UDC [611.018.51+615.47]:612.086.2

PROFILING OF ENERGY CONSUMPTION BY ALGORITHMS OF SHORTEST PATHS SEARCH IN LARGE DENSE GRAPHS

O.N. Karasik
Technology Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus, PhD in Technical Science
prihozhy@yahoo.com

A.A. Prihozhy
Professor at the Computer and System Software Department, Doctor of Technical Sciences, Full Professor Belarusian National Technical University
karasik.oleg.nikolaevich@gmail.com

O.N. Karasik

Tech Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus; PhD in Technical Science (2019). Interested in parallel computing on multi-core and multi-processor systems.

A.A. Prihozhy

Full professor at the computer and system software department of Belarusian national technical university, D. Sc. (Eng *(1999) and full professor (2001). His research interests include programming and hardware description languages, parallelizing compilers, and computer aided design techniques and tools for software and hardware at logic, high and system levels, and for incompletely specified logical systems. He has over 300 publications in Eastern and Western Europe, USA and Canada. Such worldwide publishers as IEEE, Springer, Kluwer Academic Publishers, World Scientific and others have published his works.*

Abstract: Reducing power consumption when solving computationally intensive problems is an important applied problem in science and industry. The paper presents the results of profiling four algorithms of finding the shortest paths between all pairs of graph vertices, which allowed us to estimate the power consumption and execution time of the algorithms on a multicore system. Profiling was performed on single-threaded implementations of the classical Floyd-Warshall algorithm, the algorithm based on vertex expansion of the graph, the homogeneous Floyd-Warshall block algorithm, and the heterogeneous block algorithm. The experiments used simple complete, oriented weighted graphs ranging in size from 1200 to 4800 vertices. Profiling performed via Intel VTune and Intel SoC Watch showed that the algorithm itself has the largest impact on power consumption. On graphs up to 1200 vertices, the power consumption is not proportionally dependent on the algorithm's execution time. For example, the slow classical Floyd-Warshall algorithm has consumed up to 20 % less energy compared to the faster block algorithms. As the graph size increases, the algorithm based on vertex expansion of the graph becomes strictly dominant; it consumed up to 25 % less energy than the blocked algorithms. With increasing the graph size, a proportional relationship between the algorithm execution time and the amount of energy consumed has been clearly established.

Keywords: multicore processor; profiling; large size graph; shortest path search algorithm; energy consumption; runtime; optimization.

Problem formulation

Software profiling [1 – 3] aims for dynamic program analysis and measurement of the program's time and memory complexity, power and energy consumption, usage of CPU instructions, duration of function calls, etc. Profiling is carried out by a profiler which instruments program source code or binary executable form. The objective of profiling is to gather information for program optimization. The tools of software analysis are important for understanding the program behavior and the utilization of computational resources. The results of software profiling can be used by developers of algorithms and computer programs, computer architects, compiler writers and consumers of software and computer systems. By means of the profiling techniques and tools, it is possible to choose the preferable programming language, compiler, algorithm, software tool, multi-core processor, multi-processor computing system accounting for time, space and power parameters.

In the paper, we have chosen as a benchmark the problem of searching for shortest paths between all pairs of vertices in a large dense graph and have chosen four competitive algorithms of solving the problem of finding optimal solutions on a multi-core system with respect to the time and power consumption parameters. The problem and algorithms have many application domains and are of high commercial importance, therefore, we need to know which combination of algorithm and its parameters yield the lowest runtime and lowest power consumption by the multi-core system. Intel VTune Profiler 2023.0 [1, 2] is chosen as a tool for profiling and measuring the algorithm and program parameters.

All pairs shortest path algorithms

All four competitive algorithms that are selected for profiling target simple complete and dense directed graphs with real edge-weights and without cycle having negative sum of weights. The graphs provide high computational load of the algorithms during profiling, which gives the correct comparison of the algorithms and the comparison of their software implementations.

Floyd-Warshall all-pairs shortest path algorithm (FW). Let $G = (V, E)$ be a simple directed graph with real edge-weights consisting of a set V, $|V| = N$, of vertices and a set E of edges. Let W be the cost adjacency matrix for G: $w_{i,i} = 0$, $1 \le i \le N$; $w_{i,j}$ is the cost (weight) of edge (i, j) if $(i, j) \in E$; $w_{i,j} = \infty$ if $i \ne j$ and $(i, j) \notin E$. When G has no cycle with negative sum of weights, the dynamic programming Floyd-Warshall algorithm (*FW*) computes [4 – 7] a series of distance matrices $D^0 \ldots D^k \ldots D^N$ such that $D^0 = W$ and each element $d_{i,j}^k$ of matrix D^k, $k = 1 \ldots N$, is the length of the shortest path from i to j composed of the subset of vertices labelled 1 to k. The algorithm built on matrix D consists of three nested loops along variables k, i and j, which perform the same kind of calculations on all elements of the matrix. Each iteration along k accesses all elements of D. Totally, every element has N attempts to be updated. *FW* does not provide spatial and temporal data reference locality. The number of loops' iterations is N^3 no matter what the number of edges in the graph is.

Graph-expansion-based all-pairs shortest path algorithm (GEA). The graph-expansion-based algorithm [8] of searching for shortest paths recalculates iteratively the lengths of shortest paths at each step of adding a vertex k to graph G and adding row k and column k to matrix D. The graph-size is changed from 1 to N, and the size of matrix D is changed from 1×1 to $N \times N$. Two operations are used to carry out the changes: adding vertex k with calculating the shortest paths from vertices $1,\ldots,k-1$ to k and from k to $1,\ldots, k-1$; updating shortest paths between pairs of vertices $1,\ldots, k-1$ with accounting for paths passing through k. The operations are used for creating an initial version of pseudocode. The algorithm is inferred by transforming the initial pseudo-code. Formal methods are used to do this: the resynchronization of calculations, reordering of operations, merging loops, etc. The main advantage of *GEA* is the increased temporal and spatial locality of data processing. It is obtained since the algorithm operates on D's submatrices of monotonically increasing size. *GEA* reduces the number of loops' iterations and makes lower the pressure on the processor caches.

Blocked Floyd-Warshall all-pairs shortest path algorithm (BFW). The authors of [9 – 14] proposed a blocked version *BFW* of the Floyd-Warshall algorithm *FW*. *BFW* divides set V of vertices into subsets $V_0 \ldots V_{M-1}$ of size S and splits matrix D into blocks of size $S \times S$ each, creating a block-matrix $B[M \times M]$, where equality $M \cdot S = N$ holds. The main loop of *BFW* performs M iterations, S times less than *FW*. Each iteration consists of three phases: calculation of diagonal block (m, m), which accounts for paths inside the subgraph on subset V_m of vertices; calculation of $(M - 1)$ cross blocks on column m through the diagonal block, which accounts for paths from vertices of V_v to vertices of V_m; calculation of $(M - 1)$ cross blocks on row m through the diagonal block, which accounts for paths from vertices of V_m to vertices of V_v; calculation of $(M - 1)^2$ peripheral blocks on the cross of row v and column u through other blocks m on the row and on the column, which accounts for paths from vertices of V_v to vertices of V_u passing through vertices of V_m. The single block calculation algorithm which implements *FW* computes all four types of blocks. The algorithm exploits spatial (sequential) data locality within each block, which increases the efficiency of the hierarchical cache memory operation. Its main loop has M

iterations, *S* times less compared to *FW*. Therefore, every iteration updates each element of the matrix as many as *S* times, performing the update locally by using one to three blocks simultaneously.

Heterogeneous blocked Floyd-Warshall all-pairs shortest path algorithm (HBFW). Starting from the homogeneous blocked algorithm, the heterogeneous shortest paths algorithm [15 – 22] distinguishes four types of blocks: diagonal, vertical of cross, horizontal of cross, and peripheral. To speed up the computations, new separate algorithms for all block types have been developed, which reduce the number of iterations in nested loops and account for the sequential reference locality of data in CPU caches. The algorithms are faster than the single block calculation algorithm due to accounting for features of the four block types and the data dependences between the blocks. They improve the spatial and temporal reference locality in processing large graphs.

It is interesting that all four algorithms have the same computational complexity. They differ each other by the way and the efficiency of utilization of the multi-core system computational resources, in particular caches.

Profiling tool

We used Intel VTune Profiler 2023.0 (and build in Intel SoC Watch utility) to measure energy consumption [1]. Intel SoC Watch is a command line tool for monitoring metrics related to power consumption on Intel architecture platforms. It can report power states for the system/CPU/GPU/devices, processor frequencies and throttling reasons, total energy consumption over a period, power consumption rate, and other metrics that provide insight into the system's energy efficiency. Intel SoC Watch collects data from both hardware and operating system with low overhead [2]. In our experiments, we focused on the measurement of energy consumption of the CPU package for a full duration of the algorithm execution. The energy consumption of the CPU package includes the energy that is consumed by all cores, by the each-core-separate L1 and L2 private caches, by the shared L3 cache and by other hardware components included into the CPU package. The energy consumption is measured in millijoules (*mJ*). We used the following parameters of the profiling tool:

socwatch.exe -f power -f hw-cpu-pstate -o <output> --program <algorithm application>

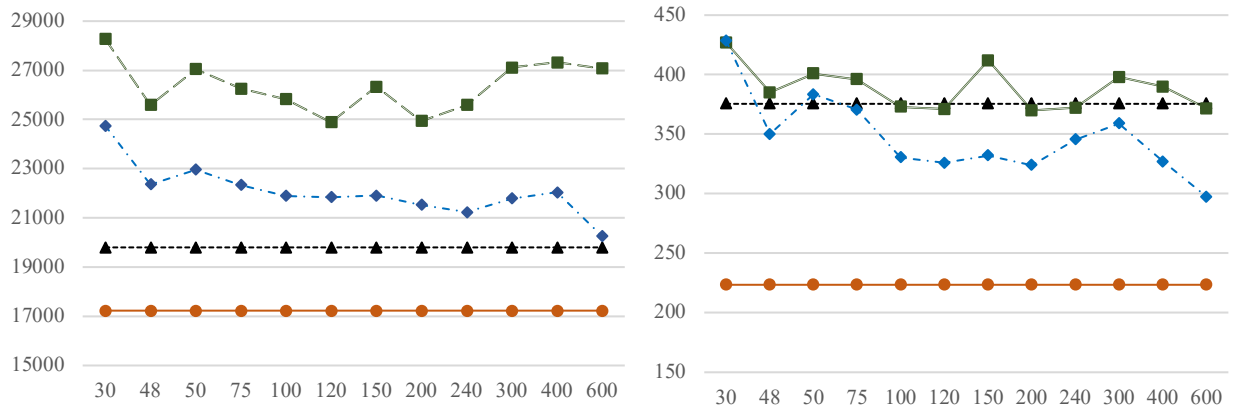
The switch "*-f power*" configures profiling of energy and power metrics; "*-f hw-cpu-pstate*" configures hardware profiling of CPU P-state residence information (the collected information is not reported in the paper); "*-o <output>*" and "*--program <algorithm application>*" switches configure application to store profiling results into *<output>* and to collect information about *<algorithm application>* execution.

Experimental results

All experimental runs of the four shortest path algorithms and their program implementations were carried out on a desktop computer equipped with an Intel Core i7-10700 CPU processor which contains 8 cores (16 hardware threads) with the support of "Intel Turbo Boost 2.0", "Intel Turbo Boost Max 3.0" and "Enhanced Intel SpeedStep" technologies. Every core is equipped with a private L1 (512KB), L2 (2MB) caches and shared L3 (16MB) cache. Its base frequency is 2.90 GHz. Due to the active "Intel Turbo Boost 2.0 Technology" the frequency can increase up to 4.70 GHz, and due to the active "Intel Turbo Boost Max 3.0 Technology" it can increase up to 4.80 GHz. The algorithms were implemented in the C++ language using GNU GCC compiler v12.2.0.

We conducted a series of experiments on multiple randomly generated complete directed weighted graphs of 1200, 2400, 3600 and 4800 vertices. Every experiment was repeated several times and results were verified against the results of original Floyd-Warshall algorithm executed on the same graphs. All runs of blocked algorithms were done on multiple block sizes: 30x30, 48x48, 50x50, 75x75, 100x100, 120x120, 150x150, 200x200, 240x240, 300x300, 600x600, 1200x1200 and 2400x2400. All block sizes divide the matrix into equal blocks without remainders (for smaller graphs some of the block sizes were omitted). To understand the energy efficiency related to the discussed algorithms we measured the execution time, the average power and the total energy consumed by the processor package. In all the experiments, the results are represented with figures and pictures corresponding to algorithms *FW*, *GEA*,

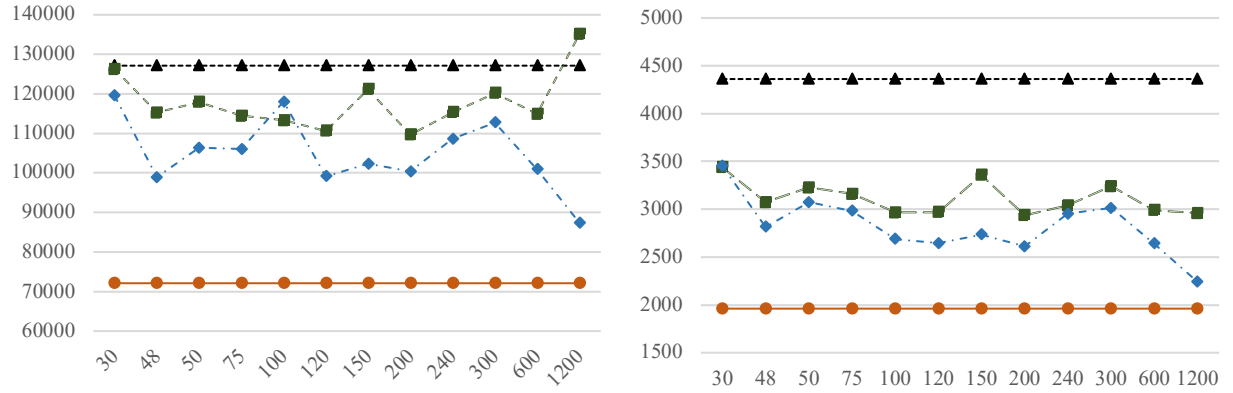
BFW and *HBFW*. Figures 1–4 depict the execution time (*ms*) and energy consumption (*mJ*) depending on the block size (and for non-block algorithms the values are repeated across all the block sizes).



a b

Figure 1. a) Energy consumption (*mJ*) and b) execution time (*ms*) of *FW* (triangle and dash line), *GEA* (circle and solid line), *BFW* (square and long-dash line) and *HBFW* (diamond and dash-dot line) algorithms across all the block sizes on the graph of 1200 vertices

On the graphs of 1200 vertices (Figure 1), algorithm *GEA* demonstrated the lowest energy consumption (13 % better than *FW*) and the lowest execution time (25 % faster than *HBFW*) against all algorithms under consideration. Both block algorithms *BFW* and *HBFW* demonstrated higher energy consumption than *FW* by 25 % and 2 % respectively. In terms of runtime, *FW* and *BFW* showed the comparable execution time, while *HBFW* outranked both showing by 21 % lower execution time.

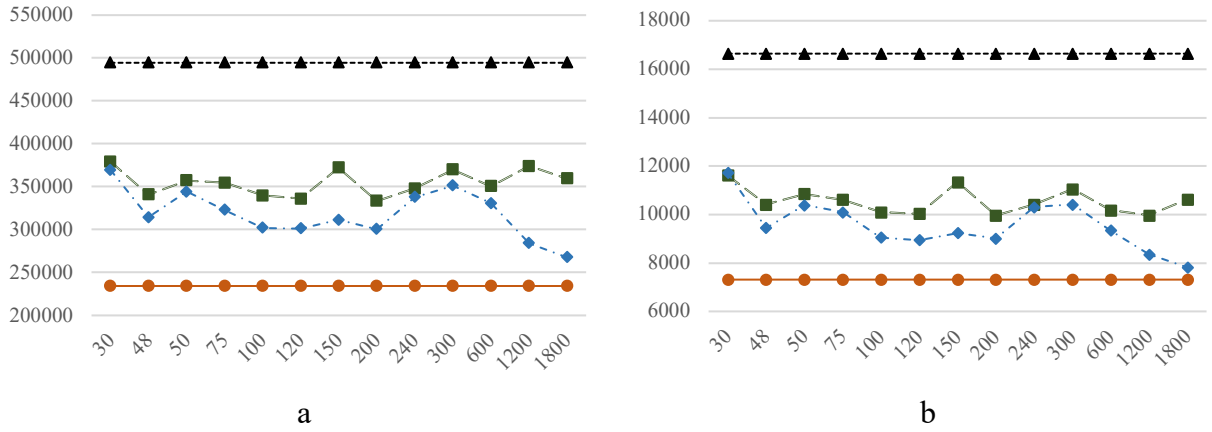


a b

Figure 2. a) Energy consumption (*mJ*) and b) execution time (*ms*) of *FW* (triangles and dash line), *GEA* (circle and solid line), *BFW* (squares and long-dash line) and *HBFW* (diamonds and dash-dot line) algorithms across all the block sizes on the graph of 2400 vertices

On larger graphs of 2400 vertices (Figure 2), the situation is slightly changed. The *GEA* algorithm remains the best one in both energy consumption and execution time. However, the blocked algorithms *BFW* and *HBFW* demonstrated lower energy consumption (by 14 % and 32 % respectively) and smaller

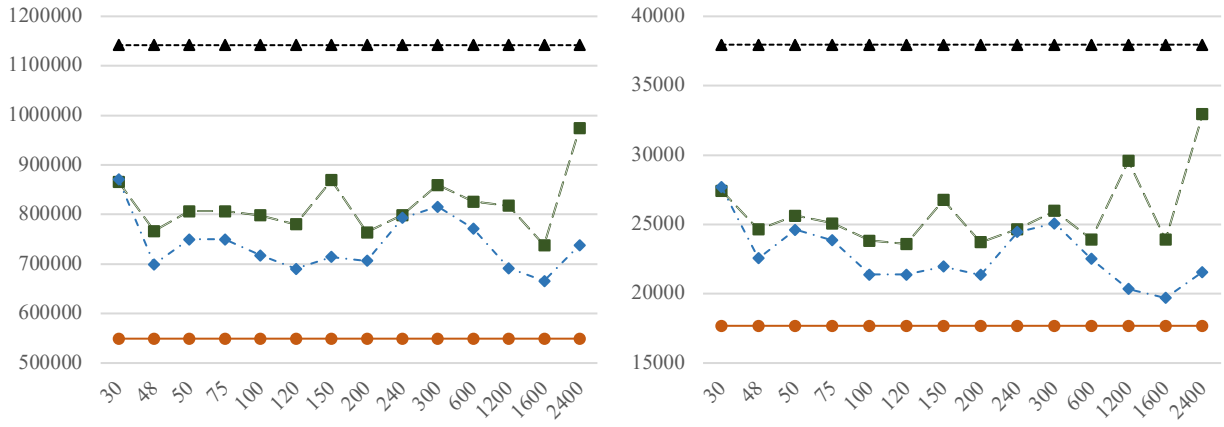
execution time (by 33 % and 49 % respectively) against *FW*. The comparison of *BFW* against *HBFW* shows that *HBFW* has lower energy consumption and lower execution time over *BFW*.



a b

Figure 3. a) Energy consumption (*mJ*) and b) execution time (*ms*) of *FW* (triangles and dash line), *GEA* (circle and solid line), *BFW* (squares and long-dash line) and *HBFW* (diamonds and dash-dot line) algorithms across all the block sizes on the graph of 3600 vertices

On graphs of 3600 and 4800 vertices (Figure 3 and Figure 4), the algorithms demonstrated the same patterns of behavior. *GEA* remains the best one among all the algorithms under profiling. Both *BFW* and *HBFW* showed consistently lower energy consumption and lower execution time than *FW*. *HBFW* continues to demonstrate the reduction of energy consumption and execution time against *BFW*.



a b

Figure 4. a) Energy consumption (*mJ*) and b) execution time (*ms*) of *FW* (triangles and dash line), *GEA* (circle and solid line), *BFW* (squares and long-dash line) and *HBFW* (diamonds and dash-dot line) algorithms across all the block sizes on the graph of 4800 vertices

Table 1 and Table 2 report the data which summarizes the information presented in Figures1 – 4 and compares the best runs of the *GEA*, *BFW* and *HBFW* algorithms over the *FW* algorithm in terms of energy consumption and execution time. We can observe that *GEA* significantly outranks all other algorithms, and *HBFW* is preferable against *BFW*. *BFW* gives better results at smaller sizes of blocks, and *HBFW* gives better results at larger block-sizes.

Table 1. Comparison of best runs (on lowerst energy consumption) of *GEA*, *BFW* and *HBFW* algorithms against *FW* algorithm along all experimental graphs in terms of energy consumption and execution time.

Graph Size	1200			2400			3600			4800		
Algorithm	GEA	BFW	HBFW	GEA	BFW	HBFW	GEA	BFW	HBFW	GEA	BFW	HBFW
Block Size		120	600		200	1200		200	1800		1600	1600
Execution Time (*ms*)	0.595	0.988	0.792	0.450	0.673	0.514	0.439	0.599	0.470	0.466	0.630	0.519
Energy Consumed (*mJ*)	0.870	1.257	1.023	0.567	0.863	0.687	0.474	0.674	0.541	0.481	0.646	0.583

Table 2. Comparison of best runs (on lowerst execution time) of *GEA*, *BFW* and *HBFW* algorithms against *FW* algorithm along all experimental graphs in terms of energy consumption and execution time.

Graph Size	1200			2400			3600			4800		
Algorithm	GEA	BFW	HBFW	GEA	BFW	HBFW	GEA	BFW	HBFW	GEA	BFW	HBFW
Block Size		200	600		200	1200		200	1800		120	1600
Execution Time (*ms*)	0.595	0.985	0.792	0.450	0.673	0.514	0.439	0.599	0.470	0.466	0.621	0.519
Energy Consumed (*mJ*)	0.870	1.260	1.023	0.567	0.863	0.687	0.474	0.674	0.541	0.481	0.683	0.583

Conclusion.

Profiling of program code executed on a multi-core system is an effective means of measuring and estimating parameters of competitive algorithms and detecting areas of their preferable usage. It can help to identify and reduce the consumption of computational and energy resources significantly when solving fundamental tasks such as searching for the shortest paths between all pairs of vertices in a graph. In particular, the research results obtained in the paper show that the recently proposed algorithm that is based on stepwise addition of vertices to the graph has convincing time and energy advantages over the well-known classical Floyd-Warshall algorithm and its blocked versions that target the increase of cache efficiency and establishing parallel computations on multicore systems.

References

[1] Intel Corporation. Intel SoC Watch and Intel VTune Profiler [Electronic resource]. Mode of access: https://www.intel.com/content/www/us/en/docs/system-bring-up-toolkit/get-started-gui-de-windows/2022-1/intel-soc-watch-and-intel-vtune-profiler.html. Date of access: 18.03.2023.

[2] Intel SoC Watch User Guide for Windows OS [Electronic resource]. Mode of access: https://www.intel.com/content/dam/develop/public/us/en/documents/intel-soc-watch-user-guide-external-windows.pdf. Date of access: 18.03.2023

[3] Prihozhy A.A. Simulation of direct mapped, k-way and fully associative cache on all pairs shortest paths algorithms. System analysis and applied information science, 2019, No. 4, pp. 10 – 18.

[4] Floyd, R.W. Algorithm 97: Shortest path. Communications of the ACM, 1962, 5(6), p.345.

[5] Singh, A., Mishra, P.K. Performance Analysis of Floyd Warshall Algorithm vs Rectangular Algorithm. International Journal of Computer Applications, Vol.107, No.16, 2014, pp. 23 – 27.

[6] Madkour, A, Aref, W.G., Rehman, F.U., Rahman, M.A., Basalamah, S. A Survey of Shortest-Path Algorithms. ArXiv:1705.02044v1 [cs.DS] 4 May 2017, 26 p.

[7] *Pettie, S. A new approach to all-pairs shortest paths on real-weighted graphs. Theoretical Computer Science. **312** (1), 2004: 47 – 74.*

[8] Prihozhy A., Karasik O. Inference of shortest path algorithms with spatial and temporal locality for Big Data processing // Сборник материалов VIII Международной научно-практической конференции. Минск: Беспринт, 2022. P. 56 – 66.

[9] Venkataraman, G., Sahni, S., Mukhopadhyaya, S. A Blocked All-Pairs Shortest Paths Algorithm. Journal of Experimental Algorithmics (JEA), Vol 8, 2003, pp. 857 – 874.

[10] Park, J.S., Penner, M., and Prasanna, V.K. Optimizing graph algorithms for improved cache performance. IEEE Trans. on Parallel and Distributed Systems, 2004, 15(9), pp. 769 – 782.

[11] Albalawi, E., Thulasiraman, P., Thulasiram, R. Task Level Parallelization of All Pair Shortest Path Algorithm in OpenMP 3.0. 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013), 2013, Los Angeles, CA, July 1-2, 2013, pp. 109 – 112.

[12] Tang, P. Rapid Development of Parallel Blocked All-Pairs Shortest Paths Code for Multi-Core Computers. IEEE SOUTHEASTCON 2014, pp. 1 – 7.

[13] Karasik O.N., Prihozhy A.A. Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation. System analysis and applied information science, 2022, No. 3, pp. 57 – 65. https://doi.org/10.21122/2309-4923-2022-3-57-65.

[14] Prihozhy A.A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms. System analysis and applied information science. 2021, No. 3, pp. 40 – 50.

[15] Прихожий, А. А. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа / А. А. Прихожий, О. Н. Карасик // Системный анализ и прикладная информатика. – № 3. – 2017. – С. 68 – 75.

[16] Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algorithm. Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics, 2023, no. 1 (266), pp. 77–83. DOI: 10.52065/2520-6141-2023-266-1-13.

[17] Карасик, О. Н., Прихожий, А. А. Потоковый блочно-параллельный алгоритм поиска кратчайших путей на графе. Доклады БГУИР. – 2018. – № 2. – С. 77 – 84.

[18] Прихожий А.А., Карасик О.Н. Исследование методов реализации многопоточных приложений на многоядерных системах // Информатизация образования. 2014. № 1. P. 43–62.

[19] Прыхожы, А. А., Карасік, А. М. Кааператыўныя блочна-паралельныя алгарытмы рашэння задач на шмат'ядравых сістэмах. Системный анализ и прикладная информатика. – 2015. – № 2. – С. 10 – 18.

[20] Prihozhy, A.A. Analysis, transformation and optimization for high performance parallel computing. Minsk: BNTU, 2019. – 229 p.

[21] Прихожий, А.А., Карасик, О.Н. Кооперативная модель оптимизации выполнения потоков на многоядерной системе. Системный анализ и прикладная информатика, – 2014. – № 4. – С. 13–20.

[22] Prihozhy A.A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms. System analysis and applied information science. 2021, No. 3, pp. 40–50.

ПРОФИЛИРОВАНИЕ ПОТРЕБЛЕНИЯ ЭНЕРГИИ АЛГОРИТМАМИ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ НА БОЛЬШИХ ПЛОТНЫХ ГРАФАХ

О.Н. Карасик
Ведущий инженер иностранного производственного унитарного предприятия «ИССОФТ СОЛЮШЕНЗ» (ПВТ, г. Минск), к.т.н.

А.А. Прихожий
Профессор кафедры «Программное обеспечение информационных систем и технологий» Белорусского национального технического университета, д.т.н., профессор

ИССофт Солюшенс (часть Кохерент Солюшенс), Беларусь
Беларуский национальный технический университет, Беларусь
E-mail: prihozhy@yahoo.com

Аннотация. Снижение энергопотребления при решении задач, требующих больших вычислительных мощностей, является важной прикладной проблемой в науке и производстве. В статье представлены результаты профилирования четырех алгоритмов поиска кратчайших путей между всеми парами вершин графа, позволившие оценить энергопотребление и время выполнения алгоритмов на многоядерной системе. Профилирование выполнялось на однопоточных реализациях классического алгоритма Флойда-Уоршалла, алгоритма, построенного на вершинном расширении графа, однородного блочного алгоритма Флойда-Уоршалла и неоднородного блочного алгоритма. В экспериментах использовались простые полные, ориентированные взвешенные графы размером от 1200 до 4800 вершин. Профилирование, выполненное посредством Intel VTune и Intel SoC Watch, показало, что наибольшее влияние на энергопотребление оказывает сам алгоритм. На графах до 1200 вершин, потребление энергии может пропорционально не зависеть от времени выполнения алгоритма. Например, медленный классический алгоритм Флойда-Уоршалла потребил до 20% меньше энергии по сравнению с более быстрыми блочными алгоритмами. С увеличением размера графа, алгоритм, построенный на вершинном расширении графа, стал однозначно доминирующим; он потребил до 25% меньше энергии, чем блочные алгоритмы. При увеличении размера графа, четко устанавливается пропорциональная зависимость между временем выполнения алгоритма и количеством потребляемой энергии.

Ключевые слова: многоядерный процессор; профилирование; граф большого размера; алгоритм поиска кратчайших путей; энергопотребление; время выполнения; оптимизация.