BSP/CGM Parallel Similarity Algorithms*†

C. E. R. Alves‡
Universidade São Judas Tadeu

E. N. Cáceres§
Universidade Federal do Mato Grosso do Sul

F. Dehne
Carleton University

S. W. Song

Universidade de São Paulo

* FAPESP, CNPq, PRONEX and NSERC

† *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures - SPAA '02*. Winnipeg, Canada, August 11-13, 2002, pp. 275-281. and *Proceedings I Brazilian Workshop on Bioinformatics*. Gramado, RS, Brazil, October 18, 2002, pp. 1-8.

‡ Doctorate student at Universidade de São Paulo

§ Visiting Professor at Universidade de São Paulo

String Editing Problem

Finding the edit distance between two strings A and C

Operations: insertion, deletion, substitution.

Edit Distance = Sum of the costs of each edit operation.

Applications in search for similarities in biosequences.

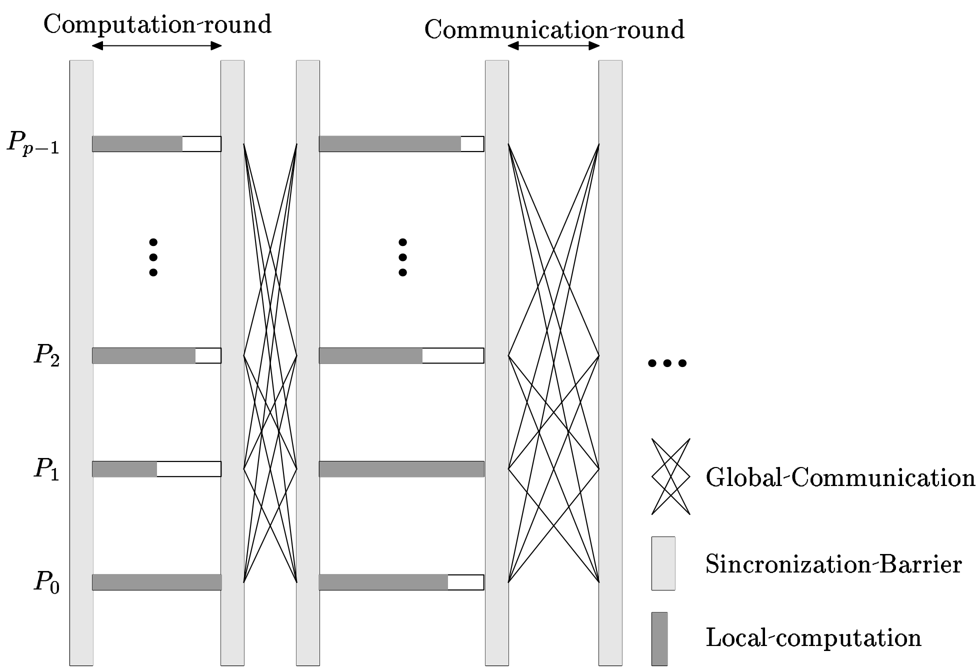
BSP/CGM Model

CGM (Coarse Grained Multicomputer) model: a "small" number of p of processors, each with its own local memory, communicating through a network.

The algorithm alternates between

- Computation rounds: each processor computes independently.
- Communication rounds: each processor sends/receives data to/from other processors.

BSP/CGM Model (cont.)



BSP/CGM Model (cont.)

Goals:

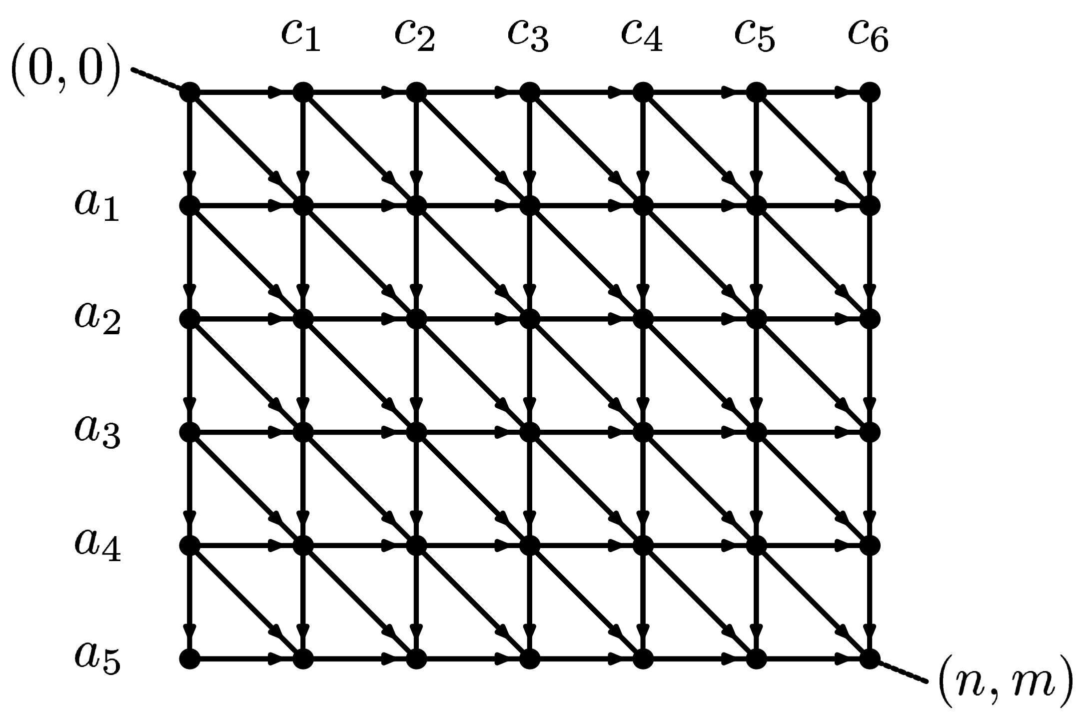
- Obtain a speed-up linear on p (for a range of values of p).
- Minimize the number of rounds.

Additional restrictions:

- The local memory of each processor is $O(N/p)$ (N is the space requirement for a sequential algorithm).
- Each processor send/receive at most $O(N/p)$ data in each round.

Dynamic Programming Approach

Illustrated by a grid directed acyclic graph.
Let $|A| = m$ and $|B| = n$.



If (r, s) has $r = s$: score $p(r, s) > 0$ (match)

If (r, s) has $r \neq s$: score $p(r, s) < 0$ (mismatch)

If we insert a space we subtract k from the score

$$S(r, s) = \max \begin{cases} S[r, s-1] - k \\ S[r-1, s-1] + p(r, s) \\ S[r-1, s] - k \end{cases}$$

Dynamic Programming (cont.)

So we can compute the values of $S(r,s)$ by using $S(r-1,s)$, $S(r-1,s-1)$ and $S(r,s-1)$ because there are only three ways of computing an alignment between $A[1\ldots r]$ and $C[1\ldots s]$:

. We can align $A[1..r]$ with $C[1..s-1]$ and match a space with $C[s]$,

. or align $A[1..r-1]$ with $C[1..s]$ and match a space with $A[r]$.

. or align $A[1..r-1]$ with $C[1..s-1]$ and match (or mismatch) $A[r]$ with $B[s]$,

Highest scoring path = best alignment.

Sequential algorithm: $O(mn)$ time.

Previous Parallel Algorithms

PRAM algorithms are known for the string editing problem.

Apostolico et al. 1990:

- CREW: $O(\log m \log n)$ time
  with $O(mn/\log m)$ processors ($n \geq m$)
- CRCW: $O(\log n (\log\log m)^2)$ time
  with $O(mn/\log\log m)$ processors
- in both case: $O(mn)$ space

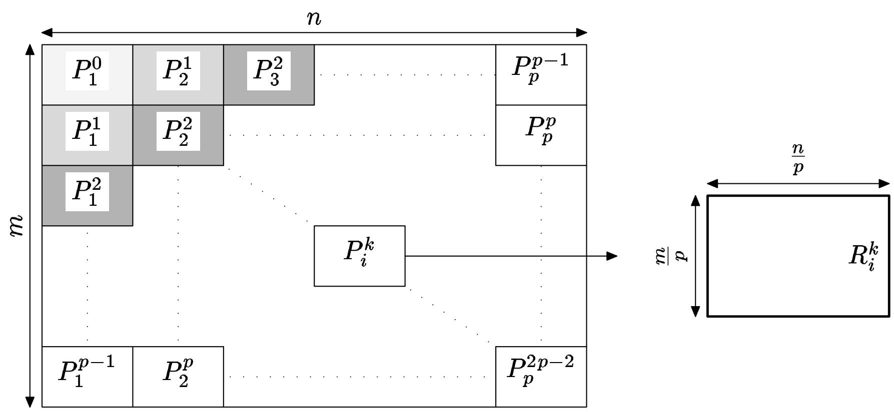
Galil and Park 1992:

- CREW: $O(\sqrt{n}\log n)$ time
  with $O(n^4)$ operations
- CREW: $O(\log^2 n)$ time
  with more processors

An $O(p)$ Commun. Rounds Algorithm

$A = \{a_1 \ldots a_m\}$, $C = \{c_1 \ldots c_n\}$ with $|A| = m$ and $|C| = n$

C is divided into p pieces of size $\frac{n}{p}$.

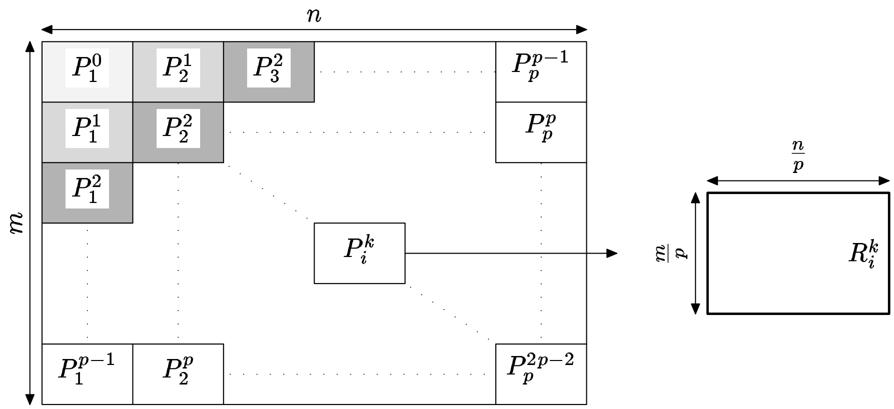


Each processor P_i receives A and the i-th piece of C.

Each P_i computes $S_i(r, s)$ of the submatrix S_i using the 3 previously computed elements $S_i(r-1, s)$, $S_i(r-1, s-1)$ and $S_i(r, s-1)$.

Processor P_i can only start to compute $S_i(r, s)$ after P_{i-1} has computed $S_{i-1}(r, s)$.

Idea of the Algorithm



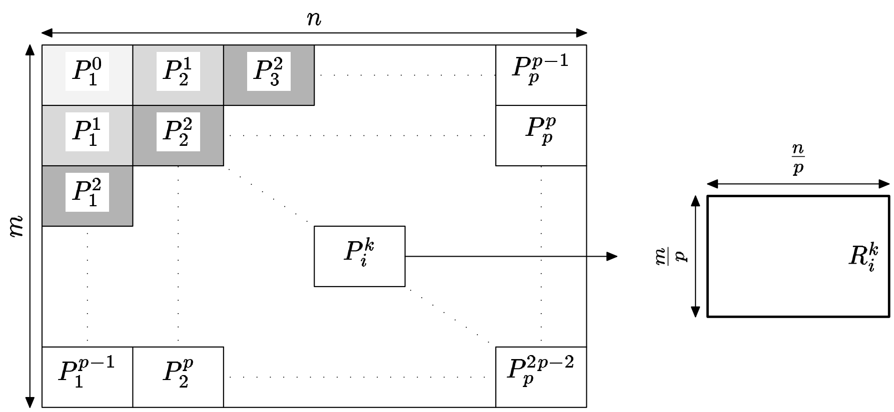
R_i^k, $1 \leq i, k \leq p$, elements of the right boundary (rightmost column) of the k-th part of submatrix S_i.

$R_i^k = \{S_i(r, i\frac{n}{p}), (k-1)\frac{m}{p} + 1 \leq r \leq k\frac{m}{p}\}$.

After computing the k-th part of the submatrix S_i, processor P_i sends the elements R_i^k to processor P_{i+1}.

Using R_i^k, processor P_{i+1} can compute the k-th part of the submatrix S_{i+1}.

Idea of the Algorithm (cont)



After $p-1$ rounds, processor P_p receives R_{p-1}^1 and computes the first part of the submatrix S_p.

In $2p-2$ rounds, processor P_p receives R_{p-1}^p and computes the p-th part of the submatrix S_p and the computation terminates.

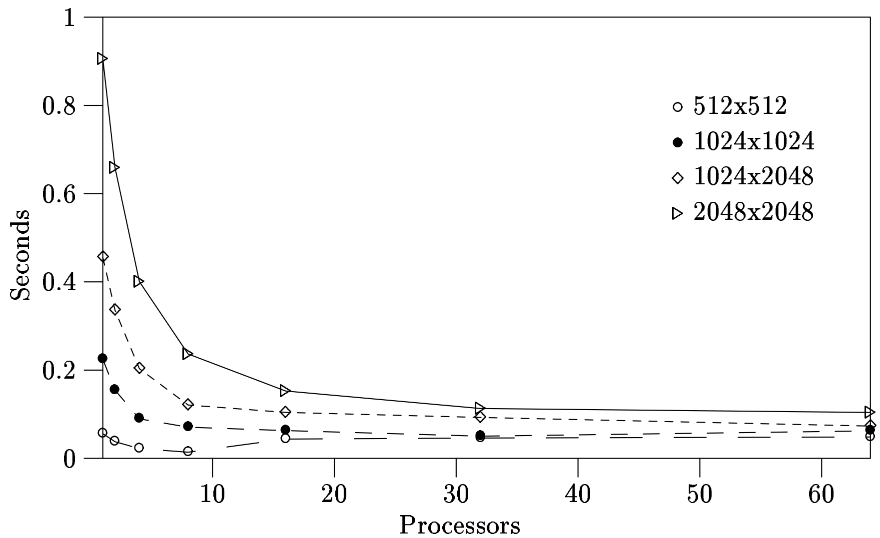
The Complete Algorithm

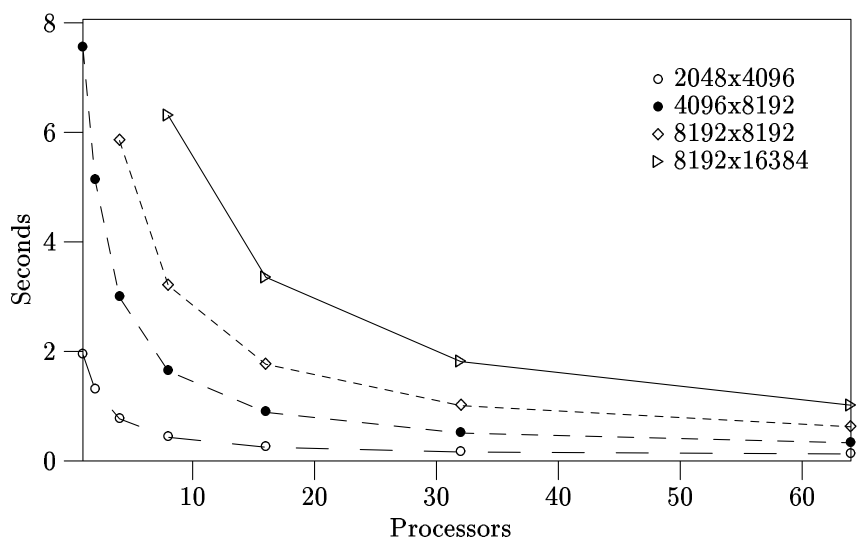
Algorithm 1 Similarity
Input: (1) The number p of processors; (2) The number i of the processor, where $1 \leq i \leq p$; and (3) The string A and the substring C_i of size m and $\frac{n}{p}$, respectively.
Output: $S(r,s) = \max\{S[r,s-1]-k, S[r-1,s-1]+p(r,s), S[r-1,s]-k\}$, where $(i-1)\frac{m}{\sqrt{p}}+1 \leq r \leq i\frac{m}{\sqrt{p}}$ and $(j-1)\frac{n}{p}+1 \leq s \leq j\frac{n}{p}$.

(1) **for** $1 \leq k \leq p$
  (1.1) **if** $i = 1$ **then**
    (1.1.1) **for** $(k-1)\frac{m}{p}+1 \leq r \leq k\frac{m}{p}$ **and** $1 \leq s \leq \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.1.2) `send(`R_i^k`,`P_{i+1}`);`
  (1.2) **if** $i \neq 1$ **then**
    (1.2.1) **receive**(R_{i-1}^k, P_{i-1});
    (1.2.2) **for** $(k-1)\frac{m}{p}+1 \leq r \leq k\frac{m}{p}$ **and** $1 \leq s \leq \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.2.3) **if** $i \neq p$ **then**
      `send(`R_i^k`,`P_{i+1}`);`

— End of Algorithm —

Implementation Results





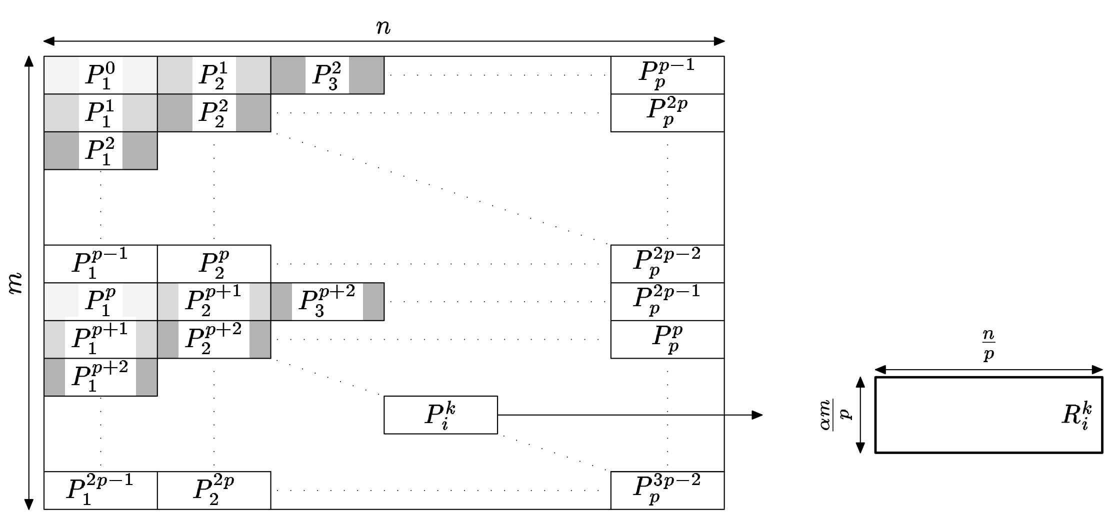
Improving this approach: a Parametrized Algorithm

The previous algorithm has a very bad load balancing.

Introduce a parameter $\alpha \leq 1$ to express the trade-off between the workload of each processor and the number of communication rounds required.

Small α means smaller workload and more communication rounds.

Case when $\alpha = 1/2$: ($3p-2$ communication rounds)



The Parametrized Algorithm

Algorithm 2 Similarity
Input: (1) The number p of processors; (2) The number i of the processor, where $1 \leq i \leq p$; and (3) The string A and the substring C_i of size m and $\frac{n}{p}$, respectively; (4) The constant α.
Output: $S(r,s) = \max\{S[r,s-1]-k, S[r-1,s-1]+p(r,s), S[r-1,s]-k\}$, where $(i-1)\frac{m}{\sqrt{p}}+1 \leq r \leq i\frac{m}{\sqrt{p}}$ and $(j-1)\frac{n}{p}+1 \leq s \leq j\frac{n}{p}$.

(1) **for** $1 \leq k \leq \frac{p}{\alpha}$

(1.1) **if** $i = 1$ **then**

(1.1.1) **for** $\alpha(k-1)\frac{m}{p}+1 \leq r \leq \alpha k\frac{m}{p}$ **and** $1 \leq s \leq \frac{n}{p}$

`compute` $S(r,s)$;

(1.1.2) `send(`R_i^k, P_{i+1}`);`

(1.2) **if** $i \neq 1$ **then**

(1.2.1) **receive**(R_{i-1}^k, P_{i-1});

(1.2.2) **for** $\alpha(k-1)\frac{m}{p}+1 \leq r \leq \alpha k\frac{m}{p}$ **and** $1 \leq s \leq \frac{n}{p}$

`compute` $S(r,s)$;

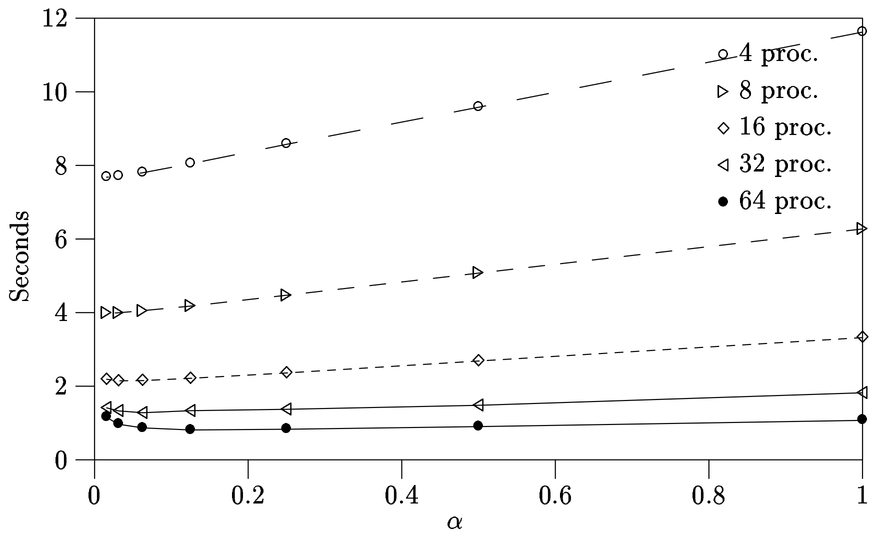
(1.2.3) **if** $i \neq p$ **then**

`send(`R_i^k, P_{i+1}`);`

— End of Algorithm —

Execution times for several values of α

Input strings: m=8000 and n=16000



Complexities

Theorem 1 *Algorithm 1 solves the string editing problem in the BSP/CGM model using* $2p - 2$ *communication rounds with local computation time of* $O(\frac{mn}{p})$ *in each processor.*

Theorem 2 *Algorithm 2 with parameter* α *solves the string editing problem in* $(1 + 1/\alpha)p - 2$ *communication rounds with local computation time of* $O(\frac{mn}{p})$ *in each processor.*

Partial Conclusion

An efficient CGM algorithm for the string editing problem.

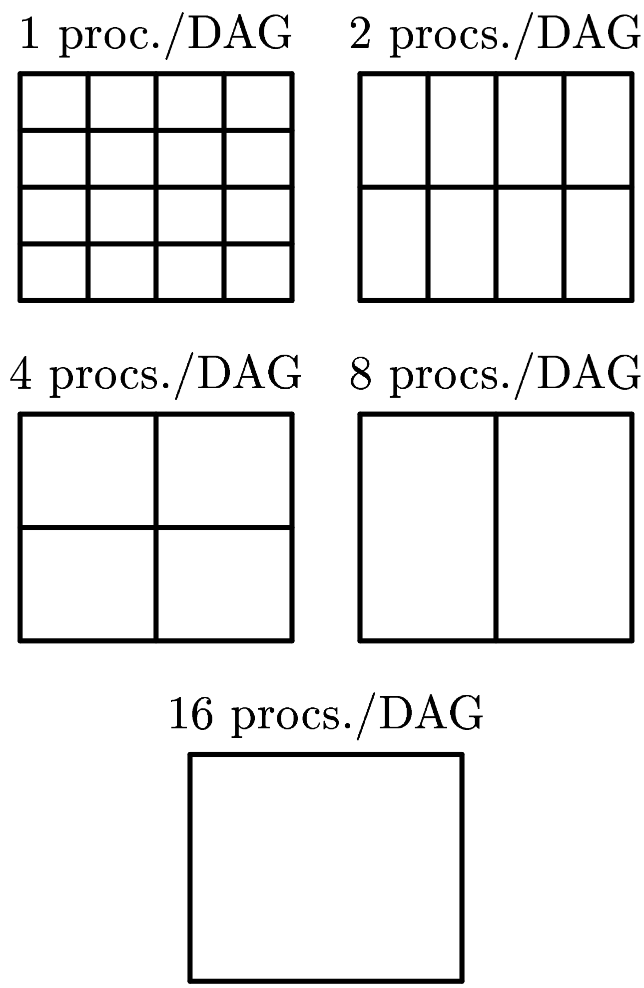
- Time and space requirements for the CGM model were met.

- The number of communication rounds is $O(p)$.

- Local computation time of $O(mn/p)$.

Can we decrease the number of communication rounds to $O(\log p)$???

Idea



String Editing Problem - extension

Finding the edit distance between A and *all* substrings of C.

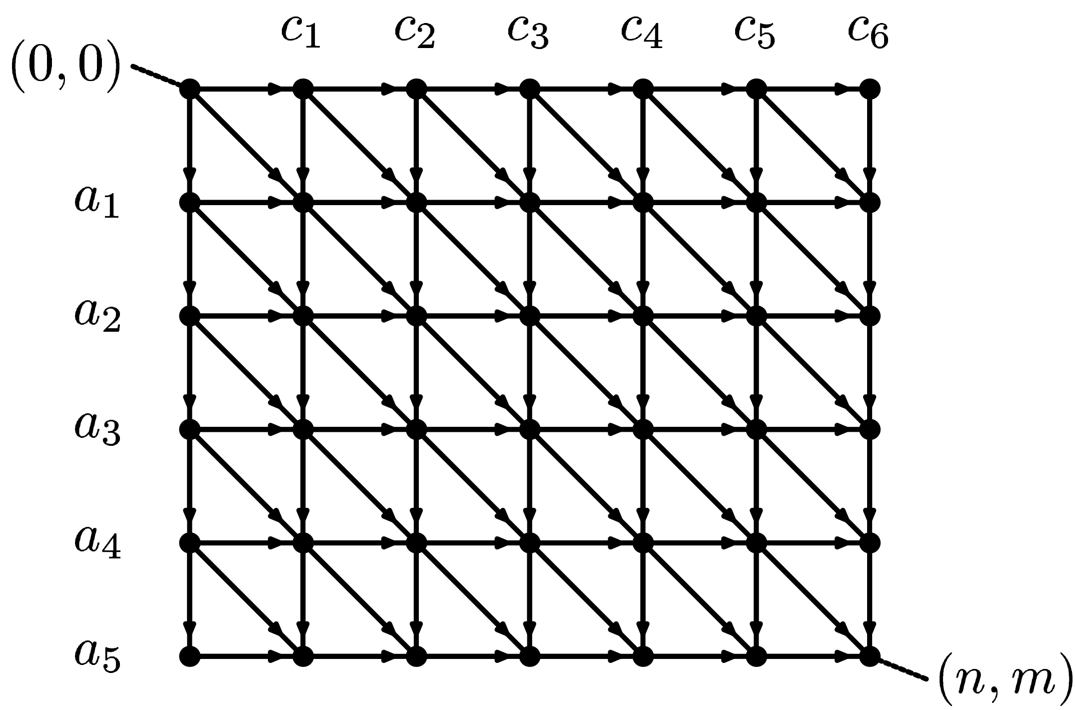
Applications of this problem:

- alignment of a string with several others that have a common substring.

- Finding tandem repeats in strings.

- Cyclic string comparison.

A common sequential approach: Dynamic Programming, best illustrated by a grid directed acyclic graph (grid DAG).

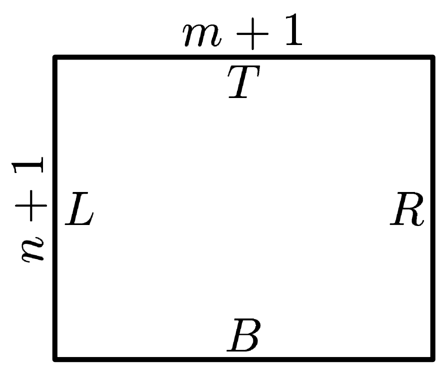


Highest scoring path = best alignment.

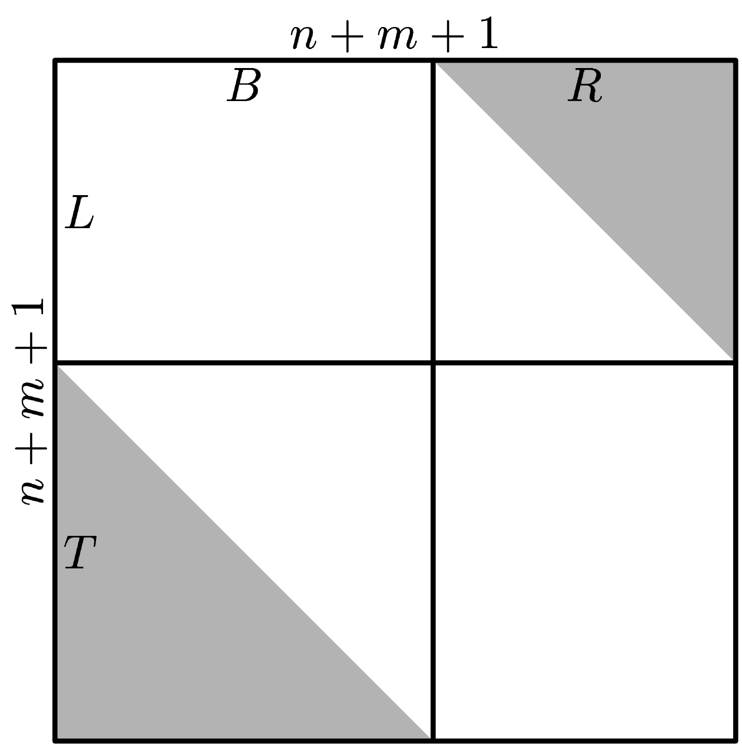
Our case: For all pairs of vertices in the borders, find the score of the best path.

Sequential algorithms exist with time $O(|A||C| \log \min\{|A|,|C|\})$

Structure of the DIST Matrix



GDAG G

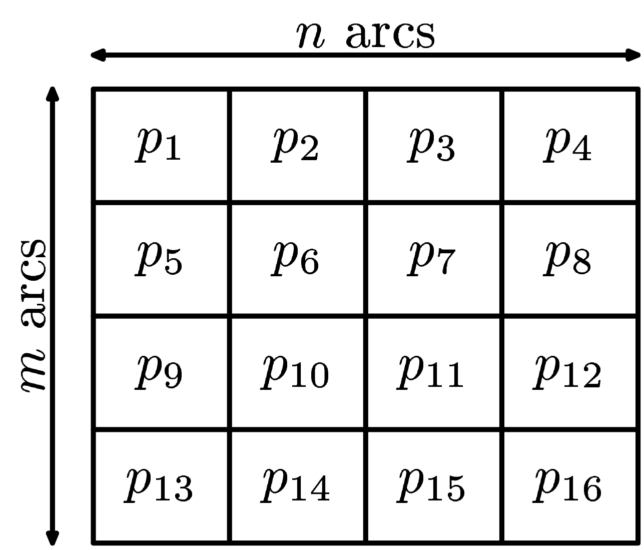


Matrix $DIST_G$

Main Strategy of the CGM Algorithm

The grid DAG is divided in p smaller DAGs, aligned in $\sqrt{p}$ rows of $\sqrt{p}$ DAGs. Each processor solves the problem sequentially.

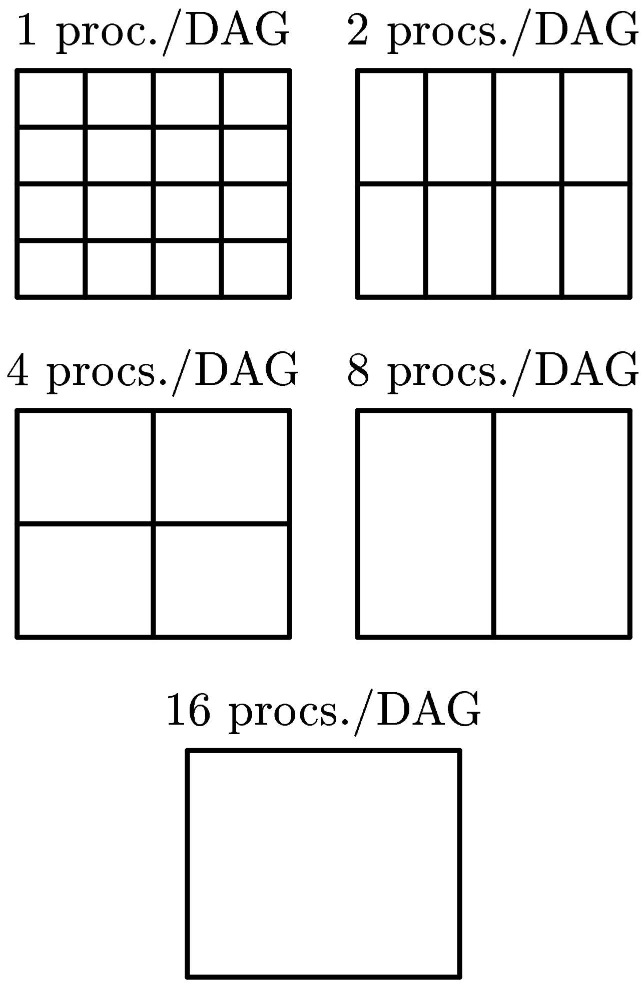
Example with $p = 16$:



$n \geq m \geq p^2$

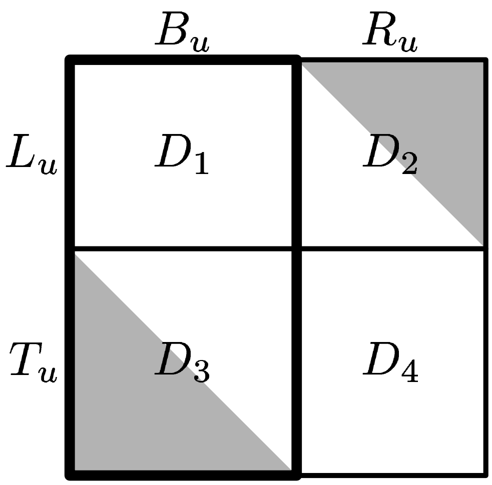
Time spent $= O\left(\frac{nm}{p} \log m\right)$

The partial solutions are joined together in $\log p$ steps, creating bigger DAGs.

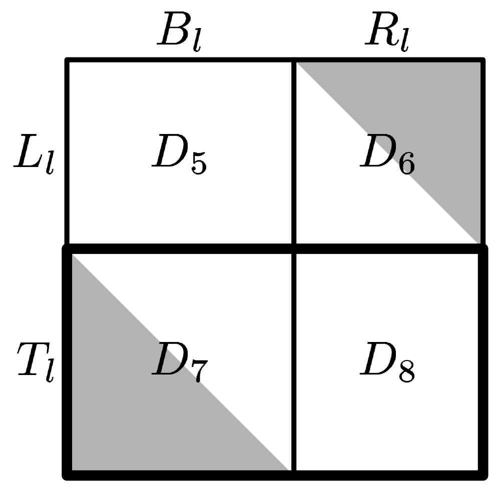


Each step takes $O\left(\frac{n^2}{p}\right)$.

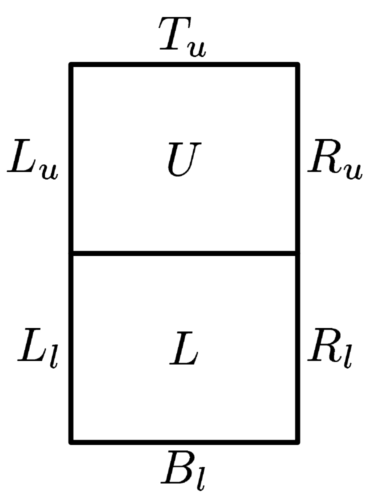
Joining Grids

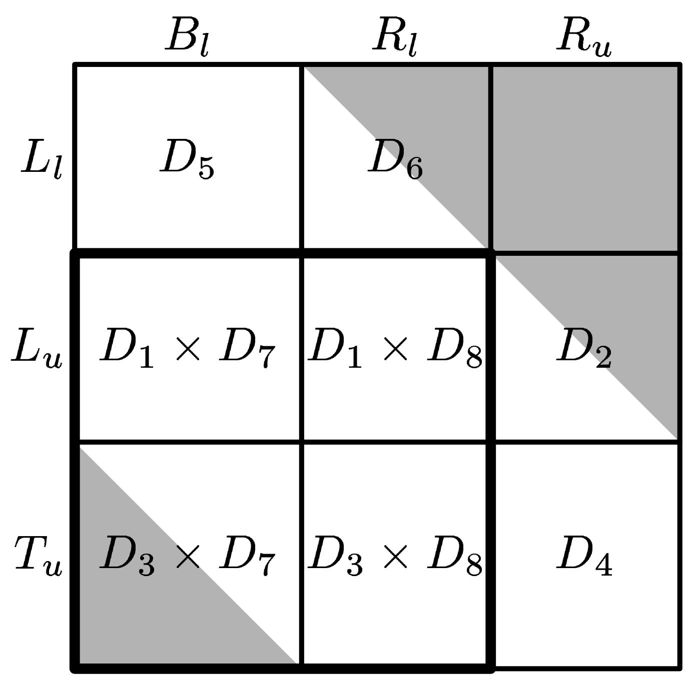


Matrix $DIST_U$



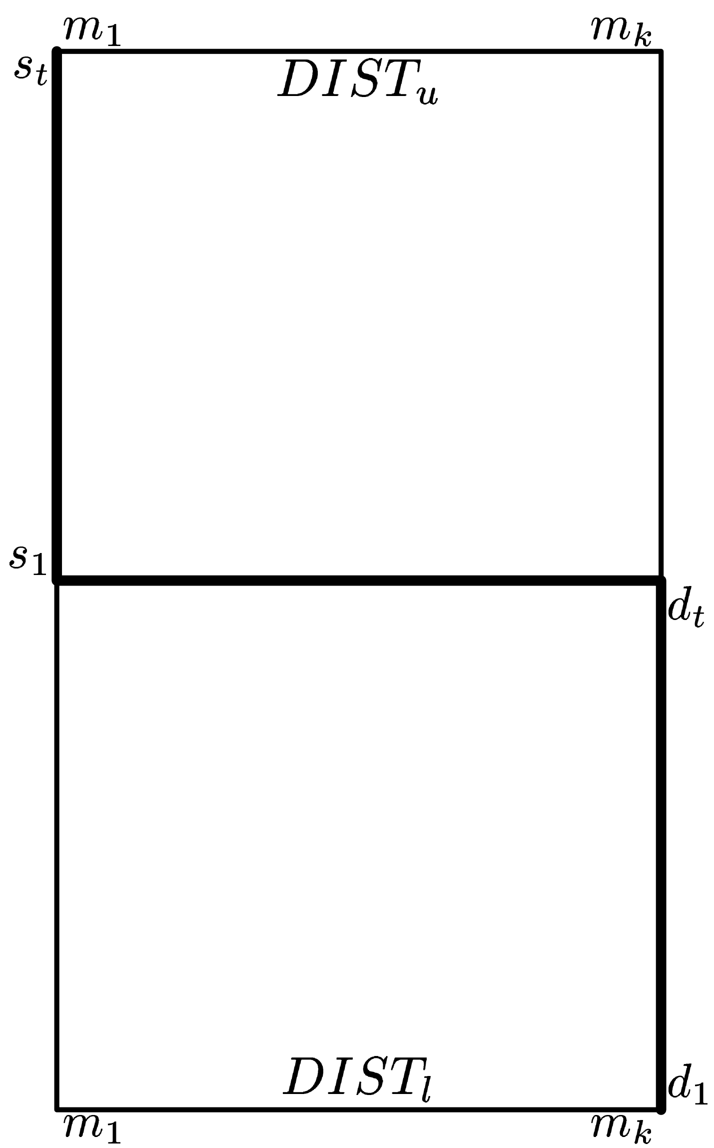
Matrix $DIST_l$





Matrix $DIST_{ul}$

An easier way to visualize the joining operation: $DIST_u$ and $DIST_l$ are displayed in a way that resembles the DAGs disposition.



source vertices

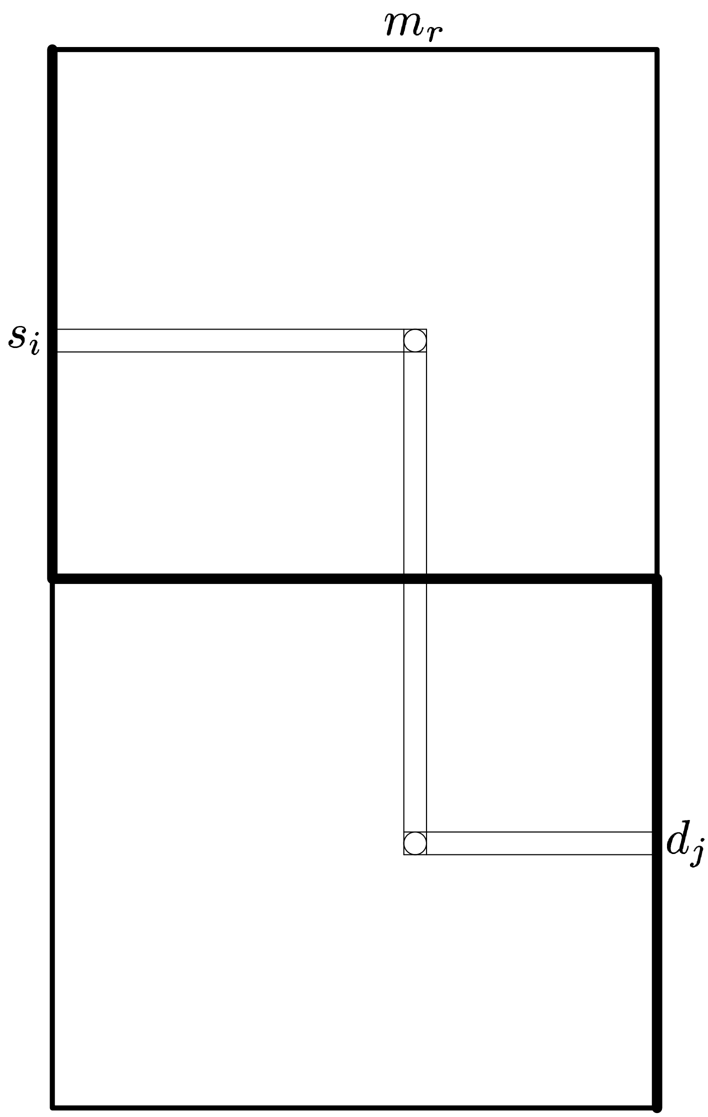
$$= s_1 \ldots s_t$$

dest. vertices

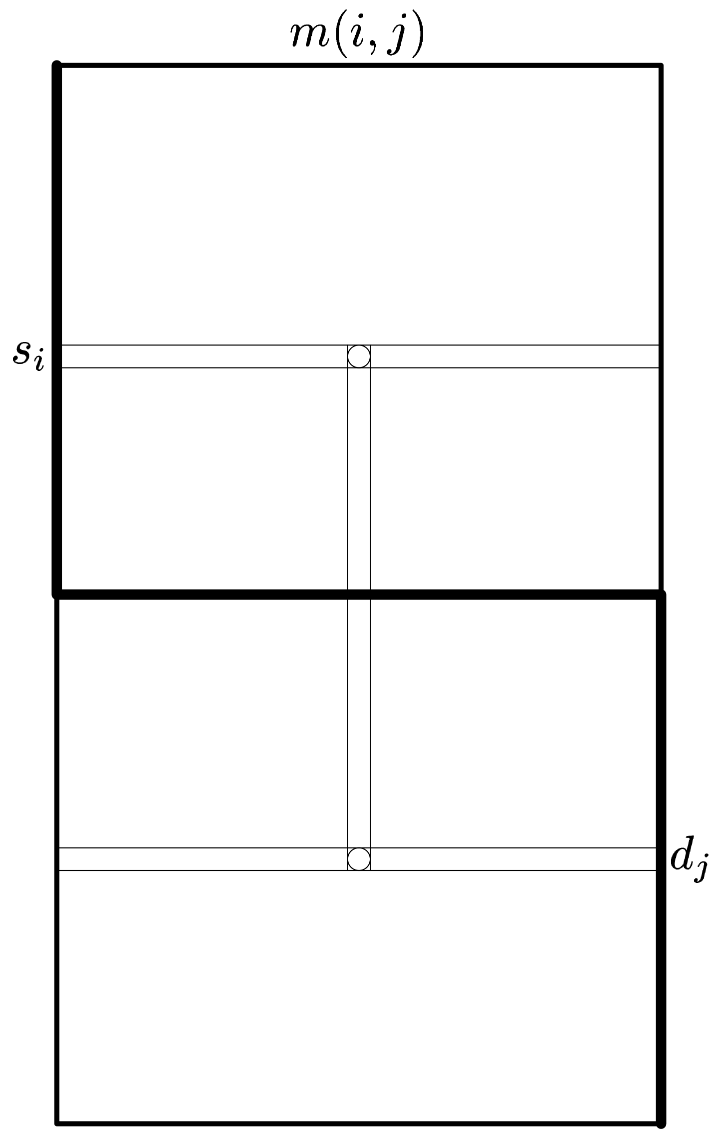
$$= d_1 \ldots d_t$$

middle vertices

$$= m_1 \ldots m_k$$

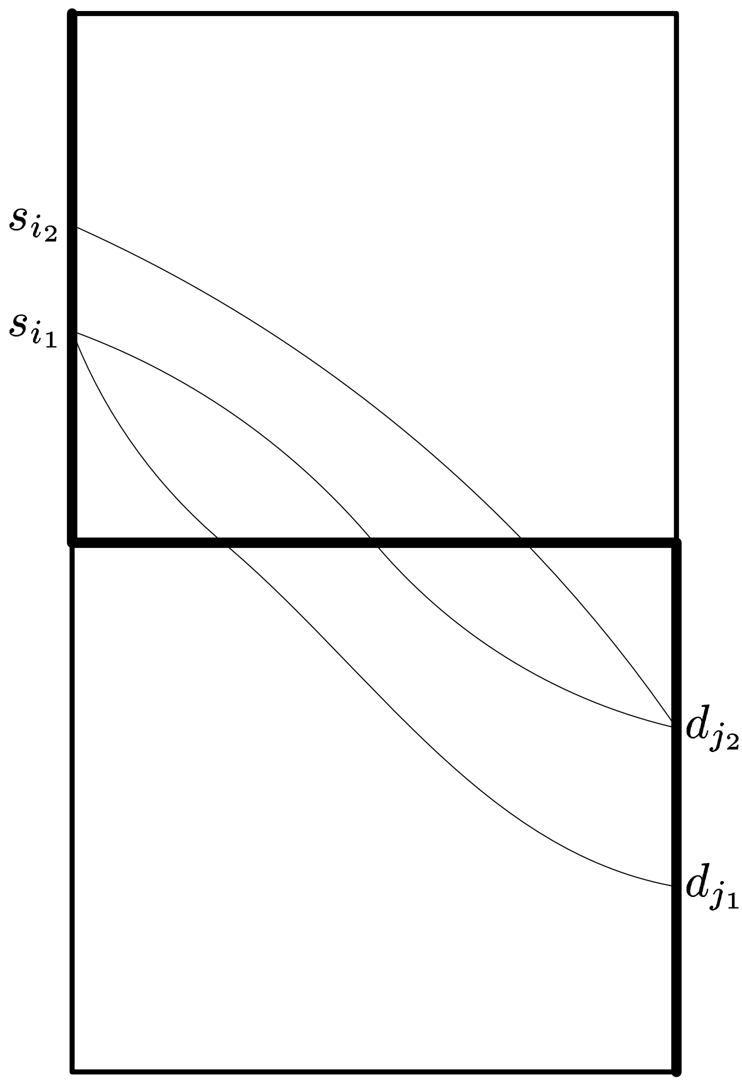


Best path from s_i to d_j through m_r:
$DIST_u(i, r) +$
$DIST_l(r, j)$



$m(i,j) = m_r$ that maximizes the previous sum.

Naïve search to find all best paths: time $= O(t^2 k)$.



Properties:

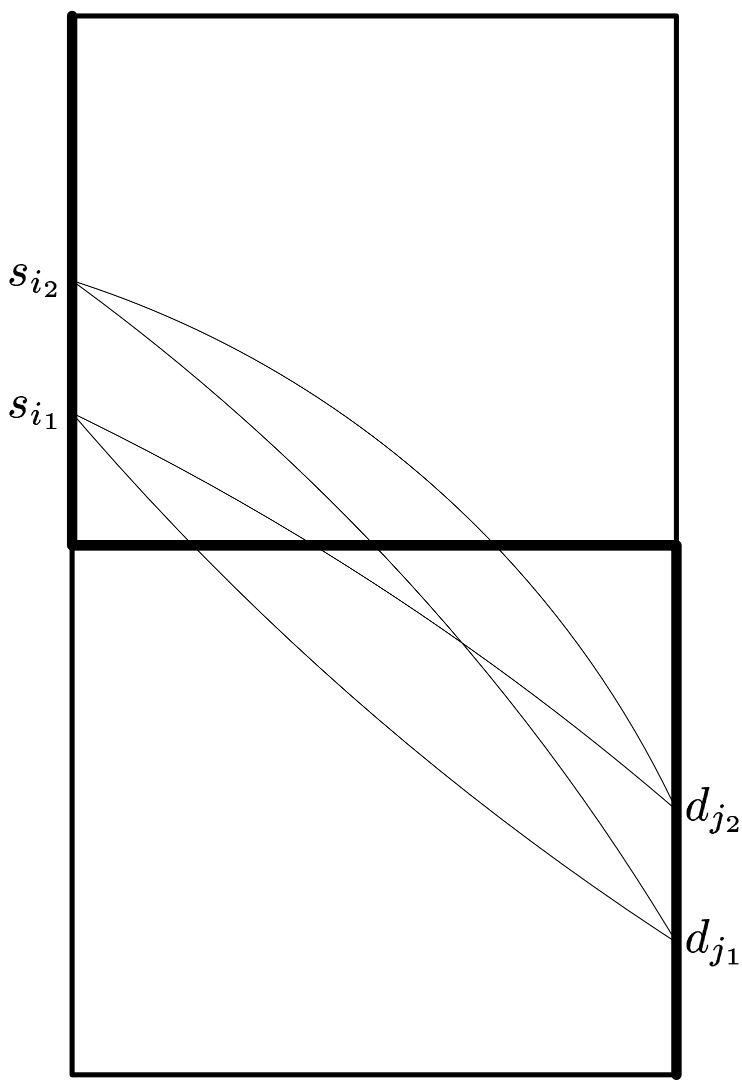
$i_1 < i_2 \Rightarrow$
$m(i_1, j) \leq m(i_2, j)$

$j_1 < j_2 \Rightarrow$
$m(i, j_1) \leq m(i, j_2)$

This properties lead to an $O(t^2 + tk)$ time sequential algorithm.

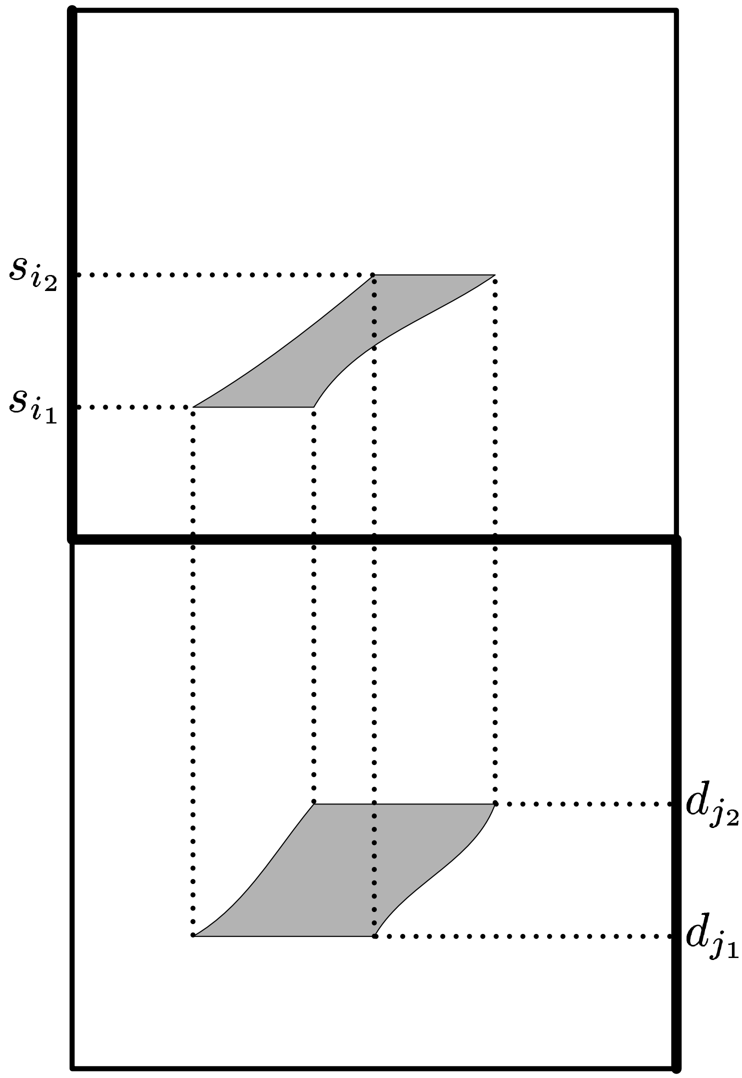
Joining Grids in Parallel

A subproblem in the joining operation:



Only sources between s_{i_1} and s_{i_2} and destinations between d_{j_1} and d_{j_2} are of interest.

A subproblem in the joining operation:



All the necessary data are contained in the shaded areas. The shapes are irregular.

Dividing the sources and the destinations in w intervals we have w^2 subproblems.

The data from $DIST_u$ ($DIST_l$) that is necessary to a subproblem is contained in a certain "area" of $DIST_u$ ($DIST_l$).

The "areas" of two distinct subproblems can overlap only in the borders.

Given the borders of the areas of a subproblem, the time and space requirements can be calculated in time $O(t/q)$.

Overview of the joining algorithm

$2q$ processors are used:

- $P_{u1}, P_{u2}, \ldots, P_{uq}$ hold $DIST_u$.
- $P_{l1}, P_{l2}, \ldots, P_{lq}$ hold $DIST_l$.

The sources and destinations are divided in $2q$ intervals, giving $4q^2$ subproblems.

Steps:

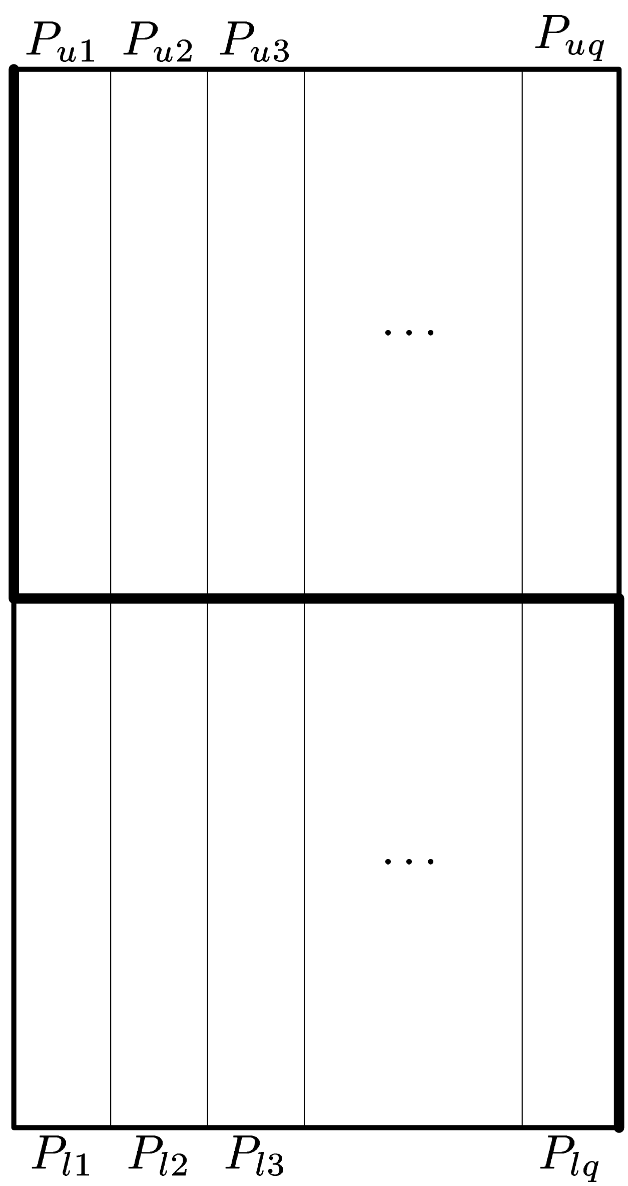
- Determine the areas of each subproblem.

- Estimate the cost to solve each subproblem.

- Distribute the subproblems among the $2q$ processors.

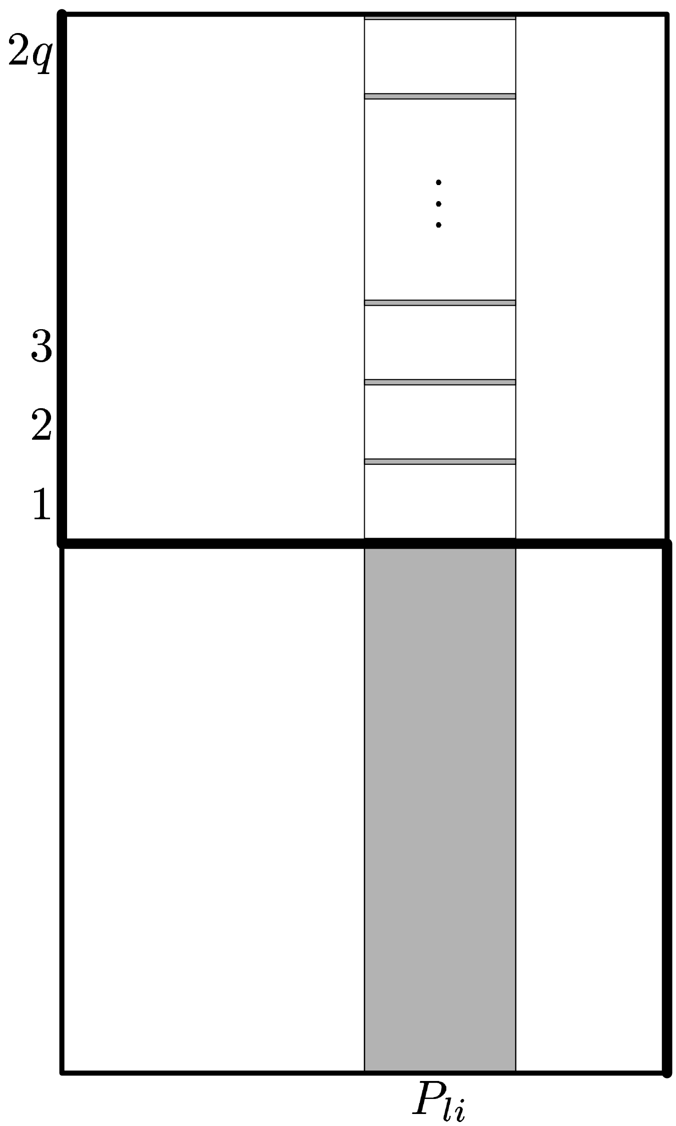
- Solve the subproblems and redistribute the results.

Data Distribution



$DIST_u$ distributed by columns, $DIST_l$ by rows.

Step1: determine the areas of each sub-problem.

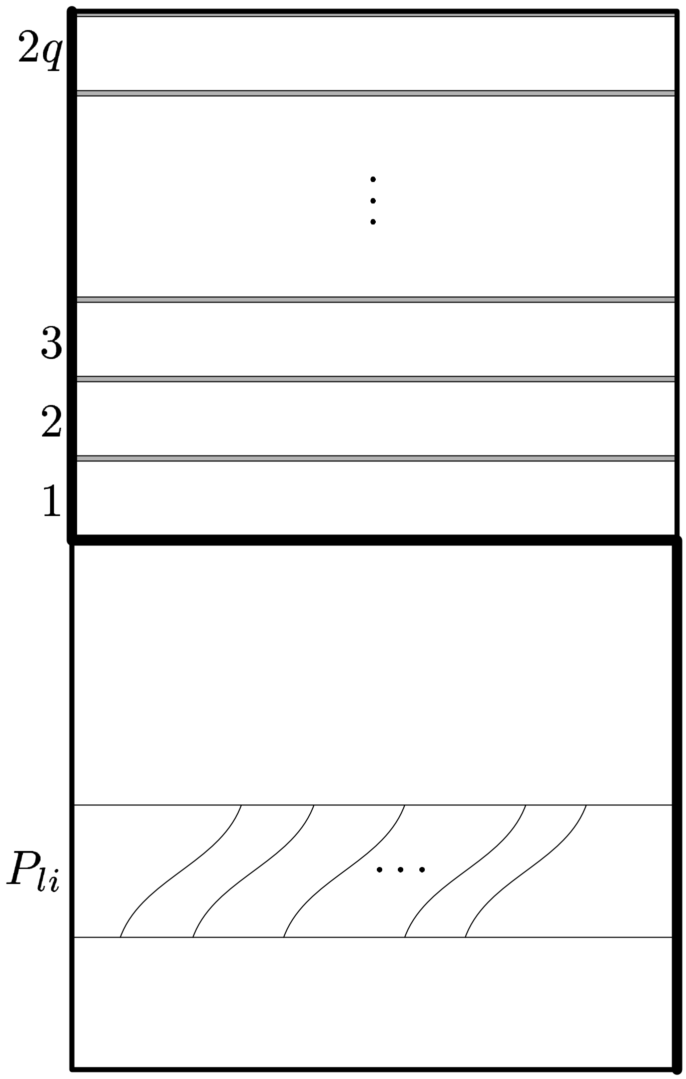


P_{li} finds the best paths from the *selected* sources to *all* destinations using *some* middle vertices.

Comm. $= O(k)$

Time $= O\left(k \log \left(\frac{t}{q}\right)\right)$

Step1: determine the areas of each sub-problem.



P_{li} choose the best paths from the *selected* sources to *some* destinations considering *all* middle vertices.

Comm. $= O(qt)$

Time $= O(t)$

Step2: estimate the time/space requirements.

Each of the $2q$ processors has (half) the information about the areas of $4q$ of the $4q^2$ subproblems.

Each processor:

- performs calculations for the time/space requirements,
- sends the results to P_{u1},
- sends informations about the borders of the subproblems to the processors that actually have the data.

Comm. $= O(q^2 + qt) = O(qt)$. Time $= O(qt)$.

Step3: distribute the subproblems among the processors.

P_{u1} totalizes the costs and performs a list scheduling.

- Biggest subproblem takes $1/2q$ of the total space and time requirements.

- Best possible solution has $O(t^2/q)$ local cost.

- List Scheduling finds a solution with local cost at most 4/3 of the cost of the best solution.

P_{u1} broadcast the results to all processors.

Comm. $= O(q^3)$. Time $= O(q^2 \log q)$.

Step4: compute subproblems.

Each processor

- sends/receives data for the subproblems,
- compute the results for his subproblems,
- distribute/receives results so the next joining step can take place.

P_{u1} broadcast the results to all processors.

Comm. $= O(kt/q)$ in two rounds. Time $= O(t^2/q)$.

Overview of the joining operation:

- 6 communication rounds, the one in Step 2 has size $O(qt)$ and limits the processor count to $\sqrt{m}$ in the overall algorithm anaysis.

- Time and space requirements are $O(t^2/q)$.

- In the overall analysis, the time and space requirements are $O(n^2/p)$.

Conclusion

An efficient CGM algorithm for the proposed problem was presented.

- Time and space requirements for the CGM model were met.
- The speed-up is linear on the number of processors p.
- The number of communication rounds is $O(\log p)$.

References

- Alves, C. E. R., Cáceres, E. N., Dehne, F. and Song, S. W. A Parameterized Parallel Algorithm for Efficient Biological Sequence Comparison. Technical Report RT-MAC-2002-06, Department of Computer Science, Institute of Mathematics and Statistics, University of So Paulo, August, 2002.