АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ
ALGORITHMIC AND PROGRAMMING

UDC 004.272.2

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

LOCALIZATION OF DATA REFERENCES IN BLOCKED HETEROGENEOUS SHORTEST PATHS ALGORITHM FOR CLUSTERED GRAPHS

The increasing size of real systems, for which the problem of shortest paths search is solved, necessitates the use of graph models that break the system into autonomous weakly interacting subsystems. Such models include clustered graphs consisting of weakly connected dense subgraphs (clusters) of different sizes. Block algorithms are an efficient solution to the problem of the all-pairs shortest paths. Heterogeneous block algorithms distinguish four types of unequally sized blocks, operate on models that are adequate to real objects, enable the use of architectural features of computing systems to be considered, and reduce the time it takes to calculate the shortest paths in large graphs. In this paper, two new algorithms for computing blocks of two types are developed. They are part of the heterogeneous block algorithm, consider the properties of clustered graphs, and are built on the specifics of the organization of computing architectures (in particular, multicore processors). An important property of these algorithms is their ability to spatially and temporally localize data references, to reduce data transfer traffic in multilevel memory, and to reduce the number of iterations of loops executed. To develop the algorithms formal methods were used to transform, optimize and prove correctness.

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

For citation: Prihozhy A. A., Karasik O. N. Localization of data references in blocked heterogeneous shortest paths algorithm for clustered graphs, 2025, no. 2 (296), pp. 83–90.
DOI: 10.52065/2520-6141-2025-296-11.

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

ЛОКАЛИЗАЦИЯ ССЫЛОК НА ДАННЫЕ В БЛОЧНОМ ГЕТЕРОГЕННОМ АЛГОРИТМЕ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ В КЛАСТЕРИЗОВАННЫХ ГРАФАХ

Увеличение размеров реальных систем, для которых решается задача поиска кратчайших путей, приводит к необходимости использования графовых моделей, разбивающих систему на автономные слабо взаимодействующие подсистемы. К таким моделям относятся графы, состоящие из различных по размеру слабосвязанных плотных подграфов (кластеров). Блочные алгоритмы являются эффективным решением задачи нахождения кратчайших путей на таких графах. Гетерогенные блочные алгоритмы выделяют четыре типа блоков неравного размера, оперируют моделями, адекватными реальным объектам, позволяют учитывать архитектурные особенности вычислительных систем, сокращают время вычисления кратчайших путей между всеми парами вершин в графах большого размера. В данной статье разработаны два новых алгоритма вычисления блоков двух типов, являющихся частью гетерогенного алгоритма. Они учитывают свойства кластеризованных графов и построены с учетом особенностей организации вычислительных архитектур (в частности, многоядерных процессоров). Важным свойством алгоритмов является способность к пространственно-временной локализации ссылок на данные, уменьшению трафика при передаче данных в многоуровневой памяти и сокращению числа итераций выполняемых циклов. Для создания алгоритмов, их преобразования, оптимизации и доказательства корректности использованы формальные методы.

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

Для цитирования: Прихожий А. А., Карасик О. Н. Локализация ссылок на данные в блочном гетерогенном алгоритме поиска кратчайших путей в кластеризованных графах // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2025. № 2 (296). С. 83–90 (На англ.).
DOI: 10.52065/2520-6141-2025-296-11.

Introduction. The classical Floyd-Warshall (*FW*) algorithm [1, 2] solves the all-pairs shortest paths problem (APSP) that is fundamental in numerous graph-based applications. The blocked *FW* algorithm (*BFW*) [3, 4] divides the graph into equally sized subgraphs and homogeneously uses a single procedure to compute all blocks of a distance matrix. The heterogeneous blocked *HBFW* algorithm [5] for dense graphs considers four types of blocks of the same size and uses a separate faster procedure for each type. Works [6 – 10] improve *BFW* and *HBFW* for solving scaling problems, efficient processor utilization, generating dataflow actor networks, reducing power consumption, etc. Work [11] generalizes *BFW* and *HBFW* for handling subgraphs and blocks of unequal sizes. The homogeneous *BFW* and *HBFW* are modified in [12, 13] for finding APSP in clustered graphs. Work [14] proposes the heterogeneous blocked shortest paths algorithm *HBSPCG* for clustered large graphs while considering bridge vertices and edges. The contribution of this paper is the development of two new sub-algorithms of *HBSPCG* for cross block types that localize data references and improve the utilization of processor memory [15].

Main part. *Clustered graphs*. A directed weighted graph $G = (V, E)$, $|V| = N$ is called clustered if it is partitioned to a set C of clusters, which are dense subgraphs connected by a small number of edges. The vertex set of cluster $c \in C$ is denoted $V(c)$ and the number of vertices in c is denoted $size(c)$. Graph G is represented by a cost-adjacent matrix $W[N\times N]$. The matrix of shortest path distances between vertices is decomposed into the blocked matrix $B[M\times M]$ where $M = |C|$. The graph is sparse if its edge density is low and approaches to 0. A subgraph is dense if its edge density is high and approaches to 1. Fig. 1 shows an example of directed clustered graph. A vertex of the cluster is called an in-bridge (out-bridge) if it has an incident bridge edge that is outgoing from (incoming to) another cluster.

Heterogeneous blocked algorithm for clustered graphs. In Work [14], we developed a new heterogeneous blocked Algorithm 1 for solving the APSP problem on large, clustered graphs while considering bridge vertices and bridge edges, unequally sized blocks, separate computation procedures for each block type. Four types of blocks (diagonal *D*0, vertical cross C1, horizontal cross C2 and peripheral P3) are computed with separate sub-algorithms *D*0*CG*, *C*1*CG*, *C*2*CG* and *P*3*CG*. Algorithm *D*0*CG* is optimized by localizing data references to the block elements to achieve higher performance, and algorithms *C*1*CG*, *C*2*CG* and *P*3*CG* are optimized to reduce the number of loop iterations due to the usage of input or output bridge vertices instead of all bridge vertices of the clusters.

Algorithm 1: Heterogeneous blocked APSP algorithm for clustered graphs (*HBSPCG*)

```
B[M×M] ← W[N×N]
for m ← 1 … M do
  D0CG(B, m)                                   // D0
  for c ∈ {1 … M} and v ≠ m do
      C1CG(C, B, c, m)                         // C1
      C2CG(C, B, m, c)                         // C2
  for c, e ∈ {1 … M} and c ≠ m and e ≠ m do
      P3CG(C, B, c, m, e)                      // P3
return B
```

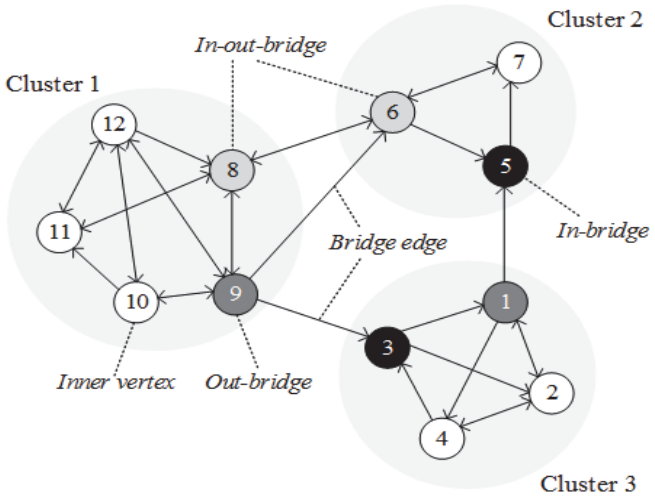

Fig. 1. Example of a clustered graph

Computation of diagonal blocks. Algorithm 2 that is proposed in [14] is an advanced alternative for the classical *FW* algorithm. Its input is the current state $B[m]^0$ of the diagonal block whose elements are considered as edge weights. Its output is the matrix of the shortest path distances between vertices of cluster m. Due to better loop iteration schemes and assignment statement grouping, *D*0*CG* outperforms *FW* in terms of processor hierarchical memory efficiency.

Algorithm 2: Computation of the diagonal square block $B[m]$ (*D*0*CGE*)

$B[m] \leftarrow B[m]^0$ $\quad S \leftarrow size[m]$
for $k \leftarrow 2, \ldots, S$ **do**
  for $i, j \in \{1, \ldots, k-1\}$ **do**
    $B[m]_{i,j} \leftarrow \min (B[m]_{i,j}, B[m]_{i,k-1} + B[m]_{k-1,j})$
    $B[m]_{i,k} \leftarrow \min (B[m]_{i,k}, B[m]_{i,j} + B[m]_{j,k})$
    $B[m]_{k,j} \leftarrow \min (B[m]_{k,j}, B[m]_{k,i} + B[m]_{i,j})$
for $i, j \in \{1, \ldots, S-1\}$ **do**
    $B[m]_{i,j} \leftarrow \min (B[m]_{i,j}, B[m]_{i,S} + B[m]_{S,j})$
return $B[m]$

Computation of vertical cross blocks. In [14], a vertical cross-block computation algorithm is proposed that uses bridge-vertices of clusters but do not localize data references. In this section, we develop an alternative algorithm that localizes data references and exploits bridge-vertices. Algorithm 3 (*C*1*CGE*) considers two clusters c and m and has two inputs: a rectangular vertical cross block $B[c, m]$ of type *C*1 and dimension $size(c) \times size(m)$, and a square diagonal block $B[m]$ of type *D*0 and dimension $size(m) \times size(m)$. It modifies the $B[c, m]$ block through the $B[m]$ block and keeps $B[m]$ unchanged. F_{in} is the first input bridge vertex in cluster m. The $B[c, m]$ block describes the shortest paths between vertices of cluster c and vertices of cluster m.

Algorithm 3: Computing vertical cross-block $B[c, m]$ by per-column extension upon bridges (*C*1*CGE*)

$S \leftarrow size(m)$
for $k \leftarrow F_{in} + 1$ **to** S **do**
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** k **do**
      if *InBridge*$(k-1)$ **then**
        $B[c, m]_{i,j} \leftarrow \min (B[c, m]_{i,j}, B[c, m]_{i,k-1} + B[m]_{k-1,j})$ // Θ_{k-1}
      if *InBridge*(j) **then**
        $B[c, m]_{i,k} \leftarrow \min (B[c, m]_{i,k}, B[c, m]_{i,j} + B[m]_{j,k})$ // Δ_k
if *InBridge*(S) **then**
    for $i \leftarrow 1$ **to** $size(c)$ **do**
      for $j \leftarrow 1$ **to** $S-1$ **do**
        $B[c, m]_{i,j} \leftarrow \min (B[c, m]_{i,j}, B[c, m]_{i,S} + B[m]_{S,j})$ // Θ_N
return $B[c, m]$

Unlike the classical blocked Ford-Warshall algorithm assuming equal sizes of all blocks, algorithm *C*1*CGE* supposes unequal hight and width of block $B[c, m]$. Moreover, it uses the input bridge vertices of cluster m to reduce the number of iterations to computate the $B[c, m]$ block. A predicate *InBridge*(v) takes value *true* if v is an input bridge-vertex of cluster m and takes value *false* otherwise. *C1CGE* is based on the ideas of column-wise block expansion, computation resynchronization, and locality of data references.

It should be noted that the competitive sub-algorithm *C*1*US* of algorithm *HBAPSPUS* [11] handles rectangular cross blocks but does not use bridge vertices of cluster m and therefore traverses all vertices of m while computing the shortest paths from cluster c to cluster m. Another competitive sub-algorithm *proc_bridges* of algorithm *CBA* [12] handles rectangular cross blocks and uses all input and output bridge vertices of cluster m, but it is homogeneous because of exploiting a single block calculation procedure for all types of blocks and does not perform the resynchronization of computations and localization of data references.

Theorem 1. Upon termination, Algorithm 3 (*C*1*CGE*) correctly computes the vertical cross-block $B[c, m]$ through the diagonal block $B[m]$.

Proof. The *C*1*CGE* algorithm starts with a one-vertex graph and one-column block $B[c, m]^1$ of dimension $size(c) \times 1$ and then iteratively adds one column k to the block $B[c, m]^{k-1}$ to obtain the block $B[c, m]^k$ of dimension $size(c) \times k$. To obtain the algorithm, the procedure shown in Fig. 2 is used. The initial state of the block $B[c, m]$ of dimension $size(c) \times size(m)$ is denoted $B[c, m]^0$.

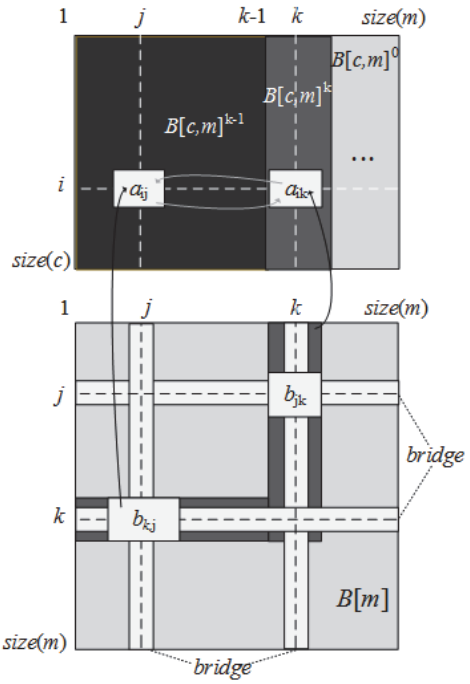

Fig. 2. Adding column k to block $B[c, m]$ in *C*1*CG* (a_{ij} denotes an element of block $B[c, m]$, and b_{jk} denotes an element of block $B[m]$)

The *C*1*CGE* algorithm uses two operations Δ_k and Θ_k for computing the block. Operation Δ_k computes using (1) the column k of $B[c, m]^k$ through the block $B[c, m]^{k-1}$ and the column k of block $B[m]$:

$$B[c,m]_{ik}^{j+1} = \min\begin{pmatrix} B[c,m]_{ik}^{j} \\ B[c,m]_{ij}^{k-1} + B[m]_{jk} \end{pmatrix} \quad (1)$$

for $i = 1 \ldots size(c), j = 1 \ldots k - 1$; vertex j is an input bridge of cluster m.

Operation Θ_k recalculates the block $B[c, m]^{k-1}$ to block $B[c, m]^k$ through column k using (2):

$$B[c,m]_{ij}^{k} = \min\begin{pmatrix} B[c,m]_{ij}^{k-1} \\ B[c,m]_{ik}^{k} + B[m]_{kj} \end{pmatrix} \quad (2)$$

for $i = 1 \ldots size(c), j = 1 \ldots k - 1$; vertex k is an input bridge of cluster m.

The behavior of *C*1*CGE* can be described as the following sequence of operation pairs (Δ_k, Θ_k):

$$(\Delta_1, \Theta_1), (\Delta_2, \Theta_2) \ldots (\Delta_k, \Theta_k) \ldots (\Delta_N, \Theta_N). \quad (3)$$

The sequence (3) is interpreted with Algorithm 4. Operation Δ_k is executed when the vertex with number j is an input bridge of cluster m, andoperation Θ_k is executed when the vertex k is an input bridge. The two nests of loops along i and j cannot be merged into one nest because the operations Δ_k and Θ_k cannot be executed simultaneously due to the mutual data dependences.

Algorithm 4: Recurrent procedure for computing vertical cross-block upon bridges (*C*1*CGE*)

$S \leftarrow size(m)$
for $k \leftarrow F_{in} + 1$ **to** S **do**
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** $k - 1$ **do**
      if *InBridge*(j) **then**
        $B[c, m]_{i,k} \leftarrow \min(B[c, m]_{i,k},$ // Δ_k
                $B[c, m]_{i,j} + B[m]_{j,k})$
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** $k - 1$ **do**
      if *InBridge*(k) **then**
        $B[c, m]_{i,j} \leftarrow \min(B[c, m]_{i,j},$ // Θ_k
                $B[c, m]_{i,k} + B[m]_{k,j})$
return $B[c, m]$

We resynchronize the computation process by rewriting sequence (3) to sequence (4).

$$\Delta_1, (\Theta_1, \Delta_2), (\Theta_2, \Delta_3) \ldots (\Theta_{k-1}, \Delta_k) \ldots (\Theta_{N-1}, \Delta_N), \Theta_N. \quad (4)$$

Fig. 3 illustrates the resynchronized process and Algorithm 5 realizes it. Its two nests of loops along i and j perform operations Θ_{k-1} and Δ_k. They can be merged into one loop nest, resulting in Algorithm 3. The theorem is proved.

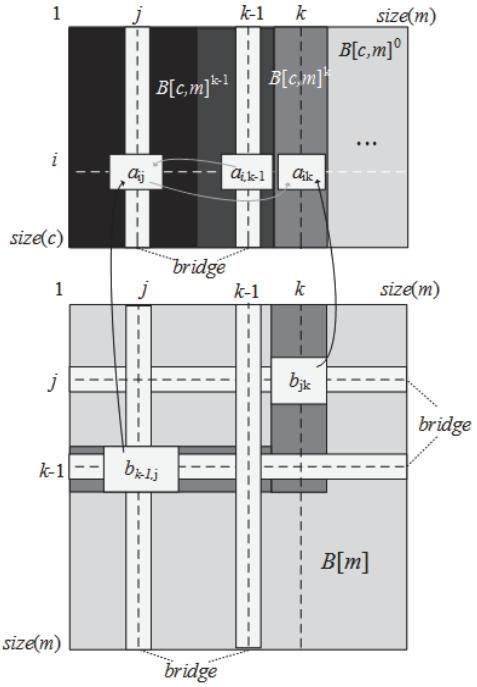

Fig. 3. Resynchronized process of adding a column

Algorithm 5: Computing vertical cross-block $B[c, m]$ by per-column extension upon bridges (*C*1*CGE*)

$S \leftarrow size(m)$
for $k \leftarrow F_{in} + 1$ **to** S **do**
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** k **do**
      if *InBridge*$(k - 1)$ **then**
        $B[c, m]_{i,j} \leftarrow \min(B[c, m]_{i,j},$
                $B[c, m]_{i,k-1} + B[m]_{k-1,j})$ // Θ_{k-1}
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** k **do**
      if *InBridge*(j) **then**
        $B[c, m]_{i,k} \leftarrow \min(B[c, m]_{i,k},$
                $B[c, m]_{i,j} + B[m]_{j,k})$ // Δ_k
if *InBridge*(S) **then**
  for $i \leftarrow 1$ **to** $size(c)$ **do**
    for $j \leftarrow 1$ **to** $S - 1$ **do**
      $B[c, m]_{i,j} \leftarrow \min(B[c, m]_{i,j},$
                $B[c, m]_{i,S} + B[m]_{S,j})$ // Θ_N
return $B[c, m]$

There is no need to perform the computation of the vertical cross-block $B[c, m]$ with Algorithm 3 if cluster m has no input bridge vertices. If the input bridge vertices are placed at the end of the cluster m vertex list, the number of iterations of the loop along k is reduced to the number of these vertices. In Algorithms 3 - 5, any of the *InBridge*$(\cdot)$ conditions can be omitted while preserving the correctness of the result.

Due to the iteration schemes of the loops along j, the number of conditional assignments is reduced up to 2 and the data references are localized in *C*1*CGE* compared to the *FW* algorithm. The loops iteratively process the $B[c, m]$ block of growing size $size(c) \times 1$

to $size(c) \times S$, so the references to the block elements are localized. Traversing only the input bridge-vertices of the m cluster further decreases the number of executions in Θ_{k-1} and Δ_k.

Computation of horizontal cross-blocks. Algorithm *C2CG* calculates the shortest paths connecting the vertices of cluster m to the vertices of cluster c and modifies the horizontal cross-block $B[m, c]$ of dimension $size(m) \times size(c)$ through the diagonal block $B[m]$. The *C2CG* algorithm assumes the sizes of blocks be unequal and exploits bridge vertices of clusters to speed up the computation of the shortest paths. Algorithm 6 is based on the idea of row-by-row block expansion and resynchronization of computations.

Theorem 2. Upon termination, Algorithm 6 (*C2CGE*) correctly computes the horizontal cross-block $B[m, c]$ through the diagonal block $B[m]$.

Algorithm 6: Calculating horizontal cross-block by row-per-row extension upon bridges (*C2CGE*)

$S \leftarrow size(m)$
for $k \leftarrow F_{out} + 1$ **to** S **do**
  for $i \leftarrow 1$ **to** k **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      if $OutBridge(k-1)$ **then**
        $B[m, c]_{i,j} \leftarrow \min(B[m, c]_{i,j}, B[m]_{i,k-1} + B[m, c]_{k-1,j})$ // Ω_{k-1}
      if $OutBridge(i)$ **then**
        $B[m, c]_{k,j} \leftarrow \min(B[m, c]_{k,j}, B[m]_{k,i} + B[m, c]_{i,j})$ // Λ_k
if $OutBridge(S)$ **then**
  for $i \leftarrow 1$ **to** $S-1$ **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      $B[m, c]_{i,j} \leftarrow \min(B[m, c]_{i,j}, B[m]_{i,S} + B[c, m]_{S,j})$ // Ω_N
return $B[m, c]$

Proof. In Algorithm 6, F_{out} is the first output bridge vertex in cluster m. The algorithm iteratively adds one row to the $B[m, c]$ block starting from a one-vertex graph and one-row block $B[m, c]^1$ of dimension $1 \times size(c)$. It computes the $B[m, c]^k$ block of dimension $k \times size(c)$ from the $B[m, c]^{k-1}$ block of dimension $(k-1) \times size(c)$. To obtain the algorithm, the procedure shown in Fig. 4 is used. The initial block state of dimension $size(m) \times size(c)$ is denoted $B[m, c]^0$.

The *C2CGE* algorithm uses two operations Λ_k and Ω_k for recomputing the block. Operation Λ_k computes the row k of block $B[m, c]^k$ through the block $B[m, c]^{k-1}$ and the row k of block $B[m]$:

$$B[m,c]_{kj}^{i+1} = \min \begin{pmatrix} B[m,c]_{kj}^{i} \\ B[m]_{ki} + B[m,c]_{ij}^{k-1} \end{pmatrix} \quad (5)$$

for $i = 1 \ldots k-1, j = 1 \ldots size(c)$; vertex i is an output bridge of cluster m. Operation Ω_k recalculates the elements of block $B[m, c]^{k-1}$ through row k of the block using (6):

$$B[m,c]_{ij}^{k} = \min \begin{pmatrix} B[m,c]_{ij}^{k-1} \\ B[m]_{ik} + B[m,c]_{kj}^{k} \end{pmatrix} \quad (6)$$

for $i = 1 \ldots k-1, j = 1 \ldots size(c)$; vertex k is an output bridge of cluster m. The behavior of *C2CGE* can be described with sequence (7) of pairs (Λ_k, Ω_k).

$$(\Lambda_1, \Omega_1), (\Lambda_2, \Omega_2) \ldots (\Lambda_k, \Omega_k) \ldots (\Lambda_N, \Omega_N). \quad (7)$$

Algorithm 7 realizes sequence (7).

Algorithm 7: Recurrent procedure for computing horizontal cross-block upon bridges

---

for $k \leftarrow F_{out} + 1$ **to** $size(m)$ **do**
  for $i \leftarrow 1$ **to** $k-1$ **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      if $OutBridge(i)$ **then**
        $B[m, c]_{k,j} \leftarrow \min(B[m, c]_{k,j}, B[m]_{k,i} + B[m, c]_{i,j})$ // Λ_k
  for $i \leftarrow 1$ **to** $k-1$ **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      if $OutBridge(k)$ **then**
        $B[m, c]_{i,j} \leftarrow \min(B[m, c]_{i,j}, B[m]_{i,k} + B[m, c]_{k,j})$ // Ω_k
return $B[m, c]$

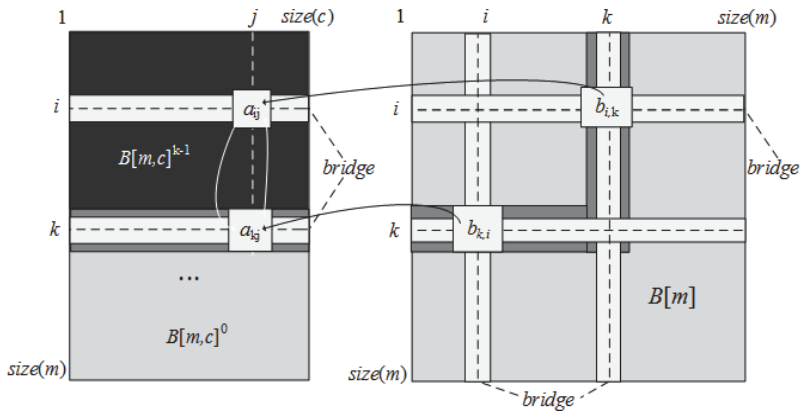

Fig. 4. Adding row k to block $B[m, c]$ in *C2CG*
(a_{ij} denotes an element of block $B[m, c]$, and b_{jk} denotes an element of block $B[m]$)

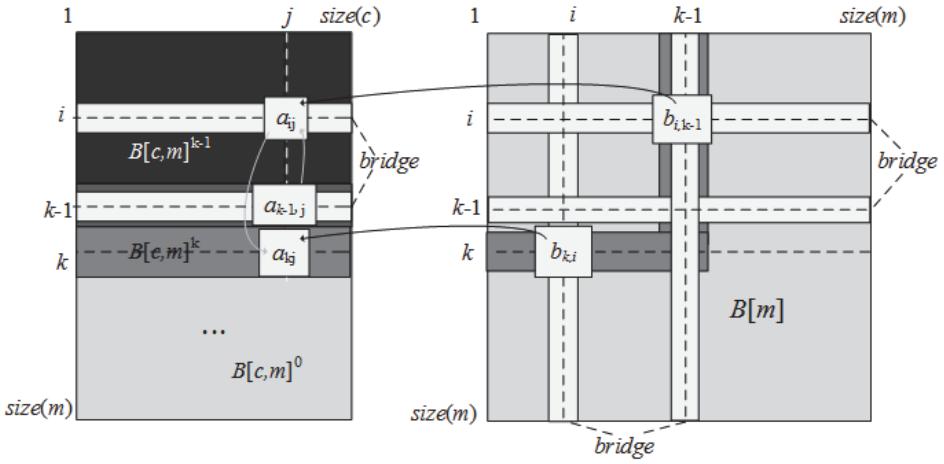

Fig. 5. Resynchronized process of adding a row

It is impossible to merge the Λ_k and Ω_k operations in the algorithm, therefore, we resynchronize the operations to obtain sequence (8).

$$\Lambda_1, (\Omega_1, \Lambda_2), (\Omega_2, \Lambda_3) \ldots (\Omega_{k-1}, \Lambda_k) \ldots (\Omega_{N-1}, \Lambda_N), \Omega_N. \quad (8)$$

The new computation process shown in Fig. 5 is realized by Algorithm 8. Two nests of loops along i and j perform operations Ω_{k-1} and Λ_k. After merging the nests, Algorithm 8 is transformed to Algorithm 6. The theorem is proved.

Algorithm 8: Calculating horizontal cross-block by row-per-row extension upon bridges (*C2CGE*)

$S \leftarrow size(m)$
for $k \leftarrow F_{out} + 1$ **to** S **do**
  for $i \leftarrow 1$ **to** k **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      if $OutBridge(k-1)$ **then**
        $B[m, c]_{i,j} \leftarrow \min(B[m, c]_{i,j}, B[m]_{i,k-1} + B[m, c]_{k-1,j})$ // Ω_{k-1}
  for $i \leftarrow 1$ **to** k **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      if $OutBridge(i)$ **then**
        $B[m, c]_{k,j} \leftarrow \min(B[m, c]_{k,j}, B[m]_{k,i} + B[m, c]_{i,j})$ // Λ_k
if $OutBridge(S)$ **then**
  for $i \leftarrow 1$ **to** $S - 1$ **do**
    for $j \leftarrow 1$ **to** $size(c)$ **do**
      $B[m, c]_{i,j} \leftarrow \min(B[m, c]_{i,j}, B[m]_{i,S} + B[c, m]_{S,j})$ // Ω_N
return $B[m, c]$

Algorithm 6 does not affect the horizontal cross-block $B[m, c]$ if cluster m has no output bridges. If the output bridge vertices are placed at the end of cluster m vertex list, the number of iterations of the loop along k is reduced to the number of these vertices. Due to the iteration schemes of the loops along i, the number of the conditional assignments is reduced up to 2 in *C2CGE* compared to the *FW* algorithm. The loops iteratively process the $B[m, c]$ block of growing size $1 \times size(c)$ to $S \times size(c)$, so the references to the block elements are localized. Traversing only the output bridge vertices of the m cluster reduces further the number of executions of Ω_{k-1} and Λ_k. Any of the $OutBridge(\cdot)$ conditions can be omitted while preserving the correctness of the result in Algorithms 6 – 8.

Computation of peripheral blocks. Algorithm 9 [14] computes a rectangular block $B[c, e]$ over two rectangular blocks $B[c, m]$ and $B[m, e]$. The vertex set $BridgesBest(m)$ is equal to $BridgesIn(m)$ if $|BridgesIn(m)| \le |BridgesOut(m)|$ and is equal to $BridgesOut(m)$ otherwise. Function $Index(v, m)$ returns the vertex v number in cluster m. *P3CG* speeds up the computation against *FW* depending on the size of the $BridgesBest(m)$ set compared to $size(m)$.

Algorithm 9: Calculation of peripheral rectangular block upon bridges and unequal block-sizes (*P3CG*)

---

for $i \leftarrow 1$ **to** $size(c)$ **do**
  for $v \in BridgesBest(m)$ **and** $k \leftarrow Index(v, m)$ **do**
    for $j \leftarrow 1$ **to** $size(e)$ **do**
      $B[c, e]_{i,j} \leftarrow \min(B[c, e]_{i,j}, B[c, m]_{i,k} + B[m, e]_{k,j})$
return $B[c, e]$

Conclusion. We have proposed new sub-algorithms *C1CGE* and *C2CGE* for computing vertical and horizontal cross-blocks of the heterogeneous all-pairs shortest path algorithm, which speed up computations and improve the locality of references to blocked data and hence aim to improve the operational efficiency of multicore processor hierarchical memory while solving the all-pairs shortest paths problem on large clustered weighted directed and undirected graphs.

References

1. Floyd R. W. Algorithm 97: Shortest path. *Communications of the ACM*, 1962, no. 5 (6), p. 345.
2. Warshall S. A theorem on Boolean matrices. *Journal of the ACM*, 1962, no. 9 (1), p. 11–12.
3. Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm. *Journal of Experimental Algorithmics (JEA)*, 2003, vol. 8, pp. 857–874.
4. 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.
5. Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algorithm. *Trudy BGTU* [Proceedings of BSTU], issue 3, Physics and Mathematics. Informatics, 2023, no. 1 (266), pp. 77–83.
6. Sangeetha D. P, Sekar S, Parvathy PR., GaneshBabu SRTR and Muthulekshmi M. Optimizing Shortest Paths in Big Data Using the Floyd-Warshall Algorithm, *International Conference on Intelligent Control, Computing and Communications (IC3)*. Mathura, India, 2025, pp. 382–387.
7. 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.
8. Kumar S., Karthik S., Srilakshmi S. and Dharun Viginesh P. Performance Analysis of Floyd-Warshall Algorithm: Sequential and Parallel Execution Using Intel oneAPI. *8th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. Coimbatore, India, 2024, pp. 205–211.
9. 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.
10. Prihozhy A. A., Karasik O. N. Influence of shortest path algorithms on energy consumption of multicore processors. *System analysis and applied information science*, 2023, no. 2, pp. 4–12.
11. Prihozhy A. A., Karasik O. N. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes. *System analysis and applied information science*, 2023, no. 4, pp. 4–13.
12. Karasik O. N., Prihozhy A. A. Blocked algorithm of finding all-pairs shortest paths in graphs divided into weakly connected clusters. *System analysis and applied information science*, 2024, no. 2, pp. 4–10.
13. Prihozhy A. A., Karasik O. N. Blocked algorithm of shortest paths search in sparse graphs partitioned into unequally sized clusters. *Big Data and Advanced Analytics X*. Minsk, 2024, pp. 262–271.
14. Prihozhy A. A., Karasik O. N. Heterogeneous blocked all-pairs shortest paths algorithm for clustered weighted graphs. *Journal of the Belarusian State University. Mathematics and Informatics*, 2025, no. 3 (Paper forthcoming).
15. 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.

Список литературы

1. Floyd R. W. Algorithm 97: Shortest path // Communications of the ACM. 1962. No. 5 (6). P. 345.
2. Warshall S. A theorem on Boolean matrices // Journal of the ACM. 1962. No. 9 (1). P. 11–12.
3. Venkataraman G., Sahni S., Mukhopadhyaya S. A Blocked All-Pairs Shortest Paths Algorithm // Journal of Experimental Algorithmics (JEA). 2003. Vol. 8. P. 857–874.
4. 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.
5. Prihozhy A. A., Karasik O. N. Advanced heterogeneous block-parallel all-pairs shortest path algo rithm // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2023. № 1 (266). С. 77–83.
6. Sangeetha D. P, Sekar S, Parvathy PR., GaneshBabu SRTR and Muthulekshmi M. Optimizing Shortest Paths in Big Data Using the Floyd-Warshall Algorithm // International Conference on Intelligent Control, Computing and Communications (IC3). Mathura, India. 2025. P. 382–387.
7. 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. P. 10–18.
8. Performance Analysis of Floyd-Warshall Algorithm: Sequential and Parallel Execution Using Intel oneAPI / S. Kumar [et al.] // 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA). Coimbatore, India. 2024. P. 205–211.
9. Prihozhy A. A. Generation of shortest path search dataflow networks of actors for parallel multicore implementation // Informatics. 2023. Vol. 20, no. 2. P. 65–84.
10. Prihozhy A. A., Karasik O. N. Influence of shortest path algorithms on energy consumption of multicore processors // System analysis and applied information science. 2023. No. 2. P. 4–12.
11. Prihozhy A. A., Karasik O. N. New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes // System analysis and applied information science. 2023. No. 4. P. 4–13.

12. Karasik O. N., Prihozhy A. A. Blocked algorithm of finding all-pairs shortest paths in graphs divided into weakly connected clusters // System analysis and applied information science. 2024. No. 2. P. 4–10.

13. Prihozhy A. A., Karasik O. N. Blocked algorithm of shortest paths search in sparse graphs partitioned into unequally sized clusters // Big Data and Advanced Analytics X. Minsk, 2024. P. 262–271.

14. Prihozhy A. A., Karasik O. N. Heterogeneous blocked all-pairs shortest paths algorithm for clustered weighted graphs // Journal of the Belarusian State University. Mathematics and Informatics. 2025. No. 3 (Paper forthcoming).

15. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms // System analysis and applied information science. 2021. No. 3. P. 40–50.

Information about the authors

Prihozhy Anatoly Alexievich − 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@bntu.by

Karasik Oleg Nikolaevich − PhD (Engineering), Tech Lead. 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 (пр-т Независимости 65, 220013, г. Минск, Республика Беларусь). E-mail: prihozhy@bntu.by

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

Received 15.04.2025