Efficient Solution of Large Sparse Linear Systems in Modern Hardware

Athanasios Fevgas, Konstantis Daloukas, Panagiota Tsompanopoulou, and Panayiotis Bozanis
Department of Electrical & Computer Engineering
University of Thessaly
Volos, Greece
e-mails: {fevgas, kodalouk, yota, pbozanis}@inf.uth.gr

***Abstract*—The solution of large-scale sparse linear systems arises in numerous scientific and engineering problems. Typical examples involve study of many real world multi-physics problems and the analysis of electric power systems. The latter involve key functions such as contingency, power flow and state estimation whose analysis amounts at solving linear systems with thousands or millions of equations. As a result, efficient and accurate solution of such systems is of paramount importance. The methods for solving sparse systems are distinguished in two categories, direct and iterative. Direct methods are robust but require large amounts of memory, as the size of the problem grows. On the other hand, iterative methods provide better performance but may exhibit numerical problems. In addition, continuous advances in computer hardware and computational infrastructures imposes new challenges and opportunities. GPUs, multi-core CPUs, late memory and storage technologies (flash and phase change memories) introduce new capabilities to optimizing sparse solvers. This work presents a comprehensive study of the performance of some, state of the art, sparse direct and iterative solvers on modern computer infrastructure and aims to identify the limits of each method on different computing platforms. We evaluated two direct solvers in different hardware configurations, examining their strengths and weaknesses both in main memory (in-core) and secondary memory (out-of-core) execution in a series of representative matrices from multi-physics and electric grid problems. Also, we provide a comparison with an iterative method, utilizing a general purpose preconditioner, implemented both on a GPU and a multi-core processor. Based on the evaluation results, we observe that direct solvers can be as efficient as their iterative counterparts if proper memory optimizations are applied. In addition, we demonstrate that GPUs can be utilized as efficient computational platforms for tackling the analysis of electric power systems.**

Keywords— smart grid; GPU; solid state NVM; SSD; multi-physics; scientific computing; sparse matrixes; direct solvers; iterative solvers

I. INTRODUCTION

Efficient solution of systems of linear equations with large sparse matrices is fundamental in the computational sciences since they study real world problems that involve such computations. Simulation of multiphysics problems and modern power systems are representative examples. Operation functions of power systems like power flow, state estimation and contingency analysis involve computationally demanding sparse matrix calculations. Multiphysics and multidomain simulation software (e.g. COMSOL, ANSYS, Code_Aster and FEniCS) utilize either direct or iterative methods to solve the underlying linear systems that arise when modeling a multi-physics problem.

The majority of algorithms that are employed in the solution of large-scale linear systems fall into two categories: direct and iterative methods. Direct methods have been widely used owing to their robustness for many types of problems. However, they do not scale well with the size of the underlying problem and they require large amounts of memory as the size of the problem grows. Recently, out-of-core (OOC) algorithms that utilize secondary storage have been designed and employed to alleviate the memory requirements of direct solvers. These algorithms are designed to efficiently fetch and access data stored in secondary storage. On the other hand, iterative solvers present a better alternative for the solution of large-scale sparse linear systems as they have limited computational and memory requirements. They involve only matrix-vector and vector inner products which allow an extremely efficient scaling with the increase of the corresponding linear system.

Another feature that have to be taken into account is the degree of parallelism that a solution algorithm offers. Direct solution algorithms contain limited parallelism, thus their mapping onto contemporary parallel architectures does not provide large benefits. On the contrary, iterative solvers comprise highly parallel operations. As a result, their mapping onto parallel architectures can greatly accelerate the solution of the underlying system and make feasible the solution of very large-scale problems.

Obviously, the performance of any solution method is tightly coupled with the underlying computational platform. Thus, a vast number of specialized algorithms have been developed that are tailored to the underlying architecture in order to take better advantage of its computational capabilities. In addition, more sophisticated techniques, such as Out-Of-Core execution, have a great potential if they are employed with state-of-the-art storage hardware, such as flash storage technologies. In this case, solution of very large-scale linear systems is feasible by utilizing the secondary storage, with limited overhead as they offer orders of magnitude greater read/write performance than conventional hard disks. Based on these observations, in this paper we present a quantitative performance study of some state-of-the-art direct and iterative solvers in contemporary hardware architectures. We focus on the solution of large-scale, sparse linear systems that mainly arise from power systems simulation or challenging multiphysics/multidomain problems. The rest of the papers is organized as following: in section II a mathematical background is provided along with a short description of the

problems and challenges concern power systems and multiphysics simulations. Section III briefly describes emerging technologies that influence of are going to influence. The linear solvers were used in this study are presented in section IV. The experiments and a discussion on the results are in section V and at the end conclusions and future work are given in VI.

II. Background

A. Sparse Linear Systems

The efficient solution of linear systems of equations of the form, $Ax = b$ is fundamental for a wide range of physical problems. In several cases the coefficient matrix A arising from the solution of partial differential equations is large and sparse. The solution vector x can be conducted either with iterative or direct methods. Direct methods are robust and predictable, they do not experience numerical problems but the amount of available main memory is a common constraint in large systems [1][2]. On the other hand, iterative methods usually face numerical problems and may not even converge [2]. In addition, their performance depends on the existence of a sufficient preconditioner for the specific problem.

Most direct sparse methods rely on Gaussian elimination and factorize the coefficient matrix A to a product of a lower L and an upper U triangular matrix ($A = LU$). If A is symmetric positive definite, then the product is simplified to$A = LL^T$ (Cholesky factorization). In general, direct solvers conduct the solution of a sparse linear system in four phases: a) ordering, b) analysis and symbolic factorization, c) numerical factorization, and d) forward and backward substitution including iterative refinement. In the ordering phase a permutation of A is produced in order to reduce fill-in during pivoting down to diagonal. Symbolic factorization computes the non-zero pattern of the factors and calculates the numbers of non-zeros in each row and column as well. The non-zero pattern of the factors is utilized for building the necessary data structures for the numerical factorization and for the distribution of data and computations in parallel implementations [3]. The numerical values of the factors are computed at the numerical factorization phase. This is the most time and memory consuming stage. It can be performed using different techniques (e.g. left-looking, right-looking, multifrontal, supernodal) based to access pattern of the entries of A [4]. Finally, the solution of the system is conducted solving the triangular systems $Ly = b$ (forward elimination) and $Ux = y$ (backward substitution) using the stored factors. The factors of A are sparse but not as sparse as A itself. The additional fill-ins of the factors increases main memory requirements as the size of the problem increases. This is deteriorated due to parallelization on modern multi-core systems. To overcome this issue some direct solvers incorporate out of core algorithms that use secondary storage to retain data that do not fit in main memory.

On the other hand, iterative methods belong to the general category of relaxation methods. Starting with an initial solution guess, they provide a partial solution in each step which eventually converge to the desired solution, with a predefined accuracy level. The most widely used are the iterative methods based on Krylov subspaces. They form a basis of the sequence of successive matrix powers times the initial residual, which is called the Krylov sequence. Then, the approximations to the solution are formed by minimizing the residual over the subspace formed. Typical examples of Krylov-subspace methods are the Conjugate Gradient (CG) for SPD systems and the Generalized Minimal Residual Method (GMRES) for general systems. The main characteristic of iterative methods is their limited computational requirements, as they only comprise matrix-vector and vector-vector operations. In addition, owing to a recurrence property, they greatly reduce their memory requirements. As a result, they are ideal candidate for the solution of very large-scale linear systems. However, their convergence rate is not known beforehand depends on the condition number of the underlying system. In order to alleviate this problem, the technique of preconditioning is utilized that transforms the original system to a new one with more favorable properties and accelerates convergence rate. The most widely-used general purpose preconditioners are the one based on incomplete factors, such as Incomplete Cholesky (IC), Incomplete LU with no fill-ins (ILU0), and Incomplete LU with threshold and pivoting (ILUTP) [1].

B. Power Systems

The production and distribution of electric power is fundamental for human progress. The electric power is produced in power plants and is transported to the consumers via an extended power grid. Power grid has been kept up unchangeable over the past decades. Smart grid, is a recent effort towards to the upgrade of the power grid exploiting the advances in information and communication technologies. The modernization of power grid is tightly coupled with high performance computing (HPC). Several key operation functions of power grid like, state estimation, contingency analysis, power flow and economic dispatch impose real time computation constrains [5][6].

Power or load flow provides a steady state simulation of a power system and it is essential for contingency analysis. A power flow study computes the voltages and voltage angles in each power bus by studying numerically the electric power flow. According to [7] the 85% of computing time in a power flow study is consumed to solving sparse linear systems. Several recent works propose either direct [7] or iterative methods [8][9] utilizing up to date hardware for power flow analysis.

State estimation aims to enhance the situational awareness of a power system and is vital for its reliable operation. It provides an estimate for the power system state utilizing measurements derived by the SCADA system. However, the measured data are unreliable [10] or even exposed to malicious attacks [11]. The weighted least squares algorithm (WLS) is the most popular method for absorbing bad measurements improving grid stability and reliability. Both direct and iterative methods have been utilized by WLS for solving large sparse linear systems in each iteration of the state estimation algorithm [10].

Extensive failures in a power grid inflict significant economic and social implications and require considerable economic and human resources to be addressed [12]. Contingency analysis attempts to discover possible failures of a power system utilizing measurements from SCADA. Alarms raised by contingency analysis enable grid operators to carry out preventive and corrective control actions [13]. Each contingency

case can be considered as a power flow run [5] and many cases can run in parallel [5][13].

Power grid applications involve highly demanding computational problems derived from complex mathematical models. Thus, the efficient solution of large sparse linear systems is critical for these applications.

C. Multiphisycs

The term multiphysics concerns simulations of multiple physical phenomena that interact among them. They are of great importance for sciences and engineering as enable scientists and engineers to enhance their understanding on a physical model. From mathematics perspective real world physical phenomena are quite complicated to be studied with a single model. Therefore, they are modeled as multidomain and multiphysics problems of Partial Deferential Equations (PDEs). Domain Decomposition, Schwarz Splitting and Interface Relaxation methods have been utilized to treat such problems so far. Multiphysics simulation software involve several components like gradient optimizers, wavelets, multidimensional FFT/IFFT, sparse and dense linear solvers, etc.

The solving stage of multiphysics simulations involves the solution of sparse linear systems using either direct or iterative methods. Direct methods are able to solve any system arise from Finite Element modeling while iterative ones usually require more customization. COMSOL is a representative commercial multiphysics simulation software that exploits MUMPS, PARDISO and SPOOLES as direct solvers. MOOSE is an open source multiphysics framework that relies on PetSc toolkit. PetSc interfaces several direct (e.g. Matlab, PASTIX, MUMPS, SuperLU, SuiteSparse) and iterative solvers.

III. Modern Hardware for High Performance Computations

In the recent years, flash memory is widely utilized storage medium. Solid state disks (SSD) based on flash memories lack of mechanical and moving parts, provide low power consumption, and high random read/write performance. Increased reliability and decreased cost make them the storage medium of choice. The development of external memory algorithms for solving systems with large matrices was a popular research topic in the near past. The performance issues due to the bandwidth and latency of magnetic disks were addressed by utilizing clusters with distributed memory and high bandwidth interconnections, however at high cost. Nowadays, flash storage presents new opportunities for out-of-core computing. The performance of enterprise flash motivated authors of [14] to investigate the out-of-core sparse matrix vector multiplication (SpMV) on a small SSD test-bed cluster. In our previous work [15] we show that the performance of OOC direct solvers can be significantly benefited from flash storage compared to traditional magnetic disks.

Another emerging technology for solid state storage is Phase Change Memory (PCM). Compared to flash, PCM provides orders of magnitudes better read and write performance, better endurance and lower power consumption. It is byte addressable and does not require an erase operation before rewritten (i.e. supports in-place updates). There are two main approaches for using PCM in the memory hierarchy: the first proposes its utilization as secondary storage [16][17][18] and the other as non-volatile main memory alongside with DRAM [19][20]. Experimental results from early PCM based SSD prototypes [16][17][18] show that are already competitive to commercial enterprise level flash devices. Upcoming memory technologies like memristor (ReRam) and STT-RAM are expected to provide even better performance [21].

Emerging parallel architectures like general purpose graphics processing units (GPGPUs), coprocessors/accelerators, even high-end x86 multicore processors are of special interest for the solution of sparse linear systems. Several studies have been presented that evaluate the utilization of modern processing subsystems for sparse matrix computations [22][23]. NVidia's late, Kepler GPU architecture provides high level of thread parallelism and can achieve 1.66Tflops in a single GPU configuration (K40 model). Its architecture is based in SMX multiprocessor with 192 single precision CUDA cores and 64 double precision units per multiprocessor. Moreover it incorporates up to 12GB of memory. Intel's Xeon phi coprocessor with 1.2Tflops offers high computational performance utilizing up to 61 cores (244 hardware threads) and 16GB of memory. Intel Xeon latest multicore processors incorporate up to 18 cores and are capable of more than a half Teraflop per socket.

Upcoming hardware architectures like disaggregated server rack are going revolutionize high performance computations. Disaggregated rack architecture proposes the replacement of the traditional rack as a set of self-contained machines, to a pool of CPU, memory storage and network resources connected through a high-bandwidth and low-latency network [24][25][26]. This approach enables the dynamic construction of computing systems by allocating, each time, the required resources from these pools, depending to the workload demands [26].

IV. Expirements

A. Sparse Solver Libraries

In order to evaluate the performance of direct and iterative solvers in the solution of large-scale sparse linear systems arising from electrical grid simulations and multiphysic/multidomain problems, we utilized four typical representative and state-of-the-art direct and iterative solvers. We employed the Intel MKL Pardiso and the INRIA PASTIX as our direct solvers and the GMRES and PCG iterative solvers from the PARALUTION library [27].

PARDISO [28] is a shared memory multiprocessing parallel direct solver for large sparse symmetric and unsymmetric linear systems. Intel MKL provides a version of PARDISO with out-of-core functionality exploiting external memory to retain matrix factors. Specifically, Intel MKL 11.1.2 is employed and PARDISO is set to use a parallel (OpenMP) version of Metis [29] for ordering. PASTIX [30] is a high performance parallel direct solver for sparse linear systems. It relies on both POSIX threads (within a node) and MPI (within different nodes) for parallelization. PASTIX can, also, exploit secondary storage to preserve matrix factors, reducing the required amount of main memory for the solution of large systems. We used PASTIX version 5.1.4 along with SCOTCH (ver. 6.0.4) for ordering and OpenBLAS (ver. 0.2.14) in our experiments.

Regarding the iterative solvers, we utilized PARALUTION, a sparse linear algebra library focusing on exploring fine-grained parallelism on modern processors and accelerators including multi/many-core CPU and GPU platforms. It provides a large number of iterative solvers and various preconditioners, ranging from general-purpose to more sophisticated. We have employed the Preconditioner Conjugate Gradients (PCG) and the Generalized Minimum Residuals (GMRES) methods as the iterative solvers in our experiments, while we have chosen the ILUTP preconditioner mainly due to its robustness. The PCG method is employed for the solution of SPD systems while the GMRES method for systems where the system matrix has no special properties.

ID	NAME	TYPE	SIZE	Non-Zeros
M1	Inline_1	R, S, PD	503,712	18,660,027
M2	Aster_perf-11a	R, S, I	853,632	71,098,992
M3	Audikw_1	R, S, I	943,695	39,297,771
M4	Nice20mc	R, S, I	715,923	28,066,527
M5	Flan_1565	R, S, PD	1,564,794	114,165,372
M6	StocF_1465	R, S, PD	1,465,137	21,005,389
M7	kkt_power	R,S	2,063,494	12,771,361
M8	Atmosmodl	R, U, I	1,489,752	10,319,760
M9	StepDHC_DIRK_1	R, U	409,600	6,533,136
M10	StepDHC_DIRK_2	R, U	1,638,400	26,173,456
M11	StepDHC_DIRK_3	R, U	6,553,600	104,775,696

TABLE I. TEST MATRICES AND THEIR CHARACTERISTICS

B. Expiremental Methodology

Our evaluation focuses on the performance of each direct/iterative solver in terms of execution time and memory requirements. We aim to highlight possible strengths and weaknesses of each method (direct/iterative) and identify their performance limits in various hardware platforms. Thus, we utilize three different types of hardware: a) a flash SSD equipped WorkStation (WS1), b) a GPU enabled WorkStation (WS2) and c) a High Performance multicore Server (HPS). The SSD workstation is a DELL Precision T3500 workstation equipped with 24GB of DDR3 RAM and a 4-core Intel Xeon W3550 3.06GHz CPU. It is configured with an Intel 520 SSD 240GB (connected to SATA-III interface) as booting device and an OCZ Revodrive 350 PCIe 480GB as additional storage device for the experiments. The GPU workstation is a DELL T5500 with 24GB DDR3 RAM, a 6-core Intel Xeon E5645 2,4GHz CPU and a Tesla C2075 GPU. Last, the high performance server is a HP BL460c Gen 9 blade server with 64GB DDR4 RAM and 2 Intel Xeon CPUs E5-2695 v3 at 2.30GHz with 14 cores each. Both SSD workstation and 28-core server run 64-bit Centos 6.6 (kernel 2.6.32-504.12.2) while GPU workstation runs 64-bit Ubuntu 14.10 (kernel 3.16.0-31-generic).

Real, symmetric and unsymmetric linear systems of equations are examined. Test data M1-M8 were derived from Sparse Matrix Collection of the University of Florida [31]. Coefficient matrices M9-M11 arising from the tensor product discretization of linear parabolic multi domain problems by the Discontinuous Hermite Collocation coupled with Diagonally Implicit Runge-Kutta method [32][33]. A detailed description of the test sets is given in Table 1.

Execution time and main memory requirements were measured for each case. PASTIX and PARDISO solvers were evaluated using the SSD enabled workstation and the 28-core server. An effort to exploit Tesla GPU along with CHOLMOD direct solver from SuiteSparse package gave results only for M1 and M9 systems due to small GPU memory size. Therefore, we present results only from the two other hardware platforms (WS1, HPS). On the other hand, the iterative solvers from PARALUTION were evaluated in all hardware platforms. Regarding the OOC algorithms, their performance depends on

TABLE II. DIRECT SOLVERS RESULTS

	PASTIX						PARDISO					
	WS1 (4-CORE, SSD)				HPS (28-CORE)		WS1 (4-CORE, SSD)				HPS (28-CORE)	
MATRIX	IC exec. time (sec)	IC mem. (GB)	OOC exec. time (sec)	OOC mem. (GB)	IC exec. time (sec)	IC mem. (GB)	IC exec. time (sec)	IC mem. (GB)	OOC exec. time (sec)	OOC mem. (GB)	IC exec. time (sec)	IC mem. (GB)
M1	16.15	1.99	16.99	1.84	9.57	2.08	10.44	1.81	15.03	0.67	4.30	2.27
M2	406.31	15.50	1219.94	3.91	73.20	16.10	347.35	14.73	408.87	2.31	44.97	17.02
M3	227.76	10.40	701.62	3.91	44.60	11.00	216.82	10.56	289.11	1.51	33.11	12.60
M4	204.23	9.19	480.75	3.90	40.92	9.59	178.13	9.06	225.24	1.08	24.54	10.82
M5	172.63	13.50	667.04	3.92	40.76	13.80	149.08	12.75	202.28	2.17	23.79	14.10
M6	200.39	9.84	358.22	3.90	52.29	10.10	195.94	9.52	241.94	0.95	29.06	10.98
M7	261.87	5.61	648.03	3.91	88.14	5.99	99.49	3.82	202.20	1.30	18.55	6.09
M8	512.03	16.80	1427.50	3.95	125.06	17.10	503.39	15.40	559.38	1.22	65.00	17.06
M9	12.42	1.56	12.70	1.48	8.09	1.63	7.26	1.33	15.85	0.44	2.67	1.70
M10	69.09	7.10	167.05	3.97	36.66	7.38	42.88	6.05	88.40	1.76	11.93	7.66
M11	Not enough memory	-	Not enough memory	-	216.95	32.8	Not enough memory	-	Not enough memory	-	74.87	34.07

TABLE III. ITERATIVE SOLVERS RESULTS

	PARALUTION				
	All cases	WS1 (4-CORE, SSD)	HPS (28-core)	WS2 (GPU)	All cases
MATRIX	Iterations	Exec. time (sec)	Exec. time (sec)	Exec. time (sec)	Mem. (GB)
M1	13	0.91	5.20	0.19	1.15
M2	65	12.73	8.76	2.68	2.42
M3	17	2.67	8.75	0.44	2.2
M4	34	3.82	3.23	0.89	1.66
M5	37	10.87	8.71	2.03	3.6
M6	33	0.46	0.74	0.11	2.62
M7	77	28.81	29.89	8.66	3.23
M8	52	0.25	0.14	0.04	3.52
M9	53	16.79	11.34	1.21	1.03
M10	64	18.98	11.75	2.90	4.04
M11	162	254.63	135.96	Not enough memory	13.01

the available main memory. Thus, we restricted the maximum amount of memory that OOC algorithms can use to 4GB in order to conduct more realistic experiments.

C. Results

Table II summarizes the results for the direct solvers. The in-core execution utilizing the 28 cores of HPS server is 1.5 to 5.5 times faster than PASTIX and 2.4 to 7.7 times faster than PARDISO, compared to the in-core execution in the 4 cores of WS1. The IC runs in the 28-core platform consume slightly more main memory than the runs in the 4-core platform. On the other hand, the out-of-core execution of PASTIX is up to 3.2 times slower but requires up to 4.2 time less memory than the in-core execution of WS1. Similarly, the out-of-core solution algorithm of PARDISO is up to 2.2 times slower than the corresponding in-core but consumes up to 12.6 times less memory. PARDISO achieves better performance and has less memory requirements compared to PASTIX in all cases. Particularly, the IC execution in the 28-core HPS server is faster than PARDISO by a factor ranging from 1.3 to 4.7 and the OOC by a factor ranging from 1.1 to 3.2 respectively.

Table III presents the experimental results for the iterative solution methods. Comparing the execution time between the multi-core platforms, we can observe that in 4 cases the utilization of more cores leads to increased execution times. This can mainly be attributed to the limited parallelism found in the backward and forward substitution phases for the specific systems. For the other cases, execution on the 28-core platform results to a speedup between 1.1X and 1.9X. Iterative solvers can greatly benefit from the vast amount of computational resources found in modern GPUs. As we can observe, execution on the GPU outperforms the execution on the 4-core and the 28-core platforms by a factor ranging from 3.3 up to 13.8 and from 3.2 up to 26.1 respectively (with the most values being between 3 and 10).

Furthermore, iterative methods outperform direct ones in the majority of cases for the 4-core WS1. This is not the case for the 28-core platform, where direct methods outperform iterative algorithms in 5 cases, mainly for systems where the system matrix exhibits a low sparsity ratio. Utilizing the GPU platform allows iterative solvers to achieve the best performance in all cases that was able to run.

Another important aspect of a linear solution algorithm is its memory requirements. Direct solvers exhibit increased memory demands as the size of the linear system increases. Thus, linear system M11 was solved only on the HPS platform that comprises a large amount of main memory. Even OOC execution failed due to the 4GB limit that we have set. We increased the available amount of memory to 8GBs and PARDISO was able to solve M11 in 615sec. On the other PASTIX was not possible to provide a solution even with 20GB of RAM available to the OOC solver. It is remarkable that PARDISO is almost twice faster than PARALUTION in the 28 core execution.

V. CONCLUSIONS

The efficient solution of sparse linear systems of equations is of great importance for a wide range of scientific and engineering problems, including power systems and multi-physics simulations. In this paper, we presented a quantitative analysis of the performance of state-of-the-art direct and iterative linear system solution algorithms on modern hardware architectures, including multicore CPUs, GPU-based platforms and flash-based SSDs. Experimental evaluation on a series of representative large-scale linear systems demonstrated that iterative methods outperform direct ones in most cases and are capable to exploit GPUs' processing power more efficiently due to smaller memory requirements. Moreover, results unveiled that flash SSDs and OOC algorithms can be a better alternative and can alleviate the increased memory requirements of the in-core solution algorithms. However, the efficiency of a method

depends, in many cases, on the characteristics of the coefficient matrix itself. Thus, the apt selection of the appropriate method is needed.

Relying to the above we believe that contribution of modern hardware is capable to lead to important savings of computational time in numerical linear algebra, but further research is needed in order to exploit its full advantages. Our future work aims to further improve the efficiency of out-of-core algorithms in non-volatile memories.

ACKNOWLEDGMENT

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

REFERENCES

[1] Y. Saad, "Iterative methods for sparse linear systems", Siam, 2003.

[2] V. Rotkin, Vladimir, S. Toledo, "The design and implementation of a new out-of-core sparse Cholesky factorization method" ACM Transactions on Mathematical Software (TOMS) vol. 30, no 1, pp. 19-46, 2004.

[3] A. Gupta, "Parallel sparse direct methods: A short tutorial", IBM Research Report, 2010.

[4] J. Hogg, J. Scott, "New Parallel Sparse Direct Solvers for Multicore Architectures", *Algorithms* vol. 6, no 4, pp. 702-725, 2013.

[5] Z. Huang, J. Nieplocha, "Transforming power grid operations via high performance computing", Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, pp. 1-8, IEEE, 2008.

[6] R. Amgai, J. Shi, S. Abdelwahed, Y. Fu, "Research trends in high performance computing application on large scale power system operation", In Grand Challenges in Modeling & Simulation. 2012.

[7] T. Chagnon, J. Johnson, P. Vachranukunkiet, P. Nagvajara, C. Nwankpa, "Sparse lu decomposition using fpga", In Proc. of the 9th Int. Workshop on State-of-the-Art in Scientific and Parallel Computing, Trondeim, Norway. 2008.

[8] R. Idema, D. J. Lahaye, C. Vuik, L. van der Sluis, "Scalable Newton-Krylov solver for very large power flow problems", Power Systems, IEEE Transactions on, 27(1), pp. 390-396, 2012.

[9] X. Li, F. Li, "GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method", Electric Power Systems Research, 116, pp. 87-93, 2014.

[10] J. Neplocha, D. Chavaria-Miranda, V. Tipparaju, Z. Huang, A. Marquez, "A parallel WLS state estimator on shared memory computers", Int. Conf. in Power Engineering, IPEC 2007, pp.395-400, 2007.

[11] Y. Liu, P. Ning,P., M. K. Reiter, "False data injection attacks against state estimation in electric power grids", ACM Transactions on Information and System Security (TISSEC), vol. 14, no 1, 2011.

[12] M. Alamaniotis, G. Rong, L. H. Tsoukalas, "Towards an energy internet: A game-theoretic approach to price-directed energy utilization," In Energy-Efficient Computing and Networking, pp. 3-11. Springer Berlin Heidelberg, 2011.

[13] G. A. Ezhilarasi, K. S. Swarup, "Parallel contingency analysis in a high performance computing environment" In proc. Int. Conf. on Power Systems, 2009, ICPS'09, pp. 1-6. IEEE, 2009.

[14] Z. Zheng, E. Saule, H.M. Aktulga, C. Yang, E.G. Ng, P. Maris, J.P. Vary, U.V. Catalyurek, "An out-of-core dataflow middleware to reduce the cost of large scale iterative solvers", In Parallel Processing Workshops (ICPPW), pp. 71-80, 2012.

[15] A. Fevgas, P. Tsompanopoulou, P. Bozanis, "Exploring the Performance of Out-of-Core Linear Algebra Algorithms in Flash based Storage", 6th International Conference on Numerical Analysis (NumAn 2014), Chania, Crete, Greece, pp. 97-102, 2014. (http://lib.amcl.tuc.gr/handle/triton/36)

[16] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, S. Swanson. "Onyx: a protoype phase change memory storage array", In Proc. of the 3rd USENIX conference on Hot topics in storage and file systems, pp. 2-2, 2011.

[17] M. Athanassoulis, B. Bhattacharjee, M. Canim, A. K. Ross. "Path processing using solid state storage", In Proc. of the 3rd Int. Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS 2012), no. EPFL-CONF-180073. 2012.

[18] I. Koltsidas, P. Mueller, R. Pletka, T. Weigold, E. Eleftheriou, M. Varsamou, A. Ntalla, E. Bougioukou, A. Palli, T. Antonakopoulos, "PSS: A prototype storage subsystem based on PCM", 5th Non-Volatile Memories Workshop (NVMW 2014).

[19] S. Chen, Shimin, Q. Jin, "Persistent B+-trees in non-volatile main memory." Proceedings of the VLDB Endowment 8, no. 7, pp. 786-797, 2015.

[20] W. Hu, G. Li, J. Ni, D. Sun, K. L. Tan, "Bp-Tree: A Predictive B+-Tree for Reducing Writes on Phase Change Memory", IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 10, pp. 2368-2381, 2014.

[21] S. Swanson, A. M. Caulfield, "Refactor, reduce, recycle: Restructuring the i/o stack for the future of storage", Computer vol. 8, pp. 52-59, 2013.

[22] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. R. Bishop, "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units", SIAM Journal on Scientific Computing, vol. 36, no. 5, pp. C401-C423, 2014.

[23] E. Saule, K. Kaya, Ü. V. Çatalyürek, "Performance evaluation of sparse matrix multiplication kernels on intel xeon phi", In Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, pp. 559-570, 2014.

[24] C. C. Tu, "Memory-Based Rack Area Networking", PhD diss., Stony Brook University, 2014.

[25] C. S. Li, H. Franke, C. Parris, V. Chang, "Disaggregated Architecture for At Scale Computing", In, Emerging Software as a Service and Analytics 2015 Workshop (ESaaSA 2015), in conjunction with CLOSER 2015, Lisbon, 2015.

[26] B. Abali, R. J. Eickemeyer, H. Franke, C. S. Li, M. A. Taubenblatt, "Disaggregated and optically interconnected memory: when will it be cost effective?", arXiv preprint arXiv:1503.01416, 2015.

[27] D. Lurkarski. PARALUTION project, http://www.paralution.com

[28] O. Schenk, "Scalable parallel sparse lu factorization methods on shared memory multiprocessors", PhD Thesis, 2000.

[29] G. Karypis, V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", SIAM Journal on Scientific Computing, vol 20, no. 1, pp. 359-392, 1998.

[30] P. Hénon, P. Ramet, J. Roman, "PASTIX: a high-performance parallel direct solver for sparse symmetric positive definite systems", Parallel Computing vol. 28, no. 2, pp. 301-321, 2002..

[31] The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices (last accessed 2/25/2015).

[32] I.E. Athanasakis, E.P. Papadopoulou and Y.G. Saridakis, "Tensor Product Discontinuous Hermite Collocation for linear & nonlinear Parabolic Multidomain Problems in 2+1 dimensions", (work in progress - personal communication).

[33] I.E. Athanasakis, E.P. Papadopoulou and Y.G. Saridakis, "Discontinuous Hermite Collocation and diagonally implicit RK3 for a brain tumour invasion model." Proceedings of WCE 2013 Vol I, IAENG, pp. 241-246, 2013.