Neural, Parallel, and Scientific Computations 22 (2014) 623-638

# PERFORMANCE SCALABILITY AND ENERGY CONSUMPTION ON A SHARED MEMORY PLATFORM

**E. M. KARANIKOLAOU AND M. P. BEKAKOS**
**Dept. of Electrical & Computer Engineering, Democritus university of Thrace, 67100 Xanthi, Greece**

**Abstract.** Herein, the performance evaluation of a shared memory computer complex, in conjunction with the total consumed energy, is investigated. The shared platform under investigation is evaluated for the power its processors/cores demand at the idle and fully utilized state. In proportion to the parallelized percentage each time, the estimations of the theoretical model were compared to the experimental results achieved on the basis of the performance/power and performance/energy ratio metrics. An analytical formula for evaluating the experimental energy consumption has been developed, while the experimental vehicle was a widely known algorithm with different parallelization percentages.
**Keywords** - performance evaluation, power/energy consumption, shared memory platform, superlinear speedup

## 1. INTRODUCTION

According to Amdahl's law [1], an algorithm's sequential computation considerably limits the maximum potentially achievable speedup. This implies that any sequential execution will severely diminish the overall performance scalability for parallel implementations regardless the amount of additional computing resources.

Amdahl's law was used as an argument against massively parallel processing (MPP). On the other hand, Gustafson's law, proposed in 1988 [5], was used to justify MPP. However, a careful analysis reveals that these two laws are identical.

Nowadays, computer architects face a new significant challenge, beyond performance, there is need for energy efficiency. Power/energy becomes an apparent obstacle to realize performance scaling; thus, low power techniques and algorithms for multicore systems, such as DVS (Dynamic Voltage Scaling) have been a major design trend over the last years [8, 9]. Moreover, in an era where energy is a very valuable resource, there is a crucial need to define the tradeoffs between performance and energy consumption on shared memory platforms. Computer engineers into their forthcoming multiprocessor designs should, very carefully, take into account that architecture's power consumption does not exceed its power capability [2]. In other words, the level of consumption required by each processor would limit the maximum number to be utilized; this would force the designer to strive for maximizing each processor's power efficiency. While Amdahl's law mainly focuses on performance scalability, current interest lies on the power scalability/energy efficiency of multiprocessor designs. Hence, an augmentation of Amdahl's law is required to cope with the new architectural challenges taking into account the implication of energy scalability.

Received October 15, 2014 

A formula which describes the performance of a multiprocessor platform in conjunction with the consumed power was introduced by Woo and Lee [6]. Therein, the estimation of performance/power was based upon the Amdahl's law assuming an ideal system without overheads. Herein, an analytical formula for a shared memory platform has been developed in order to experimentally evaluate the consumed power; the experimental results achieved were compared to the estimations of the theoretical model [6] on the basis of the performance/power and performance/energy ratio metrics

## 2. AMDAHL'S AND GUSTAFSON'S LAWS COMPARISON

Amdahl's law has been widely applied for the performance estimation of parallel programs, initially as an argument against MPP. The parallel processing reputation was consolidated by Gustafson's law and various experimental results which justified MPP.

The essential difference of two laws lies on the serial percentages implied, which by many researchers were considered as *identical*. A careful analysis of both laws reveals that these two serial percentages are related with a simple equation [4]. By altering the serial percentage depended on the number of processors in Gustafson's law, to an independent serial percentage, the resulting formula proves to be identical to Amdahl's formula. In other words, there is only one law with two slightly varying formulations.

Another important issue often bypassed is the prerequisite to applying Amdahl's law. For the correct application of this law, the serial algorithm should retain its structure, so that the same number of instructions to be processed in both serial/parallel implementations for the same input. It is quite common the parallel implementation to be straightforward formed out of the serial one of the same algorithm. It has been proved that there is a class of non-structure persistent serial algorithms, namely algorithms which lose their structure when are been partitioned. In this class parallel implementations formed out from serial ones can exhibit quite peculiar performance. For such cases the law is open to abuse.

### 2.1. Equivalence of Laws

Let us consider both laws formulas for the theoretical speedup and the respective serial percentages they refer to, as well as the formula linearly connecting these two percentages. The following parameters have been defined,

| | |
|---|---|
| $T_{S_{eq}(n)}$ | Uniprocessor execution time complexity of a program's serial segment. |
| $T_{P_{ar}(n)}$ | Uniprocessor execution time complexity of a program's parallel segment. |
| $T\left(\frac{P_{ar}(n)}{p}\right)$ | p-processor execution time complexity of a program's parallel segment. |
| $T_{S_{eq}(n)} + T_{P_{ar}(n)}$ | Uniprocessor total execution time complexity of a program including both serial/parallel segments. |
| $T_{S_{eq}(n)} + T\left(\frac{P_{ar}(n)}{p}\right)$ | p-processor total execution time complexity of a program including both serial/parallel segments. |

$$f_{G[Seq(n)]} = \frac{T_{S_{eq}(n)}}{T_{S_{eq}(n)} + T\left(\frac{P_{ar}(n)}{p}\right)}$$ The *scaled* percentage of a program's serial segment

$$1 - f_{G[Seq(n)]} = \frac{T\left(\frac{P_{ar}(n)}{p}\right)}{T_{S_{eq}(n)} + T\left(\frac{P_{ar}(n)}{p}\right)}$$ The *scaled* percentage of a program's parallel segment

$$f_{S_{eq}(n)} = \frac{T_{S_{eq}(n)}}{T_{S_{eq}(n)} + T_{P_{ar}(n)}}$$ The *non-scaled* percentage of a program's serial segment

$$1 - f_{S_{eq}(n)} = \frac{T_{P_{ar}(n)}}{T_{S_{eq}(n)} + T_{P_{ar}(n)}}$$ The *non-scaled* percentage of a program's parallel segment

Note that, p appears in both percentages of the scaled percentage of program's serial/parallel segment, while it is not in the non-scaled percentage case.

According to Amdahl's law the formula estimating the maximum theoretical speedup is:

$$S_{pth}(n,p) \le \frac{T_{S_{eq}(n)} + T_{P_{ar}(n)}}{T_{S_{eq}(n)} + \frac{T_{P_{ar}(n)}}{p}} \tag{2.1}$$

which using the non-scaled percentages is transformed to:

$$S_{pth}(n,p) \le \frac{1}{f_{S_{eq}(n)} + \left[1 - f_{S_{eq}(n)}\right]/p} \tag{2.2}$$

hence,

$$S_{pth}(n,p) \le_{p\to\infty} \frac{1}{T_{S_{eq}(n)} \Big/ (T_{S_{eq}(n)} + T_{P_{ar}(n)})} \tag{2.3}$$

which, when $f_{Seq(n)}$ is taking values close to zero, the results achieved justify the argument against massively parallel processing.

On the other hand, Gustafson proposed a formula calculating the total parallel time complexity using p processors,

$$T_{S_{eq}(n)} + T_{P_{ar}(n)} = f_{G[Seq(n)]} + \left(1 - f_{G[Seq(n)]}\right) \times p$$

$$T_{S_{eq}(n)} + T\left(\frac{P_{ar}(n)}{p}\right) = f_{G[Seq(n)]} + \left(1 - f_{G[Seq(n)]}\right) = 1$$

$$S_{pth}(n,p) = \left(f_{G[Seq(n)]} + \left(1 - f_{G[Seq(n)]}\right) \times p\right) \Big/ \left(f_{G[Seq(n)]} + \left(1 - f_{G[Seq(n)]}\right)\right) \tag{2.4}$$

which is formula:

$$S_{pth}(n,p)=\frac{(T_{S_{eq}(n)}+T\left(\frac{P_{ar}(n)}{p}\right))\left(\frac{T_{S_{eq}(n)}}{T_{S_{eq}(n)}+T\left(\frac{P_{ar}(n)}{p}\right)}+\frac{T\left(\frac{P_{ar}(n)}{p}\right)}{T_{S_{eq}(n)}+T\left(\frac{P_{ar}(n)}{p}\right)}\text{x p}\right)}{T_{S_{eq}(n)}+T\left(\frac{P_{ar}(n)}{p}\right)} \quad (2.5)$$

The mistake was hidden in the misunderstanding of the two laws, namely the misuse of $f_{G[Seq(n)]}$ in place of

$$f_{S_{eq}(n)}=\frac{T_{S_{eq}(n)}}{T_{S_{eq}(n)}+T_{P_{ar}(n)}}$$

in Amdahl's $S_{pth}(n,p)$, which is formula (2.2).

These two laws are *equivalent* and the two serial percentages are related with the following equation, without introducing $(T_{Seq(n)} + T_{Par(n)})$,

$$f_{S_{eq}(n)}=\frac{1}{1+\frac{(1-f_{G[Seq(n)]})\times p}{f_{G[Seq(n)]}}} \quad (2.6)$$

Hence, for the correct use of Amdahl's law the $f_{G[Seq(n)]}$ should be altered to $f_{Seq(n)}$ through (2.6). More thorough and further analysis can be found in [10].

## 3. PERFORMANCE/WATT AND PERFORMANCE/JOULE METRICS

Many of the evaluation metrics describe a parallel system's performance regarding the speedup achieved in terms, either of time complexity or instructions per processor cycle as compared to the serial implementation.

Today, beyond performance, computer engineers face another significant challenge: energy efficiency. Parallel systems should be carefully designed so that their power consumption does not exceed their power capabilities. Hence, new evaluation metrics should be introduced measuring overall performance, while taking into account power and energy as well.

The Performance/Watt and Performance/Joule metrics [6] will be discussed; these metrics are based on Amdahl's law introducing in his speedup formula the power/energy concept targeting to the performance evaluation of an ideal parallel system. These metrics will be set herein following the time basis concept for the augmentation of Amdahl's law; the theoretical estimations will be investigated and attempted to be approximated with real time experimental results. An experimental scenario will be executed on a multiprocessor platform and the carrying vehicle will be a parallel algorithm of specific structure.

The total energy/power consumed by the shared memory platform in hand, for the complete execution of a parallel algorithm, is calculated. A primary target pursued when developing the experimental scenario was the modeling and the introduction of the overheads due to the interprocessor communication, not accounted in [6]; any interprocessor communication or non-parallel execution will rapidly diminish the performance scalability regardless the amount of additional computing resources.

Apparently, power is becoming more critical factor than performance especially in the scaling up of multiprocessor systems.

### 3.1 Amdahl's Law Extension

Amdahl's law demonstrated a simple analytical model to provide computer designers and researchers a useful tool for the better realization of speedup scaling. Current technological advances require this law's extension to account for the power scaling and the energy efficiency implications in the multiprocessor architectural designs [14-16]. Herein, the formulas for Performance/Joule and Performance/Watt [6] will be investigated for the case of energy-balanced shared memory platforms.

Amdahl's formula estimating the maximum achievable theoretical speedup, $S_{pth}(n,p)$, in case of solving a problem of size n, using a complex of p processors is (2.2), where, $f_{Seq(n)}$, is the time complexity fraction regarding the serial computation of the algorithm. To model the power consumption for a parallel system, a new parameter, k, is defined; it shows the fraction of power a processor consumes in *idle* state ($0 \le k \le 1$), under the assumption that a superscalar processor in fully active state consumes a power of 1. By default, the amount of power a *fully active* processor consumes during the serial computation stage equals 1, whereas the remaining (p-1) processors consume power (p-1)*k; hence, the parallel complex during the serial computation stage will consume total power of 1+(p-1)*k. During the parallel computation stage, p fully active processors will consume p amount of power. Since the time complexity formula for the serial and parallel code segments is given by $f_{Seq(n)}$ and $1-f_{Seq(n)}$, respectively, the formula for the average power consumption of the parallel complex is

$$W = \frac{f_{S_{eq}(n)} \times \left(1+\left(p-1\right)\times k\right) + \left(1- f_{S_{eq}(n)} \middle/ p\right) \times p}{f_{S_{eq}(n)} + \left(1- f_{S_{eq}(n)}\right) \middle/ p}$$

$$W = \frac{1+\left(p-1\right)\times k \times f_{S_{eq}(n)}}{f_{S_{eq}(n)} + \left(1- f_{S_{eq}(n)}\right) \middle/ p} \tag{3.1}$$

The metric Performance per Watt (Perf/W) (3.2) determines the achievable performance (keeping constant the cooling capacity [7]) based on the average consumed power. In fact, it is the reciprocal of energy consumed, since, according to Amdahl's law, the performance (speedup) is the reciprocal of the total execution time. Because Perf/W of a single processor execution is 1, the Perf/W benefit of a multiprocessor platform is given by

$$\frac{Perf}{W} = \frac{1}{f_{S_{eq}(n)} + \left(1- f_{S_{eq}(n)}\right) \middle/ p} \times \frac{f_{S_{eq}(n)} + \left(1- f_{S_{eq}(n)}\right) \middle/ p}{1+\left(p-1\right)\times k \times f_{S_{eq}(n)}}$$

$$= \frac{1}{1+\left(p-1\right)\times k \times f_{S_{eq}(n)}} \tag{3.2}$$

Since, as stated above, $Perf/W = 1/J(oule)$ is a related metric, *Perf/Joule* (3.3) can be defined, for evaluating the maximum achievable performance based on the given quantity of energy. Perf/J is equivalent to the reciprocal of energy-delay product [7].

$$\frac{\text{Perf}}{\text{J}} = \frac{1}{f_{S_{eq}(n)} + \left(1 - f_{S_{eq}(n)}\right)/\text{p}} \times \frac{1}{1 + (\text{p}-1) \times \text{k} \times f_{S_{eq}(n)}} \tag{3.3}$$

Perf/W and Perf/J metrics are based on Amdahl's law giving the fraction of the maximum achievable speedup to the minimum consumed power/energy. These metrics are biased and intended only for use on multicore platforms, since all unavoidable overheads resulting utilizing a conventional parallel platform are ignored. This implies that the serial part is actually increased by default, thus faulty maximizing the part with the inherent parallelism; the total time complexity when executing the algorithm is minimized resulting to a minimum consumed energy as well. The Perf/J and Perf/W evaluated are the maximum possible and always unachievable.

The primary target in this paper is the investigation of an experimental scenario applied to shared memory platforms; a classic easily parallelizable algorithm is used, the unavoidable overheads are introduced as basic elements and the experimental results obtained are further discussed.

## 4. POWER/ENERGY EFFICIENCY OF SHARED MEMOTY PLATFORMS

The main difference when compared to the model in [6] is that all unavoidable overheads are taken into account during the experimental phase. The parameter k in (3.2, 3.3) is used for normalizing the experimental values obtained; the parameters, $W_{f(ull)}$, $W_{i(dle)}$, define the power each processor consumes in full utilization and in idle state, respectively. Hence, the total energy consumed in a real system is expected to be greater than the energy estimated in the model [6].

### 4.1 Experimental Scenario

The basic characteristic of the shared memory platform is that a fully synchronous parallel operation is assumed. The job scheduling follows the scheme of *one parallel thread per available processor core*. The schematic description of the synchronous parallel operation is described in Fig. 4.1; the total energy consumption is calculated according to this scenario.

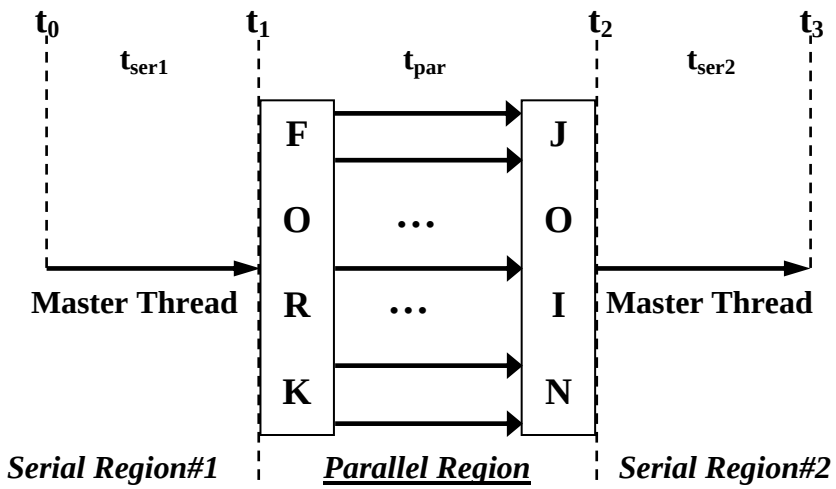



**Fig. 4.1:** Shared Memory Programming Scheme

Master thread commences at time, $t_0$, and continues to time, $t_1$, for a period of, $t_{ser1}$; it is the only core in full utilization executing the serial region of code and consuming full core power, while all other cores remain in idle state; thus, the consuming power of the whole platform consumes $W_{f/p}$, depending on the number of cores, *p*, that are utilized on each problem's execution. At time, $t_1$, using a parallel construct, the master thread creates a team of parallel threads. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads. All cores during time interval, $t_{par}$, are in full utilization state. When the team threads complete the statements in the parallel region construct, at time, $t_2$, they synchronize and terminate, leaving only the master thread until time $t_3$, in order to produce the final result. During the serial region, which is the time interval, $t_{ser2}$, master core is the only core consuming full power, while all other cores remain in idle state. The time needed for thread creation (FORK) and destroy (JOIN) is negligible; thus, it does not affect the total energy calculation.

To conclude, master core consumes power $W_f$ for the total execution time, while all other cores consume power $W_f$ during the whole computational stage, whereas the $t_{ser1}$ + $t_{ser2}$ are excluded; in $t_{ser1}$ + $t_{ser2}$ interval all other cores consume $W_i$ due to their idle status.

Let us define the time intervals,

$t_{ser1} = t_1 - t_0$ *(Serial Region#1)*, $t_p = t_2 - t_1$ *(Parallel Region)*, $t_{ser2} = t_3 - t_2$ *(Serial Region#2)*

The total energy consumed by the shared memory platform is calculated by the following formula

$$E_{total} = \underbrace{t_{par} \times W_{f/p}}_{\text{p cores fully utilized}} + \underbrace{(t_{ser1} + t_{ser2}) \times W_{f/1}}_{\substack{\text{master core fully utilized} \\ \text{(p-1) cores in idle status}}} \quad (4.1)$$

where $W_{f/p}$ is the total power consumption of all the active cores, *p*, in full utilization; while $W_{f/1}$ is the power consumption of the master proc during period of, $t_{ser}$, where it is the only proc fully utilized for executing the serial segment of code. In the expression $W_{f/1}$, since all cores belong to the same shared memory platform, the power consumed by the remaining nodes, being in idle state, is also included.

The performance/power ratio will be considered equal to the total parallel execution time measured, over the actual consumed power (for the same p-platform). Performance is represented by the total parallel execution time in order to perform a straightforward comparison with the Performance/Watt and Performance/Joule metrics given in [6]. Moreover, the performance/power ratio will be also considered equal to the speedup measured, over the actual consumed power.

The total execution time results from the parallel execution of the algorithm, summing up all measured time intervals, which are given in Fig 4.1. With the use of specialized electrical equipment (Wattmeter) for monitoring the supply rate of electrical energy, the average power consumption during the parallel execution of the algorithm was also measured. Hence, the actual Perf/W is

$$\frac{\text{Perf}}{\text{W}} = \frac{\text{t}_{\text{total(p)}}}{\text{W}}, \tag{4.2}$$

where $t_{total(p)}$ is the total parallel execution time.

Moreover, the average power consumption multiplied by a time period equals the consumed energy for that specific period under a certain electric load; thus, the total consumed energy during the parallel execution of the algorithm can be also calculated. Hence, the actual Perf/J is

$$\frac{\text{Perf}}{\text{J}} = \frac{\text{t}_{\text{total(p)}}}{\text{E}_{\text{total}}} \tag{4.3}$$

In case that performance is considered equal to the speedup measured, then the total parallel execution time, $t_{total(p)}$, should be altered to actual Speedup, $S_{p(n,p)}$, into the equations (4.2) and (4.3).

## 5. EXPERIMENTAL RESULTS

The shared memory platform's performance evaluation was carried out using OpenMP [11] standard, which provides a portable, scalable model for developers of shared memory parallel applications. The shared memory platform was a TYAN [12] advanced platform, based upon multiple Opteron [13] multicore processors with a common address space. The experimental vehicle was the parallel matrix multiplication algorithm, since it exhibits a high level inherent parallelism and offers various parallelization percentages according to the problem size selected. All the experiments were carried out in an isolation mode; namely, the platform in hand was inaccessible from other users and/or processes.

### 5.1 Total Parallel Execution Time Evaluation

In Fig. 5.1.1 is shown the total parallel execution time in seconds according to the selected problem size and the number of cores used. The representative problem sizes presented herein, concern matrix dimensions from 1K up to 5K, which progressively scale up, in steps of 1K. The evaluated values of time resulted taking the average of multiple program executions. A problem size of *mK x mK* represents the multiplication of two square matrices of size *m*.

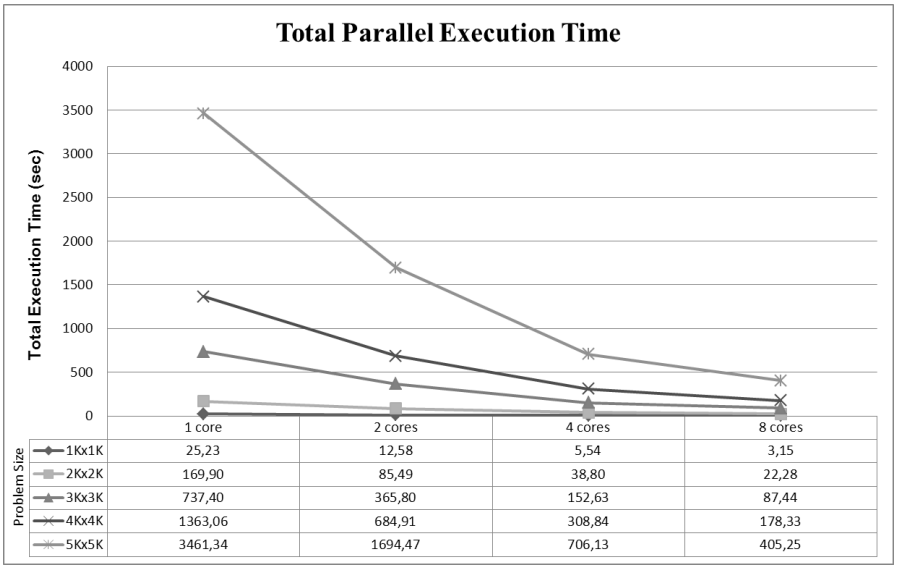
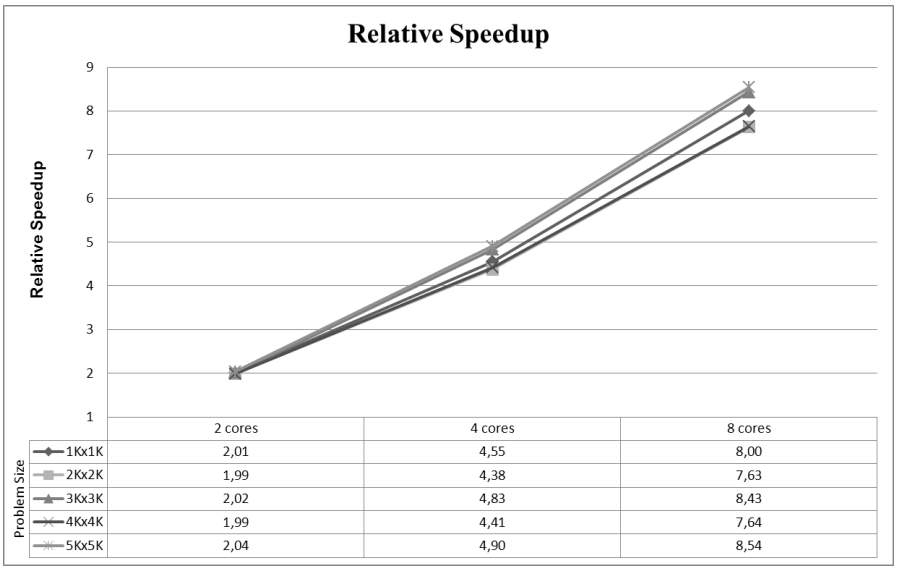



**Fig. 5.1.1:** Evaluation of total parallel execution time in seconds

## 5.2 *Relative* and *Traditional* Speedup

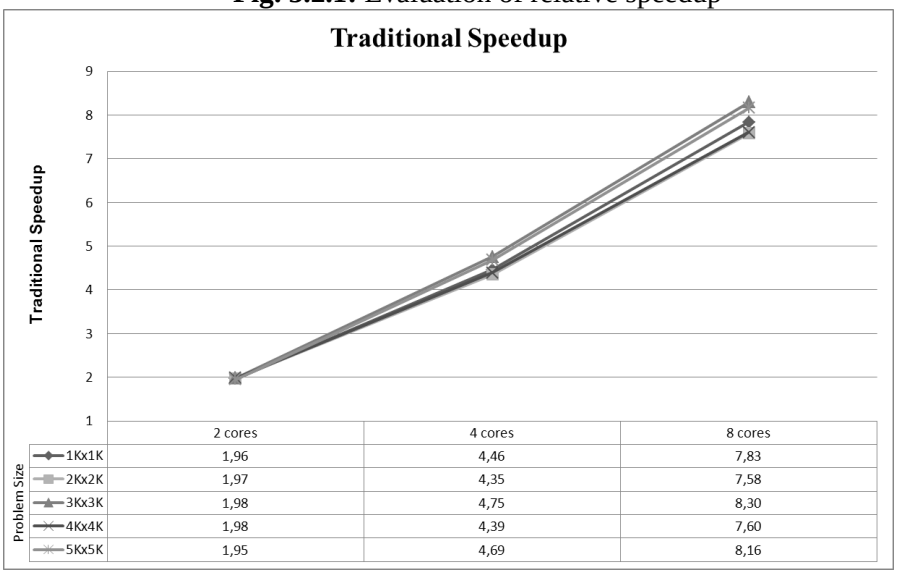
Herein, the relative and traditional speedup that the algorithm achieves will be discussed. The *relative speedup* is calculated as the ratio of the serial execution time to the parallel execution time, depending on the number of cores that are used in order to solve the problem. In this case, the serial execution time is the elapsed time for the execution of the parallel algorithm/implementation on the platform, using one core, which obviously is not the same algorithm as an actually optimized sequential algorithm. It shows how well the problem is coping with an increasing number of cores and thus provides an indication regarding the scalability of the parallel implementation. Without considering any possible parallel overhead, the limit to the speedup that can be achieved is set by the fraction of the program that can be run in parallel [19].

On the other hand, the *traditional speedup* is calculated as the ratio of the execution time of the serial algorithm to the execution time of the parallel algorithm. Traditional speedup highlights all algorithmic properties that had to be sacrificed to achieve the parallel version [19]. In addition, none of the parallel implementation penalties is hidden in this comparison; thus, the speedup is not exaggerated. The execution time of the serial algorithm is by default smaller than the serial execution time considered in case of the relative speedup, since there is no overhead due to the parallel constructs. Hence, the traditional speedup is always smaller than the relative one. In Fig. 5.2.1 is shown the relative speedup of the parallel algorithm, while in Fig. 5.2.2 is shown the traditional speedup of the parallel algorithm.




| Problem Size | 2 cores | 4 cores | 8 cores |
|---|---|---|---|
| 1Kx1K | 2,01 | 4,55 | 8,00 |
| 2Kx2K | 1,99 | 4,38 | 7,63 |
| 3Kx3K | 2,02 | 4,83 | 8,43 |
| 4Kx4K | 1,99 | 4,41 | 7,64 |
| 5Kx5K | 2,04 | 4,90 | 8,54 |

**Fig. 5.2.1:** Evaluation of relative speedup




| Problem Size | 2 cores | 4 cores | 8 cores |
|---|---|---|---|
| 1Kx1K | 1,96 | 4,46 | 7,83 |
| 2Kx2K | 1,97 | 4,35 | 7,58 |
| 3Kx3K | 1,98 | 4,75 | 8,30 |
| 4Kx4K | 1,98 | 4,39 | 7,60 |
| 5Kx5K | 1,95 | 4,69 | 8,16 |

**Fig. 5.2.2:** Evaluation of traditional speedup

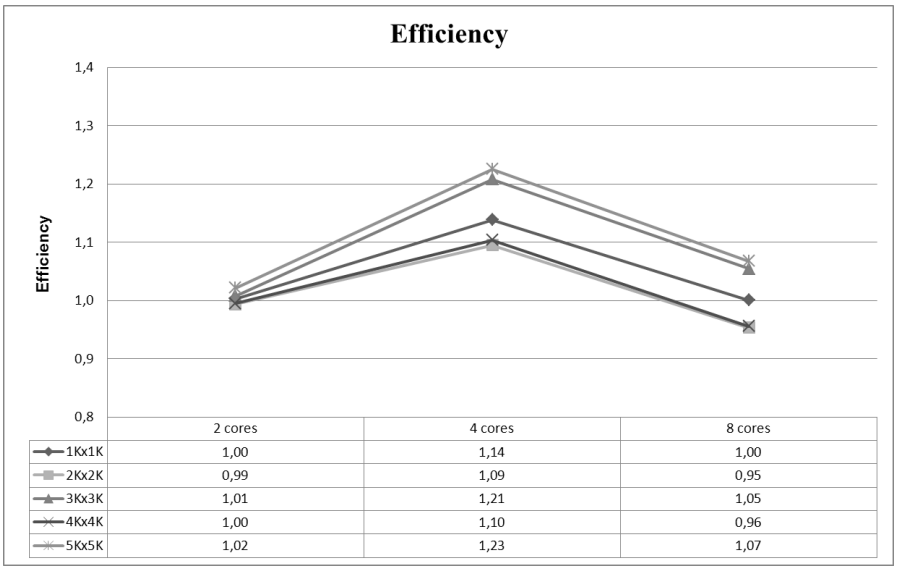
From the above Figures, it can be observed that the achieved speedup for the parallel matrix multiplication on the shared memory platform, in hand, proves to be *superlinear* for most of the cases described. Superlinear speedup can result whenever problem size per processor is reduced, whether from fixed size or fixed time performance evaluation. The decrease must be enough so that the usual sources of parallel inefficiency (load imbalance, serial algorithm steps and interprocessor communication) are compensated [17]. The superlinear speedup does not really result from parallel execution. It appears because each CPU core is now working on a smaller set of memory. The problem data handled by any one CPU core fits better in cache, so each CPU core executes faster than the single CPU could do; total computation time decreases due to more page/cache hits. A superlinear speedup is welcome, but it indicates that the sequential program was being held back by memory/cache effects.

Superlinear performance practically is impossible. However, parallel programmers do not always think of strategies that would be of common sense in case of a "real world"

problem. Hence, parallelism will sometimes lead to superlinear results. But if the most efficient method is applied to the problem in the first place, the illusion of superlinear performance would never probably appear [20].

### 5.3 Efficiency

Efficiency is traditionally defined as the speedup achieved divided by the number of utilized processors. During execution phase, each CPU core is either performing computation, communication (through main memory) or is being idle. The efficiency is a useful measure regarding the percentage of a CPU's core time spent for useful computation. An efficiency of 100% means that every CPU core spends 100% of its time performing useful computation. In Fig. 5.3.1 is shown the efficiency of the parallel algorithm achieved on the shared platform in hand, for the relative speedup case. Although typical values for efficiency lie always in the range 0 to 1, here some values exceed this range because of the resulted superlinear speedup, described in section 5.2.




| | 2 cores | 4 cores | 8 cores |
|---|---|---|---|
| 1Kx1K | 1,00 | 1,14 | 1,00 |
| 2Kx2K | 0,99 | 1,09 | 0,95 |
| 3Kx3K | 1,01 | 1,21 | 1,05 |
| 4Kx4K | 1,00 | 1,10 | 0,96 |
| 5Kx5K | 1,02 | 1,23 | 1,07 |

**Fig. 5.3.1:** Efficiency

### 5.4 Power Consumption

In Fig. 5.4.1 is shown the power consumption by the shared memory platform according to the number of cores used. *Standby(p)* bar, represents the measured power consumption when all cores are in idle state, while the other bars show the power consumption for one to eight cores being in full active state, respectively. The power consumption of the platform in idle state is remarkable, due to the fact that all cores are active and thus consuming power irrelevant to their actual use. It must be noted that when the platform is in idle state, it consumes approximately the 77,35% of the total power consumed when all cores are in full utilization.

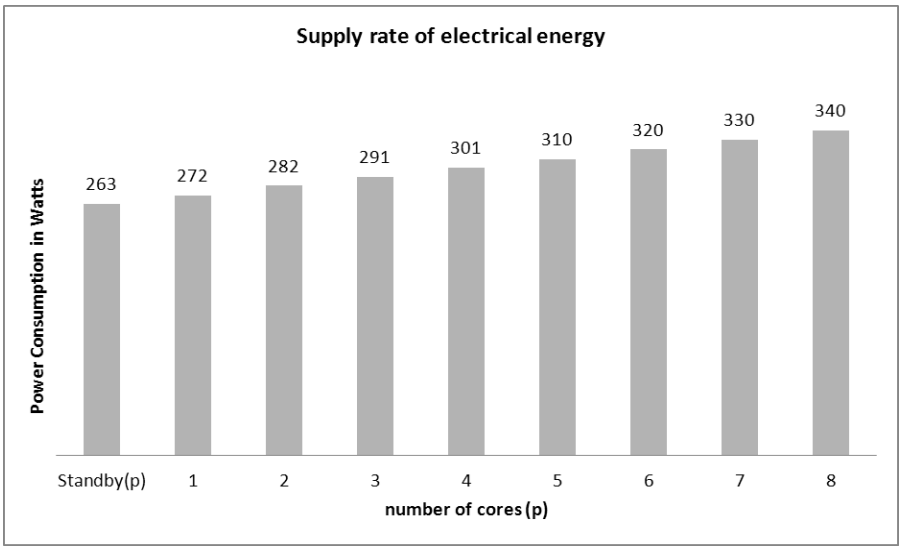



**Fig. 5.4.1:** Measured supply rate of electrical energy in Watts

From the above analysis, it is obvious that the value of parameter, k, which represents the fraction of power a core consumes in idle state (rf. 3.1, 3.2, 3.3), will vary, depending on the number of cores utilized. Its value can be evaluated by the ratio of $W_i$ to $W_f$. The values of k for the shared memory platform, depending on the number p of active cores, are given in table 5.4.1.

**Table 5.4.1:** Evaluation of parameter k

| **p** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| **k** | 0,9669 | 0,9326 | 0,9038 | 0,8738 | 0,8484 | 0,8219 | 0,797 | 0,7735 |

## 5.5 Estimation and Evaluation of Perf/W and Perf/J Metrics

In this section, the metrics of Perf/W and Perf/J are estimated and evaluated. The percentage of parallelism, f, for the given algorithm should be evaluated, prior to the estimation of the Perf/W and Perf/J values through formulas (3.2, 3.3). This applies to all selected problem sizes; thus, there are different values of parallelization percentage evaluated, as the parallel segment of the code is scaling up along with the problem size (rf. [5]). The evaluation of the parallelization percentage for each case is the ratio of the execution time of the segment that can be parallelized to the total execution time of the serial program. The values of, f, for the specific algorithm, are shown in Fig. 5.5.1.

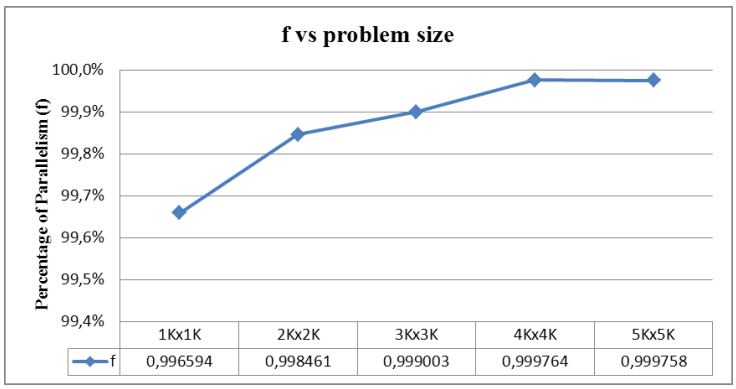
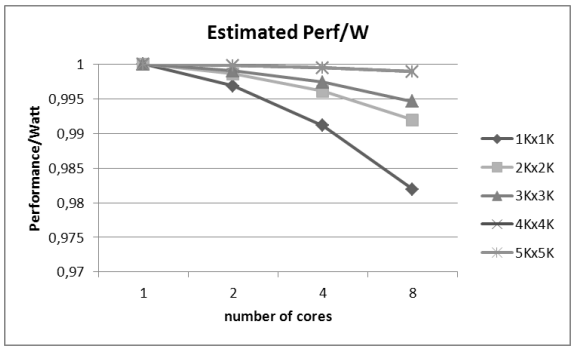
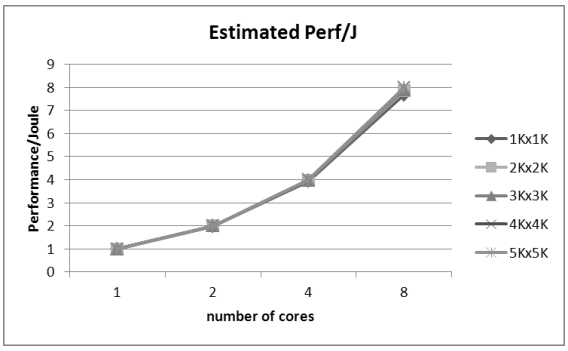



**Fig. 5.5.1:** Percentage of parallelism vs. problem size

Finally, the estimated values of Perf/W and Perf/J can be evaluated. In Fig. 5.5.2(a) are shown the estimated values of metric Perf/W, while in Fig. 5.5.2(b) are shown the estimated values of metric Perf/J.



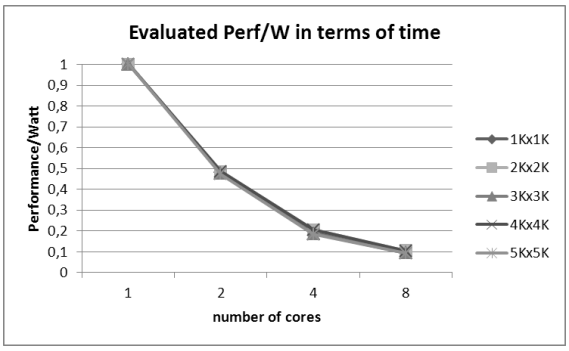

(a) Estimated Perf/W



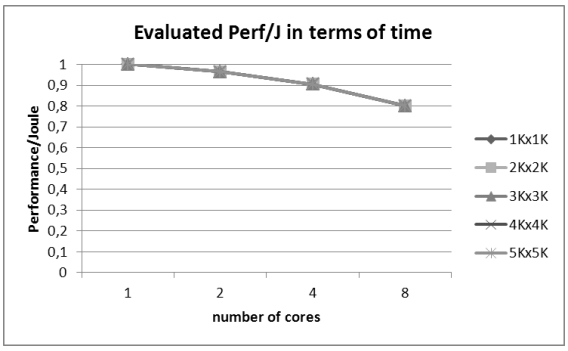

(b) Estimated Perf/J

**Fig. 5.5.2:** Estimated values of metrics Perf/W, Perf/J

The evaluated metrics of Perf/W and Perf/J are presented with the performance following not only the time basis concept, but, also, the speedup basis concept. Measured/Evaluated values of Perf/W and Perf/J are normalized, rescaling the results to the unit interval; thus, all the results are given in a scale between 0 and 1, in order to perform a straightforward comparison with the estimated values when using (3.2, 3.3), since the values of Perf/W, Perf/J of a single-core execution have been assumed as equal to *one* [6]. The results of the algorithm for the shared memory platform are presented in the Figures below.

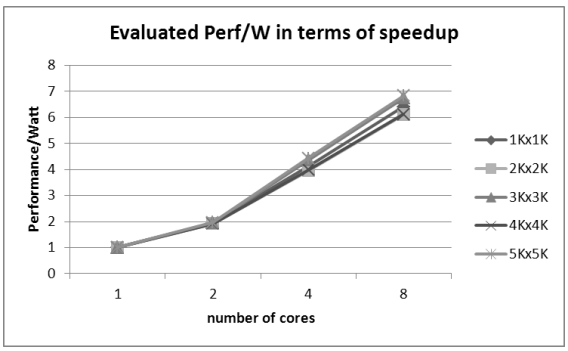



(a) Evaluated Perf/W

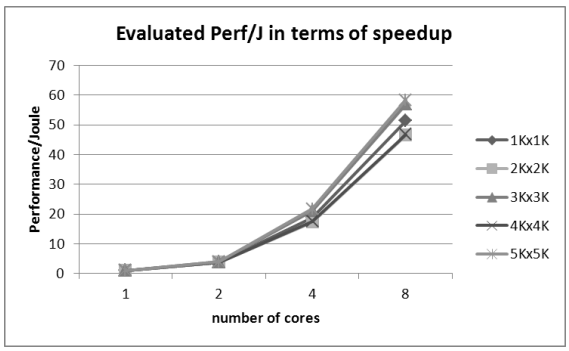



(b) Evaluated Perf/J

**Fig. 5.5.3:** Evaluated values of metrics Perf/W, Perf/J in terms of time




(c) Evaluated Perf/W




(d) Evaluated Perf/J

**Fig. 5.5.4** Evaluated values of metrics Perf/W, Perf/J in terms of speedup

The diagrams given in Figs. 5.5.3 and 5.5.4 show the relation between the metrics of Perf/W and Perf/J taking into consideration, both, the number of cores and the problem sizes. All the above diagrams were calculated from the values presented in Table 5.5.1.

From the values of the metrics that are given in Table 5.5.1 and graphically shown in Figs. 5.5.3 and 5.5.4, significant differences are observed between estimated and evaluated values. These differences are due to the fact that the estimated values are obtained using (3.2, 3.3), where, f, represents percentage parallelization and not time. This leads in a faulty estimation of the average power consumption using (3.1), since for the calculation of energy only pure time values are required. It should be noted that, the evaluated values are obtained from the ratio of the real parallel execution time or the real speedup achieved, respectively, to the real average power/energy consumption, as it was explained earlier.

**Table 5.5.1:** Estimated and Evaluated values of metrics Perf/W and Perf/J

| **Evaluated Perf/W in terms of time** | | | | | | **Evaluated Perf/J in terms of time** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** | | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** |
| **1** | 1 | 1 | 1 | 1 | 1 | **1** | 1 | 1 | 1 | 1 | 1 |
| **2** | 0,4808 | 0,4854 | 0,4785 | 0,4847 | 0,4722 | **2** | 0,9646 | 0,9646 | 0,9646 | 0,9646 | 0,9646 |
| **4** | 0,1986 | 0,2064 | 0,1871 | 0,2048 | 0,1844 | **4** | 0,9041 | 0,9039 | 0,9038 | 0,9038 | 0,9037 |
| **8** | 0,1001 | 0,1050 | 0,0949 | 0,1047 | 0,0937 | **8** | 0,8014 | 0,8007 | 0,8004 | 0,8004 | 0,8003 |
| **Evaluated Perf/W in terms of speedup** | | | | | | **Evaluated Perf/J in terms of speedup** | | | | | |
| | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** | | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** |
| **1** | 1 | 1 | 1 | 1 | 1 | **1** | 1 | 1 | 1 | 1 | 1 |
| **2** | 1,9353 | 1,9169 | 1,9444 | 1,9196 | 1,9703 | **2** | 3,8827 | 3,8093 | 3,9197 | 3,8203 | 4,0248 |
| **4** | 4,1159 | 3,9579 | 4,3665 | 3,9888 | 4,4300 | **4** | 18,7379 | 17,3308 | 21,0961 | 17,6048 | 21,7153 |
| **8** | 6,4138 | 6,1062 | 6,7498 | 6,1177 | 6,8356 | **8** | 51,3307 | 46,5670 | 56,9186 | 46,7601 | 58,3844 |
| | | | | | | | | | | | |
| **Estimated Perf/W** | | | | | | **Estimated Perf/J** | | | | | |
| | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** | | **1Kx1K** | **2Kx2K** | **3Kx3K** | **4Kx4K** | **5Kx5K** |
| **1** | 1 | 1 | 1 | 1 | 1 | **1** | 1 | 1 | 1 | 1 | 1 |
| **2** | 0,997 | 0,999 | 0,999 | 1,000 | 1,000 | **2** | 1,99 | 1,99 | 2,00 | 2,00 | 2,00 |
| **4** | 0,991 | 0,996 | 0,997 | 0,999 | 0,999 | **4** | 3,925 | 3,966 | 3,978 | 3,996 | 3,996 |
| **8** | 0,982 | 0,992 | 0,995 | 0,999 | 0,999 | **8** | 7,673 | 7,853 | 7,902 | 7,980 | 7,980 |

## 7. CONCLUSIONS

Herein, the performance evaluation of a shared memory platform, in conjunction with the consumed energy, was investigated. An analytical model for calculating the total energy consumption was also introduced. In accordance with the parallelized percentage each time, the performance scalability to the power/energy consumption can be calculated.

A shared memory platform was evaluated for the power its processors/cores demand at the idle and fully utilized state; the estimations of the theoretical model were compared to the experimental results obtained on the basis of the performance/power and performance/energy ratio metrics. The superlinear speedup obtained, and the differences

between the estimated and the evaluated results were highlighted and justified. The fact that the increasing demands for faster performance in the computer domain is never satisfied and the need for keeping the power/energy consumption in manageable levels, urges the research community to invent new and more precise analytical models in order to predict performance and energy/power consumption in the many-core era. Moreover, new metrics that will describe and compare different platforms in a direct way also have to be invented.

The use of more accurate metrics and simple analytical models at the early design stages, it would assist to avoid misguidance and costly decisions. While performance per watt is useful, absolute power requirements are also important. Claims of improved performance per watt may be used to mask increasing power demands. For instance, though newer generation GPU architectures may provide better performance per watt, continued performance increases can negate the gains in efficiency and the GPUs continue to consume large amounts of power. And, last, energy required for climate control of the computer's surroundings is often not counted in the wattage calculation, but it is, also, quite significant.

## REFERENCES


1. Amdahl, G. (1967). Validity of the Single-Processor Approach to Achieving Large Scale Computing capabilities: *AFIPS Conference Proceedings*.
2. Mudge, T. (2001). Power: A First-Class Architectural Design Constraint, *Computer*, pp. 52-58.
3. Benner, R.E., Gustafson, J.L., Montry, G.R. (1988). Development and analysis of scientific application programs on a 1024-processor hypercube, SAND 88-0317: Sandia National Laboratories.
4. Shi, Y. (1996). Reevaluating Amdahl's Law and Gustafson's Law, Computer and Information Sciences Department: Temple University.
5. Gustafson, J.L. (1988). Reevaluating Amdahl's Law, *CACM* , v.31(5), pp. 532-533.
6. Woo, D.H., Lee, H.H.S. (2008). Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era, *IEEE Computer*, v.41(12), pp. 24-31.
7. Gonzalez, R., Horowitz, M. (1996). Energy Dissipation in General-Purpose Microprocessors, *IEEE J. Solid-State Circuits*, v.31(9), pp. 1277-1284.
8. Lu, J., Guo, Y. (2011). Energy-Aware Fixed-Priority Multi-core Scheduling for Real-Time Systems, *RTCSA-IEEE 17th International Conference Proceedings*, v.1, pp. 277-281.
9. Yang, L., Lin, M., Yang, T. (2012). Multi-core Fixed Priority DVS Scheduling, *Algorithms and Architectures for Parallel Processing*, Lecture Notes in Computer Science, v.7439, pp. 517-530.
10. Karanikolaou, E.M., Milovanović, E.I., Milovanović, I.Ž., Bekakos, M.P. (2014). Performance scalability and energy consumption on distributed and many-core platforms, *The Journal of Supercomputing*, v.70.1, pp. 349-364.
11. The OpenMP® API specification for parallel programming, Last visit: Sept.2014, http://openmp.org
12. TYAN - server motherboards, server barebones for HPC, GPU, Cloud Computing and embedded applications, Last visit: Sept.2014, http://www.tyan.com
13. AMD Server Processors, Last visit: Sept.2014, http://www.amd.com/en-us/products/server
14. Londono, S.M., Gyvez, J.P. (2010). Extending Amdahl's Law for Energy Efficiency, *ICEAC Conference Proceedings*, pp. 1-4.
15. Marowka, A. (2012).Extending Amdahl's Law for Heterogeneous Computing. I*EEE ISPA Conference* Proceedings, pp. 309-316.
16. Cassidy, A.S., Andreou, A.G. (2012). Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy, *IEEE Transactions on Computers*, v.61(8), pp. 1110-1126.
17. Gustafson, J.L. (1990). Fixed time, tiered memory, and superlinear speedup, *Proc. Fifth Conf. on Distributed Memory Computers*, pp. 1255-1260.
18. Sutter, H. (2008). Effective Concurrency-Herb considers how to set superlinear speedups by harnessing more resources, *Dr Dobb's Journal-Software Tools for the Professional Programmer*, pp. 52-55.

19. Speedup and Amdahl's law, http://people.nas.nasa.gov/~schang/origin_parallel.html#amdahl
20. Farnham, K. (2008). Superlinearity Is Impossible; We Just Don't Always Think Correctly, Last visit: Sept.2014, https://software.intel.com/
21. Akenine-Möller, T. & Johnsson, B. (2012). Performance per What? *Journal of Computer Graphics Techniques (JCGT)*, v.1(1), pp. 37-41.