swSpTRSV: a Fast Sparse Triangular Solve with Sparse Level Tile Layout on Sunway Architecture

Xinliang Wang¹,³ Weifeng Liu² Wei Xue¹,³ Li Wu¹,³
Outline

1. Background
2. Sunway architecture
3. Sparse Level Tile layout
4. Producer-Consumer Pairing method
5. Experiment
6. Conclusion
Example: $Lx = b$

Compute a solution vector $x$ from the sparse linear system, where $L$ is a square lower triangular sparse matrix, and $b$ is the right-hand vector.

System:

\[
\begin{align*}
1^* x_0 &= a \\
1^* x_1 &= b \\
2^* x_1 + 1^* x_2 &= c \\
3^* x_0 + 1^* x_3 &= d
\end{align*}
\]

Solution:

\[
\begin{align*}
x_0 &= a \\
x_1 &= b \\
x_2 &= c - 2b \\
x_3 &= d - 3a
\end{align*}
\]
Example: $Lx = b$
Compute a solution vector $x$ from the sparse linear system, where $L$ is a square lower triangular sparse matrix, and $b$ is the right-hand vector.

\[
L = \begin{bmatrix}
1 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
2 & 1 & 1 & 1 \\
3 & 1 & 1 & 1 \\
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix}
\]

System:
\[
\begin{align*}
1x_0 &= a \\
1x_1 &= b \\
2x_1 + 1x_2 &= c \\
3x_0 + 1x_3 &= d
\end{align*}
\]

Solution:
\[
\begin{align*}
x_0 &= a \\
x_1 &= b \\
x_2 &= c - 2b \\
x_3 &= d - 3a
\end{align*}
\]
Example: \( Lx = b \)

Compute a solution vector \( x \) from the sparse linear system, where \( L \) is a square lower triangular sparse matrix, and \( b \) is the right-hand vector.

Use case:
In direct methods for solving a sparse linear system \( Ax = b \), \( A \) can be first decomposed to \( LU \), then be solved by \( LUx = b \). This is done by calling two sparse triangular solves \( Ly = b \) and \( Ux = y \).
In iterative solvers, incomplete \( LU \) preconditioner uses sparse triangular solves in a similar way.
A sequential method based on CSC layout

Output: *x;
1: for i = 0 to n - 1 do
2: \[ x[i] = b[i] / val[col_ptr[i]] \]
3: for j = col_ptr[i] + 1 to col_ptr[i + 1] - 1 do
4: \[ b[row_idx[j]] = b[row_idx[j]] - val[j] * x[i] \]
5: end for
6: end for

\[ Lx = b \]
A few cores: Level-set method

Parallel in each level and sequential inter level
A few cores: Level-set method

Parallel in each level and sequential inter level
A few cores: Level-set method

Parallel in each level and sequential inter level

clarencewxl@gmail.com
A few cores: Level-set method

Parallel in each level and sequential inter level
A few cores: Level-set method

Parallel in each level and sequential inter level
A few cores: Level-set method

Parallel in each level and sequential inter level
More cores: P2P method (CPU/MIC)

- No full-synchronization
- Only synchronize between Thread 0 and Thread 2

More cores: Sync-free method (GPU)

- Thread 0 and 2 modify the same value by atomic operations.

Background

Problem

Architecture

Sparse Triangular Solve

Sunway Processor

clarencewxl@gmail.com
# Sunway TaihuLight: Overview

## Entire System

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Performance</td>
<td>125 PFlops</td>
</tr>
<tr>
<td>Linpack Performance</td>
<td>93 Pflops / 74.4%</td>
</tr>
<tr>
<td>Total Memory</td>
<td>1310.72 TB</td>
</tr>
<tr>
<td>Total Memory Bandwidth</td>
<td>5591.45 TB/s</td>
</tr>
<tr>
<td># nodes</td>
<td>40,960</td>
</tr>
<tr>
<td># cores</td>
<td>10,649,600</td>
</tr>
</tbody>
</table>
### SW26010 Processor

**Diagram:**
- **Computing Core**
- **Registers**
- **SPM**
- **Transfer Agent (TA)**
- **Row Communication Bus**
- **Column Communication Bus**
- **Data Transfer Network**
- **8*8 CPE Mesh**
- **NoC**
- **Memory**
- **LDM Level**
- **Register Level**
- **Computing Level**

**Table:**

<table>
<thead>
<tr>
<th>Name</th>
<th>MPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (GHz)</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>SIMD width</td>
<td>4 double / 8 integer</td>
<td>4 double / 8 integer</td>
</tr>
<tr>
<td>Peak Gflop/s in D.P.</td>
<td>23.2</td>
<td>742.4</td>
</tr>
<tr>
<td>Data L1 / L2 cache</td>
<td>32KB / 256KB</td>
<td>64KB (SPM) / -</td>
</tr>
<tr>
<td>Instruction L1 / L2 cache</td>
<td>32KB / +</td>
<td>16KB / 64KB</td>
</tr>
<tr>
<td>Memory capacity (GB)</td>
<td>32 (shared)</td>
<td></td>
</tr>
</tbody>
</table>
SW26010 Processor

<table>
<thead>
<tr>
<th>Name</th>
<th>MPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (GHz)</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>SIMD width</td>
<td>4 double / 8 integer</td>
<td>4 double / 8 integer</td>
</tr>
<tr>
<td>Peak Gflop/s in D.P.</td>
<td>23.2</td>
<td>742.4</td>
</tr>
<tr>
<td>Data L1 / L2 cache</td>
<td>32KB / 256KB</td>
<td>64KB (SPM) / -</td>
</tr>
<tr>
<td>Instruction L1 / L2 cache</td>
<td>32KB / +</td>
<td>16KB / 64KB</td>
</tr>
<tr>
<td>Memory capacity (GB)</td>
<td>32 (shared)</td>
<td></td>
</tr>
</tbody>
</table>

clarencewxl@gmail.com
**SW26010 Processor**

**Direct Memory Access (DMA) 22.6 GB/s**

---

<table>
<thead>
<tr>
<th>Name</th>
<th>MPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (GHz)</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>SIMD width</td>
<td>4 double / 8 integer</td>
<td>4 double / 8 integer</td>
</tr>
<tr>
<td>Peak Gflop/s in D.P.</td>
<td>23.2</td>
<td>742.4</td>
</tr>
<tr>
<td>Data L1 / L2 cache</td>
<td>32KB / 256KB</td>
<td>64KB (SPM) / -</td>
</tr>
<tr>
<td>Instruction L1 / L2 cache</td>
<td>32KB / +</td>
<td>16KB / 64KB</td>
</tr>
<tr>
<td>Memory capacity (GB)</td>
<td>32 (shared)</td>
<td></td>
</tr>
</tbody>
</table>

---

**clarencewxl@gmail.com**
**SW26010 Processor**

**Global Load/Store**
(Gload/Gstore) **1.5 GB/s**

---

### Specifications

<table>
<thead>
<tr>
<th>Name</th>
<th>MPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (GHz)</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>SIMD width</td>
<td>4 double / 8 integer</td>
<td>4 double / 8 integer</td>
</tr>
<tr>
<td>Peak Gflop/s in D.P.</td>
<td>23.2</td>
<td>742.4</td>
</tr>
<tr>
<td>Data L1 / L2 cache</td>
<td>32KB / 256KB</td>
<td>64KB (SPM) / -</td>
</tr>
<tr>
<td>Instruction L1 / L2 cache</td>
<td>32KB / +</td>
<td>16KB / 64KB</td>
</tr>
<tr>
<td>Memory capacity (GB)</td>
<td>32 (shared)</td>
<td></td>
</tr>
</tbody>
</table>
Register Communication

Get C

Put

Get R

Get C

Put

Get R

Get C

Put

Get R

Get C

Put

...

...

...

...

...
Register Communication

Get C  
Put  
Get R

Get C  
Put  
Get R

Get C  
Put  
Get R

\[ \text{putr} \leftrightarrow \text{getr} \]

\[ \text{putc} \leftrightarrow \text{getc} \]
SW26010 Processor

- Manual cache system (SPM)
- Direct memory access (DMA)
- Limited register communication
Mismatch between SpTRSV and Sunway

- Branch code to check whether cache is miss or not;
- The cost of the branch is high

```c
Output: *x;
1: for i = 0 to n - 1 do
2:     x[i] = b[i] / val[col_ptr[i]];
3:     for j = col_ptr[i] + 1 to col_ptr[i + 1] - 1 do
4:         if b[row_idx[j]] is not in SPM then
5:             Swap b[row_idx[j]] into SPM
6:         end if
7:         b[row_idx[j]] = b[row_idx[j]] - val[j] * x[i];
8:     end for
9: end for
```

- Cost much even cache hit
- Hurt the instruction pipeline
- Difficult to prefetch
- Manual cache system
- Direct memory access
- Register communication
Mismatch between SpTRSV and Sunway

Limitation of register communication: only happen in the same column or row

- Manual cache system
- Direct memory access
- Register communication

CPE (0,0) → CPE (0,1) → CPE (1,1)
Mismatch between SpTRSV and Sunway

Limitation of register communication: only happen in the same column or row

- Manual cache system
- Direct memory access
- Register communication

Communication cycle + Random communication size ≈ Dead-Lock

Contributions

• Sparse Level Tile (SLT) Layout:
  • Manual Cache System
  • Direct Memory Access (DMA)

• Producer-Consumer pairing method:
  • Register Communication
Contributions

- Manual Cache System
- Direct Memory Access (DMA)

- Sparse Level Tile layout
  - Make sure all the computation is cache-hit;
  - Replace fine-grained, random and unprefetchable memory access with course-grained, predictable and prefetchable memory access;
Contributions

• Register Communication

• Producer-Consumer pairing method:
  • Make communication cycle and random communication size not happen at the same time;
Sparse Level Tile (SLT) Layout

Each Tile only targets on sub-vector $\mathbf{x}$ and $\mathbf{b}$

- fine-grained, random, unprefetchable
- course-grained, predictable, prefetchable
Sparse Level Tile (SLT) Layout: Step 1

Step 1:
- Divide the x and b into multiple Regions

Each offdiagonal nonzero import $b_i = b_i - l_{ij} * x_j$, using $x_j$ from an X-Region to update $b_i$ from a B-Region
Step 2:
- Separate the original levels into multiple levels if crossing multiple X-Regions.
Sparse Level Tile (SLT) Layout: Step 3

Step 3:
- Combine the diagonal nonzeros with their “nearest” off-diagonal nonzeros to consist Diagonal-Tiles
- The width is #diagonal nonzeros
- The height is (width + Region Size)

Benefit:
- Guarantee the corresponding x elements and b elements must be cached.
Step 4:
- Combine residual nonzeros to consist Offdiagonal-Tiles
- Both the width and the height is smaller than Region Size

Benefit:
- Guarantee the corresponding x elements and b elements must be cached
- Course-grainedly load x elements based on X-region
Sparse Level Tile (SLT) Layout: Step 5

Step 5:
- Sort the Tiles;
- Each Tile has an ID, which is the maximum B-Region each Tile modifies;
- Tiles with smaller ID are stored in front of those with bigger ID;
- Diagonal-Tiles are stored in front of Offdiagonal-Tiles.

Benefit:
- Increase the data-reuse of b;
The SpTRSV process based on SLT layout

Input: $l$, $li$, $lj$, tiles, sizes, $idx$, $b$
Output: $x$

1: function SpTRSV($x$, $b$, $l$, $li$, $lj$, tiles, sizes, $idx$)
2:   CACHE $cx[\text{REGION\_SIZE}]$ // cache for $x$
3:   CACHE $cb[\text{REGION\_SIZE}]$ // cache for $b$
4:   Replenish $cb$ from $b$ for $\text{REGION\_SIZE}$
5:   for $t = 0 \rightarrow \text{tiles} - 1$ do
6:       $nodnz \leftarrow idx[t + 1] - idx[t] - sizes[t]$
7:       if Tile is Diagonal then
8:           Copy $cb$ to $cx$ for $sizes[t]$
9:           Replenish $cb$ from $b$ for $sizes[t]$
10:          Vector Division for $sizes[t]$ // $x_j = b_j/l_{jj}$
11:         Store $cx$ to $x$ for $sizes[t]$
12:       else if Tile is OffDiagonal then
13:           Load $x$ to $cx$ for $\text{REGION\_SIZE}$
14:       end if
15:       $cb = cb - \text{Tile}_t \times cx$ for $nodnz$ // $b_i = b_i - l_{ij}x_j$, $\text{Tile}_t$
16:       means the submatrix consist of the nonzeros in Tile $t$
17:   end for
18: end function
The SpTRSV process based on SLT layout

Input: \( l, li, lj, tiles, sizes, idx, b \)
Output: \( x \)

1: function SpTRSV(\( x, b, l, li, lj, tiles, sizes, idx, b \))
2: CACHE \( cx[\text{REGION\_SIZE}] \) // cache for \( x \)
3: CACHE \( cb[\text{REGION\_SIZE}] \) // cache for \( b \)
4: Replenish \( cb \) from \( b \) for \( \text{REGION\_SIZE} \)
5: for \( t = 0 \rightarrow \text{tiles} - 1 \) do
6: \( \text{nodnz} \leftarrow \text{idx}[t + 1] - \text{idx}[t] - \text{sizes}[t] \)
7: if Tile is Diagonal then
8: Copy \( cb \) to \( cx \) for \( \text{sizes}[t] \)
9: Replenish \( cb \) from \( b \) for \( \text{sizes}[t] \)
10: Vector Division for \( \text{sizes}[t] \) // \( x_j = b_j/l_{jj} \)
11: Store \( cx \) to \( x \) for \( \text{sizes}[t] \)
12: else if Tile is OffDiagonal then
13: Load \( x \) to \( cx \) for \( \text{REGION\_SIZE} \)
14: end if
15: \( cb = cb - \text{Tile}_t \times cx \) for \( \text{nodnz} \) // \( b_i = b_i - l_{ij}x_j \), \( \text{Tile}_t \)
16: means the submatrix consist of the nonzeros in Tile \( t \)
17: end function

Inherit from previous Tile
Load from main memory
Obtained from \( x_j = b_j/l_{jj} \)
Used in \( b_i = b_i - l_{ij}x_j \)

\( b_0 \)
\( b_1 \)
\( b_2 \)
\( b_3 \)
The SpTRSV process based on SLT layout

Input: \( l, l_i, l_j, \) tiles, sizes, idx, \( b \)
Output: \( x \)

1: function SpTRSV(\( x, b, l, l_i, l_j, \) tiles, sizes, idx) 
2: CACHE \( cx[\text{REGION\_SIZE}] \) // cache for \( x \)
3: CACHE \( cb[\text{REGION\_SIZE}] \) // cache for \( b \)
4: Replenish \( cb \) from \( b \) for \( \text{REGION\_SIZE} \)
5: for \( t = 0 \rightarrow \) tiles - 1 do
6: \( \text{nordz} = \text{idx}[t + 1] - \text{idx}[t] - \text{sizes}[t] \)
7: if Tile is Diagonal then
8: Copy \( cb \) to \( cx \) for \( \text{sizes}[t] \)
9: Replenish \( cb \) from \( b \) for \( \text{sizes}[t] \)
10: Vector Division for \( \text{sizes}[t] \) // \( x_j = b_j/l_{jj} \)
11: Store \( cx \) to \( x \) for \( \text{sizes}[t] \)
12: else if Tile is OffDiagonal then
13: Load \( x \) to \( cx \) for \( \text{REGION\_SIZE} \)
14: end if
15: \( cb = cb - \text{Tile}_t \times cx \) for nordz // \( b_i = b_i - l_{ij}x_j, \) \( \text{Tile}_t \)
16: means the submatrix consist of the nonzeros in Tile \( t \)
17: end for
18: end function

Inherit from previous Tile
Load from main memory
Obtained from \( x_j = b_j/l_{jj} \)
Used in \( b_i = b_i - l_{ij}x_j \)

\( b_0 \)
\( b_1 \)
\( b_2 \)
\( b_3 \)

clarencewxl@gmail.com
The SpTRSV process based on SLT layout

```
Input: l, ll, lj, tiles, sizes, idx, b
Output: x
1: function SpTRSV(x, b, l, ll, lj, tiles, sizes, idx)  
2:   CACHE cx[REGION_SIZE] // cache for x  
3:   CACHE cb[REGION_SIZE] // cache for b  
4:   Replenish cb from b for REGION_SIZE  
5:   for t = 0 -> tiles - 1 do  
6:     nodnz ← idx[t + 1] - idx[t] - sizes[t]  
7:     if Tile is Diagonal then  
8:       Copy cb to cx for sizes[t]  
9:       Replenish cb from b for sizes[t]  
10:      Vector Division for sizes[t] // x_j = b_j / l_jj  
11:     Store cx to x for sizes[t]  
12:     else if Tile is OffDiagonal then  
13:       Load x to cx for REGION_SIZE  
14:     end if  
15:     cb = cb - Tile_t × cx for nodnz // b_i = b_i - l_i_j x_j, Tile_t  
16:       means the submatrix consist of the nonzeros in Tile_t  
17:   end function
```
The SpTRSV process based on SLT layout

Input: \( l, li, lj, tiles, sizes, idx, b \)

Output: \( x \)

```plaintext
1: function SpTRSV(x, b, l, li, lj, tiles, sizes, idx) 
2:   CACHE cx[REGION_SIZE] // cache for x 
3:   CACHE cb[REGION_SIZE] // cache for b 
4:   Replenish cb from b for REGION_SIZE 
5:   for \( t = 0 \rightarrow tiles - 1 \) do 
6:     nodnz ← idx[\( t + 1 \)] - idx[t] - sizes[t] 
7:     if Tile is Diagonal then 
8:       Copy cb to cx for sizes[t] 
9:       Replenish cb from b for sizes[t] 
10:      Vector Division for sizes[t] // \( x_j = b_j/l_{jj} \) 
11:     else if Tile is OffDiagonal then 
12:       Load x to cx for REGION_SIZE 
13:     end if 
14:     cb = cb - Tile_t \times cx for nodnz // \( b_i = b_i - l_{ij}x_j \), Tile_t means the submatrix consist of the nonzeros in Tile t 
15:   end for 
16: end function
```

clarancewxl@gmail.com
The SpTRSV process based on SLT layout

Input: $l, li, lj, tiles, sizes, idx, b$

Output: $x$

1: function SpTRSV($x, b, l, li, lj, tiles, sizes, idx$)
2:     CACHE $cx[\text{REGION\_SIZE}]$ // cache for $x$
3:     CACHE $cb[\text{REGION\_SIZE}]$ // cache for $b$
4:     Replenish $cb$ from $b$ for $\text{REGION\_SIZE}$
5:     for $t = 0 \rightarrow \text{tiles} - 1$ do
6:         $nodnz \leftarrow \text{idx}[t + 1] - \text{idx}[t] - \text{sizes}[t]$
7:         if Tile is Diagonal then
8:             Copy $cb$ to $cx$ for $\text{sizes}[t]$
9:             Replenish $cb$ from $b$ for $\text{sizes}[t]$
10:        Vector Division for $\text{sizes}[t]$ \// $x_j = b_j/l_{jj}$
11:        Store $cx$ to $x$ for $\text{sizes}[t]$
12:    else if Tile is OffDiagonal then
13:        Load $x$ to $cx$ for $\text{REGION\_SIZE}$
14:        $cb = cb - \text{Tile}_t \times cx$ for $nodnz$ \// $b_i = b_i - l_{ij}x_j$, $\text{Tile}_t$ means the submatrix consist of the nonzeros in Tile $t$
15:    end if
16: end for
17: end function
The SpTRSV process based on SLT layout

Input: $l$, $li$, $lj$, tiles, sizes, $idx$, $b$
Output: $x$

1: function SpTRSV($x$, $b$, $l$, $li$, $lj$, tiles, sizes, $idx$)
2: CACHE $cx[\text{REGION\_SIZE}]$ // cache for $x$
3: CACHE $cb[\text{REGION\_SIZE}]$ // cache for $b$
4: Replenish $cb$ from $b$ for $\text{REGION\_SIZE}$
5: for $t = 0 \rightarrow \text{tiles} - 1$ do
6:  $\text{nodnz} \leftarrow idx[t + 1] - idx[t] - \text{sizes}[t]$
7:  if Tile is Diagonal then
8:      Copy $cb$ to $cx$ for $\text{sizes}[t]$
9:      Replenish $cb$ from $b$ for $\text{sizes}[t]$
10:     Vector Division for $\text{sizes}[t]$ // $x_j = b_j/l_{jj}$
11:    Store $cx$ to $x$ for $\text{sizes}[t]$
12:  else if Tile is OffDiagonal then
13:     Load $x$ to $cx$ for $\text{REGION\_SIZE}$
14:  end if
15:  $cb = cb - \text{Tile}_t \times cx$ for $\text{nodnz}$ // $b_i = b_i - l_{ij}x_j$, $\text{Tile}_t$
16:     means the submatrix consist of the nonzeros in Tile $t$
17: end function

Inherit from previous Tile
Load from main memory
Obtained from $x_j = b_j/l_{jj}$
Used in $b_i = b_i - l_{ij}x_j$

---

clarencewxl@gmail.com
Sparse Level Tile (SLT) Layout

- Manual Cache System
- Direct Memory Access (DMA)

- Sparse Level Tile layout
  - Make sure all the computation is cache-hit;
  - Replace fine-grained, random and unprefetchable memory access with course-grained, predictable and prefetchable memory access;
Producer-Consumer pairing method

- Register Communication

- Producer-Consumer pairing method:
  - Make communication cycle and random communication size not happen at the same time;
### Producer-Consumer pairing method

**Producers**

- **CPE 0,0**
- **CPE 0,3**
- **CPE 0,4**
- **CPE 0,7**

**Consumers**

- **CPE 7,0**
- **CPE 7,3**
- **CPE 7,4**
- **CPE 7,7**

**Equations**

- \( x_j = b_j / l_{jj} \)
- \( \Delta_{ij} = l_{ij} x_j \)
- \( b_i = b_i - \Delta_{ij} \)

**Values**

<table>
<thead>
<tr>
<th>Producers</th>
<th>Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPE 0,0</td>
<td>CPE 7,0</td>
</tr>
<tr>
<td>CPE 0,3</td>
<td>CPE 7,3</td>
</tr>
<tr>
<td>CPE 0,4</td>
<td>CPE 7,4</td>
</tr>
<tr>
<td>CPE 0,7</td>
<td>CPE 7,7</td>
</tr>
</tbody>
</table>

**Notes**

- **4 columns**
- **8 rows**

**Contact**

clarencewxl@gmail.com
Producer-Consumer pairing method

\[ x_i = \frac{b_i}{l_{ii}} \]

Producers

<table>
<thead>
<tr>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consumers

8 rows

clarencewxl@gmail.com
**Producer-Consumer pairing method**

Send $b$ from Consumers to Producers

\[ x_j = \frac{b_j}{l_{jj}}; \]

1. Acyclic communication
2. Pre-known communication size

<table>
<thead>
<tr>
<th>CPE 0,0</th>
<th>...</th>
<th>CPE 0,3</th>
<th>CPE 0,4</th>
<th>...</th>
<th>CPE 0,7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPE 7,0</td>
<td>...</td>
<td>CPE 7,3</td>
<td>CPE 7,4</td>
<td>...</td>
<td>CPE 7,7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Producer-Consumer pairing method

#### Share x across the same column

\[ \Delta_{ij} = l_{ij} x_j \]

1. Acyclic communication
2. Pre-known communication size

<table>
<thead>
<tr>
<th>CPE</th>
<th></th>
<th>CPE</th>
<th></th>
<th>CPE</th>
<th></th>
<th>CPE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Producer-Consumer pairing method**

Send $\Delta$ from Producers to Consumers

<table>
<thead>
<tr>
<th>CPE</th>
<th></th>
<th>CPE</th>
<th></th>
<th>CPE</th>
<th></th>
<th>CPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>⋮</td>
<td>0,3</td>
<td>⋮</td>
<td>0,4</td>
<td>⋮</td>
<td>0,7</td>
</tr>
<tr>
<td>7,0</td>
<td>⋮</td>
<td>7,3</td>
<td>⋮</td>
<td>7,4</td>
<td>⋮</td>
<td>7,7</td>
</tr>
</tbody>
</table>

\[ b_i = b_i - \Delta_{ij} \]

1. Acyclic communication
2. Pre-known communication size
• Sparse Level Tile layout
  • cache always hit;
  • course-grained memory access

• Producer-Consumer pairing method:
  • Dead-lock free;
## Experimental Setup

<table>
<thead>
<tr>
<th>Testbeds</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A single CG of SW26010 @ 34 GB/s</td>
<td></td>
</tr>
<tr>
<td>A single chip of NVIDIA K80 @ 240 GB/s</td>
<td></td>
</tr>
<tr>
<td>An Intel Xeon Phi 7120 KNC @ 352 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

- 3 platforms
## Experimental Setup

<table>
<thead>
<tr>
<th>Testbeds</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single CG of SW26010 @ 34 GB/s</td>
<td>1. A sequential method on MPE</td>
</tr>
<tr>
<td></td>
<td>2. A basic level-set method on CPEs</td>
</tr>
<tr>
<td></td>
<td>3. swSpTRSV method on CPEs</td>
</tr>
<tr>
<td>A single chip of NVIDIA K80 @ 240 GB/s</td>
<td>1. The method from cusparse v1</td>
</tr>
<tr>
<td></td>
<td>2. The method from cusparse v2</td>
</tr>
<tr>
<td></td>
<td>3. The synchronization-free method [1]</td>
</tr>
<tr>
<td>An Intel Xeon Phi 7120 KNC @ 352 GB/s</td>
<td>1. The method from Intel MKL</td>
</tr>
<tr>
<td></td>
<td>2. The P2P synchronization method [2]</td>
</tr>
</tbody>
</table>

- 3 platforms
- 3+3+2 = 8 methods

---


## Experimental Setup

<table>
<thead>
<tr>
<th>Testbeds</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single CG of SW26010 @ <strong>34 GB/s</strong></td>
<td>1. A sequential method on MPE</td>
</tr>
<tr>
<td></td>
<td>2. A basic level-set method on CPEs</td>
</tr>
<tr>
<td></td>
<td>3. swSpTRSV method on CPEs</td>
</tr>
<tr>
<td>A single chip of NVIDIA K80 @ <strong>240 GB/s</strong></td>
<td>1. The method from cusparse v1</td>
</tr>
<tr>
<td></td>
<td>2. The method from cusparse v2</td>
</tr>
<tr>
<td></td>
<td>3. The synchronization-free method [1]</td>
</tr>
<tr>
<td>An Intel Xeon Phi 7120 KNC @ <strong>352 GB/s</strong></td>
<td>1. The method from Intel MKL</td>
</tr>
<tr>
<td></td>
<td>2. The P2P synchronization method [2]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Parallelism</th>
<th>#Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$[2^0, 2^5]$</td>
<td>15.97</td>
</tr>
<tr>
<td>B</td>
<td>$[2^5, 2^{10}]$</td>
<td>287.59</td>
</tr>
<tr>
<td>C</td>
<td>$[2^{10}, 2^{15}]$</td>
<td>7064.68</td>
</tr>
<tr>
<td>D</td>
<td>$[2^{15}, 2^{20}]$</td>
<td>358216.55</td>
</tr>
<tr>
<td>Total</td>
<td>$[2^0, 2^{20}]$</td>
<td>30010.37</td>
</tr>
</tbody>
</table>

Parallelism = #nonzeros/#levels

- 3 platforms
- 3+3+2 = 8 methods
- 2057 benchmarks from the University of Florida Sparse Matrix Collection


The cheapest cost is **0.3ms**, and the most expensive cost is **28.2s**. The harmonic average is **3.3ms**.
Methods on Sunway Processor

1. Best performance: **3406 MFlops**
2. Compared with sequential method:
   - Average speedup: **7.8**;
   - Best speedup: **117.3**;
3. Compared with level-set method:
   - Average speedup: **6.9**;
   - Best speedup: **38.5**;

Group A

Group B

Group C

Group D
The power of 20 typical benchmarks

The largest power is **38.18 Watt**. And the best performance/power can reach **89.22 Mflops/W**.
Different Methods on Different Processors

- Outperform MKL and P2P method on KNC in 1856 benchmarks
- Outperform cuSparse and SyncFree method on K80 in 1672 benchmarks
- swSpTRSV can achieve the best performance in 1624 benchmarks

<table>
<thead>
<tr>
<th>Method</th>
<th>Group A</th>
<th>Group B</th>
<th>Group C</th>
<th>Group D</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKL</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>P2P</td>
<td>0</td>
<td>3</td>
<td>49</td>
<td>25</td>
<td>77</td>
</tr>
<tr>
<td>cuSparse v1</td>
<td>0</td>
<td>0</td>
<td>66</td>
<td>107</td>
<td>173</td>
</tr>
<tr>
<td>cuSparse v2</td>
<td>0</td>
<td>10</td>
<td>143</td>
<td>6</td>
<td>159</td>
</tr>
<tr>
<td>SyncFree</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>swSpTRSV</td>
<td>225</td>
<td>1002</td>
<td>376</td>
<td>21</td>
<td>1624</td>
</tr>
<tr>
<td>Total</td>
<td>249</td>
<td>1015</td>
<td>634</td>
<td>159</td>
<td>2057</td>
</tr>
</tbody>
</table>

- The number of benchmarks in each group that one method can achieve the best performance compared with other methods

clarencexwl@gmail.com
Conclusion

Method:
1. Sparse Level Tile layout;
   • Manual Cache System
   • Direct Memory Access (DMA)
2. Producer-Consumer pairing method;
   • Register Communication

Performance:
• An average speedup of 7.8, compared with the sequential method on MPE;
• An average speedup of 6.9, compared with the basic level-set method on CPEs;
• Achieve the best performance in 1624/2057 (78.95%) benchmarks
Code: https://github.com/clarencewxl/swSpTRSV.git

Sparse triangular solve for Sunway architecture

swSpTRSV

Sparse triangular solve for Sunway architecture

1. Login to Sunway TaihuLight
   - Land the homepage of the National Supercomputing Center in Wuxi: http://www.nsccwx.cn/wxcyw/
   - Select one VPN service: ‘Telecom’, ‘Unicom’ or ‘China Mobile’ on the top of the website. Please choose the best one for better connection.
   - Login to Sunway TaihuLight Supercomputer: ssh 41.0.0.188

Please email to clarencewxl@gmail.com for further questions.
Login: http://www.nsccwx.cn/wxcyw/
Thanks, Q&A

Welcome to Wuxi!
Welcome to TaihuLight!