# Distributed NVRAM Cache – Optimization and Evaluation with Power of Adjacency Matrix

Artur Malinowski, Pawel Czarnul

# Distributed NVRAM cache – optimization and evaluation with power of adjacency matrix

Artur Malinowski and Paweł Czarnul

Dept. of Computer Architecture,
Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Gdansk, Poland
artur.malinowski@pg.gda.pl,pczarnul@eti.pg.gda.pl

**Abstract.** In this paper we build on our previously proposed MPI I/O NVRAM distributed cache for high performance computing. In each cluster node it incorporates NVRAMs which are used as an intermediate cache layer between an application and a file for fast read/write operations supported through wrappers of MPI I/O functions. In this paper we propose optimizations of the solution including handling of write requests with a synchronous mode, additional modes preventing data preloading from a file and synchronization on file close if the solution is used as temporary cache only. Furthermore, we have evaluated the solution for a real application that computes powers of an adjacency matrix of a graph in parallel. We demonstrated superiority of our solution compared to a regular MPI I/O implementation for various powers and numbers of graph nodes. Finally, we presented good scalability of the solution for more than 600 processes running on a large HPC cluster.

**Keywords:** NVRAM, distributed cache, graph processing, performance optimization

## 1 Introduction

High performance computing (HPC) was always related to aiming at better and better hardware. At the beginning, it mainly involved building larger clusters with better CPUs. A breakthrough emerged with manycore processors, such as graphics processing unit (GPU) used for general purpose processing (e.g. NVIDIA CUDA, OpenCL), coprocessors designed especially for computations (e.g. Intel® Xeon Phi™, Epiphany architecture) or even specially designed CPUs (e.g. Sunway processors used in Sunway TaihuLight, the most powerful supercomputer in TOP500 list, June 2016 edition).

Apart from a processing unit, another essential component of computer architecture is memory. With memory the situation is much different. Typical improvement of RAM is based on a higher capacity and better parameters (e.g. higher frequency, lower latency) of next generation Double Data Rate (DDR) RAM devices, while modern storage is always connected with superior SSDs. So far, no hardware device that would extend memory properties has acquired significant popularity.

Universal memory, that is only hypothetical for now, is one of possible candidates for breakthrough in memory technologies. Combining advantages of RAM (byte-level access, high speed and bandwidth, low latency) with advantages of SSD (large capacity and persistence) should definitely increase performance of many computer systems. Although not available on the market yet, many technologies are being researched to create a practical device and several of these seem to be promising [20]. Moreover, recent press reports suggest, that we can expect devices with parameters between NAND based memory (used in SSD) and DRAM soon [9][10]. In order to describe such memory in this paper, we would use the NVRAM term (non-volatile, random access memory) keeping in mind, that this memory is expected to have a byte-level access.

Full replacement of main memory and storage by a single, universal memory would probably trigger the need for redesigning the architecture of computing systems at multiple levels, but such a huge change cannot be expected to be introduced at once. Instead of this, we assume that first NVRAM devices would be used as complementary memory together with RAM and storage. For that reason, we decided to focus on possibilities coming from incorporating supplementary NVRAM devices into HPC platforms.

In 2016 we proposed the idea of NVRAM distributed cache located as an additional layer between a file system and a parallel distributed application [16]. The extension was transparent to the developer, because of its compatibility with the well-known Message Passing Interface (MPI) I/O API [18]. The motivation for this solution was improving performance and making the development process easier. Initial testing with a set of benchmarks gave promising results. Within this paper, we present further research on our MPI I/O NVRAM distributed cache. The research includes performance optimizations, as well as evaluation of the solution with a real life application – computing power of graph adjacency matrix. In fact, the application could be used e.g. for social network analysis or calculating shortest path lengths between multiple nodes simultaneously.

## 2 Related work

In 2009, Kryder and Kim presented a set of thirteen emerging non-volatile memory technologies, that had a potential to replace NAND Flash by 2020 [14]. A report, published by Wong and Salahuddin in 2015, agrees on the candidates to the universal memory technology, but did not try to predict when real devices would appear on the market [20]. Scientist, that conduct research on magnetoresistive RAM (MRAM) [1][4], spin-transfer-torque MRAM (STT-MRAM) [15][19], or phase change memory (PCM) [2][23], suggest, that we can expect it soon.

3D XPoint™ technology, announced by Intel® and Micron® in 2015 [9][10], is probably the closest to be used in production environment – first 3D XPoint™ Intel® Optane™ devices are expected in 2016 [11]. Comparison between NVMe NAND SSD (nowadays the fastest SSDs) and a prototype of 3D XPoint™ technology powered device gave promising results – the time to access a 4kB block from an application was reduced almost 7 times [6]. Unfortunately, at the mo-

ment of writing this paper, we do not have any more Intel® Optane™ performance results.

Emerging memory technologies triggered research on architectures, algorithms, and applications that would benefit from properties of the new hardware. NVMcached, key-value cache for byte-addressable NVRAM, is an example – designing the system taking into consideration the new memory type allowed to improve performance up to 2.85 times [21]. Another exemplary research on this topic could be a log-structured file system NOVA [22], or our idea for checkpointing in NVRAM using the MPI One-sided API [5].

There are also solutions based on NVRAM related with speeding up I/O operations in HPC. Two papers concern Active NVRAM – a device that, apart from memory component, includes low power CPU [13][12]. Although the proposed architecture has potential benefits, computing units are not expected in first production devices. Another set of solutions use SSD devices. Systems like S4D-Cache [7] or SLA-Cache [8] significantly improve parameters of PFS, however, we believe that differences between typical SSD and byte-addressable NVRAM require more dedicated solutions.

## 3 Proposed solution

### 3.1 NVRAM distributed cache architecture

This section is a short summary of our previous research on NVRAM distributed cache. In a typical MPI application, MPI I/O is used in order to communicate with parallel file system (PFS). Figure 1 shows a difference between the classical approach and our solution. Instead of calling an MPI I/O implementation directly, an application uses NVRAM cache routines. This process is transparent to the programmer as the cache API is the same as MPI I/O API. NVRAM cache communicates with PFS through MPI I/O, which gives an instant support for many file systems.

We assume that each node is equipped with its own NVRAM device and it participates in a distributed cache. On each node a single thread, called a cache manager, is spawned – it is responsible for creation and management of its part of the cache. When a file is opened, it is split into equally sized parts, one for each cache manager. Then, the cache manager is responsible for prefetching its whole part of file (in the file opening phase), serving read, write and sync requests from the application, flushing all of the data into PFS (in the file closing phase). The main advantages of the extension, in comparison to typical solutions, are:

- low latency, caused by serving requests as fast as possible by omitting complicated data rearrangement algorithms,
- fully decentralized management with no communication between cache managers – cache parts are assigned at the beginning, so each application process knows exactly where data is,
- minimal meta-data – no cache blocks, no *fetched* flags (all data prefetched), no *dirty* flags (all data treated as dirty).

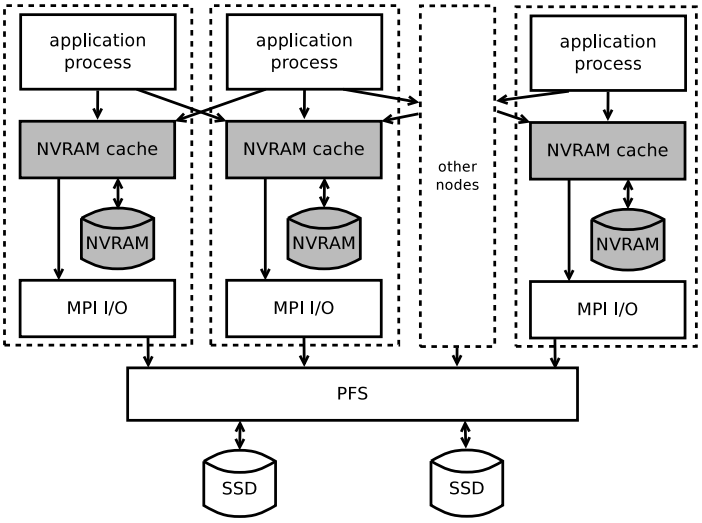


**Fig. 1.** Exemplary architecture of the solution for cluster nodes, components within a single node are inside dashed bracket. NVRAM cache layer marked with gray illustrates the difference between classical architecture and the one extended with NVRAM cache. Number of nodes in the solution is not limited.

It is clear, that a significant overhead is introduced with prefetching the whole file and flushing it back in the end. Moreover, the solution is aimed at applications that access small chunks of data (gain from byte addressing of NVRAM) from spread file locations (no drawback from omitting staging algorithms). As it was shown in previous papers [16, 17], for long running and data intensive HPC applications that operate on small data parts, our solution performs better than unmodified MPI I/O. In this paper, we want to evaluate it with an application that does not meet those criteria strictly.

Another important issue is connected with persistence of NVRAM. We have shown, that our solution could be used to prevent data from being damaged – a consistent state of the file could be recreated from the cache [17]. Although it allows to recover only from several failure types, its low overhead and ease of programming could be a solution complementary or even in some cases competitive to checkpointing.

### 3.2 Extension optimizations

The most significant performance optimization applies to serving write requests. In the first version of the extension, a cache manager responded to the process in an asynchronous way, before the exact data was written into a device. Data consistency was provided by sequential processing of the cache manager thread. Although such a strategy reduced the latency slightly when the number of requests was kept at a low level, it caused performance drop with more data-intensive

applications. In most popular MPI implementations write requests started queuing in a cache manager, that caused unpredictable and huge growth of latency for successive requests. The solution based on changing communication into a synchronous version and sending a response after all the processing was done, fixed the performance drop.

Another optimization is connected with omitting unnecessary overhead in opening and closing a file. Introduced improvements rely on better support for three `MPI_File_open` access modes:

- `MPI_MODE_CREATE`: if the file does not exist, prevent from prefetching data,
- `MPI_MODE_RDONLY`: prevent from synchronization at file closing,
- `MPI_DELETE_ON_CLOSE`: prevent from synchronization at file closing.

In special cases like treating the file as a huge distributed shared memory, most of the cache overhead related to file opening and closing is avoided.

### 3.3 Graph processing application

The graph theory has many real world applications like obtaining social network properties (e.g. Facebook, Twitter), processing maps and locations (e.g. Google Maps, GPS based navigation systems), optimizing layout of connections (e.g. designing cellular network layout) or preparing some recommendations (e.g. Netflix, Google PageRank). A great deal of algorithms hidden behind those applications are computational demanding and without further optimization they will not be able to handle increasing volume of data. In this paper we propose how to extend a selected algorithm with our MPI I/O NVRAM distribute cache. As an exemplary algorithm we have chosen the transformation, that could provide multiple graph properties, among others:

- number of paths of length n that connect two vertices,
- shortest path lengths,
- number of triangles in the graph.

Many different data structures could be used for representation of graphs. The selected problem requires checking often whether two vertices are adjacent, so a representation that minimizes complexity of this operation would be beneficial. Complexity of such query in adjacency matrix is O(1). Disadvantages of an adjacency matrix are irrelevant in the context of the selected algorithm. Slow adding or removing vertices is negligible because of constant size of a graph. Large memory consumption is unimportant since NVRAM distributed cache provides storage of size limited to the sum of all NVRAM capacities in a cluster. In our implementation, an adjacency matrix graph representation is used.

With an adjacency matrix, in order to search for walks of particular length between vertices, matrix multiplication could be used. Assuming $A$ is the adjacency matrix, in the matrix $A^n$ each element $a_{n(i,j)}$ represents the number of walks of length $n$ connecting vertex $i$ with vertex $j$. The idea is illustrated with the exemplary $A$, $A^2$ and $A^3$:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}, A^2 = A \cdot A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}, A^3 = A \cdot A \cdot A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

From the above matrices one can read for instance:

- there are 2 paths of length 2 that connect vertex 4 with vertex 3 ($a_{2(4,3)} = 2$),
- the shortest path from vertex 3 to vertex 1 is 2 (smallest $n$ where $a_{n(3,1)} > 0$ is 2),
- the number of triangles in the graph is 2 ($\frac{1}{3}\sum_{i=1}^{v} a_{3(i,i)} = 2$).

To calculate power of a matrix efficiently, in this application, communication-avoiding Cannon's algorithm is used [3].

## 4 Experiments

The MPI I/O NVRAM extension is designed for applications of specific properties. To benefit most from the extension, a data-intensive application should access small data chunks from spread locations and run long enough to compensate for the overhead for initialization and deinitialization. Implementation of an algorithm for graph adjacency matrix is an attempt of validating the extension with an application that does not strictly possess these properties. The application accesses larger data chunks at once from neighboring locations, what is especially convenient for parallel file systems we want to compete with. Within this section we want to prove, that proposed MPI I/O NVRAM distributed cache is beneficial for a wide range of applications by showing a case study of an application that does not meet our cache requirements.

### 4.1 Testbed environment

The extension was tested on two clusters: Lap06 and K2 described in detail in Tables 1 and 2. Each node in Lap06 is equipped with an NVRAM hardware simulation platform, set to pessimistic values. Lower times would result in even better results for the NVRAM version. In contrast, K2 simulates NVRAM using tmpfs but its size allows to measure scalability.

### 4.2 Performance tests

Calculating power of a matrix can have different real-world applications, including graph processing. For some of them, storing a final matrix is crucial (e.g. searching for the number of walks of a particular length), but other use it only as intermediate values (e.g. searching for the number of triangles). From the perspective of MPI I/O, if the application does not need the final matrix stored on disk, `MPI_MODE_DELETE_ON_CLOSE` could be used in order to trigger additional optimizations. For that reason, most test cases are split into groups according to *delete on close* mode (on and off).

**Table 1.** Lap06 and K2 clusters – hardware and software configuration

| | Lap06 | K2 |
|---|---|---|
| Number of computing nodes | 6 | 96 |
| Number of PFS nodes | 2 | 3 |
| CPU | 2 x Intel® Xeon® E5-4620, 2.20 GHz (2.60 GHz turbo) | 2 x Intel® Xeon® E5345, 2.33 GHz |
| RAM | 15GB | 8GB |
| Network | 40Gb/s Infiniband | 10Gb/s Ethernet |
| Storage | SSD | HDD |
| NVRAM simulation | 17GB, hardware simulation | 4GB, tmpfs |
| MPI implementation | MPICH 3.2 | |
| PFS | Orange-FS 2.8.7 | |

**Table 2.** NVRAM simulation platform parameters in Lap06 cluster

| | |
|---|---|
| Additional latency to access the data | 2000ns |
| Time required to flush data on device | 600ns |
| Memory bandwidth | 9.5GB/s |

As the execution time does not depend on graph properties other than its size, for each test case we generated $n$ nodes and connected each two nodes with the probability of 2%. All scenarios apply to low graph powers, so each value of the adjacency matrix is stored in a 1 byte cell, that results in the size of the final file equal to $n^2$ bytes.

**Performance optimizations** As stated in section 3, handling of write requests with a synchronous mode was the most noticeable improvement over the previously proposed extension. Comparison of synchronous and asynchronous mode for graph processing application is presented in Figure 2. Results show 30% reduction of application's execution time for small computed graph powers. Moreover, while increasing the power of a graph, the performance gain is greater.

**Results with *delete on close* mode off** According to Figure 3, execution time of the application grows linearly with the power. The greater the power of a matrix, the greater execution time and greater performance gain from the NVRAM distributed cache. For example, with power of 2 the proposed extension is less than 20% faster than unmodified MPI I/O, while for larger powers it is more than 40% faster. Increasing performance gain is caused by overhead for initialization and deinitialization that is independent from the power.

Figure 4, and 5 present an exponential growth of execution time for different sizes of input graphs. Plots prepared for power of 2 show that for small powers with *delete on close* mode off the proposed extension is beneficial only for small input data. The chart with results of power of 8 is an example that for higher powers the NVRAM distributed cache is superior for each input size.

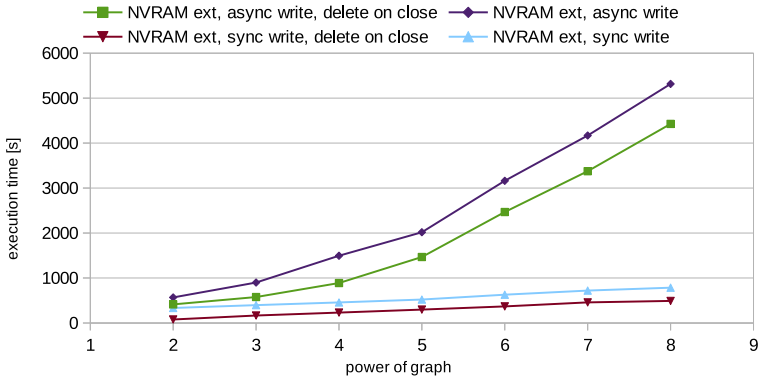


**Fig. 2.** Power of adjacency matrix, comparison between synchronous and asynchronous write. Values for fixed graph size (10 000 nodes) and different powers. Lap06 cluster.

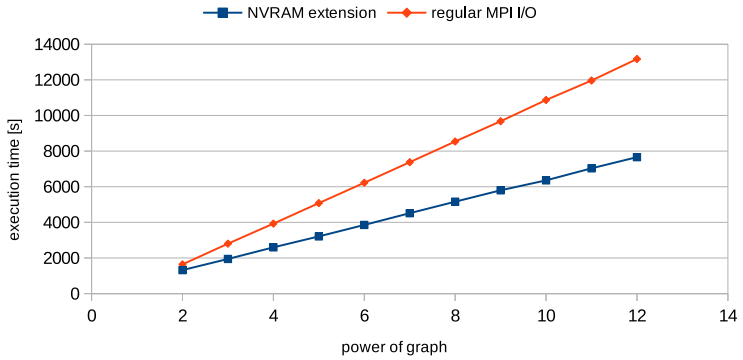


**Fig. 3.** Power of adjacency matrix, comparison between unmodified MPI I/O and proposed extension. Values for fixed graph size (20 000 nodes) and different powers. Lap06 cluster. *Delete on close* mode off.

**Results with *delete on close* mode on** The *delete on close* mode allows to omit the phase of synchronization between the NVRAM distributed cache and the parallel file system. For that reason, execution time of an application is reduced, what is especially important for smaller powers. Figure 6 shows that the proposed solution performs significantly better than the regular MPI I/O for powers starting with 2, while Figure 7 proves, that the extension gives performance gain both for small and large size of input.

**Scalability** Figure 8 illustrates a respectable scalability of the algorithm, as well as the proposed NVRAM cache. With an increasing size of a cluster environment,

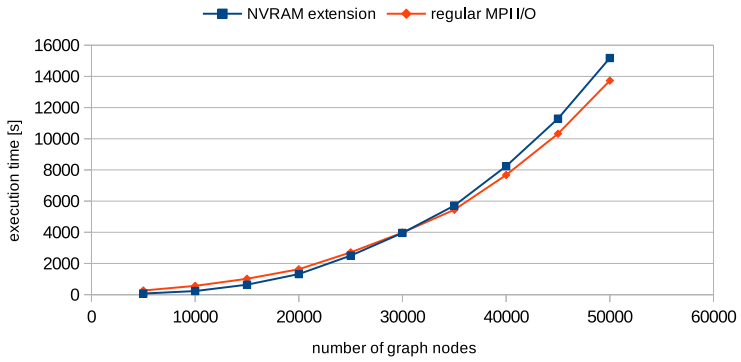


**Fig. 4.** Power of adjacency matrix, comparison between unmodified MPI I/O and proposed extension. Values for fixed power (2) and different graph sizes. Lap06 cluster. *Delete on close* mode off.

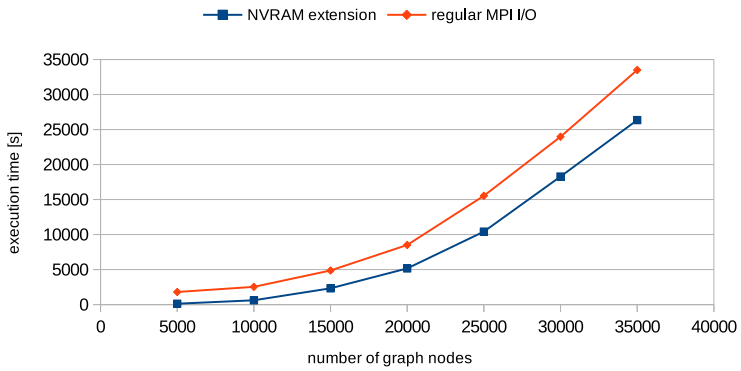


**Fig. 5.** Power of adjacency matrix, comparison between unmodified MPI I/O and proposed extension. Values for fixed power (8) and different graph sizes. Lap06 cluster. *Delete on close* mode off.

average load of a single node related to I/O operations is constant, because a higher number of I/O requests from computing nodes is handled by a higher number of cache managers. A typical PFS environment usually is not so flexible and in case of an unexpected heavy load requires hardware reconfiguration.

## 5 Summary and future work

Motivation of this research was related to expected incorporation of emerging memory technologies in production devices soon, which was justified in the intro-

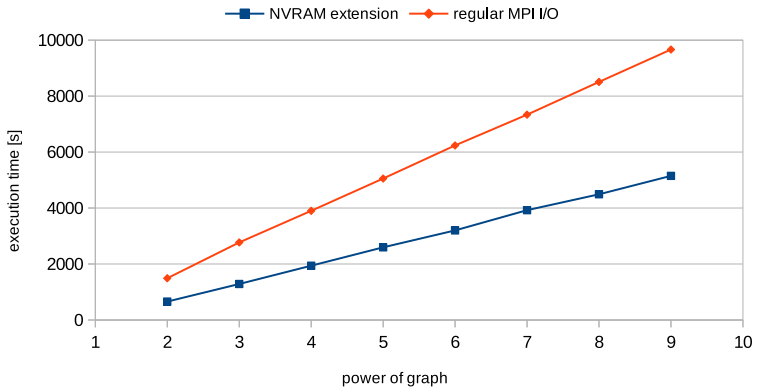


**Fig. 6.** Power of adjacency matrix, comparison between unmodified MPI I/O and proposed extension. Values for fixed graph size (20 000 nodes) and different powers. Lap06 cluster. *Delete on close* mode on.

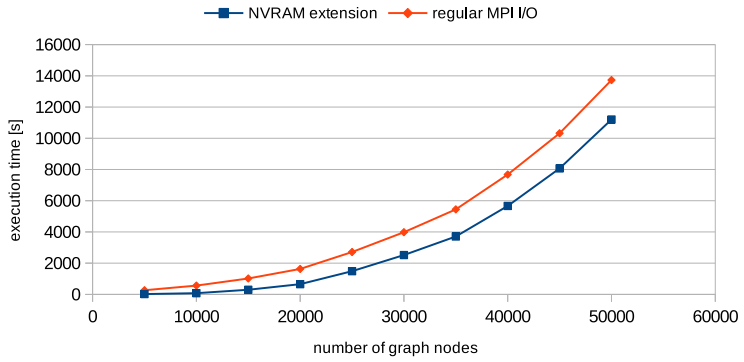


**Fig. 7.** Power of adjacency matrix, comparison between unmodified MPI I/O and proposed extension. Values for fixed power (2) and different graph sizes. Lap06 cluster. *Delete on close* mode on.

duction and related work. Within this paper we described an optimized version of MPI I/O distributed cache supported by byte-addressable NVRAM. Then we focused on its evaluation with an application that computes powers of adjacency matrix using Cannon's algorithm. Presented results of experiments prove, that with the tested application our solution performed better than regular MPI I/O.

Our future plans include further optimization of the extension and its evaluation with a wider range of applications. Moreover, we also want to focus on the mechanism, that would allow to predict the benefit from our extension without actual running of an application.

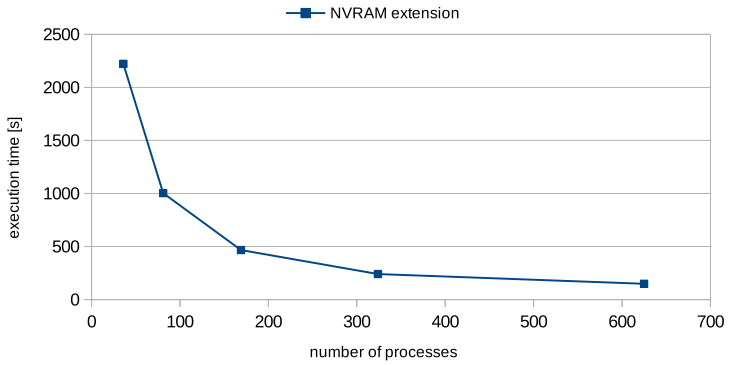


**Fig. 8.** Power of adjacency matrix, scalability. Values for fixed power (2) and fixed graph size (20 000 nodes). K2 cluster. *Delete on close* mode off.

**Acknowledgments.** The research in the paper was supported by a Grant from Intel Technology Poland.

## References

1. Åkerman, J.: Toward a Universal Memory. Science 308(5721), 508–510 (2005), http://science.sciencemag.org/content/308/5721/508
2. Alpert, A., Luo, R., Asheghi, M., Pop, E., Goodson, K.: Analytical model of graphene-enabled ultra-low power phase change memory. In: 2016 15th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm). pp. 670–674 (May 2016)
3. Cannon, L.E.: A cellular computer to implement the Kalman Filter Algorithm. Ph.D. Thesis. Montana State University (1969)
4. Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., Chen, Y.: Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In: Design Automation Conference, 2008. 45th ACM/IEEE (2008)
5. Dorożynski, P., Czarnul, P., Malinowski, A., Czuryło, K., Dorau, Ł., Maciejewski, M., Skowron, P.: Checkpointing of Parallel MPI Applications Using MPI One-sided API with Support for Byte-addressable Non-volatile RAM. Procedia Computer Science 80 (2016), international Conference on Computational Science 2016, ICCS 2016
6. Foong, A., Hady, F.: Storage As Fast As Rest of the System. In: 2016 IEEE 8th International Memory Workshop (IMW). pp. 1–4 (May 2016)
7. He, S., Sun, X.H., Feng, B.: S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems. In: Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. pp. 514–523 (June 2014)
8. He, S., Wang, Y., Sun, X.H.: Improving Performance of Parallel I/O Systems through Selective and Layout-Aware SSD Cache. IEEE Transactions on Parallel and Distributed Systems 27(10), 2940–2952 (Oct 2016)
9. Intel Corporation: Intel and Micron Produce Breakthrough Memory Technology (July 2015), http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology

10. Intel Corporation: Introducing Breakthrough Memory Technology (July 2015), http://www.intel.com/content/www/us/en/architecture-and-technology/non-volatile-memory.html
11. Intel Corporation: Introducing Intel Optane Technology Bringing 3D XPoint Memory to Storage and Memory Products (July 2015), https://newsroom.intel.com/press-kits/introducing-intel-optane-technology-bringing-3d-xpoint-memory-to-storage-and-memory-products/
12. Kannan, S., Milojicic, D., Talwar, V., Gavrilovska, A., Schwan, K., Abbasi, H.: Using Active NVRAM for Cloud I/O. In: Open Cirrus Summit (OCS), 2011 Sixth. pp. 32–36 (Oct 2011)
13. Kannan, S., Gavrilovska, A., Schwan, K., Milojicic, D., Talwar, V.: PUsing Active NVRAM for I/O Staging. In: Proceedings of the 2Nd International Workshop on Petascal Data Analytics: Challenges and Opportunities. pp. 15–22. PDAC '11, ACM, USA (2011), http://doi.acm.org/10.1145/2110205.2110209
14. Kryder, M., Kim, C.S.: After Hard Drives — What Comes Next? Magnetics, IEEE Transactions on 45(10), 3406–3413 (Oct 2009), http://dx.doi.org/10.1109/TMAG.2009.2024163
15. Makarov, A., Sverdlov, V., Selberherr, S.: Magnetic Tunnel Junctions with a Composite Free Layer: A New Concept for Future Universal Memory, pp. 93–101. John Wiley & Sons, Inc. (2013), http://dx.doi.org/10.1002/9781118678107.ch6
16. Malinowski, A., Czarnul, P., Dorożynski, P., Czuryło, K., Dorau, Ł., Maciejewski, M., Skowron, P.: 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. Annals of Computer Science and Information Systems, vol. 9, pp. 133–140. PTI (2016), http://dx.doi.org/10.15439/2016F52
17. Malinowski, A., Czarnul, P., Maciejewski, M., Skowron, P.: A Fail-Safe NVRAM Based Mechanism for Efficient Creation and Recovery of Data Copies in Parallel MPI Applications, pp. 137–147. Springer International Publishing (2017), http://dx.doi.org/10.1007/978-3-319-46586-9_11
18. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.1 (2015), http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
19. Wolf, S.A., Lu, J., Stan, M.R., Chen, E., Treger, D.M.: The Promise of Nanomagnetics and Spintronics for Future Logic and Universal Memory. Proceedings of the IEEE 98(12), 2155–2168 (Dec 2010)
20. Wong, H.S.P., Salahuddin, S.: Memory leads the way to better computing. Nature nanotechnology 10, 191–194 (2015)
21. Wu, X., Ni, F., Zhang, L., Wang, Y., Ren, Y., Hack, M., Shao, Z., Jiang, S.: NVMcached: An NVM-based Key-Value Cache. In: Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems. pp. 18:1–18:7. APSys '16, ACM, USA (2016), http://doi.acm.org/10.1145/2967360.2967374
22. Xu, J., Swanson, S.: NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In: 14th USENIX Conference on File and Storage Technologies (FAST 16). pp. 323–338. USENIX Association, Santa Clara, CA (Feb 2016)
23. Zilberberg, O., Weiss, S., Toledo, S.: Phase-change memory: An architectural perspective. ACM Comput. Surv. 45(3), 29:1–29:33 (Jul 2013), http://doi.acm.org/10.1145/2480741.2480746