On issues concerning Cloud environments in scope of scalable multi-projection methods

Byron E. Moutafis, Christos K. Filelis-Papadopoulos, George A. Gravvanis
Department of Electrical and Computer Engineering,
School of Engineering,
Democritus University of Thrace
University Campus, Kimmeria, GR 67100, Xanthi, Greece.
{vmoutafi, cpapad, ggravvan}@ee.duth.gr

John P. Morrison
Irish Centre for Cloud Computing and Commerce,
Department of Computer Science,
University College Cork,
Cork, Ireland.
j.morrison@cs.ucc.ie

***Abstract*— Over the last decade, Cloud environments have gained significant attention by the scientific community, due to their flexibility in the allocation of resources and the various applications hosted in such environments. Recently, high performance computing applications are migrating to Cloud environments. Efficient methods are sought for solving very large sparse linear systems occurring in various scientific fields such as Computational Fluid Dynamics, N-Body simulations and Computational Finance. Herewith, the parallel multi-projection type methods are reviewed and discussions concerning the implementation issues for IaaS-type Cloud environments are given. Moreover, phenomena occurring due to the "noisy neighbor" problem, varying interconnection speeds as well as load imbalance are studied. Furthermore, the level of exposure of specialized hardware residing in modern CPUs through the different layers of software is also examined. Finally, numerical results concerning the applicability and effectiveness of multi-projection type methods in Cloud environments based on OpenStack are presented.**

Keywords—semi-coarsening, aggregation, algebraic domain decomposition, high performance computing, parallel hybrid solver, sparse and dense matrix computations, cloud computing.

I. INTRODUCTION

Cloud computing has gained significant attention over the last decade. A gradually increasing number of software products and applications are migrating to Cloud environments. Cloud computing environments have multiple advantages over classical computing techniques, such as flexibility in resource allocation, virtually unlimited storage and a unified environment offering tools that allow "on-the fly" deployment of services. There are various deployment models offered by Cloud Service Providers (CSPs) including: a) Infrastructure as a Service (IaaS), b) Platform as a Service (PaaS), c) Software as a Service (SaaS), [1]. A major problem in modern Cloud environments is over-provisioning of resources which adversely affects utilization, thus leading to increased power consumption. Management and organization of such large scale heterogeneous parallel systems further impact performance and power consumption, [1,2,3]. Recently, a Self-Organizing, Self-Management system, namely CloudLightning, [4], has been proposed to tackle the aforementioned problems. Moreover, CloudLightning aims to improve the performance, while simultaneously reducing power consumption, of High Performance Computing (HPC) applications in IaaS Environments by returning explicit control over resources back to the CSP, [4]. The CloudLightning approach introduces the notion of vRacks, which are composed of homogeneous resources and represents a partitioning of one or more physical racks. The various vRacks will be composed of different hardware types. Thus, different homogeneous vRacks form a collection of heterogeneous resources within a collection of resource known as a Cell. Cells are composed of multiple physical racks and multiple Cells comprise the entire cloud in an architecture which is akin to a Warehouse Scale Computer (WSC), [3,4]. This new system enables multilevel control through local decision making, concerning resource allocation, leading to reduced intercommunications and improving service delivery. The local decision making is performed by specialized software components that are responsible for selecting adequate resources based on parameters such as: performance, energy consumption, available hardware, etc. More information on the CloudLightning system can be found in [4].

One of the HPC use cases involved in the CloudLightning system is general sparse and dense matrix computations. This use case, among others, involves the solution of large sparse linear systems. The solution of such systems is usually performed in large HPC clusters and supercomputers, using domain decomposition techniques, [5]. Modern Cloud environments can be used for solving large sparse linear systems. The performance and efficiency of classic domain decomposition type methods for Cloud environments is extensively examined in [6,7,8]. Recently, an algebraic domain decomposition method, based on Multi-Projection, has been proposed and used for solving large sparse linear systems in large scale hybrid parallel systems. The method has been experimentally proven to be scalable up to a large number of processors for solving general sparse linear systems, [9,10].

Herewith, the Multi-Projection method, presented in detail for HPC clusters in [9,10], is reviewed and implementation details with respect to Cloud environments, based on OpenStack, are given, [11]. Moreover, the exposure of specialized hardware residing in modern CPUs, with respect to the virtualization strategy, i.e., AVX vector units is

investigated. The impact on performance due to multiple Virtual Machines running on the same resource, known as the "noisy neighbor" problem as well as other Cloud environment related phenomena, is also investigated. The effect of this performance degradation in the simulation of a Cloud environment is also discussed. Numerical results for the performance and efficiency of Multi-Projection methods for various problems are given. Furthermore, comparisons of Cloud based HPC, based on OpenStack, with a HPC cluster are discussed. The main contributions include the derivation of a linear performance degradation model with respect to observed performance degradation due to "noisy neighbor" phenomenon, as well as the exposure of specialized hardware, in modern Cloud environments. Moreover, the effect of such phenomena in general sparse and dense matrix computations is also studied, since Multi-Projection methods include various types of such computations existing in most scientific computing algorithms.

In Section 2, a brief introduction of an OpenStack-based Cloud Environment with respect to the available hardware is given. In Section 3, a review of the Multi-Projection domain decomposition type methods is given along with discussions concerning the implementation in Cloud Environments. In Section 4, the exposure of specialized hardware as well as the performance degradation due to Cloud environment related phenomena, such as the "noisy neighbor" problem, is discussed. In Section 5, numerical results obtained from experimentation on a Cloud environment with different workloads and for different types of problems are given. Moreover, comparative results against an HPC cluster are presented.

II. OpenStack Cloud Environment

OpenStack is an open-source software platform for deploying Infrastructure-as-a-Service (IaaS) type cloud computing environments. The OpenStack platform is composed of multiple core components including the following, [11]:

- Horizon: It is a web-based management and administration service used for provisioning, managing and monitoring resources.
- Heat: It is an orchestration framework for managing the lifecycle of infrastructure and applications. Moreover the Heat is based on TOSCA and provides ReST and CloudFormation based APIs.
- Nova: It is the compute service that manages virtual machine instances with respect to a chosen hypervisor such as KVM, Xen, Hyper-V, etc.
- Cinder: It is the block storage service which handles the creation and attachment of volumes to compute instances.
- Neutron: It is the software defined networking service used for tasks such as IP address management, DNS, DHCP, load balancing, etc.
- Glance: It is the image service used to provide image recovery, registration and delivery services to the Compute service.
- Swift: It is the object storage service used to handle storage and retrieval of arbitrary data in the Cloud environment.
- Keystone: It is the authentication and authorization service for the entire cloud infrastructure.

The OpenStack platform supports multiple other components such as Sahara or Manila used for data processing using clusters or shared file system services, respectively. A minimal setup for an OpenStack based Cloud environment, [11], is depicted in Fig.1. In Fig. 1, the lines represent network connections between components and ovals network interfaces as required by the Neutron networking component. Thus, the Network node requires three different network interfaces (one of them connected to an external network), the Controller node one network interface and the Compute node two network interfaces. The Controller node has the Identity, the Image service and the management part of the Compute service, as well as supporting services such as NTP, message queue and database services. Additional services can reside in the Controller node such as Object and Block storage, Telemetry, Orchestration, etc. The Network node has the networking plug-in and includes several agents used to manage networks by providing switching, routing NAT and DHCP services, [11]. The Compute node includes the hypervisor part of the Compute service, which manages virtual machines and images as well as part of the networking plug-in. It should be noted that a more complete OpenStack based cloud environment requires block and object storage services.

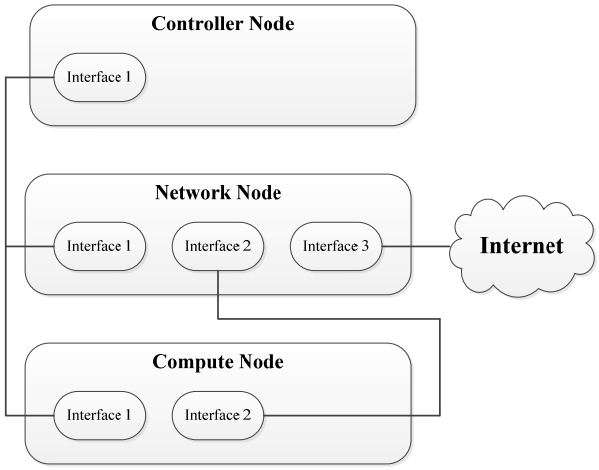

Fig. 1. Minimal service requirements for OpenStack based Cloud environment.

OpenStack supports multiple Type 1 and Type 2 hypervisors, [12]. Herewith, a Type 2 hypervisor is used, namely Kernel-based Virtual Machine (KVM), which is a hardware accelerated variant of QEMU, [13]. Type 2 or hosted hypervisors operate on top of the preconfigured Operating System (OS) abstracting guest operating systems. However, the distinction between Type 1 and Type 2 hypervisors is not always clear, for instance in the case of KVM, the host OS

operates as a Type 1 hypervisor. In the OpenStack environment, resource handling is realized through the Nova component using libvirt software library, [11]. Each Virtual Machine is spawned using the Nova component with respect to user requirements e.g., number of vCPUs, available memory, operating system, etc.

Our OpenStack deployment uses the KVM hypervisor and is running six services: Nova, Swift, Heat, Cinder and Neutron. The underlying hardware consists of 7x Dell M600; 1x Dell 2950; and 2x Dell R410 servers. It should be mentioned that in the current installation of OpenStack each vCPU is mapped to an actual CPU.

III. Multi-Projection Type Methods

The Multi-Projection methods are categorized into the class of domain decomposition methods for solving large sparse linear systems. Let us consider a sparse linear system of the following form:

$$Ax = b, \tag{1}$$

where A is the coefficient matrix, b is the right hand side vector and x is the solution vector. The linear system (1) can be solved by preconditioned Krylov subspace iterative methods, such as PGMRES(m), [14], in conjunction with Multi-Projection methods, [9,10].

A. *Multi-Projection Method (MPM)*

Let us consider the domain Ω, partitioned algebraically into ndoms non-overlapping subdomains Ω_j, j=0,…,ndoms-1, using graph-partitioning algorithms, such as Metis, [15], on the graph corresponding to the coefficient matrix A. In each subdomain Ω_j, m_j components are contained. The Multi-Projection method is based on the oblique projection of domain Ω into semi-coarse subdomains Z_j, which contain m_j fine and ndoms-1 aggregated (coarse) components. The fine components are derived from subdomain Ω_j and each of the aggregated (coarse) components corresponds to a subdomain Ω_k, with $k \neq j$. The prolongation matrices $V_j \in \mathbb{R}^{n\times(m_j+(ndoms-1))}: Z_j \to \Omega$, associate the respective semi-coarse subdomains with the domain Ω. Let us consider the i^{th} column of an (n×n) identity matrix e_i and a column vector p_j, whose elements are $1/m_j$ if the element belongs to Ω_j, otherwise, they are zero. The first m_j columns of the matrix V_j are e_i column vectors, corresponding the indices Ω_j and the remaining (ndoms-1) columns are p_i column vectors that correspond to every remaining subdomain Ω_i. In Fig. 2, the domain Ω, discretized with h=1/3 and partitioned into 4 subdomains Ω_j, and the semi-coarse subdomain Z_0, are depicted. A linear system of the following form should be solved in each subdomain Z_j:

$$V_j^T A V_j x_j = V_j^T b = b_j, \tag{2}$$

where the solution vector x_j has two parts, x_F and x_C, which are associated with fine components and aggregated (coarse) components, respectively.

The x_F is used for updating the components of the solution vector x corresponding to a subdomain Ω_j. Let us define the matrix $W_j : Z_j \to \Omega$ that maps x_F to the respective fine components of domain Ω and discards the auxiliary components x_C. The i^{th} column of W_j is an e_i column vector, corresponding the indices Ω_j, if i belongs to Ω_j, otherwise it is a zero column vector. The product of the MPM preconditioner by a vector can be described by the following algorithmic procedure, [9]:

For j=0,…,ndoms-1

$$\text{Compute } y_j = W_j W_j^T V_j A_j^{-1} V_j^T x \tag{3}$$

End For

$$y = \sum_{j=0}^{ndoms-1} y_j \tag{4}$$

The MPM is inherently parallel and thus suitable for distributed-memory parallel systems.

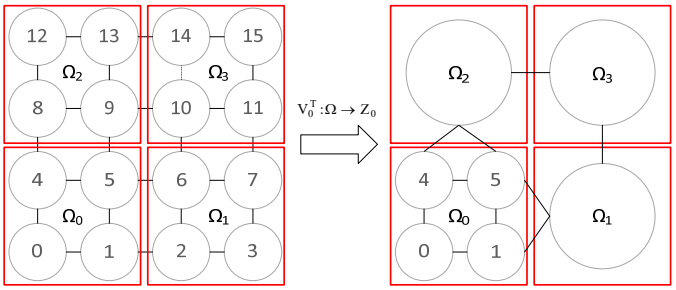

Fig. 2. Domain Ω and subdomain Z_0

B. *Multi-Projection Method with Subspace Compression (MPMSC)*

The MPMSC method is a variant of MPM that utilizes a subspace compression technique to decrease the memory requirements, [10]. Moreover, in the MPM method the ndoms-1 should be less than m_j, otherwise the local linear systems have more aggregated (coarse) components than fine components. The aggregated (coarse) components of a subdomain Z_j are re-aggregated together to reduce the number of aggregates (naggs) and therefore the size of the local linear system is decreased. Consider the respective compressed subdomains $\tilde{Z}_j$ and their prolongation matrices $\tilde{V}_j \in \mathbb{R}^{n\times(m_j+naggs)}: \tilde{Z}_j \to \Omega$. The local linear systems are of the following form:

$$\tilde{V}_j^T A \tilde{V}_j \tilde{x}_j = \tilde{V}_j^T b = \tilde{b}_j, \tag{5}$$

and the product of the MPMSC preconditioner by a vector is given by the following algorithmic procedure, [10]:

For j=0,…,ndoms-1

$$\text{Compute } y_j = W_j W_j^T \tilde{V}_j A_j^{-1} \tilde{V}_j^T x \tag{6}$$

End For

$$y = \sum_{j=0}^{ndoms-1} y_j \quad (7)$$

Further details concerning the subspace compression technique are given in [10].

C. Implementation Details

Multi-Projection-type Methods are used as a preconditioner in the Parallel Preconditioned restarted Generalized Minimum Residual Method, namely (PPGMRES(m)), [16,17]. The algorithm of PPGMRES(m) along with discussion concerning implementation issues are given in [9]. The IaaS cloud computing model provides virtual machines, where each of them has a number of vCPUs (virtual CPUs). Therefore, the hybrid parallel implementation of PPGMRES(m) in conjunction with Multi-Projection-type Methods is suitable for the cloud environment. Every subdomain of the Multi-Projection-type Methods is mapped to a virtual machine. The MPM and MPMSC are designed to require only minimum internode communication, since its subdomain is handled separately. The local linear system of each subdomain is solved in parallel by a direct method. Both PARDISO, [18,19,20], and a variant of sparse LU, [21], have been used. It should be noted that the factorization phase of the direct method is included in the pre-processing step.

IV. Impact on Peformance and Simulation

The performance of HPC applications in Cloud environments is affected by phenomena such as the "noisy neighbor" phenomenon, the heterogeneity of hardware or the varying network speeds due to software defined networking or bottlenecks.

A. Related work on Cloud related phenomena

The "noisy neighbor" phenomenon is caused by the simultaneous execution of applications by different users in the same physical resource. The three major factors that affect performance in modern NUMA architecture multicore systems are: a) remote access penalty, b) shared resource contention and c) cross-chip data sharing overhead. The shared last-level cache (LLC) is also a major factor that affects performance, [22,23]. Moreover, as the number of applications that run simultaneously on a physical resource increases, the more the performance decreases, since the available cache per application is reduced, [23]. With the aforementioned in mind, it can be seen that the execution of applications in Cloud environments leads to varying results from the aspect of performance, especially in the case of multiple virtual machines running in the same physical resource, [22,23]. Multiple solutions have been proposed to tackle the problems of scheduling virtual processors, including real time hypervisors, [22], and weighted scheduling strategies, [23]. Moreover, in [23] performance degradation in examined for various types of applications. Various researchers have proposed different solutions to the problem of scheduling, especially in scope of real time systems, [24,25,26]. Adaptive reservation based approaches have been also proposed, [27].

The heterogeneity of underlying hardware is a major concern in Cloud Environments, since it affects the symmetry of computations, especially in parallel distributed memory applications. High Performance Computing applications heavily rely on symmetry of the underlying hardware. In order to ensure symmetry of computations the CloudLightning system allocates resources belonging to homogeneous vRacks. Moreover, the allocation of resources can be performed with locality criteria in order to avoid large variations in network speeds, [4].

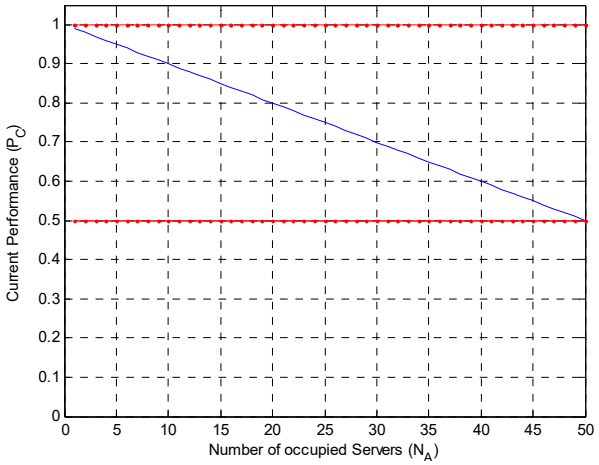

Fig. 3. Performance degradation in a vRack with N_S=50, P_O=1.0, P_W=0.5.

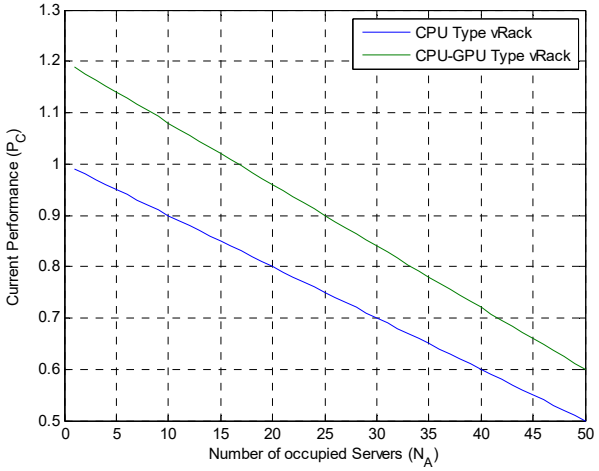

Fig. 4. Performance degradation for two vRacks composed of different types of servers.

B. Linear performance degradation model for the CloudLightning system

The CloudLightning system involves three major use cases: a) Oil and Gas Exploration, b) Ray-Tracing and c) Genomics. These use cases are composed of general sparse and dense matrix computations and have irregular memory access patterns. Multi-Projection type methods involve general sparse and dense matrix computations, thus model the three major use cases involved in CloudLightning. These types of computations are expected to suffer degradation ranging from 100% to 10%, [23]. The simulation of a Cloud environment, such as

CloudLightning, [4], should take into account the performance degradation occurring due to the aforementioned issues. Let us consider a vRack, composed of N_S (homogeneous) servers and N_A the number of servers occupied. Then a linear model can be used to adjust the performance of underlying simulated hardware:

$$P_C = P_O - \frac{N_A}{N_S}(P_O - P_W) \quad (8)$$

where $P_C \in [P_W, P_O] \subset \mathbb{R}$, $P_W \in (0,\infty) \subset \mathbb{R}$, $P_O \in (P_W,\infty) \subset \mathbb{R}$ with $P_O > P_W$ are the current performance index, the worst performance index and the optimal performance index, respectively. It should be noted that P_C, P_O and P_W are dimensionless numbers and can be computed as the ratio of performance metrics, such as GFLOPS, obtained in baseline system and another system, with respect to a certain procedure (or collection of procedures). In example, if the matrix multiplication algorithm is computed 20% faster on a CPU-GPU system compared to a baseline CPU system then its P_O=1.2. Equivalently, P_C can be defined as the ratio of performance metrics obtained from the same CPU-GPU system in presence of performance degrading phenomena, such as "noisy neighbor", using as denominator the same baseline metrics as before. Thus, this linear performance degradation model, based on differential performance indices, can be used to express the expected extension of required time to complete a given task when there are other users that simultaneously utilize the same physical resources. From Equation (8) it can be seen that current performance index is analogous to the number of servers N_A that are already occupied in the vRack. An example of performance degradation in a vRack with N_S=50, P_O=1.0, P_W=5 is presented schematically in Fig. 3.

It should be noted that the linear model of Equation (8) can be used to model the performance degradation of more complicated types of resources grouped in vRacks i.e., CPU-GPU pairs. Moreover, vRacks that group different types of servers can be modelled simultaneously i.e., let us consider two vRacks composed of 50 servers each. vRack 1 has CPU type servers and vRack 2 has CPU-GPU type servers. Let as also consider an application that has implementations for CPU type servers and CPU-GPU type servers, where the CPU-GPU type implementation is 20% faster. Furthermore, the worst case performance is 0.5 and 0.6, for CPU type servers and CPU-GPU type servers, respectively, for the aforementioned application. Then the linear model is of the following form:

$$P_C^{vRack}(N_A) = \begin{cases} 1.0 - 0.5\frac{N_A}{50}, vRack = 1 \\ 1.2 - 0.6\frac{N_A}{50}, vRack = 2 \end{cases}. \quad (9)$$

The performance degradation model (9) is schematically represented in Fig. 4. Analogously, linear performance degradation model (9) can be extended to multiple types of hardware with respect to a baseline implementation with optimal performance index set to P_O=1.

Another important issue arising in Cloud environments is the exposure of specialized hardware i.e., vector units that reside in modern CPUs, available to the user through the virtual machines. These vector units are known to enhance performance of computations. This type of specialized hardware and other types of specialized hardware such as GPUs or SSD disks have to be explicitly prescribed to the scheduler of the Nova component in order to allocate sufficient resources, [11]. In this example, the Nova component should be configured to allow either "host-pass through" or "host-model" in order for specialized hardware residing in CPUs, such as vector units, to be visible to the user, [11]. The "host-pass through" property enables full visibility of the underlying hardware which favors performance, but hinders portability in migrating a VM to other hardware. The "host-model" property reveals the model of the underlying processor (with respect to processor family), thus enabling improved portability in VM migration, in the expense of slightly reduced performance. Herewith, the OpenStack Nova component is configured with "host-model" property to ensure portability during the VM migration.

In the following Section, numerical results depicting the applicability and effectiveness of Multi-Projection type methods obtained are given.

V. Numerical Results

In order to assess the performance of the OpenStack based Cloud environment various model problems were solved using Multi-Projection type methods. The first model problem is the 3D Poisson equation in three space variables, subject to Dirichlet boundary conditions. The Poisson equation was discretized with the 7-point stencil. The right hand side of the linear systems was computed as the product of coefficient matrix A with the solution vector set to $[0,1,...,n-1]^T$, where n denotes the order of the linear system. Two additional model problems, obtained from the Florida University sparse matrix collection, were considered, [28]:

- apache2 with size n=715176 and number of nonzero elements equal to 4817870,
- af_shell3 with size n=504855 and number of nonzero elements equal to 17562051.

The right hand side vector b for the above model problems was computed as the product of coefficient matrix A with the solution vector set to $[0,1,...,n-1]^T$. The restart parameter of PPGMRES(m) was set to 20 and the maximum iterations was set to 500 iterations. The termination criterion for the Multi-Projection methods, for all experiments, was $\|r\|_2 < 10^{-8}\|b\|_2$, where r is the residual vector, b is the right hand side vector and $\|.\|_2$ is the 2-norm. The execution time is given in "seconds. hundreds (ss.hh)". A cluster was deployed in the OpenStack based Cloud environment consisting of 4 VMs, with 2 vCPUs each, running CENTOS 7. It should be noted that two different implementations of the MPM and MPMSC method where used for assessing performance. The first Cloud based implementation denoted hereafter as (MKL) is based on Intel MKL and PARDISO, [18,19,20,29]. The second Cloud based implementation denoted hereafter as (BLAS) was based on BLAS and a variant of sparse LU, [21,30]. For assessing the

scalability of the MPM and MPMSC methods additional results were obtained using a BlueGene/P (BG/P) supercomputer with the following specifications: (CPU: 1024x Quad-Core PowerPC-450 850Mhz, RAM: 4GB/node, inter-connect: 3D-Torus network with bandwidth 5.1 GBps). It should be noted that "Subds" denote the number of subdomains and "Aggs" denote the number of aggregates.

In Table 1, the performance and convergence behavior of the MPM and MPMSC methods for various numbers of processors (BG/P) and for the 3D Poisson problem with n=125000 are given. In Table 2, the performance and convergence behavior of the MPM and MPMSC methods for various numbers of processors (BLAS) and for the 3D Poisson problem with n=125000 are given. In Fig 5, the strong scalability of the MPM and MPMSC methods for different implementations and for the 3D Poisson problem with n=125000 is given. From Tables 1, 2 and Fig. 5 can be seen that the strong scalability of the MPM and MPMSC methods is better in the BG/P system than the Cloud system, [6,7,8]. In Fig. 6, the strong scalability of the MPM and MPMSC methods for different compatibility modes of Intel MKL for the 3D Poisson problem with n=512000 is presented. From Fig. 6, it can be observed that the use of AVX units greatly improves the performance for both methods. In Fig 7, the weak scalability of MPM and MPMSC methods (MKL) for the 3D Poisson problem with 64000 unknowns per node, is given.

TABLE I. PERFORMANCE AND CONVERGENCE BEHAVIOR OF THE MPM AND MPMSC METHODS FOR VARIOUS NUMBERS OF PROCESSORS (BG/P) FOR THE 3D POISSON PROBLEM WITH N=125000.

Method	Nodes	Cores	Subds	Aggs	Iter.	Performance
MPM	2	4	16	15	2(11)	2276.8078
	3	6	24	23	2(10)	996.7118
	4	8	32	31	2(10)	505.7879
MPMSC	2	4	16	3	3(8)	992.2571
	3	6	24	6	3(10)	586.7975
	4	8	32	6	3(12)	233.6427

TABLE II. PERFORMANCE AND CONVERGENCE BEHAVIOR OF THE MPM AND MPMSC METHODS FOR VARIOUS NUMBERS OF PROCESSORS (BLAS) FOR THE 3D POISSON PROBLEM WITH N=125000.

Method	Nodes	Cores	Subds	Aggs	Iter.	Total
MPM	2	4	16	15	2(11)	179.5582
	3	6	24	23	2(10)	96.4939
	4	8	32	31	2(10)	59.2548
MPMSC	2	4	16	3	3(8)	84.2075
	3	6	24	6	3(10)	72.2509
	4	8	32	6	3(12)	28.6298

From Fig. 7, it can be seen that the weak scalability is not constant with the increase of cores, since the underlying hardware is heterogeneous in nature. Moreover, the weak scalability of the schemes is affected by the interconnection speeds. In Tables 3 and 4, the performance and convergence behavior of the MPM and MPMSC methods, respectively, for various number of cores and for different problems are presented. In Table 5, the performance, convergence behavior and performance degradation of the MPM and MPMSC methods for single processor executions (MKL) and various problems are given.

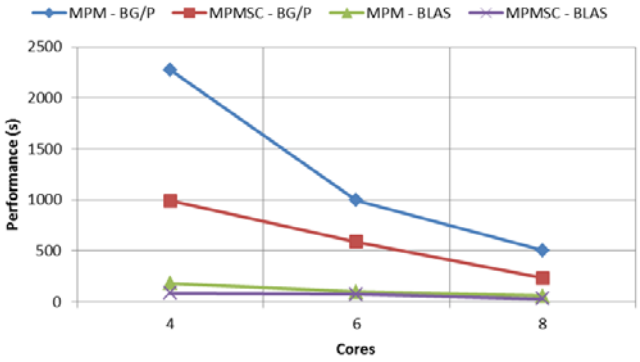

Fig. 5. Strong scalability of the MPM and MPMSC methods for different implementations for the 3D Poisson problem.

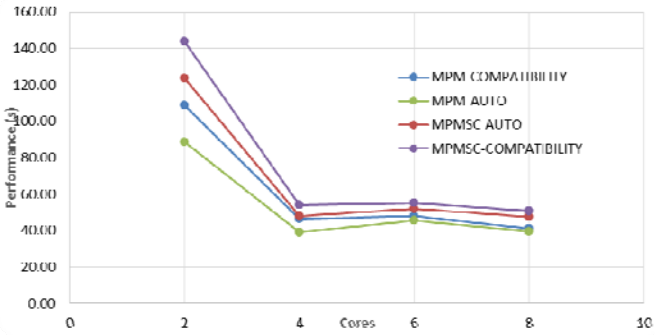

Fig. 6. Strong scalability of the MPM and MPMSC methods for different compatibility modes of Intel MKL for the 3D Poisson problem with n=512000.

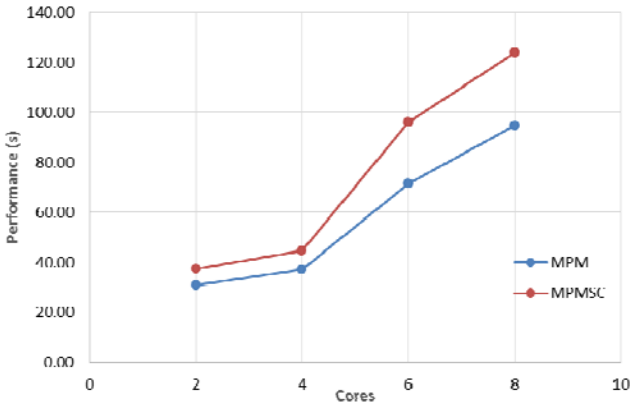

Fig. 7. Weak scalability of the MPM and MPMSC methods (MKL) for the 3D Poisson problem with 64000 unknowns per node.

It should be mentioned that the additional workload to the available resources in the OpenStack based Cloud environment is created using the Prime95 Stress testing

software, [31]. Moreover, the latest version of Prime95 has AVX2 support. The Prime95 was run in Blend mode (pre-set mode) stressing both available CPUs and memory with variable size FFT.

TABLE III. PERFORMANCE AND CONVERGENCE BEHAVIOR OF THE MPM METHOD FOR VARIOUS NUMBERS OF PROCESSORS (MKL) FOR VARIOUS PROBLEMS.

Problem	Nodes	Cores	Subds	Aggs	Iter	Performance
apache2	1	1	4	3	4(18)	236.5840
	1	2	8	7	7(12)	187.4231
	2	2	8	7	7(12)	144.0143
	2	4	16	15	7(18)	87.9084
	3	3	12	11	8(12)	188.9736
	3	6	24	23	6(19)	102.7597
	4	4	16	15	7(18)	151.9132
	4	8	32	31	7(18)	107.6127
af_shell3	1	1	4	3	2(7)	103.0759
	1	2	8	7	2(17)	65.7306
	2	2	8	7	2(17)	51.6204
	2	4	16	15	3(16)	42.0841
	3	3	12	11	3(10)	73.8761
	3	6	24	23	3(19)	52.9229
	4	4	16	15	3(16)	68.0442
	4	8	32	31	4(6)	54.2996

TABLE IV. PERFORMANCE AND CONVERGENCE BEHAVIOR OF THE MPMSC METHOD FOR VARIOUS NUMBERS OF PROCESSORS (MKL) FOR VARIOUS PROBLEMS.

Problem	Nodes	Cores	Subds	Aggs	Iter	Performance
apache2	1	1	4	2	13(4)	551.2392
	1	2	8	3	18(12)	428.7588
	2	2	8	3	18(12)	327.5805
	2	4	16	5	17(3)	198.6106
	3	3	12	5	18(20)	410.0613
	3	6	24	7	25(14)	362.4505
	4	4	16	5	17(3)	318.6274
	4	8	32	10	21(15)	280.1670
af_shell3	1	1	4	1	0(11)	37.5874
	1	2	8	4	3(9)	78.2034
	2	2	8	4	3(9)	60.7310
	2	4	16	5	3(8)	36.6561
	3	3	12	5	2(15)	59.3741
	3	6	24	7	2(13)	38.2436
	4	4	16	5	3(8)	61.6374
	4	8	32	11	3(10)	43.6878

The Blend mode automatically allocates required memory and uses in-place as well as small FFTs that stress cache memory and FPU units, [31].

TABLE V. PERFORMANCE, CONVERGENCE BEHAVIOR AND PERFORMANCE DEGRADATION OF THE MPM AND MPMSC METHODS FOR SINGLE PROCESSOR EXECUTIONS (MKL) AND FOR VARIOUS MODES AND VARIOUS PROBLEMS.

Method	Wload	Mode	Problem	Subds/ Aggs - Iter	Perf.	Degr. (%)
MPM	NO	A	3D Poisson n=512000	4/3 - 2(14)	239.9149	-
	YES				266.6759	11.15
	NO	C			188.0013	-
	YES				209.4785	11.42
MPMSC	NO	A	3D Poisson n=512000	4/1 - 3(10)	215.7437	-
	YES				229.8989	6.56
	NO	C			265.9169	-
	YES				298.5315	12.26
MPM	NO	A	apache2 n=715176	4/3 - 4(8)	236.5840	-
	YES				242.0023	2.29
MPMSC	NO	A	apache2 n=715176	4/2 - 13(4)	551.2392	-
	YES				567.1074	2.88
					Average Degradation	***7.76***

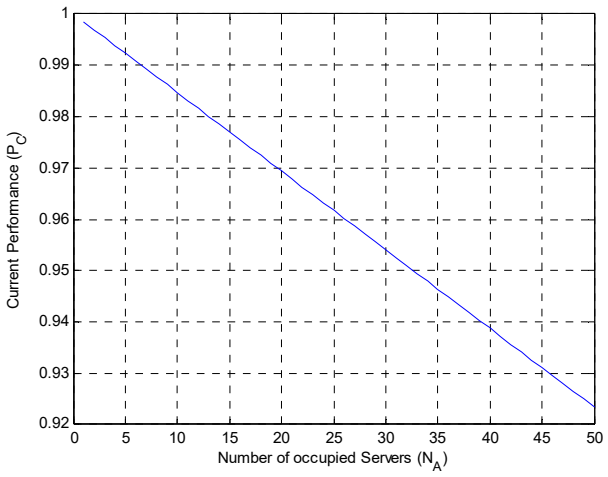

Fig. 8. Performance degradation in a vRack with N_S=50, P_O=1.0, P_W=0.9234.

The experiments presented in Table 5, have an average degradation in performance 7.76% when the Cloud environment is in maximum load. Thus, a linear model can be derived using equation (8): $P_C = 1-(1-0.9234)(N_A/N_S)$. The linear performance degradation model of a vRack composed of the hardware presented in Section 2, based on the obtained experimental results, is schematically represented in Fig. 8. Cloud environments offer the flexibility of dynamically allocating resources of variable capacity with respect to vCPU

number or available memory, however the scalability, compared to dedicated HPC clusters, is limited and the performance is affected by other users simultaneously using the same physical resources, as presented in Fig. 5,6 and Table 5.

VI. Conclusion

The presented methods have been shown to be scalable in Cloud environments for solving various types of problems. Moreover, the use of vector units substantially improves performance, thus the "host-model" or "host-passthrough" property should be used in the configuration of the Nova component, especially for HPC applications in an OpenStack based Cloud environment. Furthermore, a linear performance degradation model has been derived based on experimental results obtained from the solution of various problems with and without additional workload in the OpenStack based environment. The linear performance degradation model can be used for large scale simulation of Cloud environments, based on the CloudLightning system, [4]. Future work will concentrate in the augmentation of the linear performance degradation model with the impact of intercommunications as well as its refinement based on results obtained in a CloudLightning based Cloud environment.

Acknowledgment

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through the CloudLightning project (http://www.cloudlightning.eu) under Grant Agreement Number 643946.

The authors would like to express their thanks to Professor Dana Petcu, Department of Computer Science, West University of Timisoara (UVT), for the provision of computational facilities through the UVT HPC Center.

References

[1] D.C. Marinescu, A. Paya, J.P. Morrison and P. Healy, "Distributed Hierarchical Control versus an Economic Model for Cloud Resource Management," arXiv: 1503.01061v4 [cs.DC], 2015.

[2] A. Barroso and U. Hozle, "The case for energy-proportional computing,". IEEE Computer, vol. 40, issue 12, pp. 33-37, 2007.

[3] A. Barroso, J. Clidaras and U. Hozle, "The Data-center as a Computer; an Introduction to the Design of Warehouse-Scale Machines," 2nd ed., Morgan & Claypool, 2013.

[4] T. Lynn, H. Xiong, D. Dong, B. Momani, G. Gravvanis, C. Filelis-Papadopoulos, A. Elster, M.M. Khan, D. Tzovaras, K. Giannoutakis, D. Petcu, M. Neagul, I. Dragon, P. Kuppudayar, S. Natarajan, M. McGrath, G. Gaydadjiev, T. Becker, A. Gourinovitch, D. Kenny, J. Morrison, "CLOUDLIGHTNING: A Framework for a Self-Organising and Self-Managing Heterogeneous Cloud", in Proceedingsof the 6th International Conference on Cloud Computing and Services Science (CLOSER), Rome, Italy, Volume 1, pp. 333-338, 2016.

[5] T.F. Chan and T.P. Mathew, "Domain decomposition algorithms," Acta Numerica, vol. 3, pp. 61-143, 1994.

[6] S-N. Srirama, O. Batrashev, P. Jakovits and E. Vainikko, "Scalability of parallel scientific applications on the cloud", Scientific Programming, vol. 19, pp. 91-105, 2011.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman and J. Good, "The cost of doing science on the cloud: the montage example," in International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008, Austin, TX, USA, IEEE CS Press, pp. 1-12, 2009.

[8] L. Ismail and R. Barua, "Implementation and performance evaluation of a distributed conjugate gradient method in a cloud computing environment," Software-Practive and Experience, pp. 1-27, 2010.

[9] B. Moutafis, C.K. Filelis-Papadopoulos and G.A. Gravvanis, "Multi-Projection preconditioned methods based on semi-aggregation techniques," submitted.

[10] B. Moutafis, C.K. Filelis-Papadopoulos and G.A. Gravvanis, "Scalability of Multi-Projection preconditioned methods based on subspace compression," submitted.

[11] OpenStack, "Open source cloud computing software," https://www.openstack.org/, June 2016.

[12] G.J. Popek and R.P. Goldberg, "Formal requirements for virtualizable third generation architectures," Communications of the ACM, vol. 17, issue 7, pp. 412-421, 1974.

[13] QEMU, "Open source processor emulator," http://www.qemu.org, June 2016.

[14] Y. Saad, "Iterative methods for sparse linear systems". SIAM, 2003.

[15] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," SIAM J. Sci. Comput., vol. 20, issue 1, pp. 359-392, 1998.

[16] Y. Saad and M.H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," SIAM J. Sci. Comput. vol. 7, issue 3, pp. 856-869, 1986.

[17] C.C. Douglas, G. Haase and U. Langer, "A tutorial on elliptic PDE solvers and their parallelization," SIAM, 2003.

[18] O. Schenk and K. Gärtner, "On fast factorization pivoting methods for sparse symmetric indefinite systems," Electronic Transactions on Numerical Analysis vol. 23, issue 1, pp. 158-179, 2006.

[19] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO". Future Generation Computer Systems vol. 20, issue 3, pp. 475-487, 2004.

[20] O. Schenk and K. Gärtner and W. Fichtner, "Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors," BIT, vol. 40, issue 1, pp. 158-176, 2000.

[21] T.A.Davis, "Direct methods for sparse linear systems,". SIAM, 2006.

[22] S. Xi, C. Lu, C.D. Gill, M. Xu, L.T.X. Phan, I. Lee and O. Sokolsky, "RT-OpenStack: cpu resource management for real-time cloud computing," in IEEE 8th International Conference on Cloud Computing, 2015, New York City, NY, IEEE CS press, pp. 179-186, 2015.

[23] J. Rao, K. Wang, X. Zhou and C-Z. Xu, "Optimizing virtual machine in NUMA multicore systems," in IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013), 2013, Shenzen, IEEE CS press, pp. 306-317, 2013.

[24] Z. Deng and J.W-S. Liu, "Scheduling real-time applications in an open environment," in IEEE 18th Real-Time Systems Symposium, 1997.

[25] I. Shin and I. Lee, "Compositional real-time scheduling framework," in IEEE 25th International Real-Time Systems Symposium, 2004.

[26] H. Leontyev and J.H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," Real-Time Systems, 2009.

[27] S. Groesbrink, L. Almeida, M de Sousa and S.M. Petters, "Towards certifiable adaptive reservations for hypervisor-based virtualizations," in IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014, Berlin, IEEE CS Press, pp. 13-24, 2014.

[28] T.A. Davis and Y. Hu, "The University of Florida sparse matrix collection," ACM Transactions on Mathematical Software, vol. 38, issue 1, number 1, pp. 1-25, 2011.

[29] Intel MKL, "Intel Math Kernel Library", https://software.intel.com/en-us/intel-mkl, June 2016.

[30] C.L. Lawson, R.J. Hanson, D.R. Kincaid and F.T. Krogh, "Basic linear algebra subprograms for Fortran usage," ACM Transactions on Mathematical Software (TOMS), vol. 5, issue 3, pp. 308-323, 1979.

Prime95, "Great Internet Mersenne Prime Search," http://www.mersenne.org/, June 2016.