DOI: 10.14529/jsfi150101

AlgoWiki: an Open Encyclopedia of Parallel Algorithmic Features

Vladimir V. Voevodin[1][2], *Alexander S. Antonov*[1][3], *Jack Dongarra*[4][5]

The main goal of this project is to formalize the mapping of algorithms onto the architecture of parallel computing systems. The basic idea is that features of algorithms are independent of any computing system. A detailed description of a given algorithm with a special emphasis on its parallel properties is made once, and after that it can be used repeatedly for various implementations of the algorithm on different computing platforms. Machine-dependent, part of this work is devoted to describing features of algorithms implementation for different parallel architectures. The proposed description of algorithms includes many non-trivial features such as: parallel algorithm complexity, resource of parallelism and its properties, features of the informational graph, computational cost of algorithms, data locality analysis as well as analysis of scalability potential, and many others. Descriptions of algorithms form the basis of AlgoWiki, which allows for collaboration with the computing community in order to produce different implementations and achieve improvement. Project website: `http://algowiki-project.org/en/`.

Keywords: algorithm structure, resource of parallelism, parallel computing, efficiency, performance, supercomputers, scalability, data locality, encyclopedia of algorithmic features.

Introduction

Computers evolve quickly, and there have been at least six generations of computing architecture over the last forty years that caused the need for radical changes in software. Vector computers, vector-parallel, massive parallel computers, shared-memory nodes, clusters of shared-memory computers, computers with accelerators... The computing community has survived this, but in each evolution, the software basically had to be rewritten from scratch as each new generation of machines required singling out new properties in the algorithms, and that was reflected in the software.

Alas, there is no reason to hope that the situation will change for the better in the future. Vendors are already considering various prospective architectures, featuring light and/or heavy computing cores, accelerators of various types, SIMD and data-flow processing concepts. In this situation, codes will yet again have to be rewritten in order to utilize the full potential of future computers. It's an endless process that — understandably — doesn't make software developers any happier.

However, the situation isn't quite hopeless. Indeed, new computing systems require a full review of the legacy code. But the algorithms themselves don't change; only the requirements that new computers present to the structure of algorithms and programs change. To support vector computing, data parallelism needs to be built into the innermost loops within a program. The key concern in ensuring the efficient usage of massive parallel computers is to find a representation of an algorithm whereby a large number of computing nodes can work independently from each other, minimizing data exchange. This is true for every generation of parallel com-

[1] Moscow State University, Moscow, Russia
[2]voevodin@parallel.ru
[3]asa@parallel.ru
[4] University of Tennessee, Knoxville, USA
[5]dongarra@eecs.utk.edu

puting systems: the new architecture requires taking a new look at the properties of existing algorithms, in order to find the most efficient way of implementing them.

There are two facts that matter in this situation: the algorithms themselves don't change much, and their properties do not depend on the computing system. This means that once algorithm's properties have been described in detail, this information can be used repeatedly for any computing platform — existing today or in the future.

The idea of a deep *a priori* analysis of properties of algorithms and their implementation formed the basis for the AlgoWiki project. The main purpose of the project is to present a description of the fundamental properties of various algorithms giving a complete understanding of both their theoretical potential and the particular aspects of their implementation for various classes of parallel computing systems.

The description of all algorithms in AlgoWiki consists of two parts. The first part describes algorithms and their properties. The second part is dedicated to describing particular aspects of their implementation on various computing platforms. This division is made intentionally, to highlight the machine-independent properties of algorithms which determine their potential and the quality of their implementations on parallel computing systems, and to describe them separately from a number of issues related to the subsequent stages of programming and executing the resulting programs.

AlgoWiki provides an exhaustive description of an algorithm. In addition to classical algorithm properties such as serial complexity, AlgoWiki also presents additional information, which together provides a complete description of the algorithm: its parallel complexity, parallel structure, determinacy, data locality, performance and scalability estimates, communication profiles for specific implementations, and many others.

In AlgoWiki, details matter. Classical basic algorithms must be supplemented with their important practical modifications. For example, AlgoWiki contains a description of a basic point-structured real version of Cholesky factorization for a dense symmetrical positive-definite matrix. In practice, modifications of the algorithm are just as important: a block-structured version, a version for a dense complex-Hermitian matrix (both point-structured and block-structured), versions of the algorithm for sparse matrices, etc. It is also important to consider the use of Cholesky factorization in iterative methods, etc.

Equally important are the details related to particular aspects of an algorithm's implementation on specific parallel computing platforms: multi-core processors, SMP, clusters, accelerators, using vector processing and so on. In many cases it is necessary to go one step lower, describing the implementation of an algorithm, for example, not just for a specific accelerator, but to single out relatively important individual cases, such as GPU and Xeon Phi. At the same time, when we provide data about execution time, performance and scalability, we are not only laying out some estimates of the possible implementation quality of a given algorithm on a specific computer, but also setting the foundation for comparative analysis of various computing platforms with regards to the algorithms provided in AlgoWiki.

The outcome of the AlgoWiki project is an open encyclopedia of algorithm properties and the particular aspects of their computer implementation. It was started with a focus on using Wiki technologies, enabling the entire computing community to collaborate on algorithm descriptions. Currently, the encyclopedia is actively being expanded by outside experts; a multi-lingual version is also in the works which will eventually become the main version. The pilot version of the encyclopedia is available at `http://algowiki-project.org/en/`.

1. Background and Related Work

Today, efficiency and parallelism support throughout the entire supercomputing software stack are the central topics in all supercomputing forums worldwide. The same issues are being discussed within major international projects that formulate key challenges and prepare roadmaps of supercomputing software development for decades to come, e.g., International Exascale Software Project [1], Big Data and Extreme Computing [2], European Exascale Software Initiative I and II [3].

In many cases, research on the process of mapping algorithms onto parallel computing system architecture comes down to studying types of algorithm structures which are used as building blocks for a number of programs in many different subject areas. For example, a group of researchers at Berkeley are using the term “motifs” (previously “dwarfs”) to describe methods that use certain common templates for computations and communications [4, 5]. Sandia National Laboratories runs the Mantevo project aimed at studying mini-applications, which represent computational kernels for various scientific and engineering applications [6, 7]. A similar approach is used by the TORCH project [8–10].

Studies of parallel algorithm properties have been conducted for a long time, starting with this fundamental work [11], and a rather wide range of algorithms has been considered in the work [12]. In many cases the authors stick to one specific subject area; for example, parallel algorithms for linear algebra are reviewed in the works [13–15].

In this project, the key aspect is using an algorithm graph (also known as an information graph or dependency graph in literature) for identifying and utilizing algorithm properties. The idea of true information dependency and its usage for software transformation was described in [16], and laid the foundation for a theory of studying the properties of algorithms and programs developed in [17–19].

There are no direct analogues to AlgoWiki — an open encyclopedia of algorithm properties being built within this project. While there are a number of projects attempting to classify and describe properties, and write up implementations of various algorithms [20–23], none of them follows a unified predefined structure to describe all relevant algorithm properties and parallel implementations for various target architectures.

2. A Description of Algorithm Structure and Properties

All fundamentally important issues affecting the efficiency of the resulting parallel programs must be reflected in the description of the properties and structure of the algorithms. With this in mind, an algorithm description structure was developed, which formed the basis for the AlgoWiki encyclopedia. The encyclopedia offers standardized elements for different sections and recommendations for building them, so that descriptions of different algorithms could easily be compared.

Sections 3 and 4 of this paper describe the two parts that form the description of each algorithm within AlgoWiki. Structure of these sections repeats exactly the structure of the description (ten subsections for the Part I and seven subsections for the Part II).

3. AlgoWiki Encyclopedia: "PART I. Algorithm Structure and Properties"

Algorithm properties are independent of the computing system, and, in this regard, this part of AlgoWiki is an important thing by itself. An algorithm description is only prepared once; afterwards, it is used repeatedly for implementation within various computing environments. Even though this part is dedicated to the machine-independent properties of algorithms, things to consider during the implementation process or links to respective items in the second part of AlgoWiki are acceptable here.

3.1. General description of algorithms

This section contains a general description of the problems the algorithm is intended to address. The description shows specific features of the algorithm itself and objects it works with.

3.2. Mathematical description of algorithms

A mathematical description of the problem to be addressed is presented as a combination of formulas, as it is commonly described in textbooks. The description must be sufficient for an unambiguous understanding of the description by a person who is familiar with mathematics.

3.3. Computational kernel of algorithms

The computational kernel (the part of algorithm that takes up most of the processing time) is separated and described here. If an algorithm has more than one computational kernel, each one is described separately.

3.4. Macro structure of algorithms

If an algorithm relies on other algorithms as its constituent parts, this must be specified in this section. If it makes sense to provide the further description of the algorithm in the form of its macro structure, the structure of macro operations is described here. Typical macro operations include finding the sum of vector elements, dot product, matrix-vector multiplication, solving a system of linear equations of small rank, calculating the value of a function at a specific point, searching for the minimum value in an array, matrix transposition, reverse matrix calculation, and many others.

The macro structure description is very useful in practice. The parallel structure for most algorithms is best seen at the macro level, while reflection of all operations would clutter the picture.

The choice of macro operations is not fixed; by grouping macro operations in different ways it is possible to emphasize different properties of the algorithms. In this regard, several macro structure options can be shown here to provide an additional aspect of its overall structure.

3.5. A description of algorithms' serial implementation

This section describes the steps that need to be performed in the serial implementation of this algorithm. To a certain degree, this section is redundant, as the mathematical description already contains all the necessary information. However, it is still useful; the implementation of the algorithm is clearly laid out, helping to unambiguously interpret the properties and estimates presented below.

3.6. Serial complexity of algorithms

This section of the algorithm description shows an estimate of its serial complexity, i.e., the number of operations that need to be performed if the algorithm is executed serially (in accordance with section 3.5). For different algorithms, the meaning of an operation used to evaluate its complexity can vary greatly. This can include operations on real numbers, integers, bit operations, memory reads, array element updating, elementary functions, macro operations, etc. LU factorization is dominated by arithmetic operations on real numbers, while only memory reads and writes are important for matrix transposition; this must be reflected in the description.

For example, the complexity of a vector elements pairwise summation is $n-1$. The complexity of fast Fourier transformation (Cooley-Tukey algorithm) for vector lengths equals a power of two — $n \log_2 n$ complex addition operations and $(n \log_2 n)/2$ complex multiplication operations. The complexity of a basic Cholesky factorization algorithm (point-structured version for a dense symmetrical and positive-definite matrix) is n square root calculations, $n(n-1)/2$ division operations, and $(n^3 - n)/6$ multiplication and addition (subtraction) operations each.

3.7. Information graph

This is a very important part of the description. This is where one can show (see) the parallel structure of the algorithm, for which there is a description and a picture of its information graph [19]. There are many interesting options for reflecting the information structure of algorithms in this section. Some algorithms require showing a maximum-detail structure, while using only the macro structure is sufficient for others. A lot of information is available in various projections of the information graph, which make its regular components stand out, while minimizing insignificant details.

Overall, the task of displaying an algorithm graph is non-trivial. To begin with, the graph is potentially endless, as its number of vertices and edges is determined by the values of external variables, which can be very large. Situations like this can usually be saved by the "similarity" approach, which makes graphs for different values of an external variable "similar": in most cases it is enough to present a relatively small-sized graph, and the graphs for other values will be "exactly the same". In practice, it isn't always so simple, and one needs to be very careful.

Further, an algorithm graph is potentially a multi-dimensional object. The most natural system for placing the vertices and edges of the information graph is based on the nesting loops in the algorithm implementation. If the nesting loop level does not exceed three, the graph can be placed in the traditional three-dimensional space; more complex loop constructs with a nesting level of 4 or more require special graph display and presentation methods.

Fig. 1 shows the information structure of a matrix multiplication algorithm and of an algorithm for solving a system of linear algebraic equations with a block-structured bidiagonal matrix. A more complex example is shown in fig. 2, it demonstrates the information structure of a Cholesky algorithm with input and output data.

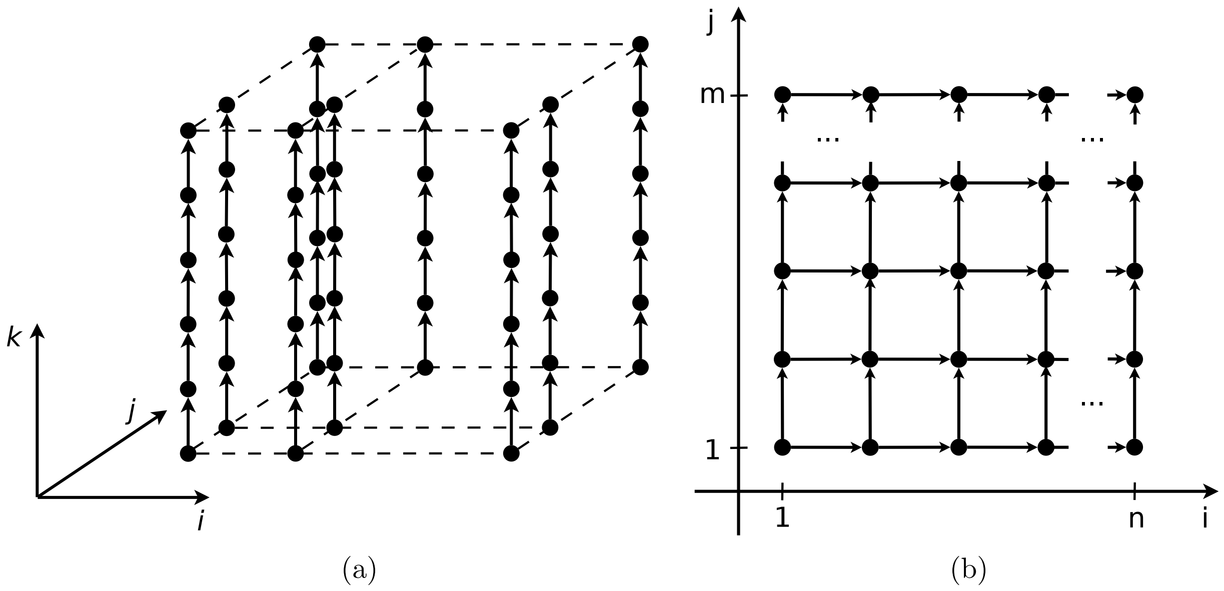


Figure 1. Information structure of a matrix multiplication algorithm (a) and of an algorithm for solving a system of linear algebraic equations with a block-structured bidiagonal matrix (b)

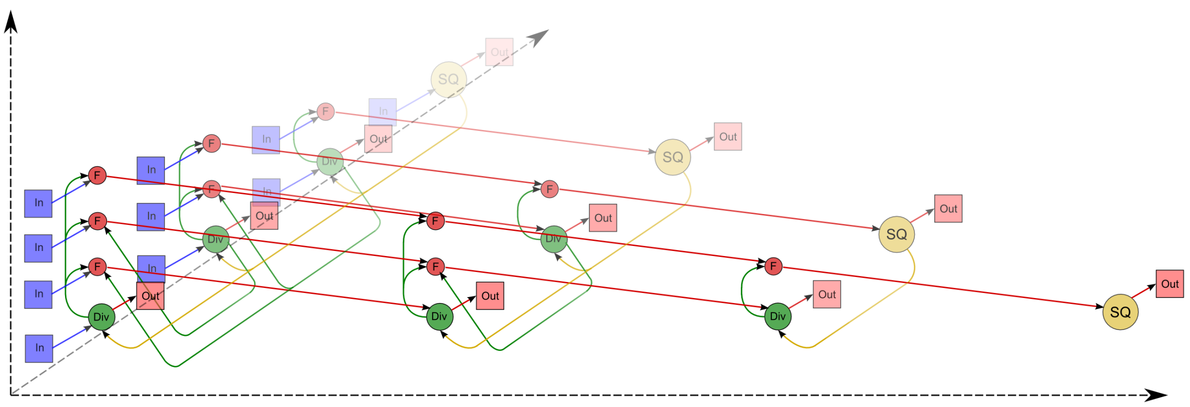


Figure 2. Information structure of a Cholesky algorithm with input and output data: SQ is the square-root operation, F is the operation a-bc, Div is division, In and Out indicate input and output data

3.8. Describing the resource parallelism of algorithms

This section shows an estimate of the algorithm's parallel complexity: the number of steps it takes to execute the algorithm assuming an infinite number of processors (functional units, computing nodes, cores, etc.). The parallel complexity of an algorithm is understood as the height of its canonical parallel form [19]. It is necessary to indicate in terms of which operations the estimate is provided. It is also necessary to describe the balance of parallel steps by the number and type of operations, which determines the layer width in the canonical parallel form and the composition of operations within the layers.

Parallelism in algorithms frequently has a natural hierarchical structure. This fact is highly useful in practice and must be reflected in the description. As a rule, such hierarchical parallelism structures are well reflected in the serial implementation of the algorithm through the program's loop profile, which can well be used to reflect the parallelism resource.

Describing the parallelism resource for an algorithm requires specifying key parallel branches in terms of finite and mass parallelism. The parallelism resource isn't always expressed in simple terms, e.g., through coordinate parallelism; the skewed parallelism resource is equally important. Unlike coordinate parallelism, skewed parallelism is much harder to use in practice, but it is an important option to keep in mind, as sometimes it is the only option. A good illustration is the algorithm structure shown in fig. 1b: there is no coordinate parallelism, but skewed parallelism is there, and using it reduces the complexity from $n \times m$ in serial execution to $(n + m - 1)$ in the parallel version.

For example, let's look at the algorithms, for which serial complexity has been considered in section 3.6. The parallel complexity for vector elements pairwise summation is $\log_2 n$, with the number of operations at each level decreasing from $n/2$ to 1. The parallel complexity of the fast Fourier transformation (Cooley-Tukey algorithm) for vector lengths equal to a power of two is $\log_2 n$. The parallel complexity of a basic Cholesky factorization algorithm (point-structured version for a dense symmetrical and positive-definite matrix) is n steps for square root calculations, $(n - 1)$ steps for division operations and $(n - 1)$ steps for multiplication and addition operations.

3.9. Input/output data description

This section contains a description of the structure, features and properties of the algorithm's input and output data: vectors, matrices, scalars, arrays, a dense or sparse data structure, and their total amount.

3.10. Algorithm properties

This section describes other properties of the algorithm that are worth considering in the implementation process. As noted above, there is no connection to any specific computing platform, but since implementation issues always prevail in a project, there is a distinct need to discuss additional properties of an algorithm.

The *computational power* of an algorithm is the ratio of the total number of operations to the total amount of input and output data. Despite its simplicity, this ratio is exceptionally useful in practice: the higher the computational power, the less the overhead cost of moving data for processing, e.g., on a co-processor, accelerator, or another node within a cluster. For example, the computational power of the dot product operation is 1; the computational power of multiplying two square matrices is $2n/3$.

The issue of utmost importance is the *algorithm stability.* Anything related to this notion, particularly the stability estimate, must be described in this section.

The *balance* of the computing process can be viewed from different perspectives. This includes the balance between different types of operations, particularly arithmetic operations (addition, multiplication, division) together or between arithmetic operations and memory access operations. This also includes the balance of operations between parallel branches of the algorithm.

The *determinacy of an algorithm* is also important in practice, and it is understood as the uniformity of the computing process. From this point of view, the classical multiplication of dense matrices is a highly deterministic algorithm, as its structure, given a fixed matrix size, does not depend on the elements of the input matrices. Multiplying sparse matrices, where matrices are stored in a special format, is no longer deterministic: data locality depends on the structure of the input matrices. An iteration algorithm with precision-based exit is also not deterministic, as the number of iterations (and therefore the number of operations) changes depending on the input data.

A serious issue affecting the indeterminacy of a parallel program is a change in the order of execution for associative operations. A typical example is the use of global MPI operations, e.g., finding the sum of elements in a distributed array. An MPI runtime system dynamically chooses the order of operations, assuming associativity; as a result, rounding errors change between runs, leading to changes in the final output of the program. This is a serious and quite common issue today in massive parallel systems, which translates to lack of reproducible results in parallel program execution. This feature is characteristic for the second part of AlgoWiki, dedicated to algorithm implementations. But the issue is quite important and the respective considerations should be mentioned here as well.

It should be noted that in some cases, a lack of determinacy can be “rectified” by introducing the respective macro operations, which makes the structure both more deterministic and more understandable.

“Long” edges in the information graph are a sign of potential challenges in placing data in the computer’s memory hierarchy during the program execution. On the one hand, edge length depends on the specific coordinate system chosen for placing graph vertices, so long as edges can disappear in a different coordinate system (but that can lead to even more long edges appearing elsewhere). On the other hand, regardless of the coordinate system, their presence can signal the need to store data for a long time at a specific hierarchy level, which imposes additional restrictions on the efficiency of the algorithm implementation. One reason for long edges is a scalar value broadcasted over all iterations of a specific loop: in this case long edges do not cause any serious issues in practice.

Compact packing of the information graph is of interest in developing specialized processors or implementing an algorithm on FPGAs; it can also be included in this section.

4. AlgoWiki Encyclopedia: “PART II. Software Implementation of Algorithms”

The Part II of the algorithm description in AlgoWiki deals with all of the components of the implementation process for an algorithm described in Part I. Both the serial and parallel implementations of an algorithm are considered. This part shows the connection between the properties of the programs implementing the algorithm and the features of the computer architecture they are executed on. Data and computation locality are explained, and the scalability and efficiency of parallel software are described along with the performance achieved with a given program. This is also the place to discuss specific aspects of implementation for different computing architecture classes and to provide references to implementations in existing libraries.

4.1. Features of algorithms’ serial implementation

This section describes the aspects and variations of implementing an algorithm within a serial program that affect its efficiency. In particular, it makes sense to mention the existence of block-structured versions of the algorithm implementation, further describing the prospective advantages and drawbacks of this approach. Another important aspect is related to the options for organizing work with data, variations on data structures, temporary arrays and other similar issues. The required parallelism resource and memory amount for different implementation options need to be specified.

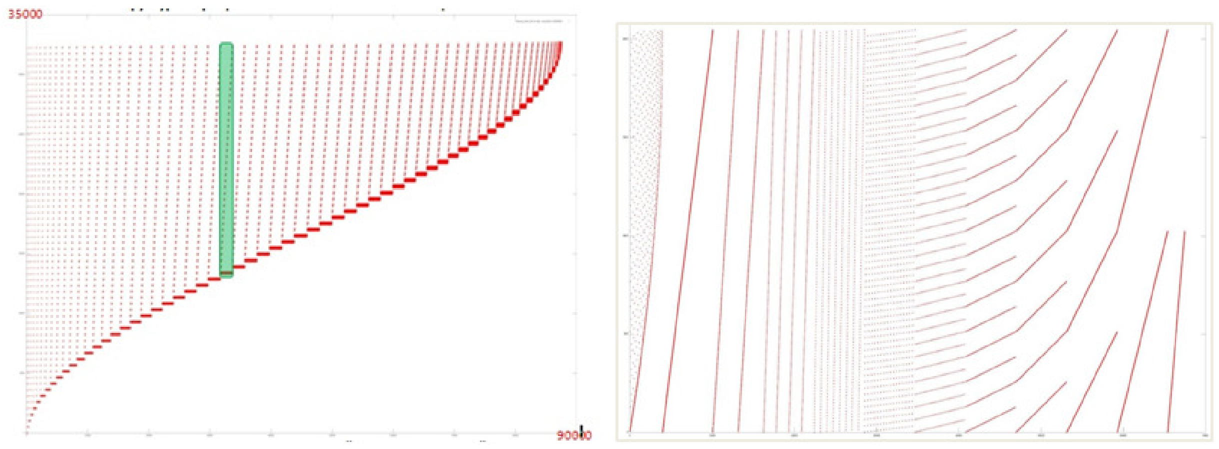
Another important feature is a description of the precision of operations within the algorithm. In practice, it is rarely necessary to perform all arithmetic operations on real numbers with 64-bit floating point arithmetic (double precision), as it doesn't affect the stability of the algorithm or the precision of the results obtained. In this case, if most operations can be performed on a float single precision data, and some fragments require switching to the double, this must be specified here.

Based on information from section 3.8, when describing the serial version of the program, it is worth noting the possibility of equivalent transformaions for programs implementing this algorithm. For example, parallelism of iterations of the innermost loop is normally used for vectorization. However, in some cases this parallelism can be "moved" up the nested loops, which enables more efficient implementation of this algorithm on multi-processor multi-core SMP computers.

4.2. A description of data and computation locality

The issues of data and computation locality are rarely studied in practice, but locality is what affects the program execution efficiency on modern computing platforms. This section provides an estimate of data and computation locality in the program, considering *both the temporal and spacial locality.* Positive and negative locality-related facts should be mentioned, along with what situations could arise under what circumstances. This section also specifies how locality changes when passing from a serial to a parallel implementation. Key patterns in the program's interaction with memory are identified here. Note any possible interrelation between the programming language used and the degree of locality in the resulting programs.

Separately, memory access profiles are specified for computational kernels and key fragments. Fig. 3 shows memory access profiles for programs implementing Cholesky factorization and fast Fourier transformation, which illustrates the difference between the locality properties of the two algorithms.



(a) (b)

Figure 3. Memory access profiles for programs implementing Cholesky factorization (a) and fast Fourier transformation (b)

4.3. Possible methods and considerations for parallel implementation of algorithms

This is a rather big section that must describe key facts and statements that define a parallel program. These can include:

- a hierarchically presented parallelism resource based on the program's loop structure and program call graph;
- a combination (hierarchy) of mass parallelism and finite parallelism;
- possible ways to distribute operations between processes/threads;
- possible strategies to distribute data;
- an estimate of the number of operations, amount and number of data transfers (both the total number and the share for each parallel process);
- an estimate of data locality, etc.

This section should also include recommendations or comments regarding the algorithm's implementation using various parallel programming technologies: MPI, OpenMP, CUDA, or using vectorization directives.

4.4. Scalability of algorithms and their implementation

This section is intended to show the algorithm's scalability limits on various platforms. The impact of barrier synchronization points, global operations, data gather/scatter operations, estimates of strong or weak scalability for the algorithm and its implementations.

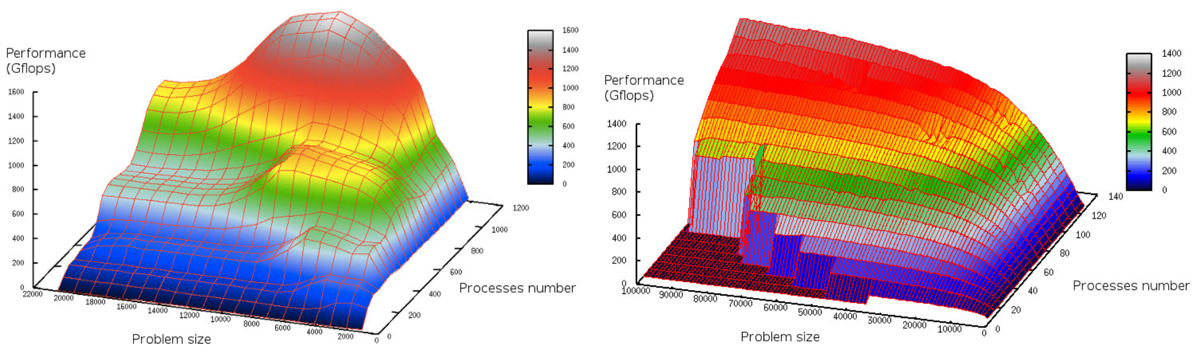
An algorithm's scalability determines the properties of the algorithm regardless of the computer being used. It shows how much the algorithm enables the computer's actual performance to increase with a potentially infinite increase in the number of processors. The scalability of parallel programs is determined in connection with a specific computer and shows how much the performance of this computer running this program can increase when utilizing more computing power.

The central idea of this section is to show the actual scalability of the program implementing a given algorithm on various computing platforms, depending on the number of processors and the size of the problem. It is important to understand a correlation between the number of processors and the size of the problem, so as to reflect all features in behavior of the parallel program, particularly achieving maximum performance, and more subtle effects arising, for example, from the algorithm's block structure or memory hierarchy.

Fig. 4a shows the scalability of a classical matrix multiplication algorithm, depending on the number of processes and the size of the problem. The chart shows visible areas with greater performance, reflecting cache memory levels. Fig. 4b shows the scalability of the Linpack benchmark.

4.5. Dynamic characteristics and efficiency of algorithm implementation

This is a rather large section of AlgoWiki, as evaluating an algorithm's efficiency requires a comprehensive approach and careful analysis of all steps — from the computer architecture to the algorithm itself. Efficiency is understood rather broadly in this section: this includes the efficiency of program parallelization, and the efficiency of program execution relative to the peak performance of computing systems.



(a) (b)

Figure 4. Scalability of programs implementing classical matrix multiplication (a) and Linpack benchmark (b)

In addition to the actual performance, all major reasons should be described that limit further increase in the performance of a given parallel program on a specific computing platform. This is not an easy task, as there is no commonly accepted methodology or respective tools that facilitate such an analysis at the present time. One needs to estimate and describe the efficiency of interaction with the memory subsystem (features of the program's memory access profile), the efficiency of using a resource of parallelism built into the algorithm, the efficiency of interconnect usage (features of the communication profile), the efficiency of input/output operations, etc. Sometimes overall program efficiency characteristics are sufficient; in some cases it is important to show lower-level system monitoring data, such as CPU load, cache misses, Infiniband network usage intensity, etc.

In our studies of parallel programs for the AlgoWiki project, we use Job Digest [24], a tool for building reports on program performance. Fig. 5 shows some results of the execution of a program implementing ten iterations of a Cholesky decomposition for dense real positive-definite matrices.

Fig. 5a shows that, during the runtime of the program, the processor usage level is about 50%. That is a good result for programs executed without the usage of the Hyper-threading technology. Fig. 5b shows the number of floating point operations per second during the Cholesky decomposition exection time. To the end of each iteration, the number of opertations increases intensively. From fig. 5c it follows that the number of L1 cache-misses is large enough (about 25 millions per second on the average for all nodes). Fig. 5d shows that the interconnest IB) is intensively used at each interation. To the end of each iteration, the data transfer intensity increases significantly. Overall, the data obtained from the monitoring system allows one to come to the conclusion that this program was working in an efficient and stable manner. The memory and communication environment usage is intensive, which can lead to an efficiency reduction with an increase of the matrix order or the number of processors in use.

4.6. Conclusions for different classes of computer architecture

This section should contain recommendations on implementing the algorithm for various architecture classes. If the architecture of a specific computer or platform has any specific features affecting the implementation efficiency, this must be noted here.

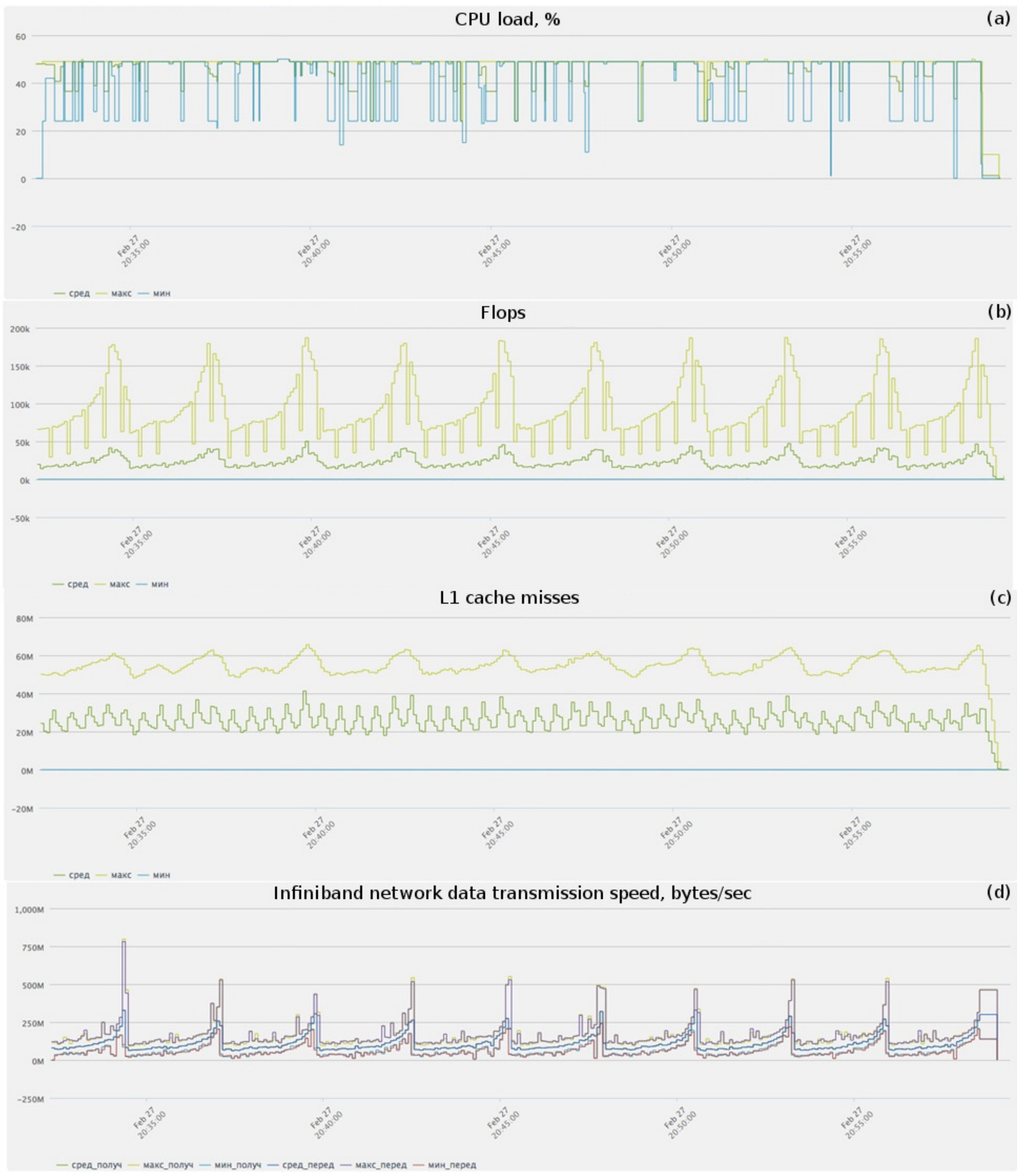


Figure 5. Ten iterations of a Cholesky decomposition for dense real positive-definite matrices: CPU load during program run (a); number of floating point operations per second (b); L1 cache misses (c); Infiniband network data transmission speed, bytes/sec (d)

It is important to point out both positive and negative facts with regards to specific classes of computers. Possible optimization techniques or even “tips and tricks” in writing programs for a target architecture class can be described.

4.7. Existing implementations of algorithms

For many algorithm-computer pairs, good implementations have already been developed which can and should be used in practice. This section is here to provide references to existing serial and parallel implementations of an algorithm that are available for use today. It indicates whether an implementation is open-source or proprietary, what type of license it is distributed under, the distributive location and any other available descriptions. If there is any information on the particular features, strengths and/or weaknesses of various implementations, these can be pointed out here.

Conclusion

AlgoWiki Encyclopedia is an exceptionally large-scale project, which, despite its youth, is actively being developed today and has a number of areas for future development. Materials from the encyclopedia can be used efficiently for a comparative analysis of computing platforms over different applications. We are talking about a direct extension of the methodology used in the Top500 list, which is currently based on just the Linpack test, and by which it is criticized by many researchers. With AlgoWiki, an additional item can be included in Part II for each algorithm to present performance data for various supercomputing platforms. This extension will come naturally for AlgoWiki, and given the encyclopedia's open nature and potential for community contribution, it can become a supplement to the Top500 list, expandable to any algorithm.

An important issue is using the AlgoWiki algorithm information structure in the education process. It is not enough to know the mathematical description, it is vital to understand the structure and special features of every basic step, from formulating the algorithm to its execution. This knowledge is vital in the supercomputing world where everything must be done with the ideas of supercomputing co-design. But this is also necessary for today's ordinary computers, where even smartphones and tablets have become parallel. All of the problems of parallel computing have become important everywhere — from supercomputers to mobile gadgets; this is what brought the AlgoWiki project forward.

The results were obtained with the financial support of the Russian Science Foundation, Agreement N 14-11-00190. The research presented in Sections 4.4 and 4.5 is supported by the Russian Foundation for Basic Research, grant N 13-07-00790.

References

1. Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.C., Barkai, D., Berthou, J.Y., Boku, T., Braunschweig, B. and others: The International Exascale Software Project roadmap // International Journal of High Performance Computing Applications, Vol. 25, No. 1, p.3–60 (2011). DOI: 10.1177/1094342010391989.

2. Big Data and Extreme-scale Computing (BDEC), http://www.exascale.org

3. EESI project — The European Exascale Software Initiative, http://www.eesi-project.eu

4. Asanovi, K., Bodik R., Demmel J., Keaveny T., Keutzer K., Kubiatowicz J. D., et al.:

The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)

5. Asanovi, K., Bodik R., Catanzaro B., Gebis J. J., Husbands P., Keutzer K., et al.: The Landscape of Parallel Computing Research: A View from Berkeley (2006)

6. Christesen C.: Mantevo Views. A Flexible System for Gathering and Analyzing Data for the Mantevo Project. Thesis, College of St. Benedict/St. John's University (2007)

7. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Edwards, H.C., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving Performance via Mini-applications. Sandia National Laboratories, Report SAND2009-5574 (2009). DOI: 10.2172/993908.

8. Kaiser, A., Williams, S., Madduri, K., Ibrahim, K., Bailey, D., Demmel, J., Strohmaier, E.: TORCH Computational Reference Kernels: A Testbed for Computer Science Research, Lawrence Berkeley National Laboratory, Paper LBNL-4172E (2010). DOI: 10.2172/1004197.

9. Kaiser, A., Williams, S., Madduri, K., Ibrahim, K., Bailey, D., Demmel, J., Strohmaier, E.: A Principled Kernel Testbed for Hardware/Software Co-Design Research, Proceedings of the 2nd USENIX Workshop on Hot Topics in Parallelism (HotPar) (2010). DOI: 10.2172/983482.

10. Strohmaier, E., Williams, S., Kaiser, A., Madduri, K., Ibrahim, K., Bailey, D., Demmel, J.: A Kernel Testbed for Parallel Architecture, Language, and Performance Research, International Conference of Numerical Analysis and Applied Mathematics (ICNAAM) (2010)

11. Knuth, D.: The Art of Computer Programming, Volumes 1-4A Boxed Set 3rd Edition (Reading, Massachusetts: Addison-Wesley (2011)

12. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing. WH Press. Cambridge university press (2007). DOI: 10.1145/1874391.187410.

13. Ortega, J.M.: Introduction to Parallel and Vector Solution of Linear Systems, Plenum Press, New York, USA (1988). DOI: 10.1007/978-1-4899-2112-3_1.

14. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J, Eijkhout, V., Pozo, R., Romine, C., Van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition, SIAM, Philadelphia (1994). DOI: 10.1137/1.9781611971538.

15. Saad, Y.: Iterative methods for sparse linear systems, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia (2003)

16. Polychronopoulos, C.D.: Compiler optimizations for enhancing parallelism and their impact on architecture design. IEEE Trans. on Computers, Vol.37, N.8 (1988). DOI: 10.1109/12.2249.

17. Voevodin, V.V.: Mathematical foundations of parallel computing, World Scientific Publishing Co., Series in computer science. Vol.33. 343 p. (1992)

18. Voevodin, V.V.: Information structure of sequential programs. Russ. J of Num. An. and Math Modelling. Vol.10, N3. 279–286 (1995)

19. Voevodin, V.V., Voevodin, Vl.V.: Parallel computing. BHV-St.Petersburg, 608 p. (in Russian) (2002)

20. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, http://www.netlib.org/linalg/html_templates/Templates.html

21. A Library of Parallel Algorithms, http://www.cs.cmu.edu/s̃candal/nesl/algorithms.html

22. Scientific and educational Internet site of RCC on numerical analysis, http://num-anal.srcc.msu.ru/ (in Russian)

23. Wikipedia, List of algorithms, https://en.wikipedia.org/wiki/List_of_algorithms

24. Adinets A.V., Bryzgalov P.A., Voevodin Vad.V., Zhumatii S.A., Nikitenko D.A., Stefanov K.S.: Job Digest: an approach to dynamic analysis of job characteristics on supercomputers, Numerical methods and programming: Advanced Computing, Vol. 13, Section 2, Pp. 160–166 (2012)

Received March 15, 2015.

.