АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ
ALGORITHMIC AND PROGRAMMING

УДК 004.272.2

A. A. Prihozhy[1], O. N. Karasik[2]
[1]Belarusian National Technical University
[2]ISsoft Solutions (part of Coherent Solutions)

ADVANCED HETEROGENEOUS BLOCK-PARALLEL ALL-PAIRS SHORTEST PATH ALGORITHM

The problem of finding shortest paths between all pairs of vertices in a large-size graph has many application domains in industry, technology, science, economics, and society. The algorithms solving the problem and proposed in the literature target either lowering a computational complexity or efficient exploitation of computational resources. This paper proposes the advanced heterogeneous block-parallel shortest paths algorithm that is a result of further development and improvement of known blocked algorithms. Starting from the homogeneous blocked algorithm, it distinguishes four types of blocks: diagonal, vertical of cross, horizontal of cross, and peripheral. To speed up the computations, 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 improve the spatial and temporal reference locality in big data processing. Experiments carried out on a server equipped with two Intel Xeon E5-2620 v4 processors have shown the speedup of up to 60–70% the proposed single- and multi-threaded advanced heterogeneous blocked algorithms yield over the single- and multiple-threaded homogeneous blocked Floyd – Warshall algorithms.

Keywords: shortest path, blocked algorithm, heterogeneous algorithm, multi-core system, throughput.

For citation: 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.

А. А. Прихожий[1], О. Н. Карасик[2]
[1]Белорусский национальный технический университет
[2] Иностранное производственное унитарное предприятие «Иссофт Солюшенз»

УСОВЕРШЕНСТВОВАННЫЙ РАЗНОРОДНЫЙ БЛОЧНО-ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ НА ГРАФЕ

Задача нахождения кратчайших путей между всеми парами вершин графа имеет множество областей применения в промышленности, технике, науке, экономике и обществе. Предложенные в литературе алгоритмы, решающие эту задачу на графах большого размера, направлены либо на снижение вычислительной сложности, либо на эффективное использование ресурсов вычислительной системы. В статье предлагается усовершенствованный разнородный блочно-параллельный алгоритм поиска кратчайших путей, который является результатом дальнейшего развития и улучшения известных блочных алгоритмов. Отталкиваясь от однородного блочного алгоритма, он различает четыре типа блока: диагональный, вертикальный перекрестный, горизонтальный перекрестный и периферийный. Для ускорения вычислений в статье разработаны отдельные алгоритмы для каждого типа блока, уменьшающие количество итераций во вложенных циклах и учитывающие строковое размещение данных в кэшах процессора. Алгоритмы улучшают пространственную и временную локальность при обработке больших данных. Эксперименты, проведенные на сервере, оснащенном двумя процессорами Intel Xeon E5-2620 v4, показали повышение производительности предложенных одно- и многопоточных усовершенствованных разнородных блочных алгоритмов до 60–70% по сравнению с одно- и многопоточными блочными алгоритмами Флойда – Уоршелла.

Ключевые слова: кратчайший путь, блочный алгоритм, разнородный алгоритм, многоядерная система, производительность.

Для цитирования: Прихожий А. А., Карасик О. Н. Усовершенствованный разнородный блочно-параллельный алгоритм поиска кратчайших путей на графе // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2023. № 1 (266). С. 77–83. DOI: 10.52065/2520-6141-2023-266-1-13 (На англ.).

Introduction. The problem of searching for shortest paths in weighted graphs is widely exploited to find optimal solutions in industry, technology, science, economics, and society. The algorithms [1–15] of solving the problem either try to decrease the computational complexity, or to efficiently account for the features of computer architectures to speedup computations. In the paper, we consider all pairs shortest path problem on large graphs. The blocked algorithms [10–15] aim for efficient parallelization on multi-processor (multi-core) systems and utilization of CPU hierarchical memory. The contribution of the paper is the development of an advanced fast blocked heterogeneous algorithm for shortest paths search in two versions: single- and multiple-threaded parallel.

Main part. Let a graph $G = (V, E)$ be constructed of a vertex set V of cardinality N and a set E of edges represented by matrix W of weights. If $w_{i,j} \neq \infty$, it is a weight of edge (i, j), otherwise $(i, j) \notin E$. In the blocked Floyd – Warshall (*BFW*) Algorithm 1, a matrix of path distances is decomposed into matrix $B[M \times M]$ of blocks with block-size S and block-count $M = N / S$. Figure 1 illustrates the process and order of calculating blocks of $D0$, $C1$, $C2$ and $P3$ types. The universal *BCA* Algorithm 2 is a classic Floyd – Warshall algorithm, in which $b1_{i,j}$, $b2_{i,k}$ and $b3_{k,j}$ are elements of blocks B^1, B^2 and B^3 respectively. It recalculates each block element S times; therefore, has the computational complexity of $O(S^3)$. The universalism of *BCA* is a source of slowing down the computations in *BFW*.

In the paper, instead of the single *BCA*, we introduce four other $D0$, $C1$, $C2$ and $P3$ algorithms, which have the same names as the block types have, and have the numbers of 1, 2, 2 and 3 arguments respectively. The objective of our work is to make the new algorithms $D0$, $C1$, $C2$ and $P3$ faster by accounting for features of the block types and the data dependences between blocks. Each block has the row-major layout in CPU hierarchical memory and support spatial locality during *BFW* operation. At $M = 2$, *BFW* calls the four algorithms in the order as follows:

$D0(B_{0,0})$, $C1(B_{1,0}, B_{0,0})$, $C2(B_{0,1}, B_{0,0})$;
$P3(B_{1,1}, B_{1,0}, B_{0,1})$, $D0(B_{1,1})$, $C1(B_{0,1}, B_{1,1})$;
$C2(B_{1,0}, B_{1,1})$, $P3(B_{0,0}, B_{0,1}, B_{1,0})$.

The following procedure lies in the basis of $D0$, $C1$, $C2$ and $P3$ construction:

Algorithm 1: Blocked Floyd–Warshall (*BFW*)

Input: A number N of graph vertices
Input: A matrix $W[N\times N]$ of graph edge weights
Input: A size S of block
Input: A number $M \leftarrow N / S$ of blocks
Output: A blocked matrix $B[M\times M]$ of path distances
  $B \leftarrow W$
  for $m \leftarrow 0$ **to** $M - 1$ **do**
    $B_{m,m}^{m+1} \leftarrow BCA(B_{m,m}, B_{m,m}, B_{m,m})$ // D0
    for $v \leftarrow 0$ **to** $M - 1$ **do**
      if $v \neq m$ **then**
        $B_{v,m}^{m+1} \leftarrow BCA(B_{v,m}, B_{v,m}, B_{m,m})$ // C1
        $B_{m,v}^{m+1} \leftarrow BCA(B_{m,v}, B_{m,m}, B_{m,v})$ // C2
    for $v \leftarrow 0$ **to** $M - 1$ **do**
      if $v \neq m$ **then**
        for $u \leftarrow 0$ **to** $M - 1$ **do**
          if $u \neq m$ **then**
            $B_{v,u} \leftarrow BCA(B_{v,u}, B_{v,m}, B_{m,u})$ // P3
  return B

Algorithm 2: Block calculation *BCA*

Input: S is size of block
Input: B^1, B^2, B^3 are input blocks
Output: B^1 is recalculated block
  for $k \leftarrow 0$ **to** $S - 1$ **do**
    for $i \leftarrow 0$ **to** $S - 1$ **do**
      for $j \leftarrow 0$ **to** $S - 1$ **do**
        $sum \leftarrow b2_{i,k} + b3_{k,j}$
        if $b1_{i,j} > sum$ **then** $b1_{i,j} \leftarrow sum$;
return B^1

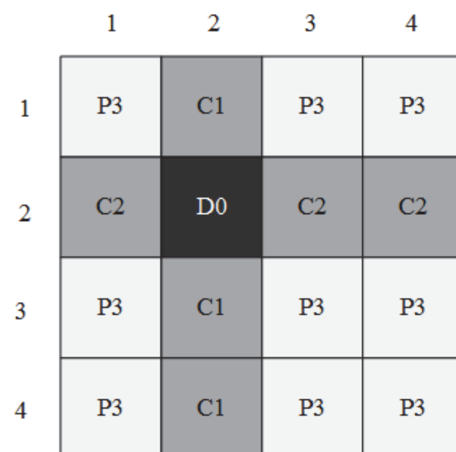

Fig. 1. Illustration of *BFW* operation: cross moves from top-left to bottom-right corner; $C1$ and $C2$ are executed after $D0$, and $P3$ after $C1$ and $C2$

The graph is extended by stepwise adding of vertices.

A sequence $B^1(1)\ldots B^1(k)\ldots B^1(S)$ of states of the B^1 block is generated where k is a current number of rows and/or columns the block has.

The elements of row k and/or column k that are path distances from and to vertex k respectively are calculated over elements of matrix $B^1(k-1)$.

In their turn, the matrix $B^1(k-1)$ is recalculated into $B^1(k)$ over the elements of row k and/or column k.

*Algorithm D*0. Since $D0$ calculates block B^1 over itself without involving other blocks, both row k and column k are added to $B^1(k-1)$. $D0$ starts operation from $B^1(1)$ of size 1×1 and step-by-step increases the block size (Fig. 2). The above-described procedure calculates row k, column k, and $B^1(k)$ over $B^1(k-1)$.

We have applied a set of formal transformations to the procedure and obtained Algorithm 3 ($D0$) of calculating the diagonal block. The transformations include 1) reordering of operations; 2) resynchronizing operation executions from neighbour iterations of loop to calculate $b1_{i,j}(k-1)$ over $b1_{i,j}(k-2)$, $b1_{i,k-1}(k-1)$, and $b1_{k-1,j}(k-1)$, and to calculate $b1_{i,k}(k)$ and $b1_{k,j}(k)$ over $b1_{i,j}(k-1)$, $b1_{i,k}(k-1)$ and $b1_{k,j}(k-1)$ within the same iteration; 3) merging nested loops. $D0$ calculates blocks $b1_{i,j}(k-1)$, $i, j = 0\ldots k-1$ as:

$$b1_{i,j}(k-1) = \min\left\{b1_{i,j}(k-2), z\right\};$$
$$z = b1_{i,k-1}(k-1) + b1_{k-1,j}(k-1).$$

The element $b1_{i,k}(k)$ of column k is calculated using the equation as follows:

$$b1_{i,k}(k) = \min_{j=0\ldots k-1}\left(b1_{i,j}(k-1) + b1_{j,k}(k-1)\right).$$

The element $b1_{k,j}(k)$ of row k is calculated by the following equation:

$$b1_{k,j}(k) = \min_{i=0\ldots k-1}\left(b1_{k,i}(k-1) + b1_{i,j}(k-1)\right).$$

In algorithm $D0$, two loops along i and j have the iteration scheme that produces k iterations, which is less than S iterations in BCA. The loops consequently process matrices of growing size $[1 \times 1]$, $[2 \times 2]$ … $[S \times S]$, which indicate the $D0$ has a temporal locality property. Comparison of $D0$ and BCA shows that $D0$ has smaller number of iterations in the nested loops and reduces the cache pressure. Reduction ρ in the number of executions of the most nested loop body is evaluated with

$$\rho(\mathrm{DO/BCA}) = 6/(2 + 3/S + 1/S^2).$$

When $S \to \infty$, the reduction $\rho(D0/BCA) \to 3$. The number of accesses to memory is also reduced.

*Algorithm C*1. Since it calculates a vertical block B^1 of cross over itself and block B^3, column k is added to $B^1(k-1)$ without adding row k (Fig. 3). $C1$ starts operation with $B^1(1)$ of size $S\times 1$ and increases the block-size in column-wise manner.

Algorithm 3: Calculating diagonal block $D0$

Input: A diagonal block B^1
Input: A size S of block
Output: A recalculated diagonal block B^1
for $k \leftarrow 2$ **to** S **do**
  $k1 \leftarrow k-1$
  for $i \leftarrow 1$ **to** $k1$ **do**
    for $j \leftarrow 1$ **to** $k1$ **do**
      $s_2 \leftarrow b1_{i,k1} + b1_{k1,j}$ **if** $b1_{i,j} > s_2$ **then** $b1_{i,j} \leftarrow s_2$
      $s_0 \leftarrow b1_{i,j} + b1_{j,k}$ **if** $b1_{i,k} > s_0$ **then** $b1_{i,k} \leftarrow s_0$
      $s_1 \leftarrow b1_{k,i} + b1_{i,j}$ **if** $b1_{k,j} > s_1$ **then** $b1_{k,j} \leftarrow s_1$
$k1 \leftarrow S-1$
for $i \leftarrow 1$ **to** $k1-1$ **do**
  for $j \leftarrow 1$ **to** $k1-1$ **do**
    $s_2 \leftarrow b1_{i,k1} + b1_{k1,j}$ **if** $b1_{i,j} > s_2$ **then** $b1_{i,j} \leftarrow s_2$
return B^1

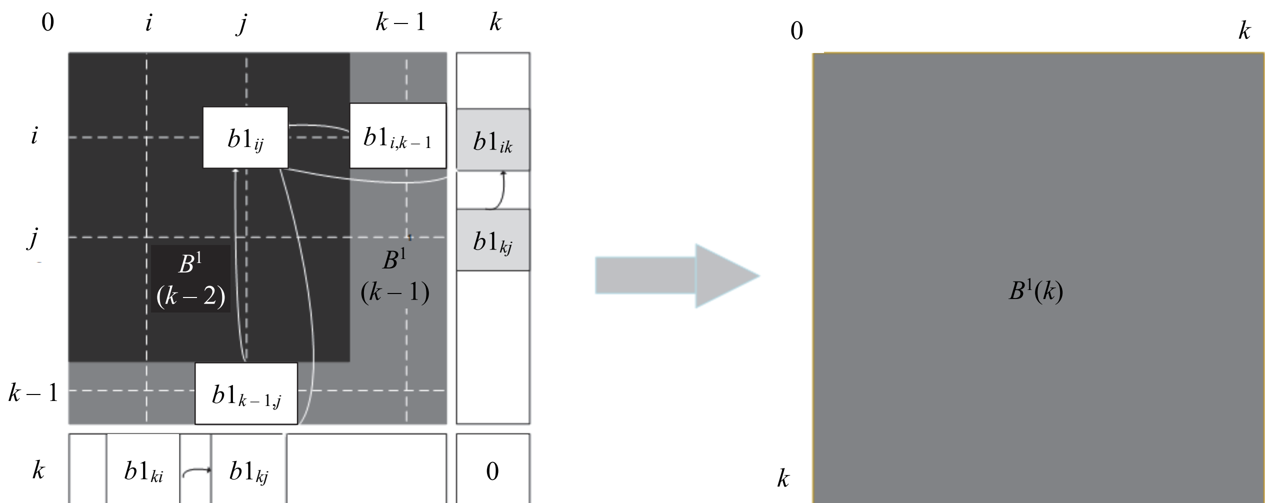

Fig. 2. Recurrent calculation of diagonal block $D0$

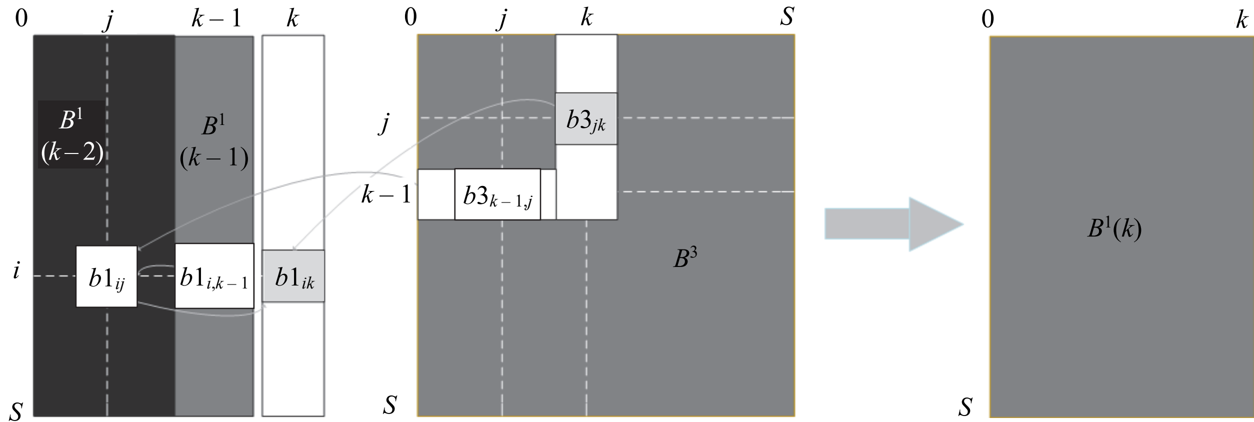

Fig. 3. Recurrent calculation of block $C1$ of cross

The above-described procedure calculates column k and $B^1(k)$ over $B^1(k-1)$. We have applied to the procedure the transformations like those from the $D0$ case and obtained Algorithm 4 ($C1$) of calculating the vertical block of cross. Algorithm $C1$ calculates $b1_{i,j}(k-1)$, $i = 0\ldots S-1$, $j = 0\ldots k-2$ using equation

$$b1_{i,k}(k) = \min\left\{\begin{matrix} b1_{i,j}(k-2), \\ b1_{i,k-1}(k-1) + b3_{j,k}. \end{matrix}\right\}$$

The equation as follows aims for calculating element $b1_{i,k}(k)$ of column k.

$$b1_{i,k}(k) = \min_{j=0\ldots k-1}\left(b1_{i,j}(k-1) + b3_{j,k}\right).$$

In algorithm $C1$, the loop along j has the iteration scheme that produces k iterations, which is less than S iterations in BCA. The loop consequently processes matrices of growing size $[S \times 1]$, $[S \times 2]$ … $[S \times S]$, which indicate that $C1$ has a property of temporal reference locality. Comparison of $C1$ against BCA shows that $C1$ decreases the cache pressure. In $C1$, the overall number of iterations of the most nested loop is smaller than in BCA, and the reduction is evaluated by

$$\rho(\mathrm{C1}/\mathrm{BCA}) = 2/(1 + 5/S + 4/S^2).$$

Algorithm 4: Calculating vertical block $C1$ of cross

Input: A block B^1
Input: A block B^3
Input: A size S of block
Output: A recalculated block B^1
for $k \leftarrow 1$ **to** $S-1$ **do**
  $k1 \leftarrow k-1$;
  for $i \leftarrow 0$ **to** $S-1$ **do**
    for $j \leftarrow 0$ **to** $k1$ **do**
      $s_2 \leftarrow b1_{i,k1} + b3_{k1,j}$ **if** $b1_{i,j} > s_2$ **then** $b1_{i,j} \leftarrow s_2$
      $s_0 \leftarrow b1_{i,j} + b3_{j,k}$ **if** $b1_{i,k} > s_0$ **then** $b1_{i,k} \leftarrow s_0$
$k1 \leftarrow S-1$
for $i \leftarrow 0$ **to** $S-1$ **do**
    for $j \leftarrow 0$ **to** $k1-1$ **do**
$s_2 \leftarrow b1_{i,k1} + b3_{k1,j}$ **if** $b1_{i,j} > s_2$ **then** $b1_{i,j} \leftarrow s_2$
return B^1

When $S \to \infty$, the reduction $\rho(C1/BCA) \to 2$. The number of accesses to memory is also reduced.

Algorithm C2. Since it calculates a horizontal block B^1 of cross over itself and block B^2, only row k is added to $B^1(k-1)$ without adding column k (Fig. 4). $C2$ starts operation with $B^1(1)$ of size $1{\times}S$ and increases the block size in row-wise manner.

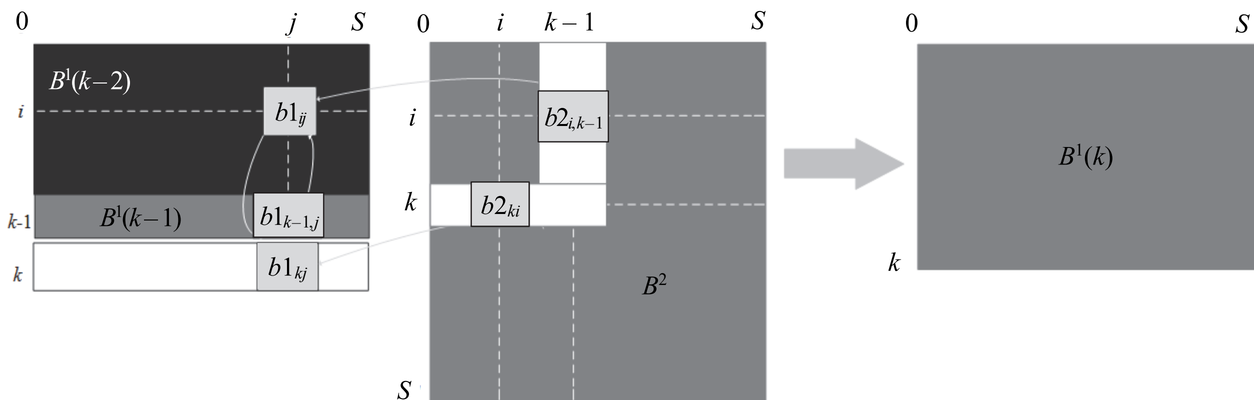

Fig. 4. Recurrent calculation of block $C2$ of cross

The above-described procedure calculates row k and block $B^1(k)$ over $B^1(k-1)$. Transformations like those from the $D0$ case have been applied to the procedure, yielding Algorithm 5 ($C2$) of calculating the horizontal block of cross. $C2$ calculates $b1_{i,j}(k-1)$, $i = 0\dots k-2$, $j = 0\dots S-1$ using equation

$$b1_{i,j}(k-1) = \min \begin{Bmatrix} b1_{i,j}(k-2), \\ b2_{i,k-1}(k-1) + b1_{k-1,j}(k-1). \end{Bmatrix}$$

Algorithm 5: Calculating horizontal block *C2* of cross

Input: A block B^1
Input: A block B^2
Input: A size S of block
Output: A recalculated block B^1

```
for k ← 1 to S – 1 do
    k1 ← k – 1;
    for i ← 0 to k1 do
        for j ← 0 to S – 1 do
            s2 ← b2[i,k1] + b1[k1,j]    if b1[i,j] > s2 then b1[i,j] ← s2
            s0 ← b2[k,i] + b1[i,j]      if b1[k,j] > s0 then b1[k,j] ← s0
k1 ← S – 1
for i ← 0 to k1 – 1 do
    for j ← 0 to S – 1 do
        s2 ← b2[i,k1] + b1[k1,j]    if b1[i,j] > s2 then b1[i,j] ← s2
return B1
```

The following equation aims for calculating element $b1_{i,k}(k)$ of column k.

$$b1_{k,j}(k) = \min_{i=0\dots k-1} \left(b2_{k,i} + b1_{i,j}(k-1) \right).$$

In algorithm $C2$, the loop along i has the iteration scheme that produces k iterations, which is less than S iterations in *BCA*. The loop sequentially processes matrices of growing size $[1 \times S]$, $[2 \times S] \dots [S \times S]$, which indicate the $C2$ obtains a property of temporal reference locality that reduces the cache pressure. In $C2$, the reduction of the overall number of iterations of the most nested loop and of the overall number of accesses to memory is the same as in $C1$.

The nest of three loops across k, i and j of algorithms $D0$, $C1$ and $C2$, carries out computation of block B^1. It calculates variables $s0$, $s1$ and $s2$, and matrix elements $b1_{i,j}$, $b1_{i,k}$ and $b1_{k,j}$ upon elements of blocks B^1, B^2 and B^3. It traverses the rows and columns $k-1$, k and i multiple times. In contrast to the rows which support sequential reference locality in hierarchical memory, the columns are deployed in different lines and increase the data traffic in hierarchical cache memory. To improve the data locality, we have transformed the algorithms to collect the column elements in one-dimensional arrays and then access them intensively.

Algorithm P3. It calculates a peripheral block B^1 over itself and the B^2 and B^3 blocks. Since the graph extension-based procedure cannot be used effectively in this case, we derive $P3$ from transforming the classical Floyd-Warshall Algorithm 2. To provide the sequential reference locality for all blocks B^1, B^2 and B^3, we have reordered the three $k-i-j$ nested loops to the nest $i-k-j$ of loops.

Results. We implemented the homogeneous blocked Floyd-Warshall algorithm (*BFW*) and the advanced heterogeneous blocked algorithm (*HBA*) in C++ language using GNU GCC compiler v10.2.0. Each algorithm has two implementation versions: single-thread and multiple-threaded. The algorithms were parallelized at task level and compiled to multiple-threaded applications by means of OpenMP 4.5. We carried out the vectorisation of operations in all program codes to speed up the computations. The experiments were done on a rack server equipped with two Intel Xeon E5-2620 v4 processors. Each processor contains 8 cores and 16 hardware threads. Every core is equipped with private L1 (32 KB) and L2 (256 KB) caches, and the cores within each processor share inclusive L3 (20 MB) cache.

To measure algorithm parameters on the multi-core system architecture, we used the Intel VTune Profiler 2021.8. The experiments were conducted on randomly generated complete weighted graphs. The paper describes results for graphs of 4800 vertices. Each experiment was carried out multiple times and verified.

Table 1 reports the execution time of the single-thread sequential *BFW* and *HBA*. *BFW* consumed from 35.4 to 54.0 sec of CPU, meanwhile *HBA* consumed from 29.6 to 35.2 sec. The speedup of *HBA* over *BFW* is from 6.1% to 59.6%.

Table 1

Execution time and comparison of single-thread *HBA* and *BFW* on graph 4800

Block-matrix	Block count	*BFW*, sec	*HBA*, sec	Speedup, %
2×2	4	53.99	33.82	59.6
3×3	9	35.40	29.61	19.6
4×4	16	35.83	31.90	12.3
5×5	25	36.15	32.41	11.5
6×6	36	36.03	33.13	8.8
8×8	64	37.34	35.19	6.1

Table 2 reports the execution time of multiple-threaded *BFW* and *HBA* parallelized by OpenMP. *BFW* consumed from 11.2 to 57.5 sec of CPU, meanwhile *HBA* consumed from 10.9 to 43.6 sec. The speedup of *HBA* over *BFW* is from 0.4% to 71.2%.

We explain the reduction of speedup at increasing M by the rapid growth of the number of peripheral blocks and by the insignificant speedup of $P3$ over *BCA*, and by the fact that $P3$ loses $D0$, $C1$ and $C2$ significantly.

Table 2

Execution time and comparison of multiple-threaded *HBA* and *BFW* on graph 4800

Block-matrix	Block count	*BFW*, sec	*HBA*, sec	Speedup, %
2×2	4	56.01	32.73	71.2
3×3	9	57.50	38.27	50.2
4×4	16	50.37	35.64	41.3
5×5	25	56.19	43.64	28.8
6×6	36	31.89	31.76	0.4
8×8	64	11.21	10.94	2.4

Indeed, the fraction of $P3$ blocks in the total block count is evaluated as $(M-1)^2 / M^2$ and is 0.25, 0.44, 0.56, 0.64, 0.69, 0.77 and 0.94 for M = 2, 3, 4, 5, 6, 8 and 32 respectively. Therefore, *HBA* is more effective for non-large values of M.

Conclusion. The proposed advanced heterogeneous block-parallel all pairs shortest path algorithm no matter single- or multiple-threaded considers the features of four types of blocks and overcomes the homogeneous blocked Floyd-Warshall algorithm by up to 60–70% regarding the reduction of execution time on multi-core systems. The $D0$ algorithm of calculating the diagonal block is a promising alternative to the classic Floyd-Warshall algorithm with respect to the throughput since it has the property of spatial and temporal references locality and can handle efficiently the processor's hierarchical memory.

References

1. 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.
2. Glabowski M., Musznicki B., Nowak P. and Zwierzykowski P. Review and Performance Analysis of Shortest Path Problem Solving Algorithms. *International Journal on Advances in Software*, 2014, vol. 7, no. 1&2, pp. 20–30.
3. Floyd R.W. Algorithm 97: Shortest path. *Communications of the ACM*, 1962, no. 5 (6), p. 345.
4. Prihozhy A.A., Mattavelli M., Mlynek D. Data dependences critical path evaluation at C/C++ system level description. *International Workshop PATMOS'2003*, Springer, 2003, pp. 569–579.
5. 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)*. Los Angeles, CA, July 1–2, 2013, pp. 109–112.
6. Prihozhy A. A. *Analiz, preobrazovaniye i optimizatsiya dlya vysokoproizvoditel'nykh parallel'nykh vychisleniy* [Analysis, transformation and optimization for high performance parallel computing]. Minsk, BNTU Publ., 2019. 229 p. (In Russian).
7. Prihozhy A. A., Casale-Brunet S., Bezati E., Mattavelli. M. Pipeline Synthesis and Optimization from Branched Feedback Dataflow Programs. *Journal of Signal Processing Systems*, Springer Nature, 2020, vol. 92, pp. 1091–1099. Available at: https://doi.org/10.1007/s11265-020-01568-5 (accessed 29.09.2022).
8. Prihozhy A. A., Karasik O. N. Inference of shortest path algorithms with spatial and temporal locality for big data processing. *Big Data and Advanced Analytics: sbornik nauchnykh statey VIII Mezhdunarodnoy nauchno-prakticheskoy konferentsii* [Big Data and Advanced Analytics: Proceedings of VIII International Conference]. Minsk, Bestprint Publ., 2022, pp. 56–66.
9. 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. Available at: https://doi.org/10.21122/2309-4923-2019-4-10-18 (accessed 29.01.2023). (In Russian).
10. Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm. *Journal of Experimental Algorithmics* (*JEA*), 2003, vol. 8, pp. 857–874.
11. Park J. S., Penner M., and Prasanna V. K. Optimizing graph algorithms for improved cache performance. *IEEE Trans. on Parallel and Distributed Systems*, 2004, no. 15 (9), pp.769–782.
12. Prihozhy A. A., Karasik O. N. Heterogeneous blocked all-pairs shortest paths algorithm. *System analysis and Applied Information Science*, 2017, no. 3, pp. 68–75. Available at: https://doi.org/10.21122/2309-4923-2017-3-68-75 (accessed 29.01.2023) (In Russian).
13. Karasik O. N., Prihozhy A. A. Threaded block-parallel algorithm for finding the shortest paths on graph. *Doklady BGUIR* [Reports BSUIR], 2018, no. 2, pp. 77–84 (In Russian).
14. Karasik O. N., Prihozhy A. A. Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation. *Systemnyy analiz i prikladnaya informatika* [System Analysis and Applied Information Science], 2022, no. 3, pp. 57–65 (In Russian).
15. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms. *Systemnyy analiz i prikladnaya informatika* [System Analysis and Applied Information Science], 2021, no. 3, pp. 40–50 (In Russian).

Список литературы

1. Survey of Shortest-Path Algorithms / Madkour A. [et al]. ArXiv: 1705.02044v1 [cs. DS] 4 May 2017, 26 p.

2. Review and Performance Analysis of Shortest Path Problem Solving Algorithms / Glabowski M. [et al] // International Journal on Advances in Software. 2014. Vol. 7, no. 1&2. P. 20–30.

3. Floyd R.W. Algorithm 97: Shortest path // Communications of the ACM. 1962. No. 5 (6). P.345.

4. Prihozhy A. A., Mattavelli M., Mlynek D. Data dependences critical path evaluation at C/C++ system level description // International Workshop PATMOS'2003. Springer, 2003. P. 569–579.

5. 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). Los Angeles, CA, July 1–2. 2013. P. 109–112.

6. Прихожий А. А. Анализ, преобразование и оптимизация для высокопроизводительных параллельных вычислений. Минск: БНТУ, 2019. 229 с.

7. Prihozhy A. A., Casale-Brunet S., Bezati E., Mattavelli. M. Pipeline Synthesis and Optimization from Branched Feedback Dataflow Programs // Journal of Signal Processing Systems, Springer Nature, 2020. Vol. 92. P. 1091–1099. URL: https://doi.org/10.1007/s11265-020-01568-5 (accessed 29.09.2022).

8. Прихожий А. А., Карасик О. Н. Вывод алгоритмов поиска кратчайших путей с временной и пространственной локальностью для обработки больших данных. Big Data and Advanced Analytics: сб. науч. ст. VIII Междунар. науч.-практ. конф., Минск, 11–12 мая 2022 года. Минск: Бестпринт, 2022. С. 56–66.

9. Прихожий А. А. Моделирование кэш прямого отображения и ассоциативных кэш на алгоритмах поиска кратчайших путей в графе // Системный анализ и прикладная информатика. 2019. № 4. С. 10–18. URL: https://doi.org/10.21122/2309-4923-2019-4-10-18 (дата обращения: 29.01.2023).

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

11. Park J. S., Penner M., and Prasanna V. K. Optimizing graph algorithms for improved cache performance // IEEE Trans. on Parallel and Distributed Systems. 2004. No. 15 (9). P. 769–782.

12. Прихожий А. А., Карасик О. Н. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа // Системный анализ и прикладная информатика. 2017. № 3. С. 68–75. URL: https://doi.org/10.21122/2309-4923-2017-3-68-75 (дата обращения: 29.01.2023).

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

14. Прихожий А. А., Карасик О. Н. Настройка блочно-параллельного алгоритма поиска кратких путей на эффективную многоядерную реализацию // Системный анализ и прикладная информатика. 2022. №. 3. С. 57–65.

15. Прихожий А. А. Оптимизация размещения данных в иерархической памяти для блочных алгоритмов поиска кратчайших путей // Системный анализ и прикладная информатика. 2021. №. 3. С. 40–50.

Information about the authors

Prihozhy Anatoly Alekseevich – DSc (Engineering), Professor, Professor, Department of Computer and System Software. Belarusian National Technical University (65, Nezavisimosti ave., 220013, Minsk, Republic of Belarus). E-mail: prihozhy@yahoo.com

Karasik Oleg Nikolayevich – PhD (Engineering), Lead Engineer, ISsoft Solutions (5, Chapaeva str., 220034, Minsk, Republic of Belarus). E-mail: karasik.oleg.nikolaevich@gmail.com

Информация об авторах

Прихожий Анатолий Алексеевич – доктор технических наук, профессор, профессор кафедры программного обеспечения информационных систем и технологий. Белорусский национальный технический университет (220013, г. Минск, пр. Независимости 65, Республика Беларусь). E-mail: prihozhy@yahoo.com

Карасик Олег Николаевич – кандидат технических наук, ведущий инженер. Иностранное производственное унитарное предприятие «Иссофт Солюшенз». (220034, г. Минск, ул. Чапаева, 5, Республика Беларусь). E-mail: karasik.oleg.nikolaevich@gmail.com

Поступила после доработки 15.11.2022