Parallel Prefix Algorithms for Registration of Arbitrarly Long Electron Micrograph Series

Marcin Copik

Master's Colloquuium
June 14th, 2017

Outline

Acquisition technique

- **transmission electron microscopy** (TEM)
- an electron beam passes through a sample
- the interaction of electrons with the specimen is mapped to a display device
- the quality of data is limited by changes in the sample induced by electrons

Data sample

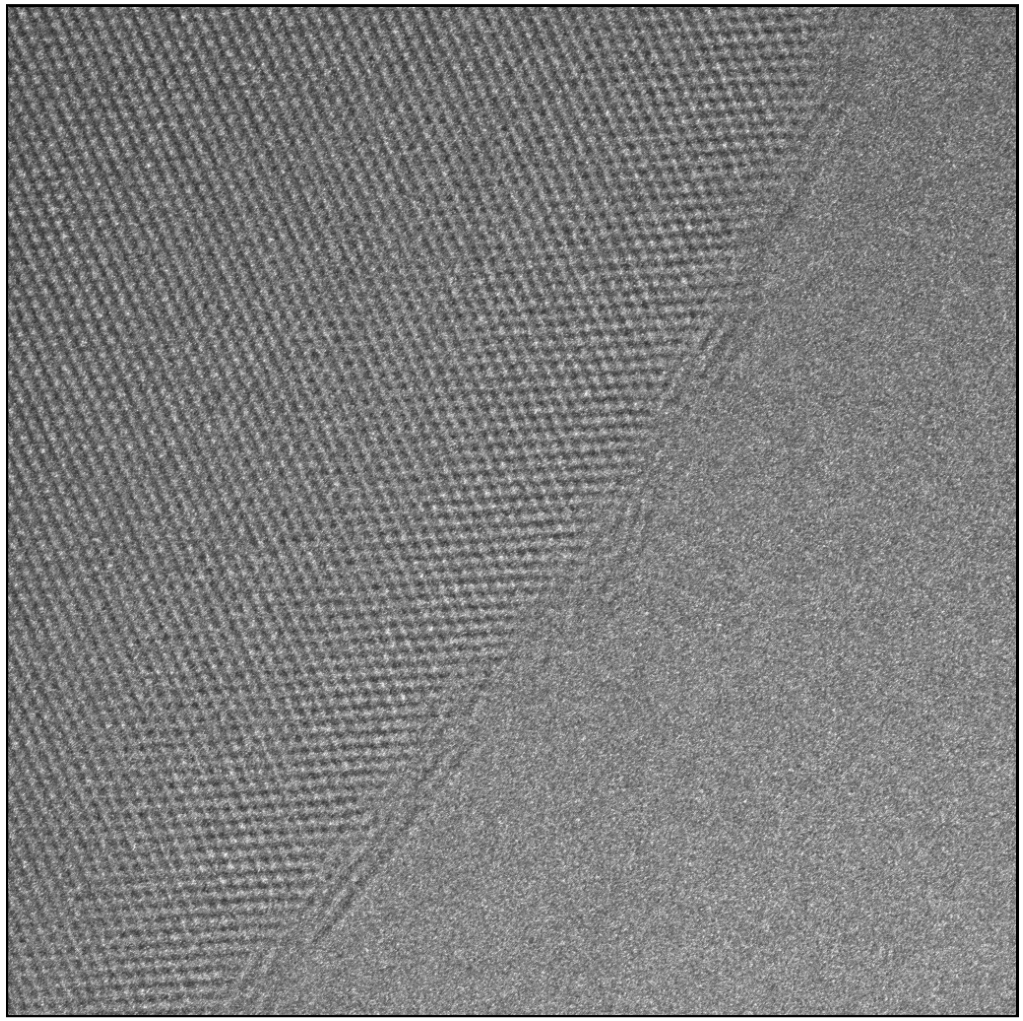


Figure: An example of frame acquired with TEM.

All images courtesy of the University of Manchester.

Improved acqusition

First step: replace a single high–dose frame with a series of low–dose frames

- decreases the risk of beam damage
- captures motion of the observed object
- captured frames may have a low *signal–to–noise* ratio
- data analysis obstructed by *sample drift* during the acquisition

Improved acqusition

First step: replace a single high–dose frame with a series of low–dose frames

- decreases the risk of beam damage
- captures motion of the observed object
- captured frames may have a low *signal–to–noise* ratio
- data analysis obstructed by *sample drift* during the acquisition

Second step: image registration for a series of frames

- aligning frames to the first one removes the sample motion
- aligned frames can be averaged to represent the sample in one frame
- quality of results depends on the image processing
- image registration can be computationally expensive - **can we parallelize that?**

Data sample

Example: aluminum oxidation

- acquisition performed with an *ultrahigh vacuum high-resolution transmission electron microscopy* (UHV HRTEM)
- capture 400 frames per second
- 2.7 GB of data per a second of acquisition

Image registration problem

Input: images $\mathcal{R}$, $\mathcal{T}$

Image registration problem

Input: images $\mathcal{R}$, $\mathcal{T}$

Goal: find a deformation ϕ such that

$$\mathcal{R} \approx \mathcal{T} \circ \phi$$

Image registration problem

Input: images $\mathcal{R}$, $\mathcal{T}$

Goal: find a deformation ϕ such that

$$\mathcal{R} \approx \mathcal{T} \circ \phi$$

which means

given a dissimilarity measure $\mathcal{M}$

$$\mathcal{M}[\mathcal{R}, \mathcal{T} \circ \phi] \rightarrow \mathit{min}$$

Registration method

Gradient–based multilevel registration

- find a rigid deformation

$$\phi(x) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} t_0 \\ t_1 \end{pmatrix}$$

- solve the registration problem for multiple image sizes, and at each level use a standard gradient descent optimization
- small image features are not preserved on lower levels, decreasing the likelihood of solver converging to a local minimum
- the number of levels is known a priori, but the total number of iterations is unknown because of a gradient descent solver
- as far as we are aware, a method with constant execution time does not exist for non–convex problems

Series registration

Given a series of images

$$f_0, f_1, f_2, \ldots, f_n$$

Series registration

Given a series of images

$$f_0, f_1, f_2, \ldots, f_n$$

How to register two neighboring frames?

A registration method introduced before

$$\phi_{i,i+1} = \mathrm{A}(f_i,\ f_{i+1})$$

Series registration

Given a series of images

$$f_0, f_1, f_2, \ldots, f_n$$

How to register two neighboring frames?

A registration method introduced before

$$\phi_{i,i+1} = \mathrm{A}(f_i, f_{i+1})$$

How to register two non-neighboring frames?

$$\phi_{0,2}, \phi_{0,3}, \ldots, \phi_{0,n}$$

Series registration

Given a series of images

$$f_0, f_1, f_2, \ldots, f_n$$

How to register two neighboring frames?

A registration method introduced before

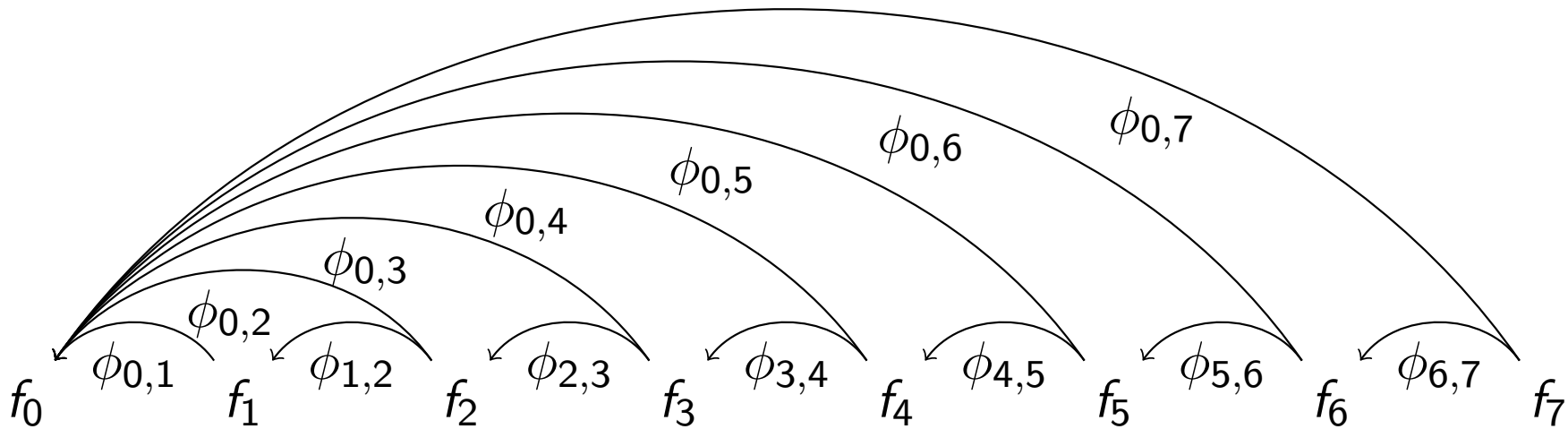
$$\phi_{i,i+1} = \mathrm{A}(f_i,\ f_{i+1})$$

How to register two non-neighboring frames?

$$\phi_{0,2}, \phi_{0,3}, \ldots, \phi_{0,n}$$

$$\phi_{i,k} = \mathrm{B}(\phi_{i,j},\ \phi_{j,k})$$

Series registration



Series registration

Full registration for a series

Let's define a new operator $\odot_B$

$$\phi_{i,j} \odot_B \phi_{j,k} = \mathrm{B}(\phi_{i,j},\ \phi_{j,k})$$

Then, the registration becomes

$$\begin{aligned}
\phi_{0,1} &= \phi_{0,1} \\
\phi_{0,2} &= \phi_{0,1} \odot_B \phi_{1,2} \\
\phi_{0,3} &= \phi_{0,1} \odot_B \phi_{1,2} \odot_B \phi_{2,3} \\
&\dots \\
\phi_{0,n} &= \phi_{0,1} \odot_B \phi_{1,2} \odot_B \cdots \odot_B \phi_{n-1,n}
\end{aligned}$$

For input deformation $\phi_{i,i+1}$, apply $\odot_B$ i times to first i input deformations.

Series registration

Full registration for a series

Let's define a new operator $\odot_B$

$$\phi_{i,j} \odot_B \phi_{j,k} = \mathrm{B}(\phi_{i,j},\ \phi_{j,k})$$

Then, the registration becomes

$$\begin{aligned}
\phi_{0,1} &= \phi_{0,1} \\
\phi_{0,2} &= \phi_{0,1} \odot_B \phi_{1,2} \\
\phi_{0,3} &= \phi_{0,1} \odot_B \phi_{1,2} \odot_B \phi_{2,3} \\
&\dots \\
\phi_{0,n} &= \phi_{0,1} \odot_B \phi_{1,2} \odot_B \cdots \odot_B \phi_{n-1,n}
\end{aligned}$$

For input deformation $\phi_{i,i+1}$, apply $\odot_B$ i times to first i input deformations.

A **prefix sum** of a sequence $\phi_{0,1}, \phi_{1,2}, \dots, \phi_{n-1,n}$

Series registration

A parallel approach

- *preprocessing step* transforms a sequence of images

$$f_0, f_1, \ldots, f_n$$

into a sequence of neighbor deformations

$$\phi_{0,1}, \phi_{1,2}, \ldots, \phi_{n-1,n}$$

- a prefix sum registers each frame to the first image, producing a new sequence

$$\phi_{0,1}, \phi_{0,2}, \ldots, \phi_{01,n}$$

Series registration

A parallel approach

- *preprocessing step* transforms a sequence of images

$$f_0, f_1, \ldots, f_n$$

into a sequence of neighbor deformations

$$\phi_{0,1}, \phi_{1,2}, \ldots, \phi_{n-1,n}$$

trivially parallelizable

- a prefix sum registers each frame to the first image, producing a new sequence

$$\phi_{0,1}, \phi_{0,2}, \ldots, \phi_{01,n}$$

Series registration

A parallel approach

- *preprocessing step* transforms a sequence of images

$$f_0, f_1, \ldots, f_n$$

into a sequence of neighbor deformations

$$\phi_{0,1}, \phi_{1,2}, \ldots, \phi_{n-1,n}$$

trivially parallelizable

- a prefix sum registers each frame to the first image, producing a new sequence

$$\phi_{0,1}, \phi_{0,2}, \ldots, \phi_{01,n}$$

a parallel prefix sum is necessary

Outline

Prefix sum

Inclusive prefix sum

$$y_i = x_0 \odot x_1 \odot \cdots \odot x_{i-1} \odot x_i$$

- also known under names scan, cumulative scan, partial sum
- heavily researched for parallel prefix adders in electronics
- a basic parallel programming pattern
- dozens of applications: sorting, image processing, tridiagonal solvers, graph and tree algorithms etc.
- an active research area on GPUs
- multiple implementations: C++/C++17, MPI_Scan, IntelTBB, Boost.Compute

Prefix sum

Inclusive prefix sum

$y_i = x_0 \odot x_1 \odot \cdots \odot x_{i-1} \odot x_i$

Exclusive prefix sum

$y_i = x_0 \odot x_1 \odot \cdots \odot x_{i-1}$

Inclusive $\rightarrow$ Exclusive

Shift elements by one position to the right, insert identity element at the beginning

Exclusive $\rightarrow$ Inclusive

Shift elements by one position to the left, compute last position

Parallel Prefix Sum

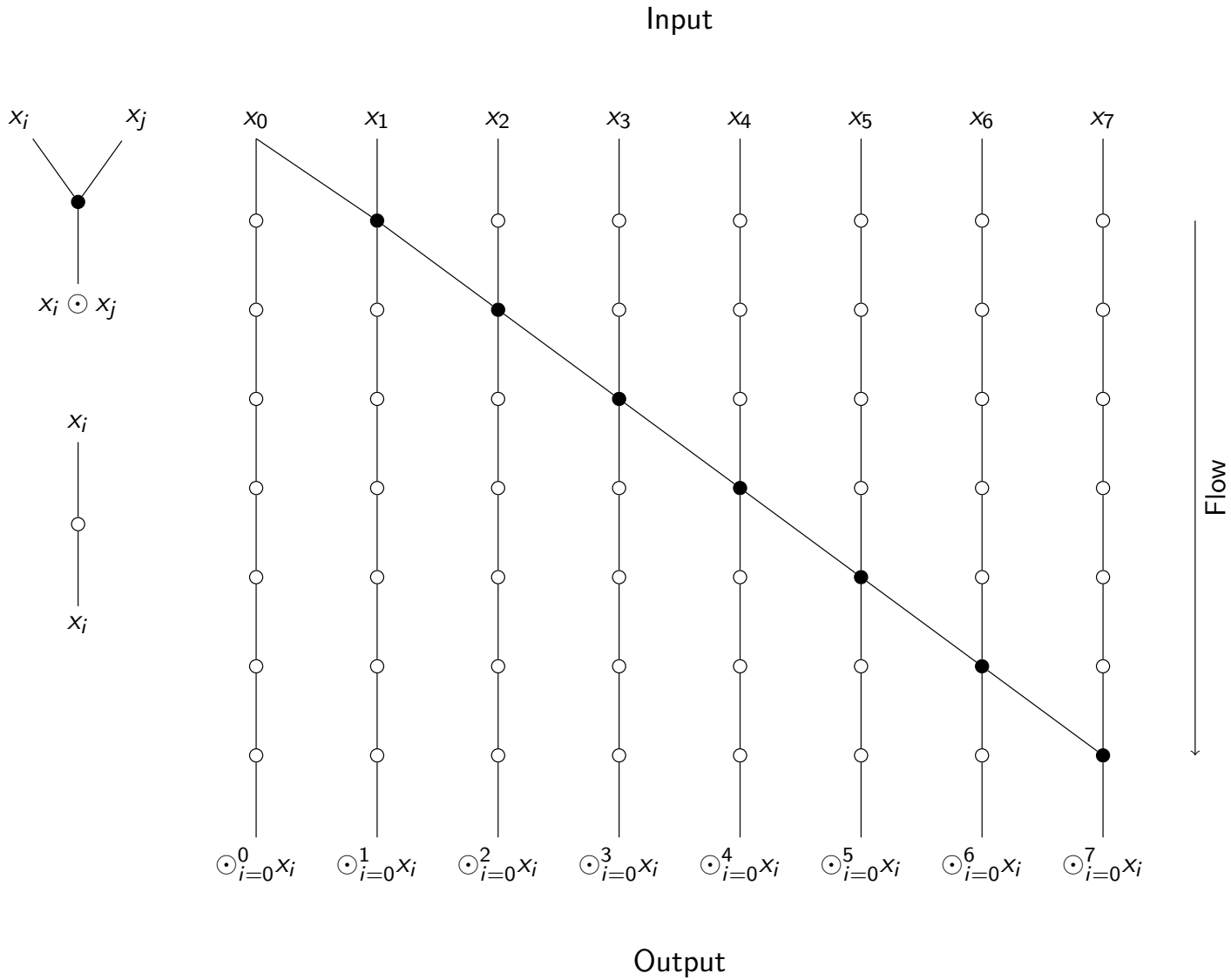
Parallel prefix algorithms

- Sklansky, 1960
- Kogge–Stone, 1973
- Brent–Kung, 1980
- Blelloch, 1989
- multiple other variants and combinations

Properties

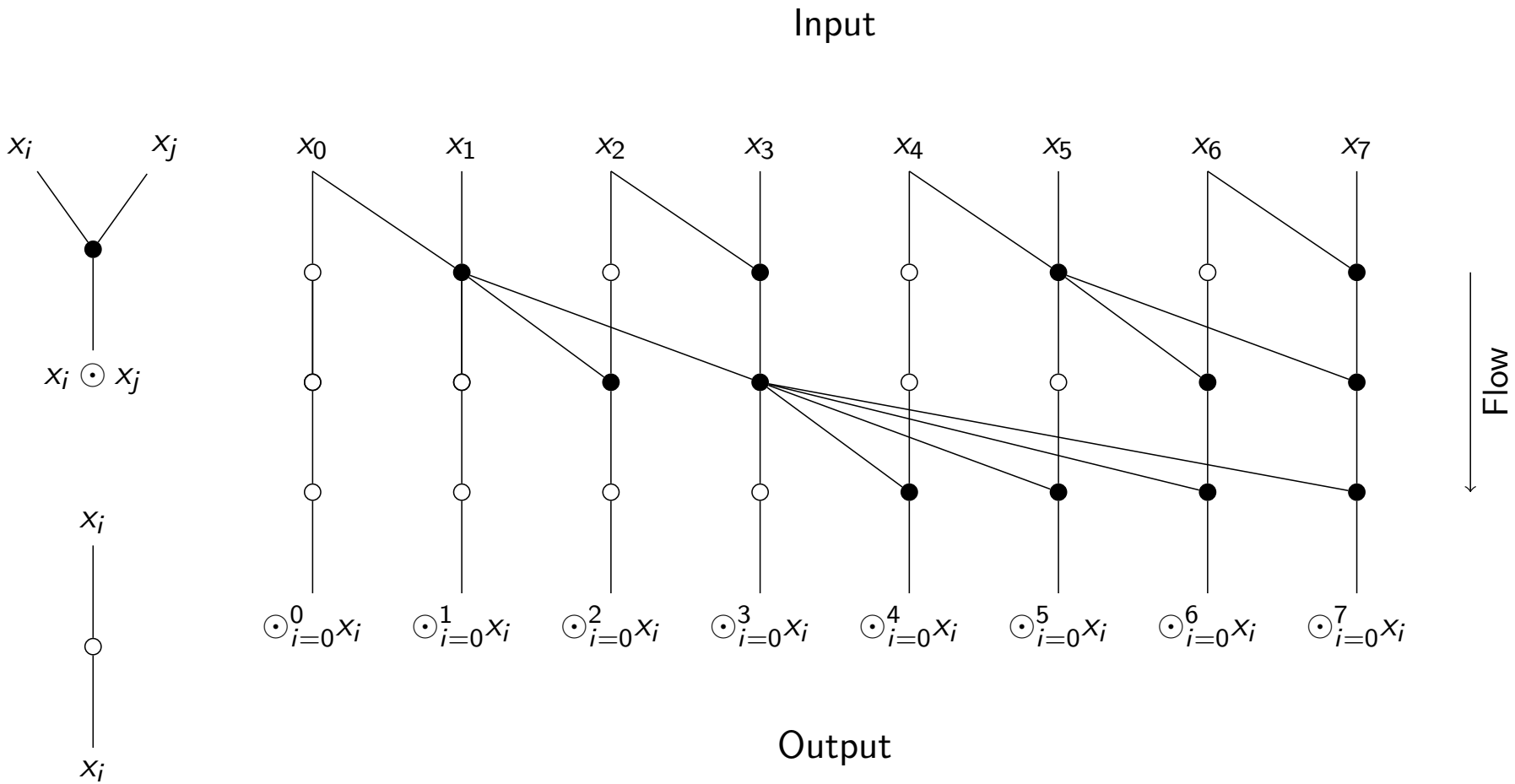
- **span**, depth - the length of a critical path in the algorithm
- **work**, size - how many times is the operator applied?

Serial


Input
x_i
x_j
$x_i \odot x_j$
x_i
x_i
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
Flow
$\odot_{i=0}^{0} x_i$
$\odot_{i=0}^{1} x_i$
$\odot_{i=0}^{2} x_i$
$\odot_{i=0}^{3} x_i$
$\odot_{i=0}^{4} x_i$
$\odot_{i=0}^{5} x_i$
$\odot_{i=0}^{6} x_i$
$\odot_{i=0}^{7} x_i$
Output

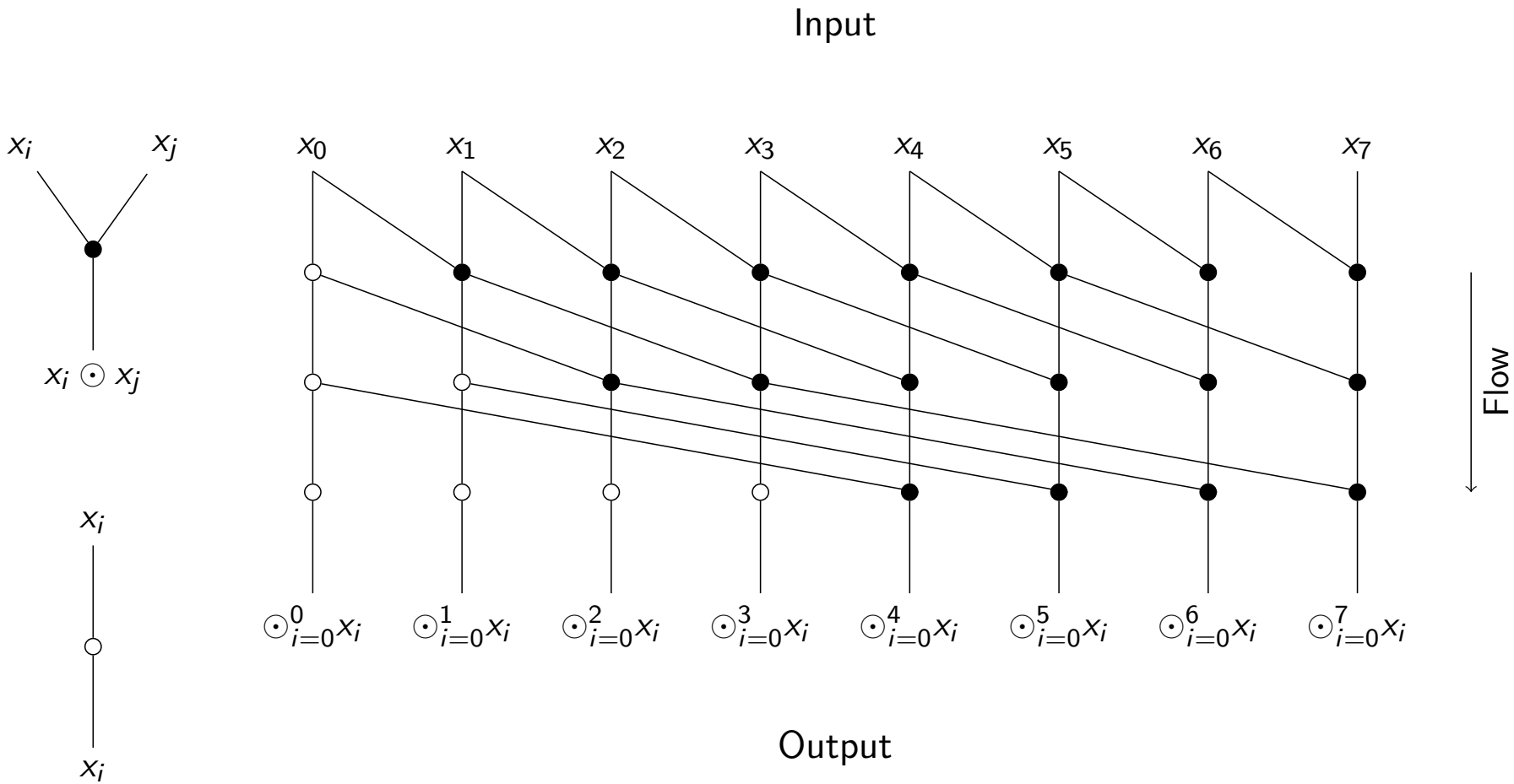
$$S(N) = N - 1$$

Sklansky



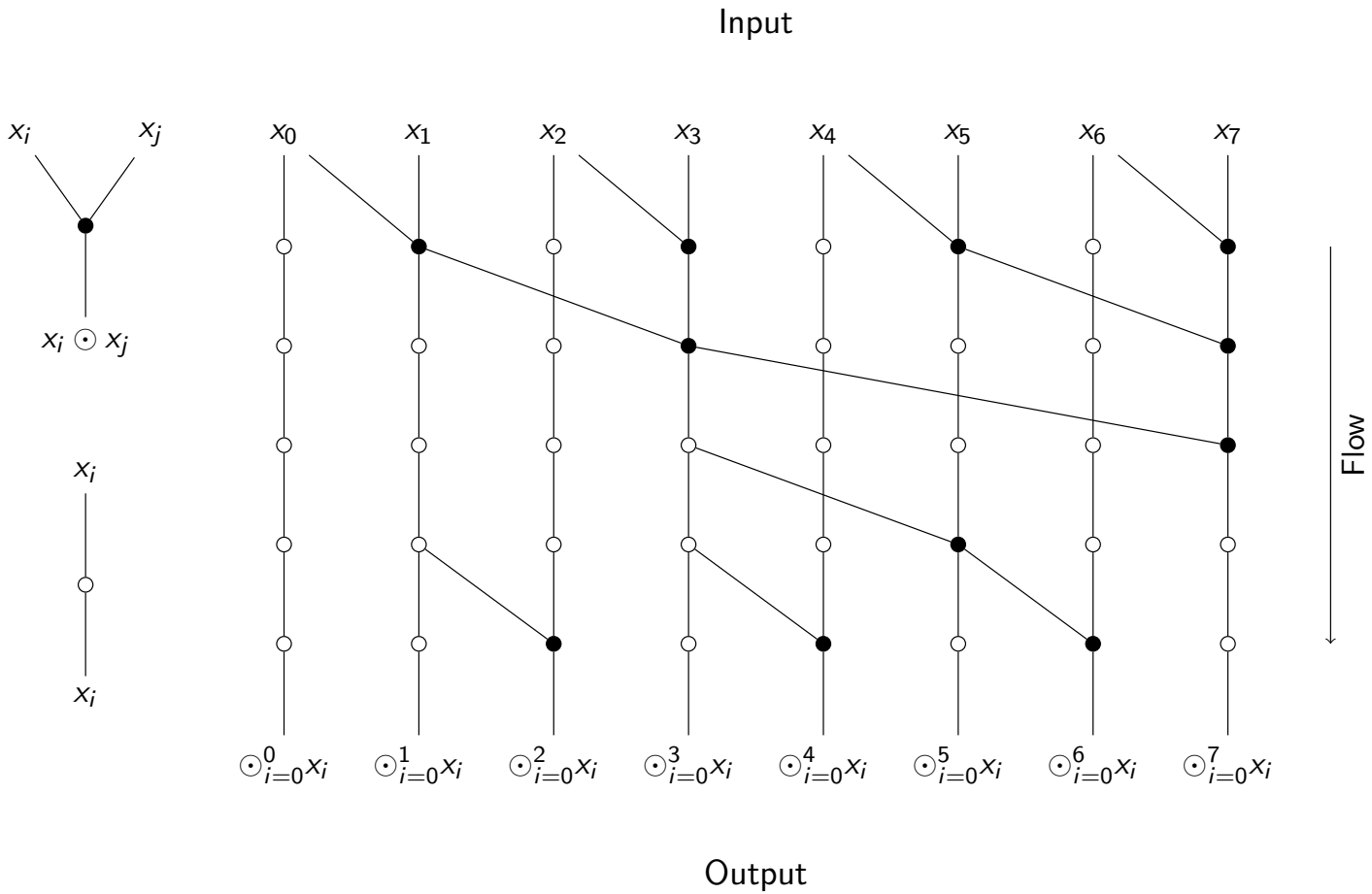
$$S(N) = \log_2 N$$

Kogge–Stone



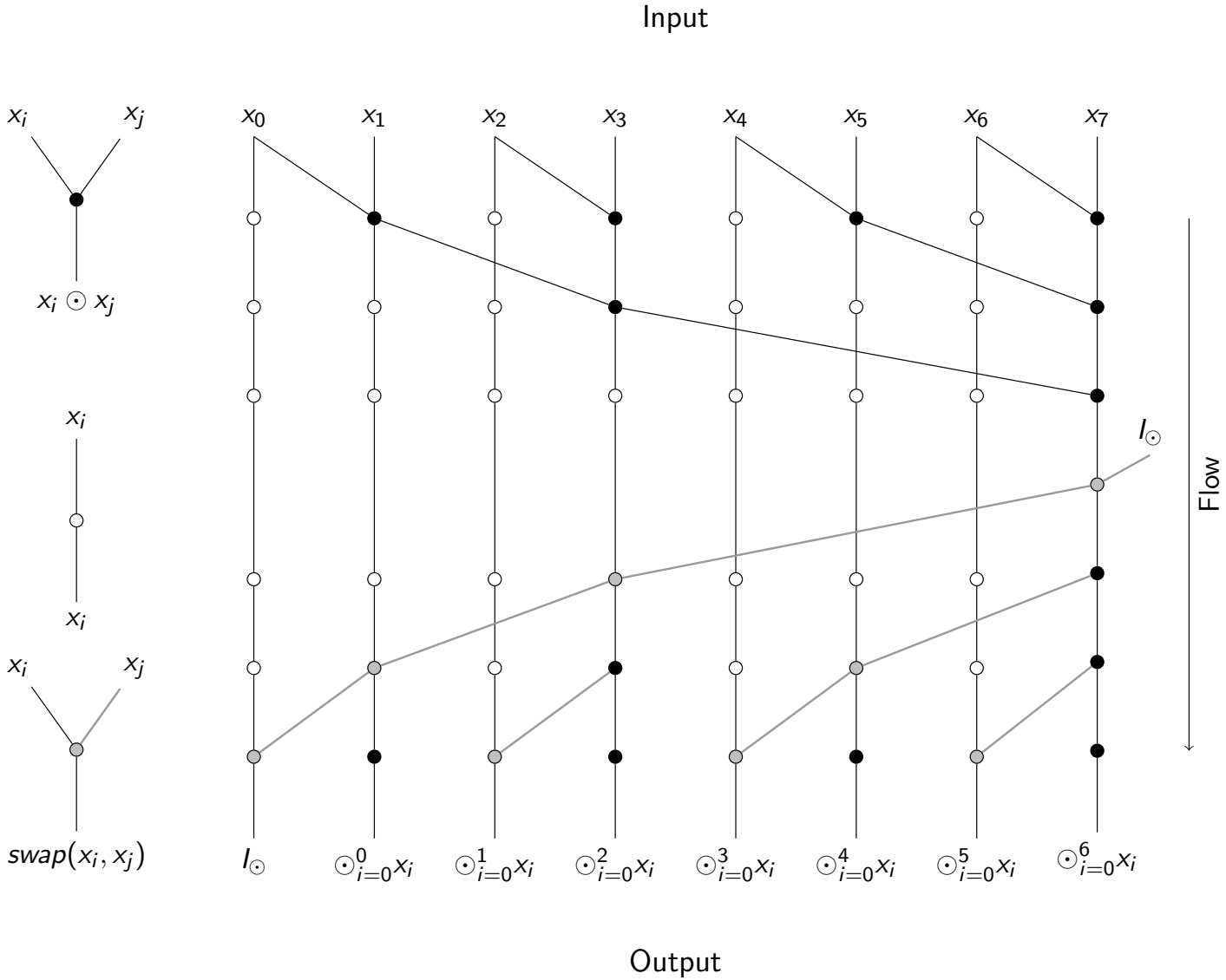
$$S(N) = \log_2 N$$

Brent–Kung



$$S(N) = 2\log_2 N - 1$$

Blelloch



$$S(N) = 2 \log_2 N$$

Comparison

Name	Type	Span	Work
Sequential	Inclusive	$N - 1$	$N - 1$
Blelloch	Exclusive	$2 \cdot \log_2 N$	$2(N - 1)$
Brent–Kung	Inclusive	$2 \cdot \log_2 N - 1$	$2 \cdot N - \log_2 N - 2$
Kogge–Stone	Inclusive	$\log_2 N$	$N \cdot \log_2 N - N + 1$
Sklansky	Inclusive	$\log_2 N$	$\frac{N}{2} \cdot \log_2 N$

Properties

- decrease in span is accompanied by an increase in work
- there is a class of **zero–deficient** parallel prefix algorithms, where a decrease in span is equal to an increase in work
- span–optimal algorithms can not be zero–deficient

Outline

Problem statement

Distributed implementation

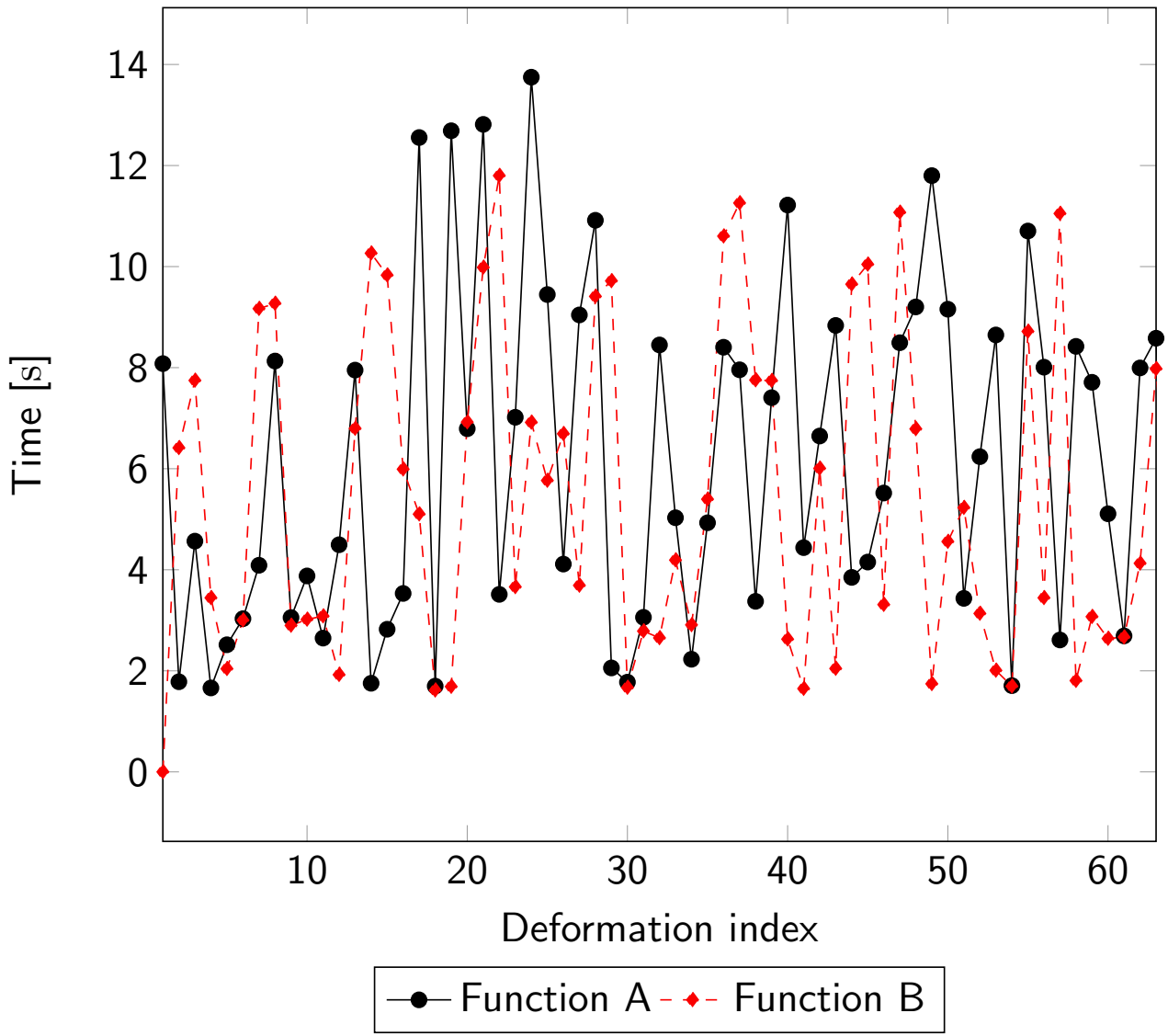
- register $N + 1$ images on P MPI ranks
- image files accessible by all ranks, only deformations are shared between workers
- in a rigid case, the communication cost is equal to sending three floating–point numbers

Related work

- pipelined binary trees (2006)
- 2Tree algorithm exploiting bidirectional communication (2009)
- research focused on reducing communication cycles and improving bandwidth

Computational intensity

Execution time for functions **A** and **B** for serial matching of 64 images.



Variance of execution time

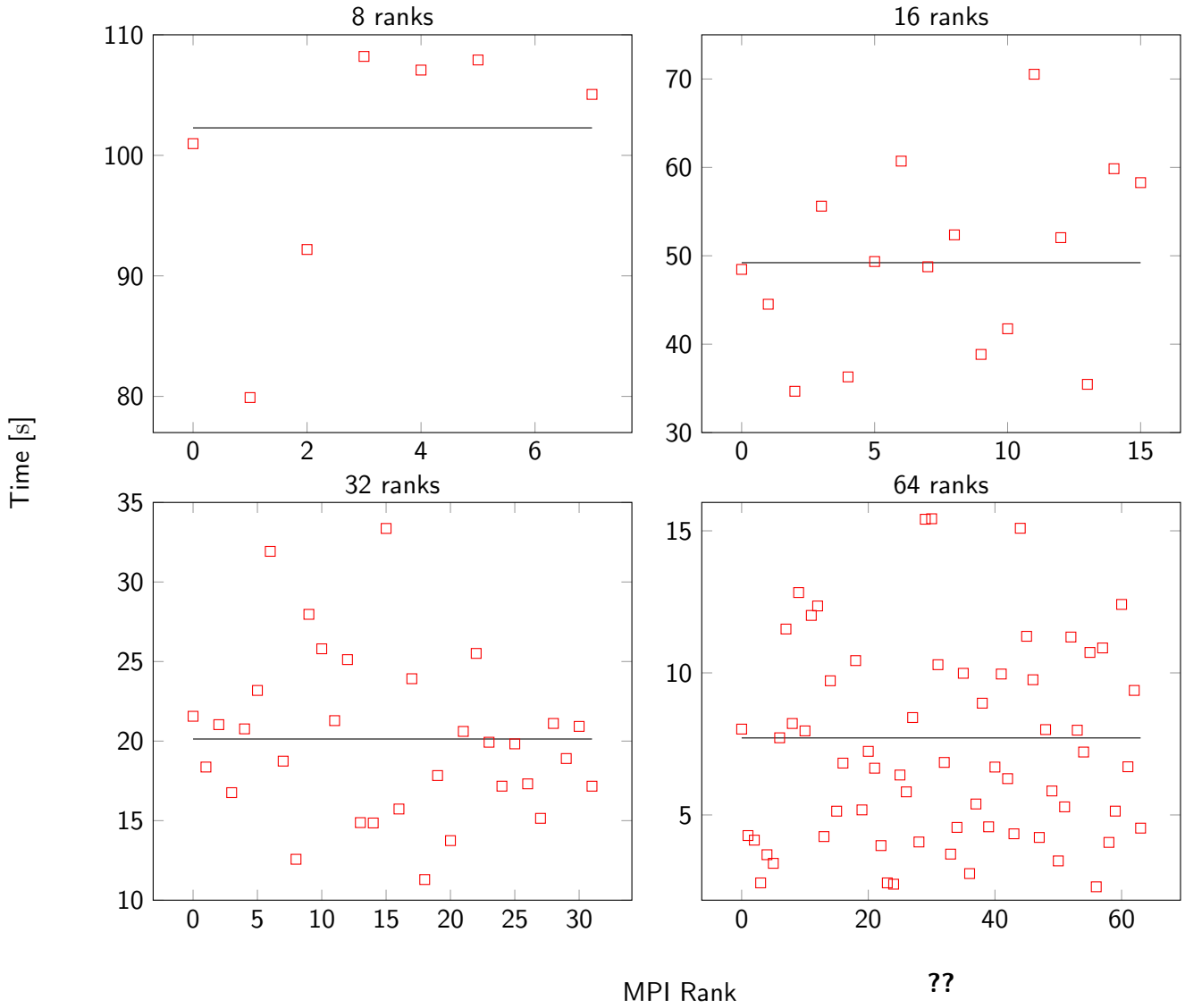


Figure: Execution time of function **B** for 128 images

Associativity

In registration of f_0 and f_4, is there a difference between

$$\phi_{0,2} = \mathsf{B}(\phi_{0,1}, \phi_{1,2})$$
$$\phi_{0,3} = \mathsf{B}(\phi_{0,2}, \phi_{2,3})$$
$$\phi_{0,4} = \mathsf{B}(\phi_{0,3}, \phi_{3,4})$$

and

$$\phi_{0,2} = \mathsf{B}(\phi_{0,1}, \phi_{1,2})$$
$$\phi_{2,4} = \mathsf{B}(\phi_{2,3}, \phi_{3,4})$$
$$\phi_{0,4} = \mathsf{B}(\phi_{0,2}, \phi_{2,4})$$

Associativity

In registration of f_0 and f_4, is there a difference between

$$\phi_{0,2} = \mathsf{B}(\phi_{0,1}, \phi_{1,2})$$
$$\phi_{0,3} = \mathsf{B}(\phi_{0,2}, \phi_{2,3})$$
$$\phi_{0,4} = \mathsf{B}(\phi_{0,3}, \phi_{3,4})$$

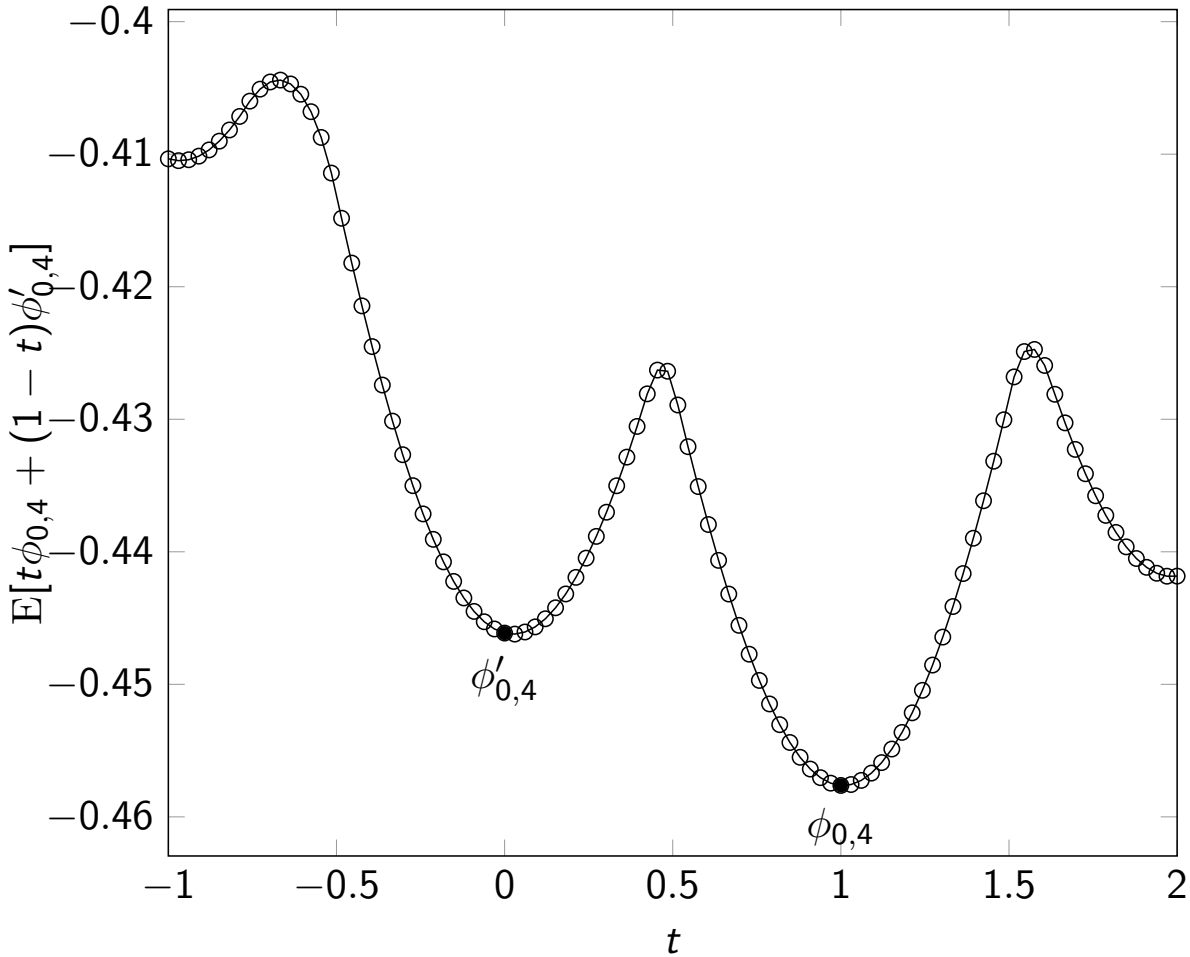
and

$$\phi_{0,2} = \mathsf{B}(\phi_{0,1}, \phi_{1,2})$$
$$\phi_{2,4} = \mathsf{B}(\phi_{2,3}, \phi_{3,4})$$
$$\phi_{0,4} = \mathsf{B}(\phi_{0,2}, \phi_{2,4})$$

Yes.

Approximate associativity

The evaluation of energy functional in neighborhood of $\phi_{0,4}$ and $\phi'_{0,4}$



Registration operator

Literature describes prefix sums where the operator

- is simple and not computationally intensive
- has a stable and deterministic execution time
- is associative

Registration operator

Literature describes prefix sums where the operator

- is simple and not computationally intensive
- has a stable and deterministic execution time
- is associative

In the image registration, the prefix sum operator is

- is complex and computationally intensive
- has an unpredictable execution time
- is **approximately** associative i.e. changing the order of execution may result in a different, but still correct solution

Algorithm

Assumptions

- reduce the span to minimum - **"do more work but in parallel"**
- communication cost is much smaller than computation cost
- $\log_2 P < \log_2 N \implies$ a global prefix sum with one data element per rank
- equal distribution of work between MPI ranks

Algorithm

Lineout

- rank J obtains a sequence of data $x_{l_J}, x_{l_J+1}, \ldots, x_{r_J}$ with $K = \frac{N}{P}$ elements
- rank J performs a local prefix sum, reducing sequence to x_{l_J,r_J}

$$S(N,P) = \frac{N}{P} - 1$$

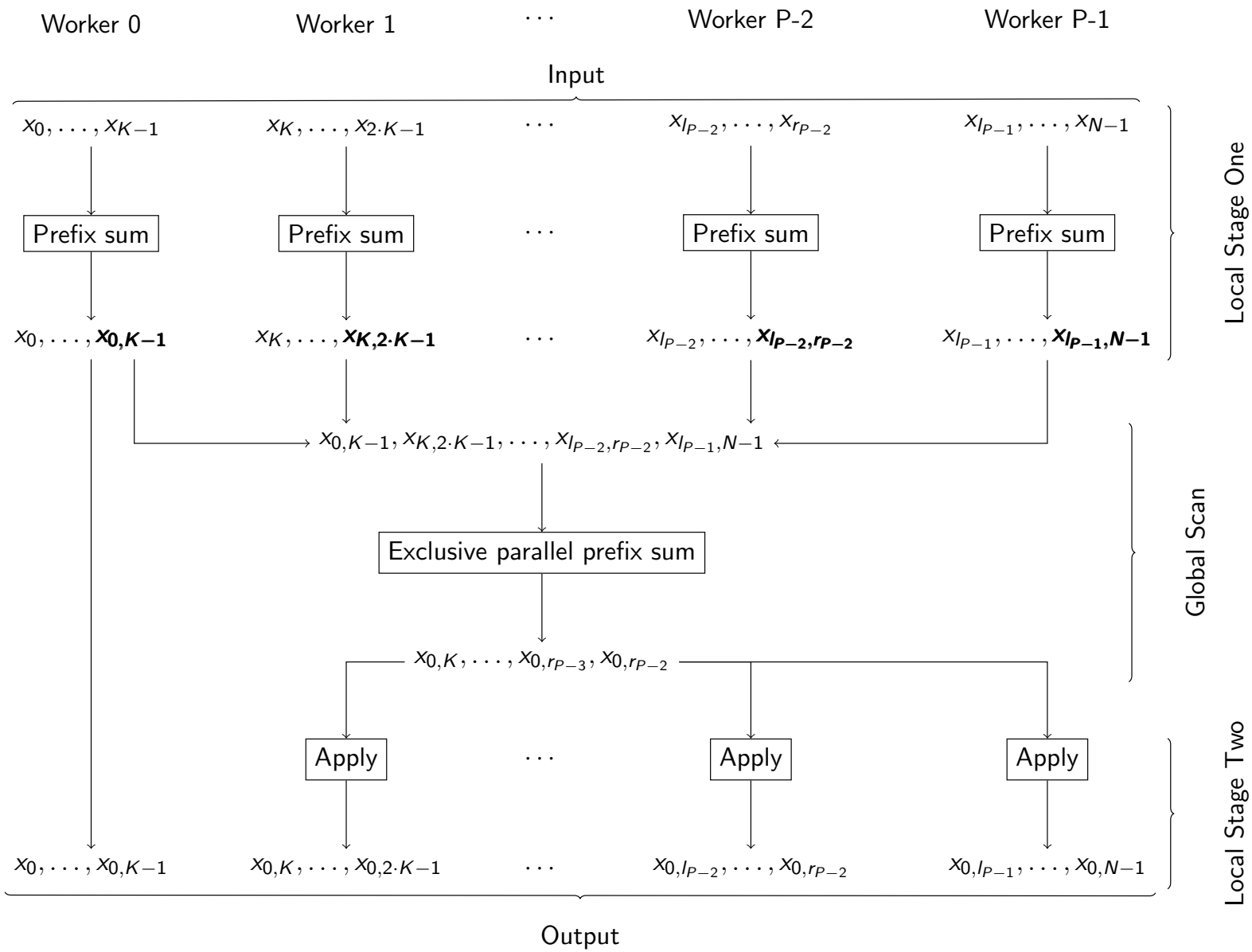
- a global exclusive prefix sum on reduced values x_{l_J,r_J} is performed. rank I obtains a result $x_{0,r_{J-1}}$

$$S(N,P) = C_1 \log_2 P$$

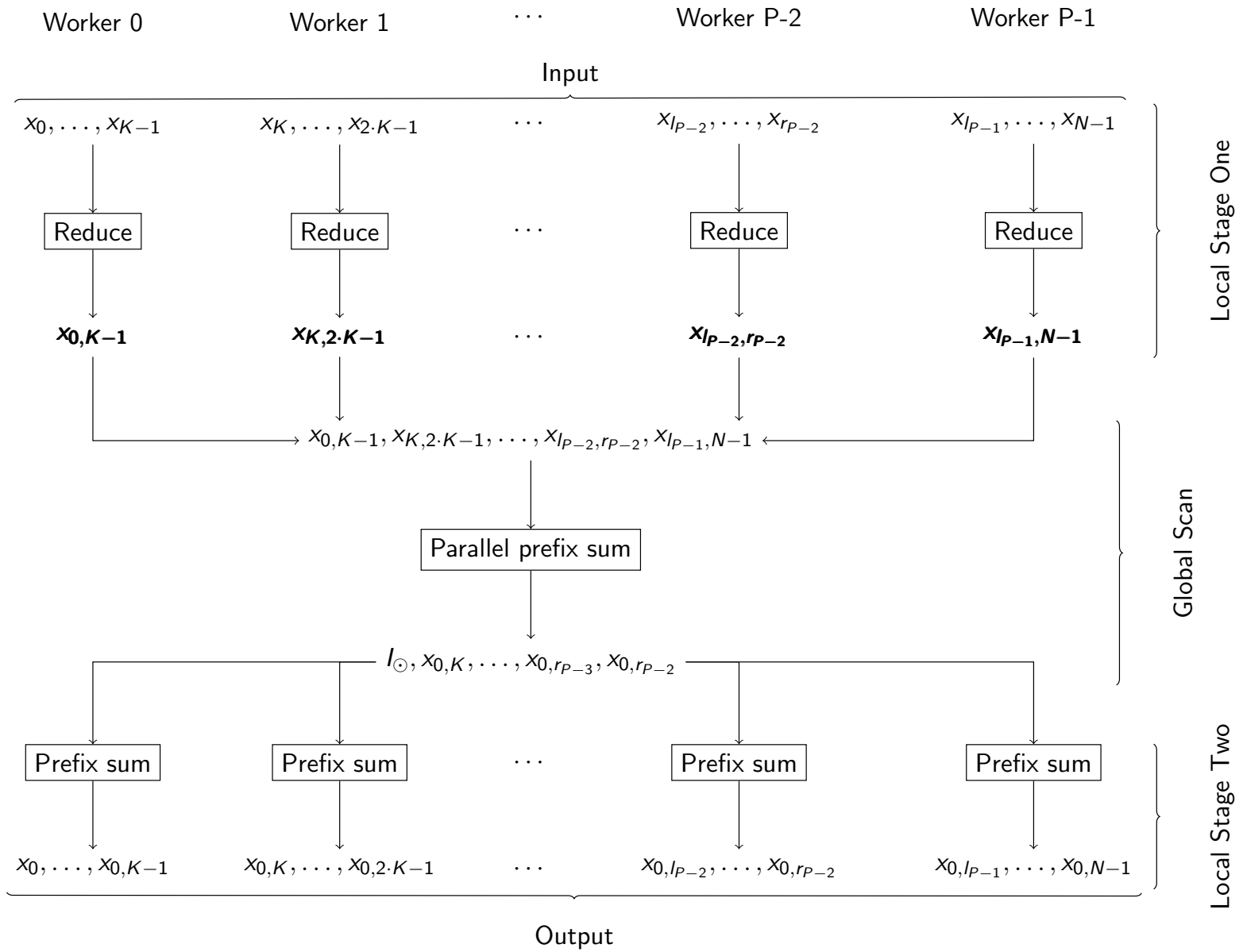
- $x_{0,r_{J-1}}$ and x_{l_J,l_J+i} are combined to form x_{0,l_J+i}

$$S(N,P) = \frac{N}{P}$$

Algorithm


Worker 0
Worker 1
· · ·
Worker P-2
Worker P-1
Input
$x_0, \ldots, x_{K-1}$
$x_K, \ldots, x_{2 \cdot K-1}$
· · ·
$x_{l_{P-2}}, \ldots, x_{r_{P-2}}$
$x_{l_{P-1}}, \ldots, x_{N-1}$
Prefix sum
Prefix sum
· · ·
Prefix sum
Prefix sum
$x_0, \ldots, \boldsymbol{x_{0,K-1}}$
$x_K, \ldots, \boldsymbol{x_{K,2 \cdot K-1}}$
· · ·
$x_{l_{P-2}}, \ldots, \boldsymbol{x_{l_{P-2},r_{P-2}}}$
$x_{l_{P-1}}, \ldots, \boldsymbol{x_{l_{P-1},N-1}}$
Local Stage One
$x_{0,K-1}, x_{K,2 \cdot K-1}, \ldots, x_{l_{P-2},r_{P-2}}, x_{l_{P-1},N-1}$
Exclusive parallel prefix sum
$x_{0,K}, \ldots, x_{0,r_{P-3}}, x_{0,r_{P-2}}$
Global Scan
Apply
Apply
· · ·
Apply
$x_0, \ldots, x_{0,K-1}$
$x_{0,K}, \ldots, x_{0,2 \cdot K-1}$
· · ·
$x_{0,l_{P-2}}, \ldots, x_{0,r_{P-2}}$
$x_{0,l_{P-1}}, \ldots, x_{0,N-1}$
Local Stage Two
Output

Alternative formulation



Distributed prefix sum

Span

$$S(N, P) = \frac{N}{P} + C_1 \log_2 P + \frac{N}{P} + C_2$$

constant C_2 depends on choice of strategy and global prefix sum

Speedup

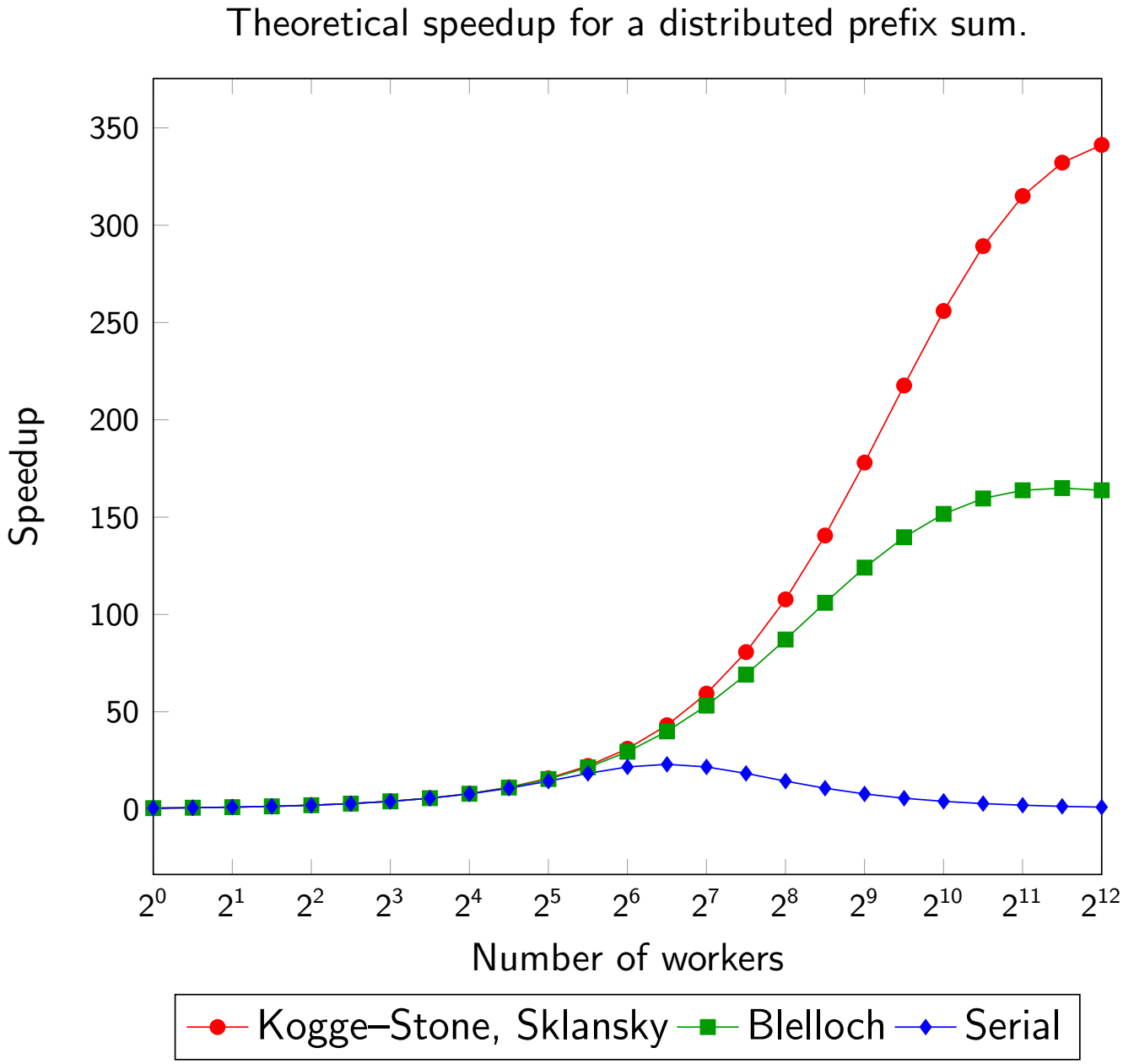
$$SP(N, P) = \frac{N - 1}{\frac{2N}{P} + C_1 \log_2 P + C_2}$$

Global prefix sum

Algorithms

- Kogge–Stone
- Sklansky
- Blelloch
- OpenMPI
  - `MPI_Scan`
  - `MPI_Exscan`
- IntelMPI
  - `MPI_Scan`, `MPI_Exscan`
  - *topology–aware* versions of standard algorithms

Theoretical comparison



Outline

Results

Scenarios

- strong scaling
- weak scaling
- multithreading

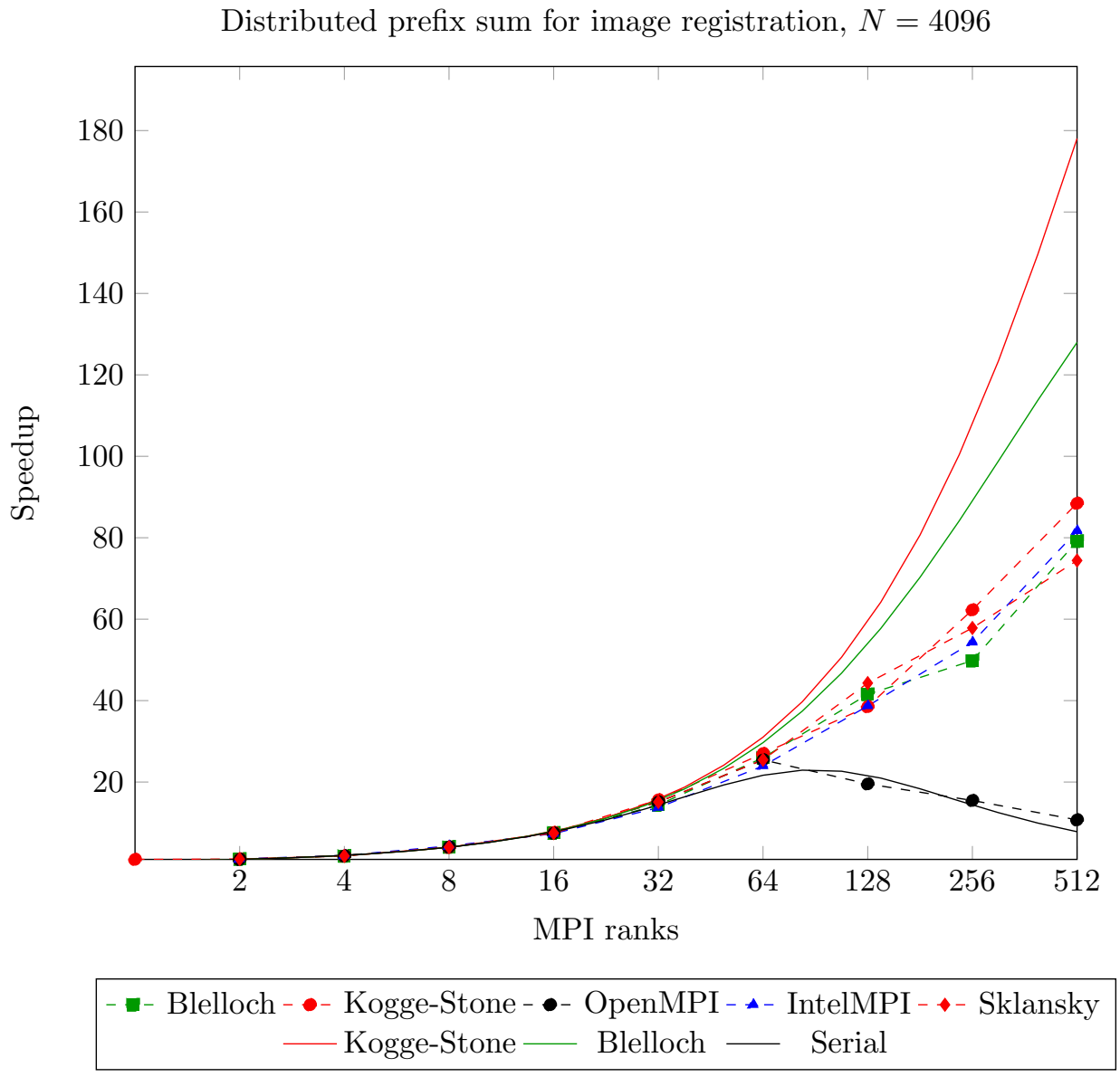
Other results

- for OpenMPI, an inclusive prefix sum attains the best performance
- for IntelMPI, an exclusive prefix sum with a *topology–aware* algorithm attains the best performance
- the alternative strategy does not improve the performance

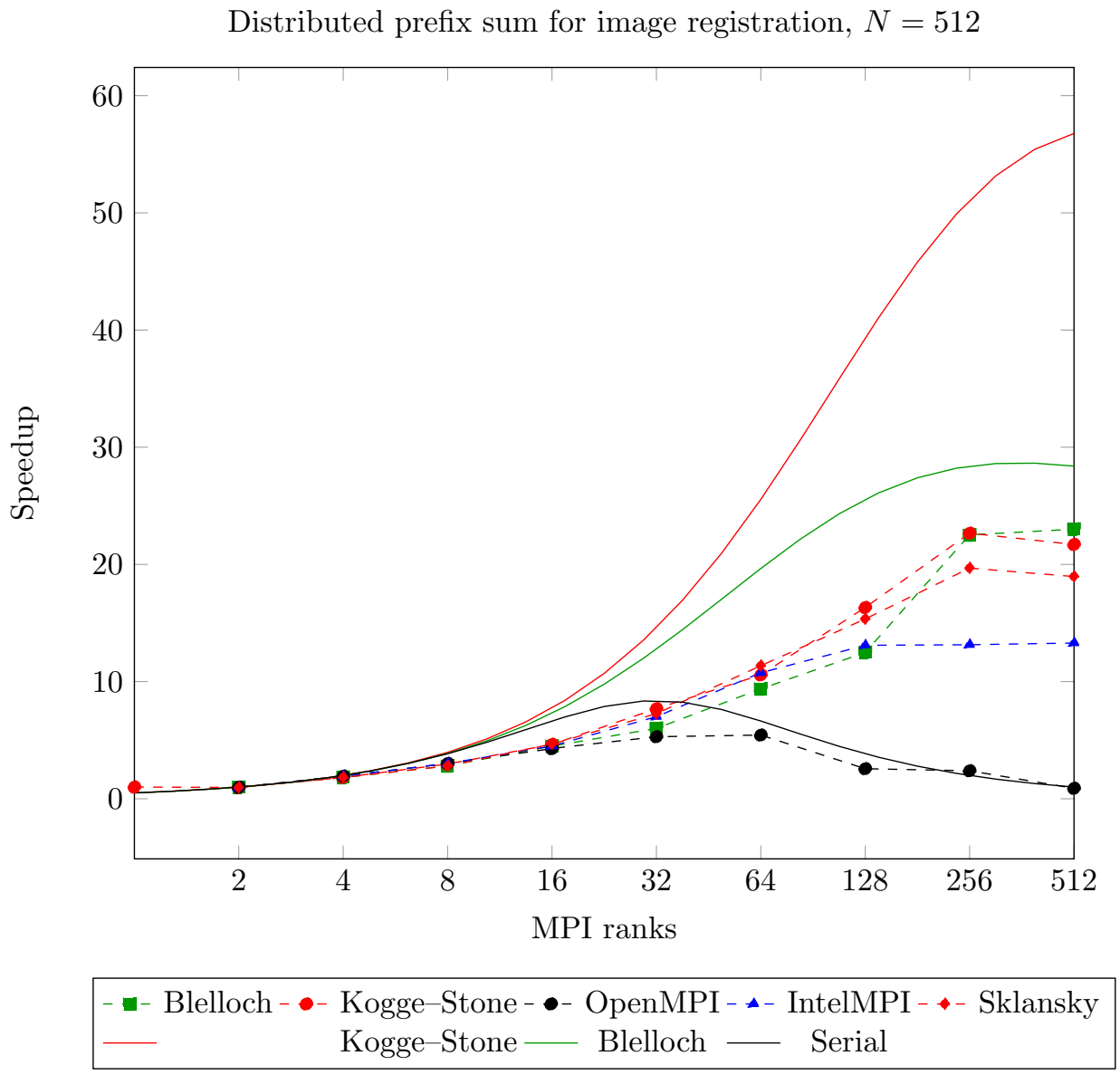
Strong scaling

- test the algorithm for an increasing number of MPI ranks P and a constant size N
- with a decreasing chunk of work per rank, the global stage starts to dominate the runtime

Strong scaling



Strong scaling



Weak scaling

- test the algorithm for a constant chunk of work per rank $\frac{N}{P}$
- the span of a distributed prefix sum can be given in a generic form

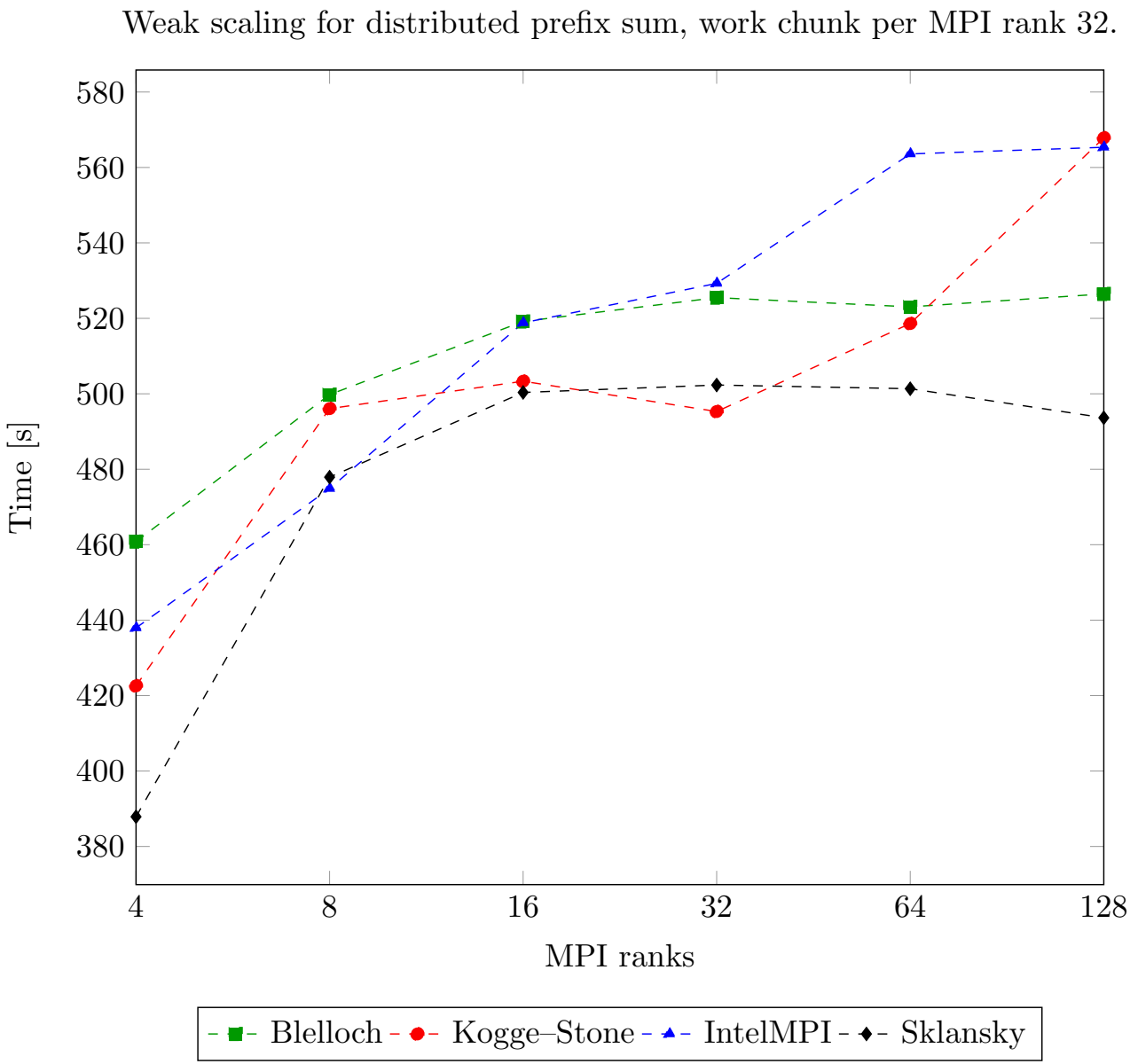
$$S(N,P) = \frac{2N}{P} + C_1 \log_2 P + C_2,\ C_1, C_2 \in \mathbb{N}$$

then

$$\begin{aligned} S(2N,2P) &= \frac{4N}{2P} + C_1 \log_2 2P + C_2 \\ &= S(N,P) + C_1 \end{aligned}$$

thus, an increase in execution time is expected

Weak scaling



Multithreading

- instead of a distributed prefix sum on P ranks, use $\frac{P}{T}$ ranks and allocate T threads for each rank to parallelize the registration process
- the image registration operator can be parallelized with estimated speedup 1.67 times on two threads and 3 times on four threads
- results are presented for GCC OpenMP runtime, the Intel OpenMP runtime did not perform significantly different

Multithreading

Outline

Summary

Conclusions

- as expected, our problem does not scale well on a large number of processors
- weak scaling is far from ideal as well
- the actual performance does not meet theoretical predictions because of an unbalanced workload
- the span–optimal Kogge–Stone algorithm provides the best performance in many experiments, but not in all of them

Summary

Conclusions

- as expected, our problem does not scale well on a large number of processors
- weak scaling is far from ideal as well
- the actual performance does not meet theoretical predictions because of an unbalanced workload
- the span–optimal Kogge–Stone algorithm provides the best performance in many experiments, but not in all of them

Possible improvements

- balancing the workload in last stage
- a shared–memory implementation may reduce effects of load imbalance
- **further parallelization of the binary operator**