IOP Conf. Series: Materials Science and Engineering **158** (2016) 012055 doi:10.1088/1757-899X/158/1/012055

Scalability of parallel finite element algorithms on multi-core platforms

S P Kopysov, A K Novikov, N S Nedozhogin and V N Rychkov

Institute of Mechanics, Ural Branch of the Russian Academy of Sciences, 34 T. Baramzinoy, Izhevsk, Russia 426067

E-mail: s.kopysov@gmail.com

Abstract. The speedup of element-by-element FEM algorithms depends not only on peak processor performance but also on access time to shared mesh data. Eliminating memory boundness would significantly speed up unstructured mesh computations on hybrid multi-core architectures, where the gap between processor and memory performance continues to grow. The speedup can be achieved by ordering unknowns so that only those elements are processed in parallel which do not have common nodes. Therefore, memory conflicts are minimized. FEM assembly is performed with respect to the ordering, which defines how to compose vectors. Mesh can be partitioned into disjoint subdomains by using different layer-by-layer schemes. In this work, we evaluated several partitioning schemes (block, odd, even, and their modifications) on multi-core platforms, using Gunther's Universal Law of Computational Scalability. We performed numerical experiments with element-by-element matrix-vector multiplication on unstructured meshes on multi-core processors accelerated by MIC and GPU. With ordering, we achieved 5-times speedup on CPU, 40-times speedup on MIC, and 200-times speedup on GPU.

1. Introduction

Performance and scalability modeling of computational algorithms on modern multi-core/many-core platforms is a crucial problem. On multi-core platforms, it is important to take into account different types of delays, which can be caused by: exchange of shared rewritable data between the processor caches and between the processors and main memory (i), synchronization locks (serialization) shared data available for recording (ii), waiting for memory access to complete operations (iii), etc. Evolution of parallel speedup models derived from Amdahl's law is described in [1,2]. In particular, they refer to Hill-Marty model [3], which includes additional parameters defining a number of computing hardware resources. However, estimation of these parameters for real platforms is not trivial.

For over a decade, the Universal Scalability Law has been successfully applied to model diverse software applications on modern hardware platforms [4,5] but it has been largely ignored by the parallel simulation community. This model accounts for waiting for access to shared resources, retrograde scaling (latency due to exchange of data between caches) and resource limitations. In this work, we apply this model to FEM simulations. To the best of our knowledge, this is the first application of this model to parallel numerical algorithms.

Predictive models based on the results of real computational experiments allow for detecting and eliminating the sources of delays. Many FEM algorithms demonstrate irregular memory accesses, which significantly reduces the efficiency of parallel processing. The irregularity is caused by

unstructured meshes. As a result, the global assembly phase becomes the main performance bottleneck in many FEM algorithms [6].

This paper is structured as follows. In Section 2, we show how element-by-element FEM can be parallelized on shared memory architecture. In Section 3, we present new partitioning schemes, which significantly improve data locality and performance of element-by-element FEM algorithms. In Section 4, we describe experimental platforms and their memory access features. In Section 5, we develop the Gunther's Universal Law of computational scalability in relation to proposed algorithms and multi-core architectures. Section 6 concludes the paper.

2. Element-by-element FEM on shared memory platforms

In element-by-element schemes [7], assembling is a part of solving the finite element system of equations. Assembly operator is applied not to matrices but to vectors resulting from the matrix-vector multiplication:

$$q = Kp = \sum_{e=1}^{m} C_e^T \widetilde{K}_e C_e p = \sum_{e=1}^{m} C_e^T \tilde{q}_e = \sum_{e=1}^{m} q_e, \quad (1)$$

where q, p, q_e are the vectors of size N; K is the global matrix $N \times N$; $\widetilde{K}_e$ is the local stiffness matrix $N_e \times N_e$ of a finite element e; C_e is the incidence matrix $N_e \times N$ that maps the local space if numbers of unknowns (degrees of freedom) $[1,2,\dots,N_e]$ into the global space $[1,2,\dots,N]$; m is the number of finite elements. This mapping can be done either by indirect indexing of mesh nodes to unknowns (i) or by multiplying by the incidence matrix, which can be efficiently implemented on GPUs (ii).

Let us consider a parallel version of (1) for shared memory. Here vectors p and q are in the shared memory of parallel processes/threads, and matrices $\widetilde{K}_e$, C_e, C_e^T and sparse vector q_e are in the local memory of processes. Then summation of the vectors qe that belong to different processes but have nonzero components with the same indices may result in conflicts and cause errors.

The product $q = Kp$ can be replaced by two operations: the element-by-element multiplication $\tilde{q}_e = \widetilde{K}_e C_e p, \ e = 1,2,\dots,m$ and assembling $q_e = \sum_{e=1}^{m} C_e^T \tilde{q}_e$. These operations have different computation and communication costs because of different data locality, especially in case of unstructured meshes. They have different memory access patterns, determined by local and global node indexing, and therefore have different potential for parallelization.

Assume that each process computes over $m_i \approx m / n$ finite elements, where i is the process number, and n is the number of processes. Then the product can be expressed in the matrix form as follows:

$$q = Kp = \sum_{i=1}^{n} C^{T(i)} \widetilde{K}^{(i)} C^{(i)} p = \sum_{i=1}^{n} C^{T(i)} \tilde{q}^{(i)} = \mathcal{A}(\tilde{q}), \quad (2)$$

where $C^{(i)}$ is the incidence matrix assembled from m_i matrices C_e; $\widetilde{K}^{(i)}$ is the block-diagonal matrix united from m_i matrices $\widetilde{K}_e$ as blocks; $\tilde{q}^{(i)}$ is the vector assembled from m_i vectors $\tilde{q}_e$; and $\mathcal{A}$ is the assembly operator for the vector q.

To make this algorithm efficient on shared memory platforms, it is necessary to minimize concurrent memory accesses. This can be achieved by splitting finite elements into sets that do not share nodes with each other. Then, these sets are assigned to different processes/threads, and the assembly operation (2) is performed on nonadjacent elements.

3. FEM algorithms with layer-by-layer partitioning

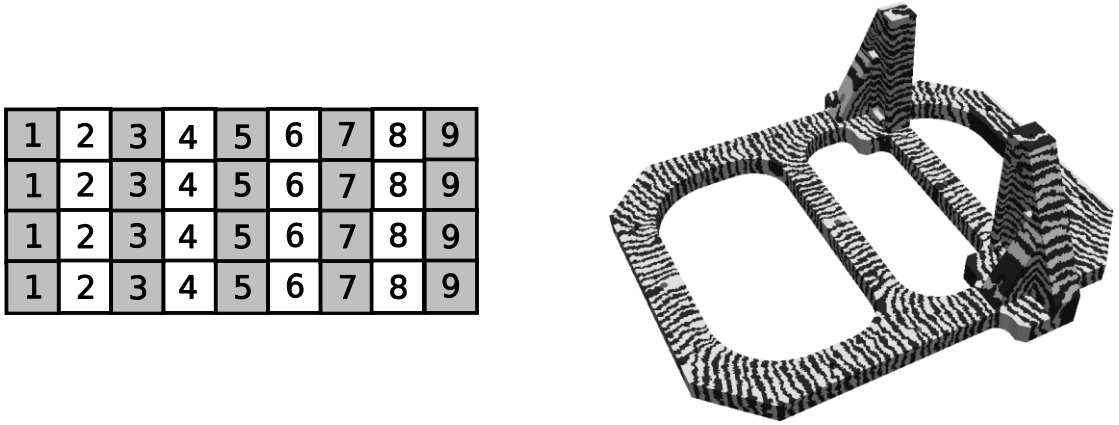
In this section, we present new partitioning schemes, which significantly improve data locality and performance of element-by-element FEM algorithms on shared memory platforms.

We assume that two subsets of mesh cells are nonadjacent at some moment of time if the cells simultaneously taken from these subsets do not contain common vertices. The moment of time is taken

when the subsets are accessed for computations. This means that the cells in a subset and the subsets themselves can be connected topologically.

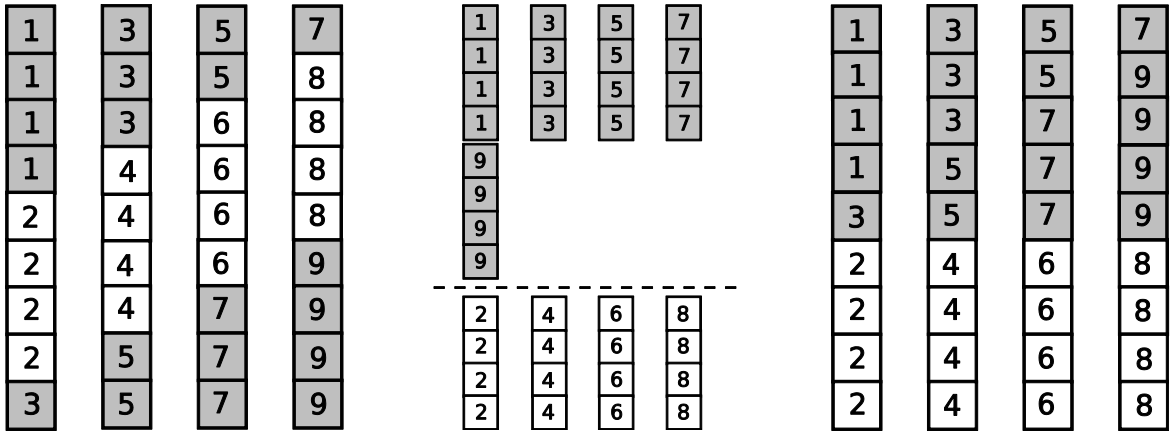
We partition the mesh Ω into nonoverlapping layers of cells $s_j, j = 1,2,\ldots,n_s$ (figure 1(a)), then combine the layers to get the subsets of cells (subdomains) $\Omega_i, i = 1,2,\ldots,n_\Omega$. We use different schemes to construct subdomains Ω_i from layers. Our target is to reorder cells so that the cells with common vertices are not accessed simultaneously from parallel processes (threads).

In the block scheme, the layers are combined consecutively, then these combined layers are divided into subdomains with approximately equal number of finite elements (figure 2(a)). In the layer index parity scheme, the layers are enumerated (figure 2(c)). The dashed line in figure 2 (b) represents the end of one parallel OpenMP section and the beginning of another.



(a) (b)

Figure 1. Layer-by-layer mesh partitioning: (a) scheme; (b) result: unstructured mesh with 137 layers, 485843 tetrahedrons.



(a) (b) (c)

Figure 2. Combining layers into blocks for four processes ($n_\Omega = 4$): (a) block scheme (bl); (b) layer index parity scheme (ev); (c) balanced layer index parity scheme (ev+bal).

We compared these schemes by measuring the performance and load imbalance of the element-by-element matrix-vector product without assembling the vectors (this operation is fully parallel). We performed experiments with the unstructured mesh shown in figure 1 on eight-core processor Xeon E5-2690. Figure 3 shows the speedup obtained with four types of the mesh partitioning: block (bl),

odd-parity (ev), balanced odd-parity (ev+bal), and multilevel graph partitioning by METIS). The load imbalance in percent to equal load is shown in percentage above the bars. The load imbalance of (bl) and (ev+bal) is neglected as it was essentially less than 1% in all experiments. Achieving the linear speedup is limited by load imbalance. The load imbalance caused by the proposed mesh partitioning schemes proved to be less than the one caused by the multilevel partitioning of the mesh dual graph [8].

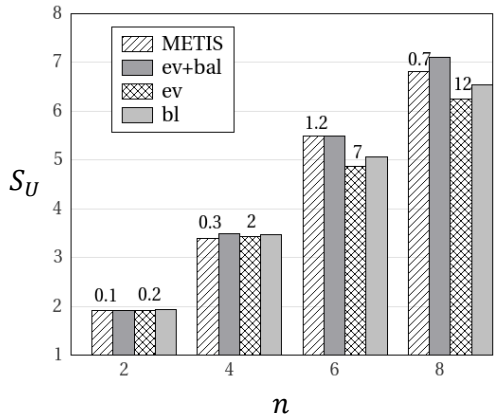


Figure 3. The speedup of the matrix-vector product on Xeon E5-2690 for different mesh partitioning schemes and numbers of processes ($n = 2, 4, 6, 8$).

The rest of the paper will be detected the sources of delays in some parallel FEM operations using layer-by-layer partitioning on multi-core/many-core platforms.

4. Experimental platforms and their memory access features

In this section, we describe experimental platforms and their memory access features.

Parallel execution of FEM algorithms on hybrid multi-core platforms significantly increases the frequency of memory access so that memory bandwidth becomes a performance bottleneck. Multiple processes/threads compete for memory, which results in memory contention. This situation is further complicated by multi-level caches, which allow for reusing frequently accessed data but require maintaining data consistency (cache coherence). Memory can be utilized more efficiently if multiple writes to the same memory location issued by different processes are executed sequentially (i), and there is a gap in time between write and read operations issued from different processes to the same memory location (ii).

We performed experiments with parallel element-by-element FEM algorithms on several platforms, which represent most popular multi-core architectures: CPU — Xeon E5-2609, Opteron 8435, Xeon E5-2690; MIC — Xeon Phi 7110X; GPU — GTX 980. Platforms based on these architectures have significant differences in memory access.

Important for the scalability analyzing of parallel algorithms are methods to solve problems of memory access organization in modern architecture. Some of them settled in the hardware or the compiler and should be somehow taken into account when describing the architecture of the computer system on which the computation will be performed and the behavior that we want to predict.

5. Universal scalability estimation

Model of computing is the next higher level of abstraction and is more formal model of the appropriate architecture model. It allows you to build a cost function reflecting the time required to perform the algorithm on the resources of a computing system, a given architecture model. This computing model should provide a simple method for the design and evaluation of parallel algorithms.

In the work of Gunther [5] (see Bibliography therein) was demonstrated the possibility of the representation and evaluation of scalability / speedup on the basis of generalization of Amdahl's law,

taking into account overheads relating to the agreement of calculations between identical cores / processors and the maintenance of a coherent state of the system.

Gunther's model (USL) is considered to be universal in the sense that it is not expected to have any properties of the algorithm, software, or a specific multi-core architecture of the system, and others.

The model assumes that speedup is function that has the global maximum value and depends on the number of cores/processors, the degree of contention, and the lack of coherency.

The use of this model is to measure the acceleration of parallel computing $S(n) = t_1 / t_n$, where t_1, t_n — algorithm execution time of one or n — threads and to obtain the most accurate form for an speedup function of the specific algorithm and computing system with estimation of two model parameters.

The speedup of an arbitrary computing system (algorithmic, program and architectural) is performed a rational function.

Speedup of S_U with an increase in the number of parallel processes n in the USL model is written as

$$S_U(n) = \frac{n}{1 + \sigma(n-1) + \lambda n(n-1)}, \quad (3)$$

where parameters σ, λ defined within $0 \le \sigma$, $0 < \lambda < 1$. At $\lambda = 0$ ratio (3) takes the form of Amdahl's law with the part of sequential computing, defined as $\sigma = (1 - f)$, where f — parallel fraction. The denominator in (3) consists of three terms, each has a definite physical interpretation: parallelism — linear scalability is possible if the individual components of the system (CPU, threads, etc.) can work without any interaction $\sigma, \lambda = 0$; scalability is limited contention — conflicts between different parts of a computer system (algorithm), caused by the effects of serialization and queuing $\sigma > 0, \lambda = 0$; scalability is limited coherence — delays caused by the maintenance of the system in a consistent state, is associated with a cache memory in a different parts of software and hardware of the computer system $\sigma, \lambda > 0$.

Thus, the parameters of the model can be represented in charge of two specific mechanisms that limit the linear acceleration of the parallel algorithm with increasing of the number of parallel processes, or cores. Note, that the second term increases linearly and the third — square with increasing of the number n of interacting processes. Then, computational algorithms, architectures can be classified and described in accordance with the settings σ, λ. The absolute values of the model parameters do not specify the detailed mechanisms and data for computing organizing and working with memory, but fairly well characterized model of parallelism and architecture of the system when considering the obtained speedup for one and the same algorithm as data partitioning on different types of multi-core architectures. However, it is possible to assess and predict the incidence of excessive costs on the agreement and synchronization of access to shared data between threads.

Based on the data obtained of parallel speedup $S = t_1 / t_n$ for the above algorithms of layer-by-layer partitioning determined scalability (3), (4) of parallel operation of vectors assembly $q = \mathcal{A}(\tilde{q})$ on the unstructured mesh for several architectures and technologies of parallel programming with increasing of the parallel processes number n (for technologies OpenMP and CUDA see table 1). Numerical experiments carried out over the assembly vectors results element-by-element matrix-vector product for adaptive unstructured mesh, shown in figure 1. The partition by parity numbers of layers with alignment load was chosen for comparison parameters of Gunther's model on different processor architectures. This optimized data movement and memory access. Thus, the calculations were performed on the same data, to separate them by threads number used the same single algorithm and software OpenMP with parallelization of cycle for finite elements by thread number. To improve the effectiveness of the assembly operations on the GPU algorithm was modified by the introduction of new levels of parallelization.

Table 1. USL parameters of the assembly operation $\boldsymbol{q} = \boldsymbol{\mathcal{A}}(\tilde{\boldsymbol{q}})$ on the unstructured mesh (figure 1) for experimental platforms.

USL parameter	Xeon E5-2609	Opteron 8435	Xeon E5-2690	Xeon Phi 7110X	GeForce GTX 980
n	4	6	8	61	2048
σ	0.1	0.021	0.08	0.006	0.0018
λ	0.0001	0.02	0.0001	0.00006	0.00000044
R^2	0.99	0.982	0.9965	0.994	0.999
$S_{U_{max}} \,/\, n_{U_{max}}$	8.41 / 94	3.56 / 7	10.09 / 96	46.76 / 129	318.96 / 1511
$S_{max} \,/\, n_{max}$	3.1 / 4	3.48 / 5	5.19 / 8	39.43 / 60	219 / 384

The non-linear regression analysis was used to find the model parameters in (3). The results are shown in table 1. The degree of correspondence between experimental S and data computed by USL — S_U is estimated with R^2 measure. The obtained values of R^2 explain the change of speedup with dependence from n for some σ and λ.

The obtained model parameters σ, λ allowed us to estimate the number of processor cores, at which the maximum speedup of $n_{U_{max}} = \sqrt{(1-\sigma)\lambda}$ is achieved for the algorithm executed on the CPU and computing accelerators. Among the obtained results, we note a slowdown of computations at parallel vectors assembly in the shared memory model for graph partitioning with a critical section. The critical section is not required in the proposed layer-by-layer partitioning algorithms, all restrictions on access to the shared memory are allowed, and as a result, good scalability are achieved on universal computing processors and accelerators. The figure 4 shows the experimental measuring results (marks) of the speedup for assembly operations are for different architectures and universal processors and curves constructed by the model of USL on (3).

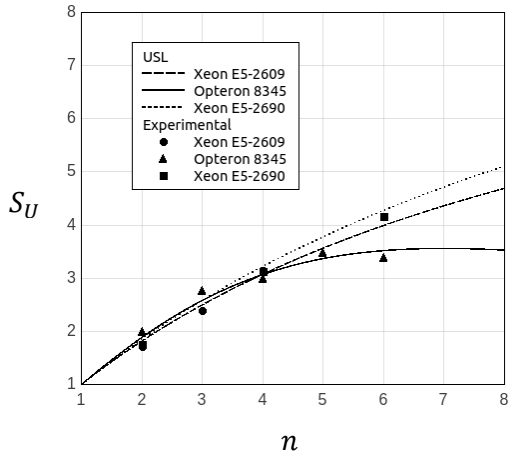


Figure 4. Scalability of assembly operation on multi-core CPUs.

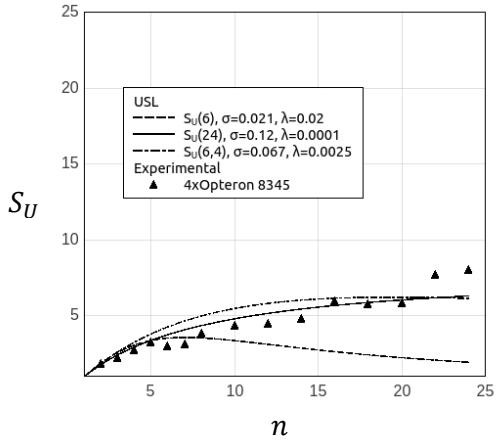


Figure 5. Scalability of assembly operation on multi-socket node.

A significant three-five-fold speedup of parallel assembly is achieved even for unstructured meshes, which is about half the maximum possible speedup while increasing the number of cores on the order for the architecture of Xeon E5 processors. In the six-core Opteron processors, with a three-level cache organization costs for agreement of data location lead to the fact that the maximum speedup occurs only five cores involved.

Scalability on computing accelerators

Element-by-element assembly vectors can efficiently be carried out on the same computing accelerators (see figure 6), which is confirmed by the experimental results and the estimation of the USL model parameters for the advanced accelerators MIC with enlarged cores number. For comparison, experiments were performed on quasi-structured meshes. The assembly operation on MIC is performed with almost the same scalability, which corresponds to the obtained values of the model parameters σ, λ, characterizing cache coherency among all used cores (speedup of about 40 times on the 61 cores). In this assembly algorithm, memory access is hidden behind the calculation rate (minimum parameters σ, λ among all architectures) at computing on the graphics card GTX 980 and the maximum acceleration could be achieved with such an architecture with 1511 cores (see figure 7). Experimentally achieved speedup is limited only by the ability at a given mesh to obtain the desired number of layers and the presence of an appropriate amount of RAM.

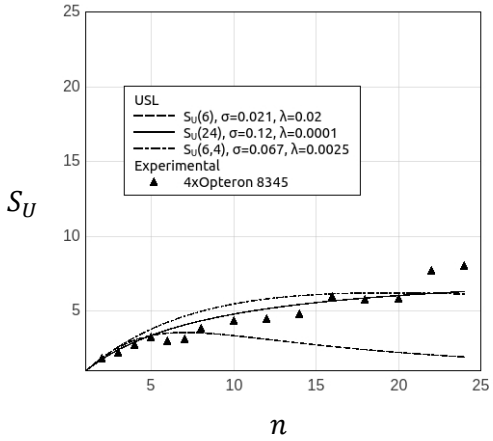


Figure 6. Scalability of assembly operation on MIC.

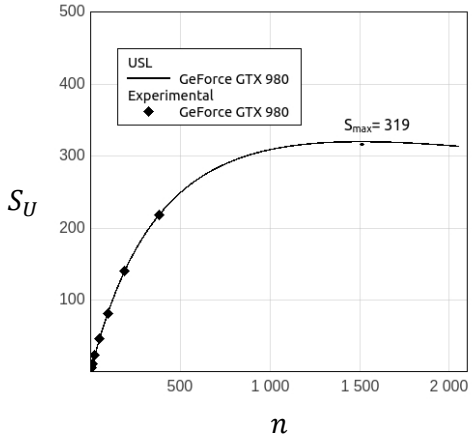


Figure 7. Scalability of assembly operation on GPU.

Scalability on multi-socket node

Since the universal model has no internal structure, it can be used for more general simulation, but only homogeneous computing system, such as multiprocessor (multi-socket) computing nodes containing the same processors and clusters consisting of computing nodes with processors of the same architecture and performance.

Define the parameters of a multiprocessor computing node as: p — the number of multicore processors in a single compute node; $n_p n \,/\, p$ — number of cores per one processor; σ_p — parameter characterizing contention between processors; λ_p — data consistency across multiple processors of a computing node.

Then we can write a generalization of ratio (3) to define a universal scalability of computational node containing p processors/sockets

$$S_U(n,p) = \frac{pS_U(n)}{1 + \sigma_p(p-1)S_U(n) + \lambda_p(p-1)pS_U^2(n)}, \tag{4}$$

where $S_U(n,p)$ — speedup, and determined taking into account the overhead multi-sockets node based on the speedup of a multi-core processor $S_U(n)$. At $p = 1$, $\sigma_p = 0$ last expression is converted to (3).

To use this model to consider when evaluating the speedup of multi-sockets node $S_U(n,p)$, that all of its processors should also be uniform. In the future, the ratio of (3) can also be generalized to evaluate the speedup computing cluster $S_U(n,p,n_c)$, containing n_c — computing nodes connected by a communication network and determining the appropriate parameters σ_c, λ_c.

For the four sockets node with Opteron 8345 processors and non-uniform memory access were measured computational costs and three speedup estimations were realized (see figure 5). The first curve $S_U(6)$ constructed according to a six-core processor, and shows a significant difference from the experimental results with the model of increasing the number of cores.

The curve denoted by $S_U(24)$ is obtained in the approximation of the results for the model (3) when performing assembly operations on all available twenty four cores excluding the non-uniform memory access neighboring processors. In this case, evaluation of the acceleration reaches seven in the presence of a large number of cores. When model accounting for one processor (3) and NUMA architecture four sockets node in the model (4) obtained more reliable results predicted maximum acceleration of about six.

6. Conclusion

The results of the speedup simulation characterize fairly well used model of parallel computing for consideration obtained speedup of the same algorithm and data sharing on different types of multi-core architectures. On the basis of this model it is possible to consider the applicability of the algorithms change, architecture or software in order to minimize the values of the parameters σ, λ and reduce costs.

In this work, we used USL model for scalability analysis of element-by-element FEM operations on modern multi-core platforms. We extended this model for multi-socket, MIC, GPU architectures, and demonstrated that it adequately approximates the scalability.

Acknowledgments

This work was partially supported by RFBR according to the research projects №14–01–00055-a, №16–01–00129-a.

References

[1] Yavits L, Morad A and Ginosar R 2014 *Parallel Computing* D **40** 1
[2] Al-Babtain B M, Al-Kanderi F J, Al-Fahad M F and Ahmad I 2013 *Int. J. of New Comp. Arch. and their Appl*. **3** 30
[3] Hill M D and Marty M R 2008 *Computer* **41** 33
[4] Gunther N J 2002 *CoRR*. **cs.DC/0210017**
[5] Gunther N J, Puglia P and Tomasette K 2015 *Comm. of the ACM* **58** 46
[6] Markall G R, Slemmer A, Ham D A, Kelly P H J, Cantwell C D and J S S 2013 *Int. J. Numer. Meth. Fluids* **71** 80
[7] Kopysov S P, Kuzmin I M, Nedozhogin N S, Novikov A K, Rychkov V N, Sagdeeva Y A and Tonkov L E 2014 *Computer Research and Modeling* **6** 79
[8] Karypis G and Kumar V 1998 *SIAM Journal on Scientific Computing* **20** 359