УДК 004
DOI: 10.21122/2309-4923-2023-4-4-13

PRIHOZHY A.A., KARASIK O.N.

NEW BLOCKED ALL-PAIRS SHORTEST PATHS ALGORITHMS OPERATING ON BLOCKS OF UNEQUAL SIZES

Belarusian National Technical University
Minsk, Republic of Belarus

In real-world networks, many problems imply finding the All-Pairs Shortest Paths (APSP) and their distances in a graph. Solving the large-scale APSP problem on modern muti-processor (multi-core) systems is the key for various application domains. The computational cost of solving the problem is high, therefore in many cases approximate solutions are considered as acceptable. The blocked APSP algorithms are a promising approach which can exploit many processors (cores) and their caches in parallel mode efficiently. At the same time, to our best knowledge, all blocked algorithms of the Floyd-Warshall family use blocks of equal sizes. This property limits application of the algorithms. In this paper we propose new blocked algorithms which divide the input graph into unequal subgraphs and divide the matrix of distances between pairs of vertices into blocks of unequal sizes. The algorithms describe the dense subgraphs by the adjacency matrix and describe sparse subgraphs and connections between them by the adjacency list. This approach allows the Floyd-Warshall family algorithms to be used together with Dijkstra family algorithms. It can be applied to large graphs decomposed into dense (clusters) and sparse subgraphs. A new heterogeneous algorithm can significantly reduce the computation time of blocks depending on the block type and size. The contribution of the paper is the development of a new family of blocked APSP algorithms which can handle blocks of unequal sizes, save and extend the advantages of the state-of-the-art algorithms operating on blocks of equal sizes. The proposed algorithms are implemented as single- and multiple-threaded parallel applications for multi-core systems.

Keywords: *APSP problem, blocked algorithm, unequal sizes of blocks, heterogeneous algorithm, multi-core pro-cessor, muti-threaded implementation*

Introduction

The problem of finding shortest and longest paths between vertices of a large, weighted graph [1–6] has many applications, such as Internet route planners, traffic road networks, traffic simulations in computer networks, car/robot navigation systems, courier-scheduling optimization, biological information mining, web searching, social networks, etc. The interest in this problem has significantly increased recently due to the emergence of het-erogeneous parallel computing systems combining the classical and increasingly powerful CPUs with modern powerful hardware acceler-ators [7–9]. The computational complexity of shortest paths algorithms depends on the graph type [3, 4]: directed or nondirected, weighted or not weighted, dense or sparse, what is the edge weight (integer, real, positive, negative, etc.). There are different formulations of the shortest path problem: between two vertices; between the source (sink) and each other vertex (single source and single sink – SSSP); between each pair of vertices (all pairs shortest paths – APSP); all vertices must be in the path or not. For each formulation, a set of competitive algorithms has been developed.

The focus of this paper is on the all-pairs shortest paths problem (APSP) and on the blocked algorithms [10, 11] of solving the problem. The state-of-the-art APSP-algorithms decompose the dense graph into equally sized subgraphs and decompose the path distance matrix into blocks of the same size. The key contribution of the paper is the extension of the algorithms which leads to the use of blocks of unequall sizes. The corollary of the extension is the emerging possibility of modifying the APSP-algorithms which handle efficiently the dense graphs to solve the shortest paths problem on sparse graphs.

All-pairs shortest paths algorithms

Let $G = (V, E)$ be a simple directed graph with real edge-weights consisting of a set V, $|V| = N$, of vertices numbered 1 through N and a set E of edges. Let W be a cost adjacency matrix for G. So, $w(i, i) = 0$, $1 \le i \le N$; $w(i, j)$ is the cost (weight) of edge (i, j) if $(i, j) \in E$ and $w(i, j) = \infty$ if $i \ne j$ and $(i, j) \notin E$.

Let d_{ij} be a length of a shortest path from vertex i to vertex j, and D be a matrix of distances between all pairs of vertices $i, j \in V$, $i \ne j$. Let P be a matrix whose element pij is a vertex that is previous for vertex j in a path from i to j. The objective of an APSP-algorithm is to compute the D and P matrices for a given graph G. Two main families of the algorithms exist to solve the APSP problem: 1) based on the Dijkstra SSSP-algorithm [1]; 2) based on the Floyd-Warshall APSP-algorithm [2].

The first family includes the Dijkstra algorithm [1], the Bellman-Ford algorithm [12], the Johnson algorithm [13], the Harish and Narayanan algorithm [14], and others [15]. The time complexity of the Dijkstra algorithm that targets directed graphs with positive edge-weights is $O(|V|\cdot lg(|V|)+|E|)$ for SSSP. On graphs of this type, the application of the Dijkstra algorithm to each graph vertex solves the APSP problem with the time complexity of $O(|V|^2\cdot lg(|V|+|V|\cdot|E|)$. The time complexity of the Bellman-Ford algorithm for a graph with positive/negative edge-weights is $O(|V|^2+|V|\cdot|E|)$ for SSSP. It is higher than the Dijkstra algorithm's time complexity. The Johnson algorithm uses the Dijkstra and Bellman-Ford algorithms as subroutines and solves the APSP problem with negative edge-weights in $O(|V|^2\cdot lg(|V|+|V|\cdot|E|)$ time. On sparse graphs, the Johnson algorithm outper-forms the APSP algorithms from the Floyd-Warshall family. The Harish and Narayanan algorithm is a parallelized version combining the characteristics of the Dijkstra and Bellman-Ford algorithms. It was developed for the implementation on GPUs.

The second family includes among others the Floyd-Warshall (FW) algorithm [2], the blocked Floyd-Warshall algorithm (BFW) proposed in [6, 10, 11] by Katz, Venkataraman and others, the graph extension-based algorithm (GEA) and the heterogeneous blocked APSP algorithm (HBAPSP) both proposed by Prihozhy and Karasik in [16–18].

The *FW* algorithm is described with three nested loops. It performs a relaxation (min, +) operation on elements of matrix D. Its time complexity is $\Theta(|V|^3)$ no matter how many positive/negative edges the graph contains. The algorithm is simple in the organization of computations. This property is an advantage of the al-gorithm. The algorithm tries to recalculate all elements of matrix D in every iteration of the most outer loop. This property is a drawback of the algorithm. The algorithm is parallelised by OpenMP [19].

The *GEA* algorithm calculates the shortest paths while stepwise adding vertices to graph G. Therefore, the shortest path lengths (real positive/negative numbers) are represented by a sequence of matrices $D[1\times1]$, …, $D[|V|\times|V|]$. The size of D is increased by 1) adding and computing a new row and column and 2) recomputing previous elements of D. The resynchronization of these operations was carried out by formal methods and allowed to reduce the number of iterations in loops and to improve the spatial and temporal data references locality in GEA. As a result, GEA reduces the cache pressure in multi-core processors and speeds up the search of shortest paths.

The blocked *BFW* algorithm solves two problems: 1) localizes the data accesses within blocks (tiles) and to increase the efficiency of hierarchical memory operation; 2) parallelizes computations at the block level. *BFW* divides the graph into subgraphs of equal sizes and splits the matrix of shortest paths distances into equally sized square blocks (tiles), creating a uniformly blocked matrix of the $M\times M$ dimension. In each iteration of the most outer loop, a diagonal block is calculated first, blocks on the cross associated with the diagonal block are calculated second (possibly in parallel), and all other peripheral blocks are calculated third (possibly in parallel). Eeach block is recalculated M times using the *FW* algorithm. *BFW* is easily parallelised by OpenMP in fork-join style. It balances the workload in symmetric multiprocessing sharedmemory systems.

Unlike *BFW*, the *HBAPSP* algorithm does not use *FW* for recalculating each block. It distinguishes the blocks of four types: diagonal, vertical of cross, horizontal of cross, and peripheral. It provides a separate unique block calculation algorithm of higher performance for each block type. The four algorithms account for the features of the corresponding block types. They reduce the number of iterations in nested loops, exploit the references locality of data in CPU caches, and speedup the computations. The diagonal blocks are calculated by *GEA*. OpenMP parallelizes *HBAPSP* at task level in fork-join style.

The basic ideas of *BFW* were fruitfully used in several works, which contribute in solving the shortest paths problem:

1. A recursive blocked *FW* algorithm [10].
2. Efficient usage of GPUs [7–9].
3. Solving sparse graph scaling problem [20].
4. Optimization of data allocation in hier-archical memory [21].
5. Improving cache performance for APSP [11, 17, 22].
6. A cooperative threaded algorithm [23, 24].
7. Selection of optimal block-size [25].
8. Reducing energy consumption [26].
9. Shortest paths search dataflow networks of actors for multicore implementation [27].

The state-of-the-art blocked shortest paths algorithms cannot handle blocks of unequal sizes, therefore, cannot decompose graphs into unequally sized subgraphs do not match the heterogeneous computing systems, etc.

Decomposition of matrix of paths lengths into blocks of unequal sizes

In the paper, we propose to decompose the graph G into subgraphs and decompose the matrix B into blocks of unequal sizes defined by vector $S = (S_1 \ldots S_M)$ (Figure 1). While M blocks are square on the principal diagonal of B (block B_{ii} has the $V_i\times V_i$ size), all other blocks are rectangular in general case (block Bij has the $V_i\times V_j$ size for $i, j = 1\ldots M$, $i \neq j$). All blocks in row i have the height of V_i, and all blocks in column j have the width of V_j. Matrix P of previous vertices in the shortest paths has the same structure.

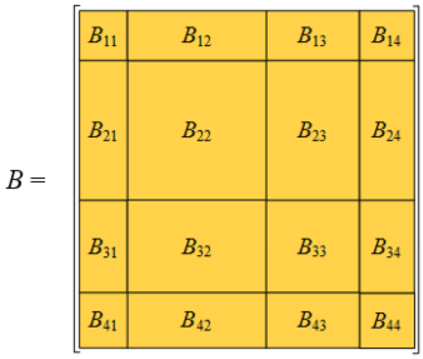


Figure 1. Decomposition of matrix of shortest paths lengths into matrix of blocks of unequal sizes

There are several advantages to such an approach. It extends the classic blocked shortest paths search algorithms to a more general case. The decomposition of an input graph into subgraphs can be derived from the natural origin structure of the graph. From the computational point of view, a large graph can be decomposed into subgraphs that have different properties (for instance, dense or sparse), which allows to choose the most appropriate computational algorithm for each subgraph. It also allows to select the most appropriate block size (from the hardware perspective) to be used for majority of the blocks even if graph isn't evenly divided by it. Unequally sized blocks of both matrices B and P can be assigned to processors of different capabilities, which enables the speedup on the heterogeneous multiprocessor system while solving the shortest paths problem.

Extension of blocked Floyd-Warshall algorithm to unequal block-sizes

Assuming unequal sizes of blocks, we extend the known blocked Floyd-Warshall homogeneous algorithm *BFW* to an allpairs shortest path algorithm *BFWUS*, which can handle a block-matrix B of unequally sized blocks. Algorithm 1 describes the *BFWUS*. In a loop along m it recalculates each of M^2 blocks of matrix B, therefore, it performs M^3 recalculations of blocks in total.

Algorithm 1: Extension of blocked Floyd-Warshall algorithm accounting for blocks of unequal sizes (*BFWUS*)

Input: A number N of input graph vertices
Input: A matrix $W[N{\times}N]$ of graph edge weights
Input: A number M of blocks
Input: A vector $S = (S_1 \ldots S_M)$ of sizes of vertex subsets
Output: A blocked matrix $B[M{\times}M]$ of path distances
Output: A blocked matrix $P[M{\times}M]$ of previous vertices in shortest paths

for $i, j \leftarrow 1$ **to** N **do**
  if $W(i, j) \neq \infty$ **then**
    $P^{\text{init}}(i, j) \leftarrow i$
  else
    $P^{\text{init}}(i, j) \leftarrow undefined$
$B[M \times M] \leftarrow W[N \times N]$ $P[M \times M] \leftarrow P^{\text{init}}[N \times N]$
for $m \leftarrow 1$ **to** M **do**
  $BCUS(S, B, P, m, m, m)$ // $D0$
  for $v \leftarrow 1$ **to** M **do**
    if $v \neq m$ **then**
      $BCUS(S, B, P, v, m, m)$ // $C1$
      $BCUS(S, B, P, m, m, v)$ // $C2$
  for $v \leftarrow 1$ **to** M **do**
    if $v \neq m$ **then**
  for $u \leftarrow 1$ **to** M **do**
    if $u \neq m$ **then**
      $BCUS(S, B, P, v, m, u)$ // $P3$
return B, P

Algorithm 2 describes a block-calculation algorithm *BCUS* with the feature of processing blocks of unequal sizes. The algorithm's inputs are three blocks $B_{v,u}$, $B_{v,m}$ and $B_{m,u}$ of which two or three can be identical. The sizes of blocks are $S_v{\times}S_u$, $S_v{\times}S_m$ and $S_m{\times}S_u$ respectively. BCUS consists of three nested loops. It makes $S_v{\times}S_m{\times}S_u$ attempts to update the values of elements of block $B_{v,u}$ no matter the three blocks are dense or sparse. The order of loops is essential. The loop along k must be the outer, it cannot be reordered with other loops.

Algorithm 2: Calculation of blocks of unequal sizes (*BCUS*)

Input: A vector S of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ of path distances
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Input: Indices v, m, u of vertex subsets
Output: Recalculated matrix B regarding block $B_{v,u}$
Output: Recalculated matrix B regarding block $P_{v,u}$

for $k \leftarrow 1$ **to** S_m **do**
  for $i \leftarrow 1$ **to** S_v **do**
    for $j \leftarrow 1$ **to** S_u **do**
      $sum \leftarrow B_{v,m}(i, k) + B_{m,u}(k, j)$
      if $B_{v,u}(i, j) > sum$ **then**
        $B_{v,u}(i, j) \leftarrow sum$
        $P_{v,u}(i, j) \leftarrow Pm,u(k, j)$
return B, P

There are four calls of *BCUS* in *BFWUS* (Algorithm 1), which correspond to four types of blocks: $D0$ (diagonal), $C1$ (vertical in cross), $C2$ (horizontal in cross) and $P3$ (peripheral). The calls differ each other by the actual parameters, of which three first describe the vector of sizes and the matrices of blocks, and three others select the blocks.

Figure 2 depicts the process of stepwise recalculation of unequally sized blocks of the modified matrix B in algorithm *BFWUS*. For matrix $B[4{\times}4]$, the process consists of four steps. At step 1, block B_{11} is diagonal $D0$. It is calculated first. Then, blocks B_{21}, B_{31}, B_{41} of type $C1$ and blocks B_{12}, B_{13}, B_{14} of type $C2$ are calculated through B_{11} possibly in parallel. After that, all other blocks of type $P3$ are calculated possibly in parallel through blocks of types $C1$ and $C2$: B_{ij} is calculated through B_{i1} and B_{1j} at $i, j = 2, 3, 4$. At steps 2, 3 and 4 blocks B_{22}, B_{33} and B_{44} become diagonal of type $D0$, and the calculation procedure repeats in the same manner.

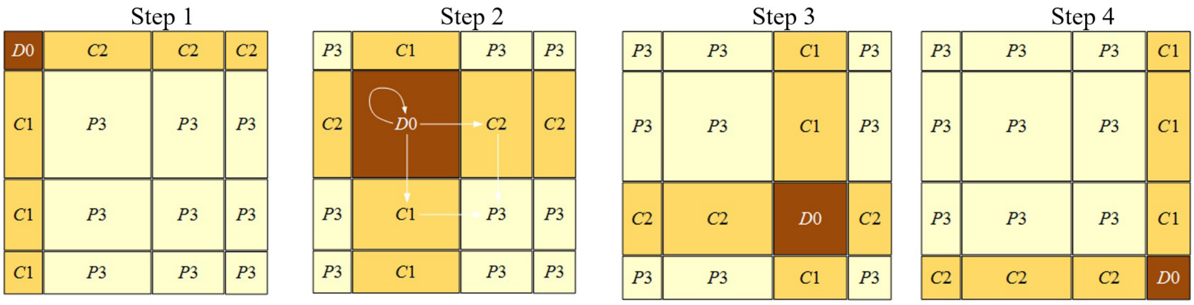


Figure 2. Illustration of *BFWUS* operation: cross moves from top-left to bottom-right corner of blocked matrix B; firstly, block $D0$ is calculated through itself; secondly, blocks $C1$ and $C2$ are calculated through $D0$; thirdly, blocks $P3$ are calculated through $C1$ and $C2$; wite arrows in step 2 show data dependence between blocks

Generalization of heterogeneous blocked all-pairs shortest paths algorithm

We extend the known blocked heterogene-ous algorithm *HBAPSP* [16–18] to a blocked heterogeneous APSP algorithm *HBAPSPUS*, which can handle unequally sized blocks. *HBAPSP* was proposed as a means of consider-ing the features of the four types of blocks at the aim of speeding up their computation. *HBAPSPUS* described by Algorithm 3 allows the blocks to have unequal sizes, which further extends the property of performing computations heterogeneously and extends the nonuniformity of the processor's cores load. Unlike *BFWUS* including four calls of the same algorithm *BCUS* with six input parameters, *HBAPSPUS* calls four different block calculation algorithms: *D0US* with two parameters, *C1US* with four parameters, *C2US* with four parameters and *BCUS* with six parameters. The calls of different algorithms course different computational load. The computational complexity of Algorithm 3 is equal to the computational com-plexity of Algorithm 1. Moreover, the algorithms have the same parallelization potential. The algorithms yield the same values of matrix B, although they can yield different values of matrix P. The reason is different shortest paths with the same length may exist between two vertices.

Algorithm 4 (*D0US*) generalizes the diagonal block calculation algorithm proposed in [16–18]. In *D0US*, $b_{i,j}$ and $p_{i,j}$ are elements of blocks $B_{m,m}$ and $P_{m,m}$. The algorithm calculates the diagonal square blocks $B_{m,m}$ and $P_{m,m}$. of size $[S_m \times S_m]$ through themself without involving other blocks. Therefore, it consumes fewer amount of data against algorithms calculating blocks of other types.

Like in *BCU*S, the main execution part of *D0US* includes three nested loops, but the loops along i and j have an updated iteration scheme producing a smaller number of iterations. *D0US* starts operation from a part of $B_{m,m}$ ($P_{m,m}$) having the size $[1\times1]$ and step-by-step increases the size of the part. The loops consequently process growing matrices of size $[1\times1]$, $[2\times2]$ … $[S_m \times S_m]$, which support the *D0US*'s property of temporal locality. *D0US* has up to three times a smaller number of executions of the body of the most nested loop than *BCUS* has and reduces the cache pressure.

Algorithm 3: Heterogeneous blocked shortest paths search upon unequal block-sizes (*HBAPSPUS*)

Input: A number N of graph vertices
Input: A matrix $W[N\times N]$ of graph edge weights
Input: A number M of blocks
Input: Vector $S = (S_1 \dots S_M)$ of sizes of vertex subsets
Output: A blocked matrix $B[M\times M]$ of path distances
Output: A blocked matrix $P[M\times M]$ of previous vertices in shortest paths

```
B[M×M] ← W[N×N]
for m ← 1 to M do
    D0US(S, B, P, m)                      // D0
    for v ← 1 to M do
        if v ≠ m then
            C1US(S, B, P, v, m)           // C1
            C2US(S, B, P, m, v)           // C2
    for v ← 1 to M do
        if v ≠ m then
            for u ← 1 to M do
                if u ≠ m then
                    BCUS(S, B, P, v, m, u)   // P3
return B, P
```

Algorithm 5 (*C1US*) calculates $C1$-type vertical rectangular blocks $B_{v,m}$ and $P_{v,m}$ of size $[S_v \times S_m]$ of cross through themself and blocks $B_{m,m}$ and $P_{m,m}$ of size $[S_m \times S_m]$ without involving third blocks. It generalizes the similar-purpose algorithm proposed in [18]. In *C1US*, $b1_{i,j}$ and $b3_{k,j}$ ($p1_{i,j}$ and $p3_{k,j}$) are elements of blocks $B_{v,m}$ and $B_{m,m}$ ($P_{v,m}$ and $P_{m,m}$) respectively. The loop along j of *C1US* has the iteration scheme that produces $k-1$ iteration, which is smaller than the number of corresponding iterations in *BCA*. The two nests of loops consequently process matrices of growing size $[S_v \times 1]$, $[S_v \times 2]$ … $[S_v \times S_m]$. It means that *C*1*US* has the property of temporal references locality, decreases the number of accesses to memory and reduces the cache pressure in comparison to *BCUS*. Moreover, the overall number of iterations of the most nested loop of *C*1*US* is twice smaller than in *BCA*.

Algorithm 4: Calculation of diagonal blocks of unequalsizes (*D*0*US*)

Input: A vector S of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ of path distances
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Input: Index m of vertex subsets
Output: Matrix B recalculated regarding block $B_{m,m}$
Output: Matrix P recalculated regarding block $P_{m,m}$

```
for k ← 2 to S_m do
  k1 ← k – 1
  for i ← 1 to k1 do
    for j ← 1 to k1 do
      a_2 ← b_{i,k1} + b_{k1,j}
      if b_{i,j} > a_2 then
        b_{i,j} ← a_2    p_{i,j} ← p_{k1,j}
      a_0 ← b_{i,j} + b_{j,k}
      if b_{i,k} > a_0 then
        b_{i,k} ← a_0    p_{i,k} ← p_{j,k}
    a_1 ← b_{k,i} + b_{i,j}
    if b_{k,j} > a_1 then
      b_{k,j} ← a_1    p_{k,j} ← p_{i,j}
k1 ← S_m
for i ← 1 to k1 – 1 do
  for j ← 1 to k1 – 1 do
    a_2 ← b_{i,k1} + b_{k1,j}
    if b_{i,j} > a_2 then
      b_{i,j} ← a_2    p_{i,j} ← p_{k1,j}
return B, P
```

Algorithm 6 (*C*2*US*) calculates a *C*2-type horizontal rectangular cross blocks $B_{m,v}$ and $P_{m,v}$ of size $[S_m \times S_v]$ through blocks $B_{m,m}$ and $P_{m,m}$ of size $[S_m \times S_m]$ and themself without involving third blocks. It generalizes the similar-purpose algorithm proposed in [18]. The loop along i of *C*2*US* has the iteration scheme that produces $k-1$ iteration. It is less than the number of corresponding iterations in *BCA*. All the loops consequently process matrices of growing size $[1 \times S_v]$, $[2 \times S_v]$ … $[S_m \times S_v]$. Like *D*0*US* and *C*1*US*, algorithm *C*2*US* has the property of temporal references locality, decreases the number of accesses to memory and reduces the cache pressure in comparison to *BCUS*. The overall number of iterations of the most nested loop of *C*2*US* is twice smaller than in *BCA*.

HBAPSPUS calls the universal block calculation algorithm *BCUS* to compute all peripheral *P*3 blocks. Since the blocks of *D*0, *C*1 and *C*2 types are calculated by other algorithms, the three nested loops along *k*, *i* and *j* can be reordered arbitrarily in *BCUS*.

Algorithm 5: Calculating vertical block of cross upon unequal block-sizes (*C*1*US*)

Input: A vector S of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ of path distances
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Input: Indices v and m of vertex subsets
Output: Matrix B recalculated regarding block $B_{v,m}$
Output: Matrix P recalculated regarding block $P_{v,m}$

```
B1 ← B_{v,m}   B3 ← B_{m,m}   P1 ← P_{v,m}   P3 ← P_{m,m}
for k ← 1 to S_m do
  k1 ← k–1;
  for i ← 1 to S_v do
    for j ← 1 to k 1 do
      a_2 ← b1_{i,k1} + b3_{k1,j}
      if b1_{i,j} > a_2 then
        b1_{i,j} ← a_2    p1_{i,j} ← p3_{k1,j}
      a_0 ← b1_{i,j} + b3_{j,k}
      if b1_{i,k} > a_0 then
        b1_{i,k} ← a_0    p1_{i,k} ← p3_{j,k}
k1 ← S_m
for i ← 1 to S_v do
  for j ← 1 to k1 ← 1 do
    a_2 ← b1_{i,k1} + b3_{k1,j}
    if b1_{i,j} > a_2 then
      b1_{i,j} ← a_2    p1_{i,j} ← p3_{k1,j}
return B, P
```

Algorithm 6: Calculating horizontal block of cross upon unequal block-sizes (*C*2*US*)

Input: A vector S of sizes of graph vertex subsets
Input: A blocked matrix $B[M \times M]$ of path distances
Input: A blocked matrix $P[M \times M]$ of previous vertices in shortest paths
Input: Indices m and v of vertex subsets
Output: Matrix B recalculated regarding block $B_{m,v}$
Output: Matrix P recalculated regarding block $P_{m,v}$

```
B1 ← B_{m,v}   B2 ← B_{m,m}   P1 ← P_{m,v}   P2 ← P_{m,m}
for k ← 1 to S_m – 1 do
  k1 ← k–1;
  for i ← 1 to k1 do
    for j ← 1 to S_v do
      a_2 ← b2_{i,k1} + b1_{k1,j}
      if b1_{i,j} > a_2 then
        b1_{i,j} ← a_2    p1_{i,j} ← p1_{k1,j}
      a_0 ← b2_{k,i} + b1_{i,j}
      if b1_{k,j} > a_0 then
        b1_{k,j} ← a_0    p1_{k,j} ← p1_{i,j}
k1 ← S_m
for i ← 1 to k1 ← 1 do
  for j ← 1 to S_v do
    a_2 ← b2_{i,k1} + b1_{k1,j}
      if b1_{i,j} > a_2 then
        b1_{i,j} ← a_2    p1_{i,j} ← p1_{k1,j}
return B, P
```

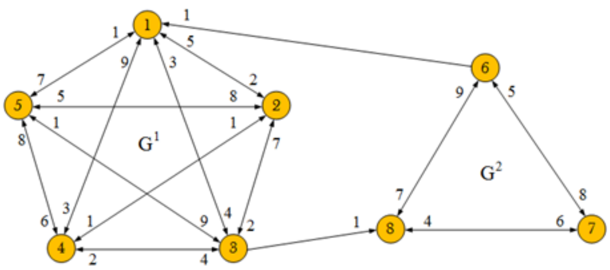
Searching shortest paths in graphs consisting of weakly connected dense subgraphs

The all-pairs shortest paths search can be significantly speeded up if the large graph is decomposed into a set of dense weakly connected subgraphs. In this case, we can compute the shortest paths in dense subgraphs by using the APSP algorithms on adjacency matrix (such as *FW*, *GEA* and others), and can compute the shortest paths between the dense subgraphs by using the single source and single sink algorithms on adjacency lists (such that Dijkstra, Harish-Narayanan and others). With this idea in mind, we can extend *HBAPSPUS* by considering sparse alternatives of the block calculation algorithms *C1US*, *C2US* and *BCUS*.

Let consider an example weighted graph G depicted in Figure 3. The graph contains two complete directed subgraphs G^1 and G^2 constructed on two subsets {1, 2, 3, 4, 5} and {6, 7, 8} of vertices respectively.

Just arc (3, 8) connects the first subgraph to the second one. Just arc (6, 1) connects the second subgraph to the first one. Matrix $B[2\times 2]$ is constructed of four blocks and is initialized using the rows and columns of W: B_{11} is the crossing of rows 1–5 and columns 1–5, B_{22} is the crossing of rows 6–8 and columns 6–8, B_{12} is the crossing of rows 1–5 and columns 6–8, and B_{21} is the crossing of rows 6–8 and columns 1–5. Blocks B_{12} and B_{21} are sparse since their elements are mostly initialized to ∞ (infinity).

HBAPSPUS updates each block twice. It performs eight (min, +) matrix operations (denoted ⊗) on four blocks:



a

$$W = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array}\begin{array}{c} \begin{array}{cccccccc} 1&2&3&4&5&6&7&8 \end{array} \\ \left[\begin{array}{ccccc|ccc} 0&2&4&3&7&\infty&\infty&\infty\\ 5&0&2&1&5&\infty&\infty&\infty\\ 3&7&0&2&1&\infty&\infty&1\\ 9&1&4&0&8&\infty&\infty&\infty\\ 1&8&9&6&0&\infty&\infty&\infty\\ \hline 1&\infty&\infty&\infty&\infty&0&8&7\\ \infty&\infty&\infty&\infty&\infty&5&0&4\\ \infty&\infty&\infty&\infty&\infty&9&6&0 \end{array}\right] \end{array}$$

b

Figure 3. Example directed graph G consisting of two weakly connected complete subgraphs G^1 and G^2:
a) graphical view of G; *b*) matrix W of graph edge weights decomposed into four blocks

Algorithm *D0US* performs operations 1 and 5, *C1US* performs operations 2 and 6, *C2US* performs operations 3 and 7, and *BCUS* performs operations 4 and 8. Figure 4 shows matrices B and P which are a result of executing *HBAPSPUS*. Since subgraphs G^1 and G^2 are connected by single arc in each direction, rows 6, 7 and 8 of block B_{21} are heavily dependent (differ by a constant). Columns 6, 7 and 8 of block B_{12} are heavily dependent too (Figure 4*a*). Moreover, rows of peripheral block P_{21} are identical; columns of peripheral block P_{12} are also identical (Figure 4*b*). To reduce the algorithms runtime and memory consumption, we revise the block calculation algorithms *C1US*, *C2US* and *BCUS* and improve them with respect to the sparseness of blocks of types *C*1, *C*2 and *P*3.

$$B = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array}\begin{array}{c} \begin{array}{cccccccc} 1&2&3&4&5&6&7&8 \end{array} \\ \left[\begin{array}{ccccc|ccc} 0&2&4&3&5&14&11&5\\ 4&0&2&1&3&12&9&3\\ 2&3&0&2&1&10&7&1\\ 5&1&3&0&4&13&10&4\\ 1&3&5&4&0&15&12&6\\ \hline 1&3&5&4&6&0&8&6\\ 6&8&10&9&11&5&0&4\\ 10&12&14&13&15&9&6&0 \end{array}\right] \end{array}$$

a

$$P = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array}\begin{array}{c} \begin{array}{cccccccc} 1&2&3&4&5&6&7&8 \end{array} \\ \left[\begin{array}{ccccc|ccc} 0&0&0&0&2&7&7&2\\ 4&1&1&1&2&7&7&2\\ 4&3&2&2&2&7&7&2\\ 4&3&1&3&2&7&7&2\\ 4&0&0&0&4&7&7&2\\ \hline 5&0&0&0&2&5&5&2\\ 5&0&0&0&2&6&6&6\\ 5&0&0&0&2&7&7&7 \end{array}\right] \end{array}$$

b

Figure 4. Matrices of blocks: *a*) matrix B of shortest paths lengths in example graph G; *b*) matrix P of previous vertices: p_{ij} is a predecessor of vertex j in a shortest path from i to j

1. $B_{11}[5\times5] \leftarrow B_{11}[5\times5] \otimes B_{11}[5\times5]$
2. $B_{21}[3\times5] \leftarrow B_{21}[3\times5] \otimes B_{11}[5\times5]$
3. $B_{12}[5\times3] \leftarrow B_{11}[5\times5] \otimes B_{12}[5\times3]$
4. $B_{22}[3\times3] \leftarrow B_{21}[3\times5] \otimes B_{12}[5\times3]$
5. $B_{22}[3\times3] \leftarrow B_{22}[3\times3] \otimes B_{22}[3\times3]$
6. $B_{12}[5\times3] \leftarrow B_{12}[5\times3] \otimes B_{22}[3\times3]$
7. $B_{21}[3\times5] \leftarrow B_{22}[3\times3] \otimes B_{21}[3\times5]$
8. $B_{11}[5\times5] \leftarrow B_{12}[5\times3] \otimes B_{21}[3\times5]$.

Results. We have developed a software in C++ which implements on multicore processors the homogeneous blocked *BFWUS* algorithm and heterogeneous blocked *HBAPSPUS* algorithm on blocks of unequal sizes. A validation module is a part of the software which checks the correctness of blocked matrices B and P as compared with the shortest paths obtained by the Floyd-Warshall algorithm.

Conclusion

The paper has shown that the state-of-the-art blocked all-pairs shortest paths algorithms can be extended at the aim of handling the unequally sized dense and sparse subgraphs of the large, decomposed graph and calculating the unequally sized blocks of matrices representing the shortest paths and their lengths. The proposed algorithms aim at considering the nature of the large graphs and their partitioning into subgraphs as well at speeding up the computation of shortest paths between vertices on high-performance multi-processor systems.

REFERENCES

1. **Dijkstra E.W.** A note on two problems in connexion with graphs. Numerische Mathematik, 1959, vol. 1(1), pp. 269–271.

2. **Floyd R.W.** Algorithm 97: Shortest path. Communications of the ACM, 1962, no. 5(6), p. 345.

3. **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.

4. **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.

5. **Prihozhy A., Mlynek D.** Design of parallel implementations by means of abstract dynamic critical path based profiling of complex sequential algorithms. Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation: 16th International Workshop, PATMOS 2006, Montpellier, France, September 13-15, 2006, pp. 1–11.

6. **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. **doi:** 10.1007/s11265-020-01568-5

7. **Katz G.J., Kider J.T.** All-pairs shortest-paths for large graphs on the GPU. GH'08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. ACM, 2008, pp. 47–55.

8. **Ortega-Arranz H., Torres Y., Llanos D.R, and Escribano A.G.** The all-pair shortest-path problem in shared-memory heterogeneous systems. High-Performance Computing on Complex Environments, 2013, pp. 283–299.

9. **Djidjev H., Thulasidasan S., Chapuis G., Andonov R. and Lavenier D.** Efficient multi-GPU computation of all-pairs shortest paths. IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014, pp. 360–369.

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. **Bellman R.E.** On a routing problem. Quarterly of Applied Mathematics, 1958, vol. 16, no. 1, pp. 87–90.

13. **Johnson D.B.** Efficient Algorithms for Shortest Paths in Sparse Networks. J. ACM, 1977, vol. 24 no. 1, pp. 1 – 13.

14. **Harish P., Narayanan P.J.** Accelerating large graph algorithms on the GPU using CUDA. International conference on high-performance computing. Springer, 2007, pp. 197–208.

15. **Meyer U. and Sanders P.** Δ-stepping: a parallelizable shortest path algorithm. Journal of Algorithms, vol. 49, no. 1, 2003, pp. 114–152.

16. **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. (In Russian).

17. **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: proceedings of VIII international conference]. Minsk, Bestprint Publ., 2022, pp. 56–66.

18. **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

19. **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.

20. **Yang S., Liu X., Wang Y., He X., Tan G.** Fast All-Pairs Shortest Paths Algorithm in Large Sparse Graph. ICS '23: Proceedings of the 37th International Conference on Supercomputing, 2023, pp. 277–288.

21. **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.

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. **doi:** 10.21122/2309-4923-2021-3-40-50

23. **Prihozhy A.A., Karasik O.N.** Cooperative block-parallel algorithms for task execution on multi-core system. System analysis and applied information science, 2015, no. 2, pp. 10–18.

24. **Karasik O.N., Prihozhy A.A.** Threaded block-parallel algorithm for finding the shortest paths on graph. Doklady BGUIR, 2018, no. 2, pp. 77–84. (In Russian).

25. **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. **doi:** 10.21122/2309-4923-2022-3-57-65

26. **Prihozhy A.A., Karasik O.N.** Influence of shortest path algorithms on energy consumption of multi-core processors. System analysis and applied information science, 2023, no. 2, pp. 4–12. **doi:** 10.21122/2309-4923-2023-2-4-12

27. **Prihozhy A.A.** Generation of shortest path search dataflow networks of actors for parallel multicore implementation. Informatics, 2023, vol. 20, no. 2, pp. 65–84. **doi:** 10.37661/1816-0301-2023-20-2-65-84

ЛИТЕРАТУРА

1. **Dijkstra E.W.** A note on two problems in connexion with graphs. Numerische Mathematik, 1959, vol. 1(1), pp. 269–271.

2. **Floyd R.W.** Algorithm 97: Shortest path. Communications of the ACM, 1962, no. 5(6), p. 345.

3. **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.

4. **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.

5. **Prihozhy A., Mlynek D.** Design of parallel implementations by means of abstract dynamic critical path based profiling of complex sequential algorithms. Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation: 16th International Workshop, PATMOS 2006, Montpellier, France, September 13-15, 2006, pp. 1–11.

6. **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. **doi:** 10.1007/s11265-020-01568-5

7. **Katz G.J., Kider J.T.** All-pairs shortest-paths for large graphs on the GPU. GH'08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. ACM, 2008, pp. 47–55.

8. **Ortega-Arranz H., Torres Y., Llanos D.R. and Escribano A.G.** The all-pair shortest-path problem in shared-memory heterogeneous systems. High-Performance Computing on Complex Environments, 2013, pp. 283–299.

9. **Djidjev H., Thulasidasan S., Chapuis G., Andonov R. and Lavenier D.** Efficient multi-GPU computation of all-pairs shortest paths. IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014, pp. 360–369.

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. **Bellman R.E.** On a routing problem. Quarterly of Applied Mathematics, 1958, vol. 16, no. 1, pp. 87–90.

13. **Johnson D.B.** Efficient Algorithms for Shortest Paths in Sparse Networks. J. ACM, 1977, vol. 24, no. 1, pp. 1–13.

14. **Harish P., Narayanan P.J.** Accelerating large graph algorithms on the GPU using CUDA. International conference on high-performance computing. Springer, 2007, pp. 197–208.

15. **Meyer U., Sanders P.** Δ-stepping: a parallelizable shortest path algorithm. Journal of Algorithms, vol. 49, no. 1, 2003, pp. 114–152.

16. **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. – 2017. – № 3. – С. 68–75.

17. **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. – С. 56–66.

18. **Прихожий, А.А.** Усовершенствованный разнородный блочно-параллельный алгоритм поиска кратчайших путей на графе / А.А. Прихожий, О.Н. Карасик // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. – 2023. – № 1(266). – С. 77–83. **doi:** 10.52065/2520-6141-20

19. **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.

20. **Yang S., Liu X., Wang Y., He X., Tan G.** Fast All-Pairs Shortest Paths Algorithm in Large Sparse Graph. ICS '23: Proceedings of the 37th International Conference on Supercomputing, 2023, pp. 277–288.

21. **Прихожий, А.А.** Моделирование кэш прямого отображения и ассоциативных кэш на алгоритмах поиска кратчайших путей на графе / А.А. Прихожий // Системный анализ и прикладная информатика. – 2019. – № 4. – С. 10–18.

22. **Прихожий, А.А.** Оптимизация размещения данных в иерархической памяти для блочных алгоритмов поиска кратчайших путей / А.А. Прихожий // Системный анализ и прикладная информатика. – 2021. – № 3. – С. 40–50. **doi:** 10.21122/2309-4923-2021-3-40-50

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

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

25. **Карасик, О.Н.** Настройка блочно-параллельного алгоритма поиска кратких путей на эффективную много-ядерную реализацию / О.Н. Карасик, А.А. Прихожий // Системный анализ и прикладная информатика. – 2022. – № 3. – С. 57–65. **doi:** 10.21122/2309-4923-2022-3-57-65

26. **Прихожий, А.А.** Влияние алгоритмов поиска кратчайших путей на энергопотребление многоядерных про-цессоров / А.А. Прихожий, О.Н. Карасик // Системный анализ и прикладная информатика. – 2023. – № 2. – С. 4–12. **doi:** 10.21122/2309-4923-2023-2-4-12

27. **Прихожий, А.А.** Генерация потоковых сетей акторов поиска кратчайших путей для параллельной многоядерной реализации / А.А. Прихожий // Информатика. – 2023. – Т. 20. – № 2. – С. 65–84. **doi:** 10.37661/1816-0301-2023-20-2-65-84

ПРИХОЖИЙ А.А., КАРАСИК О.Н.

НОВЫЕ БЛОЧНЫЕ АЛГОРИТМЫ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ МЕЖДУ ВСЕМИ ПАРАМИ ВЕРШИН ГРАФА, РАБОТАЮЩИЕ НА БЛОКАХ НЕРАВНЫХ РАЗМЕРОВ

Белорусский национальный технический университет
г. Минск, Республика Беларусь

Многие задачи на реальных сетях предполагают поиск кратчайших путей между всеми парами вершин графа и расстояний между вершинами (APSP). Решение крупномасштабной задачи APSP на современных многопроцессорных (многоядерных) системах является ключевым для различных областей применения. Вычислительные затраты на ее решение высоки, поэтому во многих случаях приемлемыми считаются приближенные решения. Перспективным подходом, позволяющим эффективно использовать множество процессоров (ядер) и их кэши в параллельном режиме, являются блочные алгоритмы APSP. В то же время, насколько нам известно, в блочных алгоритмах семейства Флойда-Уоршалла все блоки имеют одинаковый размер. Это свойство ограничивает применение алгоритмов. В статье предлагаются новые блочные алгоритмы, которые разбивают граф на неравные подграфы и разбивают матрицу расстояний между парами вершин на блоки неравного размера. Алгоритмы описывают плотные подграфы матрицей смежности, а разреженные подграфы и связи между ними – списком смежности. Такой подход позволяет совместно использовать алгоритмы семейства Флойда-Уоршалла с алгоритмами семейства Дейкстры. Он может быть применен к большим графам, декомпозированным на плотные (кластеры) и разреженные подграфы. Новый гетерогенный алгоритм может существенно сократить время вычисления блоков в зависимости от типа и размера. Вклад статьи заключается в разработке нового семейства блочных алгоритмов APSP, которые работают с блоками неравных размеров, сохраняют и расширяют преимущества алгоритмов, работающих с блоками равных размеров. Предложенные алго-ритмы реализованы в виде одно- и многопоточных параллельных приложений для многоядерных систем.

Ключевые слова: *Задача APSP, блочный алгоритм, неравные размеры блоков, разнородный алгоритм, многопоточная реализация, многоядерный процессор*

Anatoly Prihozhy is full professor at Computer and system software department of Belarus national technical university, D. Sc. (Eng) (1999) and Full Professor (2001). His research interests include programming languages, 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.

Прихожий А.А., профессор кафедры «Программное обеспечение информационных систем и технологий» Белорусского национального технического университета, д.т.н. (1999), диплом профессора (2001). В сферу его научных интересов входят языки программирования и описания аппаратуры, распараллеливающие компиляторы, методы и средства автоматизированного проектирования программных и аппаратных средств на логическом, высоком и системном уровнях, а также не полностью определенных логических систем. Имеет более 300 публикаций в Восточной и Западной Европе, США и Канаде. Его работы опубликованы в таких мировых издательствах, как IEEE, Springer, Kluwer Academic Publishers, World Scientific и других.

E-mail: prihozhy@bntu.by

Karasik Oleg is a Technology Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus, and PhD (Eng) (2019). His research interests include parallel multithreaded applications and the parallelization for multicore and multiprocessor systems.

О.Н. Карасик, ведущий инженер иностранного производственного унитарного предприятия «ИССОФТ СОЛЮШЕНЗ» (часть Coherent Solutions), г. Минск, Беларусь, к.т.н. (2019). В сферу его научных интересов входят параллельные многопоточные приложения и распараллеливание для многоядерных и многопроцессорных систем.

E-mail: karasik.oleg.nikolaevich@gmail.com