ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

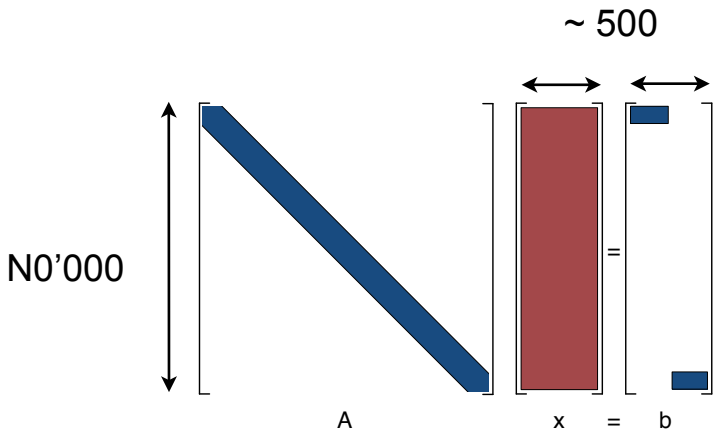
IIES
Departement Informationstechnologie
und Elektrotechnik

SPIKE Final Presentation

Otto Bibartiu
Mauro Calderara

Solving the linear system

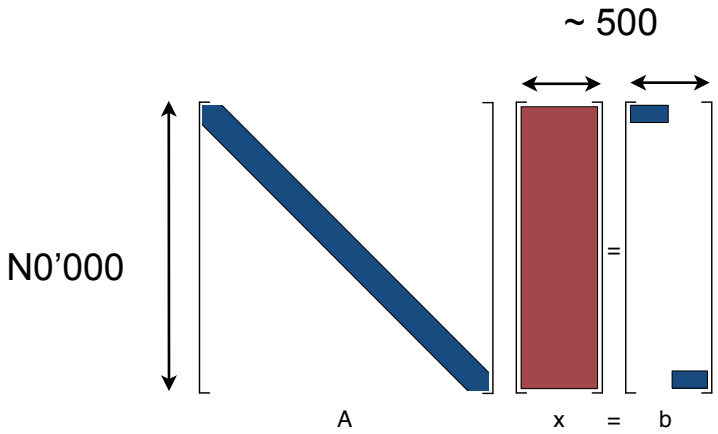
- Solve $A \cdot \vec{x} = \vec{b}$



- Previous Work: intel SPIKE, Pardiso, UMFPACK
- Algorithm well documented (papers by Polizzi & Sameh, even a Wikipedia article)

Solving the linear system

- Solve $A \cdot \vec{x} = \vec{b}$

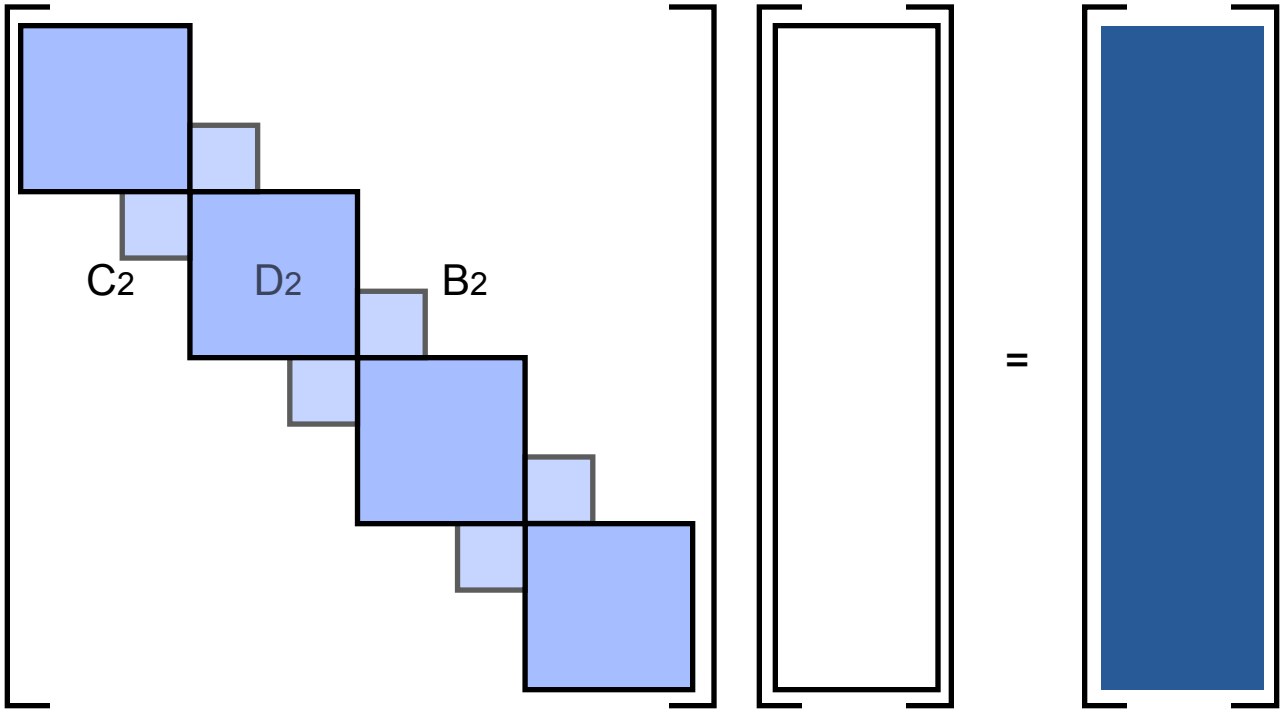


- On a super computer

- Previous Work: intel SPIKE, Pardiso, UMFPACK
- Algorithm well documented (papers by Polizzi & Sameh, even a Wikipedia article)

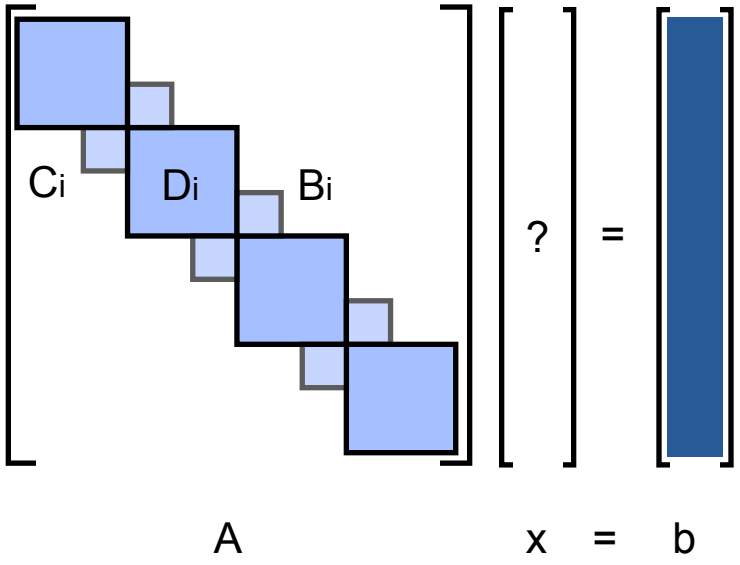
General SPIKE

- For block tridiagonal matrices



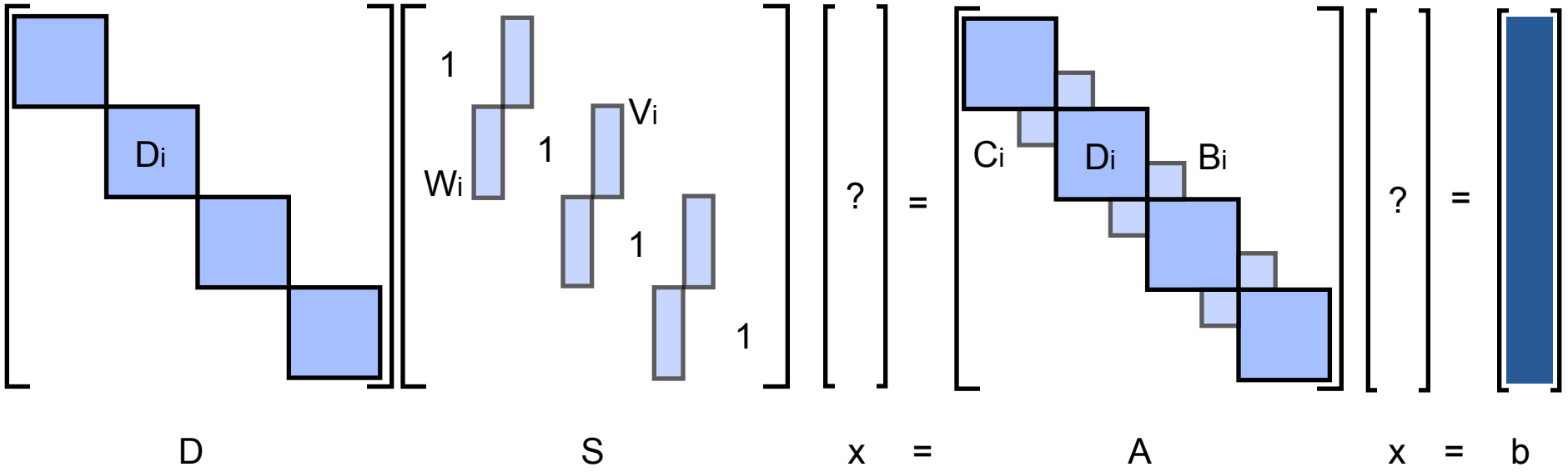
General SPIKE

- Rewrite A as A=D*S



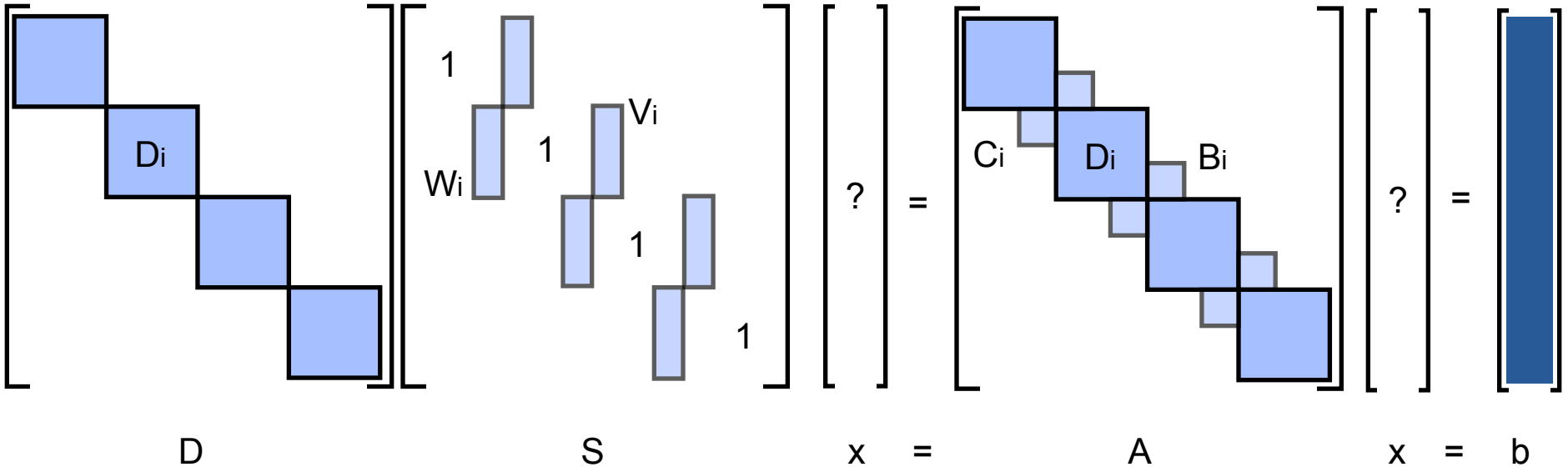
General SPIKE

- Rewrite A as A=D*S

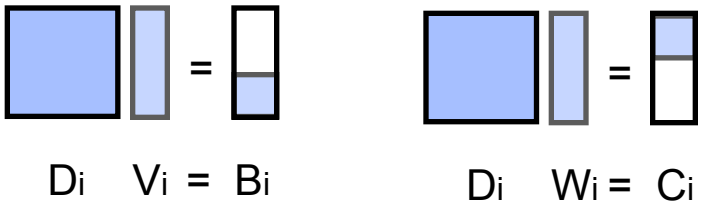


General SPIKE

- Rewrite A as A=D*S

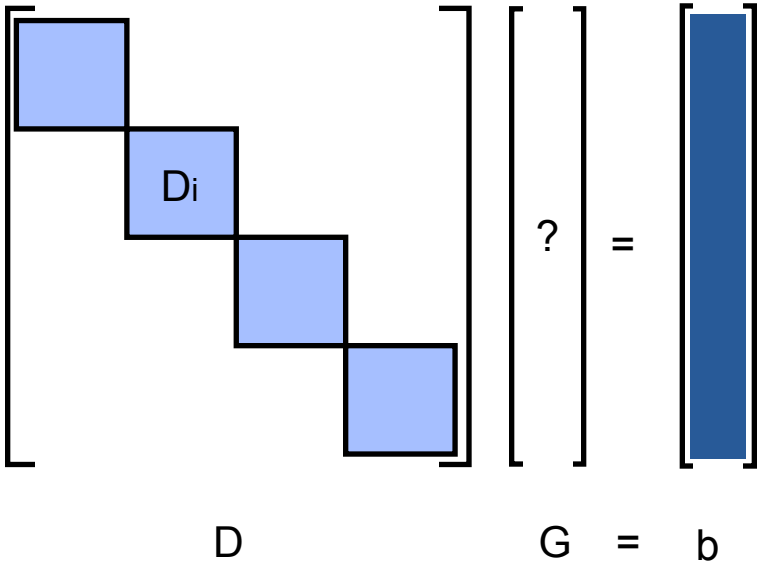


- For the spikes:



General SPIKE

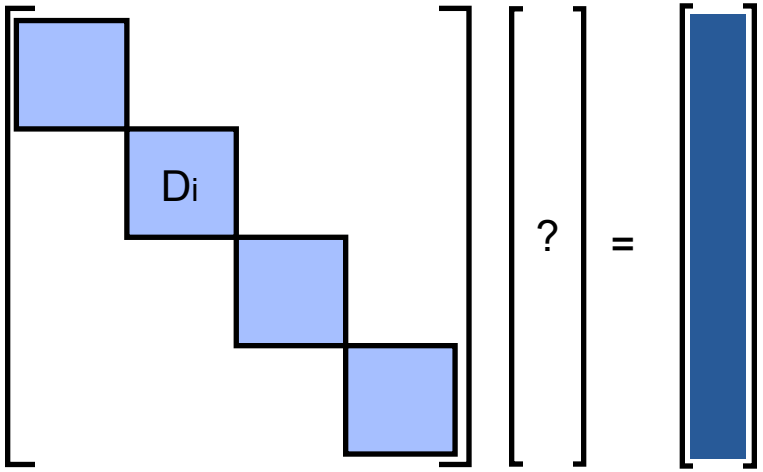
- Define G := S*x



- + Calculate spikes
- + Solve DG=b for G

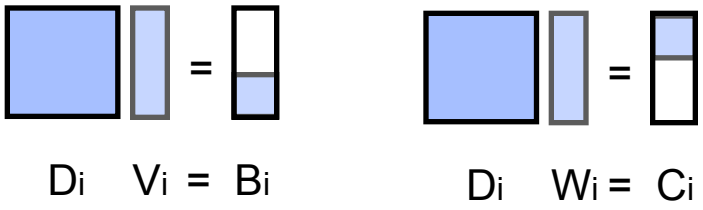
General SPIKE

- Define G := S*x



- + Calculate spikes
- + Solve DG=b for G

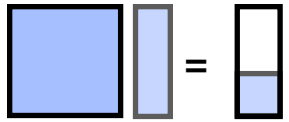
- For the spikes:



What's the benefit?

What's the benefit?

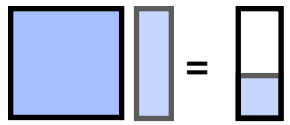
- Calculate spikes
  - mutually independent → perfectly parallel solving



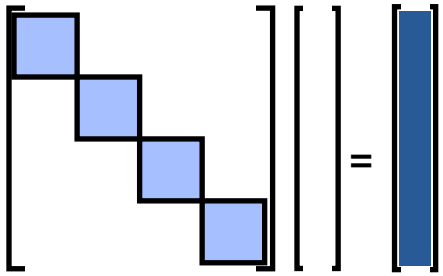
$D_i \quad V_i = B_i$

What's the benefit?

- Calculate spikes
  - mutually independent → perfectly parallel solving
- Solve DG=b
  - mutually independent → perfectly parallel solving
  - completely independent from spike generation
  - most of RHS is zero anyway in our case

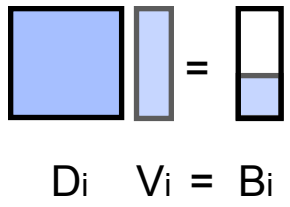


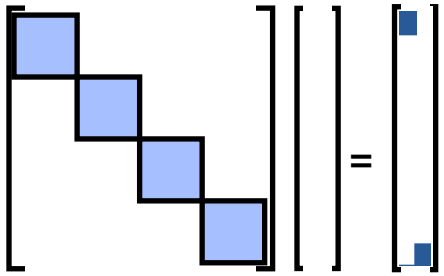
$D_i \quad V_i = B_i$



What's the benefit?

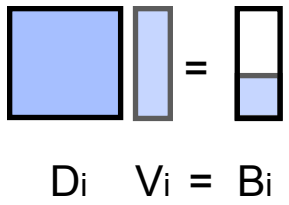
- Calculate spikes
  - mutually independent → perfectly parallel solving
- Solve DG=b
  - mutually independent → perfectly parallel solving
  - completely independent from spike generation
  - most of RHS is zero anyway in our case



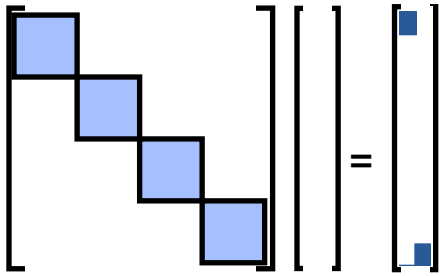


What's the benefit?

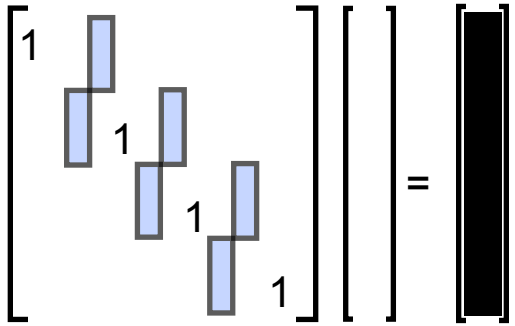
- Calculate spikes
  - mutually independent → perfectly parallel solving



- Solve DG=b
  - mutually independent → perfectly parallel solving
  - completely independent from spike generation
  - most of RHS is zero anyway in our case

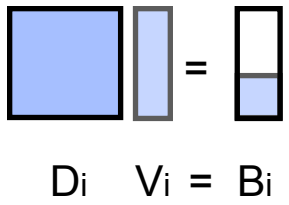


- Solve Sx=G
  - System as big as original one
  - Can be reduced (to p * 2b)

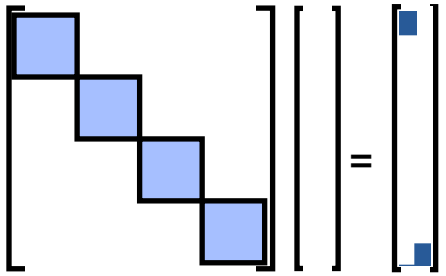


What's the benefit?

- Calculate spikes
  - mutually independent → perfectly parallel solving



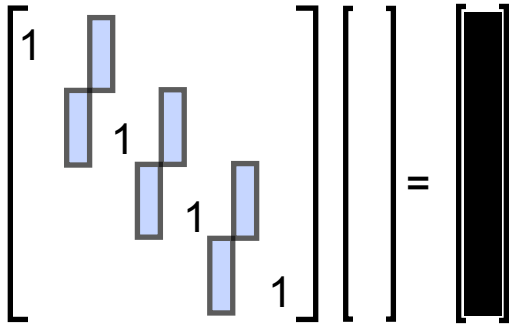
- Solve DG=b
  - mutually independent → perfectly parallel solving
  - completely independent from spike generation
  - most of RHS is zero anyway in our case



↑ Preprocessing ↑

↓ Postprocessing ↓

- Solve Sx=G
  - System as big as original one
  - Can be reduced (to p * 2b)



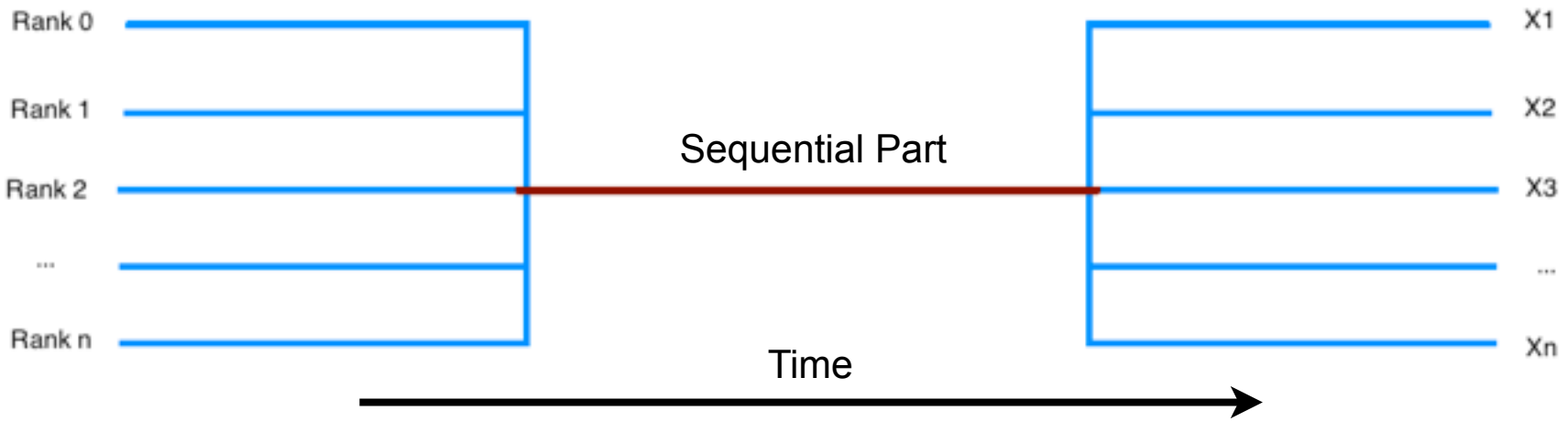
ONE SIDED MPI

MPI 3.0 One-Sided Communication

- To develop the Spike project we used mpich and openmpi.
- General approach:

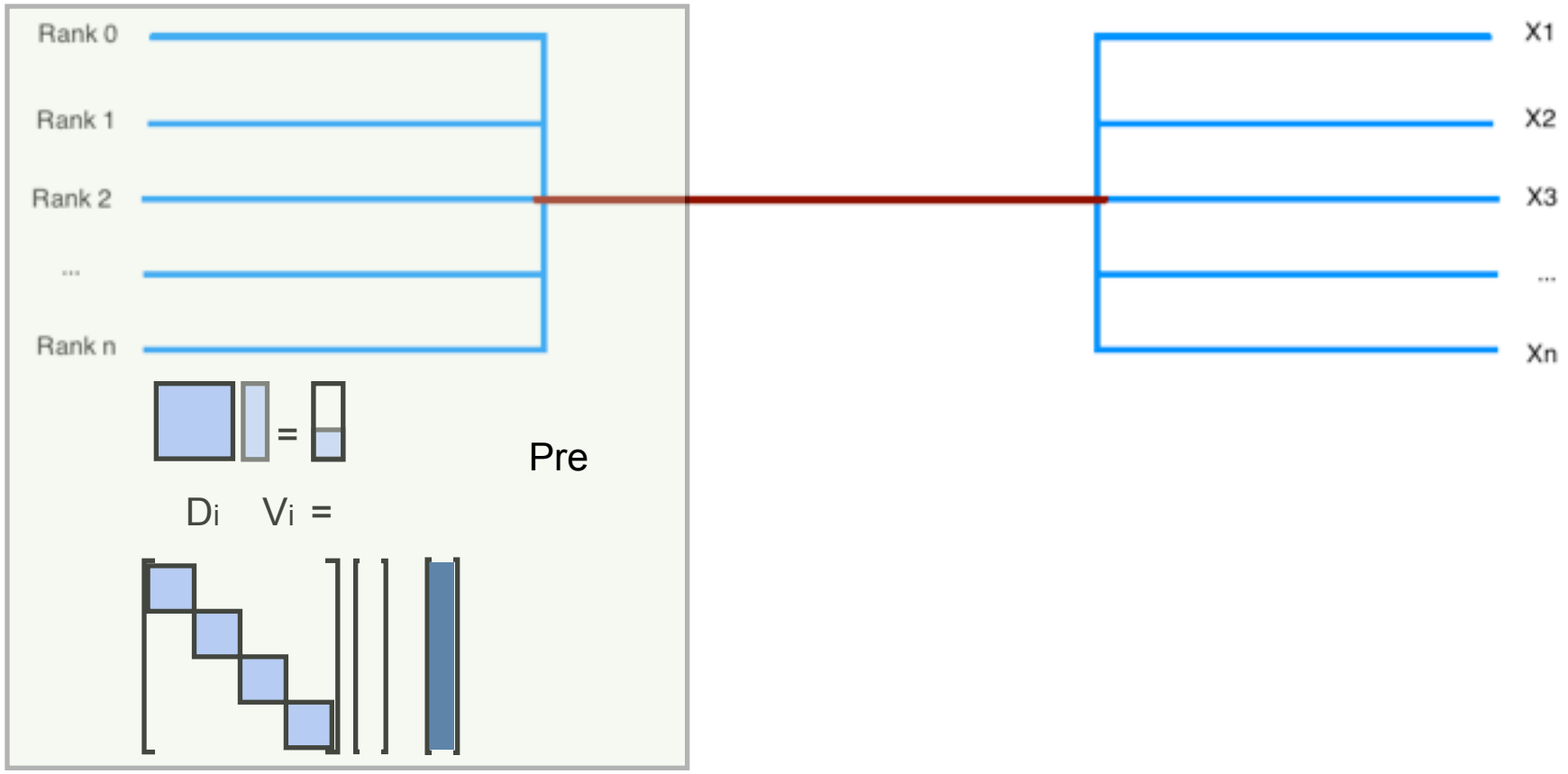
MPI 3.0 One-Sided Communication

- For the Spike project we used mpich and openmpi. General approach:



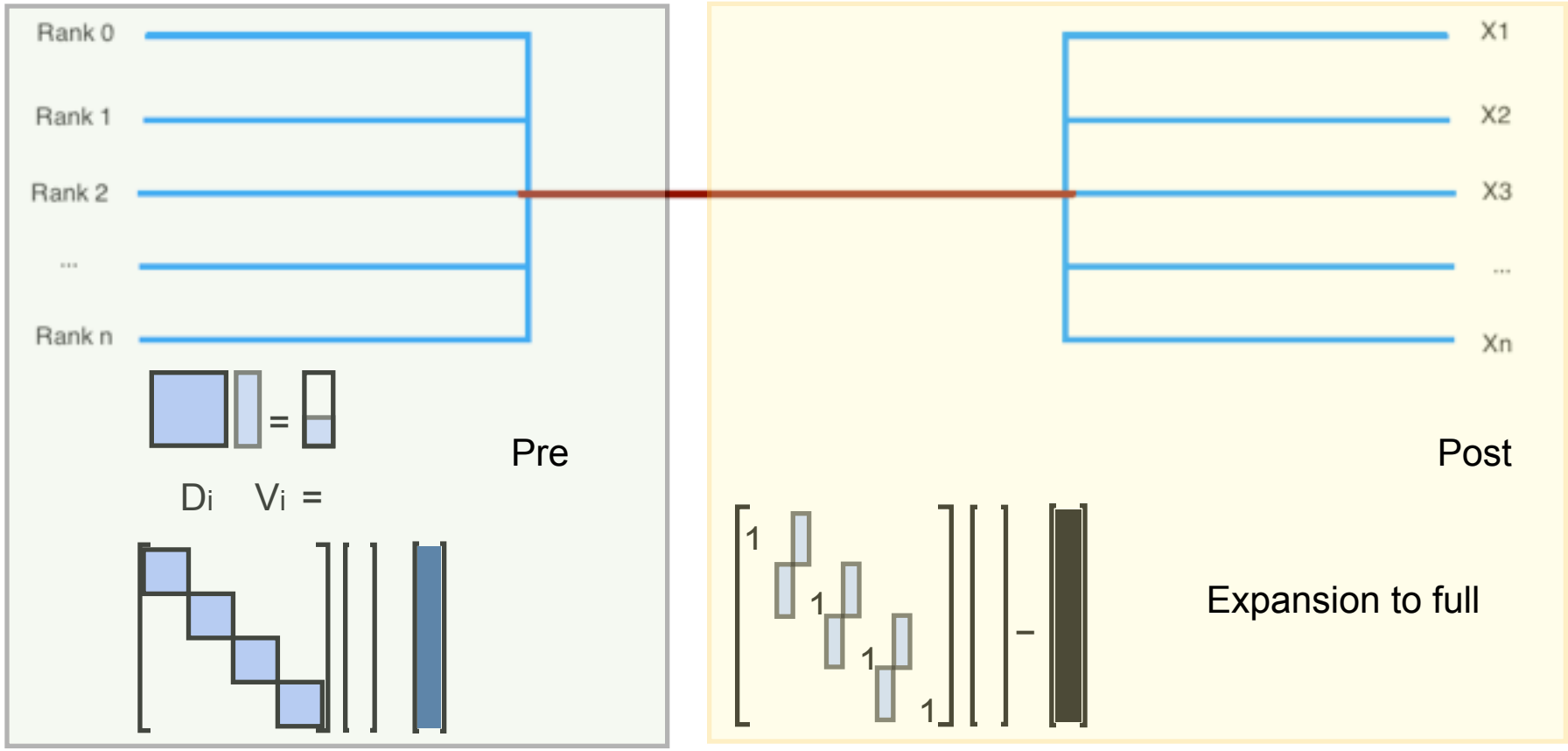
MPI 3.0 One-Sided Communication

- Program has two parts
- Preprocess



MPI 3.0 One-Sided Communication

- Program has two parts
- Preprocess + Postprocess



MPI 3.0 One-Sided Communication

- Communication scheme
- One sided Communication.

- Remote memory access (RMA) to the master ranks global shared memory vie PUT and GET
- Master rank cannot trace back who accessed
- Use is not arbitrary, we need some collective calls to organize communication (fences).

MPI 3.0 One-Sided Communication: Windows

- Every Rank in the Group has to create a Window.
- A Window denotes a piece of memory which is global for the other ranks. (collective call)
- MPI_WIN win;
- MPI_WIN_CREATE(base, size, disp_unit, info, comm, win);

MPI 3.0 One-Sided Communication

PUT

- Each rank of the group can PUT memory into the window of another rank.
- MPI_PUT(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win);

GET

- Each rank of the group can GET (read) memory from the window of another rank.
- MPI_GET(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win);

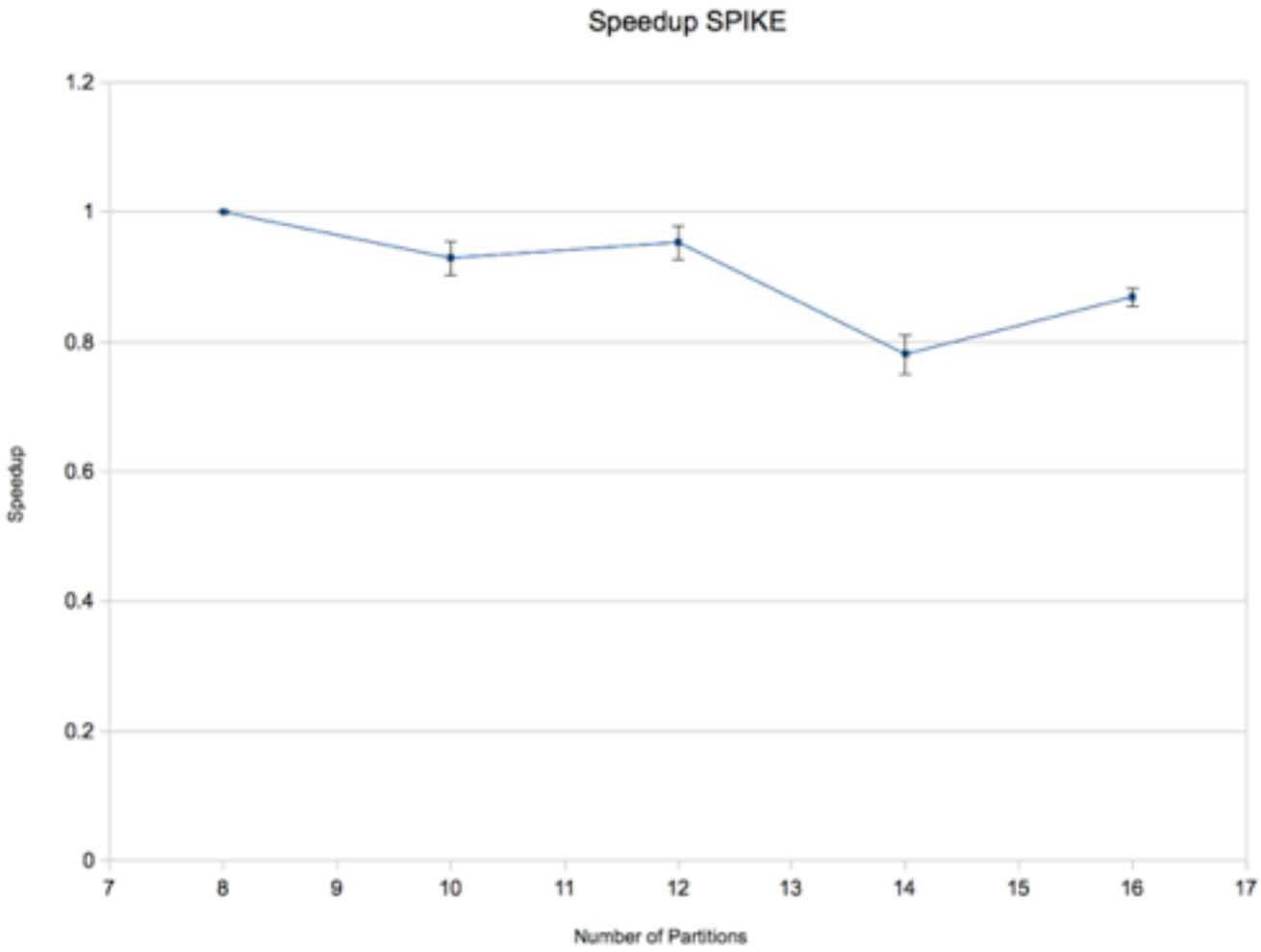
MPI 3.0 One-Sided Communication:

Fences MPI_WIN_FENCE(assert, win)

- ... are synchronizations call on windows.
- communication restriction
- PUT and GET are asynchronous, a fence waits until all commutation on the window are finished
- can be customized with assert

BENCHMARK RESULTS

Strong scaling (not so strong for SPIKE)

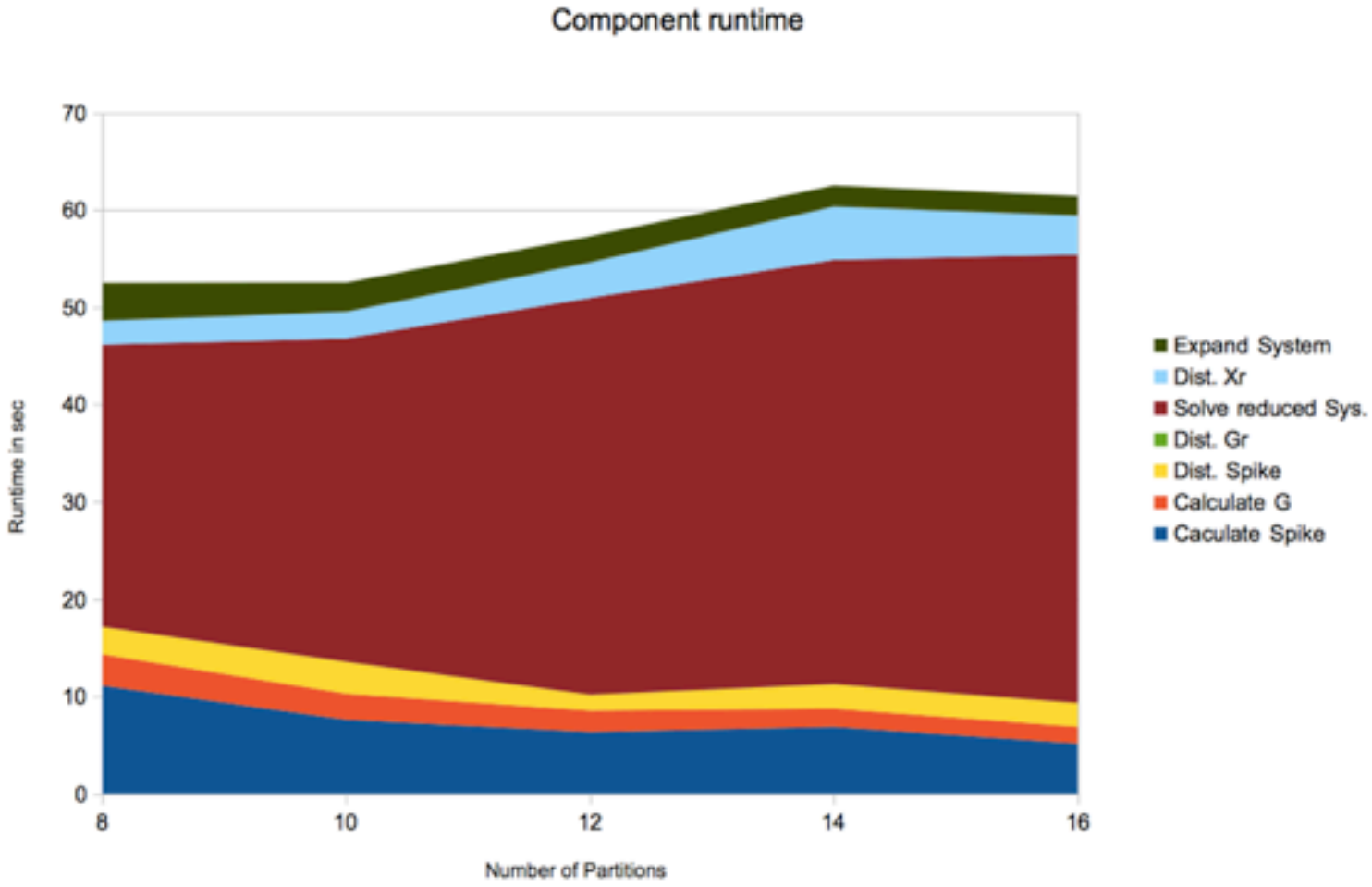


Matrix:

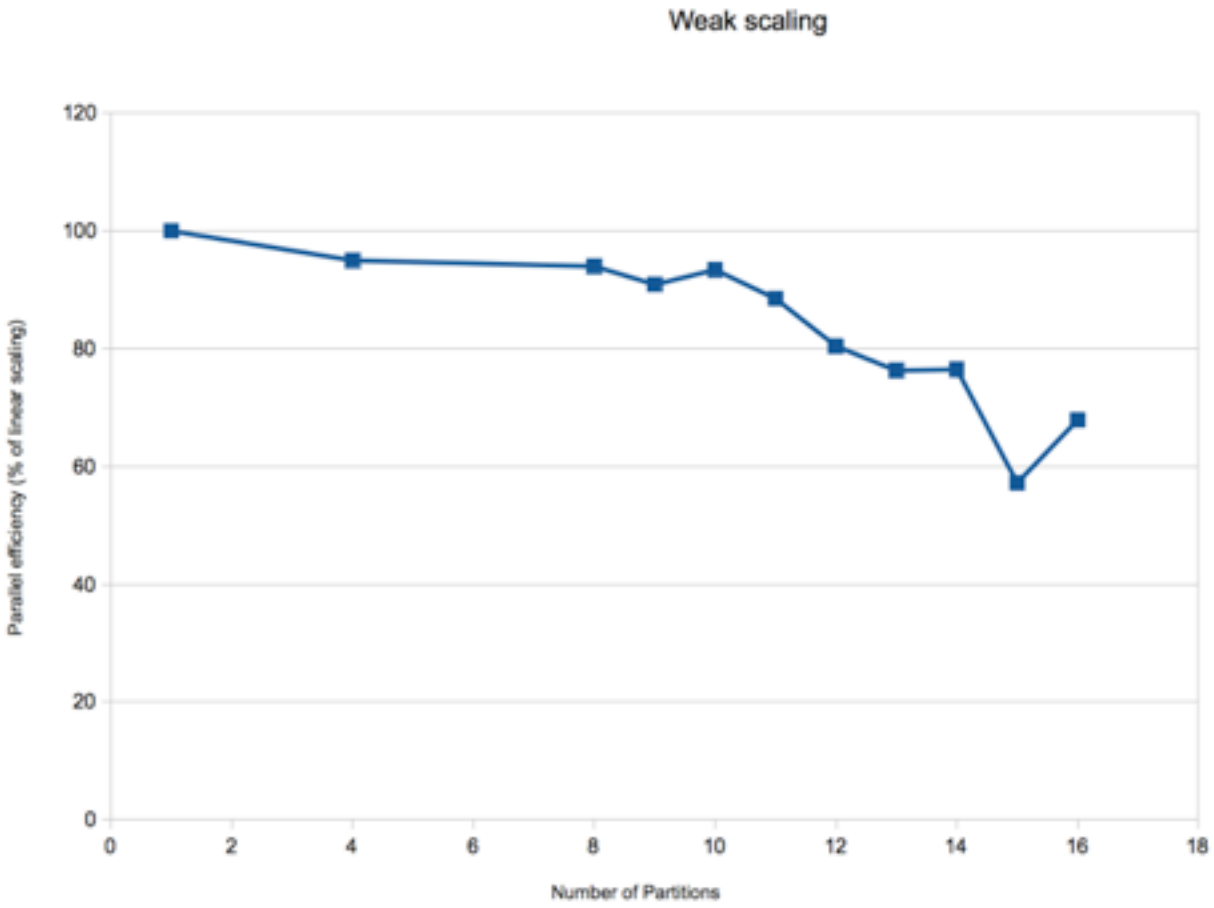
size = 40’000

BW = 1’200

More partitions → bigger sequential system



Weak scaling is stronger



Matrix:

size = 80’000

BW = 2’500

QUESTIONS?

References and documents

- “A parallel hybrid banded system solver: the SPIKE algorithm”, Polizz & Sameh, 2005 (idea is much older)
- intel SPIKE
- “MPI: A Message-Passing Interface Standard (v. 3.0)”, MPI Forum

Challenges

Challenges

- Generally: Spike is an algorithm of algorithms

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above
    - Use SPIKE again

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above
    - Use SPIKE again
      - Sequentially: Robust strategy but limited parallelism

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above
    - Use SPIKE again
      - Sequentially: Robust strategy but limited parallelism
      - Recursively: Faster, more parallelism and more potential for optimization but more complicated and bound to hit machine limits because of our thick spikes

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above
    - Use SPIKE again
      - Sequentially: Robust strategy but limited parallelism
      - Recursively: Faster, more parallelism and more potential for optimization but more complicated and bound to hit machine limits because of our thick spikes
      - Will use a combination of both

Challenges

- Generally: Spike is an algorithm of algorithms
  - Pick good sub-algorithms
  - Pick good parameters & strategies
  - Our bandwidth is comparably big
- Specifically: Getting the spikes of the middle block
  - Requires solving a linear system (again)
  - Strategies:
    - Use UMFPACK, Mumps, Pardiso, etc.
      - Same problems as above
    - Use SPIKE again
      - Sequentially: Robust strategy but limited parallelism
      - Recursively: Faster, more parallelism and more potential for optimization but more complicated and bound to hit machine limits because of our thick spikes
      - Will use a combination of both
- It all has to run (fast) on the machines we have

Some terminology and concepts

Some terminology and concepts

- Concurrency paradigms
  - Message Passing Interface (MPI):
    - split across CPUs and computers
  - Threads:
    - split across CPUs on one computer

Some terminology and concepts

- Concurrency paradigms
  - Message Passing Interface (MPI):
    - split across CPUs and computers
  - Threads:
    - split across CPUs on one computer

- Dense vs. Sparse Linear Algebra
  - Dense:
    - stores and operates on full matrices (incl. zeros)
    - O(n^3)
  - Sparse:
    - stores and operates only on non-zero elements
    - O(n) (in best case)

What you should remember

- There's this nice trick to parallelize linear system solving
- Parallelization virtually always incurs some extra cost
- There are parameters to tune, but picking them is chiefly constrained by the structure of your matrix A and the machine you want to run this on