® E5-4620
RAM	15GB
Network	40Gb/s Infiniband, 10Gb/s Ethernet
Storage	SSD
NVRAM simulation	17GB, hardware simulation

TABLE 4.2
Clusters' software configuration

Operating system	CentOS release 6.5
MPI implementation	MPICH 3.2
PFS	Orange-FS 2.9.6

TABLE 4.3
NVRAM simulation platform parameters

additional latency before accessing the data	2000ns
additional latency before flushing the data on device	600ns
memory bandwidth divider	4

megapixels. According to previous expectations, less demanding algorithms like blur suffer from overhead for opening and closing a file. This issue does not occur with blur multi-pass – for this scenario we were able to observe more than 90% reduction of the execution time.

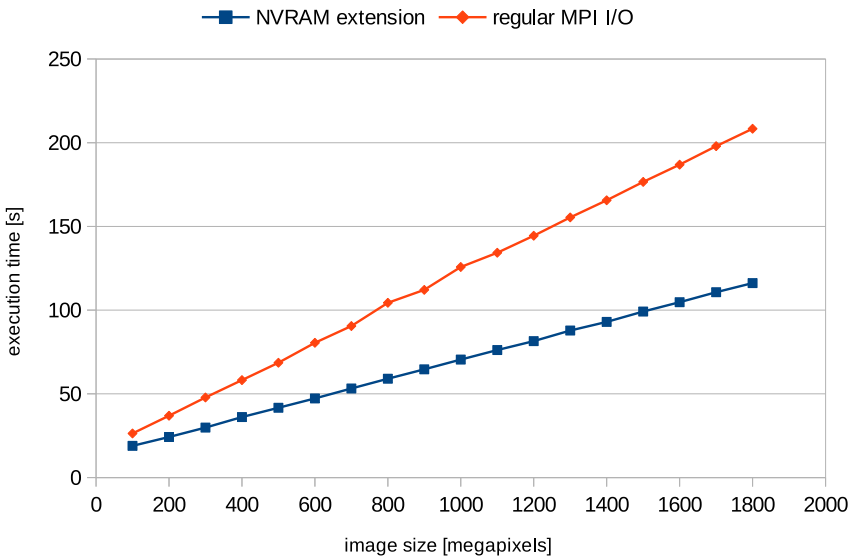


FIG. 4.1. *Image processing results, blur filter, 6 nodes, 42 processes, 512kB application buffers*

4.2.2. Various number of images. The next set of tests was focused on testing effectiveness of overlapping image processing with opening/closing a file. In order to verify the approach we compared the proposed application running with multiple files and the same application, but executed for each single image sequentially. Results presented in Fig. 4.3 and 4.4 demonstrate reduced execution times for application with overlapping.

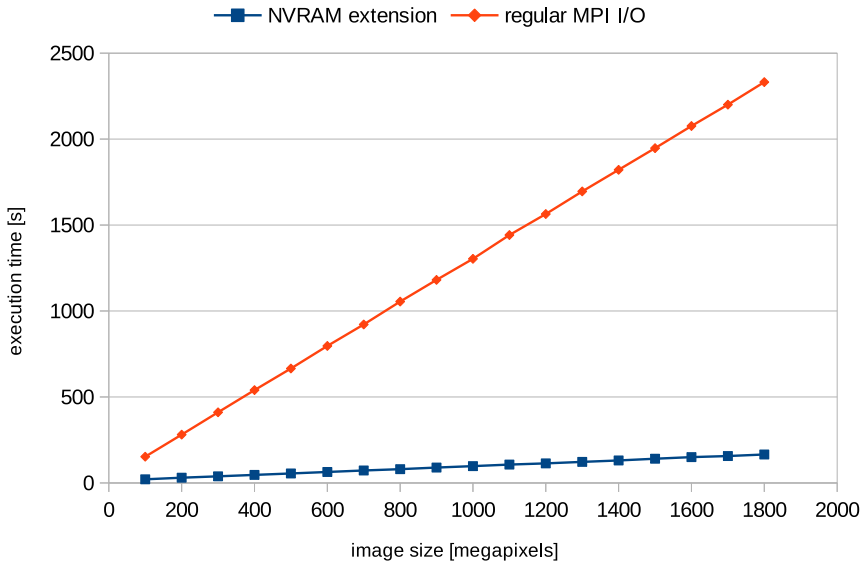


FIG. 4.2. *Image processing results, blur multi-pass filter, 6 nodes, 42 processes, 512kB application buffers*

Unfortunately, the gain from implementation of overlapping was lower that expected. For a relatively simple filter – Sobel – we observed reduction of execution time at the level of 5%. More complicated algorithms like blur multi-pass allow for saving more time – for test scenario illustrated in Fig. 4.4 it was about 10%.

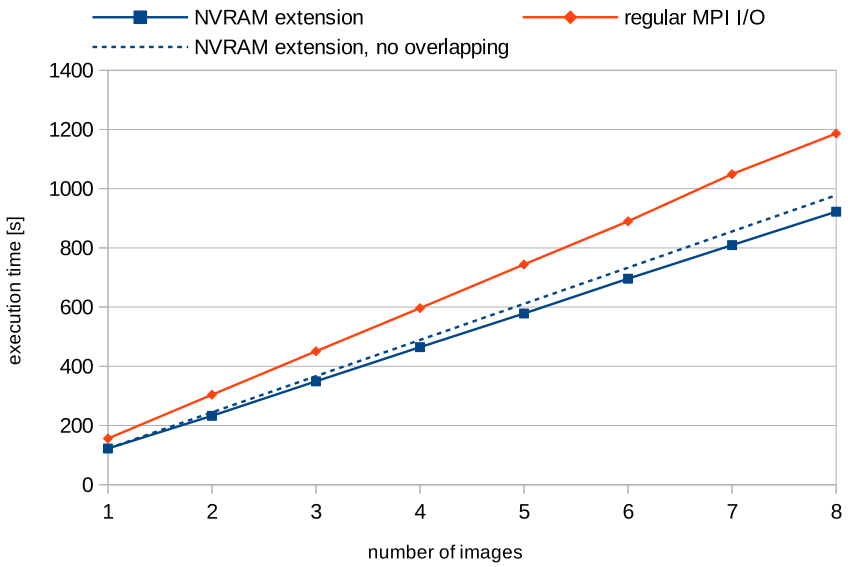


FIG. 4.3. *Image processing results, Sobel filter, 6 nodes, 42 processes, 512kB application buffer, each image of 1 gigapixel*

4.2.3. Various buffer sizes. Although our NVRAM distributed cache is designed to work well with small data chunks (even with access to single bytes), the proposed application uses internal buffers. Two buffers located in RAM, one for read and one for write operations, prevent from too frequent file requests. Results presented in Fig. 4.5 and 4.6 prove that the NVRAM cache is prepared for serving even small data chunks.

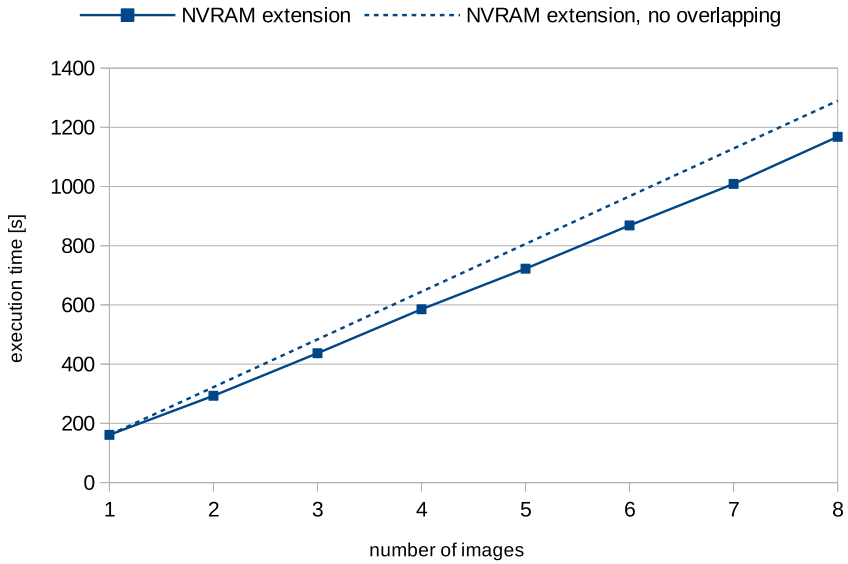


FIG. 4.4. *Image processing results, blur multi-pass filter, 6 nodes, 42 processes, 512kB application buffer, each image of 1 gigapixel*

Results were similar both for simple and more demanding processing algorithms. With unmodified MPI I/O and requests lower than 128kB the application is extremely slow, because the PFS is flooded with a large number of requests incoming frequently. In our opinion, such a result is another argument for applying the proposed architecture – implementing buffers is an additional overhead for developers, especially for more complicated, non-linear image processing algorithms. The proposed solution allows to focus more on implementing algorithms themselves, rather than on difficult I/O optimization.

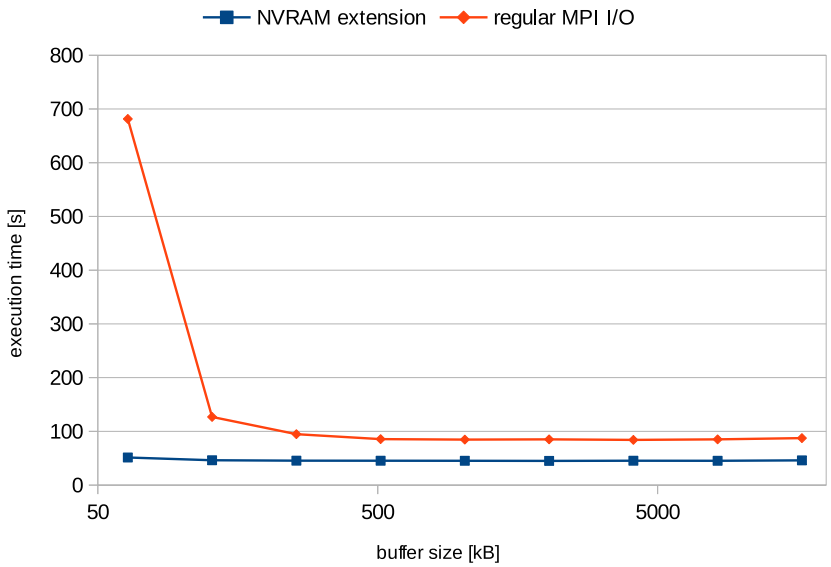


FIG. 4.5. *Image processing results, Sobel filter, 6 nodes, 42 processes, image of 0.5 gigapixel*

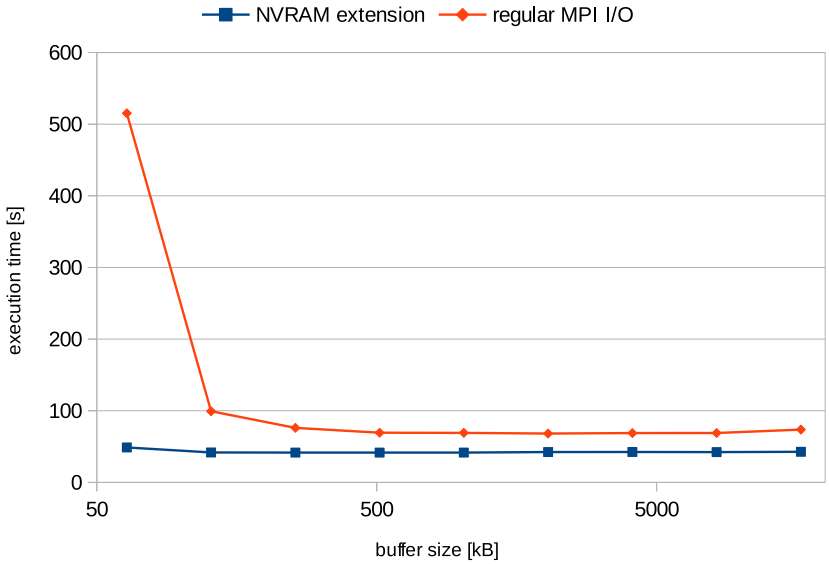


FIG. 4.6. *Image processing results, blur filter, 6 nodes, 42 processes, image of 0.5 gigapixel*

4.2.4. Scalability. One of the most important parameters of HPC applications is scalability regarded as the potential of reducing execution time with increasing hardware resources, today most often the number of a cluster nodes and consequently the number of CPUs and cores. It may seem that with processing multiple images, the application does not require good scalability because it could be executed multiple times for each image independently. In practice, when the I/O is the bottleneck of the system, running many instances of a data intensive application may result in overloading of PFS.

Figure 4.7 shows application speedup while increasing the number of nodes. Unmodified MPI I/O does not scale well because the gain from higher computational power of greater number of nodes is insignificant when PFS is more and more overloaded. The speedup of execution with NVRAM cache is also far from linear, but still significant. Scalability is of the features of the distributed architecture of NVRAM cache. The solution is designed in such way that each node participates in serving read/write requests. With an increasing number of a file accesses from a higher number of processes, the extension has more nodes to process it, so the average number of requests per node is constant. Furthermore, better scalability results are expected for more demanding algorithms with a higher ratio of computations to I/O.

4.2.5. Various NVRAM simulation parameters. As we do not have NVRAM based devices yet, we used a hardware simulation platform. Unknown properties of final devices result in necessity for testing solutions for many different configurations in the range of expected parameter values. Although NVRAM devices should outperform today's SSDs, we assumed pessimistic values at the level similar to announced SSD specifications (i.e. Intel® Optane® P4800X with typical latency of less than 10μs, up to 2400/2000MB/s read/write speed and 500k IOPS for random requests [3]). The following three tests were performed using the blur multi-pass filter.

Figure 4.8 presents comparison of execution times according to the memory bandwidth. Our nodes were equipped with DDR3 1600Mhz memory units, the simulator allowed to divide the bandwidth by a certain factor. In the plot we can observe that this parameter does not have a significant impact on the proposed application – average growth of the execution time after reducing the bandwidth was less than 2.6%. With frequent, low size requests our application depends more on the access latency rather than bandwidth.

Results shown in Fig. 4.9 concern an additional delay required to flush the cached data onto the device to make it persistent. The plot is quite similar to the previous one – even when the additional latency before flushing the data was doubled, average execution time of the application grew up about 2%. This latency is

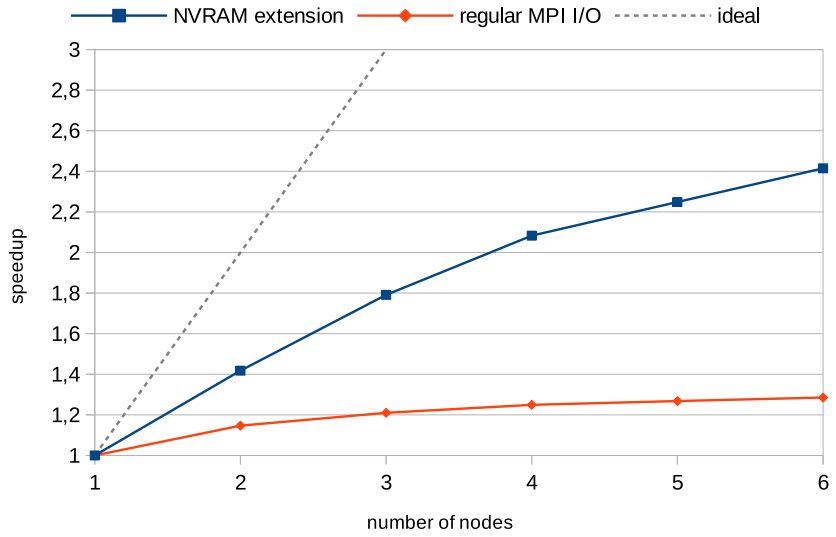


FIG. 4.7. *Image processing results, blur multi-pass filter, 512kB application buffers, image of 1.5 gigapixel*

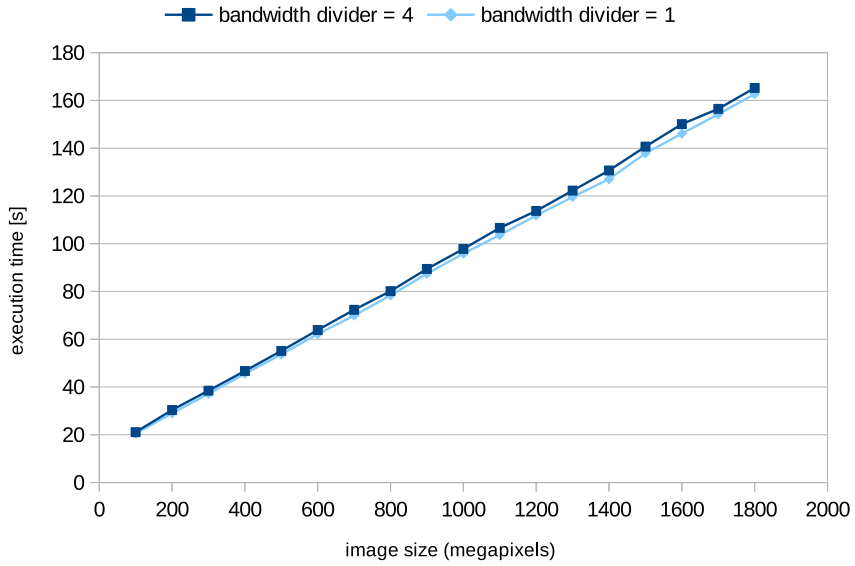


FIG. 4.8. *Image processing results, blur multi-pass filter, 6 nodes, 42 processes, 128kB application buffer*

added only for write accesses and in our application write operations are less common than reads.

A much more visible impact on execution time is shown in Fig. 4.10. In this scenario we increased the additional latency added for each memory request. Obtained results resulted in about 7% growth of processing time.

Those three experiments proved that the impact of NVRAM parameters is significant in terms of execution time, but insignificant when comparing the application with NVRAM cache and the one without. This leads to the conclusion that if only NVRAM devices provide performance at the level of SSD devices or better, the proposed architecture will be more efficient using the proposed distributed cache.

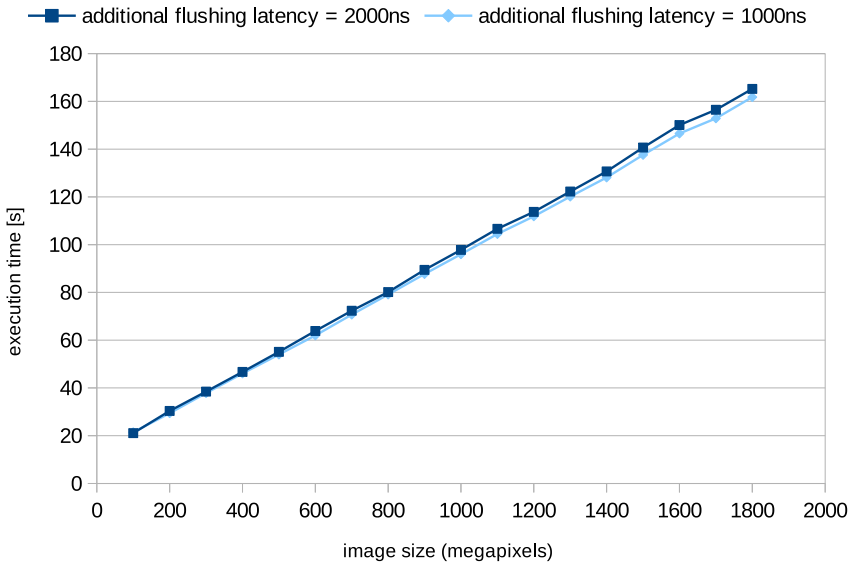


FIG. 4.9. *Image processing results, blur multi-pass filter, 6 nodes, 42 processes, 128kB application buffer*

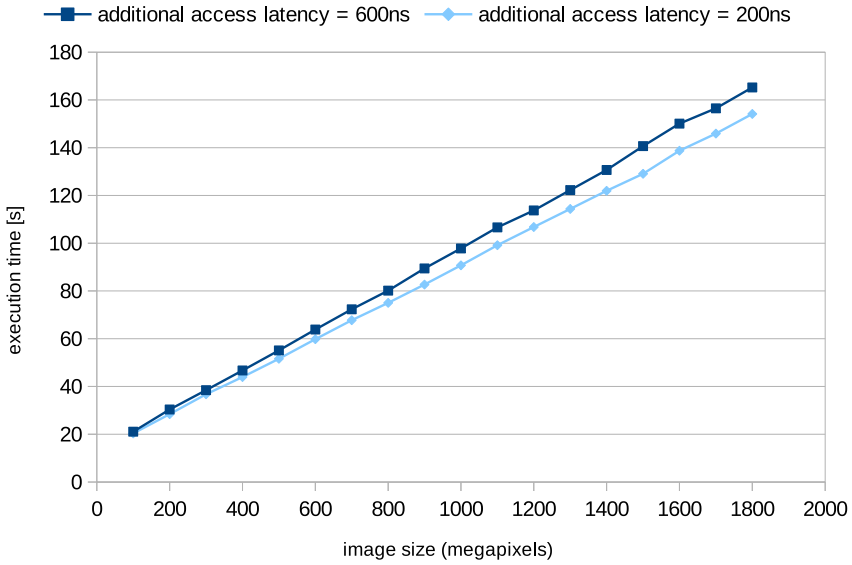


FIG. 4.10. *Image processing results, blur multi-pass filter, 6 nodes, 42 processes, 128kB application buffer*

5. Conclusions and future work. In this paper we proposed an architecture and software/hardware components for large image processing application. Motivations included observation of increasing images sizes and wide interest of efficient image processing architectures collected in the related work. In the exemplary implementation we included three image processing filters: blur, Sobel and multi-pass version of blur. The application is able to process a single image, as well as multiple images using additional optimization that involves overlapping file opening and processing for subsequent images. The most distinguishing feature of the proposed solution is application of byte-addressable NVRAM distributed cache. Emerging memory technology combined with cache design allows for reducing PFS load, making the application development more convenient

by using small data chunks efficiently and easily obtain better scalability with data intensive applications. The presented experimental results show, among others:

1. efficiency of large (more than gigapixel) image processing,
2. better performance of NVRAM cache extension compared to unmodified MPI I/O,
3. performance gain obtained for processing multiple images using overlapping file opening and processing,
4. visible scalability of the solution,
5. impact of NVRAM parameters on the application execution time.

In the future we plan to improve performance of the NVRAM cache, which should also impact efficiency of image processing with proposed architecture. Our idea involves a hybrid approach – using both plain PFS performance and cache by balancing the load at runtime. Another interesting issue is connected with moving the solution into a cloud – combining HPC and cloud processing becomes more and more popular nowadays.

Acknowledgments. The research in the paper was supported by a grant from Intel Technology Poland and afterwards partially elaborated within statutory activities of Dept. of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland.

REFERENCES

[1] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, *GpuCV: A GPU-Accelerated Framework for Image Processing and Computer Vision*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 430–439.

[2] J. Barbosa, J. Tavares, and A. J. Padilha, *Parallel Image Processing System on a Cluster of Personal Computers*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 439–452.

[3] P. Bright, *Specs for first Intel 3D XPoint SSD: so-so transfer speed, awesome random I/O*, 2017. `https://arstechnica.com/?post_type=post&p=1040631`.

[4] M. Cavus, H. D. Sumerkan, O. S. Simsek, H. Hassan, A. G. Yaglikci, and O. Ergin, *Gpu based parallel image processing library for embedded systems*, 2014 International Conference on Computer Vision Theory and Applications (VISAPP), 1 (2014), pp. 234–241.

[5] S.-D. Cosmin, *Mpi-image-processor*, March 2012. https://github.com/cosminstefanxp/MPI-Image-Processor.

[6] P. Czarnul, A. Ciereszko, and M. Fraczak, *Towards Efficient Parallel Image Processing on Cluster Grids Using GIMP*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 451–458.

[7] P. Dorozynski, P. Czarnul, A. Malinowski, K. Czurylo, L. Dorau, M. Maciejewski, and P. Skowron, *Checkpointing of parallel mpi applications using mpi one-sided api with support for byte-addressable non-volatile ram*, Procedia Computer Science, 80 (2016), pp. 30 – 40.

[8] B. V. Essen, R. Pearce, S. Ames, and M. Gokhale, *On the role of nvram in data-intensive architectures: An evaluation*, in 2012 IEEE 26th International Parallel and Distributed Processing Symposium, May 2012, pp. 703–714.

[9] P. Fernando, S. Kannan, A. Gavrilovska, and K. Schwan, *Phoenix: Memory speed hpc i/o with nvm*, in 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), Dec 2016, pp. 121–131.

[10] A. Georgantzoglou, J. d. Silva, and R. Jena, *Image processing with matlab and gpu*, in MATLAB Applications for the Practical Engineer, K. Bennett, ed., InTech, Rijeka, 2014, ch. 22.

[11] Hasselblad Press Release, *Hasselblad introduces the H6D-400c MS*, 2018. `https://www.hasselblad.com/press/press-releases/hasselblad-introduces-the-h6d-400c-ms/`.

[12] J. Huang, K. Schwan, and M. K. Qureshi, *Nvram-aware logging in transaction systems*, Proc. VLDB Endow., 8 (2014), pp. 389–400.

[13] N. Kaiser, W. Burgett, K. Chambers, L. Denneau, J. Heasley, R. Jedicke, E. Magnier, J. Morgan, P. Onaka, and J. Tonry, *The Pan-STARRS wide-field optical/NIR imaging survey*, Proc. SPIE, 7733 (2010), pp. 77330E–77330E–14.

[14] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, *Optimizing checkpoints using nvm as virtual memory*, in 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, May 2013, pp. 29–40.

[15] S. Kannan, D. Milojicic, A. Gavrilovska, and K. Schwan, *Using active nvram for i/o staging*, 2011. HP Laboratories, HPL-2011-131, http://www.hpl.hp.com/techreports/2011/HPL-2011-131.pdf.

[16] P. Karlsson, *A gpu-based framework for efficient image processing*, September 2014. LiU-ITN-TEK-A-14/043-SE, Department of Science and Technology, Linkping University, Sweden.

[17] W.-H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, *Nvwal: Exploiting nvram in write-ahead logging*, SIGOPS Oper. Syst. Rev., 50 (2016), pp. 385–398.

[18] K. Kyusik, C. Yongwoon, K. Seongmin, and K. Taeseok, *Reducing the user-perceived latency of browsers with nvram*, JSTS:Journal of Semiconductor Technology and Science, 17 (2017), pp. 23–28. DOI: 10.5573/JSTS.2017.17.1.023.

[19] J. Layton, *How persistent memory will change computing.* Admin Magazine, accessed on 26th April 2017. http://www.admin-magazine.com/HPC/Articles/Persistent-Memory, Linux New Media USA, LLC.

[20] P. Li and D. R. Chakrabarti, *Adaptive Software Caching for Efficient NVRAM Data Persistence*, Springer International Publishing, Cham, 2017, pp. 93–97.

[21] A. Malinowski and P. Czarnul, *Distributed NVRAM Cache – Optimization and Evaluation with Power of Adjacency Matrix*, Springer International Publishing, Cham, 2017, pp. 15–26.

[22] A. Malinowski, P. Czarnul, K. Czuryło, M. Maciejewski, and P. Skowron, *Multi-agent large-scale parallel crowd simulation*, Procedia Computer Science, 108 (2017), pp. 917 – 926. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[23] A. Malinowski, P. Czarnul, P. Dorożynski, K. Czuryło, Ł. Dorau, M. Maciejewski, and P. Skowron, *A Parallel MPI I/O Solution Supported by Byte-addressable Non-volatile RAM Distributed Cache*, in Position Papers of the 2016 Federated Conference on Computer Science and Information Systems, vol. 9 of Annals of Computer Science and Information Systems, PTI, 2016, pp. 133–140.

[24] A. Malinowski, P. Czarnul, M. Maciejewski, and P. Skowron, *A Fail-Safe NVRAM Based Mechanism for Efficient Creation and Recovery of Data Copies in Parallel MPI Applications*, in Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II, Springer International Publishing, 2017, pp. 137–147.

[25] A. Mochizuki, N. Yube, and T. Hanyu, *Design of a computational nonvolatile ram for a greedy energy-efficient vlsi processor*, in IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, Nov 2015, pp. 003283–003288.

[26] NASA, *Space Shuttle Mission STS-48 Press Kit*, 1991. `https://science.ksc.nasa.gov/shuttle/missions/sts-48/sts-48-press-kit.txt`.

[27] NASA/ESA, *Hubbles High-Definition Panoramic View of the Andromeda Galaxy*, 2015. `http://www.spacetelescope.org/images/heic1502a/`.

[28] C. Nicolescu and P. Jonker, *Parallel low-level image processing on a distributed-memory system*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 226–233.

[29] C. Nicolescu and P. Jonker, *A data and task parallel image processing environment*, Parallel Computing, 28 (2002), pp. 945 – 965.

[30] A. Ozdemir and T. Altilar, *Gpu based parallel image processing for plant growth analysis*, in 2014 The Third International Conference on Agro-Geoinformatics, Aug 2014, pp. 1–6.

[31] A. Remnyi, S. Sznsi, I. Bndi, Z. Vmossy, G. Valcz, P. Bogdanov, S. Sergyn, and M. Kozlovszky, *Parallel biomedical image processing with gpgpus in cancer research*, in 3rd IEEE International Symposium on Logistics and Industrial Informatics, Aug 2011, pp. 245–248.

[32] W. Ruyten and W. E. Sisson, *Message passing for parallel processing of pressure-sensitive paint images*, in Users Group Conference (DOD UGC'04), 2004, June 2004, pp. 308–312.

[33] J. M. Squyres, A. Lumsdaine, and R. L. Stevenson, *A toolkit for parallel image processing*, in SPIE Annual Meeting, San Diego, 1998.

[34] R. Thakur, W. Gropp, and E. Lusk, *Data sieving and collective I/O in romio*, Frontiers '99 - Seventh Symposium On Frontiers Massively Parallel Computation, Proc., (1999), pp. 182–189.

[35] K. Vincent, D. Nguyen, B. Walker, T. Lu, and T.-H. Chao, *Gpu processing for parallel image processing and real-time object recognition*, 2014.

[36] M. Wasi-ur Rahman, N. S. Islam, X. Lu, and D. K. D. Panda, *Can non-volatile memory benefit mapreduce applications on hpc clusters?*, in Proceedings of the 1st Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, PDSW-DISCS '16, Piscataway, NJ, USA, 2016, IEEE Press, pp. 19–24.

[37] H. Yu, Y. Wang, S. Chen, W. Fei, C. Weng, J. Zhao, and Z. Wei, *Energy efficient in-memory machine learning for data intensive image-processing by non-volatile domain-wall memory*, in 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), Jan 2014, pp. 191–196.

[38] M. Zhang, *Bentley Used NASA Tech to Create This 53-Gigapixel Car Photo*, 2016. PetaPixel, `https://petapixel.com/2016/06/23/bentley-used-nasa-tech-create-53-gigapixel-photo-car/`.

[39] Y. Zhang, J. Yang, A. Memaripour, and S. Swanson, *Mojim: A reliable and highly-available non-volatile memory system*, in Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, New York, NY, USA, 2015, ACM, pp. 3–18.

Edited by: Dana Petcu
Received: Nov 6, 2017
Accepted: Dec 23, 2017