Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations

Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel

Department of EECS, UC Berkeley

Symposium of Parallel Algorithms and Architectures (SPAA) 2014
Prague, Czech Republic

June 25, 2014

Summary

Talk overview

- Introduction of our distributed-memory cost model

Summary

Talk overview

- Introduction of our distributed-memory cost model
- Motivation via dense linear algebra algorithms

Summary

Talk overview

- Introduction of our distributed-memory cost model
- Motivation via dense linear algebra algorithms
- Presentation of a dependency graph analysis technique

Summary

Talk overview

- Introduction of our distributed-memory cost model
- Motivation via dense linear algebra algorithms
- Presentation of a dependency graph analysis technique
- Tradeoffs/lower-bounds for dense linear algebra algorithms

Summary

Talk overview

- Introduction of our distributed-memory cost model
- Motivation via dense linear algebra algorithms
- Presentation of a dependency graph analysis technique
- Tradeoffs/lower-bounds for dense linear algebra algorithms
- Synchonization-communication tradeoffs for sparse methods

Summary

Talk overview

- Introduction of our distributed-memory cost model
- Motivation via dense linear algebra algorithms
- Presentation of a dependency graph analysis technique
- Tradeoffs/lower-bounds for dense linear algebra algorithms
- Synchonization-communication tradeoffs for sparse methods

Topics omitted in talk but present in paper

- Reduction from dependency graphs to hypergraphs
- Lower bounds on balanced hypergraph cuts
- Various other proof details and technicalities

Representation of an algorithm

We cannot derive communication lower bounds for problems directly, but only specific algorithms

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

Representation of an algorithm

We cannot derive communication lower bounds for problems directly, but only specific algorithms

- We can represent an algorithm as a graph $G = (V, E)$ where
  - V includes the input, intermediate, and output values used by the algorithm

Representation of an algorithm

We cannot derive communication lower bounds for problems directly, but only specific algorithms

- 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

Representation of an algorithm

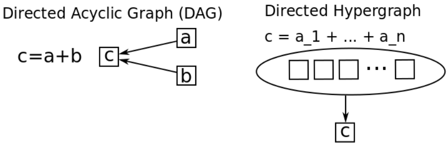
We cannot derive communication lower bounds for problems directly, but only specific algorithms

- 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 $(a, c), (b, c) \in E$

Representation of an algorithm

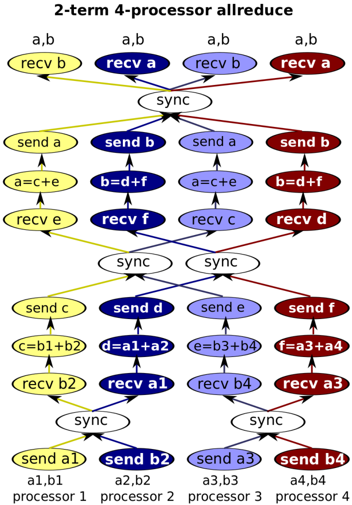
We cannot derive communication lower bounds for problems directly, but only specific algorithms

- 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 $(a, c), (b, c) \in E$
- Reduction trees may be abstracted away as hypergraph edges



Representation of a parallel schedule

- asynchronous point-to-point communication

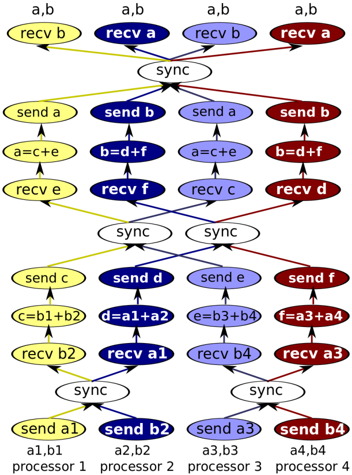


Schedule for $(a, b) = \sum_i (a_i, b_i)$

Representation of a parallel schedule

- asynchronous point-to-point communication
- progress must be guaranteed via synchronization

2-term 4-processor allreduce

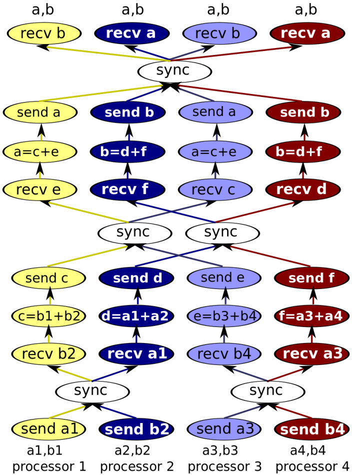


Schedule for $(a, b) = \sum_i (a_i, b_i)$

Representation of a parallel schedule

- asynchronous point-to-point communication
- progress must be guaranteed via synchronization
- synchronization can be done collectively

2-term 4-processor allreduce

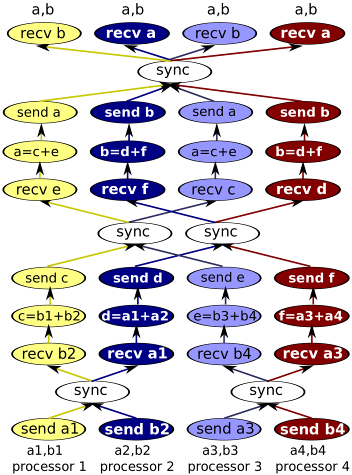


Schedule for $(a, b) = \sum_i (a_i, b_i)$

Representation of a parallel schedule

- asynchronous point-to-point communication
- progress must be guaranteed via synchronization
- synchronization can be done collectively
- cost given by critical (most expensive) path

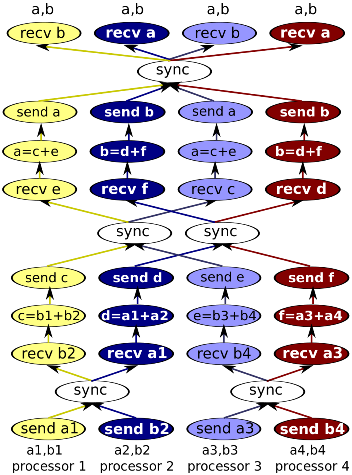
2-term 4-processor allreduce



Schedule for $(a, b) = \sum_i (a_i, b_i)$

- asynchronous point-to-point communication
- progress must be guaranteed via synchronization
- synchronization can be done collectively
- cost given by critical (most expensive) path
- efficiently simulates BSP algorithms

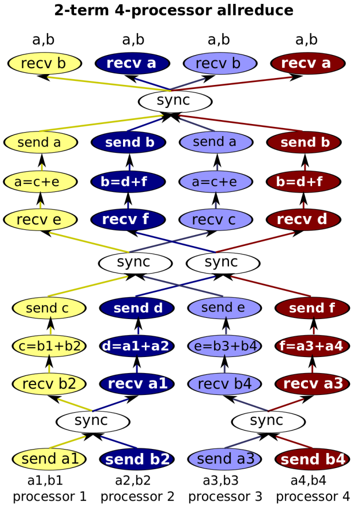
2-term 4-processor allreduce



Schedule for $(a, b) = \sum_i (a_i, b_i)$

Representation of a parallel schedule

- asynchronous point-to-point communication
- progress must be guaranteed via synchronization
- synchronization can be done collectively
- cost given by critical (most expensive) path
- efficiently simulates BSP algorithms
- efficiently simulates LogP algorithms when $L \approx o$



Schedule for $(a, b) = \sum_i (a_i, b_i)$

Cost model

We quantify interprocessor communication and synchronization costs of a parallelization via a flat network model

- γ - cost for a single computation (flop)
- β - cost for a transfer of each byte between any pair of processors
- α - cost for a synchronization between any pair of processors

Cost model

We quantify interprocessor communication and synchronization costs of a parallelization via a flat network model

- γ - cost for a single computation (flop)
- β - cost for a transfer of each byte between any pair of processors
- α - cost for a synchronization between any pair of processors

We measure the cost of a parallelization along the longest sequence of dependent computations and data transfers (critical path)

- F - critical path payload for computation cost
- W - critical path payload for communication (bandwidth) cost
- S - critical path payload for synchronization cost

Solving a dense triangular system

For lower triangular dense matrix $\mathbf{L}$ and vector $\mathbf{y}$ of dimension n, solve for $\mathbf{x}$ in

$$\mathbf{L} \cdot \mathbf{x} = \mathbf{y}.$$

Solving a dense triangular system

For lower triangular dense matrix $\mathbf{L}$ and vector $\mathbf{y}$ of dimension n, solve for $\mathbf{x}$ in

$$\mathbf{L} \cdot \mathbf{x} = \mathbf{y}.$$

Parallel algorithms for the triangular solve

- wavefront algorithms [Heath 1988]
- diamond DAG algorithms and lower bounds given by [Papadimitriou and Ullman 1987] and [Tiskin 1998]

Solving a dense triangular system

For lower triangular dense matrix $\mathbf{L}$ and vector $\mathbf{y}$ of dimension n, solve for $\mathbf{x}$ in

$$\mathbf{L} \cdot \mathbf{x} = \mathbf{y}.$$

Parallel algorithms for the triangular solve

- wavefront algorithms [Heath 1988]
- diamond DAG algorithms and lower bounds given by [Papadimitriou and Ullman 1987] and [Tiskin 1998]

For $p \in [1, n]$ processors, these algorithms have costs

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

Tradeoff between computation ($\Downarrow$ with p) and synchronization cost ($\Uparrow$ with p).

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n is

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

for a lower-triangular matrix $\mathbf{L}$.

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n is

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

for a lower-triangular matrix $\mathbf{L}$.

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

Tradeoffs:

- synchronization $\Uparrow$ with p, bandwidth and computation costs $\Downarrow$
- synchronization $\Uparrow$ with c, bandwidth cost $\Downarrow$

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n is

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

for a lower-triangular matrix $\mathbf{L}$.
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: $F_{\text{Ch}} = O(n^3/p)$
- bandwidth: $W_{\text{Ch}} = O(n^2/\sqrt{cp})$
- synchronization: $S_{\text{Ch}} = O(\sqrt{cp})$

Tradeoffs:

- synchronization $\Uparrow$ with p, bandwidth and computation costs $\Downarrow$
- synchronization $\Uparrow$ with c, bandwidth cost $\Downarrow$

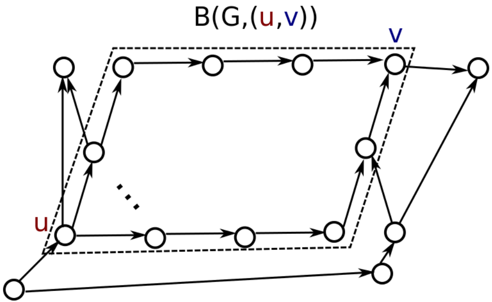
Algorithms with the same asymptotic costs also exist for LU with pairwise and tournament pivoting as well as for QR factorization, the symmetric eigenproblem, and the SVD

Dependency bubble

To prove these tradeoffs are unavoidable, we analyze interdependent computations (bubbles) in the dependency graphs of these algorithms

Definition (Dependency bubble)

Given two vertices u, v in a directed acyclic graph $G = (V, E)$, the dependency bubble $B(G, (u, v))$ is the union of all paths in G from u to v.

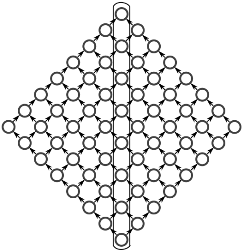


Path-expander graph

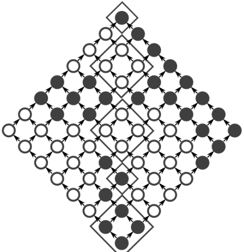
Definition ((ϵ, σ)-path-expander)

Graph $G = (V, E)$ is a (ϵ, σ)-**path-expander** if there exists a path $(u_1, \dots u_n) \subset V$, such that the dependency bubble $B(G, (u_i, u_{i+b}))$ for each i, b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.

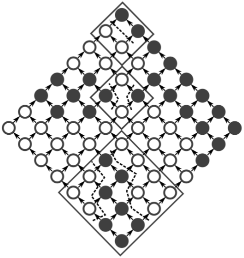
An example of a (b, b^2)-**path-expander**



Dependency path P



Computation chain



Communication chain

Scheduling tradeoffs of path-expander graphs

Theorem (Path-expander communication lower bound)

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

$$F = \Omega\left(\sigma(b) \cdot n/b\right), \quad W = \Omega\left(\epsilon(b) \cdot n/b\right), \quad S = \Omega\left(n/b\right).$$

Corollary

If $\sigma(b) = b^d$ *and* $\epsilon(b) = b^{d-1}$*, the above theorem yields,*

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

Tradeoffs for triangular solve

Theorem

Any parallelization of any dependency graph $G_{\mathrm{TRSV}}(n)$ incurs the following computation (F), bandwidth (W), and latency (S) costs, for some $b \in [1, n]$,

$$F_{\mathrm{TRSV}} = \Omega\left(n \cdot b\right), \qquad W_{\mathrm{TRSV}} = \Omega\left(n\right), \qquad S_{\mathrm{TRSV}} = \Omega\left(n/b\right),$$

and furthermore, $F_{\mathrm{TRSV}} \cdot S_{\mathrm{TRSV}} = \Omega\left(n^2\right)$.

Proof.

Proof by application of path-based tradeoffs since $G_{\mathrm{TRSV}}(n)$ is a (b, b^2)-**path-expander** dependency graph. □

With $p = n/b$ processors, we've now established,

$$F_{\mathrm{TRSV}} = \Theta(n^2/p), \qquad W_{\mathrm{TRSV}} = \Theta(n), \qquad S_{\mathrm{TRSV}} = \Theta(p)$$

Tradeoffs for Cholesky

Theorem

Any parallelization of any dependency graph $G_{\mathrm{Ch}}(n)$ incurs the following computation (F), bandwidth (W), and latency (S) costs, for some $b \in [1, n]$,

$$F_{\mathrm{Ch}} = \Omega\left(n \cdot b^2\right), \qquad W_{\mathrm{Ch}} = \Omega\left(n \cdot b\right), \qquad S_{\mathrm{Ch}} = \Omega\left(n/b\right),$$

and furthermore, $F_{\mathrm{Ch}} \cdot S_{\mathrm{Ch}}^2 = \Omega\left(n^3\right), \quad W_{\mathrm{Ch}} \cdot S_{\mathrm{Ch}} = \Omega\left(n^2\right).$

Proof.

Proof shows that $G_{\mathrm{Ch}}(n)$ is a (b^2, b^3)-**path-expander** about the path corresponding to the calculation of the diagonal of $\mathbf{L}$. □

Therefore, with $p \in [1, n^{3/2}]$ processors and $c \in [1, p^{1/3}]$,

$$F_{\mathrm{Ch}} = \Theta(n^3/p), \quad W_{\mathrm{Ch}} = \Theta(n^2/\sqrt{cp}), \quad S_{\mathrm{Ch}} = \Theta(\sqrt{cp})$$

Krylov subspace methods

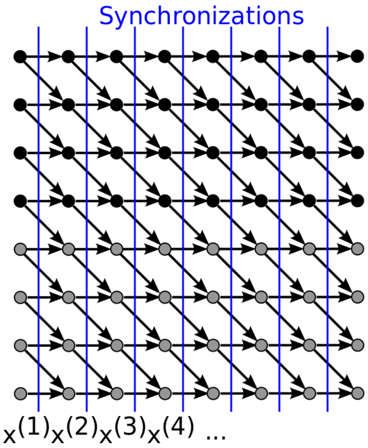
We consider the s-step Krylov subspace basis computation

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

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

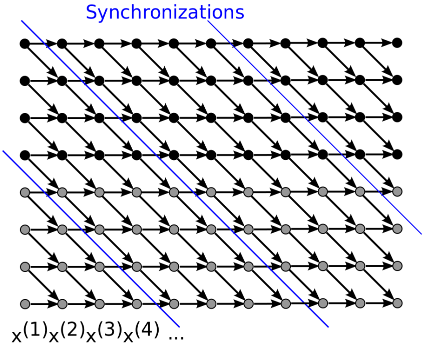
The standard algorithm (1D 2-pt stencil diagram)

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

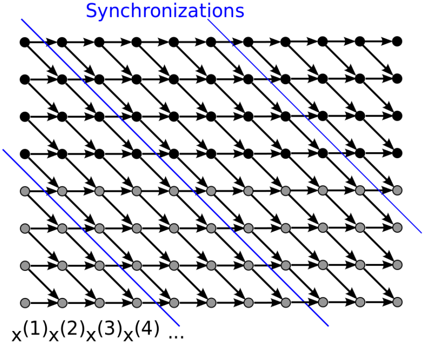


The matrix-powers kernel

Avoid synchronization by blocking across matrix-vector multiplies



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

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

Communication lower bounds for Krylov subspace methods

Theorem

Any parallel execution of an s-step Krylov subspace basis computation for a $(2m+1)^d$-point stencil on a regular mesh, requires the following computation, bandwidth, and latency costs for some $b \in \{1, \ldots s\}$,

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

and furthermore,

$$F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right), \quad W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right).$$

This lower bound is tight with respect to the matrix-powers kernel when $n^d/p \geq m^d \cdot b^d$, where n^d is the number of mesh points.

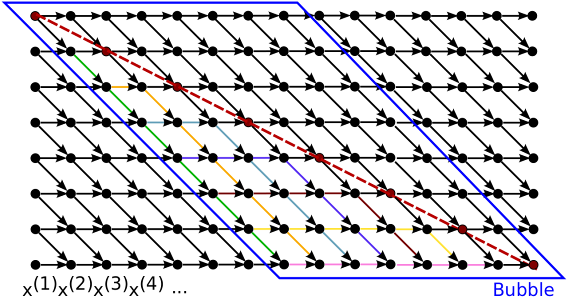
Proof of tradeoffs for Krylov subspace methods

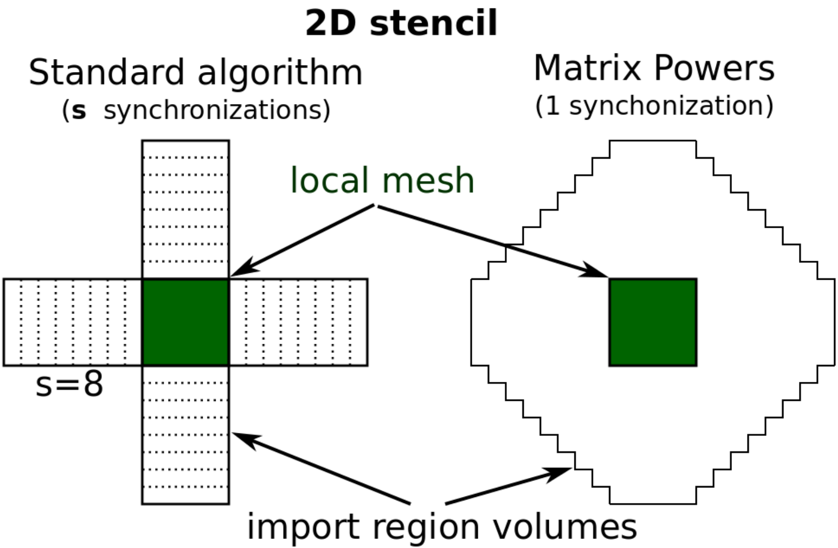
Proof.

Done by showing that the dependency graph of a s-step $(2m+1)^d$-point stencil is a $(m^d b^d, m^d b^{d+1})$-**path-expander**. □

sample graph for 2-point 1-dimensional stencil

(ignoring one direction of dependencies with respect to 3-point stencil)




2D stencil
Standard algorithm
(s synchronizations)
Matrix Powers
(1 synchonization)
local mesh
s=8
import region volumes

Summary and conclusion

Novel lower bounds on cost tradeoffs

- Cholesky factorization
  $F_{\mathrm{Ch}} \cdot S_{\mathrm{Ch}}^2 = \Omega\left(n^3\right)$ and $W_{\mathsf{Ch}} \cdot S_{\mathsf{Ch}} = \Omega(n^2)$
- s-step Krylov subspace methods on $(2m+1)^d$-pt stencils
  $F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right)$ and $W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right)$

Summary and conclusion

Novel lower bounds on cost tradeoffs

- Cholesky factorization
  $F_{\mathrm{Ch}} \cdot S_{\mathrm{Ch}}^2 = \Omega\left(n^3\right)$ and $W_{\mathsf{Ch}} \cdot S_{\mathsf{Ch}} = \Omega(n^2)$
- s-step Krylov subspace methods on $(2m+1)^d$-pt stencils
  $F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right)$ and $W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right)$

Extensions to graph algorithms

- Floyd-Warshall is analogous to Cholesky factorization
- Bellman-Ford is analogous to Krylov subspace methods
- Future work is to analyze other (e.g. power-law) graphs

Summary and conclusion

Novel lower bounds on cost tradeoffs

- Cholesky factorization
  $F_{\mathrm{Ch}} \cdot S_{\mathrm{Ch}}^2 = \Omega\left(n^3\right)$ and $W_{\mathsf{Ch}} \cdot S_{\mathsf{Ch}} = \Omega(n^2)$
- s-step Krylov subspace methods on $(2m+1)^d$-pt stencils
  $F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right)$ and $W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right)$

Extensions to graph algorithms

- Floyd-Warshall is analogous to Cholesky factorization
- Bellman-Ford is analogous to Krylov subspace methods
- Future work is to analyze other (e.g. power-law) graphs

However, there exist alternative work-efficient algorithms for some of these problems that do $O(\log(p))$ synchronizations

- Matrix inversion [Csanky 1976] (but numerically unstable)
- APSP [Tiskin 2001]

The all-pairs shortest-paths problem

Given a weighted graph $G = (V, E)$ with n vertices and a corresponding adjacency matrix $\mathbf{A}$, we seek to find the shortest paths between all pairs of vertices in G

- seek the closure, $\mathbf{A}^*$, of $\mathbf{A}$ over the tropical semiring
  - $c = c \oplus a \otimes b$ on the tropical semiring implies $c = \min(c, a + b)$
  - the identity matrix $\mathbf{I}$ on the tropical semiring is 0 on the diagonal and ∞ everywhere else
  - $\mathbf{A}^* = \mathbf{I} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \ldots \oplus \mathbf{A}^n = (\mathbf{I} \oplus \mathbf{A})^n$
  - on the sum-product ring $\mathbf{A}^* = (\mathbf{I} - \mathbf{A})^{-1}$
- on the tropical semiring it is commonly computed by the Floyd-Warshall algorithm with $W \cdot S = \Theta(n^2)$
- it is also possible to compute $\mathbf{A}^*$ via $\log n$ steps of repeated squaring (path doubling)

Tiskin's all-pairs shortest-paths 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.