Parallel Longest Increasing Subsequences in Scalable Time and Memory

Peter Krusche   Alexander Tiskin

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

PPAM 2009

What is in this talk?

Some discussion of asymptotic scalability measures for parallel algorithms.

A simple algorithmic problem that is hard to parallelize scalably...

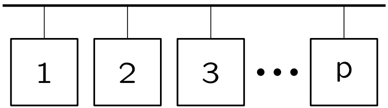
...and a scalable algorithm for it.

Outline

- Parallel Algorithms
  - The BSP Model
  - Asymptotic Scalability
- Longest Increasing Subsequences
  - The Problem
  - Sequential LIS
  - Permutation String Comparison
- Parallel LIS computation
  - Previous Work
  - Our Algorithm
- Summary and Outlook

Machine Model Ingredients

A BSP computer with p processors/ cores/threads.



External and per-processor memory.

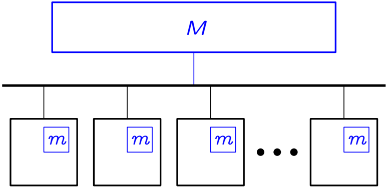
Superstep-style code execution.

Machine Model Ingredients

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Superstep-style code execution.

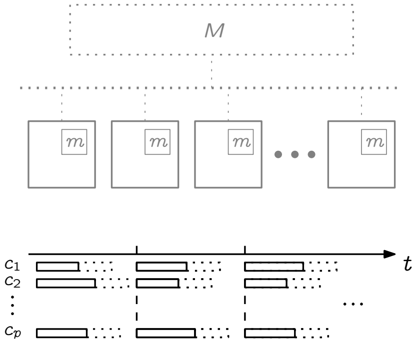


Machine Model Ingredients

A BSP computer with p processors/cores/threads.

External and per-processor memory.

Superstep-style code execution.



Algorithm Ingredients: 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).

Algorithm Ingredients: 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?

Algorithm Ingredients: 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

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

Example (Matrix Multiplication)

Algorithms achieving $W(n,p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

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

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

Example (Matrix Multiplication)

Algorithms achieving $W(n, p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

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

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

Example (Matrix Multiplication)

Algorithms achieving $W(n, p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

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 $0 < c \leq 1$).

Scalable memory:

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

(assuming $0 < c \leq 1$).

Scalable Communication and Memory

Scalable communication:

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

(assuming $0 < c \leq 1$).

Scalable memory:

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

(assuming $0 < c \leq 1$).

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on more complex parallel machine models.

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on more complex parallel machine models.

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on more complex parallel machine models.

Outline

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 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).

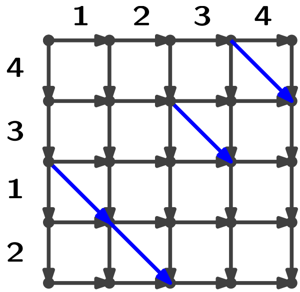
LIS via LCS

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.

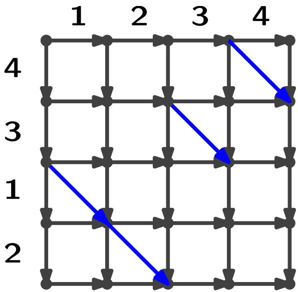
LCS grid dags and highest-score matrices

- 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.



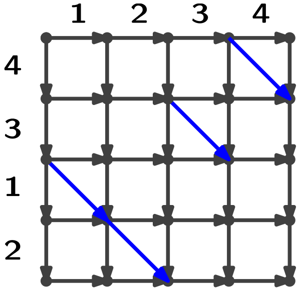
LCS grid dags and highest-score matrices

- 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.



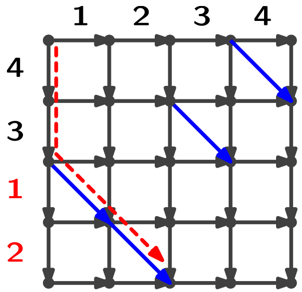
LCS grid dags and highest-score matrices

- 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.



Outline

- Parallel Algorithms
  - The BSP Model
  - Asymptotic Scalability
- Longest Increasing Subsequences
  - The Problem
  - Sequential LIS
  - Permutation String Comparison
- Parallel LIS computation
  - Previous Work
  - Our Algorithm
- Summary and Outlook

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

Semé, 2006

BSP algorithm with

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

This is asymptotically sequential.

Our Algorithm

We have a BSP algorithm for the LIS problem with

$$W(n,p) = \frac{n^{1.5}}{p}$$
$$H(n,p) = \frac{n}{\sqrt{p}}$$
$$M(n,p) = \frac{n}{\sqrt{p}}$$

(. . . which, in fact, can solve a slightly more general problem than LIS)

Our Tool: Semi-local Sequence Comparison

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 $y_i \ldots y_j$ and x.

Definition (Semi-local LCS)

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

Critical points

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

1. *For permutation string inputs of length n, $N = 2n$ such critical points are sufficient to implicitly represent the whole matrix [Schmidt:95/Alves+:06/Tiskin:05].*
2. *There is an algorithm to obtain these points in time $O(n^{1.5})$ [Tiskin:06].*

Critical points

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

1. *For permutation string inputs of length n, $N = 2n$ such critical points are sufficient to implicitly represent the whole matrix [Schmidt:95/Alves+:06/Tiskin:05].*
2. *There is an algorithm to obtain these points in time $O(n^{1.5})$ [Tiskin:06].*

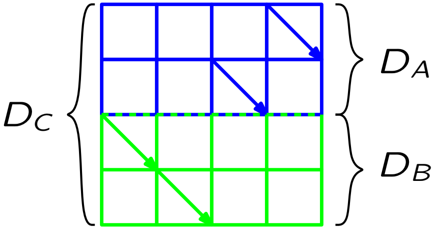
Highest-score matrices

Example (Highest-score matrix for $x =$4312 and $y =$1234)

	-1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
-5	4	4	4	4	4	4	4	4	4	4
-4	3	4	4	4	4	4	4	4	4	4
-3	2	3	3	4	4	4	4	4	4	4
-2	1	2	3	4	4	4	4	4	4	4
-1	0	1	2	3	3	3	3	4	4	4
0	-1	0	1	2	2	2	2	3	4	4
1	-2	-1	0	1	1	1	2	3	4	4
2	-3	-2	-1	0	1	1	2	3	4	4
3	-4	-3	-2	-1	0	1	2	3	4	4
4	-5	-4	-3	-2	-1	0	1	2	3	4

Highest-score Matrix Multiplication

Given the implicit highest-score matrices D_A and D_B for two adjacent grid dag blocks, we compute the distribution matrix d_C for the union of these blocks.



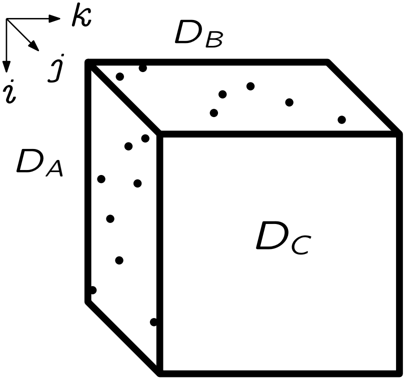
Highest-score Matrix Multiplication

We need to compute a $(\min, +)$ matrix product

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

For two implicit highest-score matrices of size N, we can compute this product in $O(N^{1.5})$ time.

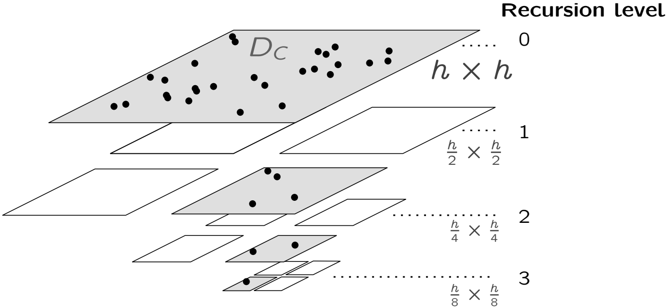
Matrix Multiplication as a Cube



Cube contains all elementary products
$\min d_A(i, j) + d_B(j, k)$.

Divide-and-conquer Multiplication

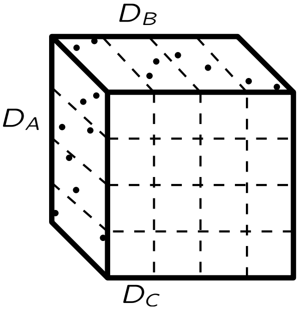
Recursive Partitioning



We locate nonzeros by recursively partitioning D_C into blocks.

Divide-and-conquer Multiplication

Relevant Nonzeros



For each block in D_C, only a subset of nonzeros in D_A and D_B are relevant.

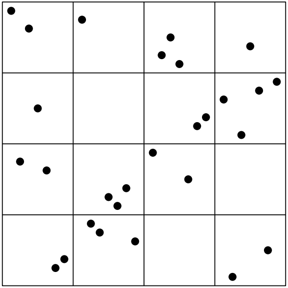
Divide-and-conquer Multiplication

- The smaller the blocks, the less nonzeros are relevant!
- All data for processing a block of size h can be stored in space $O(h)$.
- We can partition and compute the number of nonzeros in a block in time $O(h)$.

$\Rightarrow$ We get a divide-and-conquer algorithm for multiplying highest-score matrices of size N running in $O(N^{1.5})$.

Parallel Multiplication

We first partition D_C into p blocks and then finish the computation on these blocks independently [K,Tiskin:06].



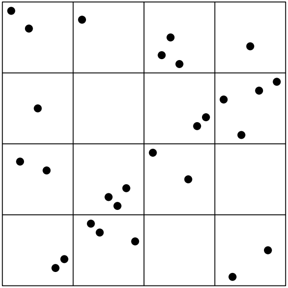
D_C

$p = 16$, $N = 49$

Problem: Nonzeros might be concentrated in $\sqrt{p}$ blocks. Then, we get $W(n, p) = O(N^{1.5}/\sqrt{p})$.

Parallel Multiplication

We first partition D_C into p blocks and then finish the computation on these blocks independently [K,Tiskin:06].



D_C

$p = 16$, $N = 49$

Problem: Nonzeros might be concentrated in $\sqrt{p}$ blocks. Then, we get $W(n, p) = O(N^{1.5}/\sqrt{p})$.

Parallel Multiplication

To apply this algorithm to the LIS problem, we need to $(\min, +)$-multiply work-optimally!

Work-optimal Multiplication

Lemma

We can locate a set of k nonzeros in a block of size h in time $O(h\sqrt{k})$.

Load-balancing

- Processors locate groups of maximally N/p nonzeros.
- We have maximally p complete groups (N nonzeros overall).
- We have maximally one incomplete group on each processor.

Work-optimal Multiplication

Lemma

We can locate a set of k nonzeros in a block of size h in time $O(h\sqrt{k})$.

Load-balancing

- Processors locate groups of maximally N/p nonzeros.
- We have maximally p complete groups (N nonzeros overall).
- We have maximally one incomplete group on each processor.

Work-optimal Multiplication

Lemma

We can locate all nonzeros using $W(N,p) = O(N^{1.5}/p)$.

Proof.

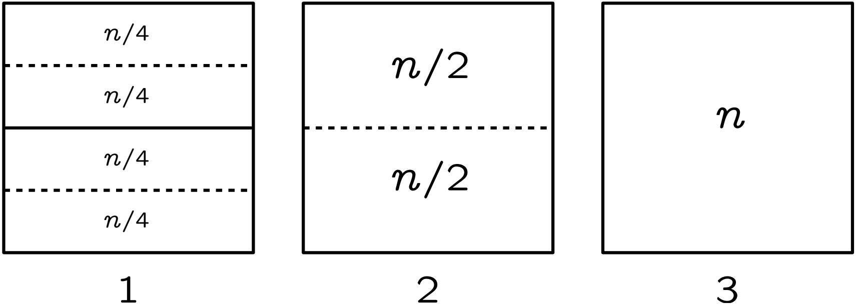
On each processor we locate $O(N/p)$ nonzeros a block of size $N/\sqrt{p}$ for maximally one complete and one incomplete group. We get

$$W(N,p) = O\left(N/\sqrt{p} \cdot \sqrt{N/p}\right) = O(N^{1.5}/p)$$

$\square$

LIS using parallel (min, +)-multiplication

We recursively merge highest score matrices in parallel.



Scalable Memory

Lemma

Our multiplication procedure requires $M(n, p) = O(n/\sqrt{p})$.

Proof ideas.

Lower bound: our blocks require an input of size $O(n/\sqrt{p})$ for the recursive partitioning.

Upper bound: We use the fact that the input strings are permutations to bound the number of matches on each processor as $O(N/\sqrt{p})$.

Matches can be distributed in a scalable fashion using e.g. sorting by regular sampling. $\square$

Discussion of Practicality

Speedup over the sequential algorithm is possible if input is distributed equally between processors.

- This is interesting if we have big records to compare – in the comparison-based model, our sequence elements do not have to be numbers.
- Speedup for large sequences is limited due to asymptotically very fast sequential algorithm.

Outline

Summary and Outlook

- We have shown a scalable approach to computing longest increasing subsequences.

- We are currently working on an improvement to the parallel multiplication algorithm:

$\Rightarrow$ We aim to reduce the work for multiplication to $W(n, p) = O((n \log n)/p)$.

This will bring us a step closer to achieving general work-optimality in a scalable fashion.

Thanks for listening!

Questions?

Parallel Algorithms
- The BSP Model
- Asymptotic Scalability

Longest Increasing Subsequences
- The Problem
- Sequential LIS
- Permutation String Comparison

Parallel LIS computation
- Previous Work
- Our Algorithm

Summary and Outlook