New algorithms for efficient parallel string comparison

Peter Krusche Alexander Tiskin

Department of Computer Science and DIMAP
University of Warwick, Coventry, CV4 7AL, UK

SPAA 2010

THE UNIVERSITY OF WARWICK

DIMAP

In this talk. . .

. . . we show new parallel algorithms for computing string alignments and longest increasing subsequences.

Our new algorithms achieve scalable computation as well as scalable communication cost.

Talk Outline

I Parallel computation and scalability

II Longest increasing subsequences and longest common subsequences

III Our new algorithms

IV Summary and outlook

Outline

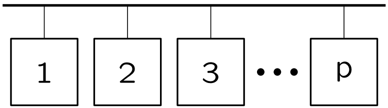
- Parallel Algorithms
  - Modelling parallel computation
  - Modelling parallel algorithms
- Longest Increasing Subsequences
  - Problem analysis
  - Previous parallel LIS algorithms
- Monge matrices
  - Monge matrices in string comparison
  - Distance multiplication
- Parallel distance multiplication
  - Sequential algorithm
  - Parallel algorithm
- Applications in string comparison

Modelling parallel computation

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Superstep-style program execution.

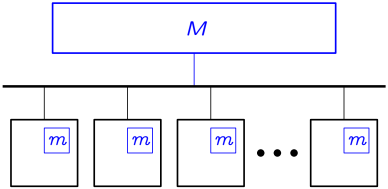


Modelling parallel computation

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Superstep-style program execution.

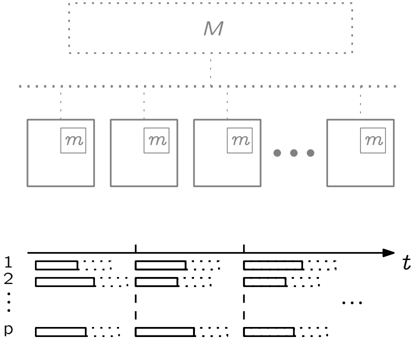


Modelling parallel computation

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Superstep-style program execution.



Sequential Algorithms

We have a problem of size n. We study

...the total work $\mathcal{W}(n)$

...the memory requirement $\mathcal{M}(n)$

...the input/output size: $\mathcal{I}(n)$

We assume that the input and output are stored in the environment (e.g. external memory).

Parallel Algorithms

Across all supersteps of the algorithm, we look at

...the computation time: $W(n, p)$

...the communication cost: $H(n, p)$

...the local memory cost: $M(n, p)$

How to do these costs relate to *scalability*?

Parallel Algorithms

Across all supersteps of the algorithm, we look at

...the computation time: $W(n, p)$

...the communication cost: $H(n, p)$

...the local memory cost: $M(n, p)$

How to do these costs relate to *scalability*?

Classical Criterion: Work Optimality

An algorithm is work-optimal (w.r.t. a sequential algorithm) if

$$W(n,p) = O\left(\frac{\mathcal{W}(n)}{p}\right).$$

We have absolute work-optimality if $\Omega(\mathcal{W}(n))$ is a lower bound on the total work for the given problem, and the given model.

Classical Criterion: Work Optimality

An algorithm is work-optimal (w.r.t. a sequential algorithm) if

$$W(n, p) = O\left(\frac{\mathcal{W}(n)}{p}\right).$$

We have absolute work-optimality if $\Omega(\mathcal{W}(n))$ is a lower bound on the total work for the given problem, and the given model.

Scalable Communication and Memory

Scalable communication:

An algorithm achieves asymptotically scalable communication if $H(n, p) = O(\mathcal{I}(n)/p^c)$

(assuming $0 < c$).

Scalable memory:

An algorithm achieves asymptotically scalable memory if $M(n, p) = O(\mathcal{M}(n)/p^c)$.

(assuming $0 < c$).

Scalable Communication and Memory

Scalable communication:

An algorithm achieves asymptotically scalable communication if $H(n, p) = O(\mathcal{I}(n)/p^c)$

(assuming $0 < c$).

Scalable memory:

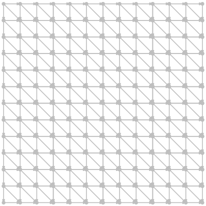
An algorithm achieves asymptotically scalable memory if $M(n, p) = O(\mathcal{M}(n)/p^c)$.

(assuming $0 < c$).

Example

Example (Grid dag dynamic programming)

Work-optimality is no problem.

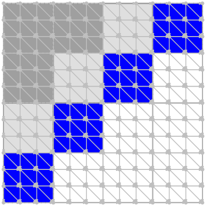


However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Example

Example (Grid dag dynamic programming)

Work-optimality is no problem.

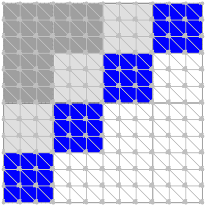


However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Example

Example (Grid dag dynamic programming)

Work-optimality is no problem.



However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Outline

Parallel Algorithms
- Modelling parallel computation
- Modelling parallel algorithms

Longest Increasing Subsequences
- Problem analysis
- Previous parallel LIS algorithms

Monge matrices
- Monge matrices in string comparison
- Distance multiplication

Parallel distance multiplication
- Sequential algorithm
- Parallel algorithm

Applications in string comparison

The Problem

Given a sequence of n numbers, to find the longest subsequence that is increasing.

`2, 9, 1, 3, 7, 5 , 6, 4, 8`

The Problem

Given a sequence of n numbers, to find the longest subsequence that is increasing.

2, 9, 1, **3**, 7, **5**, **6**, 4, **8**

The Problem

Given a sequence of n numbers, to find the longest subsequence that is increasing.

2, 9, **1**, **3**, 7, **5**, **6**, 4, **8**

(alternate solution)

Sequential LIS Algorithms

The LIS can be found by *patience sorting*.

(see [Knuth:73, Aldous/Diaconis:99, Schensted:61]).

Another approach: LIS via permutation string comparison.

(see [Hunt/Szymanski:77]).

For both algorithms, $\mathcal{W}(n) = O(n \log n)$ in the comparison-based model.

Permutation String Comparison

Definition (Input data)

Let $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_n$ be two permutation 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).

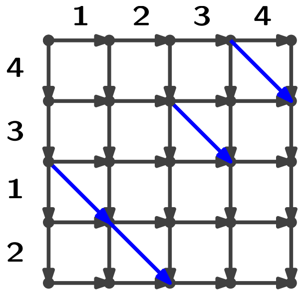
LIS via LCS

How to compute comparison-based LIS using LCS computation?

1. Copy the sequence and sort it.
2. Compute the LCS of the sequence and its sorted copy.

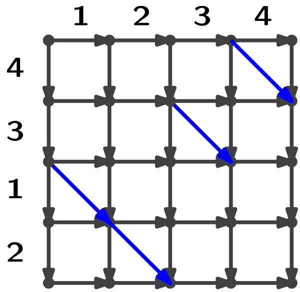
LCS grid dags and highest-score matrices

- The LCS Problem can be represented as longest path problem on a grid dag.
- In the LIS case, we have n diagonal edges of length 1.
- Horizontal edges have length 0.
- The LIS corresponds to a longest top-to-bottom path.



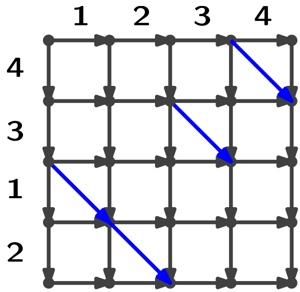
LCS grid dags and highest-score matrices

- The LCS Problem can be represented as longest path problem on a grid dag.
- In the LIS case, we have n diagonal edges of length 1.
- Horizontal edges have length 0.
- The LIS corresponds to a longest top-to-bottom path.



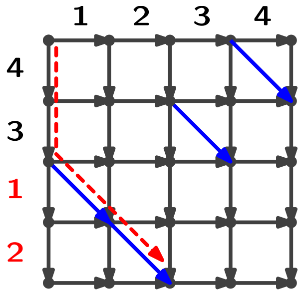
LCS grid dags and highest-score matrices

- The LCS Problem can be represented as longest path problem on a grid dag.
- In the LIS case, we have n diagonal edges of length 1.
- Horizontal edges have length 0.
- The LIS corresponds to a longest top-to-bottom path.



LCS grid dags and highest-score matrices

- The LCS Problem can be represented as longest path problem on a grid dag.
- In the LIS case, we have n diagonal edges of length 1.
- Horizontal edges have length 0.
- The LIS corresponds to a longest top-to-bottom path.



Parallel LIS Algorithms

Garcia, 2001

LIS by parallel dynamic programming.

$$W(n,p) = O(n^2/p)$$

This is not work optimal.

Parallel LIS Algorithms

Nakashima/Fujiwara, 2006

PRAM algorithm with

$$W(n,p) = O((n \log n)/p)$$

$$(\ldots \text{but only if } p < n/k^2)$$

Work-optimality is restricted:

Theorem (Erdős, 1935)

Every sequence of n integers has a monotonic subsequence of length $\geq \sqrt{n}$.

Parallel LIS Algorithms

Nakashima/Fujiwara, 2006

PRAM algorithm with

$$W(n, p) = O((n \log n)/p)$$

$$(\ldots \text{but only if } p < n/k^2)$$

Work-optimality is restricted:

Theorem (Erdős, 1935)

Every sequence of n integers has a monotonic subsequence of length $\geq \sqrt{n}$.

Parallel LIS Algorithms

Semé, 2006

BSP algorithm with

$$W(n, p) = O(n \log(n/p))$$

This is asymptotically sequential.

Our LIS algorithm

LIS computation for a sequence of length n:

$$W(n,p) = O\left(\frac{n\log^2 n}{p}\right)$$
$$H(n,p) = O\left(\frac{n\log p}{p}\right)$$
$$M(n,p) = O\left(\frac{n}{p}\right)$$

Outline

Our Tool: Semi-local Sequence Comparison

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 substring $y_i \dots y_j$ and x.

Definition (Semi-local LCS)

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

Why highest-score matrices?

Space efficiency, [Tiskin:05]

For strings x and y of lengths m and n, we can store highest-score matrix $A_{x,y}$ in $O(m+n)$ space.

Composition, [Tiskin:2009]

Consider three strings x, y, z of length n. Knowing $A_{x,z}$ and $A_{y,z}$, we can compute $A_{xy,z}$ (implicitly) in $O(n \log n)$ time.

How?

Highest-score matrices have a *Monge property*.

Why highest-score matrices?

Space efficiency, [Tiskin:05]

For strings x and y of lengths m and n, we can store highest-score matrix $A_{x,y}$ in $O(m+n)$ space.

Composition, [Tiskin:2009]

Consider three strings x, y, z of length n. Knowing $A_{x,z}$ and $A_{y,z}$, we can compute $A_{xy,z}$ (implicitly) in $O(n \log n)$ time.

How?

Highest-score matrices have a *Monge property*.

Why highest-score matrices?

Space efficiency, [Tiskin:05]

For strings x and y of lengths m and n, we can store highest-score matrix $A_{x,y}$ in $O(m+n)$ space.

Composition, [Tiskin:2009]

Consider three strings x, y, z of length n. Knowing $A_{x,z}$ and $A_{y,z}$, we can compute $A_{xy,z}$ (implicitly) in $O(n \log n)$ time.

How?

Highest-score matrices have a *Monge property*.

Monge matrices

Density matrix: **Distribution matrix:**

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = D^{\Sigma}(i+1,j) - D^{\Sigma}(i,j) - D^{\Sigma}(i+1,j+1) + D^{\Sigma}(i,j+1)$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Monge matrices

Density matrix:

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Distribution matrix:

$$D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = \boxed{D^{\Sigma}(i+1,j)} - D^{\Sigma}(i,j) - D^{\Sigma}(i+1,j+1) + \boxed{D^{\Sigma}(i,j+1)}$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Monge matrices

Density matrix:

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Distribution matrix:

$$D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = D^{\Sigma}(i+1,j) - \boxed{D^{\Sigma}(i,j)} - \boxed{D^{\Sigma}(i+1,j+1)} + D^{\Sigma}(i,j+1)$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Monge matrices

Density matrix: **Distribution matrix:**

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = D^{\Sigma}(i+1,j) - D^{\Sigma}(i,j) - D^{\Sigma}(i+1,j+1) + D^{\Sigma}(i,j+1)$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Monge matrices

Density matrix: **Distribution matrix:**

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = D^{\Sigma}(i+1,j) - D^{\Sigma}(i,j) - D^{\Sigma}(i+1,j+1) + D^{\Sigma}(i,j+1)$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Monge matrices

Density matrix: **Distribution matrix:**

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad D^{\Sigma} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D(i,j) = D^{\Sigma}(i+1,j) - D^{\Sigma}(i,j) - D^{\Sigma}(i+1,j+1) + D^{\Sigma}(i,j+1)$$

If $(D^{\Sigma})^{\square} = D$, we call D *simple*.

If D is non-negative, D^{Σ} is *Monge*.

If D is a permutation matrix, D^{Σ} is *unit-Monge*.

Distance multiplication

We compute the product

$$P_C^\Sigma = P_A^\Sigma \odot P_B^\Sigma$$

of two simple unit-Monge matrices P_A^Σ and P_B^Σ with

$$P_C^\Sigma(i,k) = \min_j (P_A^\Sigma(i,j) + P_B^\Sigma(j,k)).$$

Our inputs are the permutations corresponding to matrices P_A and P_B.

We output the permutation for P_C.

Outline

Parallel Algorithms
- Modelling parallel computation
- Modelling parallel algorithms

Longest Increasing Subsequences
- Problem analysis
- Previous parallel LIS algorithms

Monge matrices
- Monge matrices in string comparison
- Distance multiplication

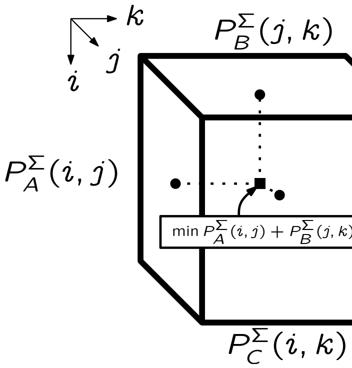
Parallel distance multiplication
- Sequential algorithm
- Parallel algorithm

Applications in string comparison

Sequential distance multiplication

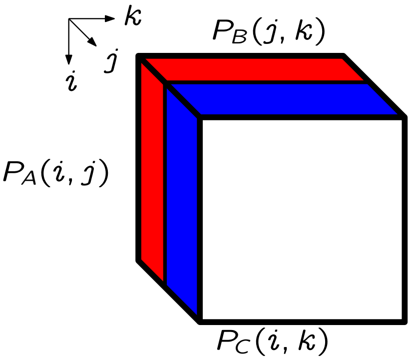
We parallelize the sequential algorithm from [Tiskin:09].

We start with the cube of elementary distance products.



Sequential distance multiplication

In each recursive step of this algorithm we split P_A and P_B into half-sized hi/lo ranges over j.

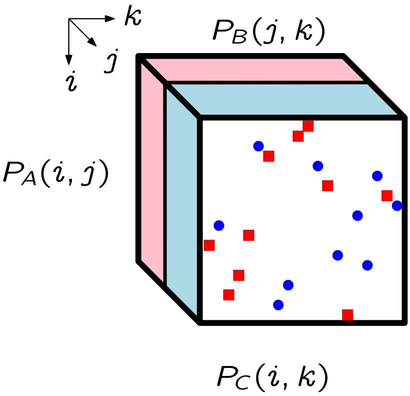


Sequential distance multiplication

The two products

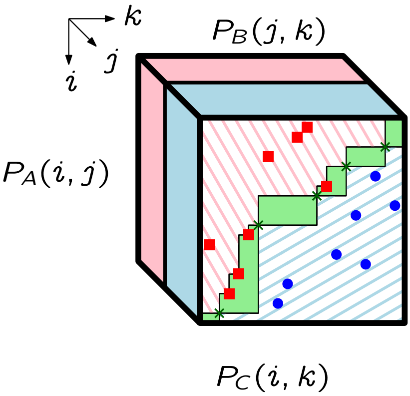
$$P^{\Sigma}_{A,lo} \odot P^{\Sigma}_{B,lo},$$
$$P^{\Sigma}_{A,hi} \odot P^{\Sigma}_{B,hi}$$

induce a permutation P'_C from which we can compute P_C.



Sequential distance multiplication

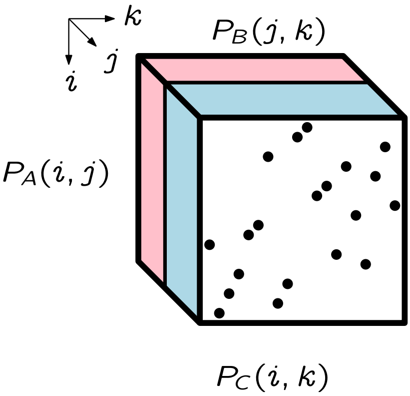
We compute the nonzeros in P_C from the nonzeros in P'_C using a linear-time sweep.



Sequential distance multiplication

$O(n)$ for the divide/conquer steps + two half-sized subproblems:

Overall time $O(n \log n)$.



Sequential distance multiplication

How to work out the nonzeros in P_C from the hi/lo products?

We have:

$$P_C^{\Sigma}(i,k) = \min(P_{C,lo}^{\Sigma}(i,k) + P_{C,hi}^{\Sigma}(0,k),\\ P_{C,hi}^{\Sigma}(i,k) + P_{C,lo}^{\Sigma}(i,n)\ ).$$

Sequential distance multiplication

How to work out the nonzeros in P_C from the hi/lo products?

Looking at the difference

$$\begin{aligned}\delta(i,k) &= (P^{\Sigma}_{C,lo}(i,k) + P^{\Sigma}_{C,hi}(0,k)) \\ &\quad -(P^{\Sigma}_{C,hi}(i,k) + P^{\Sigma}_{C,lo}(i,n)),\end{aligned}$$

we get

$$\begin{aligned}\delta(i,k) &= \textstyle\sum_{\hat{\imath}\in\langle 0:i\rangle,\hat{k}\in\langle 0:k\rangle} P_{C,hi}(\hat{\imath},\hat{k}) \\ &\quad - \textstyle\sum_{\hat{\imath}\in\langle i:n\rangle,\hat{k}\in\langle k:n\rangle} P_{C,lo}(\hat{\imath},\hat{k}).\end{aligned}$$

Sequential distance multiplication

How to work out the nonzeros in P_C from the hi/lo products?

The sign of δ tells us which nonzeros to use. We separate three areas in P_C:

$\texttt{Colour}(i,k) = \texttt{red}$ if $\delta(i,k) < 0$

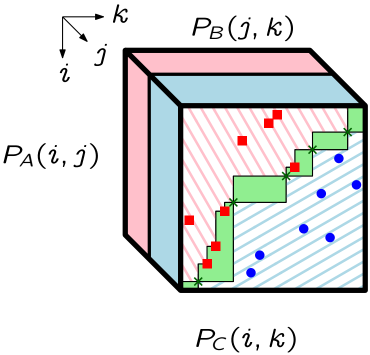
$\texttt{Colour}(i,k) = \texttt{green}$ if $\delta(i,k) = 0$

$\texttt{Colour}(i,k) = \texttt{blue}$ if $\delta(i,k) > 0$

Sequential distance multiplication

How to work out the nonzeros in P_C from the hi/lo products?

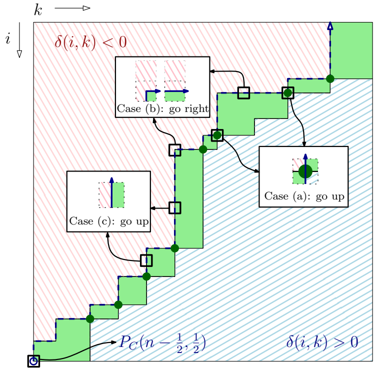
- blue areas: use nonzeros from $P_{C,hi}$
- red areas: use nonzeros from $P_{C,lo}$
- green areas: use nonzeros from $P_{C,hi}$ or $P_{C,lo}$, and "special" nonzeros at intersections.



Sequential distance multiplication

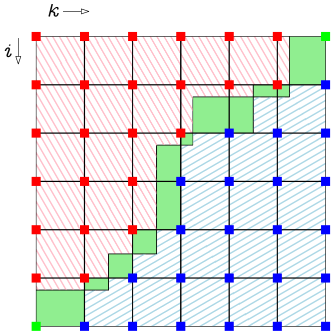
How to work out the nonzeros in P_C from the hi/lo products?

Colours can be computed incrementally ⇒ $O(n)$ time to trace boundary of green area.



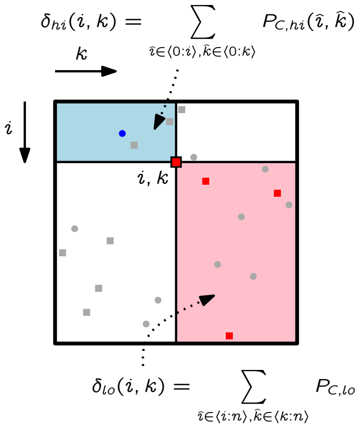
Parallel distance multiplication

We compute colours of points on a $p \times p$ grid.



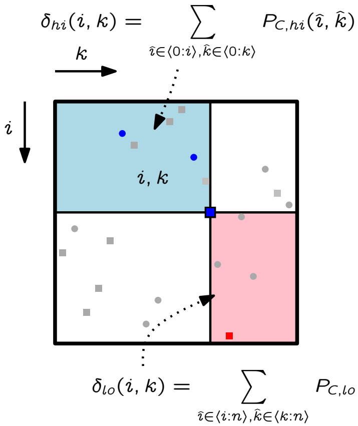
Parallel distance multiplication

$$\delta(i,k) = \delta_{hi}(i,k) - \delta_{lo}(i,k)$$



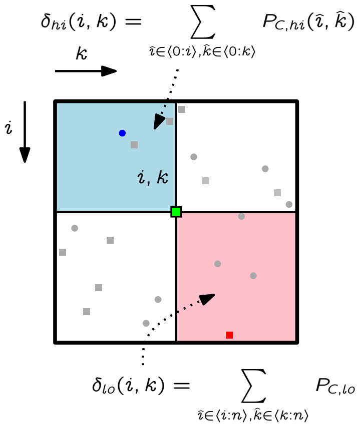
Parallel distance multiplication

$$\delta(i,k) = \delta_{hi}(i,k) - \delta_{lo}(i,k)$$



Parallel distance multiplication

$$\delta(i,k) = \delta_{hi}(i,k) - \delta_{lo}(i,k)$$

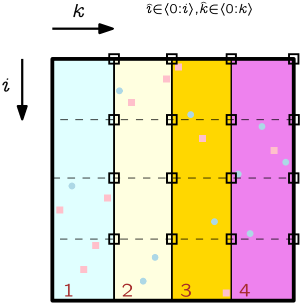


Parallel distance multiplication

Computing δ values on grid points by parallel prefix:

$$W(n,p) = O(n/p)$$
$$H(n,p) = O(p^2)$$
$$M(n,p) = O(n/p)$$
$$n > p^3$$

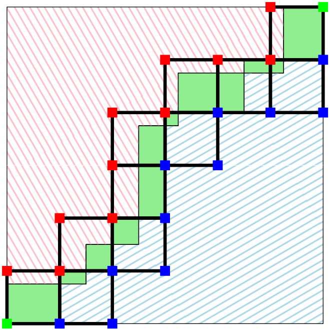
$$\delta_{hi}(i,k) = \sum_{\hat{\imath} \in \langle 0:i \rangle, \hat{k} \in \langle 0:k \rangle} P_{C,hi}(\hat{\imath}, \hat{k})$$



$$\delta_{lo}(i,k) = \sum_{\hat{\imath} \in \langle i:n \rangle, \hat{k} \in \langle k:n \rangle} P_{C,lo}$$

Parallel distance multiplication

Maximally $4p$ blocks can have a non-monochromatic set of corners.



Parallel distance multiplication

We already know the locations of the nonzeros in P_C for all monochromatic blocks.

For all i, k within the block, we have:...

- monochromatic blue blocks: $P_C(i,k) = P_{C,hi}(i,k)$
- monochromatic red blocks: $P_C(i,k) = P_{C,lo}(i,k)$
- monochromatic green blocks: $P_C(i,k) = 0$

In non-monochromatic blocks, we have to separate the green, blue and red areas in time $O(n/p)$ per block.

Parallel distance multiplication

We already know the locations of the nonzeros in P_C for all monochromatic blocks.

For all i, k within the block, we have:. . .

- monochromatic blue blocks: $P_C(i, k) = P_{C,hi}(i, k)$
- monochromatic red blocks: $P_C(i, k) = P_{C,lo}(i, k)$
- monochromatic green blocks: $P_C(i, k) = 0$

In non-monochromatic blocks, we have to separate the green, blue and red areas in time $O(n/p)$ per block.

Parallel distance multiplication

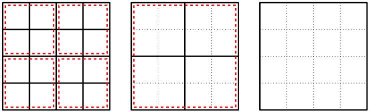
Given the nonzeros of two $n \times n$ permutation matrices P_A and P_B, distributed equally across $p < \sqrt[3]{n}$ processors, we can compute the nonzeros of a matrix P_C with $P_C^\Sigma = P_A^\Sigma \odot P_B^\Sigma$ using

$$
\begin{aligned}
W(n,p) &= O\left(\frac{n \log n}{p}\right) \\
H(n,p) &= O\left(\frac{n}{p} \log p\right) \\
M(n,p) &= O\left(\frac{n}{p}\right) \\
S &= O(\log p)
\end{aligned}
$$

Outline

LCS computation by distance multiplication

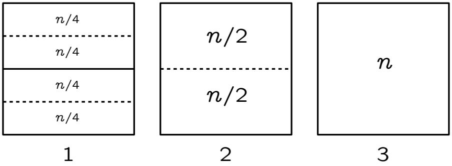
We use parallel distance multiplication in a quadtree merging scheme.



References	$W(n,p)$	$H(n,p)$	$M(n,p)$	S
McColl '95 + Wagner/Fischer+:74		$O(n)$	$O(\frac{n}{p})$	$O(p)$
McColl '95 + Alves et al. '06, Tiskin' 05	$O(\frac{n^2}{p})$	$O(n)$	$O(\frac{n}{p})$	$O(p)$
[KT:07]		$O\left(\frac{n \log p}{\sqrt{p}}\right)$	$O(\frac{n}{\sqrt{p}})$	$O(\log p)$
Shown here		$O\left(\frac{n}{\sqrt{p}}\right)$	$O(\frac{n}{\sqrt{p}})$	$O(\log^2 p)$

LIS computation by distance multiplication

We merge (horizontal) strips.



We get for $n > p^3$:

$$
\begin{aligned}
W(n,p) &= O\left(\frac{n\log^2 n}{p}\right)\\
H(n,p) &= O\left(\frac{n\log p}{p}\right)\\
M(n,p) &= O\left(\frac{n}{p}\right)\\
S &= O\left(\log^2 p\right)
\end{aligned}
$$

Summary and outlook

Summary

- We have shown new scalable algorithms for LCS/LIS computation.
- Our algorithms are scalable in communication and memory as well as computation.

Open questions

- How to achieve work-optimality for the LIS problem?
- Is $H(n, p) = O(n/\sqrt{p})$ a lower bound for LCS computation?

Thanks! Questions?

Questions?

Parallel Algorithms
- Modelling parallel computation
- Modelling parallel algorithms

Longest Increasing Subsequences
- Problem analysis
- Previous parallel LIS algorithms

Monge matrices
- Monge matrices in string comparison
- Distance multiplication

Parallel distance multiplication
- Sequential algorithm
- Parallel algorithm

Applications in string comparison