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

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

Department of EECS, UC Berkeley

February, 2014
A **problem** asks for a solution which satisfies a set of properties with respect to a set of inputs (e.g. given $A$, find triangular matrices $L$ and $U$ such that $A = L \cdot U$).
A **problem** asks for a solution which satisfies a set of properties with respect to a set of inputs (e.g. given $A$, find triangular matrices $L$ and $U$ such that $A = L \cdot U$)

An **algorithm** is a sequence of computer operations that determines a solution to the problem
A **problem** asks for a solution which satisfies a set of properties with respect to a set of inputs (e.g. given $A$, find triangular matrices $L$ and $U$ such that $A = L \cdot U$).

An **algorithm** is a sequence of computer operations that determines a solution to the problem.

A **parallelization** is a schedule of the algorithm, which partitions the operations amongst processors and defines the necessary data movement between processors.
Graphical representation of a parallel 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 \( (a, c), (b, c) \in E \)
- We can represent a set of algorithms by working with hypergraph representations \( H = (V, \bar{E}) \)
  - \( \bar{E} \) may represent the dependency of a value on a set of vertices (e.g. reduction tree)
  - e.g. to compute \( d = \sum_{i=1}^{n} c_i \), we have \( d, c_i \in V \) and hyperedges \((\{c_1, \ldots, c_n\}, \{d\}) \in \bar{E}\)
Our goal will be to bound the payload of any parallel schedule for given algorithms.
Our goal will be to bound the payload of any parallel schedule for given algorithms. The schedule must give a unique assignment/partitioning of vertices amongst \( p \) processors.

\[ V = \bigcup_{i=1}^{p} C_i \]
Parallelization schedules

- Our goal will be to bound the payload of any parallel schedule for given algorithms.
- The schedule must give a unique assignment/partitioning of vertices amongst \( p \) processors.

\[
V = \bigcup_{i=1}^{p} C_i
\]

- The schedule should give a sequence of \( m_i \) computation and communication operations.
Our goal will be to bound the payload of any parallel schedule for given algorithms.

The schedule must give a unique assignment/partitioning of vertices amongst $p$ processors:

$$V = \bigcup_{i=1}^{p} C_i$$

The schedule should give a sequence of $m_i$ computation and communication operations:

- $F_{ij}$ is the set of values computed by processor $i$ at timestep $j$.
Our goal will be to bound the payload of any parallel schedule for given algorithms.

The schedule must give a unique assignment/partitioning of vertices amongst \( p \) processors:

\[
V = \bigcup_{i=1}^{p} C_i
\]

The schedule should give a sequence of \( m_i \) computation and communication operations:

- \( F_{ij} \) is the set of values computed by processor \( i \) at timestep \( j \)
- \( R_{ij} \) is the set of values received by processor \( i \) at timestep \( j \)
Our goal will be to bound the payload of any parallel schedule for given algorithms.

The schedule must give a unique assignment/partitioning of vertices amongst \( p \) processors.

\[ V = \bigcup_{i=1}^{p} C_i \]

The schedule should give a sequence of \( m_i \) computation and communication operations:

- \( F_{ij} \) is the set of values computed by processor \( i \) at timestep \( j \)
- \( R_{ij} \) is the set of values received by processor \( i \) at timestep \( j \)
- \( M_{ij} \) is the set of values sent by processor \( i \) at timestep \( j \)
We quantify interprocessor communication and synchronization costs of a parallelization via a flat network model.

- $\gamma$ - cost for a single computation (flop)
- $\beta$ - cost for a transfer of each byte between any pair of processors
- $\alpha$ - 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
Parallel schedule example

Processor schedule

- computation
- message

F13
S12
F12
S11
R11
F11

2 GB
128 MB
60 MB
1 GB
27 MB

Edgar Solomonik
Synchronization lower bounds
6/30
Critical path for computational cost

![Diagram showing the critical path for computational cost with nodes labeled as 1 GF, 2 GF, and 5 GF connected by edges labeled as computation and message.]
Critical path for synchronization cost

Synchronization payload

-message

7 * alpha
The duality of upper and lower bounds

Given an algorithm for a problem

- A parallelization provides a schedule whose payload is an upper bound on the cost of the algorithm.
The duality of upper and lower bounds

Given an algorithm for a problem

- A parallelization provides a schedule whose payload is an upper bound on the cost of the algorithm.
- A parallelization of an algorithm's optimal under a given cost model, if there exists a matching lower bound on the cost of the algorithm.
The duality of upper and lower bounds

Given an algorithm for a problem

- A parallelization provides a schedule whose payload is an upper bound on the cost of the algorithm.
- A parallelization of an algorithms optimal under a given cost model, if there exists a matching lower bound on the cost of the algorithm.
- We will first give some parallel algorithms (upper bounds) and then show their optimality via a new technique.
The duality of upper and lower bounds

Given an **algorithm** for a problem

- A **parallelization** provides a schedule whose payload is an **upper bound** on the cost of the algorithm.
- A parallelization of an algorithms optimal under a given cost model, if there exists a matching **lower bound** on the cost of the algorithm.
- We will first give some parallel algorithms (upper bounds) and then show their optimality via a new technique.
- The focus of our technique will be on lower bounds for synchronization cost, which manifest themselves as tradeoffs.
Problem: For lower triangular dense matrix $L$ and vector $y$ of dimension $n$, solve

$$L \cdot x = y,$$

i.e., $\sum_{j=1}^{i} L_{ij} \cdot x_j = y_i$, for $i \in \{1, \ldots, n\}$.

Algorithm:

$$x = \text{TRSV}(L, y, n)$$

1. for $i = 1$ to $n$
2. for $j = 1$ to $i - 1$
3. $Z_{ij} = L_{ij} \cdot x_j$
4. $x_i = \left(y_i - \sum_{j=1}^{i-1} Z_{ij}\right) / L_{ii}$

This algorithm does not immediately yield a graph representation since a summation order is not defined, but we can infer a hypergraph dependency representation.
Dependency Hypergraph: Triangular solve

\[
\begin{bmatrix}
y_1 & x_1 \\
y_2 & z_{21} & x_2 \\
y_3 & z_{31} & z_{32} & x_3 \\
y_4 & z_{41} & z_{42} & z_{43} & x_4 \\
y_5 & z_{51} & z_{52} & z_{53} & z_{54} & x_5 \\
\end{bmatrix}
\]
One instance of the above triangular solve algorithm is a diamond DAG

- wavefront algorithms [Heath 1988]
- also algorithms given by [Papadimitriou and Ullman 1987] and [Tiskin 1998]

These Diamond DAG schedules achieve the following costs for $\mathbf{x}$ of dimension $n$, and some number of processors $p \in [1, n]$

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

We see a tradeoff between computational and synchronization cost.
Obtaining a Cholesky factorization

**Problem:** The Cholesky factorization of a symmetric positive definite matrix $A$ is

$$A = L \cdot L^T,$$

for a lower-triangular matrix $L$.

**Algorithm:**

1. $L = \text{CHOLESKY}(A, n)$
2. for $j = 1$ to $n$
   2.1. $L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk} \cdot L_{jk}}$
3. for $i = j + 1$ to $n$
   3.1. for $k = 1$ to $j - 1$
       3.1.1. $Z_{ijk} = L_{ik} \cdot L_{jk}$
   3.2. $L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} Z_{ijk}) / L_{jj}$

Again this algorithm yields only a hypergraph dependency representation.
These diagrams show (a) the vertices $Z_{ijk}$ in $V_{\text{GE}}$ with $n = 16$ and (b) the hyperplane $x_{12}$ and hyperedge $e_{12,6}$ on $H_{\text{GE}}$. 
Parallelizations of Cholesky

For $\mathbf{A}$ of dimension $n$, with $p$ processors, and some $c \in [1, p^{1/3}]$ [Tiskin 2002] in BSP and 2.5D algorithms [ES, JD 2011] achieve the costs

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

We see a tradeoff between computational and synchronization cost, as well as a tradeoff between bandwidth and synchronization costs (the latter independent of the number of processors, $p$).

Algorithms with the same asymptotic costs also exist for LU with pairwise or with tournament pivoting as well as for QR factorization.
Dependency bubble

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 $v$. 

![Diagram of a directed acyclic graph with vertices $u$ and $v$, and the dependency bubble $B(G, (u, v))$]
**Definition ((\(\epsilon, \sigma\))-path-expander)**

Graph \(G = (V, E)\) is a \((\epsilon, \sigma)\)-path-expander if there exists a path \((u_1, \ldots, u_n) \subseteq V\), such that the dependency bubble \(B(G, (u_i, u_{i+b}))\) has size \(\Theta(\sigma(b))\) and a minimum cut of size \(\Omega(\epsilon(b))\).
Theorem (Path-expander communication lower bound)

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

\[
F = \Omega \left( \sigma(b) \cdot \frac{n}{b} \right), \quad W = \Omega \left( \epsilon(b) \cdot \frac{n}{b} \right), \quad S = \Omega \left( \frac{n}{b} \right).
\]
Demonstration of proof for \((b, b^2)\)-path-expander
Theorem

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

$$F_{TRSV} = \Omega (n \cdot b), \quad W_{TRSV} = \Omega (n), \quad S_{TRSV} = \Omega (n/b),$$

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

Proof.

Proof by application of path-based tradeoffs since $G_{TRSV}(n)$ is a $(b, b^2)$-path-expander dependency graph.
Diamond DAG lower bounds were also given by

- Papadimitriou and Ullman [P.U. 1987]
- Tiskin [T. 1998]

Previously noted algorithms for triangular solve attain these lower bounds with the number of processors $p = n/b$

- computational: $F_{TRSV} = \Theta(n^2 / p)$
- bandwidth: $W_{TRSV} = \Theta(n)$
- synchronization: $S_{TRSV} = \Theta(p)$
Tradeoffs for Cholesky

Theorem

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

$$F_{GE} = \Omega (n \cdot b^2), \quad W_{GE} = \Omega (n \cdot b), \quad S_{GE} = \Omega (n/b),$$

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

Proof.

Proof by showing that $G_{GE}(n)$ is a $(b^2, b^3)$-path-expander about the path corresponding to the calculation of the diagonal elements of $L$. 
These lower bounds are attainable for $b \in [n/\sqrt{p}, n/p^{2/3}]$ by existing algorithms, so for $c \in [1, p^{1/3}]$,

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

Therefore, we have a tradeoff between communication and synchronization

$$W_{GE} \cdot S_{GE} = \Theta(n^2)$$

We conjecture that our lower bound technique is extensible to QR and symmetric eigensolve algorithms.
We consider the $s$-step Krylov subspace basis computation

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

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

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

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

and furthermore,

\[
F_{Kr} \cdot S_{Kr}^d = \Omega \left( m^d \cdot s^{d+1} \right), \quad W_{Kr} \cdot S_{Kr}^{d-1} = \Omega \left( m^d \cdot s^d \right).
\]
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.
The lower bounds may be attained via communication-avoiding $s$-step algorithms (PA1 in Demmel, Hoemmen, Mohiyuddin, and Yelick 2007)

\[
F_{Kr} = O \left( m^d \cdot b^d \cdot s \right), \quad W_{Kr} = O \left( m^d \cdot b^{d-1} \cdot s \right), \quad S_{Kr} = O \left( s / b \right),
\]

under the assumption $n/p^{1/d} = O(bm)$. 
The APSP problem seeks to find the shortest paths between all pairs of vertices in graph $G$.

The Floyd-Warshall algorithm is analogous to Gaussian elimination on a different semiring and achieves costs

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

Alexander Tiskin demonstrated that it is possible to augment path-doubling to achieve the costs

$$F_{APSP} = O(n^3/p) \quad W_{APSP} = O(n^2/\sqrt{cp}) \quad S_{APSP} = O(\log p)$$
Summary and conclusion

Proved tight synchronization lower bounds on schedules for algorithms

- triangular solve $F_{TRSV} \cdot S_{TRSV} = \Omega(n^2)$
- Gaussian Elimination
  $F_{GE} \cdot S_{GE}^2 = \Omega(n^3)$ and $W_{GE} \cdot S_{GE} = \Omega(n^2)$
- $s$-step Krylov subspace methods on $(2m + 1)^d$-pt stencils
  $F_{Kr} \cdot S_{Kr}^d = \Omega(m^d \cdot s^{d+1})$ and $W_{Kr} \cdot S_{Kr}^{d-1} = \Omega(m^d \cdot s^d)$

Our lower bounds also extend naturally to a number of graph algorithms

- Floyd-Warshall is analogous to Gaussian Elimination
- Bellman-Ford is analogous to Krylov subspace methods

However, it may be possible to find new algorithms rather than parallel schedules to solve these problems with less synchronization and communication cost (e.g. Tiskin’s APSP algorithm, 2001)