Minimizing communication in numerical linear algebra

Edgar Solomonik

Department of Electrical and Engineering and Computer Science, UC Berkeley

Department of Energy Computational Science Graduate Fellowship Program Review

July 15, 2014

Summary

Talk thesis:

Scalable numerical algorithms and software must be designed with **a priori** consideration for **parallelism**, **communication cost**, and library **abstractions**.

Summary

Talk thesis:

Scalable numerical algorithms and software must be designed with **a priori** consideration for **parallelism**, **communication cost**, and library **abstractions**.

Talk overview

- A cost model for a modern computer
- Quantification of algorithmic costs
- Communication lower bound techniques
- Dense linear algebra algorithms
- Sparse linear algebra methods

Cost model for a modern computer

What is a modern computer?

- a set of processing units with local memories connected over a network

Cost model for a modern computer

What is a modern computer?

- a set of processing units with local memories connected over a network

Such a computer has three fundamental architectural payloads

- γ - cost for a single (floating-point) computation for one processor (computation or flop cost)
- β - cost for a transfer of each byte between any pair of processors (bandwidth cost)
- α - cost for a synchronization between any pair of processors (latency or synchronization cost)

Today: $\gamma \ll \beta \ll \alpha$, in the future: $\gamma \lll \beta \lll \alpha$

Cost model for a modern computer

What is a modern computer?

- a set of processing units with local memories connected over a network

Such a computer has three fundamental architectural payloads

- γ - cost for a single (floating-point) computation for one processor (computation or flop cost)
- β - cost for a transfer of each byte between any pair of processors (bandwidth cost)
- α - cost for a synchronization between any pair of processors (latency or synchronization cost)

Today: $\gamma \ll \beta \ll \alpha$, in the future: $\gamma \lll \beta \lll \alpha$

An additional important cost, that I will not consider in this talk

- ν - cost for a transfer of each byte between fast memory (cache) and slow memory (DRAM) ($\gamma \ll \nu < \beta$)

Quantifying the cost of a parallel schedule of an algorithm

How do we quantify the execution time **T** of some schedule of some algorithm, in terms of our architectural costs (γ, β, and α)?

Traditional method: "Volume measure"

- $\bar{F}$ – the total number of (floating-point) operations done in the algorithm,
- $\bar{W}$ – the total amount of data all processors communicate,
- $\bar{S}$ – the total number of times processors synchronize.

Quantifying the cost of a parallel schedule of an algorithm

How do we quantify the execution time **T** of some schedule of some algorithm, in terms of our architectural costs (γ, β, and α)?

Traditional method: "Volume measure"

- $\bar{F}$ – the total number of (floating-point) operations done in the algorithm,
- $\bar{W}$ – the total amount of data all processors communicate,
- $\bar{S}$ – the total number of times processors synchronize.

The "Volume measure" yields lower and upper bounds on the execution time:

$$(\gamma \cdot \bar{F} + \beta \cdot \bar{W} + \alpha \cdot \bar{S})/p \leq \mathbf{T} \leq \gamma \cdot \bar{F} + \beta \cdot \bar{W} + \alpha \cdot \bar{S}$$

where p is the number of processors.

Quantifying the cost of an schedule

How do we quantify the execution time **T** of some schedule, in terms of our architectural costs (γ, β, and α)?

Better method: "Critical path measure"

- F - longest sequence of dependent or consequently performed computations in the schedule,
- W - longest sequence of dependent or consequently performed data transfers in the schedule,
- S - longest sequence of dependent of consequently performed synchronizations (bulk requests) in the schedule.

Quantifying the cost of an schedule

How do we quantify the execution time **T** of some schedule, in terms of our architectural costs (γ, β, and α)?

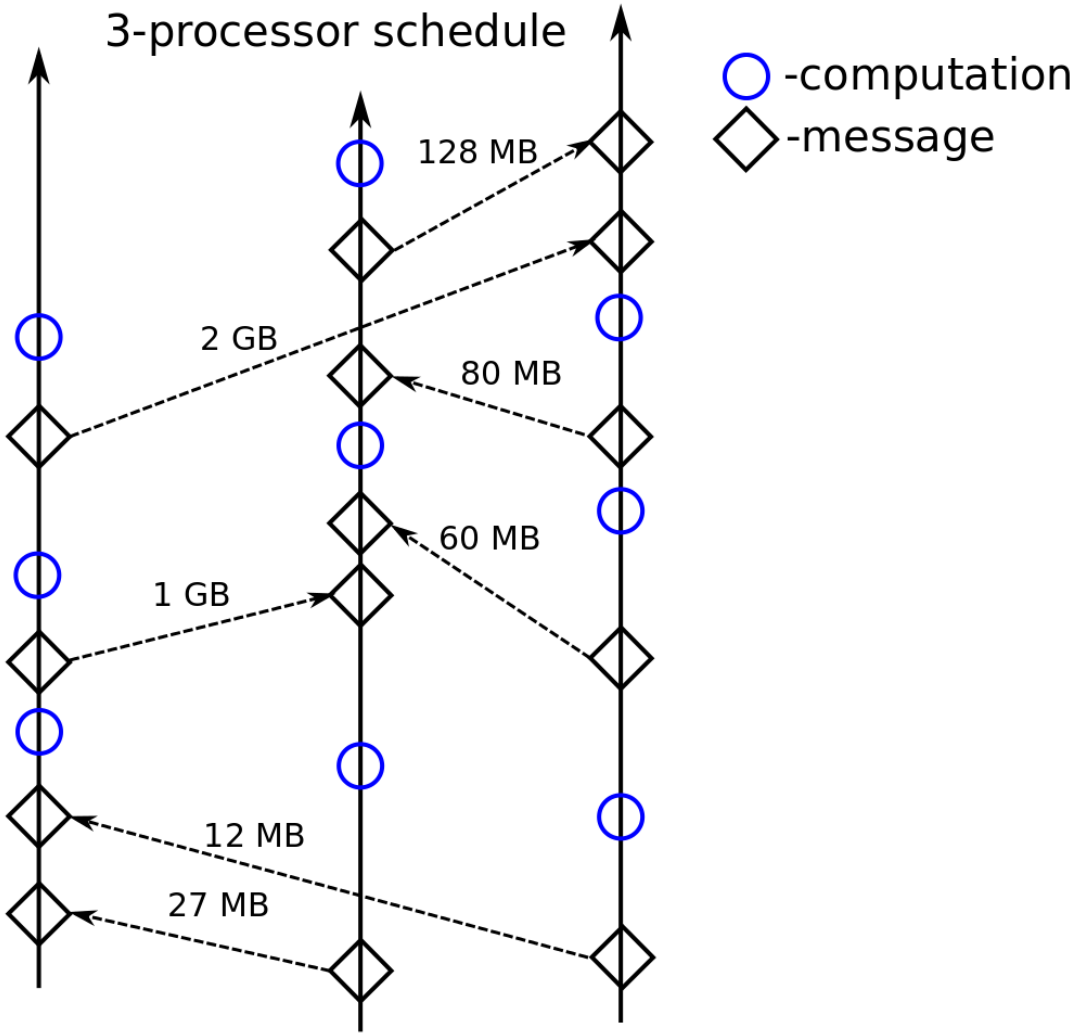
Better method: "Critical path measure"

- F - longest sequence of dependent or consequently performed computations in the schedule,
- W - longest sequence of dependent or consequently performed data transfers in the schedule,
- S - longest sequence of dependent of consequently performed synchronizations (bulk requests) in the schedule.

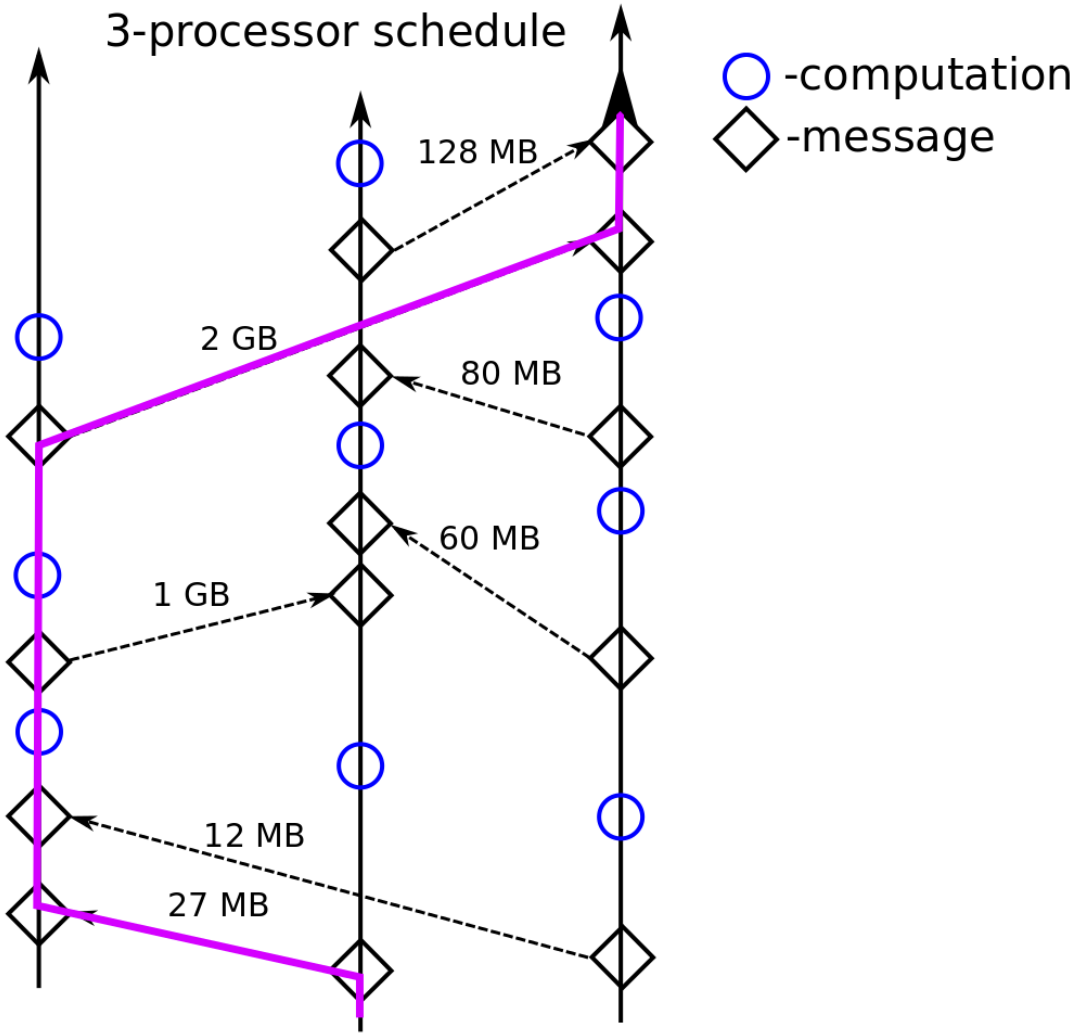
Critical path cost yields lower and upper bounds on execution time of the schedule,

$$\max(\gamma \cdot F, \beta \cdot W, \alpha \cdot S) \leq \mathbf{T} \leq \gamma \cdot F + \beta \cdot W + \alpha \cdot S.$$

Example schedule

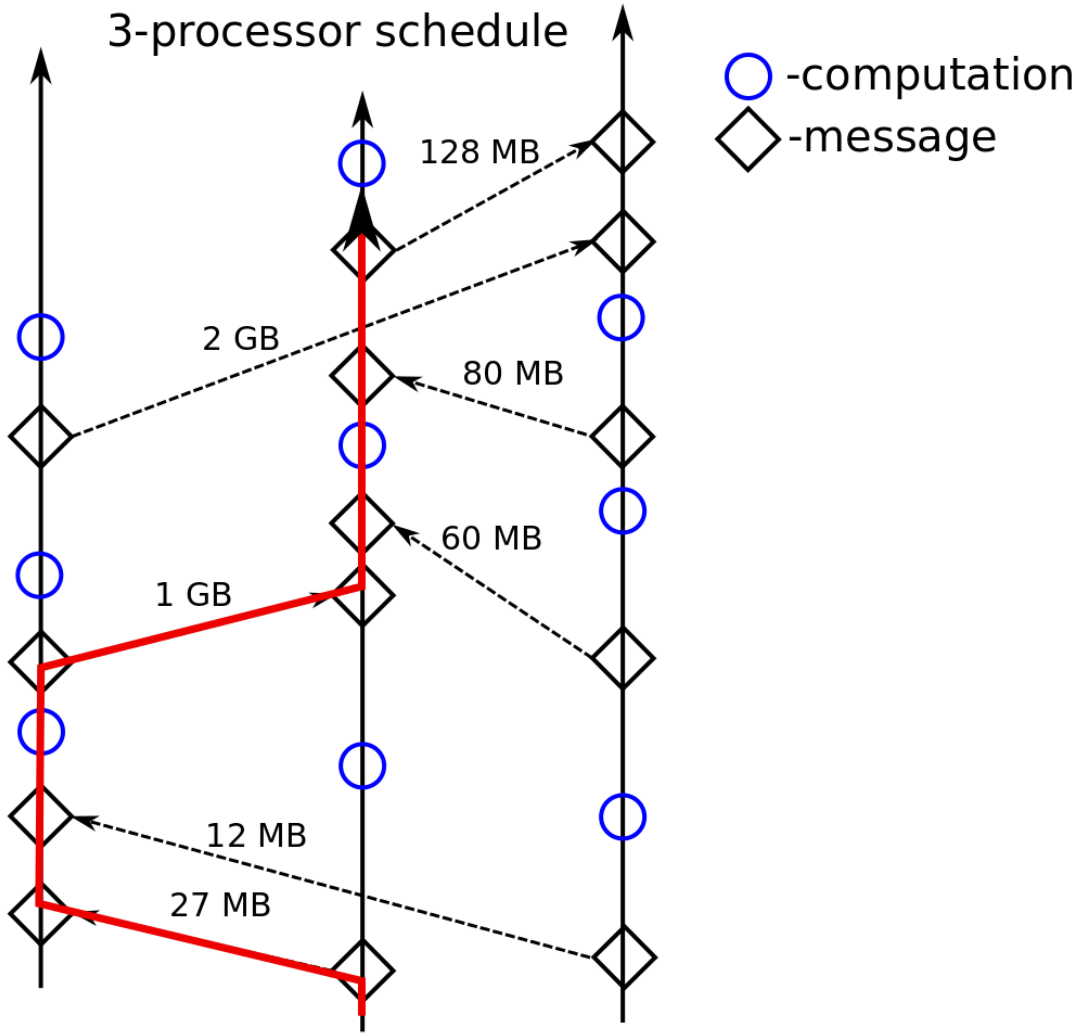


Critical path for communication cost



Critical path synchronization cost (W)

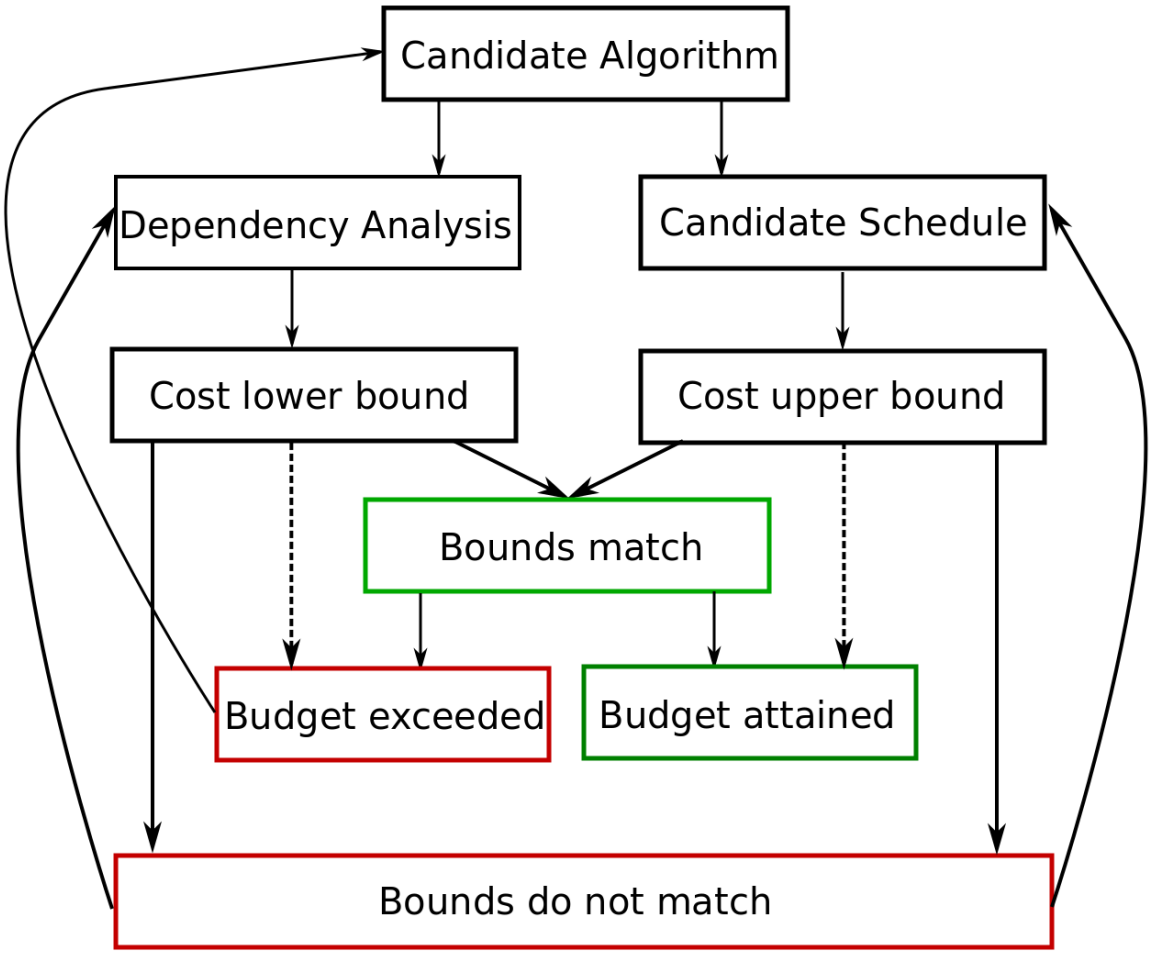
Critical path for synchronization cost



Critical path synchronization cost (S)

Designing parallel algorithms

How do we find a scalable algorithm and schedule?



Dependency graph representation of an algorithm

We can represent an algorithm as a graph $G = (V, E)$ where

- V includes the input, intermediate, and output values used by the algorithm
- E represents the dependencies between pairs of values
- e.g. to compute $c = a \cdot b$, we have $a, b, c \in V$ and $(a, c), (b, c) \in E$

Dependency graph representation of an algorithm

We can represent an algorithm as a graph $G = (V, E)$ where

- V includes the input, intermediate, and output values used by the algorithm
- E represents the dependencies between pairs of values
- e.g. to compute $c = a \cdot b$, we have $a, b, c \in V$ and $(a, c), (b, c) \in E$

A lower bound on the computation cost F is just the length longest path in the graph, $Q \subset G$, which also bounds the amount of available parallelism as $p \leq |V|/|Q|$.

Communication bandwidth lower bounds

For some algorithm $G = (V, E)$...

we can lower bound the communication cost of the schedule, by characterizing the expansion properties of the dependency graph

- We let the vertex expansion $E(G, Z)$ of a subset vertex set $Z \subset V$ be the number of vertices in $V \setminus Z$ to which Z is adjacent
- A lower bound on the amount of communication required to parallelize G on p processors is

$$W \geq \min_{Z \subset V, |Z| = |V|/p} E(Z),$$

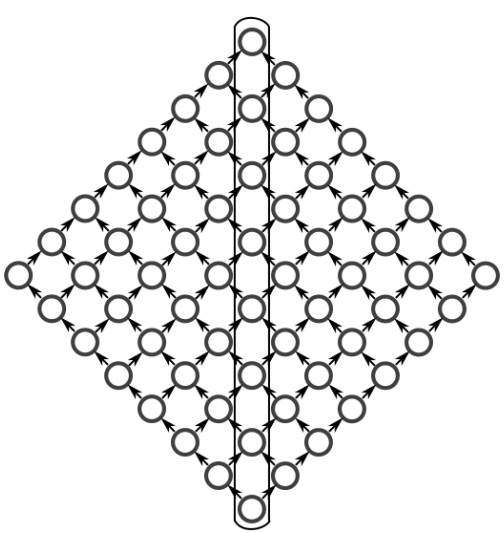
since some process must do at least $|V|/p$ work.

Path-expander graph

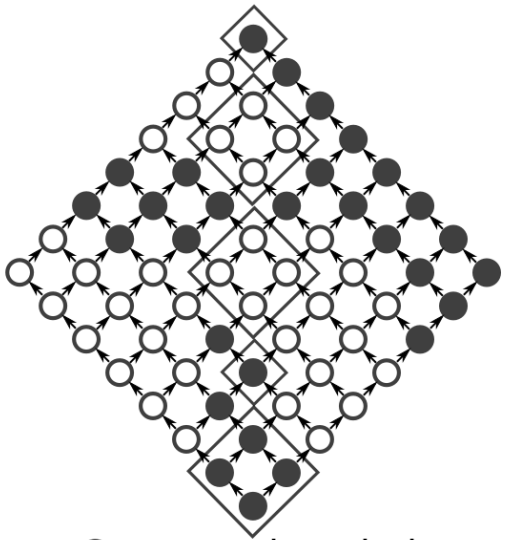
Definition (order-d-path-expander)

Graph $G = (V, E)$ is an **order-d-path-expander** if it has a path $(u_1, \ldots u_n) \subset V$, and the union of all paths between u_i and u_{i+b} for all i, b has size $\Theta(b^d)$ and a minimum cut of size $\Omega(b^{d-1})$.

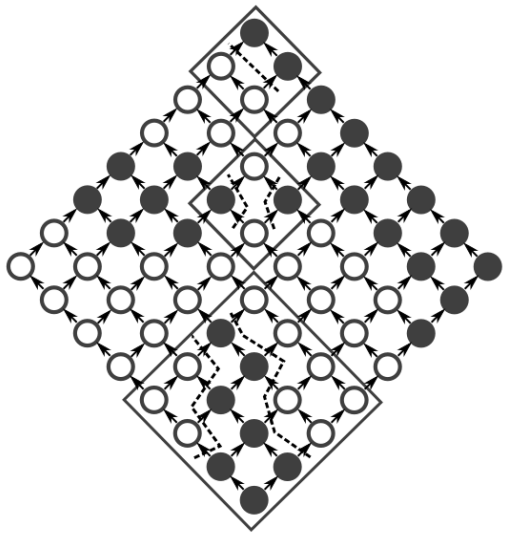
An example of a order-2-path-expander



Dependency path P



Computation chain



Communication chain

Scheduling tradeoffs of path-expander graphs

Theorem (Path-expander communication lower bound)

Any schedule of an algorithm with an **order-d-path-expander** *dependency graph about a path of length n for some $b \in [1, n]$ incurs computation (F), bandwidth (W), and latency (S) costs:*

$$F = \Omega\left(n \cdot b^{d-1}\right), \quad W = \Omega\left(n \cdot b^{d-2}\right), \quad S = \Omega\left(n/b\right),$$

which implies the following tradeoffs:

$$F \cdot S^{d-1} = \Omega\left(n^d\right), \quad W \cdot S^{d-2} = \Omega\left(n^{d-1}\right).$$

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.
With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$
[Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\text{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\text{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\text{Ch}} = \Theta(\sqrt{cp})$

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.
With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$ [Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\text{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\text{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\text{Ch}} = \Theta(\sqrt{cp})$

W_{Ch} (bandwidth) can be shown to be optimal due to its vertex expansion $E(G_{\text{Ch}}, Z) \geq |Z|^{2/3}$ for any $Z \subset V_{\text{Ch}}$.

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.
With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$ [Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\text{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\text{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\text{Ch}} = \Theta(\sqrt{cp})$

W_{Ch} (bandwidth) can be shown to be optimal due to its vertex expansion $E(G_{\text{Ch}}, Z) \geq |Z|^{2/3}$ for any $Z \subset V_{\text{Ch}}$.
S_{Ch} (synchronization) can be shown to be optimal because G_{Ch} is a (b^2, b^3)-**path-expander** about the path corresponding to the calculation of the diagonal of $\mathbf{L}$.

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix **A** of dimension n into a lower-triangular matrix **L** is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.
With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$ [Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\text{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\text{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\text{Ch}} = \Theta(\sqrt{cp})$

W_{Ch} (bandwidth) can be shown to be optimal due to its vertex expansion $E(G_{\text{Ch}}, Z) \geq |Z|^{2/3}$ for any $Z \subset V_{\text{Ch}}$.
S_{Ch} (synchronization) can be shown to be optimal because G_{Ch} is a (b^2, b^3)-**path-expander** about the path corresponding to the calculation of the diagonal of **L**.
Algorithms with the same costs exist for LU, QR, SVD, etc.

Algorithms for dense matrix factorizations

Standard algorithms for LU, QR, and SVD factorization have communication costs

$$W = O(n^2/\sqrt{p}) \quad S = O(n).$$

The algorithms from the previous slide achieved, for $c \in [1, p^{1/3}]$,

$$W = O(n^2/\sqrt{cp}) \quad S = O(\sqrt{cp}).$$

What is necessary to achieve this?

- For Cholesky, LU without pivoting, and QR via Givens rotations: change of parallel schedule

Algorithms for dense matrix factorizations

Standard algorithms for LU, QR, and SVD factorization have communication costs

$$W = O(n^2/\sqrt{p}) \quad S = O(n).$$

The algorithms from the previous slide achieved, for $c \in [1, p^{1/3}]$,

$$W = O(n^2/\sqrt{cp}) \quad S = O(\sqrt{cp}).$$

What is necessary to achieve this?

- For Cholesky, LU without pivoting, and QR via Givens rotations: change of parallel schedule
- For LU with pivoting: *change of algorithm*: pairwise pivoting or tournament pivoting instead of partial pivoting

Algorithms for dense matrix factorizations

Standard algorithms for LU, QR, and SVD factorization have communication costs

$$W = O(n^2/\sqrt{p}) \quad S = O(n).$$

The algorithms from the previous slide achieved, for $c \in [1, p^{1/3}]$,

$$W = O(n^2/\sqrt{cp}) \quad S = O(\sqrt{cp}).$$

What is necessary to achieve this?

- For Cholesky, LU without pivoting, and QR via Givens rotations: change of parallel schedule
- For LU with pivoting: *change of algorithm*: pairwise pivoting or tournament pivoting instead of partial pivoting
- For QR via Householder transformations: *change of algorithm*: 'block-Givens' (TSQR) followed by Householder reconstruction

Algorithms for dense matrix factorizations

Standard algorithms for LU, QR, and SVD factorization have communication costs

$$W = O(n^2/\sqrt{p}) \quad S = O(n).$$

The algorithms from the previous slide achieved, for $c \in [1, p^{1/3}]$,

$$W = O(n^2/\sqrt{cp}) \quad S = O(\sqrt{cp}).$$

What is necessary to achieve this?

- For Cholesky, LU without pivoting, and QR via Givens rotations: change of parallel schedule
- For LU with pivoting: *change of algorithm*: pairwise pivoting or tournament pivoting instead of partial pivoting
- For QR via Householder transformations: *change of algorithm*: 'block-Givens' (TSQR) followed by Householder reconstruction
- For SVD and the symmetric eigenproblem: *change of algorithm*: successive band reduction

Krylov subspace methods

We 'formally' consider the s-step Krylov subspace basis computation

$$\mathbf{x}^{(l)} = \mathbf{A} \cdot \mathbf{x}^{(l-1)},$$

for $l \in \{1, \ldots, s\}$ where the graph of the symmetric sparse matrix $\mathbf{A}$ is a $(2m+1)^d$-point stencil.

Krylov subspace methods

We 'formally' consider the s-step Krylov subspace basis computation

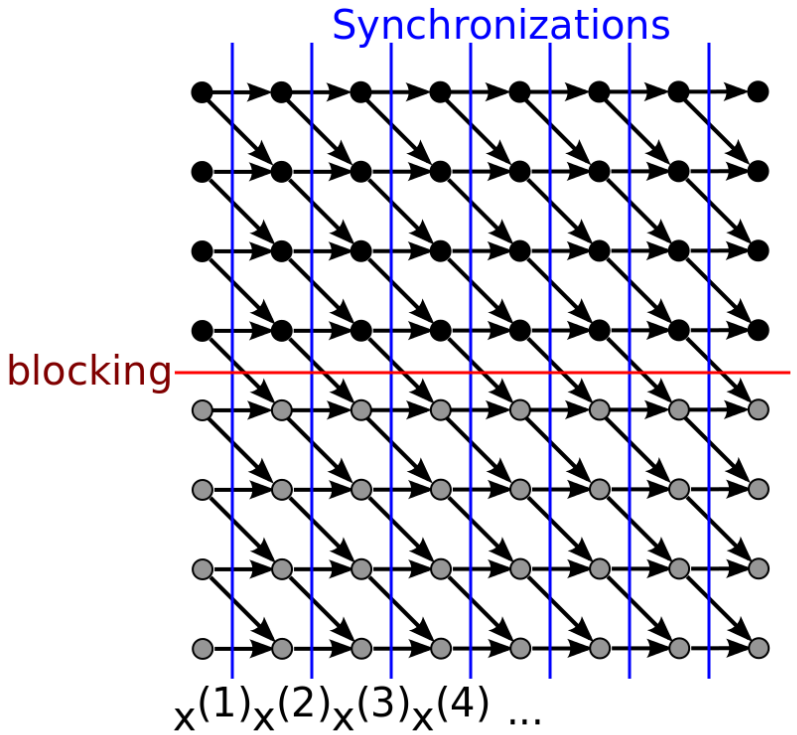
$$\mathbf{x}^{(l)} = \mathbf{A} \cdot \mathbf{x}^{(l-1)},$$

for $l \in \{1, \ldots, s\}$ where the graph of the symmetric sparse matrix $\mathbf{A}$ is a $(2m+1)^d$-point stencil.

We 'informally' consider an s-step iterative method where nodes (mesh points, particles, ...), in d-dimensional space, interact at each step with all other nodes that are within distance m.

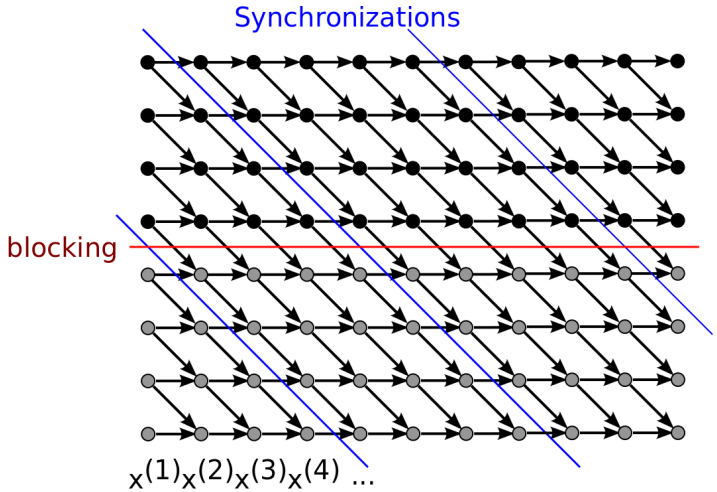
The standard algorithm (1D 2-pt stencil diagram)

Perform one matrix vector multiplication at a time, and synchronize each time



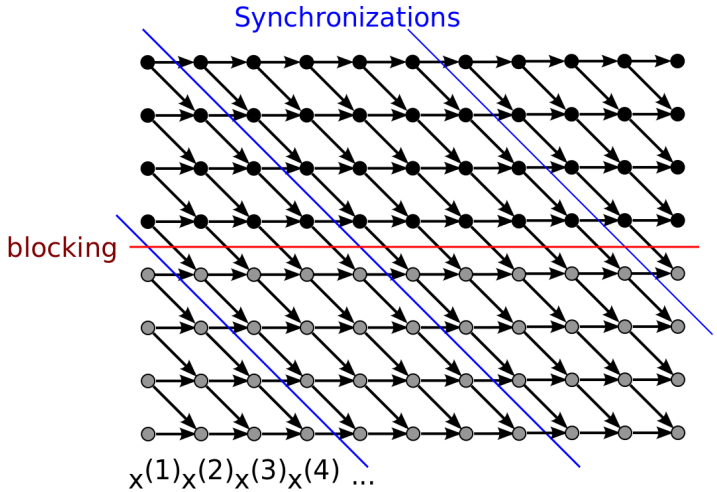
The matrix-powers kernel

Avoid synchronization by blocking across matrix-vector multiplies



The matrix-powers kernel

Avoid synchronization by blocking across matrix-vector multiplies

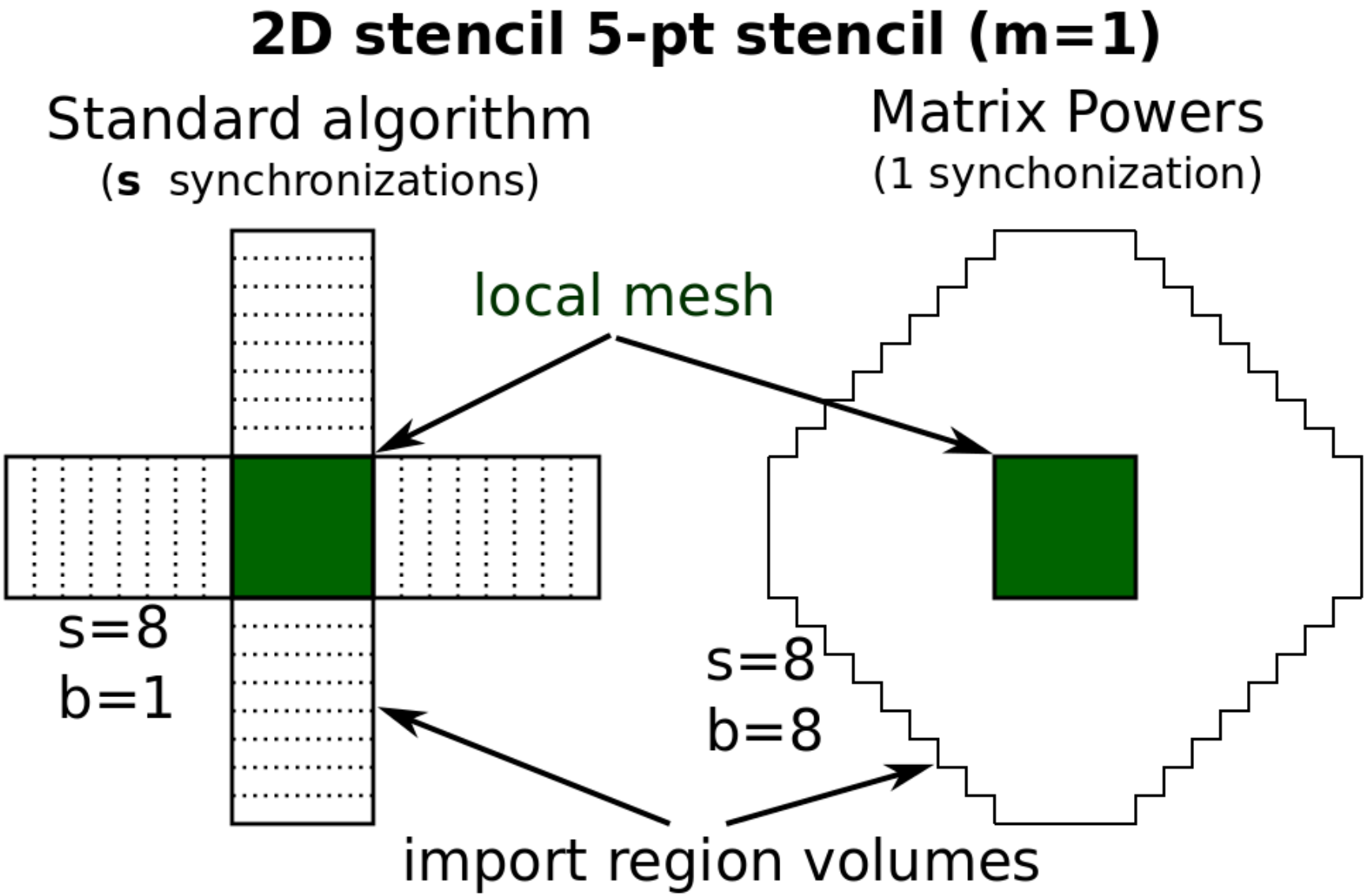


In general for a $(2m+1)^d$-point stencil, s/b invocations of the matrix-powers kernel compute an s-dimensional Krylov subspace basis with cost

$$F_{\mathrm{Kr}} = \Theta\left(m^d \cdot b^d \cdot s\right), W_{\mathrm{Kr}} = \Theta\left(m^d \cdot b^{d-1} \cdot s\right), S_{\mathrm{Kr}} = \Theta\left(s/b\right).$$

Optimal since the dependency graph of a s-step $(2m+1)^d$-point stencil is a **order-(d+1)-path-expander** with a prefactor of m^d.

Illustration of import region of the matrix-powers kernel



Summary and conclusion

High-level points summary:

- Communication cost should be the pervasive factor in the design of new numerical algorithms
- Lower bound are useful for understanding whether a better parallelization is possible or a different algorithm is necessary
- Communication cost of algorithms is dictated by dependency graph expansion properties

Collaborators and acknowledgements

Collaborators on presented work:

- Nicholas Knight, Erin Carson, James Demmel (UC Berkeley)
- Grey Ballard (formerly UC Berkeley, now Sandia National Lab)

Major collaborators on other parts of thesis work:

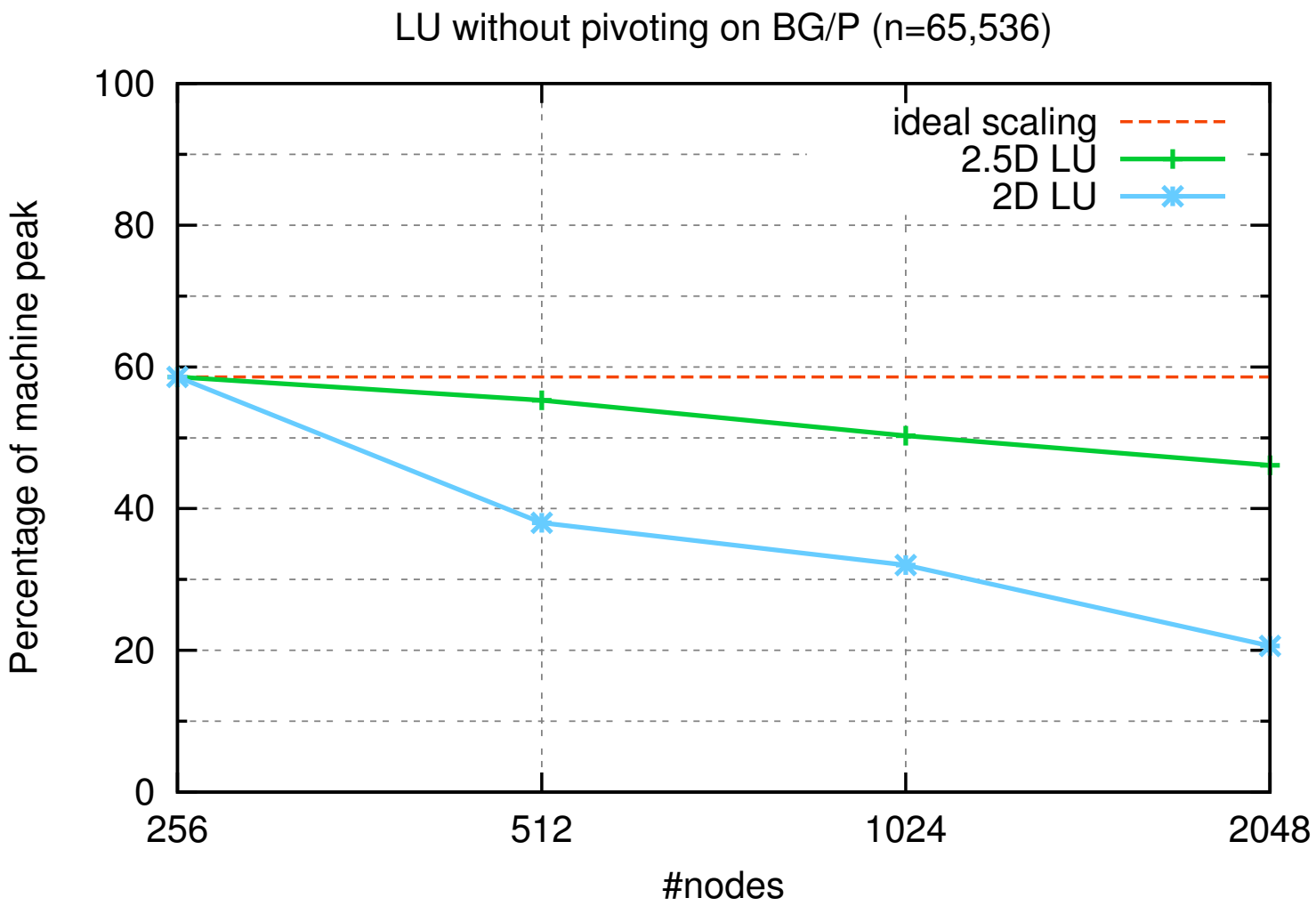
- Devin Matthews (UT Austin, CSGF fellow)
- Jeff Hammond (formerly Argonne National Lab, now Intel, CSGF alumni)
- Erik Draeger (Lawrence Livermore National Lab)
- Kathy Yelick (UC Berkeley, Lawrence Berkeley National Lab)

Support:

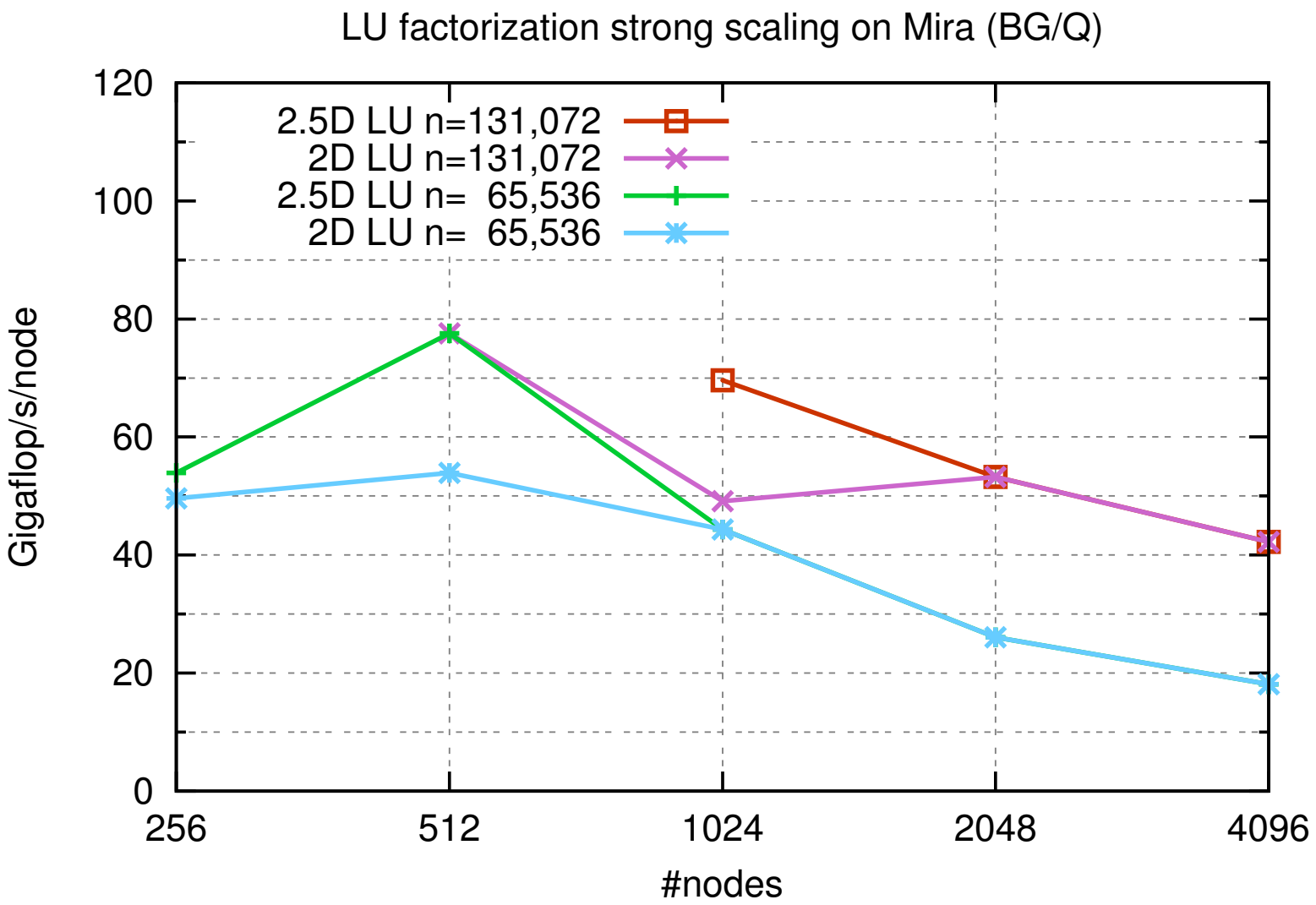
- Krell DOE Computational Science Graduate Fellowship

Backup slides

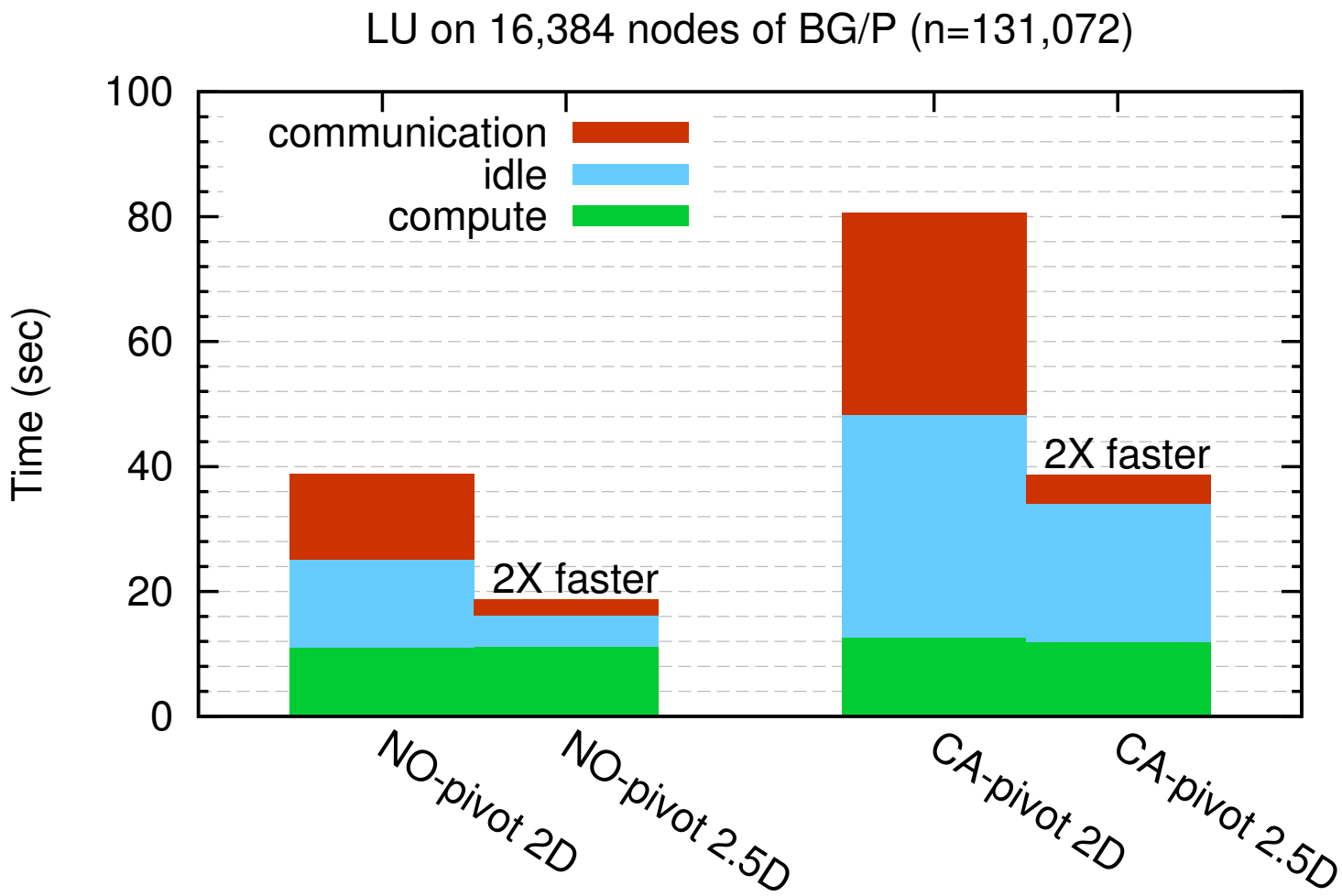
2.5D LU strong scaling (without pivoting)



Benefit of replication on BG/Q



2.5D LU on 65,536 cores



Parallel costs of Gauss-Jordan elimination

The floating point cost of Gauss-Jordan elimination is $F = \Theta(n^3/p)$. Our lower bounds may be applied since the computation has the same structure as Gaussian Elimination, so

$$F \cdot S^2 = \Omega(n^3), \quad W \cdot S = \Omega(n^2).$$

These costs are achieved for $W = O(n^2/p^{2/3})$ by schedules in

- Aggarwal, Chandra, and Snir 1990
- Tiskin 2007
- Solomonik, Buluc, and Demmel 2012

Lower synchronization cost via path doubling

We can compute the tropical semiring closure

$$\mathbf{A}^* = \mathbf{I} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \ldots \oplus \mathbf{A}^n = (\mathbf{I} \oplus \mathbf{A})^n,$$

directly via repeated squaring (path-doubling)

$$(\mathbf{I} \oplus \mathbf{A})^{2k} = (\mathbf{I} \oplus \mathbf{A})^k \otimes (\mathbf{I} \oplus \mathbf{A})^k$$

with a total of $\log(n)$ matrix-matrix multiplications, with

$$F = O(n^3 \log(n)/p)$$

operations and $O(\log(n))$ synchronizations, which can be less than the $O(p^{1/2})$ required by Floyd-Warshall.

Tiskin's path doubling algorithm

Tiskin gives a way to do path-doubling in $F = O(n^3/p)$ operations. We can partition each $\mathbf{A}^k$ by path size (number of edges)

$$\mathbf{A}^k = \mathbf{I} \oplus \mathbf{A}^k(1) \oplus \mathbf{A}^k(2) \oplus \ldots \oplus \mathbf{A}^k(k)$$

where each $\mathbf{A}^k(l)$ contains the shortest paths of up to $k \geq l$ edges, which have exactly l edges. We can see that

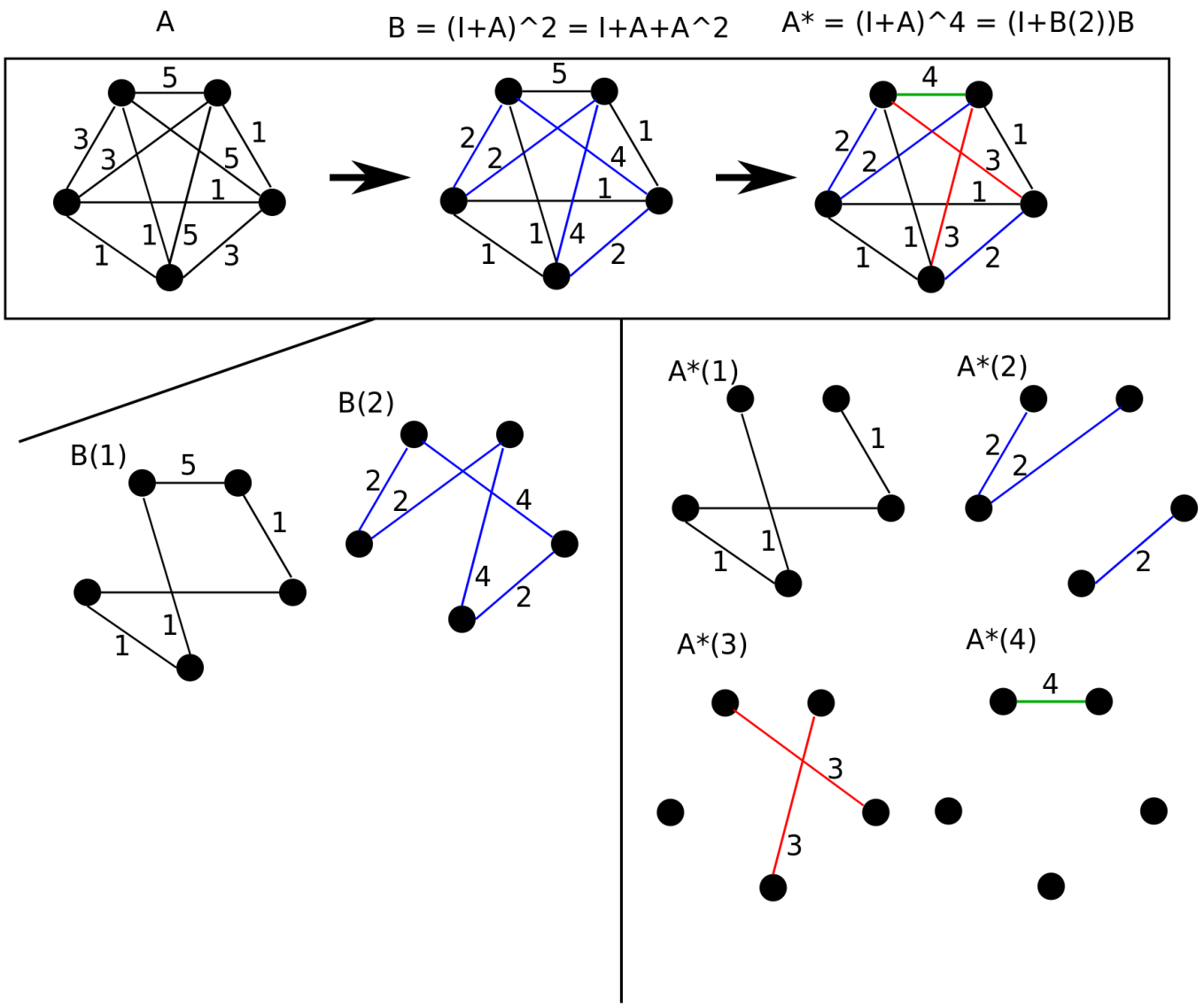
$$\mathbf{A}^l(l) \leq \mathbf{A}^{l+1}(l) \leq \ldots \leq \mathbf{A}^n(l) = \mathbf{A}^*(l),$$

in particular $\mathbf{A}^*(l)$ corresponds to a sparse subset of $\mathbf{A}^l(l)$. The algorithm works by picking $l \in [k/2, k]$ and computing

$$(\mathbf{I} \oplus \mathbf{A})^{3k/2} \leq (\mathbf{I} \oplus \mathbf{A}^k(l)) \otimes \mathbf{A}^k,$$

which finds all paths of size up to $3k/2$ by taking all paths of size exactly $l \geq k/2$ followed by all paths of size up to k.

Path-doubling (Tiskin's algorithm)



Path-doubling

Earlier caveat:

$$(\mathbf{I} \oplus \mathbf{A})^{3k/2} \leq (\mathbf{I} \oplus \mathbf{A}^k(l)) \otimes \mathbf{A}^k,$$

does not hold in general. The fundamental property used by the algorithm is really

$$\mathbf{A}^*(l) \otimes \mathbf{A}^*(k) = \mathbf{A}^*(l + k).$$

All shortest paths of up to any length are composible (factorizable), but not paths up to a limited length. However, the algorithm is correct because $\mathbf{A}^l \leq \mathbf{A}^k(l) \leq \mathbf{A}^*(k)$.

Cost of Tiskin's algorithm

Since the decomposition by path size is disjoint, one can pick $\mathbf{A}^k(l)$ for $l \in [k/2, k]$ to have size

$$|\mathbf{A}^k(l)| \geq 2n^2/k.$$

Each round of path doubling becomes cheaper than the previous, so the cost is dominated by the first matrix multiplication,

$$F = O(n^3/p) \quad W = O(n^2/p^{2/3}) \quad S = O(\log(n)),$$

solving the APSP problem with no $F \cdot S^2$ or $W \cdot S$ tradeoff and optimal flops.

More on Tiskin's APSP algorithms

Tiskin gives a way to lower the synchronization from $S = O(\log(n))$ to $O(\log(p))$. For nonnegative edge lengths it is straightforward

- compute $\mathbf{A}^p$ via path-doubling
- pick a small $\mathbf{A}^p(l)$ for $l \in [p/2, p]$
- replicate $\mathbf{A}^p(l)$ and compute Dijkstra's algorithm for n/p nodes with each process, obtaining $(\mathbf{A}^p(l))^*$
- compute by matrix multiplication

$$\mathbf{A}^* = (\mathbf{A}^p(l))^* \otimes \mathbf{A}^p$$

since all shortest paths are composed of a path of size that is a multiple of $l \leq p$, followed by a shortest path of size up to p