Superlinear Speedup for Matrix - Vector Multiplication in Different Multiprocessors

Sasko Ristov, Goran Velkoski and Marjan Gusev
Ss. Cyril and Methodius University,
Faculty of Information Sciences and Computer Engineering,
Skopje, Macedonia
Email: sashko.ristov@finki.ukim.mk, velkoski.goran@gmail.com, marjan.gushev@finki.ukim.mk

***Abstract*—In this paper we analyze the performance of sequential and parallel implementation of matrix vector multiplication algorithm on a multi-chip multi-core multiprocessor. We analyze three different multiprocessors: single-chip multi-core, multi-chip single-core and multi-chip multi-core multiprocessor. The results show that the speed and speedup depends on cache organization of multiprocessors despite the same number of cores. Although Gustafson's Law limits the speedup for parallel implementation on the number of used processors (CPU cores), we achieved a region for problem size where superlinear speedup (speedup greater than the number of processors) is achieved for each multiprocessor.**

***Index Terms*—Gustafson's law, shared memory multiprocessor, high performance computing**

I. Introduction

Nowadays, many linear algebra algorithms are used which requires huge computing power since they are computation intensive and memory demanding. The former means that the operations increase proportionally with greater problem size. The latter refers to increased storage requirements when problem size rises, which implies to increased computation, as well. The memory was the most expensive part of the computer and the algorithm lacks memory space at the beginning. But, the time is changing and todays computers lacks of computing resources instead of memory storage.

Speeding up an algorithm execution is imperative and many techniques exist: reducing the number of program steps computations (operations), reducing the calculations, speeding the hardware, adding more computing resources, using some methods for parallelization, etc.

Adding cache memory between the CPU and main memory speeds up the memory access [1], especially when the algorithm reuses the same data many times. Cache memory helps with data locality also. In this paper we use one of the most common linear algebra algorithms, i.e. matrix-vector multiplication (MVM) to analyze its behavior in sequential and parallel execution. We use it since this algorithm can be easy parallelized and it is cache intensive algorithm [2], i.e. the number of accesses of each element the vector depends of the problem size.

The paper is organized as follows. Section II analyzes the speed and speedup of the MVM algorithm. The different architectures of multiprocessors used in the experiments are described in Section III. In Section IV we present the results of the experiments executed sequential and parallel on three different multiprocessors. Finally, we conclude our work and present our plan for future research in Section VI.

II. Speed and Speedup Analysis

In this section we present the sequential and parallel implementation of MVM algorithm and analyze their speed and speedup.

A. MVM Algorithm

MV multiplication algorithm is defined in (1).

$$C_{N\cdot 1} = A_{N\cdot N} \cdot B_{N\cdot 1} \tag{1}$$

For simplification, we use squared matrix $A_{N\cdot N}$. We use double precision numbers with 8 bytes each for each matrix and vector element.

B. MVM Algorithm Implementations

We use sequential and parallel implementation of the MVM algorithm. Sequential implementation consists of one thread which multiplies the whole matrix $A_{N\cdot N}$ and vector $B_{N\cdot 1}$.

In parallel implementation, each thread multiplies the row block matrix $A_{N\cdot N/c}$ and the whole vector $B_{N\cdot 1}$, where $c \in \{2, 4, 8, 16\}$ denotes the total number of parallel threads and used CPU cores. We test the performance of the matrix and vector size that produce the best efficiency, i.e. the multiple of 16.

From memory point of view, each inner product performs M memory reads for matrix elements. For N inner products $M \cdot N$ memory reads are needed. Added M reads for the vector, the memory complexity is:

$$O(M \cdot N + M) = O(M \cdot N) \tag{2}$$

The memory complexity is calculated based on memory accesses because this is the most expensive operation. The MVM algorithm operates on two arrays representing the matrix and the vector respectively. Similar to computations, both the sequential and parallel implementations should store matrix $A_{N\cdot N}$, and vectors $B_{N\cdot 1}$ and $C_{N\cdot 1}$, or total $N^2 + 2 \cdot N$ elements.

C. MVM Speed

Let's analyze the algorithm execution. In sequential implementation, the single thread multiplies each matrix row with the vector executing $N \cdot N$ multiplications and $N \cdot N$ additions, or total $2N^2$ floating point operations.

Parallel implementation of the algorithm is realized by P cores such that each core multiplies N/P rows executing $N/P \cdot N$ multiplications and $N/P \cdot N$ additions, or total $2N^2/P$ floating point operations. Therefore, all P cores will execute total $2N^2/P \cdot P$ or the same $2N^2$ as sequential implementation.

That is, the speed $V(P)$ of the sequential and parallel implementation of the MVM algorithm is determined with one equation (3), where $T(P)$ denotes the execution time when all threads finish with their computations.

$$V(P) = \frac{2 \cdot N^2}{T(P) \cdot 10^9} \tag{3}$$

We express the speed in gigaflops.

D. Speedup Limits

Gustafson's [3] concludes that linear speedup is maximum speedup in a parallel system. He presents the domain of computing performance pattern in the log scaled Fig. 1. Fixed-size speedup (Amdahl's law) bounds speedup to the sequential part of the algorithm.

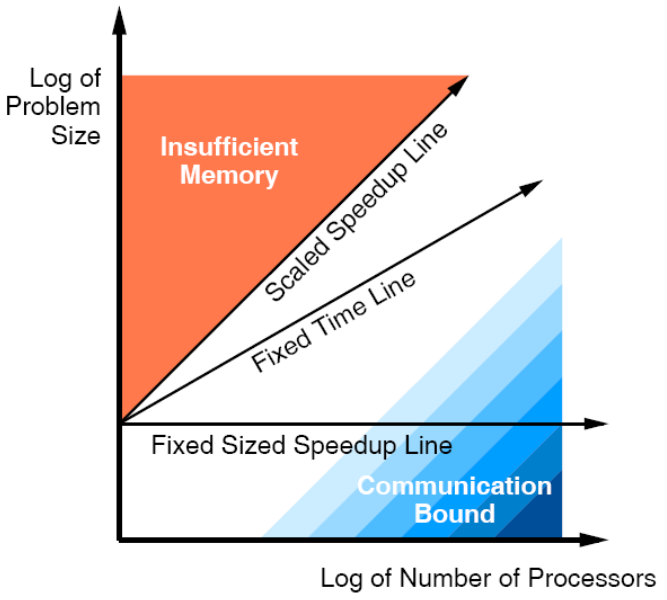

Fig. 1. Ensemble computing performance pattern [3]

A typical curve for fixed size speedup (Amdahl's Law) is presented in the log scale Fig. 2 bounded by the superlinear speedup.

In this paper we are interested in speedup behavior for parallel implementation of MVM algorithm in different multiprocessors varying matrix and vector size in range $N \in [16, 1800]$. The speedup is defined as the ratio of speeds of parallel and sequential executions, as defined in (4).

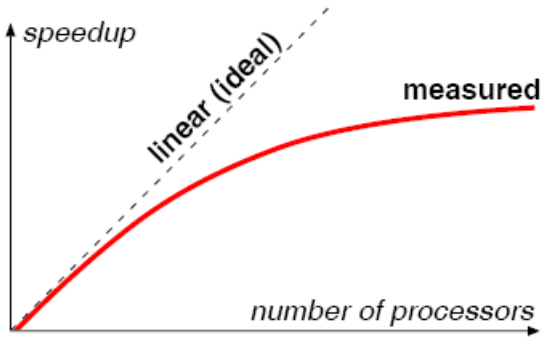

Fig. 2. Typical Speedup Curve [4]

$$S(P) = \frac{V(P)}{V(1)} \tag{4}$$

Since each processor in parallel implementation executes P times smaller number of operations, we expect that they will finish P times faster than sequential implementation of the MVM algorithm. That is, the speedup $S(P)$ has maximum value P as referred in (5).

$$S(P) \leq P \tag{5}$$

Therefore, the maximum expected speedup for different size N is depicted in Fig. 3.

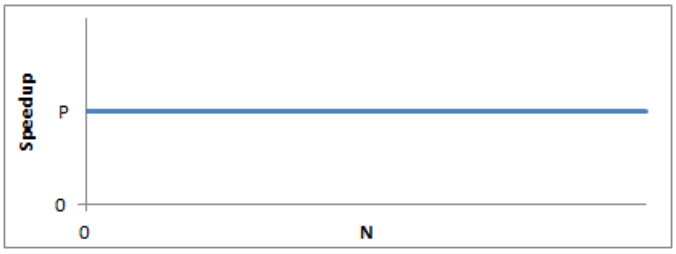

Fig. 3. Expected theoretical speedup

Since $P > 1$ for parallel implementation, speed should be in the range:

$$1 < S(P) \leq P$$

III. The Multiprocessor Environment and Experiments

In this section we present different multiprocessors used in the experiments.

A. Testing Environment

The architecture of the multiprocessor is used as platform for each experiment is depicted in Fig. 4. It consists of four CPUs AMD Opteron 8347 chips, each with four cores. Therefore, sixteen cores are available to be utilized with maximum of sixteen threads for parallel execution. Each of the

cores have 64KB of dedicated L1 cache and 512KB dedicated L2 cache. Additionally, 2MB of shared per CPU L3 cache per chip is present.

Different parallel implementations have different cache organization for each experiment. Each thread has its own private L1 and L2 caches and shares main memory. The difference is in L3 cache, sometimes it is shared per threads, sometimes it is private per core, and sometimes it is shared but per two cores per chip. This organization is explained more detailed in the following paragraphs.

B. The Experiments

We define four parallel and one sequential experiments of the MVM algorithm:

- Experiment 1 - Parallel implementation of MVM algorithm on two cores (on two separate chips by one core)
- Experiment 2 - Parallel implementation of MVM algorithm on four cores (on four separate chips by one core)
- Experiment 3 - Parallel implementation of MVM algorithm on eight cores (on four separate chips by two cores)
- Experiment 4 - Parallel implementation of MVM algorithm on sixteen cores (on four separate chips by four cores)

We chose to employ each thread on different core for all of the scenarios because this is confirmed to be best case for OpenMP parallelism. Hence, OpenMPs GOMP_CPU_AFFINITY is used for thread binding to a specific chip and core. Each experiments varies the matrix and vector size from 16 to 1800. Each test case is executed at least 10s in order to achieve reliable results.

IV. The Results of the Experiments

In this section we present the results for the speedup of each experiment.

A. Experiment 1 - 2 Cores

Experiment 1 consists of executing parallel implementation of MVM algorithm with two threads on 2 cores on the separate chip. That is, L3 cache is private per chip, but also per core. The speedup compared to sequential execution is depicted on Fig. 5.

We observe the three different regions going from left to right in Fig. 5, i.e. increasing the matrix and vector size N. The speedup grows in the first region from the left. Then the speedup is superlinear in the second region, and even in the third region on the right the speedup value is as expected, i.e. near, but lower than $P = 2$.

B. Experiment 2 - 4 Cores

In this experiment we use 4 cores on separate chips, i.e. not only that L3 cache is private per chip, but also per core. The speedup of MVM algorithm executed by using four parallel threads is depicted on Fig. 6.

The same issues are observed for the speedup $S(4)$ as the speedup S(2). However, the superlinear region is wider for $S(4)$ compared to $S(2)$.

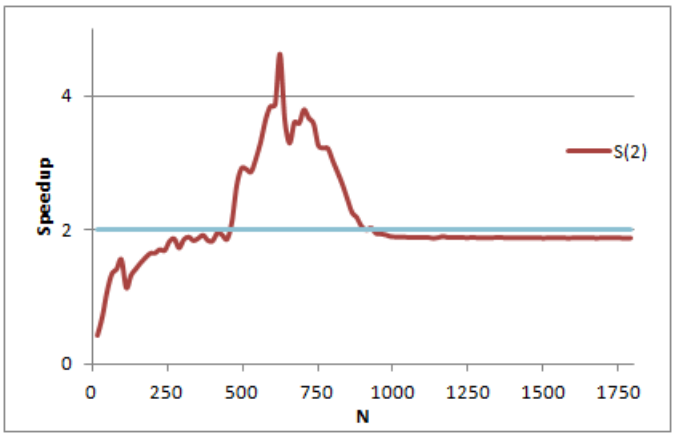

Fig. 5. Speedup for first experiment scenario

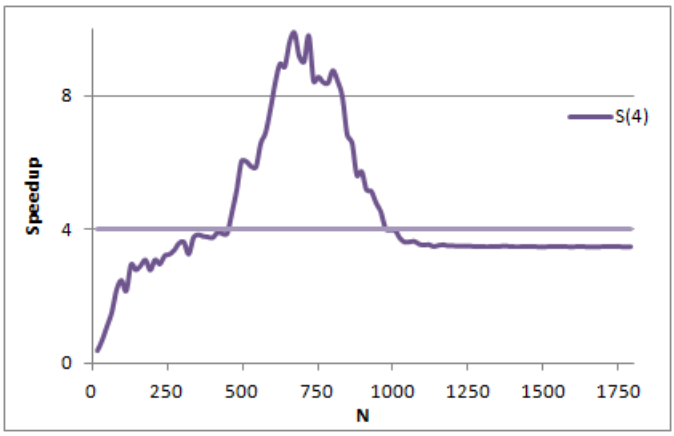

Fig. 6. Speedup for second experiment scenario

C. Experiment 3 - 8 Cores

Experiment 3 is executed on multi-chip multi-core multiprocessor with semi shared L3 cache. We also observe similar regions for this experiment as depicted in Fig. 7. But although the speedup is constant in the third region (right), it is much smaller than limit P.

We explain this with the fact that this experiment uses shared L3 caches per two cores and therefore their total capacity is smaller. Therefore, more cache misses are being generated and the speedup is smaller than expected.

D. Experiment 4 - 16 Cores

Finally, the Experiment 4 is executed on multi-chip multi-core multiprocessor with shared L3 cache per chip and 4 cores. We also observe similar regions for this experiment as depicted in Fig. 8. The third region is more emphasized with decreasing the performance, although the speedup is constant there, it is much smaller than limit P and smaller than $S(8)$.

V. Multiprocessors Comparison

In this section we compare the speed and speedup in each experiment, i.e. for different multiprocessor.

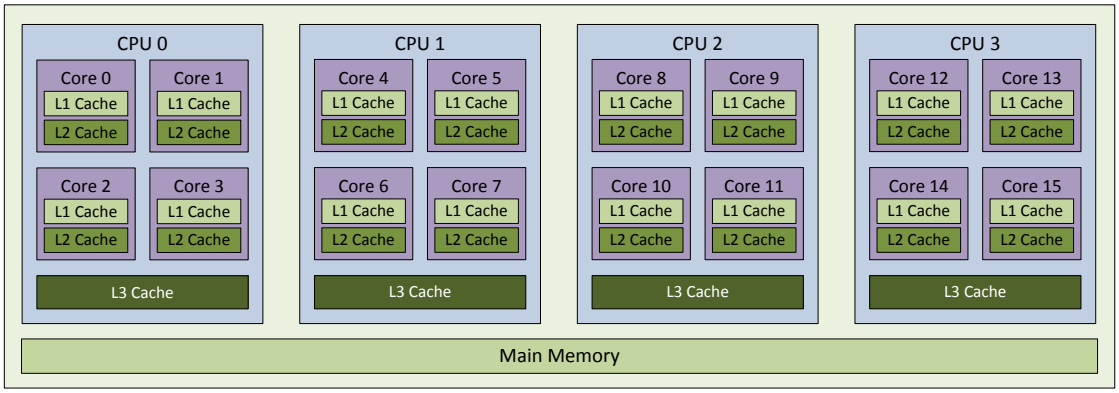

Fig. 4. Experiment platform architecture

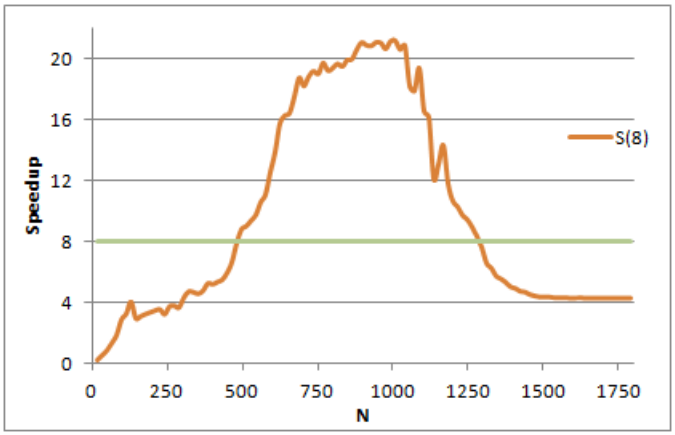

Fig. 7. Speedup for third experiment scenario

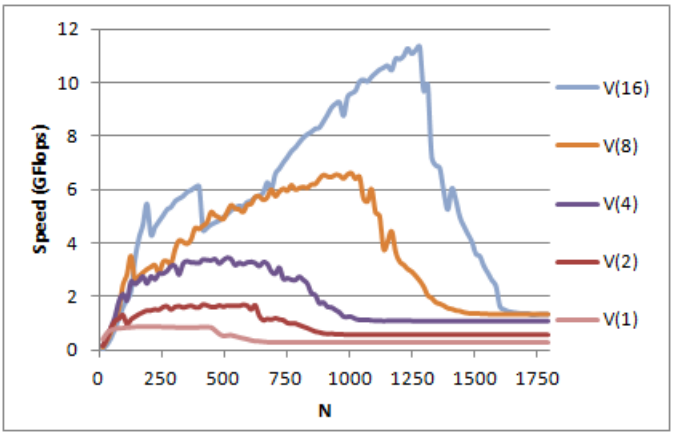

Fig. 9. Speed comparison

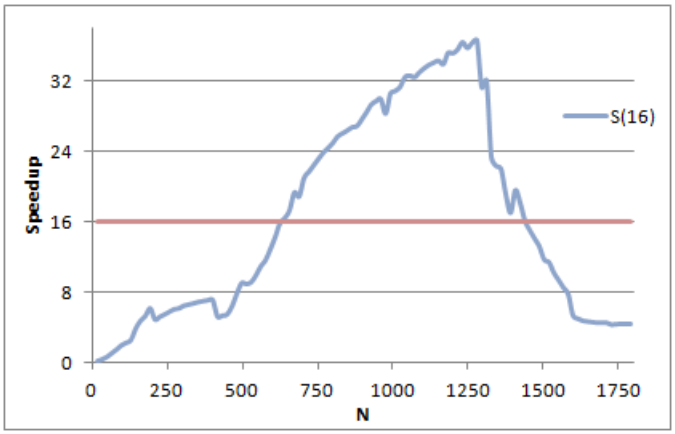

Fig. 8. Speedup for forth experiment scenario

A. The Speed

Speed is defined as number of floating point operation in second and the results of the all experiments are depicted in Fig 9.

We clearly observe that $V(P) > V(R)$ when $P > R$ for each valule of N, that is, using more resources (cores) will provide greater speed. Another important fact is that the region where the speed grows is wider for the experiments with the greater number of cores.

B. The Speedup

The more important results are obtained for the speedup in all experiments, depicted in Fig 10.

We observe that superlinear region for the experiment with greater number of processors P is wider than the counterpart with smaller number of processors, although they start earlier for the experiments with greater P. Even more, not only that superlinear speedup region is wider, but it is also higher, i.e. the maximum speedup is achieved with the greater number of processors.

VI. Conclusion and Future Work

In this paper we analyze the performance behavior of MVM algorithm on different multiprocessors with different cache organization and resources. We vary the matrix and vector

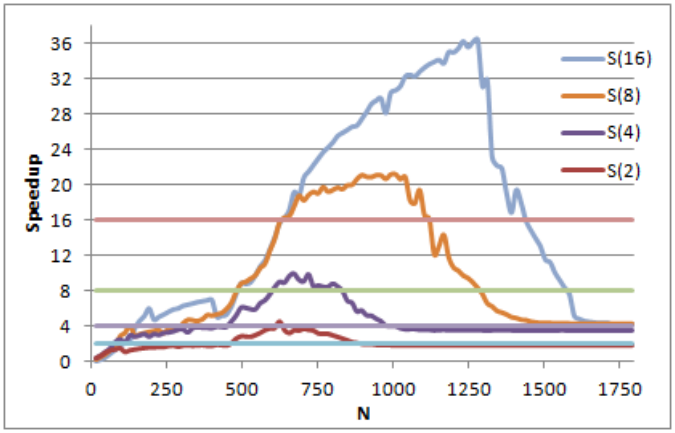

Fig. 10. Speedup comparison

size by 16 in order to have the same maximum efficiency for the algorithm and to be comparable all experiments.

The results show that the performance varies for different matrix and vector size. The speed rises until it reaches its maximum value, then starts decreasing and then saturates. All this regions are moved in the right with scaling the resources by 2. The speed difference is greater in the region where the speeds rise compared to the region on the right where the speed saturates.

The speedups has similar curves as the speed for particular experiment. Multiprocessors with private L3 cache provide almost linear speedup in the right region when the speedup saturates. The speedup also saturates for multiprocessors with shared L3 cache, but it is much lower than theoretical value P as the number of processors.

In each experiment we achieved a region with superlinear speedup. These regions are wider and higher for the experiments with greater number of processors.

We analyzed the MVM algorithm as one of the most common linear algebra algorithms. But, more interesting would be matrix matrix multiplication since it is cache intensive algorithm where each element is accessed $2 \cdot N$ times and the memory organization of the matrices is very different than the one in MVM algorithm.

REFERENCES

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

[2] S. Ristov, M. Gusev, M. Kostoska, and K. Kiroski, "Virtualized environments in cloud can have superlinear speedup," in *Proceedings of the Fifth Balkan Conference in Informatics*, ser. BCI '12. New York, NY, USA: ACM, 2012, pp. 8–13.

[3] J. Gustafson, G. Montry, and R. Benner, "Development of parallel methods for a 1024-processor hypercube," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 4, pp. 532–533, July 1988.

[4] J. L. Gustafson, "The consequences of fixed time performance measurement," in *Proceedings of the Hawaii International Conference on System Sciences*, vol. 25, 1992, p. 113.