On Characterizing the Data Movement Complexity of Computational DAGs for Parallel Execution

Venmugil Elango\textsuperscript{1}  Fabrice Rastello\textsuperscript{2}  Louis-Noël Pouchet\textsuperscript{3}  J. Ramanujam\textsuperscript{4}  P. Sadayappan\textsuperscript{1}

\textsuperscript{1}The Ohio State University  \textsuperscript{2}Inria  \textsuperscript{3}University of California Los Angeles  \textsuperscript{4}Louisiana State University

26th ACM Symposium on Parallelism in Algorithms and Architectures —Supported by the U.S. National Science Foundation—
Outline

1. Introduction
2. Background
3. Decomposition
4. Red-Blue-White Pebble Game
5. Min-Cut based I/O Complexity
6. Parallel I/O Complexity
7. Example
# Motivation

**Hardware Prospective (source: SciDAC 2010)**

Here is a table showing the expected changes in various hardware parameters over the years 2009 to 2018:

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Peak Flops/s</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>System Memory</td>
<td>0.3 PB</td>
<td>1 PB</td>
<td>5 PB</td>
<td>10 PB</td>
</tr>
<tr>
<td>Node Performance</td>
<td>125 GF</td>
<td>200 GF</td>
<td>400 GF</td>
<td>1-10 TF</td>
</tr>
<tr>
<td>Node Memory BW</td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>100 GB/s</td>
<td>200-400 GB/s</td>
</tr>
<tr>
<td>Node Concurrency</td>
<td>12</td>
<td>32</td>
<td>0(100)</td>
<td>0(1000)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>10 GB/s</td>
<td>25 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>System Size (Nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
<td>0(Million)</td>
</tr>
<tr>
<td>Total Concurrency</td>
<td>225,000</td>
<td>3 Million</td>
<td>50 Million</td>
<td>0(Billion)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
<td>300 PB</td>
</tr>
<tr>
<td>I/O</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>10 TB/s</td>
<td>20 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>Days</td>
<td>Days</td>
<td>Days</td>
<td>0(1Day)</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~10 MW</td>
<td>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>

Data movement cost ≫ Computational cost
Data movement or I/O complexity

- Interested in characterizing minimum data movement cost of any valid schedule

```plaintext
for (i=0; i<N; i++)
    // Read row A(i,:) into fast memory
    for (j=0; j<N; j++)
        // Read C(i,j) and column B(:,j) into fast memory
        for (k=0; k<N; k++)
            C[i][j] += A[i][k] * B[k][j];
    // Write C(i,j) into slow memory
```

Untiled – Computational cost: $2N^3$; Data movement cost: $N^3 + 3N^2$

```plaintext
for (it=0; it<N; it=it+T)
    for (jt=0; jt<N; jt=jt+T)
        // Read block C(it,jt) into fast memory
        for (kt=0; kt<N; kt=kt+T)
            // Read block A(it,kt) and B(kt,jt) into fast memory
            for (i=it; i<it+T; i++)
                for (j=jt; j<jt+T; j++)
                    for (k=kt; k<kt+T; k++)
                        C[i][j] += A[i][k] * B[k][j];
        // Write block C(it,jt) into slow memory
```

Tiled – Computational cost: $2N^3$; Data movement cost: $\frac{2N^3}{T} + 2N^2$
Data movement or I/O complexity

- Interested in characterizing minimum data movement cost of any valid schedule

```plaintext
for (i=0; i<N; i++)
    //Read row A(i,:) into fast memory
    for (j=0; j<N; j++)
        //Read C(i,j) and column B(:,j) into fast memory
            for (k=0; k<N; k++)
                C[i][j] += A[i][k] * B[k][j];
    //Write C(i,j) into slow memory
```

Untiled – Computational cost: $2N^3$; Data movement cost: $N^3 + 3N^2$

```plaintext
for (it=0; it<N; it=it+T)
    for (jt=0; jt<N; jt=jt+T)
        //Read block C(it,jt) into fast memory
            for (kt=0; kt<N; kt=kt+T)
                //Read block A(it,kt) and B(kt,jt) into fast memory
                    for (i=it; i<it+T; i++)
                        for (j=jt; j<jt+T; j++)
                            for (k=kt; k<kt+T; k++)
                                C[i][j] += A[i][k] * B[k][j];
                    //Write block C(it,jt) into slow memory
```

Tiled – Computational cost: $2N^3$; Data movement cost: $\frac{2N^3}{T} + 2N^2$

Data movement Complexity: Lower bound on data movement cost
Computational directed acyclic graph (CDAG)

- Directed acyclic graph
  - Vertices - Inputs, Outputs and Computational operations
  - Edges - Flow of values (data dependencies)
- No specification of order of execution
- No association of memory locations

```c
for (i = 1; i < 4; ++i)
    S += A[i-1] + A[i];
```

---

Diagram of a computational directed acyclic graph (CDAG) showing the flow of values from inputs to output.
Red-Blue pebble game

- Two types of pebbles
  - Red pebbles - Fixed number - Small fast memory
  - Blue pebbles - Unlimited - Large slow memory
- **Initial state**: Blue pebbles on all input vertices
- **Goal**: Blue pebbles on output vertices
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
Sequence of application of rules
Red-Blue pebble game - 3 red pebbles

R1 (Input) Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1
Red-Blue pebble game - 3 red pebbles

R1 (Input) Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3 R4 R3
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue $\rightarrow$ Red
R2 (Output) Red $\rightarrow$ Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
$R1 \; R1 \; R1 \; R4 \; R3 \; R4 \; R3 \; R4 \; R1$
Red-Blue pebble game - 3 red pebbles

R1 (Input) | Blue → Red
---|---
R2 (Output) | Red → Blue
R3 (Compute) | Place red pebble, if all its predecessors are red
R4 (Delete) | Remove red pebble anytime

**Complete calculation:**

R1 R1 R1 R4 R3 R4 R4 R1 R4 R3
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3 R4 R4 R1 R4 R3 R4 R3
Red-Blue pebble game - 3 red pebbles

- **R1 (Input)**: Blue → Red
- **R2 (Output)**: Red → Blue
- **R3 (Compute)**: Place red pebble, if all its predecessors are red
- **R4 (Delete)**: Remove red pebble anytime

**Complete calculation:**

R1 R1 R1 R4 R3 R4 R4 R1 R4 R3 R4 R3 R4 R1
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3 R4 R3 R4 R1 R4 R3 R4 R3 R4 R1 R4 R3
Red-Blue pebble game - 3 red pebbles

R1 (Input) Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3 R4 R3 R4 R4 R4 R3 R3 R3 R4 R3 R4 R4 R3
Red-Blue pebble game - 3 red pebbles

R1 (Input) Blue $\rightarrow$ Red
R2 (Output) Red $\rightarrow$ Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
$R1\ R1\ R1\ R4\ R3\ R4\ R3\ R4\ R1\ R4\ R3\ R4\ R3\ R4\ R1\ R4\ R3\ R4\ R3\ R4\ R3$
Red-Blue pebble game - 3 red pebbles

R1 (Input)  Blue → Red
R2 (Output) Red → Blue
R3 (Compute) Place red pebble, if all its predecessors are red
R4 (Delete) Remove red pebble anytime

Complete calculation:
R1 R1 R1 R4 R3 R4 R3 R4 R1 R4 R3 R4 R3 R4 R1 R4 R3 R4 R3 R4 R3 R4 R3 R2
I/O Cost: $\#R1 + \#R2 = 6$
**S-partitioning** - Graph partitioning

\[ V = \bigcup_{i=1}^{h} V_i \]

- \( \forall h \) \( V_i \), \( \exists \) a **dominator set**, \( |D| \leq S \)
- \( \forall h \) \( V_i \), **minimum set**, \( |M| \leq S \)
- no cyclic dependence between subsets

- A **dominator set**, \( D \), of \( V_i \) is the set of vertices in \( V \) such that every path from input \( I \) to \( V_i \) contains some vertex in the set.
- The **minimum set**, \( M \), of \( V_i \) is the set of vertices in \( V_i \) that do not have any successors belonging to \( V_i \).
2S-partitioning and I/O complexity

- Given CDAG $C$, $S$ red pebbles, and a complete calculation with I/O cost $q$,

$$S \times h(2S) \geq q \geq S \times (h(2S) - 1)$$
2S-partitioning and I/O complexity

- Given CDAG $C$, $S$ red pebbles, and a complete calculation with I/O cost $q$,

$$S \times h(2S) \geq q \geq S \times (h(2S) - 1)$$

- If $H(2S) = \min h(2S)$. Then minimum I/O cost, $Q$,

$$Q \geq S \times (H(2S) - 1)$$
2S-partitioning and I/O complexity

- Given CDAG $C$, $S$ red pebbles, and a complete calculation with I/O cost $q$,

$$S \times h(2S) \geq q \geq S \times (h(2S) - 1)$$

- If $H(2S) = \min h(2S)$. Then minimum I/O cost, $Q$,

$$Q \geq S \times (H(2S) - 1)$$

- $U(C,2S)$ – upper-bound on size of largest vertex-set

$$Q \geq S \times \left( \frac{|V|}{U(C,2S)} - 1 \right)$$
Outline

1. Introduction
2. Background
3. Decomposition
4. Red-Blue-White Pebble Game
5. Min-Cut based I/O Complexity
6. Parallel I/O Complexity
7. Example
Decomposition

- Practical algorithms - composed of many routines, e.g., Conjugate Gradient: $\text{SpMV} \rightarrow \text{dot-product} \rightarrow \text{AxPY}$
- Program is decomposed; I/O complexity for each routine calculated separately

**Disjoint decomposition:**
- $C_1, C_2, \ldots, C_n$ – Disjoint partition of $C$ (need not be acyclic)

$$Q(C_1) + Q(C_2) + \cdots + Q(C_n) \leq Q(C)$$
Decomposition and Recomputation

```c
for (i = 0; i < 4; i++)
    c[i] = a[i] + b[i]; // S1
for (i = 0; i < 4; i++)
    d[i] = c[i] * c[i]; // S2
for (i = 0; i < 4; i++)
    e[i] = c[i] + d[i]; // S3
for (i = 0; i < 4; i++)
    f[i] = d[i] * e[i]; // S4
```

Hong & Kung model depends on inputs to get the lower bound

```
CDAG with no input vertex
Q = 0
```

Need to tag some vertices as inputs for tighter bound

Recomputation prevents tagging

SPAA 2014 14 / 32
Decomposition and Recomputation

```c
for (i = 0; i < 4; i++)
    c[i] = a[i] + b[i]; // S1
for (i = 0; i < 4; i++)
    d[i] = c[i] * c[i]; // S2
for (i = 0; i < 4; i++)
    e[i] = c[i] + d[i]; // S3
for (i = 0; i < 4; i++)
    f[i] = d[i] * e[i]; // S4
```

- Hong & Kung model depends on inputs to get the lower bound
  - CDAG with no input vertex $\implies Q = 0$
- Need to tag some vertices as inputs for tighter bound
- Recomputation prevents tagging
Red-Blue-White pebble game

- Prohibits recomputation by introducing white pebbles
  - Allows to derive tighter bounds by input/output tagging/untagging
Red-Blue-White pebble game

- Prohibits recomputation by introducing white pebbles
  - Allows to derive tighter bounds by input/output tagging/untagging

**Rules:**

**R1 (Input)** Blue → Red, white

**R2 (Output)** Red → Blue

**R3 (Compute)** Place red and white pebbles, if not already white pebbled and all its predecessors have red pebble

**R4 (Delete)** Remove red pebble anytime
Outline

1. Introduction
2. Background
3. Decomposition
4. Red-Blue-White Pebble Game
5. Min-Cut based I/O Complexity
6. Parallel I/O Complexity
7. Example
Min-cut based I/O complexity

- 2S-partitioning focuses on frontiers of the vertex-sets

- Min-cut approach captures internal space requirement
Min-cut I/O complexity

Schedule wavefront:

Graph wavefront:

I/O complexity:

- $W^\text{min}_G(x)$: Min-cut wavefront at $x$
- Let $w^\text{max}_G = \max_{x \in V} \left( |W^\text{min}_G(x)| \right)$

$$2 \left( w^\text{max}_G - S \right) \leq Q(C)$$
Example: Diamond DAG

\[ 2(N - S) \leq Q(C) \]
Example: Diamond DAG with decomposition

$$4 \times 2(N/2 - S) \leq Q(C)$$
Parallel I/O Complexity

Architecture

$R_1^l, R_2^l, \cdots, R_{N_l}^l$ red pebbles of different shades
Parallel Red-Blue-White pebble game (P-RBW)

Rules:

- **R1 (Input)**  \( R_L \rightarrow R^L_i, \) white
- **R2 (Output)**  \( R^L_i \rightarrow \) blue
- **R3 (Remote get)**  \( R^L_j \rightarrow R^L_i \)
- **R4 (Move up)**  \( R^{l+1}_j \rightarrow R^l_i, \) \( i \) is child of \( j \)
- **R5 (Move down)**  \( R^{l-1}_i \rightarrow R^l_j, \) \( i \) is child of \( j \)
- **R6 (Compute)** Place \( R^1_p \) shaded red pebble and white pebble, if not already white pebbled and all its predecessors have \( R^1_p \) shade pebble
- **R7 (Delete)** Remove any shade of red pebble anytime

**Vertical I/O cost**: \( \#R4 + \#R5 \)

**Horizontal I/O cost**: \( \#R3 \)
Vertical I/O complexity

- Eg.: 4 core machine
  - Private L1 caches of size $S_1$
  - Two L2 caches: $M_1$ and $M_2$
- Vertical cost: Data moved between one L2 cache and its two children

Consider a serial processor with cache size $S = 4S_1$

Serial I/O cost:

$$Q(4S_1) \leq Q(M_1)(2S_1) + Q(M_2)(2S_1)$$

Suppose $M_1$ has higher data transfers than $M_2$

$$Q(4S_1) \leq Q(M_1)(2S_1) + Q(M_2)(2S_1) \leq 2Q(M_1)(2S_1)$$

General case:

$$Q(S_l - 1 \times N_l - 1) N_l \leq Q(M_i)(N_l - 1 \times S_l - 1)$$
Vertical I/O complexity

- Eg.: 4 core machine
  - Private L1 caches of size $S_1$
  - Two L2 caches: $M_1$ and $M_2$
- Vertical cost: Data moved between one L2 cache and its two children
- Consider a serial processor with cache size $S = 4S_1$
- Serial I/O cost: $Q(4S_1) \leq Q^{M_1}(2S_1) + Q^{M_2}(2S_1)$
- Suppose $M_1$ has higher data transfers than $M_2$

\[ Q(4S_1) \leq Q^{M_1}(2S_1) + Q^{M_2}(2S_1) \leq 2Q^{M_1}(2S_1) \]
Vertical I/O complexity

- Eg.: 4 core machine
  - Private L1 caches of size $S_1$
  - Two L2 caches: $M_1$ and $M_2$
- Vertical cost: Data moved between one L2 cache and its two children
- Consider a serial processor with cache size $S = 4S_1$
- Serial I/O cost: $Q(4S_1) \leq Q^{M_1}(2S_1) + Q^{M_2}(2S_1)$
- Suppose $M_1$ has higher data transfers than $M_2$

$$Q(4S_1) \leq Q^{M_1}(2S_1) + Q^{M_2}(2S_1) \leq 2Q^{M_1}(2S_1)$$

- General case:

$$\frac{Q(S_{l-1} \times N_{l-1})}{N_l} \leq Q^{M_l} \left( \frac{N_{l-1}}{N_l} \times S_{l-1} \right)$$
Tighter 2S-partitioning based vertical I/O complexity

- CDAG $C$ with $V$ vertices
- sub-CDAGs $C^p_1$ (with $\alpha_1 |V|$ vertices), $C^p_2$ (with $\alpha_2 |V|$ vertices) - computed by the two processors that share an L2 cache
  - $\alpha_1 |V| + \alpha_2 |V| \geq |V|/2$
Tighter 2S-partitioning based vertical I/O complexity

- CDAG $C$ with $V$ vertices
- sub-CDAGs $C^{p_1}$ (with $\alpha_1 \cdot |V|$ vertices), $C^{p_2}$ (with $\alpha_2 \cdot |V|$ vertices) - computed by the two processors that share an L2 cache
  - $\alpha_1 \cdot |V| + \alpha_2 \cdot |V| \geq |V| / 2$
- $U(2S_1)$: Size of largest vertex set of $C$
- $U^{p_1}(2S_1)$: Size of largest vertex set of $C^{p_1}$
- $U^{p_2}(2S_1)$: Size of largest vertex set of $C^{p_2}$
Tighter 2S-partitioning based vertical I/O complexity

- CDAG \( C \) with \( V \) vertices
- sub-CDAGs \( C^{p1} \) (with \( \alpha_1 \cdot |V| \) vertices), \( C^{p2} \) (with \( \alpha_2 \cdot |V| \) vertices) - computed by the two processors that share an L2 cache
  - \( \alpha_1 \cdot |V| + \alpha_2 \cdot |V| \geq |V| / 2 \)
- \( U(2S_1) \): Size of largest vertex set of \( C \)
- \( U^{p1}(2S_1) \): Size of largest vertex set of \( C^{p1} \)
- \( U^{p2}(2S_1) \): Size of largest vertex set of \( C^{p2} \)
- \( U^{p1}(2S_1) \leq U(2S_1) ; U^{p2}(2S_1) \leq U(2S_1) \);
Tighter 2S-partitioning based vertical I/O complexity

- CDAG $C$ with $V$ vertices
- sub-CDAGs $C^p_1$ (with $\alpha_1 |V|$ vertices), $C^p_2$ (with $\alpha_2 |V|$ vertices) - computed by the two processors that share an L2 cache
  - $\alpha_1 |V| + \alpha_2 |V| \geq |V|/2$
- $U(2S_1)$: Size of largest vertex set of $C$
- $U^{p1}(2S_1)$: Size of largest vertex set of $C^p_1$
- $U^{p2}(2S_1)$: Size of largest vertex set of $C^p_2$
- $U^{p1}(2S_1) \leq U(2S_1); U^{p2}(2S_1) \leq U(2S_1)$;

\[
\left( \frac{|V|/2}{U(2S_1)} - 1 \right) \times S_1 \leq \left( \frac{\alpha_1 |V|}{U^{p1}(2S_1)} - 1 \right) \times S_1 + \left( \frac{\alpha_2 |V|}{U^{p2}(2S_1)} - 1 \right) \times S_1
\]
Horizontal I/O Complexity

- Data movement cost between nodes through inter-connection networks
- \( S \)-partitioning based horizontal I/O cost (using similar reasoning as in vertical case)

\[
Q \geq S_L \times \left( \frac{|V|/N_L}{|U(2S_L)|} - 1 \right)
\]
Outline

1. Introduction
2. Background
3. Decomposition
4. Red-Blue-White Pebble Game
5. Min-Cut based I/O Complexity
6. Parallel I/O Complexity
7. Example
Example: Conjugate gradient

1. x is the initial guess
2. p ← r ← b − Ax
3. do
4.   v ← Ap //SpMV
5.   b ← (r.r) //Dot-prod
6.   a ← b/(p.v) //Dot-prod
7.   x ← x + ap //AXPY
8.   r ← r − av //AXPY
9.   g ← (r.r)/b //Dot-prod
10.  p ← r + gp //AXPY
11. until ((r.r) is small) //T timesteps

Vertical lower bound:

\[ Q \geq \frac{T \times (2(2n^d - S)) + T \times (2(n^d - S))}{P} \]
\[ = \frac{T \times (2(3n^d - 2S))}{P} \approx \frac{6n^dT}{P} \]

Horizontal upper bound: \( Q = O(4dB^{d-1}T) \)

where, \( n = \) size along each dimension, \( d = \) dimensions, \( B = n/N_{\text{nodes}}^{1/d} \), \( P = \) no. of processors
Example: Conjugate gradient - Analysis

- Machine balance = \[
\frac{\text{Peak data movement bandwidth (in GBytes/s)}}{\text{Peak computational throughput (in GFLOPs)}}
\]

<table>
<thead>
<tr>
<th>Machine</th>
<th>Vertical balance (words / FLOP)</th>
<th>Horiz. balance (words / FLOP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM BG/Q</td>
<td>0.052</td>
<td>0.006</td>
</tr>
<tr>
<td>Cray XT5</td>
<td>0.0256</td>
<td>0.005</td>
</tr>
</tbody>
</table>

For an example problem:

- Overall computational ops. = \(24n^3 T\)
- Overall memory to LLC I/O ≥ \(6n^3 T\)
- Overall inter-node I/O ≤ \(12B^2 T \times N\) nodes

The maximum required vertical words/flops is \(6 \times \frac{24}{24} = 0.25\), much higher than the architectural vertical balance = \(0.052\), leading to unavoidable bandwidth bound along the vertical direction.

The maximum required horizontal words/flops ≤ \(3\sqrt{\frac{2}{n}} \times \frac{N}{2}\) for practical problem sizes, indicating that inter-node bandwidth is not a bottleneck.
Example: Conjugate gradient - Analysis

- **Machine balance** = \( \frac{\text{Peak data movement bandwidth (in GBytes/s)}}{\text{Peak computational throughput (in GFLOPs)}} \)

<table>
<thead>
<tr>
<th>Machine</th>
<th>Vertical balance (words / FLOP)</th>
<th>Horiz. balance (words / FLOP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM BG/Q</td>
<td>0.052</td>
<td>0.006</td>
</tr>
<tr>
<td>Cray XT5</td>
<td>0.0256</td>
<td>0.005</td>
</tr>
</tbody>
</table>

- **CG for 3D problem:**
  - Overall computational ops. = \(24n^3T\)
  - Overall memory to LLC I/O \(\geq 6n^3T\)
  - Overall inter-node I/O \(\leq 12B^2T \times N_{\text{nodes}}\)

- Maximum (over all nodes) required vertical words/flops \(\geq \frac{6}{24} = 0.25\)
  - Much higher than architectural vertical balance \(\Longrightarrow\) unavoidably bandwidth bound along vertical direction

- Maximum (over all nodes) required horizontal words/flops \(\leq \frac{\sqrt[3]{N_{\text{nodes}}}}{2n}\)
  - Lower than architectural horizontal balance for practical problem sizes
    \(\Longrightarrow\) Inter-node bandwidth not a bottleneck
Example: 3D-Jacobi method

- Overall computational ops. \( = 7n^3T \)
- Overall memory to LLC I/O \( \geq n^3T / (4^{3/2}S) \)
- Overall memory to LLC I/O \( \leq 14n^3T / (3^{3/2}S) \)
- Overall inter-node I/O \( \leq 12B^2T \times N_{nodes} \)
Example: 3D-Jacobi method

- Overall computational ops. = $7n^3T$
- Overall memory to LLC I/O $\geq n^3T/(4\sqrt[3]{2S})$
- Overall memory to LLC I/O $\leq 14n^3T/(\sqrt[3]{S})$
- Overall inter-node I/O $\leq 12B^2T \times N_{nodes}$

For $S = 0.75$ MWords,

- Lower bound on vertical words/flops: $3 \times 10^{-4}$
- Upper bound on vertical words/flops: 0.022
  - Not necessarily bandwidth constrained along vertical direction
- Upper bound on horizontal words/flops: $1.714\sqrt[3]{N_{nodes}/n}$
  - Not bandwidth constrained along horizontal direction
Example: 3D-Jacobi method

- Overall computational ops. $= 7n^3T$
- Overall memory to LLC I/O $\geq \frac{n^3T}{(4^3\sqrt{2}S)}$
- Overall memory to LLC I/O $\leq 14n^3T/(3\sqrt{S})$
- Overall inter-node I/O $\leq 12B^2T \times N_{nodes}$

For $S = 0.75$ MWords,

- Lower bound on vertical words/flops: $3 \times 10^{-4}$
- Upper bound on vertical words/flops: 0.022
  - Not necessarily bandwidth constrained along vertical direction
- Upper bound on horizontal words/flops: $1.714\frac{3\sqrt{N_{nodes}}}{n}$
  - Not bandwidth constrained along horizontal direction

Jacobi might be an attractive alternative to CG on future machines despite slower convergence rate
Thank you