Efficient parallel string comparison

Peter Krusche

Department of Computer Science
University of Warwick

AFM Seminar, 23rd of April, 2007

Outline

1. Introduction
   - Parallel computation
   - Basic BSP Algorithms
   - String comparison
   - Literature
2. Sequential LCS algorithms
   - Sequential semi-local LCS
   - Divide-and-conquer semi-local LCS
3. The parallel algorithm
   - Parallel score-matrix multiplication
   - Parallel LCS computation

Outline

Why parallel algorithms?

. . . parallelism makes things more complicated. . .

But:

- everyone has a parallel computer today: Dual-Core Laptops, Playstation 3, . . .
- modern supercomputing architectures have more processors than ever
  (BlueGene/L: 131072, Grid-Computing: possibly millions)

Goals when designing parallel algorithms

- **Work-Optimality:** "Plain computation time of fastest sequential algorithm can be divided by p on p processors."
- **Reduce communication complexity:** Low communication complexity ⇒ good on slow communication networks (e.g. Grid-Computing / Clusters).
  Scalable Communication: More processors ⇒ less communication
- **Reduce latency:** Sending a single message to another processor (or reading a single byte from shared memory) incurs overhead ⇒ "Hide" latency by transferring large blocks.

Goals when designing parallel algorithms

- **Work-Optimality:** “Plain computation time of fastest sequential algorithm can be divided by p on p processors.”
- **Reduce communication complexity:** Low communication complexity ⇒ good on slow communication networks (e.g. Grid-Computing / Clusters).
  Scalable Communication: More processors ⇒ less communication
- **Reduce latency:** Sending a single message to another processor (or reading a single byte from shared memory) incurs overhead ⇒ “Hide” latency by transferring large blocks.

Goals when designing parallel algorithms

- **Work-Optimality:** "Plain computation time of fastest sequential algorithm can be divided by p on p processors."
- **Reduce communication complexity:** Low communication complexity $\Rightarrow$ good on slow communication networks (e.g. Grid-Computing / Clusters).
  Scalable Communication: More processors $\Rightarrow$ less communication
- **Reduce latency:** Sending a single message to another processor (or reading a single byte from shared memory) incurs overhead $\Rightarrow$ "Hide" latency by transferring large blocks.

Goals when designing parallel algorithms

- **Work-Optimality:** “Plain computation time of fastest sequential algorithm can be divided by p on p processors.”
- **Reduce communication complexity:** Low communication complexity $\Rightarrow$ good on slow communication networks (e.g. Grid-Computing / Clusters).
  Scalable Communication: More processors $\Rightarrow$ less communication
- **Reduce latency:** Sending a single message to another processor (or reading a single byte from shared memory) incurs overhead $\Rightarrow$ “Hide” latency by transferring large blocks.

Bulk-Synchronous Parallelism

Model for parallel computation:

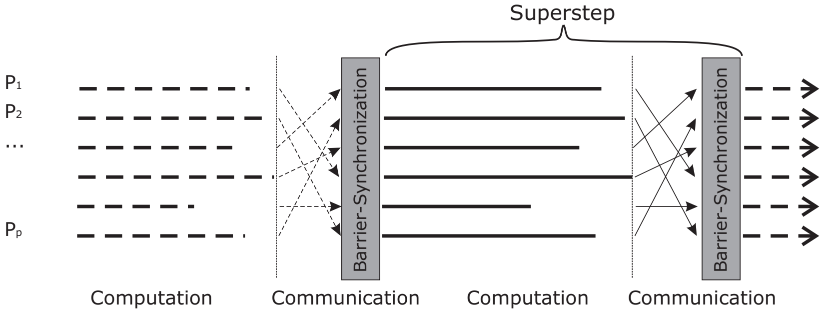
L. G. Valiant.
A bridging model for parallel computation.
Communications of the ACM, 33:103–111, 1990.

Main ideas:

- p processors working asynchronously
- Can communicate using g operations to transmit one element of data
- Can synchronise using l sequential operations.

Bulk-Synchronous Parallelism

- Computation proceeds in *supersteps*
- Communication takes place at the end of each superstep
- Between supersteps, barrier-style synchronisation takes place



Bulk-Synchronous Parallelism

Superstep s has computation cost w_s and communication $h_s = \max(h_s^{in}, h_s^{out})$.

When there are S supersteps:

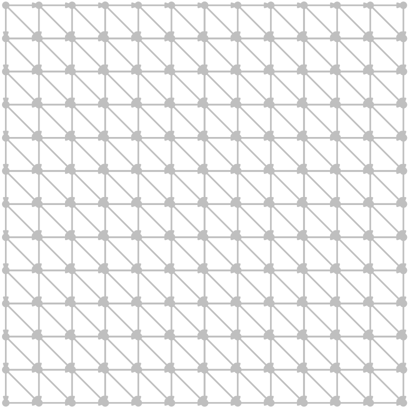
⇒ Computation work

$$W = \sum_{1 \leqslant s \leqslant S} w_s$$

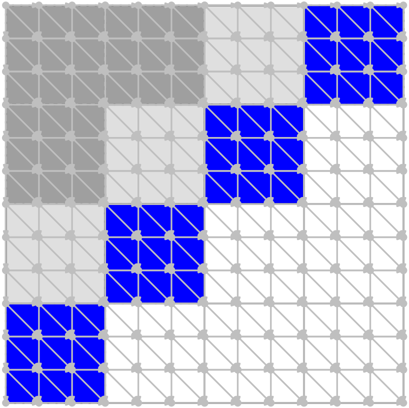
⇒ Communication

$$H = \sum_{1 \leqslant s \leqslant S} h_s$$

Formula for running time: $T = W + g \cdot H + l \cdot S$.



- Applicable for dynamic programming with grid-dag data-dependencies
- Can partition into p^2 boxes and proceed in wavefronts
- $2p - 1$ supersteps



- Applicable for dynamic programming with grid-dag data-dependencies
- Can partition into p^2 boxes and proceed in wavefronts
- $2p - 1$ supersteps

Definition (Parallel Prefix)

Given n values $x_1, x_2, \ldots, x_n$ and an associative operator $\oplus$, compute the values x_1, $x_1 \oplus x_2$, $x_1 \oplus x_2 \oplus x_3$, $\ldots$, $\bigoplus_{i=1,2,\ldots,n} x_i$.

Fact...

Under some natural assumptions, we can carry out a parallel prefix operation over n elements on a BSP computer with p processors using $W = O(\frac{n}{p})$, $H = O(p)$ and $S = O(1)$.

Definition (Input data)

Let $x = x_1 x_2 \ldots x_m$ and $y = y_1 y_2 \ldots y_n$ be two strings on an alphabet Σ.

Definition (Subsequences)

A *subsequence* u of x: u can be obtained by deleting zero or more elements from x.

Definition (Longest Common Subsequences)

An *LCS* (x, y) is any string which is subsequence of both x and y *and* has maximum possible length. Length of these sequences: *LLCS* (x, y).

The Semi-local LCS Problem

Definition (Substrings)

A *substring* of any string x can be obtained by removing zero or more characters from the beginning and/or the end of x.

Definition (Highest-score matrix)

The element $A(i, j)$ of the LCS *highest-score matrix* of two strings x and y gives the LLCS of $y_i \dots y_j$ and x.

Definition (Semi-local LCS)

Solutions to the semi-local LCS problem are represented by a (possibly implicit) highest-score matrix $A(i, j)$.

Outline

Parallel algorithms for LCS computation

W	H	S	**References**
	Global LCS		
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Wagner & Fischer'74]
	String-Substring LCS		
$O(\frac{n^2}{p})$	$O(Cp^{1/C} n \log p)$	$O(\log p)$	[Alves+'03]
$O(\frac{n^2}{p})$	$O(n \log p)$	$O(\log p)$	[Tiskin'05], [Alves+:06]
	String-Substring, Prefix-Suffix LCS		
$O(\frac{n^2 \log n}{p})$	$O(\frac{n^2 \log p}{p})$	$O(\log p)$	[Alves+'02]
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Alves+'06], [Tiskin'05]
$O(\frac{n^2}{p})$	$O(\frac{n \log p}{\sqrt{p}})$	$O(\log p)$	NEW

Parallel algorithms for LCS computation

W	H	S	**References**
	Global LCS		
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Wagner & Fischer'74]
	String-Substring LCS		
$O(\frac{n^2}{p})$	$O(Cp^{1/C} n \log p)$	$O(\log p)$	[Alves+'03]
$O(\frac{n^2}{p})$	$O(n \log p)$	$O(\log p)$	[Tiskin'05], [Alves+:06]
	String-Substring, Prefix-Suffix LCS		
$O(\frac{n^2 \log n}{p})$	$O(\frac{n^2 \log p}{p})$	$O(\log p)$	[Alves+'02]
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Alves+'06], [Tiskin'05]
$O(\frac{n^2}{p})$	$O(\frac{n \log p}{\sqrt{p}})$	$O(\log p)$	NEW

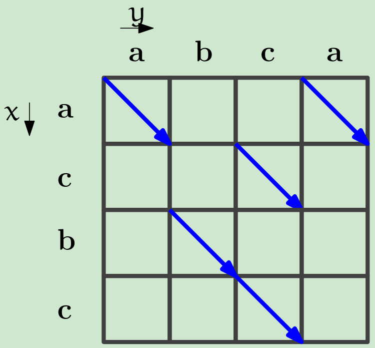
Parallel algorithms for LCS computation

W	H	S	**References**
	Global LCS		
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Wagner & Fischer'74]
	String-Substring LCS		
$O(\frac{n^2}{p})$	$O(Cp^{1/C} n \log p)$	$O(\log p)$	[Alves+'03]
$O(\frac{n^2}{p})$	$O(n \log p)$	$O(\log p)$	[Tiskin'05], [Alves+:06]
	String-Substring, Prefix-Suffix LCS		
$O(\frac{n^2 \log n}{p})$	$O(\frac{n^2 \log p}{p})$	$O(\log p)$	[Alves+'02]
$O(\frac{n^2}{p})$	$O(n)$	$O(p)$	[McColl'95]+ [Alves+'06], [Tiskin'05]
$O(\frac{n^2}{p})$	$O(\frac{n \log p}{\sqrt{p}})$	$O(\log p)$	NEW

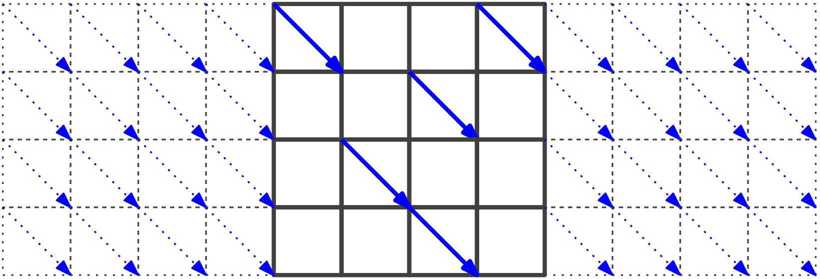
LCS grid dags and highest-score matrices

- LCS Problem can be represented as longest path problem in a Grid DAG
- String-Substring LCS Problem $\Rightarrow$ $A(i,j) =$ length of longest path from $(0,i)$ to (n,j) (top to bottom).

Example



Infinite extension of the LCS grid dag, outside the core area, everything matches:



The extended highest-score matrix is now defined on indices $[-\infty, +\infty] \times [-\infty, +\infty]$.

Definition (Critical Point)

Odd half-integer point $(i - \frac{1}{2}, j + \frac{1}{2})$ is *critical* iff. $A(i, j) + 1 = A(i - 1, j) = A(i, j + 1) = A(i - 1, j + 1)$.

Theorem (Schmidt'95, Alves+'06, Tiskin'05)

1. *We can represent a the whole extended highest-score matrix by a finite set of such critical points.*
2. *Assuming w.l.o.g. input strings of equal length n, there are $N = 2n$ such critical points that implicitly represent the whole score matrix.*
3. *There is an algorithm to obtain these points in time $O(n^2)$.*

Definition (Critical Point)

Odd half-integer point $(i - \frac{1}{2}, j + \frac{1}{2})$ is *critical* iff.
$A(i,j) + 1 = A(i-1,j) = A(i,j+1) = A(i-1,j+1)$.

Theorem (Schmidt'95, Alves+'06, Tiskin'05)

1. *We can represent a the whole extended highest-score matrix by a finite set of such critical points.*
2. *Assuming w.l.o.g. input strings of equal length* n*, there are* $N = 2n$ *such critical points that implicitly represent the whole score matrix.*
3. *There is an algorithm to obtain these points in time* $O(n^2)$.

Example (Explicit highest-score matrix)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
-4	4	4	4	4	4	4	4	4
-3	3	3	3	4	4	4	4	4
-2	2	2	3	4	4	4	4	4
-1	1	1	2	3	3	3	4	4
0	0	1	2	3	3	3	4	4
1	-1	0	1	2	2	2	3	4
2	-2	-1	0	1	1	2	3	4
3	-3	-2	-1	0	1	2	3	4

Example (Implicit score matrix)

$$\left(-\tfrac{7}{2}, \tfrac{5}{2}\right), \left(-\tfrac{5}{2}, \tfrac{3}{2}\right), \left(-\tfrac{3}{2}, \tfrac{11}{2}\right), \left(-\tfrac{1}{2}, \tfrac{1}{2}\right),$$
$$\left(\tfrac{1}{2}, \tfrac{13}{2}\right), \left(\tfrac{3}{2}, \tfrac{9}{2}\right), \left(\tfrac{5}{2}, \tfrac{7}{2}\right), \left(\tfrac{1}{2}, \tfrac{15}{2}\right)$$

Example (Explicit highest-score matrix)

	0	1	2	3	4	5	6	7
-4	4	4	4	4	4	4	4	4
-3	3	3	3	4	4	4	4	4
-2	2	2	3	4	4	4	4	4
-1	1	1	2	3	3	3	4	4
0	0	1	2	3	3	3	4	4
1	-1	0	1	2	2	2	3	4
2	-2	-1	0	1	1	2	3	4
3	-3	-2	-1	0	1	2	3	4

Example (Implicit score matrix)

$$\left(-\tfrac{7}{2}, \tfrac{5}{2}\right), \left(-\tfrac{5}{2}, \tfrac{3}{2}\right), \left(-\tfrac{3}{2}, \tfrac{11}{2}\right), \left(-\tfrac{1}{2}, \tfrac{1}{2}\right),$$
$$\left(\tfrac{1}{2}, \tfrac{13}{2}\right), \left(\tfrac{3}{2}, \tfrac{9}{2}\right), \left(\tfrac{5}{2}, \tfrac{7}{2}\right), \left(\tfrac{1}{2}, \tfrac{15}{2}\right)$$

Additional definitions

Definition (Integer ranges)

We denote the set of integers $\{i, i+1, \ldots, j\}$ as $[i : j]$.

Definition (Odd half-integers)

We denote half-integer variables using a $\hat{}$, and denote the set of half-integers $\{i + \frac{1}{2}, i + \frac{3}{2}, \ldots, j - \frac{1}{2}\}$ as $\langle i : j \rangle$.

Querying highest-score matrix entries

Theorem (Tiskin'05)

If $d(i,j)$ is the number of critical points $(\hat{\imath},\hat{\jmath})$ in the extended score matrix with $i < \hat{\imath}$ and $\hat{\jmath} < j$, then $A(i,j) = j - i - d(i,j)$.

Definition (Density and distribution matrices)

The elements $d(i,j)$ form a distribution matrix over the entries of a density (permutation) matrix D which uses odd half-integer indices and has nonzeros at all critical points $(\hat{\imath},\hat{\jmath})$ in the extended highest-score matrix:

$$d(i,j) = \sum_{(\hat{\imath},\hat{\jmath}) \in \langle i:N \rangle \times \langle 0:j \rangle} D(\hat{\imath},\hat{\jmath})$$

Querying highest-score matrix entries

Theorem (Tiskin'05)

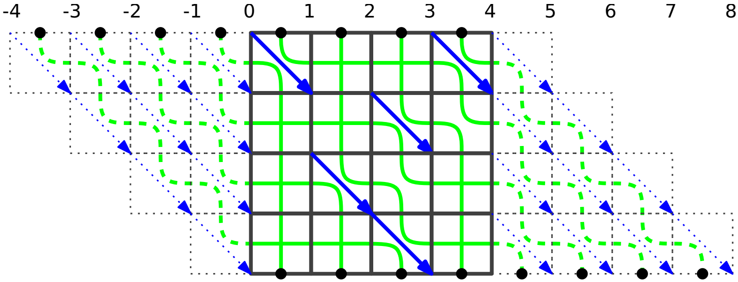
If $d(i, j)$ *is the number of critical points* $(\hat{\imath}, \hat{\jmath})$ *in the extended score matrix with* $i < \hat{\imath}$ *and* $\hat{\jmath} < j$*, then* $A(i, j) = j - i - d(i, j)$*.*

Definition (Density and distribution matrices)

The elements $d(i, j)$ form a distribution matrix over the entries of a density (permutation) matrix D which uses odd half-integer indices and has nonzeros at all critical points $(\hat{\imath}, \hat{\jmath})$ in the extended highest-score matrix:

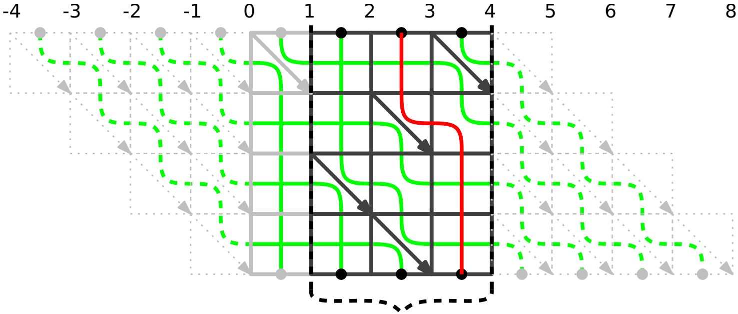
$$d(i, j) = \sum_{(\hat{\imath}, \hat{\jmath}) \in \langle i : N \rangle \times \langle 0 : j \rangle} D(\hat{\imath}, \hat{\jmath})$$

Critical points can be drawn as “seaweeds” in the grid graph



“Seaweed” chart

Critical points can be drawn as “seaweeds” in the grid graph

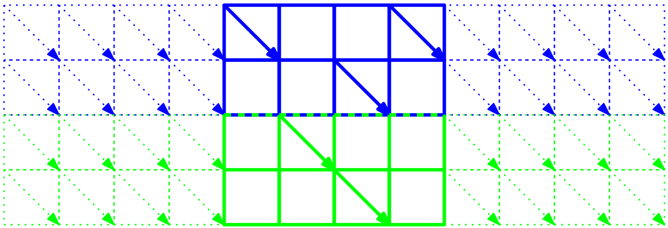


$$\mathrm{d}(1,4) = 1$$
$$A(1,4) = 4 - 1 - 1 = 2$$

Sequential highest-score matrix multiplication

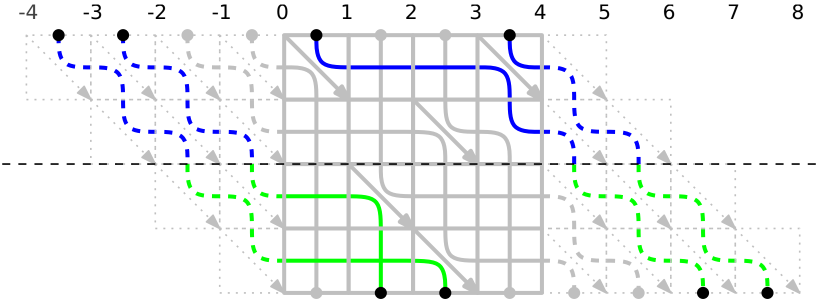
Algorithm, Tiskin'05

Given the distribution matrices d_A and d_B for two adjacent blocks of equal height M and width N in the grid dag, we can compute the distribution matrix d_C for the union of these blocks in $O(N^{1.5} + M)$.



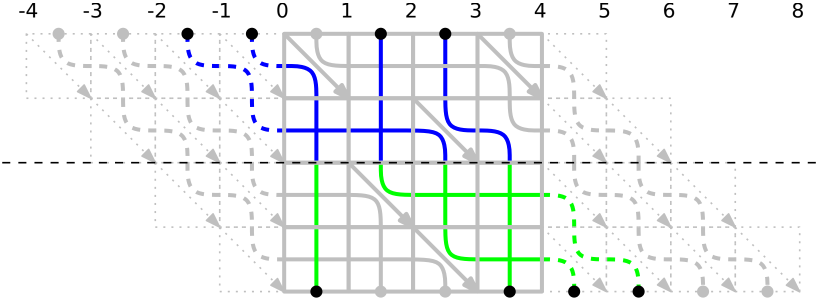
Score matrix multiplication: Two parts

“Trivial part” ($O(M)$):



Score matrix multiplication: Two parts

"Nontrivial part" ($O(N^{1.5})$):



Nontrivial part

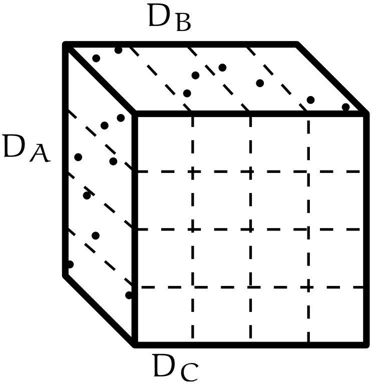
- Non-trivial part can be seen as $(\min, +)$ matrix product ($d_{A|B|C}$ are now the nontrivial parts of the corresponding distribution matrices):

$$d_C(i,k) = \min_j (d_A(i,j) + d_B(j,k))$$

- Explicit form, naive algorithm: $O(n^3)$
- Explicit form, algorithm that uses score matrix properties: $O(n^2)$
- Implicit form, divide-and-conquer: $O(n^{1.5})$

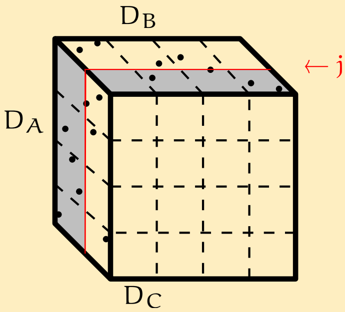
Divide-and-conquer multiplication

C-blocks and relevant nonzeros



Splitting a given set of relevant nonzeros in D_A and D_B into two sets at a position $j \in [0 : N]$, we get the number of relevant nonzeros in D_A up to column $j - \frac{1}{2}$, and the number of relevant nonzeros in D_B starting at row $j + \frac{1}{2}$:

Definition (δ-sequences)

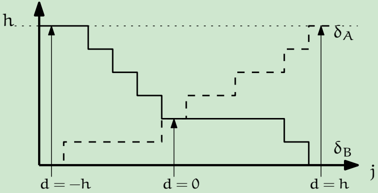


Definition (j-blocks)

Contiguous sets of j called "j-*blocks*", corresponding to a value of $d \in [-h : h]$, are defined as
$\mathcal{J}^{\square}(d) = \{j \mid \delta_A^{\square}(j) - \delta_B^{\square}(j) = d\}$.

- A j-block need not exist for every d.
- Small C-blocks $\Rightarrow$ few j-blocks, as the number of relevant nonzeros decreases with the block size.
- Can determine the j-blocks by a scan of the relevant nonzeros.

Example (j-blocks)



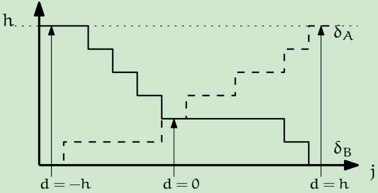
Definition (Δ-sequences)

δ's don't change inside a j-block $\Rightarrow$

$$\Delta_A^{\square}(d) = \operatorname*{any}_{j \in \mathcal{J}^{\square}(d)} \delta_B^{\square}(j)$$

$$\Delta_B^{\square}(d) = \operatorname*{any}_{j \in \mathcal{J}^{\square}(d)} \delta_B^{\square}(j)$$

Example (j-blocks)



Definition (Δ-sequences)

δ's don't change inside a j-block $\Rightarrow$

$$\Delta_A^{\square}(d) = \operatorname*{any}_{j \in \mathcal{J}^{\square}(d)} \delta_B^{\square}(j)$$
$$\Delta_B^{\square}(d) = \operatorname*{any}_{j \in \mathcal{J}^{\square}(d)} \delta_B^{\square}(j)$$

Definition (local minima)

The sequence

$$M^{\square}(d) = \min_{j \in \mathcal{J}^{\square}(d)} \left(d_A(i_0, j) + d_B(j, k_0)\right)$$

contains the minimum of $d_A(i_0, j) + d_B(j, k_0)$ in every j-block.

! We can have different values of M in one j-block.

We can use M's and Δ's to compute the number of nonzeros in a C-block.

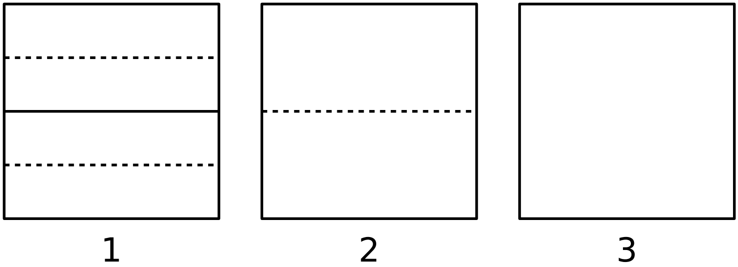
Sequences M for every C-subblock can be computed in $O(h)$:

$$
\begin{aligned}
M^{\text{◱}}(d') &= \min_{d} M^{\square}(d), \\
M^{\text{◲}}(d') &= \min_{d} M^{\square}(d) + \bar{\Delta}_{B}^{\text{◲}}(d), \\
M^{\text{◰}}(d') &= \min_{d} M^{\square}(d) + \bar{\Delta}_{A}^{\text{◰}}(d), \\
M^{\text{◳}}(d') &= \min_{d} M^{\square}(d) + \bar{\Delta}_{A}^{\text{◳}}(d) + \bar{\Delta}_{B}^{\text{◳}}(d)
\end{aligned}
$$

having $\bar{\Delta}_{A}^{(i',k',\frac{h}{2})}(d) - \bar{\Delta}_{B}^{(i',k',\frac{h}{2})}(d) = d'$ with $d' \in [-\frac{h}{2} : \frac{h}{2}]$.

- Sequences $\Delta_A(d')$ and $\Delta_B(d')$ can also be determined in $O(h)$ by a scan of the relevant nonzeros for each subblock.
- Knowing $\Delta_A(d')$, $\Delta_B(d')$ and $M(d')$ for each subblock, we can continue the recursion in every subblock.
- The recursion terminates when N C-blocks of size 1 are left.

We can use the described multiplication procedure to derive a parallel algorithm that uses $W = O(\frac{n^2}{p})$, $H = O(n \log p)$ and $S = O(\log p)$:



What can we improve?

- $O(N)$ data elements need to be transferred in every step of the merging tree.
- For very long strings, this is not desirable.
- To improve on the communication, parallelise the non-trivial step of the multiplication procedure.
- The goal is to decrease communication necessary for score-matrix multiplication to $O(N/\sqrt{p})$.

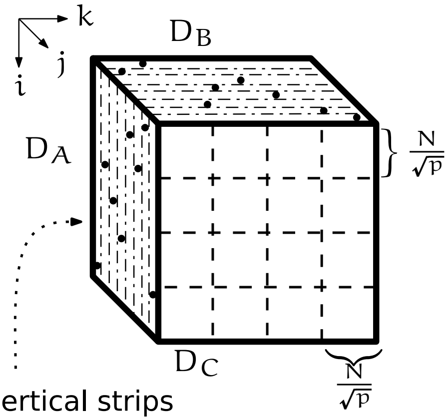
$\Rightarrow$ this achieves scalable communication

- Start the recursion at a point where there are p C-blocks.
- This is at level $\frac{1}{2}\log p$.
- Precompute and distribute the required sequences Δ and M for each C-block in parallel.
- Every C-block has size $h = \frac{N}{\sqrt{p}}$, and hence requires sequences with $O(\frac{N}{\sqrt{p}})$ values.
- After these sequences have been precomputed and redistributed, we can use the sequential algorithm to finish the computation.

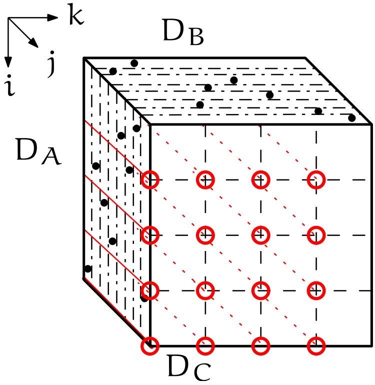
Assume that:

- $\sqrt{p}$ is an integer.
- Every processor has unique identifier q with $0 \leqslant q < p$.
- Every processor q corresponds to exactly one location $(q_x, q_y) \in [0 : \sqrt{p} - 1] \times [0 : \sqrt{p} - 1]$.
- Initial distribution of nonzeros in D_A and D_B is assumed to be even among all processors.

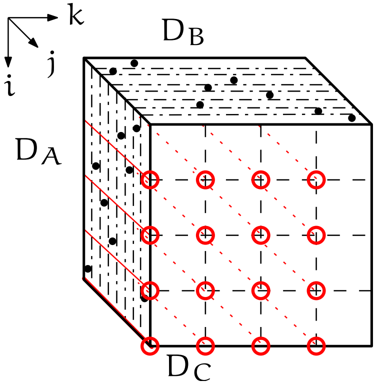
- Redistribute the nonzeros to strips of width $\frac{N}{p}$
- Send all nonzeros $(\hat{\imath}, \hat{\jmath})$ in D_A and $(\hat{\jmath}, \hat{k})$ in D_B to processor $\lfloor(\hat{\jmath} - \frac{1}{2}) \cdot p/N\rfloor$.
- Possible in one superstep using communication $O(\frac{N}{p})$.



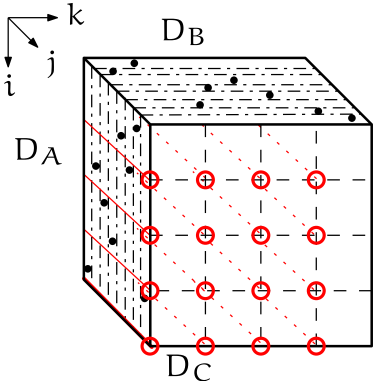
p vertical strips
$\frac{N}{p}$ nonzeros each



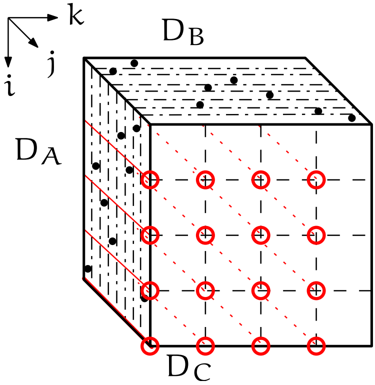
Compute the elementary $(\min, +)$ products $d_A(\circ_x, j) + d_B(j, \circ_y)$ along $j \in [0 : N]$.



Every processor holds all $D_A(\hat{\imath}, \hat{\jmath})$ and all $D_B(\hat{\jmath}, \hat{k})$ for $\hat{\jmath} \in \langle q \cdot \frac{N}{p} : (q+1) \cdot \frac{N}{p} \rangle$.

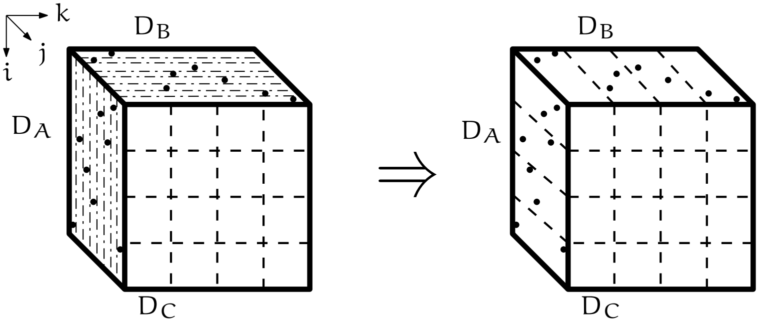


Can compute the values $d_A(\circ_x, j)$ and $d_B(j, \circ_y)$ by using parallel prefix/suffix.



After prefix and suffix computations, every processor holds N/p values $d_A(\circ_x, j) + d_B(j, \circ_y)$ for $j \in [q \cdot \frac{N}{\sqrt{p}} : (q+1) \cdot \frac{N}{\sqrt{p}}]$.

Redistribution Step



p vertical strips $\frac{N}{p}$ nonzeros each

$\sqrt{p}$ horizontal strips with $\frac{N}{\sqrt{p}}$ nonzeros each

Analysis

- Computational work bounded by the sequential recursion:
  $W = O((N/\sqrt{p})^{1.5}) = O(N^{1.5}/p^{0.75})$
- Every processor holds $O(N/p)$ nonzeros before redistribution.
- Every nonzero is relevant for $\sqrt{p}$ C-blocks.

$\Rightarrow$ $O(N/\sqrt{p})$ communication for redistributing the nonzeros.

$\Rightarrow$ $H = O(N/\sqrt{p} + p + N/\sqrt{p}) = O(N/\sqrt{p})$
(if $N/\sqrt{p} > p \quad \rightarrow N > p^{1.5}$)

- $S = O(1)$ (parallel prefix)

Analysis

- Computational work bounded by the sequential recursion:
  $W = O((N/\sqrt{p})^{1.5}) = O(N^{1.5}/p^{0.75})$
- Every processor holds $O(N/p)$ nonzeros before redistribution.
- Every nonzero is relevant for $\sqrt{p}$ C-blocks.

$\Rightarrow$ $O(N/\sqrt{p})$ communication for redistributing the nonzeros.

$\Rightarrow$ $H = O(N/\sqrt{p} + p + N/\sqrt{p}) = O(N/\sqrt{p})$
(if $N/\sqrt{p} > p \quad \rightarrow N > p^{1.5}$)

- $S = O(1)$ (parallel prefix)

Analysis

- Computational work bounded by the sequential recursion:
  $W = O((N/\sqrt{p})^{1.5}) = O(N^{1.5}/p^{0.75})$
- Every processor holds $O(N/p)$ nonzeros before redistribution.
- Every nonzero is relevant for $\sqrt{p}$ C-blocks.

$\Rightarrow$ $O(N/\sqrt{p})$ communication for redistributing the nonzeros.

$\Rightarrow$ $H = O(N/\sqrt{p} + p + N/\sqrt{p}) = O(N/\sqrt{p})$
(if $N/\sqrt{p} > p \quad \rightarrow N > p^{1.5}$)

- $S = O(1)$ (parallel prefix)

Analysis

- Computational work bounded by the sequential recursion:

  $W = O((N/\sqrt{p})^{1.5}) = O(N^{1.5}/p^{0.75})$
- Every processor holds $O(N/p)$ nonzeros before redistribution.
- Every nonzero is relevant for $\sqrt{p}$ C-blocks.

$\Rightarrow$ $O(N/\sqrt{p})$ communication for redistributing the nonzeros.

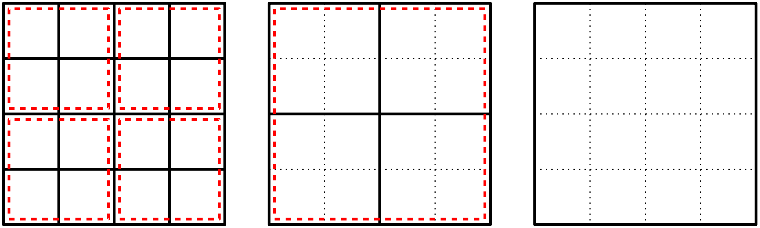
$\Rightarrow$ $H = O(N/\sqrt{p} + p + N/\sqrt{p}) = O(N/\sqrt{p})$

(if $N/\sqrt{p} > p \quad \rightarrow N > p^{1.5}$)

- $S = O(1)$ (parallel prefix)

Outline

- First, compute scores for a regular grid of p sub-dags of size $n/\sqrt{p} \times n/\sqrt{p}$
- Then merge these in a quadtree-like scheme using parallel score-matrix multiplication:



- Quadtree has $\frac{1}{2}\log_2 p$ levels
- On level l, $0 \leqslant l \leqslant \frac{1}{2}\log_2 p$, we have
  - $p_l = \frac{p}{4^l}$ (number of processors that work together on one merge)
  - $N_l = \frac{N}{2^l}$ (block size of merge)

$$\Rightarrow w_l = O\left(\frac{\left(\frac{N}{2^l}\right)^{1.5}}{\left(\frac{p}{4^l}\right)^{0.75}}\right) = O\left(\frac{N^{1.5}}{p^{0.75}}\right)$$

$$\Rightarrow h_l = O\left(\frac{\frac{N}{2^l}}{\left(\frac{p}{4^l}\right)^{0.5}}\right) = O\left(\frac{N}{p^{0.5}}\right)$$

- Quadtree has $\frac{1}{2}\log_2 p$ levels

  Hence, we get
- work $W = O(\frac{n^2}{p} + \frac{N^{1.5}\log p}{p^{0.75}}) = O(n^2/p)$ (assuming that $n \geqslant p^2$),
- communication $H = O(\frac{n \log p}{\sqrt{p}})$, and
- $S = O(\log p)$ supersteps.

Summary

This talk was about. . .

- . . . an introduction to parallel algorithms using BSP,
- . . . an overview of some semilocal string comparison algorithms,
- . . . a parallel algorithm for semilocal string comparison that is communication efficient and work-optimal, and is asymptotically better than even global LCS computation.

Algorithmic:

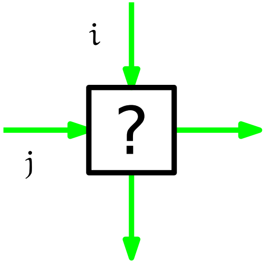
- Score matrix multiplication can also be applied to create a scalable algorithm for the iongest increasing subsequence problem.
- Adapt this algorithm for computing edit-distances.
- Study different problems from a BSP perspective.

Algorithm Engineering

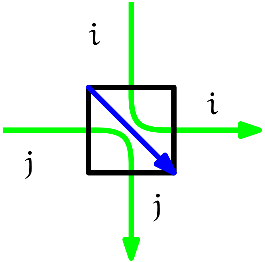
- Implement this algorithm using BSP-Tools, study load-balancing strategies
- Implement BSP tools that allow easier creation of hierarchical BSP algorithms like the one shown here
- Compare performance between “old-style” BSP on MPI and newer approaches, like skeletons and GASNet-based languages.
- Library of BSP algorithmic templates.

Thank you!
Any questions?

Seaweed-Algorithm on a single cell level:

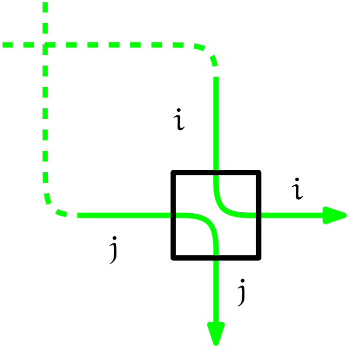


Seaweed-Algorithm on a single cell level:



We know what to do when there is a match...

Seaweed-Algorithm on a single cell level:



$i < j$ $\qquad$ $i > j$

No match $\Rightarrow$ check for double crossing...