Tradeoffs between synchronization, communication, and work in parallel schedules

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

Department of EECS, UC Berkeley

February, 2014
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
  
  For example, to compute $c = a \cdot b$, we have $a, b, c \in V$ and $(a, b), (a, c) \in E$.

For somewhat more generality, we may achieve this by working with hypergraph representations $H = (V, \overline{E})$ where

- $\overline{E}$ may represent the dependency of a value on a set of vertices (e.g. reduction tree)
  
  For example, to compute $d = \sum_{i=1}^{n} c_i$, we have $d, c_i \in V$ and hyperedges $(\{c_1, ..., c_n\}, \{d\}) \in \overline{E}$.
We can represent an algorithm as a graph $G = (V, E)$ where
- $V$ includes the input, intermediate, and output values used by the 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
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 $(a, b), (a, c) \in E$
Graphical representation of a computation

- 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 \((a, b), (a, c) \in E\)
- somewhat more generality may be achieved by working with hypergraph representations \( H = (V, \bar{E}) \)
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 $(a, b), (a, c) \in E$

somewhat more generality may be achieved 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)
Graphical representation of a computation

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

- somewhat more generality may be achieved 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 ($\{c_1, \ldots c_n\}, \{d\}) \in \bar{E}$
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 $(a, b), (a, c) \in E$

somewhat more generality may be achieved 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.
Parallel 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.

\[
V = \bigcup_{i=1}^{p} C_i
\]
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.
Parallel 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.
  - $F_{ij}$ is the set of values computed by processor $i$ at timestep $j$. 

Parallel 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:
  - $F_{ij}$ 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 $j$. 
Parallel 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:
  - \( F_{ij} \) 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 \( j \).
Parallel schedule example

Processor schedules

- computation
- message

F13
S12
F12
S11
R11
F11

2 GB
128 MB
60 MB
1 GB
27 MB
A schedule is a graph embedding

A parallel schedule must respect the dependency structure of the dependency graph of the algorithm

- The values $\bigcup_j F_{ij} = C_i \subset V$ correspond to the vertices of dependency graph $G$ computed by processor $i$
A parallel schedule must respect the dependency structure of the dependency graph of the algorithm

- The values $\bigcup_j F_{ij} = C_i \subset V$ correspond to the vertices of dependency graph $G$ computed by processor $i$
- All dependencies must be satisfied by the schedule
A schedule is a graph embedding

A parallel schedule must respect the dependency structure of the dependency graph of the algorithm

- The values $\bigcup_j F_{ij} = C_i \subset V$ correspond to the vertices of dependency graph $G$ computed by processor $i$
- All dependencies must be satisfied by the schedule
- Dependent values must be communicated or computed previously
A schedule is a graph embedding

A parallel schedule must respect the dependency structure of the dependency graph of the algorithm

- The values $\bigcup_j F_{ij} = C_i \subset V$ correspond to the vertices of dependency graph $G$ computed by processor $i$
- All dependencies must be satisfied by the schedule
- Dependent values must be communicated or computed previously
- For all non-local dependency paths in $G$, there must exist a sequence of messages in the schedule
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 $v$. 
Definition ($\epsilon, \sigma$)-path-expander

Graph $G = (V, E)$ is a ($\epsilon, \sigma$)-path-expander if there exists a path $(u_1, \ldots u_n) \subset V$, such that the dependency bubble $B(G, (u_i, u_{i+b}))$ has size $\Omega(\sigma(b))$ 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 $G = (V, E)$ about a path of length $n$ incurs the computation ($F$), bandwidth ($W$), and latency ($S$) costs

$$F = \Omega(\sigma(b) \cdot n/b), \quad W = \Omega(\epsilon(b) \cdot n/b), \quad S = \Omega(n/b).$$
An example \((b, b^2)\)-path-expander

- Dependency path P
- Computation chain
- Communication chain
For lower triangular dense $L$, solve

$$L \cdot x = y,$$

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

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

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

\[
\begin{bmatrix}
  y_1 & x_1 \\
  y_2 &  & Z^{21} & x_2 \\
  y_3 &  & Z^{31} & Z^{32} & x_3 \\
  y_4 &  & Z^{41} & Z^{42} & Z^{43} & x_4 \\
  y_5 &  & Z^{51} & Z^{52} & Z^{53} & Z^{54} & x_5 \\
\end{bmatrix}
\]
Theorem

Any parallelization of any dependency graph \( G_{\text{TRSV}}(n) \) where two processors compute \( \lfloor n^2/p \rfloor \) elements of \( \mathbb{Z} \) must incur a communication cost of

\[
W_{\text{TRSV}} = \Omega \left( n/\sqrt{p} \right).
\]

Proof.

Proof by application of lower bound on 2D lattice Hypergraph cut.
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]

Efficient algorithms for TRSV attain above lower bounds

- wavefront algorithms (Heath 1988)
- also algorithms given by [P.U 1987] and [T. 1998]
The Cholesky factorization of a symmetric positive definite matrix $A$ is

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

for a lower-triangular matrix $L$.

$L = \text{cholesky}(A, n)$

1. for $j = 1$ to $n$

2. $L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk} \cdot L_{jk}}$

3. for $i = j + 1$ to $n$

4. for $k = 1$ to $j - 1$

5. $Z_{ijk} = L_{ik} \cdot L_{jk}$

6. $L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} Z_{ijk})/L_{jj}$
These diagrams show (a) the vertices $Z_{ijk}$ in $V_{GE}$ with $n = 16$ and (b) the hyperplane $x_{12}$ and hyperedge $e_{12,6}$ on $H_{GE}$.
Theorem

Any $p$-processor parallelization of the dependency graph $G_{GE}(n)$ must incur a communication of

$$W_{GE} = \Omega \left( \frac{n^2}{p^{2/3}} \right).$$

Proof.

Employs 3D lattice hypergraph cut lower bound and assumes some work balance.
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 $L$. 
The lower bounds are attainable for Cholesky and similar costs are achievable for QR and the symmetric eigenproblem

- Tiskin’s non-pivoted recursive LU and pairwise-pivoted BSP algorithms
- 2.5D LU algorithm
- $W_{GE} = \frac{n^2}{\sqrt{cp}}$ bandwidth cost $S_{GE} = \sqrt{cp}$ synchronization cost
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.
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.

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 \( n/p^{1/d} = O(bm) \).
All-pairs shortest-paths problem

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

- seek the closure, $A^*$, of $A$ over the tropical semiring
  - $c = c \oplus a \otimes b$ on the tropical semiring implies $c = \min(c, a + b)$
  - the identity matrix $I$ on the tropical semiring is $0$ on the diagonal and $\infty$ everywhere else

\[ A^* = I \oplus A \oplus A^2 \oplus \ldots \oplus A^n = (I \oplus A)^n \]

- numerical computation on the sum-product semiring can be computed by Gauss-Jordan Elimination

\[ A^* = (I - A)^{-1} \]

- on the tropical semiring it is commonly computed by the Floyd-Warshall algorithm
The Floyd-Warshall algorithm is used to compute shortest paths between each pair of vertices using intermediate nodes \( \{1, 2, \ldots, k\} \),

\[
D = \text{Floyd-Warshall}(A, n)
\]

\[
D = A
\]

\[
\text{for } k = 1 \text{ to } n
\]

\[
\text{for } i = 1 \text{ to } n
\]

\[
\text{for } j = 1 \text{ to } n
\]

\[
d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})
\]
Gauss-Jordan elimination (Floyd Warshall algorithm)

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \rightarrow \begin{bmatrix}
A_{11}^* & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \rightarrow \begin{bmatrix}
A_{11}^* & A_{11}^* A_{12} \\
A_{21} A_{11}^* & A_{22} \oplus A_{21} A_{11}^* A_{12}
\end{bmatrix} = B
\]

\[
B = \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix} \rightarrow \begin{bmatrix}
B_{11} \oplus B_{12} B_{22}^* B_{21} & B_{12} B_{22}^* \\
B_{22}^* B_{21} & B_{22}^*
\end{bmatrix} = A^*
\]
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
We can compute the tropical semiring closure

\[ A^* = I \oplus A \oplus A^2 \oplus \ldots \oplus A^n = (I \oplus A)^n, \]

directly via repeated squaring (path-doubling)

\[ (I \oplus A)^{2k} = (I \oplus A)^k \otimes (I \oplus A)^k \]

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 $A^k$ by path size (number of edges)

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

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

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

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

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

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)

\[ A \quad B = (I+A)^2 = I + A + A^2 \quad A^* = (I+A)^4 = (I+B(2))B \]
Earlier caveat:

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

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

\[A^*(l) \otimes A^*(k) = A^*(l + k).\]

All shortest paths of up to any length are composable (factorizable), but not paths up to a limited length. However, the algorithm is correct because \(A^l \leq A^k(l) \leq A^*(k)\).
Since the decomposition by path size is disjoint, one can pick $A^k(l)$ for $l \in [k/2, k]$ to have size

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

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

$$
A^* = (A^p(l))^* \otimes A^p
$$

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 $p$
obtained synchronization cost lower bound for any parallel schedule of Gaussian elimination

same technique yields cost tradeoffs for Krylov subspace methods

on the tropical semiring these are shortest-path graph algorithms, Floyd-Warshall and Bellman-Ford

it is possible to use a different algorithm to circumvent the tradeoffs for the all-pairs shortest-paths problem

Open question: can one circumvent the tradeoffs in an algorithm that obtain the closure of a numerical matrix?