MATRIX-VECTOR MULTIPLICATION ALGORITHM BEHAVIOR IN THE CLOUD

Sasko Ristov, Marjan Gusev and Goran Velkoski

Ss. Cyril and Methodius University, Faculty of Information Sciences and Computer Engineering
Skopje, Macedonia
sashko.ristov@finki.ukim.mk, marjan.gushev@finki.ukim.mk, velkoski.goran@gmail.com

Abstract

Cache memory speeds up the memory access for memory demanding and cache intensive algorithms. Introducing different cache levels and greater cache sizes in modern multiprocessor architectures reduces cache misses. Both matrix-matrix and matrix-vector multiplications are computation and cache intensive algorithms, as well as memory demanding algorithms and their execution time directly depends on CPU cache architecture and organization. This paper focuses on matrix-vector multiplication performance while executed on modern shared memory multiprocessor with shared last level L3 cache and dedicated L1 and L2 cache for each CPU core.

Our goal is to analyze dense matrix-vector algorithm behavior and to check the hypothesis if the performance of today's virtualized servers organized either in private data-centers or in cloud computing is usually worse than traditional servers with host operating systems without using virtualization. Both sequential and parallel executions are implemented in traditional and cloud environments using the same platform and hardware infrastructure for each test case.

Keywords - Cloud Computing, HPC, Performance, Speedup

1 INTRODUCTION

Cloud computing paradigm emerges not only as the best service in ICT, but also as the only one. Gartner forecasts that public cloud services market growth will be 19.6%, or total $109 billion worldwide in 2012 [1]. Its pay-per-use billing model and allows huge amount of theoretically infinite computing and storage on-demand resources organized in virtual machine instances.

Migration onto the cloud has several challenges both for cloud service providers and customers. Cloud service providers want to decrease cost using less active hardware resources in order to decrease electrical power and thus group active virtual machine instances in less physical servers. The authors in [6] determine that reserve capacity model significantly outperforms all other models to dimensioning the capacity of cloud-based system for time-varying customer demand. Nevertheless, adding more server nodes can considerably underutilize the resources and improve the performance implementing more parallelism, which is desired for cloud service customers [5]. Moreover, the same virtual machine instance will not achieve the same performance on the same hardware at different times among the other active virtual machine instances [7].

The most common public cloud service providers' price is usually linearly proportional to offered hardware resources [2, 3, 4]. The price doubles for renting a virtual machine instance with double CPU and memory resources. However, there is performance discrepancy for different server loads. Cache intensive algorithms run faster in virtualized environment while executed in CPUs with distributed caches per core, but have huge performance drawbacks in the regions where problem size fits in cache that is shared among several CPU cores [8]. There are results published for performance of memory demanding and computation intensive algorithms using the same hardware environment but different platforms. The results show that there are performance drawbacks if they are hosted in the cloud compared to traditional environment [9].

Matrix multiplication is one of the most important linear algebra algorithms that can be easily, efficiently and effectively speeduped using parallelization. Achieving near linear speedup while executing on more processors is imperative. However, the authors in [13] found superlinear speedup for sparse symmetric matrix vector multiplication. Cache associativity has great impact on algorithm performance. Gusev and Ristov [14] proved several theorems that determine the problem size where the performance drawback appear due to set cache associativity. Padding to the first element reduces

the associativity problem [15]. In this paper we analyze the performance of dense matrix-vector multiplication algorithm while executed both in traditional host operating system and in the cloud using the same operating system, runtime environment and hardware resources. We use sequential and parallel implementations to determine the problem size regions where maximum speed and speedup is achieved. The hypothesis we would like to confirm is that the performances are degraded when moving onto cloud for a constant value. We expect this to be confirmed also for parallel execution in multicore environment.

The paper is organized as follows. Section 2 describes the testing methodology along with description of testing environment and test cases for the experiments. In Section 3 we present the results of the experiments for both platforms and both implementations. Finally, Section 4 is devoted to conclusion and future work.

2 TESTING METHODOLOGY

This section describes the testing methodology based on two different environments and 3 test cases for each environment.

2.1 Testing Algorithm

Dense matrix-vector multiplication algorithm $C_{N\cdot1} = A_{N\cdot N} \cdot B_{N\cdot1}$ is used as test data. One thread multiplies the whole matrix $A_{N\cdot N}$ with vector $B_{N\cdot1}$ for sequential test cases. For parallel test cases each thread multiplies the row matrix $A_{N\cdot N/c}$ with vector $B_{N\cdot1}$ where c denotes the total number of parallel threads and used CPU cores.

2.2 Testing Environments

We analyze two platforms as testing environments using the same hardware resources and runtime environments.

A. Hardware Resources

Both platforms use a workstation with shared memory multiprocessor Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz and 8GB RAM. The multiprocessor possesses 4 cores, each with 32 KB L1 cache and 256KB L2 cache dedicated per core, and L3 cache with total 12 MB shared per two cores (each core is using 6MB).

B. Runtime and Operating System Environments

Linux Ubuntu Server 12.04 is installed on both machines; either virtual in the cloud or traditional host operating system. C++ is used together with OpenMP as API for parallel implementation. The matrix and vector elements are double, i.e. 8 bytes each.

2.3 Testing Platforms

Two different platforms are defined, i.e. traditional and cloud platform.

A. Traditional Platform

The traditional platform consists of one traditional operating system installed on the real hardware machine described in Section 2.2.A.

B. Cloud Platform

The cloud environment uses OpenStack open source cloud solution [10]. OpenStack is deployed in dual node [11] as depicted in Fig. 1.

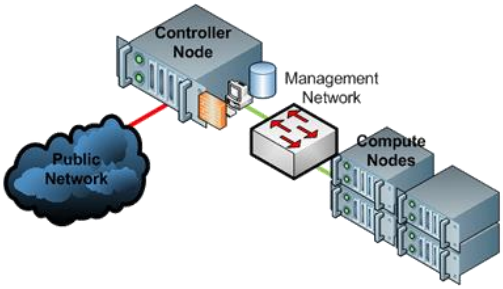

Fig. 1. OpenStack dual node deployment [11]

We use only two nodes, i.e. Controller Node and Compute Node since the algorithm is computation and cache intensive, and memory demanding executed on shared memory multiprocessor, and does not produce huge network traffic. Kernel-based Virtual Machine (KVM) virtualization standard is also used for instancing virtual machine.

The cloud environment uses the same hardware and operating system as described in Section 2.2.A for Compute Node server. Virtual machine instance is instantiated with all available hardware resources and it is installed with the same Linux Ubuntu 12.04 operating system as traditional platform.

2.4 Test Cases

We realize three groups of test cases using different number of threads:

- sequential execution on one core using one thread;
- parallel execution on two cores using two threads; and
- parallel execution on four cores using four threads.

Series of experiments are realized in each test case by varying the matrix size $N \cdot N$ and vector size N from $N = 2$ to $N = 1400$ in order to analyze the performance behavior upon different overload and variable cache storage requirements, i.e. for each cache region L_1 to L_4 [8].

Execution Time *T(P)* is measured for each test case and *Speed V(P)* is calculated as defined in (1). Variable *P* denotes the number of threads and cores.

$$V(P) = \frac{2 \cdot N^2}{T(P) \cdot 10^9} \tag{1}$$

We repeat each test case for 20 seconds and calculate the average Execution Time *T(P)* in order to achieve reliable test results.

The speed *V* is expressed in Gigaflops, i.e. the number of floating point operations in second.

Speedup S is also calculated as defined in (2) where T_{Seq} denotes the execution time for sequential execution and $T_{Par}(P)$ denotes the execution time for parallel execution with *P* threads on *P* cores.

$$S(P) = \frac{T_{Seq}}{T_{Par}(P)} \tag{2}$$

We use Speedup S in order to analyze the performance of the algorithm while executing in the same platform using different hardware resources.

Relative Speedup *R(P)* is defined in (3) in order to compare the algorithm performance in both platforms using the same hardware resources, i.e. *P* threads and CPU cores. $T_T(P)$ denotes execution time while the algorithm is executed using *P* threads on *P* cores in traditional platform and $T_C(P)$ denotes execution time while the algorithm is executed using *P* threads on *P* cores in cloud platform.

$$R(P) = \frac{T_T(P)}{T_C(P)} \tag{3}$$

3 RESULTS OF THE EXPERIMENTS

This section presents the results of the experiments realized on both platforms to determine algorithm behavior for parallel execution with two and four threads compared to sequential execution.

3.1 Speed Analysis

In this section we analyze the Speed *V* that the algorithm achieves using different number of threads in both platforms.

A. Speed in Traditional Platform

Fig. 2 presents the measured Speed *V* achieved in traditional platform using *P* = 1, 2 and 4 threads on 1, 2 and 4 cores correspondingly.

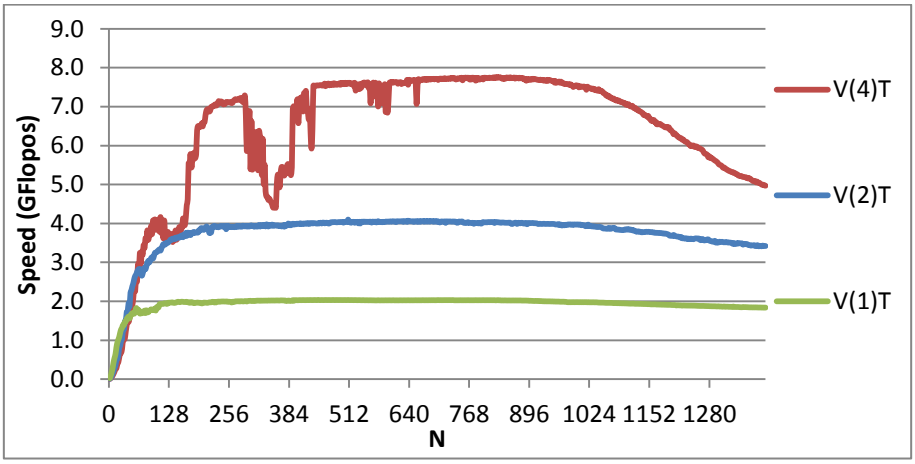

Fig. 2. Speed *V* in traditional platform

We observe that the speed is greater while using greater number of threads. However, the speed *V*(1) for the sequential execution is slightly greater than both speeds V(2) and V(4) for parallel execution for $N \leq 32$. Also, the speed *V*(2) is slightly greater than *V*(4) for $N \leq 64$. We explain this with the fact that the operating system needs more time to create the threads and schedule the tasks.

Three regions are observed for each test case. The speed increases proportionally for left region, where approximately $N \leq 128$ for *V*(1). In the middle region, where approximately $128 \leq N \leq 1024$ for *V*(1), the speed is constant although *N* increases meaning that it is independent of the problem size. Saturation occurs in the right region, i.e. for $N \geq 1024$, and the speed begins to decrease. In this region the problem size is greater than cache requirements and a lot of cache misses are being generated, affecting the performances.

Another unusual result is occurrence of speed drawbacks for *V*(4) in two regions, i.e. $120 \leq N \leq 156$ and $292 \leq N \leq 392$. We also observe similar speed drawbacks for *V*(1) and *V*(2) but they are smaller than those for *V*(4).

B. Speed in Cloud Platform

The measured Speed *V* achieved in cloud platform using *P* = 1, 2 and 4 threads on 1, 2 and 4 cores correspondingly is depicted in Fig. 3.

As in the previous case, the speed increases for greater number of threads. There is a region for $N \leq 40$, where the speed *V*(1) slightly leads the race in front of speeds V(2) and V(4) correspondingly. This happens since the time required for processing a small problem size is smaller than the time required by the operating system to establish and run a new thread.

Similar to the traditional platform, three regions are observed for each test case. The speed increases for greater *N* in the left region, identified up to approximately $N \leq 128$ for *V*(1). The middle region is identified for approximately $128 \leq N \leq 1000$ for *V*(1). The speed in the middle region is constant although *N* increases, i.e. it is independent of the problem size. The speed in the right region, (for approximately $N > 1000$) begins to decrease. The start of the third region begins for different *N* values in the observed three platforms.

The unusual results identified in the traditional environment are observed for $V(4)$ in region $302 \leq N \leq 378$. There are also similar speed drawbacks for $V(1)$ and $V(2)$ but much smaller than those for $V(4)$.

3.2 Speedup Analysis

In this section we analyze the Speed V that the algorithm achieves using different number of threads in both platforms.

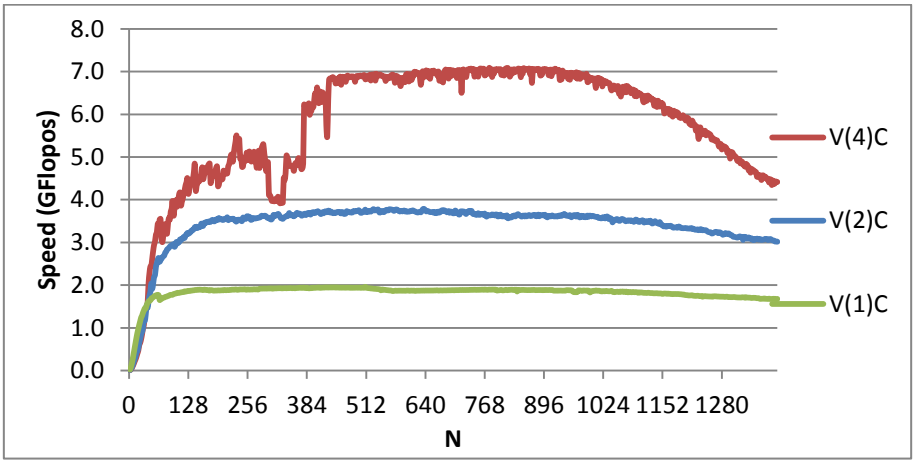

Fig. 3. Speed *V* in cloud platform

A. Speedup in Traditional Platform

Fig. 4 presents the measured Speedup S achieved in traditional platform using parallel implementation with $P = 2$ and 4 threads on 2 and 4 cores correspondingly compared to sequential implementation.

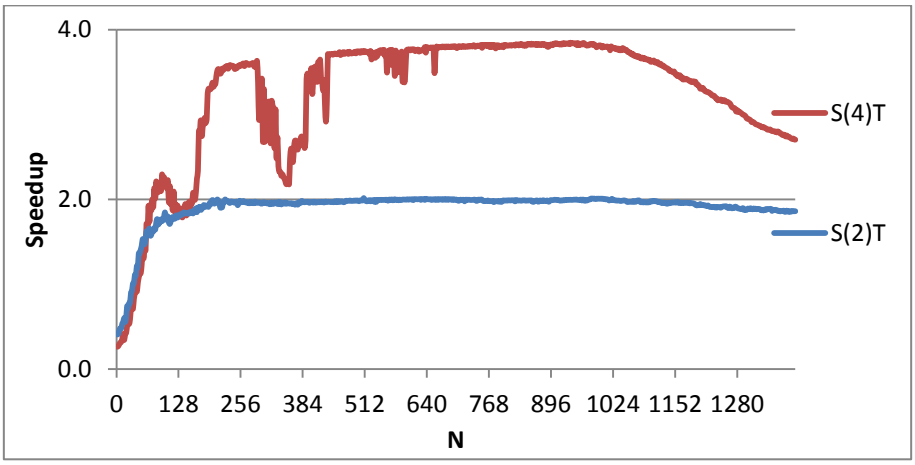

Fig. 4. Speedup *S* in traditional platform

Similar to the speed behavior, we observe that the speedup is greater for greater number of threads. Interestingly, the speedup $S(2)$ is slightly greater than $S(4)$ for $N \leq 64$, due to the already explained behavior for small problem sizes where the time required by the operating system to initiate a new thread and schedule a task is comparable to the algorithm execution times. The achieved speedup curves satisfy the Gustaffson's Law [12] showing sublinear speedup, i.e. $S(P) \leq P$ for each matrix and vector size N.

As in the speed analysis, for each test case we observe three regions with different behavior. The speedup rises in the left region, then shows stable performance in the middle region, and after $N \geq 1024$ the speedup begins to decrease.

The already explained unusual results for speed $V(4)$ are also observed for speedup $S(4)$ when executing with 4 threads.

B. Speedup in Cloud Platform

The measured Speedup *S* achieved in cloud platform using parallel implementation with *P* = 2 and 4 threads on 2 and 4 cores correspondingly compared to sequential implementation is depicted in Fig. 5.

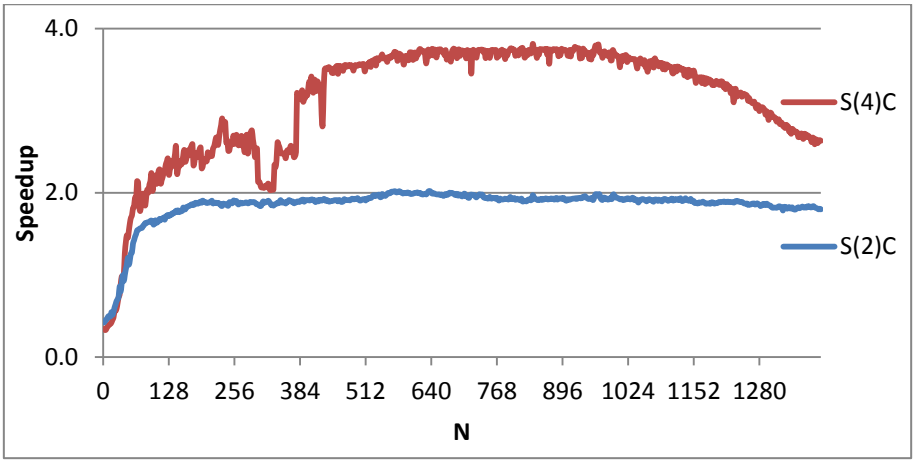

Fig. 5. Speedup *S* in cloud platform

The results are similar to the traditional environment, i.e. the speedup increases for greater number of threads. However, there is a region $N \leq 40$, where the speedup $S(2)$ is slightly greater than $S(4)$ due to already explained reasons for the left region. The Speedup curves also satisfy Gustaffson's Law [12] i.e. the speedup is sublinear.

As in previous cases, three regions are also observed, i.e. the speedup rises, stabilizes and after $N \geq 1024$ the speedup begins to decrease. The difference here is manifested for the speedup S(2) which begins to decrease much earlier, i.e. for $N \geq 740$.

Unusual results reported for traditional environment are also present for $S(4)$ in the same regions as $V(4)$.

3.3 Comparison of Different Platforms

This section compares the results for both platforms, i.e. traditional with cloud platform using the same hardware resources.

A. Speedup Comparison

Fig. 6 compares the speedups S(2) for both platforms. We can conclude that traditional platform achieves better speedup compared to cloud platform for each *N* for parallel execution using two threads on two CPU cores.

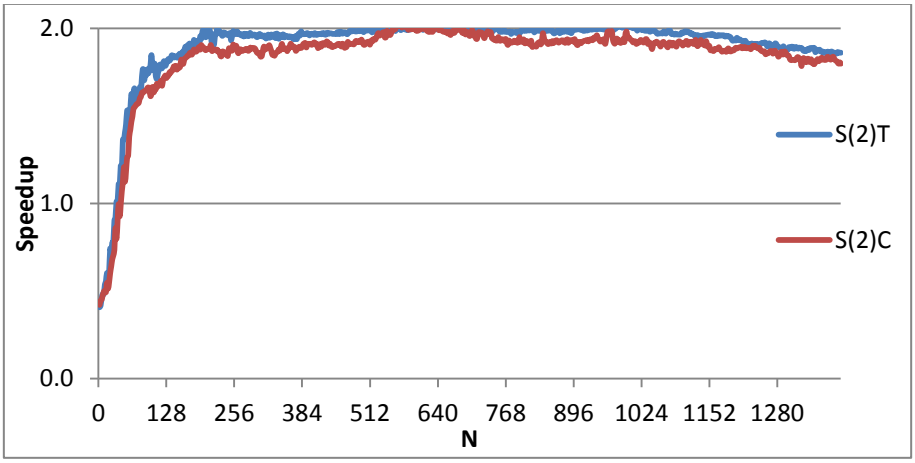

Fig. 6. Speedup *S(2)* in both platforms

The comparison for speedups S(4) in both platforms is depicted in Fig. 7. We can conclude that there is a region ($N \leq 168$) where cloud platform achieves better speedup compared to traditional platform

for each *N* for parallel execution using four threads on four CPU cores. The traditional platform achieves better speedup *S*(4) for greater *N*.

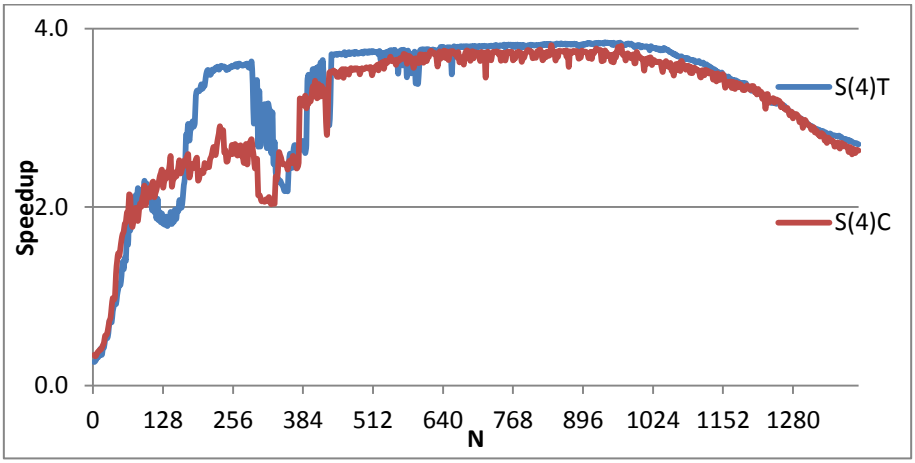

Fig. 7. Speedup *S(4)* in both platforms

B. Relative Speedup Comparison

Fig. 8 depicts the three relative speedups using *P* = 1, 2 and 4 threads on 1, 2 and 4 CPU cores between cloud and traditional platform.

We observe that there is a region (*40* ≤ *N* ≤ 62) where cloud environment provides relative speed R(1) > 1, i.e. better performance presented as smaller execution time *T*(1) or greater sped *V*(1). The cloud also achieves better performance using all four cores in region (*N* ≤ 168) and particular points in region (3*40* ≤ *N* ≤ 390). In all other points cloud provides worse performance where *R*(1) leads the race.

Calculated average relative speedups are 94.1%, 91.1% and 91.6% for *P* = 1, 2 and 4 correspondingly. That is, the hypothesis is confirmed and the average factor of degradation is 5.9% for sequential execution and approx. 8.9% and 8.4% correspondingly for 2 and 4 threads executed on 2 or 4 cores when the application is moved onto cloud. However, we have to be careful, since this behavior depends on identified region.

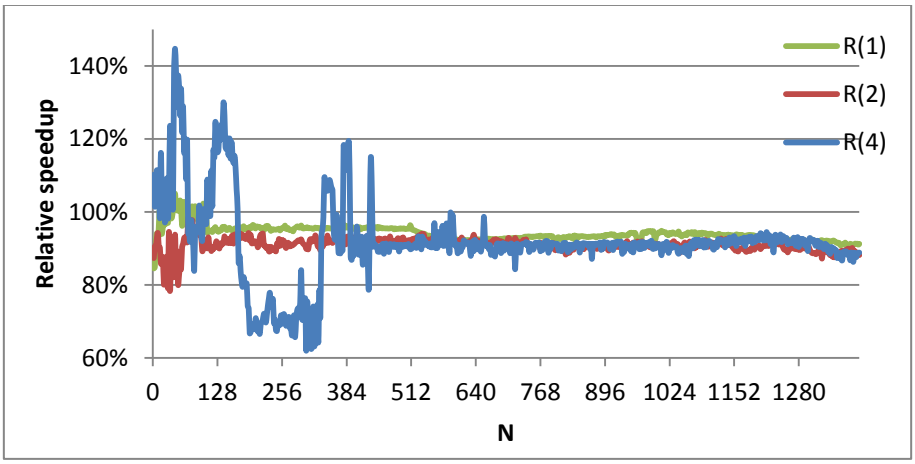

Fig. 8. Relative speedup *R* for different hardware resources

4 CONCLUSION AND FUTURE WORK

This paper analyzes the performance of dense matrix-vector multiplication algorithm using different hardware resources of 1, 2 and 4 threads and cores for each test case, and executed in different platforms, i.e. traditional host operating system and in virtual machine instance in the cloud.

The results show that there are regions (for small problem size) where the sequential execution provides slightly better performance than parallel execution using two or four cores. More generally, using smaller number of CPU cores is slightly better than greater number for both platforms. We

explain this phenomenon with the fact that operating system needs more time to create more threads and to schedule the tasks.

There are regions for small problem size where the cloud provides better speed compared to traditional platform for sequential and parallel execution with four cores. We also determine that there is a region where cloud provides better speedup but only for parallel execution with four cores.

We observe strange algorithm behavior in particular regions emphasized while executed using four threads on four cores in both platforms that will be the focus of our further research. We also plan to extend the research using different hardware resources, other cloud platforms and different hypervisors.

References

[1] Gartner, Gartner News Room, Dec. 2012. http://www.gartner.com/it/page.jsp?id=2163616

[2] Microsoft. (2012, Dec) Windows azure. [Online]. Available: http://www.windowsazure.com/pricing/

[3] Google. (2012, Dec) Compute engine. [Online]. Available: http://cloud.google.com/pricing/

[4] Amazon. (2012, Dec) EC2. [Online]. Available: http://aws.amazon.com/ec2/

[5] R. Iakymchuk, J. Napper, and P. Bientinesi, “Improving high performance computations on clouds through resource underutilization,” *in Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC ’11. New York, NY, USA: ACM, 2011, pp. 119–126.

[6] B. Bouterse, H. Perros, “Scheduling Cloud Capacity for Time-Varying Customer Demand,” *in 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*. Paris, France: IEEE, Nov. 2012, pp. 137–142.

[7] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “Ananalysis of performance interference effects in virtual environments,” *in Performance Analysis of Systems Software, 2007. ISPASS 2007*. *IEEE International Symposium on*, Apr. 2007, pp. 200 –209.

[8] M. Gusev and S. Ristov, “Matrix multiplication performance analysis in virtualized shared memory multiprocessor," *in MIPRO, 2012 Proceedings of the 35th International Convention*, IEEE Conference Publications, May 2012, pp. 264-269.

[9] S. Ristov, G. Velkoski, M. Gusev, M. Kostoska, and K. Kjirovski, “Compute and Memory Intensive Web Service Performance in the Cloud”, in *ICT Innovations 2012*, Advances in Intelligent and Soft Computing, (ed. S. Markovski and M. Gusev), Springer Verlag, Berlin Heidelberg, 2013, volume AISC 257, pp.215-224

[10] Openstack. (2012, Dec.) Openstack compute. [Online]. Available: http://openstack.org/projects/compute/

[11] Openstack. (2012, Dec.) Openstack Dual Node. [Online]. Available: http://docs.stackops.org/display/documentation/Dual+node+deployment

[12] Gustafson J. L, “Reevaluating Amdahl’s law,” *Communication of ACM May 1988*; 31(5):532–533.

[13] So B, Ghuloum A. M,Wu Y. “Optimizing data parallel operations on many-core platforms,” *The First Workshop on Software Tools for Multi-Core Systems (STMCS)*, 2006, pp 66-70.

[14] M. Gusev and S. Ristov, “Performance gains and drawbacks using set associative cache,” *Journal of Next Generation Information Technology (JNIT)*, vol. 3, no. 3, pp. 87–98, 31 Aug 2012.

[15] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J. “Optimization of sparse matrix-vector multiplication on emerging multicore platforms,” *Parallel Computing*, 2009, 35, 3, 178-194.