Proceedings of the Federated Conference on Computer Science and Information Systems pp. 889–898
DOI: 10.15439/2016F498
ACSIS, Vol. 8. ISSN 2300-5963

Superlinear Speedup in HPC Systems: why and when?

Sasko Ristov, Radu Prodan
University of Innsbruck
Innsbruck, Austria
Email:{sashko, radu}@dps.uibk.ac.at

Marjan Gusev
University Ss Cyril and Methodius, FCSE
Skopje, Macedonia
Email:marjan.gushev@finki.ukim.mk

Karolj Skala
Ruđer Bošković Institute
Zagreb, Croatia
Email:Karolj.Skala@irb.hr

***Abstract*—The speedup is usually limited by two main laws in high-performance computing, that is, the Amdahl's and Gustafson's laws. However, the speedup sometimes can reach far beyond the limited linear speedup, known as superlinear speedup, which means that the speedup is greater than the number of processors that are used. Although the superlinear speedup is not a new concept and many authors have already reported its existence, most of them reported it as a side effect, without explaining why and how it is happening.**

In this paper, we analyze several different superlinear speedup types and define a taxonomy for them. Additionally, we present several explanations and cases of superlinearity existence for different types of *granular* algorithms (tasks), which means that they can be divided into many sub-tasks and scattered to the processors for execution. Apart from frequent explanation that having more cache memory in parallel execution is the main reason, we summarize other different effects that cause the superlinearity, including the superlinear speedup in cloud virtual environment for both vertical and horizontal scaling.

***Index Terms*—Cache memory, load, parallel and distributed processing, performance.**

I. Introduction

THE goal of today's world of parallel and distributed systems is to achieve the greatest speedup, represented either as the lowest time for execution of a single task (High Performance Computing), or to execute as many tasks as possible for a given time (High Throughput Computing), when the task(s) are executed on scaled resources. Many new algorithms and computing paradigms appeared in the last decade, and new challenges have emerged to solve more complex problems faster, or to achieve greater speedup, as much as possible [1].

The speedup is usually defined as a ratio of the wall times of sequential and parallel execution of an algorithm. The target of the parallelization is to achieve the lowest execution time in order to maximize the speedup against the best sequential algorithm. Increasing the number of computing resources will increase the intra-resource's communication and requires additional operations, such as reduction operations.

Most of the authors analyze the computer only as a processing unit, focusing on the processing power, without analyzing the details of the computer as a complex system with memory and I/O devices as resources. Actually, these resources limit the speedup, or can boost its performance.

According to the Gustafson's Law [2], the speedup is limited with the number of processors, when the linear speedup is achieved. However, beyond the limits, the superlinear speedup happens in reality for plenty of reasons and it allows an increased utilization of parallel systems [3].

Many authors reported the existence of a superlinear speedup, but most of them only mentioned it as a side effect [4]. Besides reporting a superlinearity, other researchers briefly presented that the reason for achieving a superlinear speedup is because of the greater amount of cache memory in the parallel execution compared to the sequential [5]. However, these explanations are insufficient. For example, all currently produced multiprocessors contain a multi-level cache, but a superlinear speedup is not reported always. Also, it is not reported for all algorithms. Sometimes it is limited to the problem size or the number of used multiprocessors.

In this paper, we present a systematic overview of the reasons why the superlinear speedup appears. The analysis approach is to focus on granular algorithms, in both traditional and cloud virtual environments. Tthe superlinearity is reported in both environment types, explaining the reasons summarized in this paper. Data- or code-parallelism divides a single task into threads or processes and sends them for execution on different processors, thus aspiring to become a high-performance computing system with a goal to finish the task as fast as possible. On the other hand, today's service oriented architectures offer scalable web services to their customers using elastic cloud resources. The latter approach tends toward a high throughput computing system aiming at serving as many possible customers for a certain time, without reducing the service performance.

Due to its elasticity and the linear pay-as-you-go model, the cloud is preferred platform both for high performance algorithms, especially if they are low communication-intensive, such as scientific applications [6], [7]. Many scientific applications are moving from computation-intensive to data-intensive, that is, they require high throughput computing, rather than high-performance computing. This is a huge challenge in the cloud because the data transfer between the cloud compute nodes and storage is a bottleneck [8]. Despite the additional virtualization layer, a superlinear speedup is also reported for granular algorithms [9].

978-83-60810-90-3/$25.00

The rest of the paper is organized as follows. The speedup limits in parallel executions are described in Section II. Section III elaborates when and how a superlinear speedup can be achieved for a parallel implementation of some algorithm. Examples of obtained superlinear speedup for high performance algorithms are presented in Section IV. Despite the virtualization layer, the cloud environment can also achieve a superlinear speedup, as discussed in Section V. Section VI discusses several paradoxes, as well as further challenges. Finally, we conclude the paper and present the plans for future work in Section VII.

II. Speedup limitations

This section briefly explains the two main laws in the computer architecture about the limit of the maximal speedup that can be achieved when an algorithm is executed parallel with more computing resources, that is, Amdahl's [10] and Gustafson's laws. The former targets the speedup for problems with fixed problem size while the latter the algorithms that require intensive parallel processing.

Speedup $S(p)$ is defined as a ratio of the execution times of the best sequential algorithm $T(1)$ and the parallel implementation on p processors $T(p)$, as presented in (1). However, this definition holds only for fixed-time algorithms. When analyzed more broadly, the speedup should be defined as a ratio of speeds, and not of times, as defined in (2). Note, that for fixed-time algorithms, the amount of work is constant, which results in (1).

$$S(p) = \frac{T(1)}{T(p)} \tag{1}$$

$$S(p) = \frac{(\frac{ParallelWork}{ParallelTime})}{(\frac{SerialWork}{SerialTime})} \tag{2}$$

Fig. 1 presents the theoretical or ideal speedup for both laws, depending on the number of processors used in the parallel execution. The Amdahl's Law limits the speedup to the value $1/s$, as defined in (3), where s is the portion of the serial part for the fixed size program. The conclusion is that the speedup is limited regardless of the number of processors, when the problem is fixed.

$$S^{max}_{Amdahls}(p) = 1/s \tag{3}$$

The Gustafson's Law, on the other side, shows that if the problem is executed within a fixed time, the maximum value of the speedup is *linear* limited by the number of processors, as defined in (4), assuming that the problem size increases and the serial portion becomes negligible. However, in real executions, due to communication, synchronization, and resource sharing, the speedup is *sublinear*, or $S(p) < p$.

$$S^{max}_{Gustafson}(p) = p \tag{4}$$

Both the Amdahl's and Gustafson's laws calculate the maximum speedup, that is, the speedup limit of a parallel algorithm or program; they both consider that the serial part of the algorithm does not depend on the number of processors. However, this is in the ideal condition, while in a real situation, each processor does not start and finish in the same time, and the communication overhead and synchronization can harm the parallel execution when the number of processors increases.

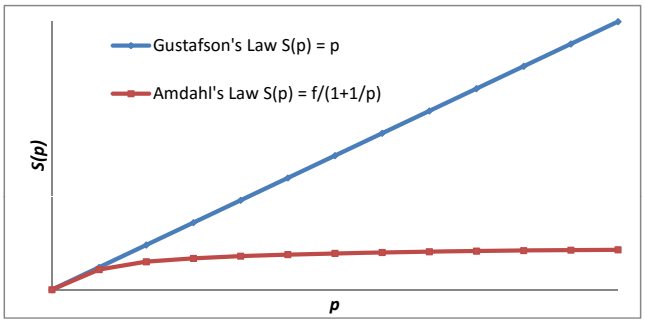


Fig. 1. Speedup for Amdahl's and Gustafson's laws

Karp and Flatt [11] introduced the *scaled serial fraction* f_k of an algorithm as defined in (6), where p is the number of processors, and s_k is the speedup that calculates the overhead (5) as a number of the executed additional arithmetic operations for parallel execution. Parameter k represents the scaling factor for the overhead in a parallel implementation using p processors.

$$s_k = \frac{k \cdot T(1,1)}{T(p,k)} \tag{5}$$

$$f_k = \frac{1/s_k - 1/p}{1 - 1/p} \tag{6}$$

Let's discuss the relation for the scaled serial fraction. If s_k = p, then $f_k = 0$, which yields to the Gustafson's Law. The results of the parallelization will still be good even if the parallel implementation achieves a small speedup, while f_k retains to a some constant value, because the algorithm has limited parallelization.

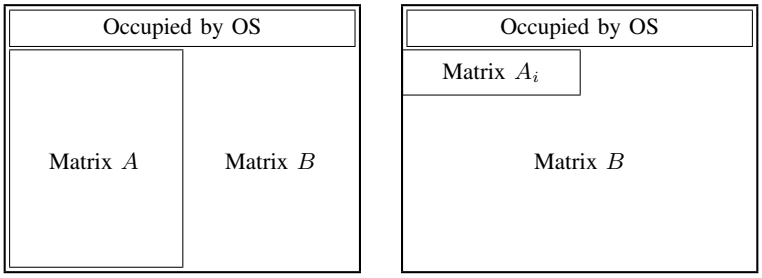
Let's rewrite (6) as (7) in order to determine the speedup that calculates the overhead s_k and yield special cases, that is, the Amdahl's and Gustafson's law.

$$s_k = \frac{1}{f_k \cdot (1 - 1/p) + 1/p} \tag{7}$$

If the scaled serial fraction $f_k = 0$, then $s_k = p$, which yields toward Gustafson's Law, while if $p \to \infty$, then $s_k = 1/f_k$, as Amdahl's Law states. For each scaled system with $f_k > 0$, the scaled speedup that calculates the overhead is sublinear, i.e. $s_k < p$.

III. Beyond the speedup limits. Why and when?

Although the limits are given by the Gustafson's Law, the speedup achieved by executing some algorithms on parallel configurations goes beyond it, achieving a superlinear speedup. This section presents several such cases, along with detailed



a) sequential version b) parallel version

Fig. 2. Cache occupancy in sequential and parallel execution

explanation about the reason for various superlinear speedup appearances.

Let's analyze when a superlinear speedup can be achieved while parallelizing a sequential problem. The CPU execution times for sequential algorithm T_s and parallel algorithm T_p are respectively defined in (8) and (9), where CC and MC represent the clock cycles required by the processor for execution of operations and memory accesses, correspondingly, and CT the time period of a single clock cycle. In a homogeneous environment, CT will be the same for both implementations.

$$T_s = (CC_s + MC_s) \cdot CT \tag{8}$$

$$T_p = (CC_p + MC_p) \cdot CT \tag{9}$$

Shi [12] classified the parallel versions of algorithms to be either structure persistent or non-persistent. The former means that the number of total operations that the algorithm executes is same both for the sequential and parallel implementation, for the same input. The latter's parallel implementation does not execute all operations, that the compatriot sequential algorithm would. A formal notation of (7) means that the scaled serial fraction $f_k < 0$, which yields toward a superlinear speedup $s_k > p$.

Gusev and Ristov [13] defined the condition when a superlinear speedup can be obtained for a shared memory multiprocessor, which is presented in (10), where a positive number ϵ exists, such that $0 \leq \epsilon \leq p$ and $CC_s = CC_p \cdot (p - \epsilon)$. The parameter ϵ represents the effect of parallelization overhead and synchronization and p the number of scaled computing resources.

$$MC_s > p \cdot MC_p + \epsilon \cdot CC_p \tag{10}$$

The superlinearity was defined for cache-intensive algorithms only (algorithms where the cache-intensive complexity represented by the average reuse of an element c is greater than 1 [14]). For example, the dense matrix-matrix multiplication algorithm has cache-intensive complexity $c = O(N)$ because each element of matrices is accessed N times for different computations. On the other hand, the cache-intensive complexity of the scalar product is $c = 1$, which yields that a superlinear speedup cannot be achieved. We must note that the cache-intensive complexity defines the level of superlinearity, that is, an algorithm with greater cache-intensive complexity ($c \gg 1$) will achieve a greater superlinear speedup.

A. Superlinear speedup for non-persistent algorithms

Typical examples of non-persistent algorithms are searching algorithms, which finish when one of the processors finds the solution, and together with all the other processors stop the execution, without finishing all operations. In this case, a superlinear speedup usually appears because CC_p is smaller than CC_s, thus compensating the overhead of parallelization. This case can be better presented if the total number of clocks are presented through the number of instructions I and CPI (clocks per instruction), as presented in (11) and (12) [15]. I_p will be smaller than I_s, which will cause a spurious superlinear speedup.

$$CC_s = I_s \cdot CPI_{CC}; \tag{11}$$

$$CC_p = I_p \cdot CPI_{CC} \tag{12}$$

Many examples can be found in the literature for superlinear speedup of the non-persistent algorithms. For example, parallel shortest path planning [16].

B. Superlinear speedup for persistent algorithms

The total number of instructions of the sequential and parallel implementations of structure persistent algorithms is the same, that is, $I_p = I_s$, which means that superlinear speedup appears due of the memory clock cycles, that is, by reducing the number of clocks per instruction for memory access CPI_{MC} in the parallel implementation. There are several different cases when CPI_{MC} in parallel implementation will be smaller than its serial compatriot. Let us explain all these cases.

1) More cache for parallel execution: The case when the parallel execution of a structure persistent algorithm can obtain a superlinear speedup due to utilizing more cache memory is the mostly reported by the researchers [17].

Since more cache memory is used in parallel execution, for some region of problem size, it can store the whole problem

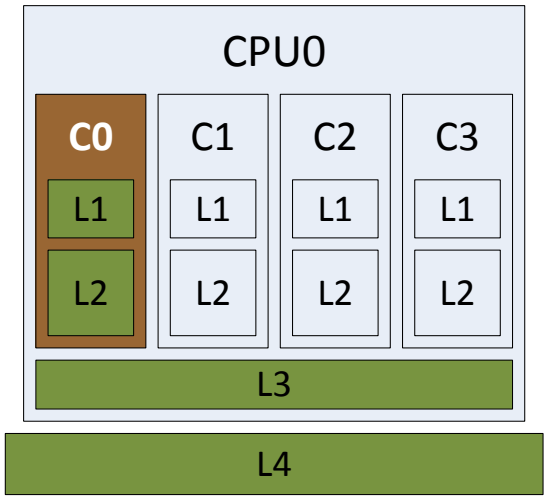


Fig. 3. Utilized memory for sequential execution

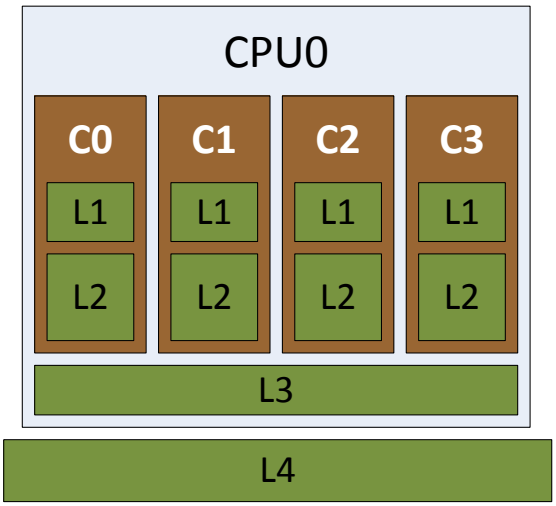


Fig. 4. Utilized multi tiered memory for loosely coupled processors for parallel execution

size, while the sequential execution cannot, as presented for storing matrices in Fig. 2 [13].

Fig. 3 [18] presents an example for utilized memory tiers of a typical multiprocessor with four cores, each with private L1 and L2 cache memory, shared L3 cache memory, and main memory represented by L4.

Velkoski et al. [18] went beyond this analysis. They have analyzed the impact of loosely and tightly coupled cores for parallel implementation and concluded that the former is superior for naive dense matrix-vector multiplication. The utilization of memory tiers of a typical multiprocessor for loosely and tightly coupled cores in parallel execution is presented in figures 4 and 5 [18]. The tightly coupled case uses all four cores on one chip, while the loosely coupled case uses one CPU core of four chips on a shared memory multiprocessor. The results show that a superlinear speedup region appears for both loosely and tightly coupled processors, which starts for the same matrix size, but the former's region is wider, as well as it achieves a greater superlinear speedup. These results clearly present that the use of more L2 cache memories for parallel execution yields a superlinear speedup in the tightly coupled processors, while more L3 cache memory generates even a greater superlinear speedup and region, despite the increased overhead of the inter-chip communication, compared to the intra-chip for tightly-coupled systems.

Another interesting example was reported by Djinevski et al. [19] achieving a superlinear speedup region on GPU, when they used one loosely coupled processing unit of all streaming multiprocessors (SMs) for parallel execution, and a single processing unit of one SM for sequential execution. The superlinear speedup regions are achieved regardless of the used number of SMs.

2) Shared cache for parallel execution: Although most of the reported superlinear speedups are obtained because of the increased cache memory in a parallel execution, a superlinear speedup is achieved in those some algorithms addressing a common shared cache. That is, a superlinear speedup can be achieved even in the tightly coupled processors.

For example, this is the case for an algorithm where the same variables (data) are used by several or all shared memory multiprocessors. If these variables are defined as shared, then fetching a variable by one processor will load it in the upper memory tier (for example, from RAM to the shared L3 cache), thus reducing the access time for the same variable by other processors.

Next, let's explain the difference when multiprocessors, which use private per core cache or shared cache, access the data from the memory. Without loosing generality, assume that the multiprocessor has one cache level and the accessed memory location is not present in the cache.

Fig. 6 presents how two multiprocessors, each with a private cache, access the same memory location. Let's assume that the instruction $Read(X)$ is executed by the processor A earlier. It will generate a cache miss, and pay the penalty for that. After fetching the variable X from the memory into its private cache, the processor B will do a similar sequence. This means that in this case, two cache misses and two memory accesses will happen.

Accessing the data in the memory by two multiprocessors that use a shared cache is presented in Fig. 7. In this environment, when the processor A accesses the variable X, a cache miss will be generated and one memory access. Now, when the processor B will require the same variable X in the near future without replacing it from the cache, a cache hit will be generated without a cache miss penalty and an additional memory access.

We can conclude that a tightly coupled multiprocessor (that uses a shared cache) can benefit when shared variables are used by reducing the cache misses and memory accesses.

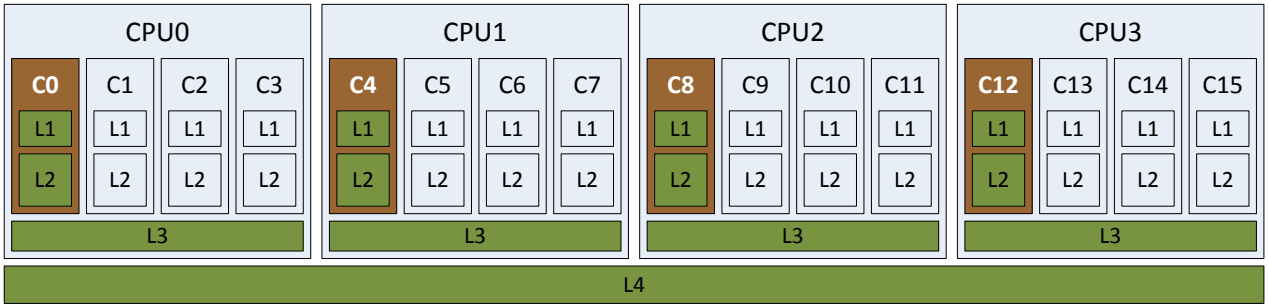


Fig. 5. Utilized multi tiered memory for loosely coupled processors for parallel execution

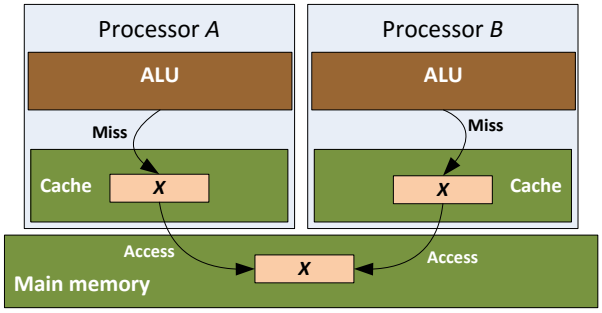


Fig. 6. Accessing the data from the memory by two multiprocessors that use private cache. Two cache misses and two memory access will happen.

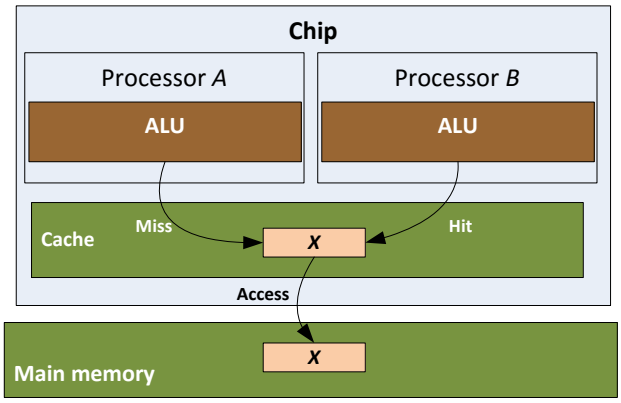


Fig. 7. Accessing the data from the memory by two multiprocessors that use shared cache. Only one cache miss, one cache hit and only one memory access will happen.

The precondition is that the time and space locality should be utilized by all processors.

Anchev et al. [20] reported a superlinear speedup for dense matrix-matrix multiplication. The reason for superlinearity is the use of a shared L3 cache, or, as we explained earlier, the implicit prefetch of the data (matrix elements).

However, we must note that the superlinear speedup was reported only for AMD Opteron, while Intel's i7 obtained a sublinear speedup only. We believe that the reason for this is due to the fact that the frequency gap between L3 and main memory is much bigger for AMD Opteron, and thus reducing the cache miss ratio and penalties, which compensates the parallelization overhead, and generates a superlinear speedup.

3) Superlinear speedup in a heterogeneous environment: Superlinear speedup is reported in a heterogeneous environment that consists of three Intel Xeon CPU + one GPU NVIDIA FX Quadro, because the heterogeneous environment schedules the tasks better than the homogenous environment and thus reduces the impact of Amdahl's Law with a limited overhead in parallel execution [21].

IV. Superlinear Speedup Regions

This section overviews several examples of granular algorithms, where the existence of a superlinear speedup is reported. We define two different region types of a superlinear speedup: 1) for some range of the number of processors, usual

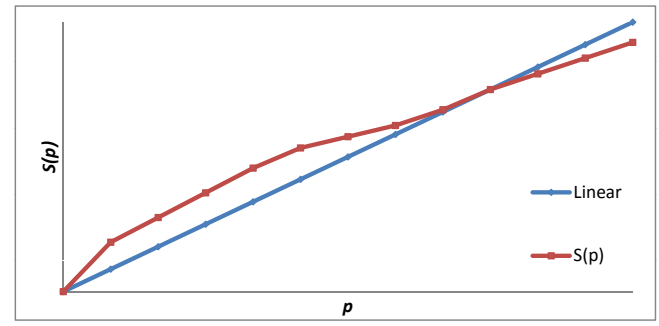


Fig. 8. Example of superlinear speedup for a particular range of number of processors (fixed problem size)

for fixed problem size, and 2) for a particular range of problem size, but fixed number of processors.

Fig. 8 presents a superlinear speedup for some range of the number of processors, usual for fixed problem size. The superlinearity usually starts even when two processors are used. However, it is lost as the scaling factor increases [22] due to the communication and synchronization overhead.

Another situation is to fix the number of processors, but

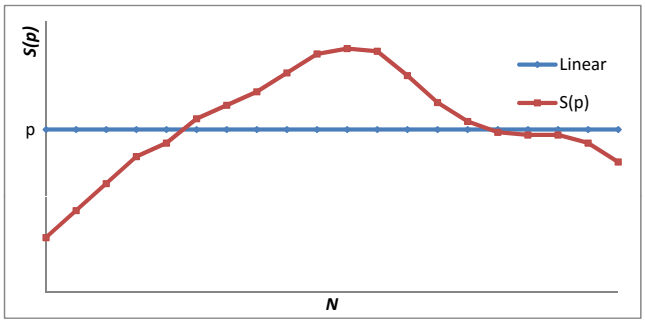


Fig. 9. Example of superlinear speedup for a particular range of problem size, but fixed number of processors

change the problem size, which can also impact the speedup, as presented in Fig. 9. We observe that there is a superlinear region for a specific range of problem size N where the speedup $S(p) > p$, while in other regions, it is sublinear.

We must note that sometimes, the speedup could be $S(p) < 1$, (sublinear speedup), which means that it is not speedup, but a slowdown. This could happen for several reasons. For example, for small problem size, which is negligible for good performance comparison, sequential execution will be faster than the time for forking threads. Another example is the case when the number of threads or processes is greater than the number of existing processors. Or more generally, a slowdown may happen due to the communication and synchronization time, or the overload of instruction in parallel execution. Further on, using the cache line for time and space locality in sequential execution can overcome the problem of the limited number of processors. Let's define it more formal, that is, the condition $MC_s < p \cdot MC_p$ will compensate $CC_p \leq p \cdot CC_s$, which will lead to a slowdown. In this case, the speedup could be achieved if problem size is divided into huge data chunks that will be scattered to the processing resources that will execute them sequentially.

Without losing generality, many authors use the *Efficiency* indicator calculated by $E(p) = \frac{S(p)}{p}$, which maps the limited speedup into the range [0, 1]. This parameter helps a lot to compare parallel executions with a different number of processors among each other. However, the value of $E(p)$ for superlinear speedup is $E(p) > 1$.

Gustafson [23] presented two cases of non-spurious superlinear speedup. Superlinear speedup can be obtained in distributed memory ensembles because of various memory speed. He also reported a superlinear speedup in cases when algorithms and tasks are with different speed.

Sometimes, parallel execution achieves a superlinear speedup because it partitions and reduces the data chunks, which can be placed in the cache memory, thus reducing the cache misses [24], [25], [26], [27], [28].

Many authors reported a superlinear speedup for parallel execution of some algorithms. However, most of them presented a likely explanation only for the superlinear speedup appearance. For example, one explanation is that the reason for achieving a superlinear speedup is because of having more cache in parallel execution. Still, in most cases the superlinear speedup is achieved for some range of the used number of processors, or for a specific problem size, or in a combination of both cases. For example, Monagan and Pearce [29] achieved a superlinear speedup for the parallel sparse polynomial division. However, they did not explain why a superlinear speedup has not appeared for extremely sparse problems, although a parallel execution uses the same amount of cache. Also, the same experiments have not reported a superlinear speedup on the Core 2 processor, although the level 3 cache has more cache than the sequential one.

Phillips et al. [30] reported a superlinear speedup, even when comparing parallel executions up to 26 processors with the one that uses two processors for continuous iterative guided spectral class rejection (CIGSCR). Peschlow et al. [31] achieved a superlinear speedup while simulating wireless networks, but only in a single range of a number of processors and for a specific number of nodes.

V. Analysis of a Superlinear Speedup in Cloud Environment

This section presents several cases where a superlinear speedup is achieved in cloud virtual environment for various types of scaling the resources.

Nowadays, cloud computing is being increasingly used for high-performance and high throughput applications. Its elastic on demand resources allow the customers to rent, for example, 1000 processors and execute a certain task, instead of building their own underutilized data center. Since the cloud's pricing strategy is linear, and expected speedup is also linear, it seems that customers will be charged fairly. In reality, the reported performance for communication-intensive high-performance algorithms shows that customers might feel that they are cheated. However, there are several cases where the superlinear speedup is achieved, despite the virtualization layer.

Customers can scale their rented resources horizontally, vertically or diagonally in the cloud. If the original configuration maps one process to a virtual machine (VM) instance hosted on a processor with one CPU core, as presented in Fig. 10 a), then Fig. 10 b), c) and d) present the three possible cloud scalings. The horizontal scaling presented in Fig. 10 d) increases the number of same VM instances and maps separate process (with a single thread) to a different VM instance. The vertical scaling presented in Fig. 10 b) increases the number of CPU cores per VM (resized VM) and maps separate threads of a single process to a different core on the same VM instance. A combination of the both scaling types yields a diagonal scaling presented in Fig. 10 c). To realize the vertical and diagonal scaling, the customer should use some API for parallelization, such as OpenMP, which will create parallel threads.

There are published papers that present a superlinear speedup in both the horizontal and vertical scaling. A superlinear speedup is reported and elaborated for cache-intensive algorithms [9] in the case of vertical scaling. Although sequential execution utilizes cache more, the superlinear speedup

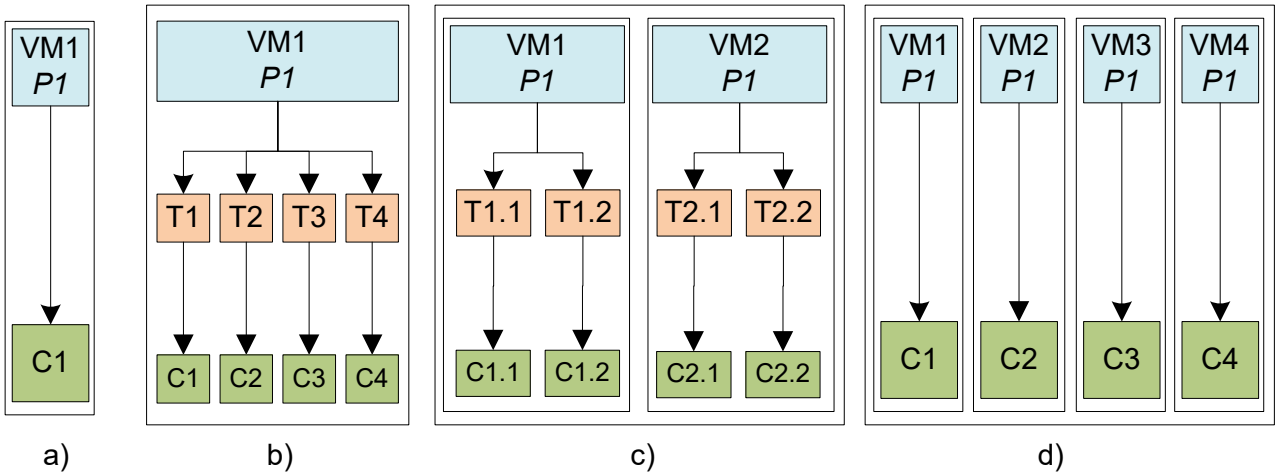


Fig. 10. Example of b) Vertical, c) Diagonal, and d) Horizontal scaling of nominal resources a)

is achieved also for horizontal scaling in the cloud, as well [14]. The authors have determined that the cloud environment handles the cases when the problem size can be fitted in the last level cache memory better, which is the reason why a superlinear speedup is achieved [32].

VI. Discussion

The superlinear speedup is achieved by many researchers, usually as a side effect without elaborating the theoretical background and explaining the details. In this paper, we have analyzed several aspects how to achieve a superlinear speedup, explaining why and when it can happen when various algorithms are executed on different platforms.

A. Superlinearity versus algorithm type

Mainly, the multi-tiered memory organization is the main reason to obtain a superlinear speedup for granular algorithms. We have classified two paradoxical cases; in the first case, the superlinearity appears because of the increased capacity of L2 and L3 cache memory, while in the second case, it is achieved because of the shared last level cache memory. A superlinear speedup is achieved in the first case, when the algorithm is executed on a loosely coupled system, while in the other case, the algorithm executed on a tightly coupled system.

The main reason is the difference of the algorithms. The loosely coupled parallel execution outperforms the tightly coupled for dense matrix-vector multiplication, in which, the matrix $A_{N \cdot N}$ is divided horizontally among processors for row-major memory, and implicit fetching is not used, while it is used only for the vector $B_{N\ cdot1}$, as presented in Fig. 11 a). However, its dimension is $O(N)$, while the matrix dimension is $O(N^2/p) \rightarrow O(N^2)$ is dominant, because the number of processors $p << N$.

On the other side, Fig. 11 b) presents the data that is accessed by the processor P_i for parallel execution of dense matrix-matrix algorithm, which shows that each processor uses the whole matrix B and therefore, implicit fetching yields a superlinear speedup. In this case, the size of shared data among all processors is bigger than the private chunks of matrix A, as well as compared to the vector's size in dense matrix-vector multiplication.

Another issue is the way of storing the matrices. Without loosing generality, we will assume that a row-major storing is used. Accessing the data of the matrix A is linearly, which utilizes the cache lines and thus reduces the cache misses regardless of the cache size. For example, when accessing the element $a_{i,j}$, the elements $a_{i+1,j}, a_{i+2,j}, \ldots a_{i+k,j}$ are also fetched in the cache. The size of k depends on the cache line and matrix element sizes. Therefore, cache misses are generated for the element $a_{i,j}$ only. Accessing the elements of the matrix B does not utilize the cache line, because a column of the matrix B is accessed linearly. In this case, the cache size is very important in order to store as much as possible a part of the matrix B.

We must note that although very naive examples of dense matrix-matrix and matrix-vector multiplications were presented, the generality is not lost. Our goal is not to prefer this algorithm for parallel execution, but just to show how and when a superlinear speedup can be obtained, paradoxically, for various algorithm - totally different reasons.

Using a multi-tiered memory is not the sine qua non for superlinearity. As we presented an example in Section III-B2, Intel i7 processor has not obtained a superlinear speedup for the same algorithm, as AMD Opteron. For example, a superlinear speedup is obtained on Cray XMT [33]. Intel Xeon achieved a superlinear speedup for two processors using the data-parallelism benchmarking (Black-Scholes), but only a sublinear speedup with dense matrix-vector multiplication [34]. Therefore, having cache memory is only one of the conditions for the existence of a superlinear speedup. An important observation is to return to the speedup definition,

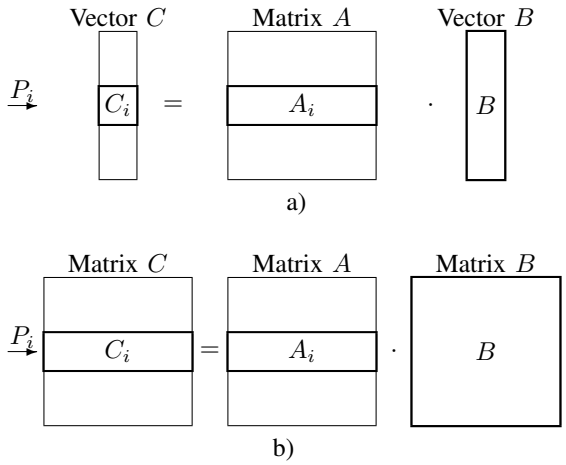


Fig. 11. Example of parallel implementations of matrix-vector and matrix-matrix multiplication.
a) Processor P_i uses chunk block A_i of matrix $A_{N \cdot N}$ and the whole shared vector $B_{N \cdot 1}$
b) Processor P_i uses chunk block A_i of matrix $A_{N \cdot N}$ and the whole shared matrix $B_{N \cdot N}$

i.e. to the benefits of parallelization that should compensate its overhead.

Also, another condition is to use cache-intensive algorithms, or to reuse of data; otherwise, having cache memory could be useless since, in both executions, each access will generate a cache miss. Even more, the superlinearity appears for some range of the number of processors, or for some problem size, or for both.

B. A new challenge: How to scale?

Cache-intensive granular algorithms, whose data reuse complexity is proportional to the problem size, will benefit from bigger cache. Many Intel's multiprocessors use a marketing trick with a huge L3 smart cache. However, one can easily check that it is not shared among all cores, but only among part of them. For example, 6MB of total 12MB cache is shared between each group of two cores. In this case, vertical scaling will utilize more the last level cache. AMD multiprocessors usually have smaller L3 cache, but it is shared among all cores of the multiprocessor. Therefore, depending on the algorithm, appropriate processor and scaling type should be chosen in order to achieve the best speedup, potentially superlinear.

On the other hand, today's cloud elastic resources can also be scaled in different ways: horizontally, vertically or diagonally, each of which can offer various performance and possibility for achieving superlinear speedup [35]. Vertical scaling provides a better speedup, but horizontal offers more flexible scaling of resources, which can minimize the cost.

C. Is the superlinear speedup always our target?

Achieving superlinear speedup does not necessarily mean that customers will obtain the maximum achievement. For example, the cache associativity in CPUs and GPUs [36] can provide a huge performance drawbacks for a specific memory pattern reading, and achieving the superlinear speedup for those inputs is not enough, but other techniques, such as padding, should be used. In the workflow executions in parallel and distributed systems, customers usually use bi-objective optimizations to minimize the makespan and cost. These two parameters are opposite one to another. Minimizing the makespan produces greater cost and vice verse.

Cloud computing customers can set a deadline for the execution requiring minimal cost, rather than minimal makespan [37]. In these cases, budget constraints and reducing the race for the speedup can yield the reduced cost for the execution. For example, although superlinear speedup is achieved in Windows Azure cloud for matrix multiplication when virtual machine instances with Windows operating system are used, Linux virtual machine instances achieved better performance cost trade-off because they are cheaper.

On the other side, there is a risk of cloud resources performance variation and instance failure during the time. Increasing the budget by duplicating the tasks on more than one instance could mitigate those risks, in order to meet the deadline [38]. Sometimes, using a bigger instance executes the task faster, rather than waiting several minutes for the deployment time to start another smaller, but an appropriate instance, which reduces the turnaround time of an activity [39].

VII. Conclusion and Future Work

Since the race in processor's frequency (Gigahertz) was abandoned a decade ago, which in the meantime has been migrated into the TOP500 race [40] for hunting ExaFLOPS, this overview of superlinearity could have an impact in the supercomputers' architecture and design, since the goal of each parallelization is to achieve the maximal speedup, which is superlinear.

The defined taxonomy for various scalings and definitions of superlinearity can open new ways for parallel and distributed systems. Defining how much to scale the resources is insufficient. One needs to define how to scale. Algorithms that can benefit from greater cache memory should scale vertically, while those that need to finish more work in a given time, should scale horizontally.

This paper overviews many reasons and presents practical cases to achieve a superlinear speedup when an algorithm is executed using various scaling. The analysis can help to maximize the utilization of the parallel and distributed hardware [41].

This work summarizes and discusses several cases for the appearance of superlinearity. Superlinear speedup in non-persistent algorithms appears due to a smaller number of executed operations. Mainly the superlinear speedup performance in persistent algorithms occurs due to the increased cache resources in the parallel computer architectures, the prefetching of shared variables in shared memory organization, or better scheduling in heterogeneous environments. The effects of the shared memory architectures also impact the performance behavior of the granular and scalable algorithms. All these analyses will guide the developers of parallel implementation

not only how to parallelize a given problem, but to choose the most appropriate environment and scaling type in order to achieve the maximal speedup.

Additionally, this analysis opens many challenges, such as finding a correlation among parallel hardware's architecture and organization, a certain form of a parallelized algorithm, a parallelization technique, the server load and input problem size, and other possible factors that impact the existence of a superlinear speedup.

Further focus will be towards modeling the speedup by considering all these factors, as well as to determine an analytical relation of a complex computer system that will enable the conditions for superlinearity. Additionally, our challenge is to model the multidimensional space of superlinearity, which will determine the value of superlinearity by considering the problem size region and the region of the number of processors. Since this paper focuses on granular high performance algorithms, we would analyze and define the taxonomy for *scalable* algorithms, in which many tasks that are coming, are scattered among parallel processing units.

ACKNOWLEDGMENT

This work is supported by the European Union's Horizon 2020 research and innovation programme under the grant agreement 644179 ENTICE: dEcentralized repositories for traNsparent and efficienT vIrtual maChine opErations.

The authors would like to acknowledge networking support by the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

REFERENCES

[1] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013. doi: 10.1111/j.1475-3995.2012.00862.x

[2] J. L. Gustafson, "Reevaluating Amdahl's law," *Communication of ACM*, vol. 31, no. 5, pp. 532–533, May 1988. doi: 10.1145/42411.42415. [Online]. Available: http://doi.acm.org/10.1145/42411.42415

[3] X. Ye, W. Dong, P. Li, and S. Nassif, "Hierarchical multialgorithm parallel circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 45–58, Jan 2011. doi: 10.1109/TCAD.2010.2067870

[4] J. Rufino, A. I. Pereira, and J. Pidanic, "copssa - constrained parallel stretched simulated annealing," in *Radioelektronika (RADIOELEKTRONIKA), 2015 25th International Conference*, April 2015. doi: 10.1109/RADIOELEK.2015.7129044 pp. 435–439.

[5] T. Ciamulski and M. Sypniewski, "Linear and superlinear speedup in parallel fdtd processing," in *2007 IEEE Antennas and Propagation Society International Symposium*, June 2007. doi: 10.1109/APS.2007.4396642. ISSN 1522-3965 pp. 4897–4900.

[6] A. Gupta and D. Milojicic, "Evaluation of hpc applications on cloud," in *Open Cirrus Summit (OCS), 2011 Sixth*, Oct 2011. doi: 10.1109/OCS.2011.10 pp. 22–26.

[7] S. A. Tsaftaris, "A scientist's guide to cloud computing," *Computing in Science Engineering*, vol. 16, no. 1, pp. 70–76, Jan 2014. doi: 10.1109/MCSE.2014.12

[8] L. Liu, M. Zhang, Y. Lin, and L. Qin, "A survey on workflow management and scheduling in cloud computing," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014. doi: 10.1109/CCGrid.2014.83 pp. 837–846.

[9] M. Gusev and S. Ristov, "Superlinear speedup in Windows Azure cloud," in *Cloud Networking (IEEE CLOUDNET), 2012 IEEE 1st International Conference on*, Paris, France, 2012, pp. 173–175.

[10] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings*, vol. 30. AFIPS Press, Reston. Va., Atlantic City, N.J., Apr. 18-20 1967. doi: 10.1145/1465482.1465560 pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[11] A. H. Karp and H. P. Flatt, "Measuring parallel processor performance," *Commun. ACM*, vol. 33, no. 5, pp. 539–543, May 1990. doi: 10.1145/78607.78614. [Online]. Available: http://doi.acm.org/10.1145/78607.78614

[12] Y. Shi, "Reevaluating amdahl's law and gustafson's law," Computer Sciences Department, Temple University, Tech. Rep. MS:38-24, Oct. 1996.

[13] M. Gusev and S. Ristov, "A superlinear speedup region for matrix multiplication," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 11, pp. 1847–1868, 2013. doi: 10.1002/cpe.3102. [Online]. Available: http://dx.doi.org/10.1002/cpe.3102

[14] ——, "Resource scaling performance for cache intensive algorithms in Windows Azure," in *Intelligent Distributed Computing VII*, ser. SCI, F. Zavoral, J. J. Jung, and C. Badica, Eds. Springer International Publishing, 2014, vol. 511, pp. 77–86. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-01571-2_10

[15] J. L. Hennessy and D. A. Patterson, "Computer Architecture, Fifth Edition: A Quantitative Approach," MA, USA, 2012.

[16] M. Otte and N. Correll, "C-forest: Parallel shortest path planning with superlinear speedup," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 798–806, June 2013. doi: 10.1109/TRO.2013.2240176

[17] A. F. P. Camargos, R. M. S. Batalha, C. A. P. S. Martins, E. J. Silva, and G. L. Soares, "Superlinear speedup in a 3-d parallel conjugate gradient solver," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1602–1605, March 2009. doi: 10.1109/TMAG.2009.2012753

[18] G. Velkoski, S. Ristov, and M. Gusev, "Loosely or tightly coupled affinity for matrix - vector multiplication," in *Information Communication Technology Electronics Microelectronics (MIPRO), 2013 36th International Convention on*. Opatija, Croatia: IEEE, May 2013. ISBN 978-953-233-076-2 pp. 228–233.

[19] L. Djinevski, S. Ristov, and M. Gusev, "Superlinear speedup for matrix multiplication in GPU devices," in *ICT Innovations 2012*, ser. Advances in Intelligent Systems and Computing, S. Markovski and M. Gusev, Eds. Springer Berlin Heidelberg, 2013, vol. 207, pp. 285–294. ISBN 978-3-642-37168-4. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37169-1_28

[20] N. Anchev, M. Gusev, S. Ristov, and B. Atanasovski, "Intel vs AMD: Matrix multiplication performance," in *Information Communication Technology Electronics Microelectronics (MIPRO), 2013 36th International Convention on*. Opatija, Croatia: IEEE, May 2013. ISBN 978-953-233-076-2 pp. 182–187.

[21] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011. doi: 10.1002/cpe.1631. [Online]. Available: http://dx.doi.org/10.1002/cpe.1631

[22] G. Kosec and M. Depolli, "Superlinear speedup in OpenMP parallelization of a local PDE solver," in *MIPRO, 2012 Proceedings of the 35th International Convention*, 2012, pp. 389–394.

[23] J. L. Gustafson, "Fixed time, tiered memory, and superlinear speedup," in *Distributed Memory Computing Conference, 1990., Proceedings of the Fifth*, vol. 2, Apr 1990. doi: 10.1109/DMCC.1990.556383 pp. 1255–1260.

[24] I. A.-S. Ibrahim, H.-W. Loidl, and P. Trinder, "High-performance cloud computing for symbolic computation domain," *Journal of Computations & Modelling*, vol. 6, no. 1, pp. 107–133, 2016. [Online]. Available: http://www.scienpress.com/journal_focus.asp?main_id=58&Sub_id=IV&Issue=1771

[25] N. Theera-Ampornpunt, S. G. Kim, A. Ghoshal, S. Bagchi, A. Grama, and S. Chaterji, "Fast training on large genomics data using distributed support vector machines," in *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2016. doi: 10.1109/COMSNETS.2016.7439943 pp. 1–8.

[26] P. E. McKenney, "Retrofitted parallelism considered grossly sub-optimal," in *Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342788.2342801

[27] J. Ichnowski and R. Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, Oct 2014. doi: 10.1109/TRO.2014.2331091

[28] S. Priyadarshi, C. S. Saunders, N. M. Kriplani, H. Demircioglu, W. R. Davis, P. D. Franzon, and M. B. Steer, "Parallel transient simulation of multiphysics circuits using delay-based partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1522–1535, Oct 2012. doi: 10.1109/TCAD.2012.2201156

[29] M. Monagan and R. Pearce, "Parallel sparse polynomial division using heaps," in *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, ser. PASCO '10. New York, NY, USA: ACM, 2010. doi: 10.1145/1837210.1837227. ISBN 978-1-4503-0067-4 pp. 105–111.

[30] R. D. Phillips, L. T. Watson, and R. H. Wynne, "An smp soft classification algorithm for remote sensing," in *Proceedings of the 19th High Performance Computing Symposia*, ser. HPC '11. San Diego, CA, USA: Society for Computer Simulation International, 2011, pp. 104–110. [Online]. Available: http://dl.acm.org/citation.cfm?id=2048577.2048591

[31] P. Peschlow, A. Voss, and P. Martini, "Good news for parallel wireless network simulations," in *Proceedings of the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '09. New York, NY, USA: ACM, 2009. doi: 10.1145/1641804.1641828. ISBN 978-1-60558-616-8 pp. 134–142. [Online]. Available: http://doi.acm.org/10.1145/1641804.1641828

[32] M. Gusev and S. Ristov, "The optimal resource allocation among virtual machines in cloud computing," in *Proceedings of The 3rd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012)*, Nice, France, 2012, pp. 36–42.

[33] S. S. Bokhari, "Parallel solution of the subset-sum problem: An empirical study," *Concurr. Comput.: Pract. Exper.*, vol. 24, no. 18, pp. 2241–2254, Dec. 2012. doi: 10.1002/cpe.2800

[34] J. Maqbool, S. Oh, and G. C. Fox, "Evaluating arm hpc clusters for scientific workloads," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5390–5410, 2015. doi: 10.1002/cpe.3602 CPE-14-0161.R2. [Online]. Available: http://dx.doi.org/10.1002/cpe.3602

[35] S. Ristov, K. Cvetkov, and M. Gusev, "Implementation of a scalable l3b balancer," *Scalable Computing: Practice and Experience*, vol. 17, no. 2, pp. 79–90, 2016. doi: 10.1109/TE.2014.2327007

[36] S. Ristov, M. Gusev, L. Djinevski, and S. Arsenovski, "Performance impact of reconfigurable L1 cache on GPU devices," in *Computer Science and Information Systems (FedCSIS 2013), Federated Conference on*, Krakow, Poland, Sep. 2013, pp. 507 – 510.

[37] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, April 2014. doi: 10.1109/TCC.2014.2314655

[38] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, July 2014. doi: 10.1109/TPDS.2013.238

[39] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013. doi: 10.1109/IPDPS.2013.61. ISSN 1530-2075 pp. 67–78.

[40] Supercomputers, "Top500," [retrieved: April, 2015]. [Online]. Available: http://www.top500.org/

[41] X. Ye, W. Dong, P. Li, and S. Nassif, "Maps: Multi-algorithm parallel circuit simulation," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '08, 2008. doi: 10.1109/ICCAD.2008.4681554. ISBN 978-1-4244-2820-5 pp. 73–78.