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

Xinliang Wang,[1,3] Weifeng Liu,[2] Wei Xue,[1,3] Li Wu[1,3]

1 

2 

3 

Outline

1. Background
2. Sunway architecture
3. Sparse Level Tile layout
4. Producer-Consumer Pairing method
5. Experiment
6. Conclusion

Sparse Triangular Solve

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:

$$\begin{cases} 1*x_0 = a \\ 1*x_1 = b \\ 2*x_1+1*x_2 = c \\ 3*x_0+1*x_3 = d \end{cases}$$

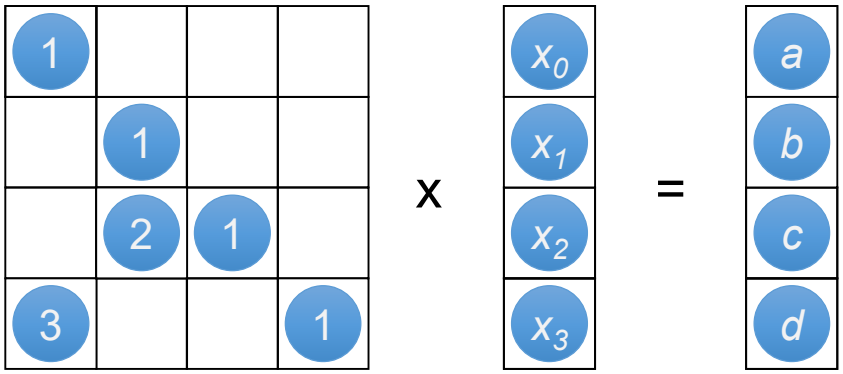
solution:

$$\begin{cases} x_0 = a \\ x_1 = b \\ x_2 = c - 2b \\ x_3 = d - 3a \end{cases}$$

Sparse Triangular Solve

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	x	b
(4x4)	(4x1)	(4x1)
nnzL = 6	*dense*	*dense*
known	*unknown*	*known*

system:

$$\begin{cases} 1*x_0 = a \\ 1*x_1 = b \\ 2*x_1+1*x_2 = c \\ 3*x_0+1*x_3 = d \end{cases}$$

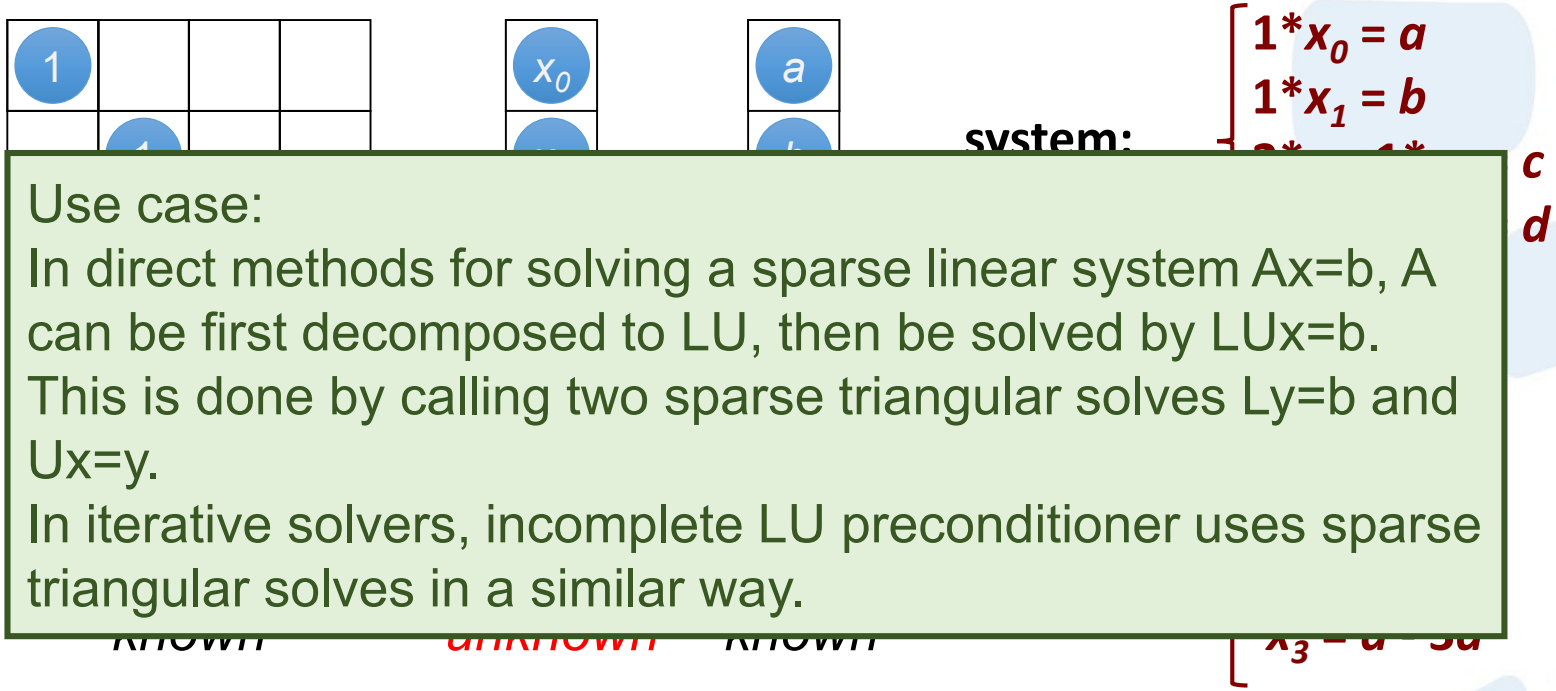
solution:

$$\begin{cases} x_0 = a \\ x_1 = b \\ x_2 = c - 2b \\ x_3 = d - 3a \end{cases}$$

Sparse Triangular Solve

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.

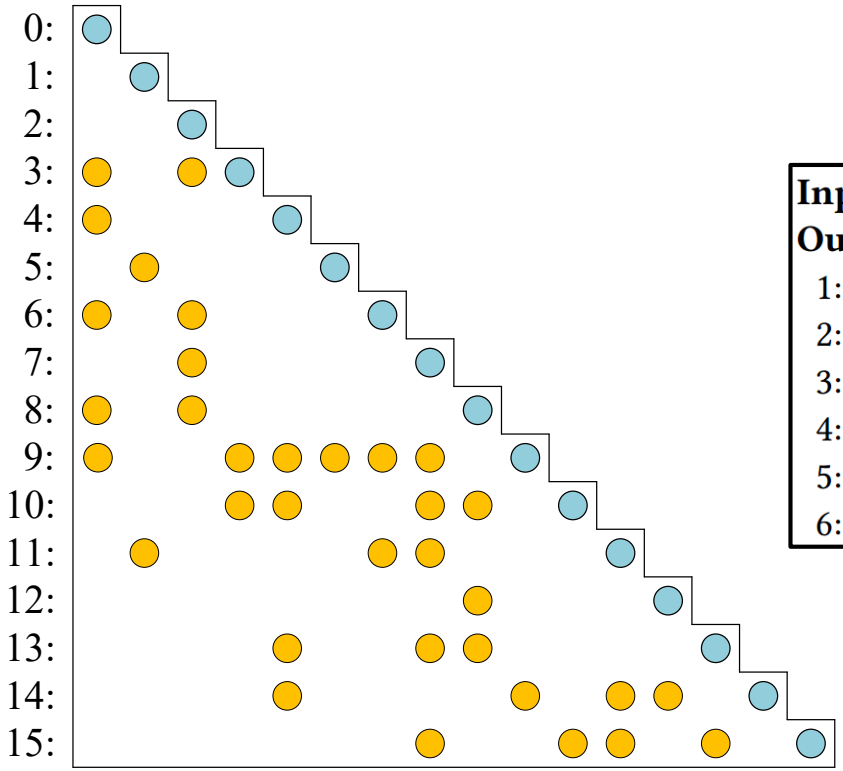


Use case:

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.

Single core: Sequential method



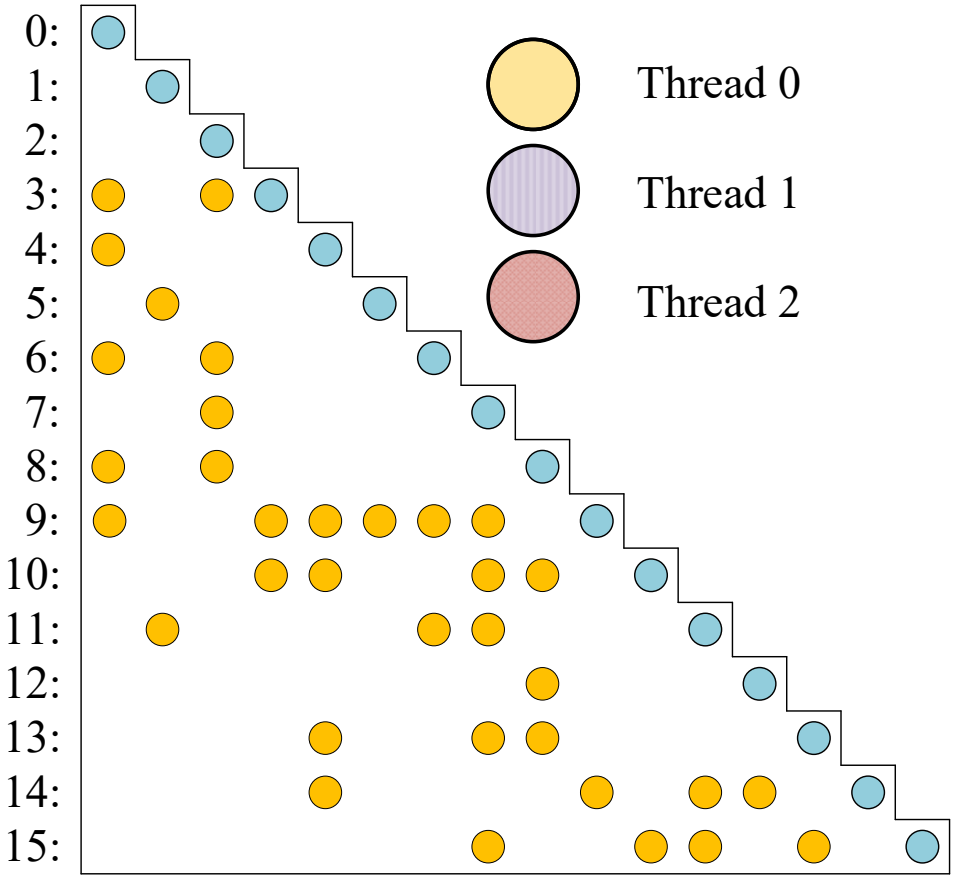
$$Lx = b$$

Input: $*col\_ptr, *row\_idx, *val, *b;$
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
```

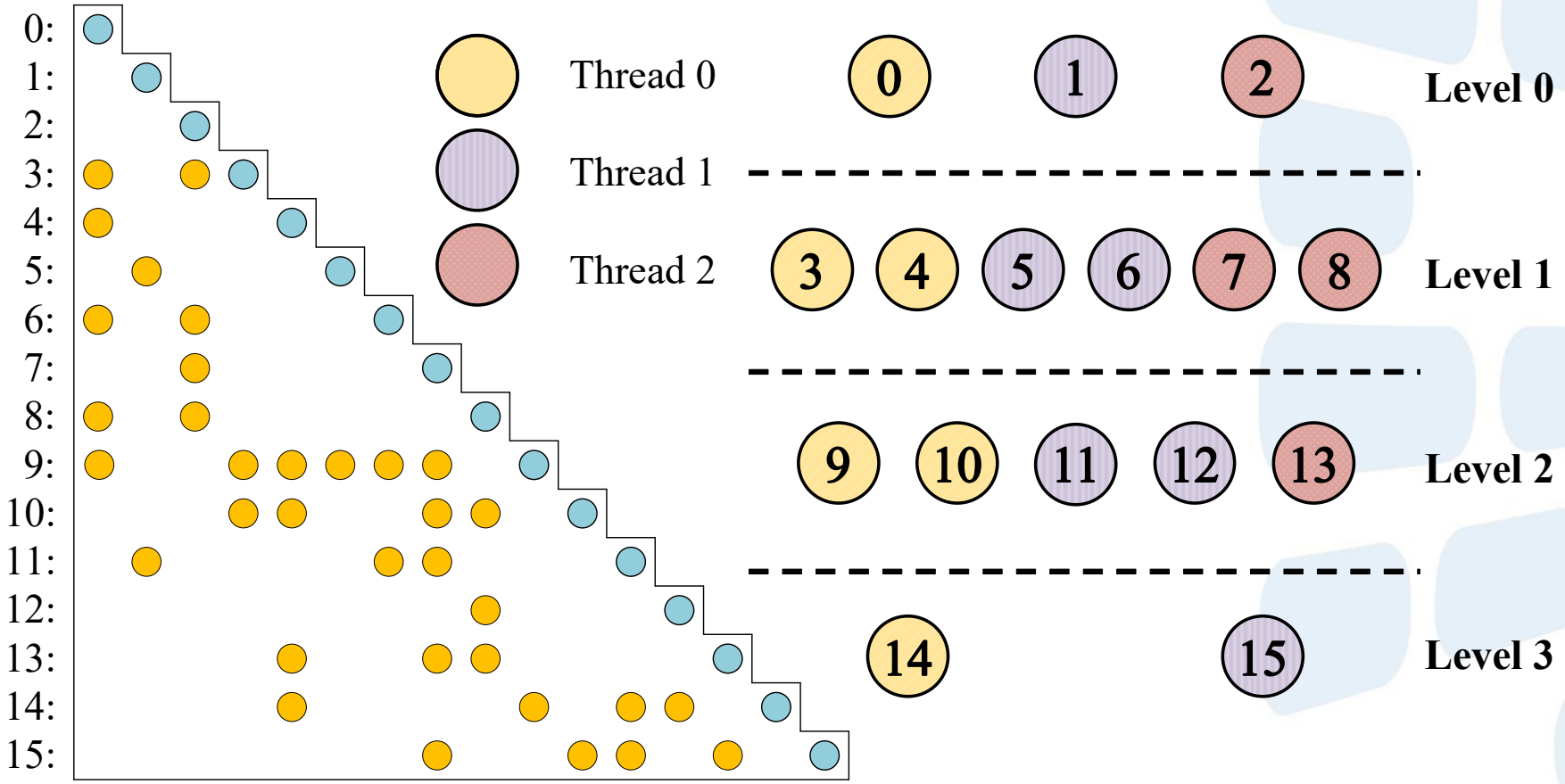
A sequential method based on CSC layout

A few cores: Level-set method



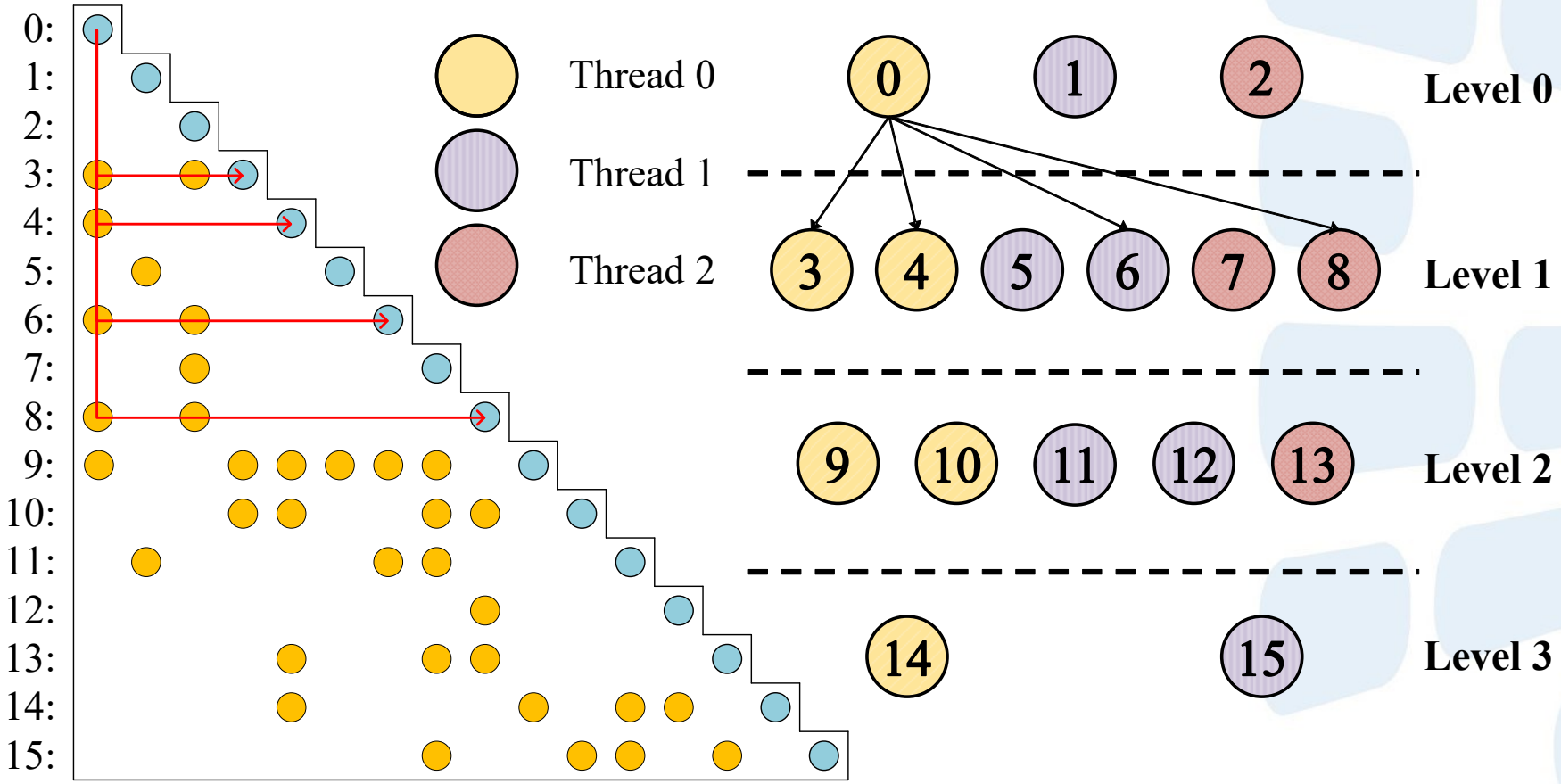
Parallel in each level and sequential inter level

A few cores: Level-set method



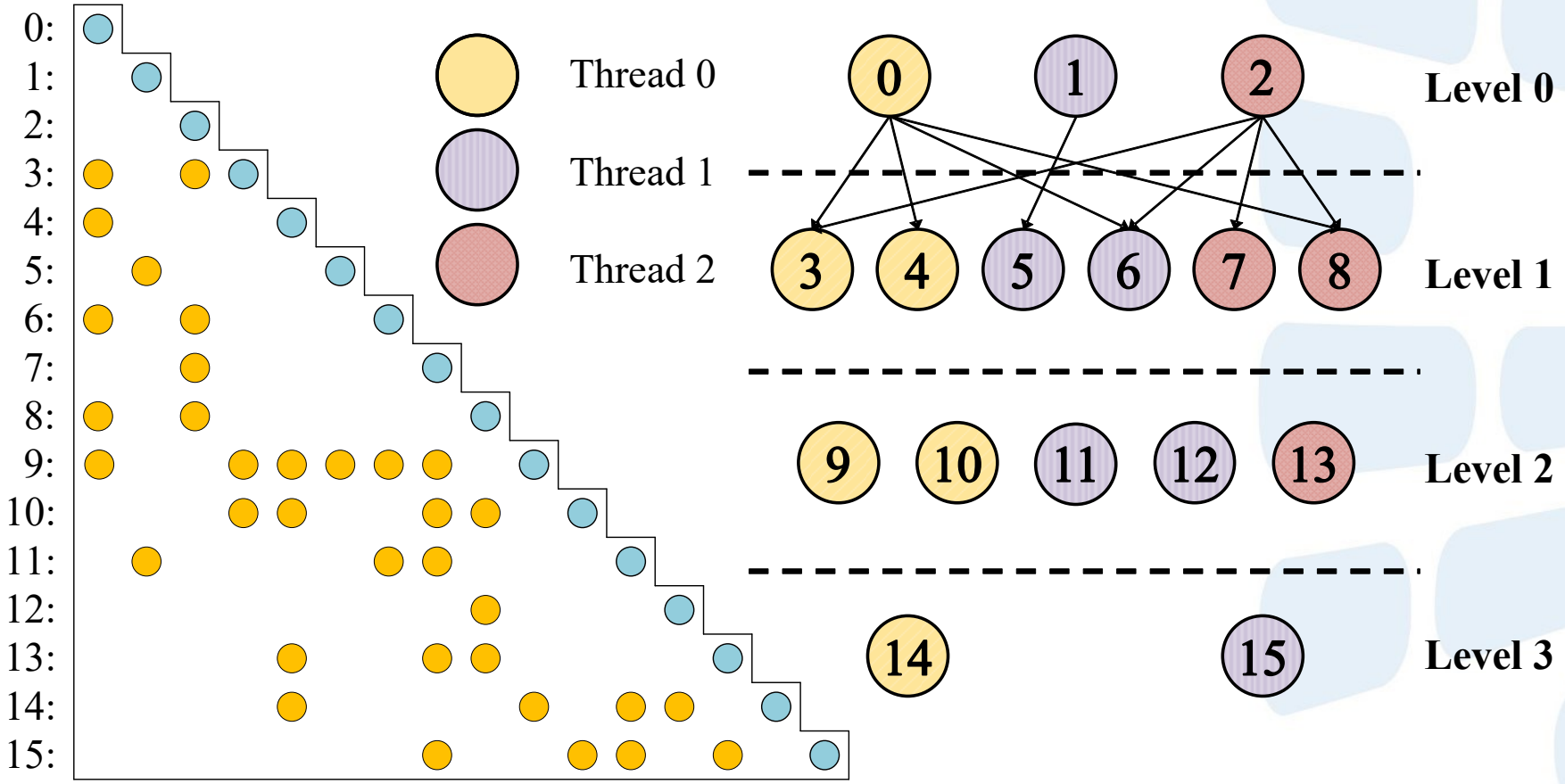
Parallel in each level and sequential inter level

A few cores: Level-set method



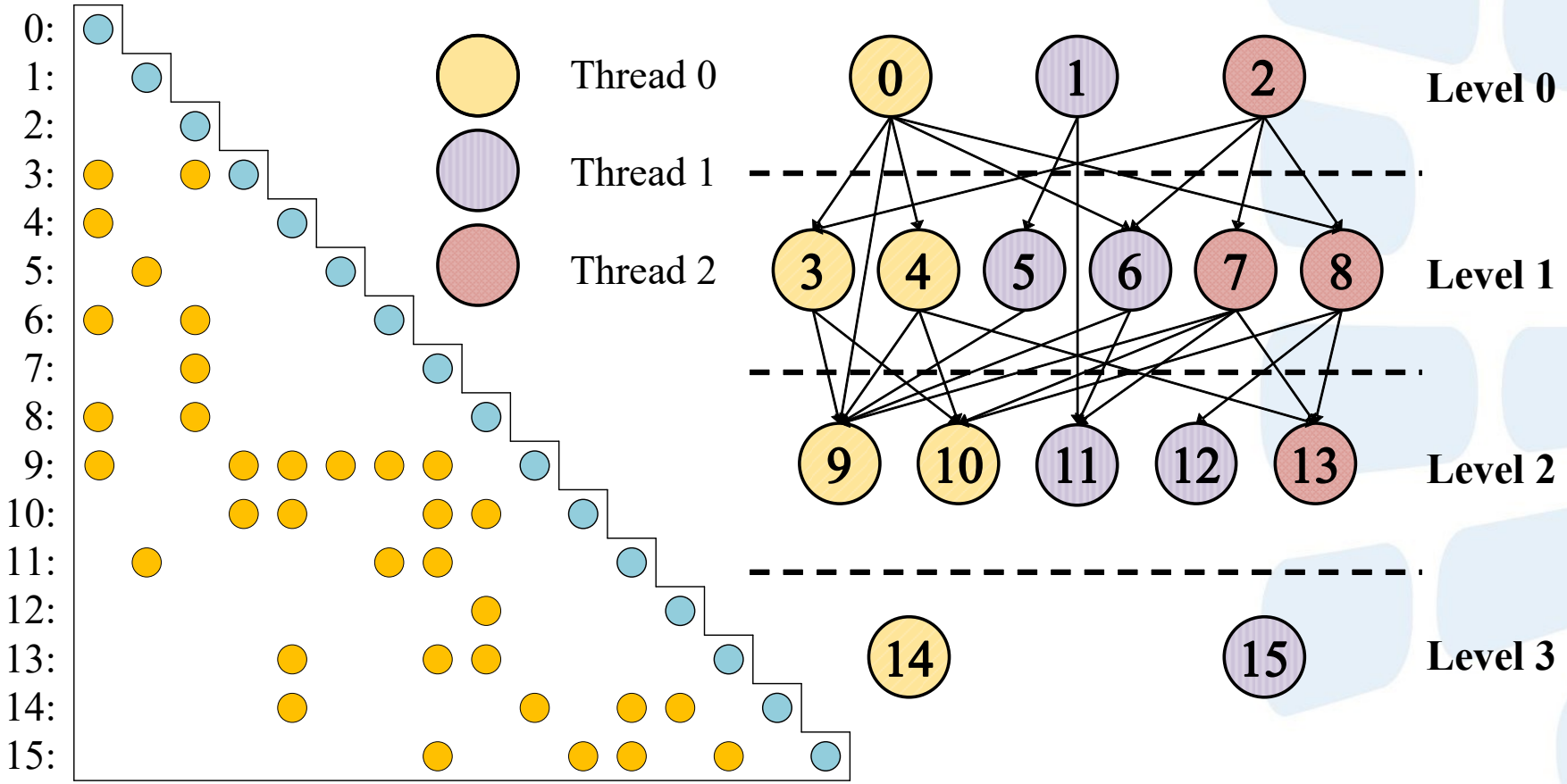
Parallel in each level and sequential inter level

A few cores: Level-set method



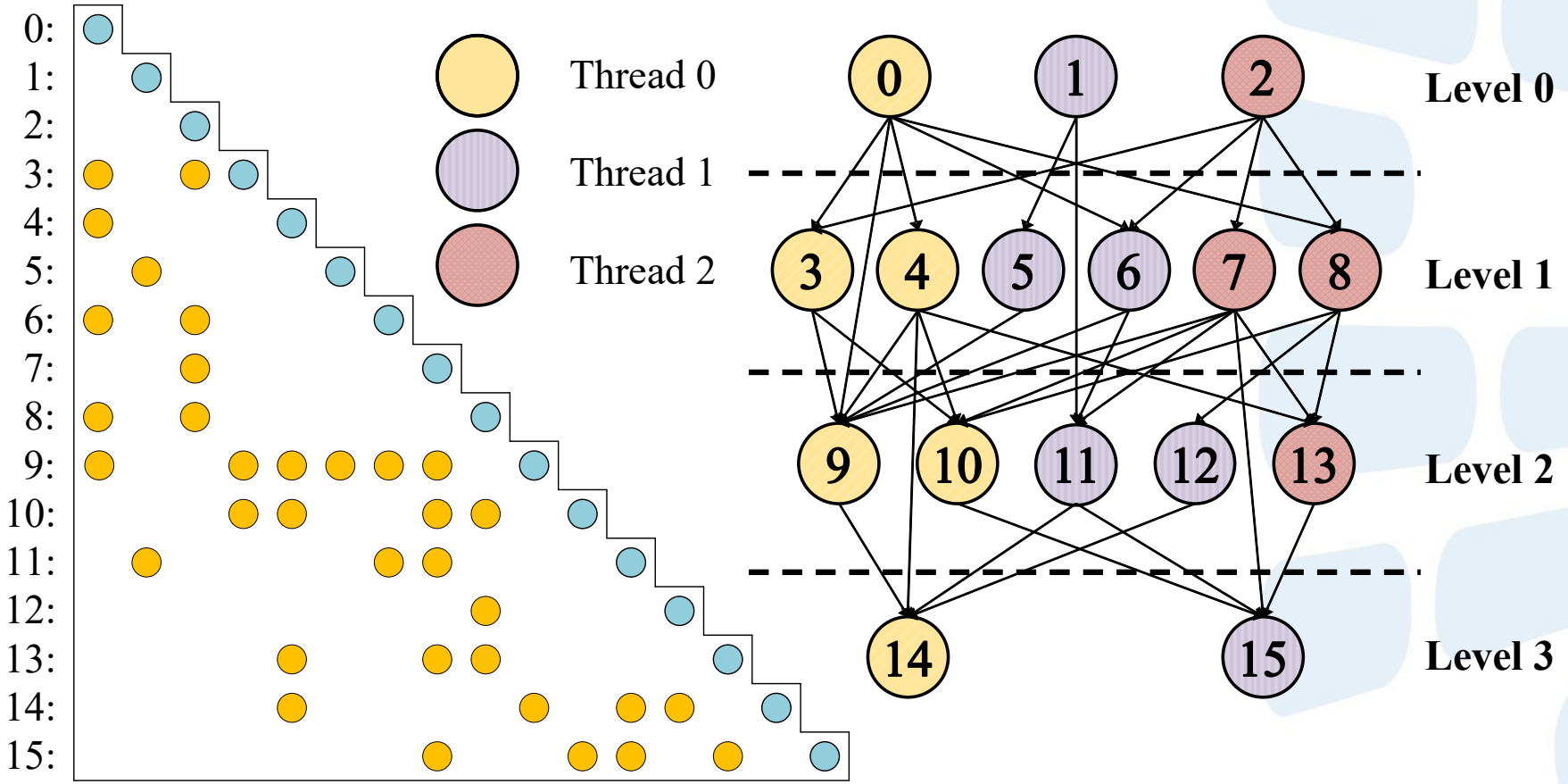
Parallel in each level and sequential inter level

A few cores: Level-set method



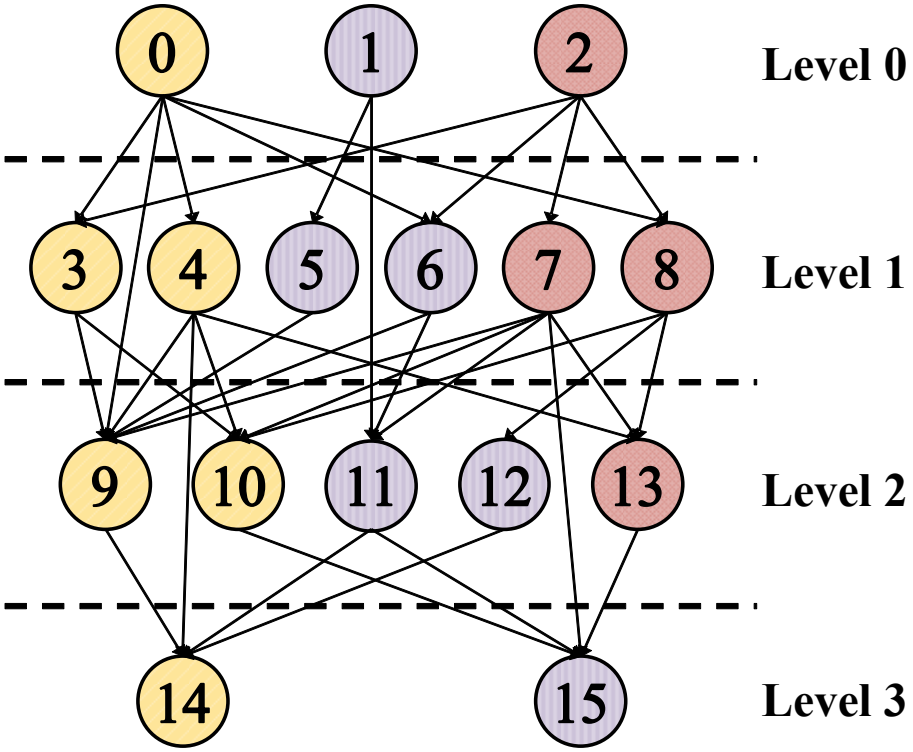
Parallel in each level and sequential inter level

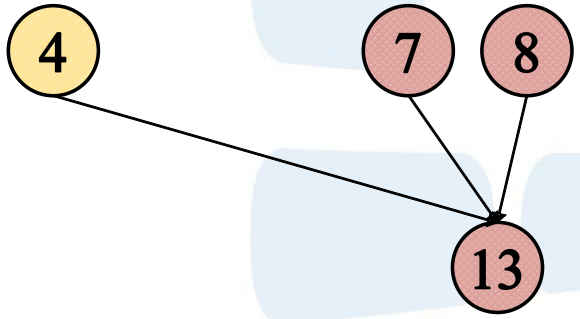
A few cores: Level-set method



Parallel in each level and sequential inter level

More cores: P2P method (CPU/MIC)

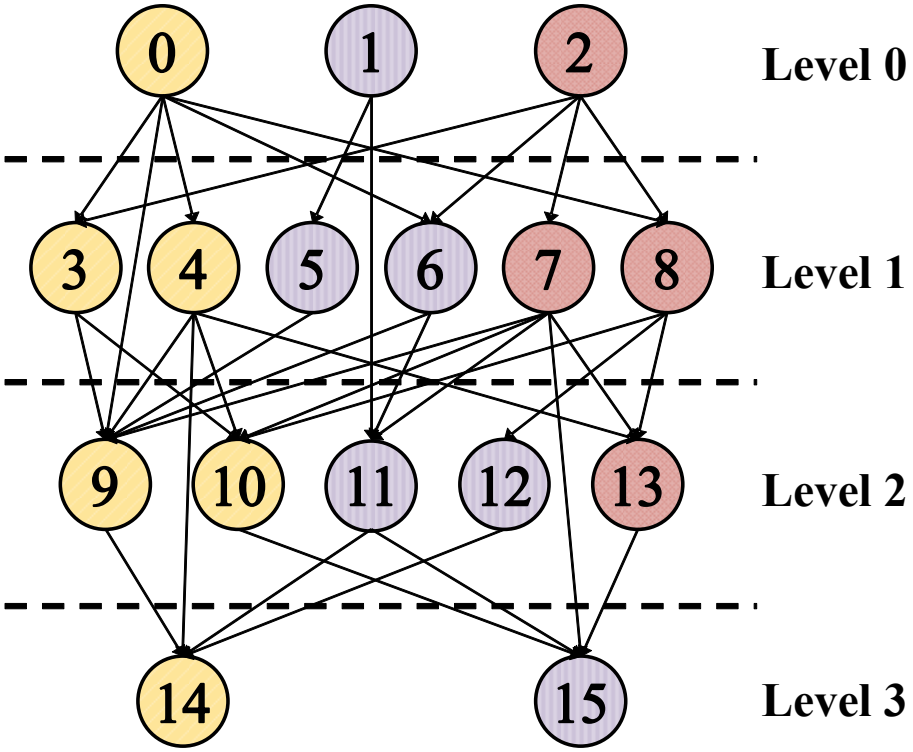


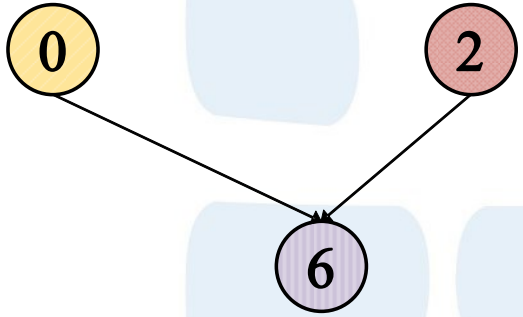


- No full-synchronization
- Only synchronize between Thread 0 and Thread 2

Park J, et al. Sparsifying synchronization for high-performance shared-memory sparse triangular solver[C] International Supercomputing Conference. Springer, Cham, 2014: 124-140.

More cores: Sync-free method (GPU)



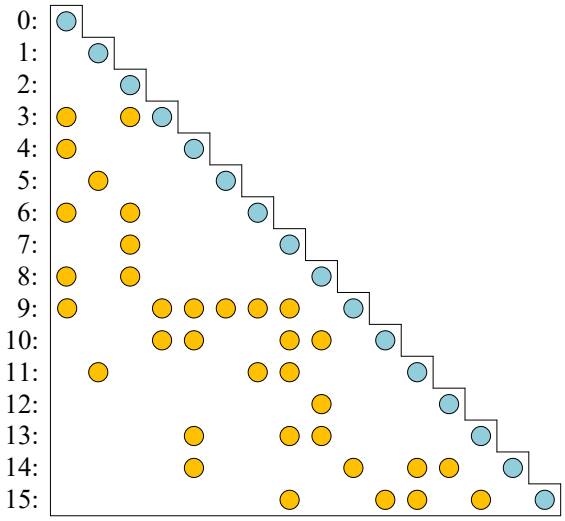


- Thread 0 and 2 modify the same value by atomic operations.

Liu W, et al. A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves[C] European Conference on Parallel Processing. Springer International Publishing, 2016: 617-630.

Background

Problem

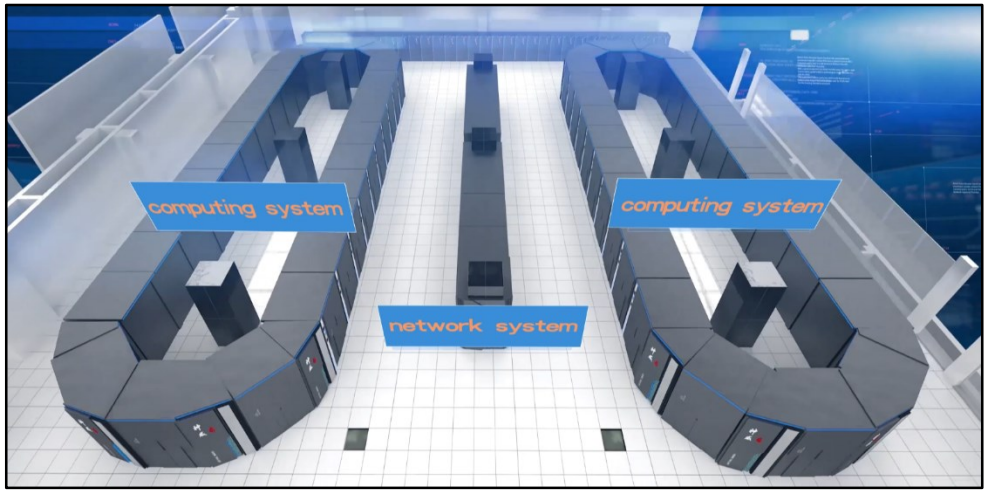


Architecture

Sparse Triangular Solve

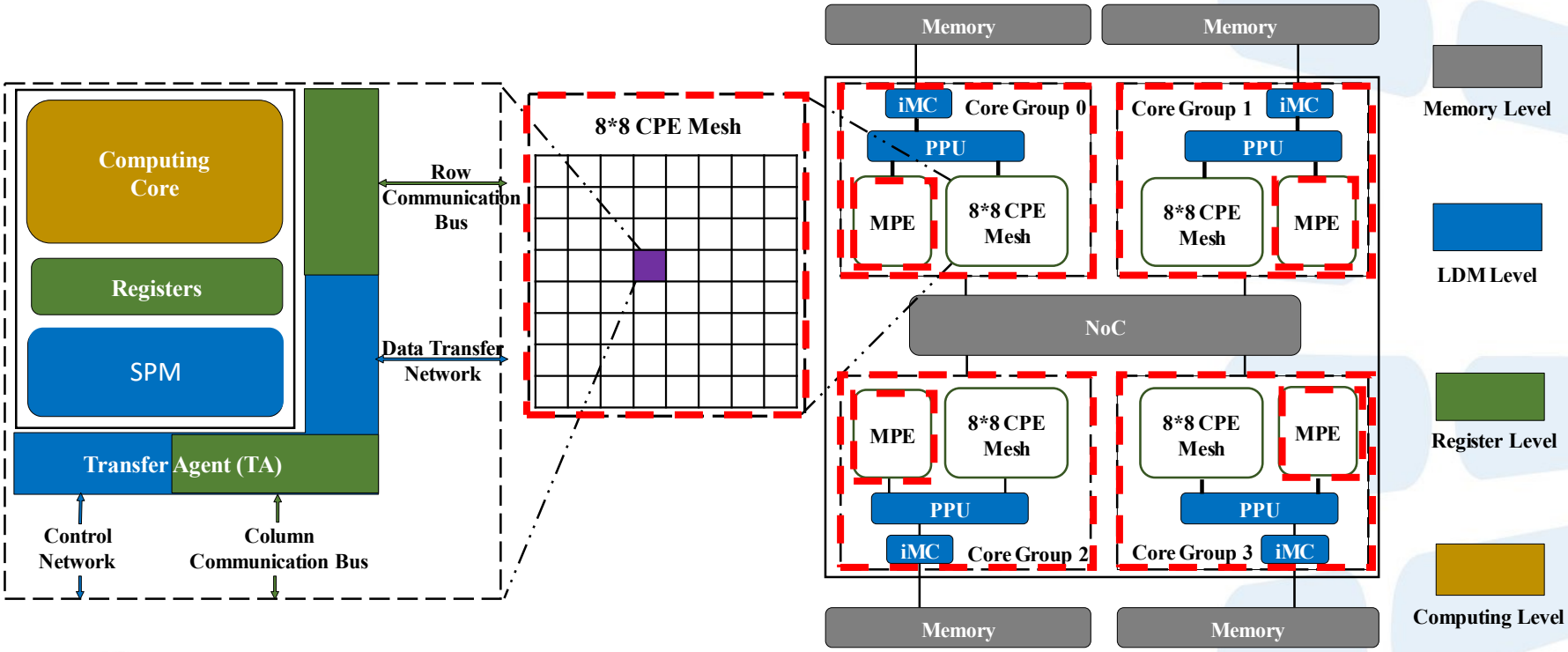
Sunway Processor

Sunway TaihuLight: Overview



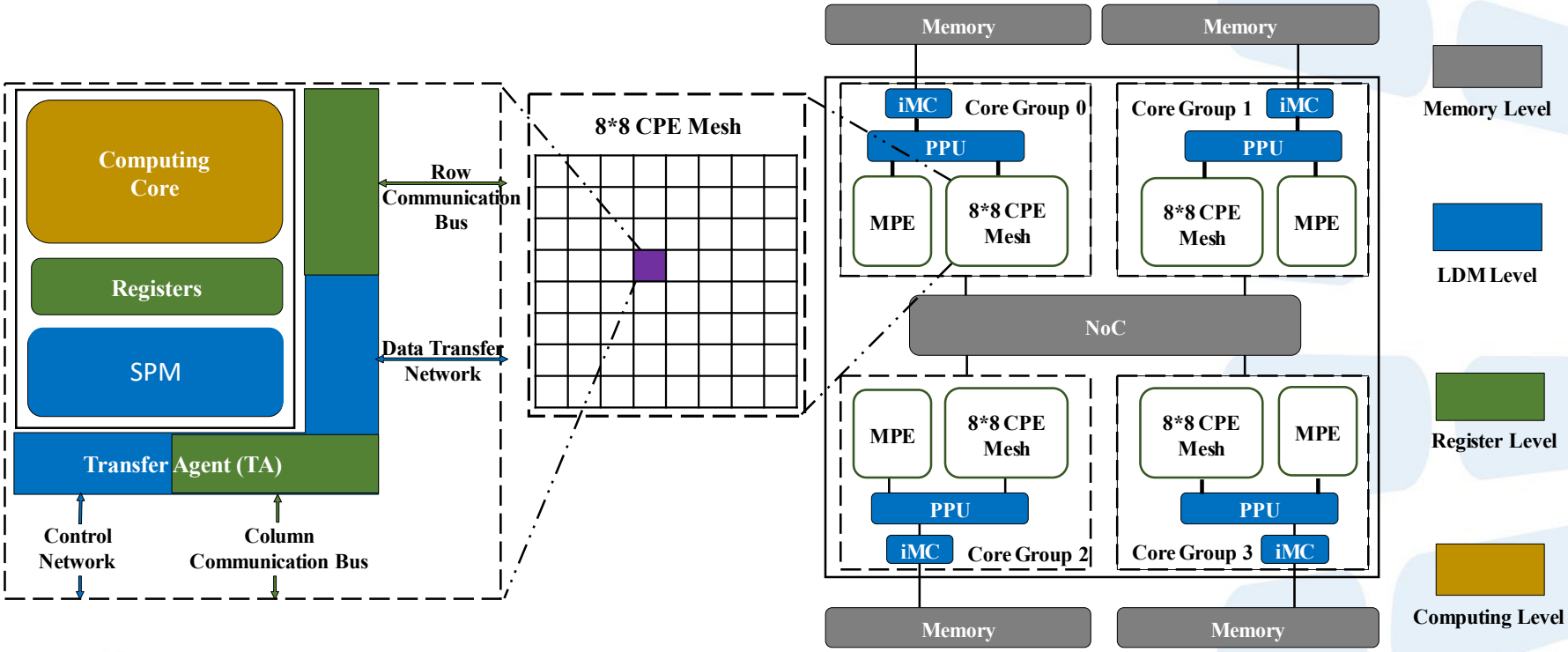
Entire System	
Peak Performance	125 PFlops
Linpack Performance	93 Pflops / 74.4%
Total Memory	1310.72 TB
Total Memory Bandwidth	5591.45 TB/s
# nodes	40,960
# cores	10,649,600

SW26010 Processor



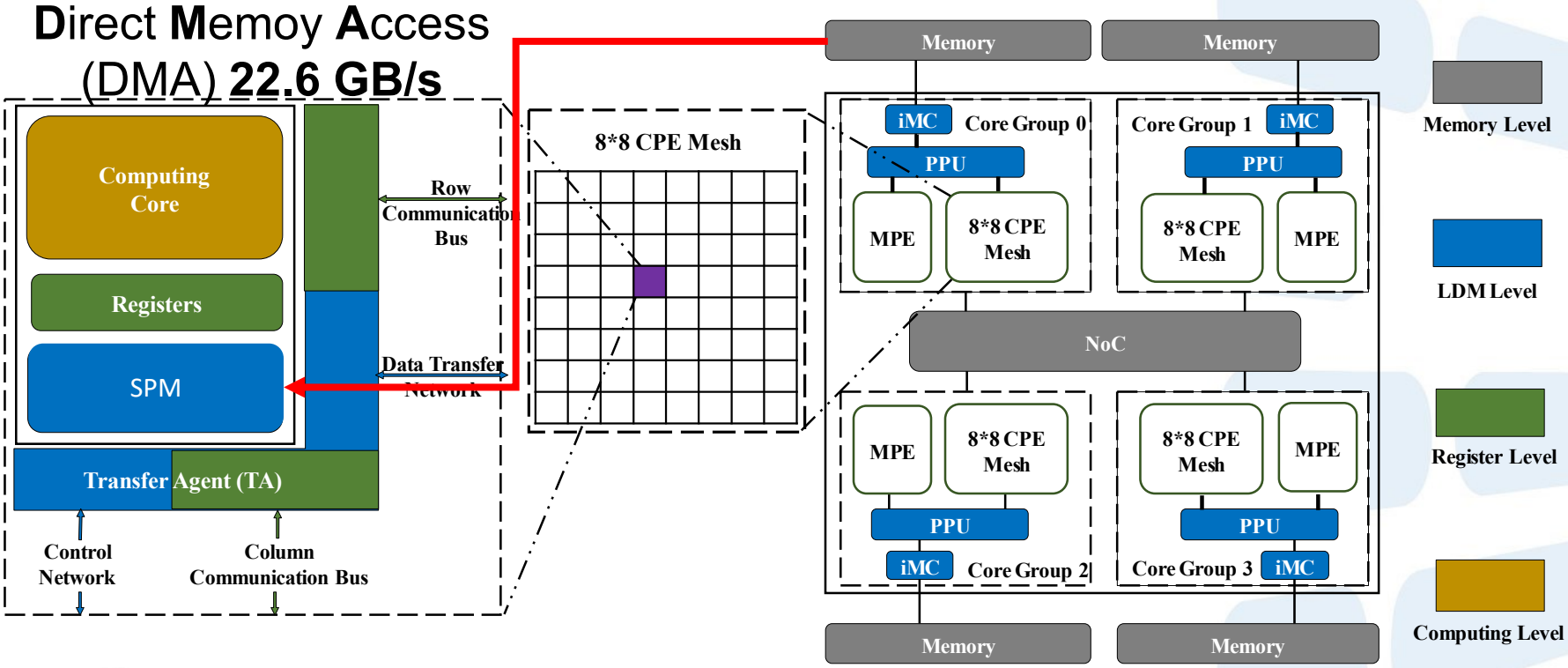
Name	MPE	CPE
Frequency (GHz)	1.45	1.45
Cores	1	64
SIMD width	4 double / 8 integer	4 double / 8 integer
Peak Gflop/s in D.P.	23.2	742.4
Data L1 / L2 cache	32KB / 256KB	64KB (SPM) / -
Instruction L1 / L2 cache	32KB / +	16KB / 64KB
Memory capacity (GB)	32 (shared)	

SW26010 Processor



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

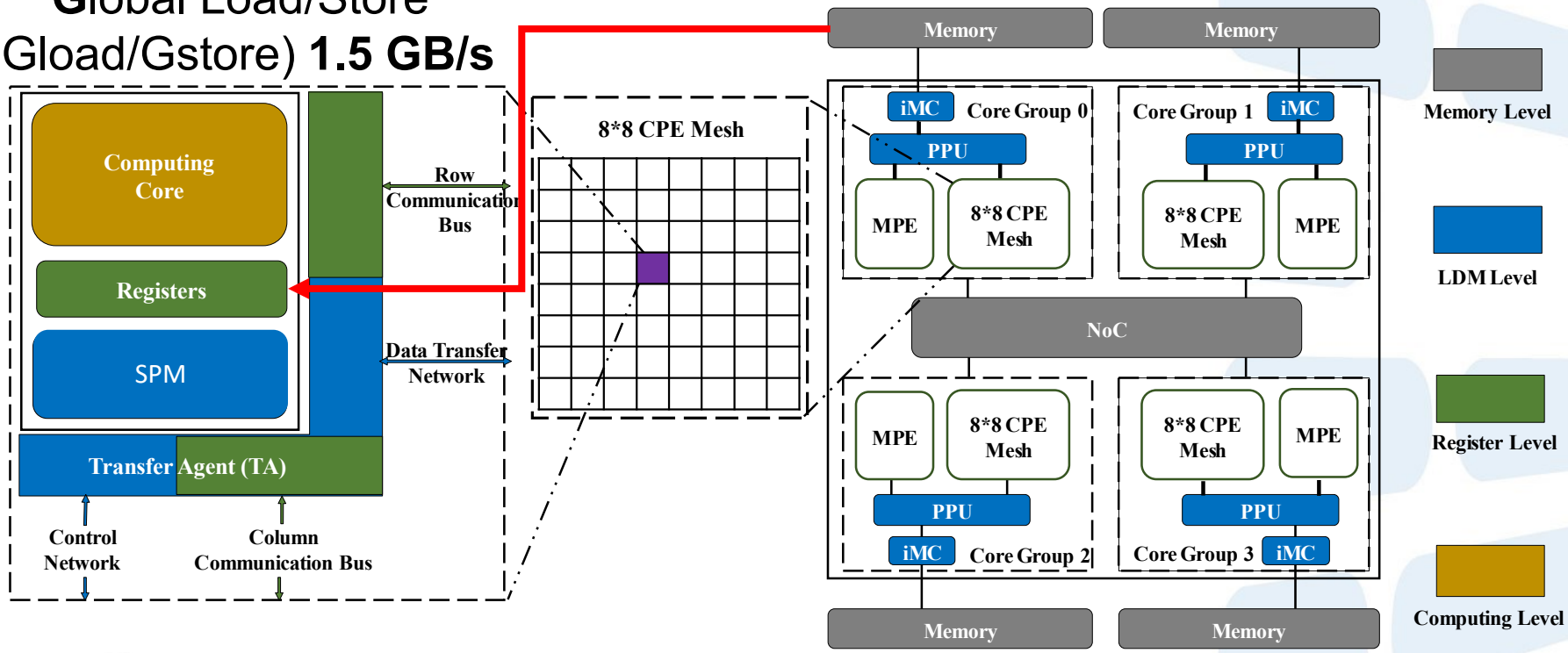
SW26010 Processor



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

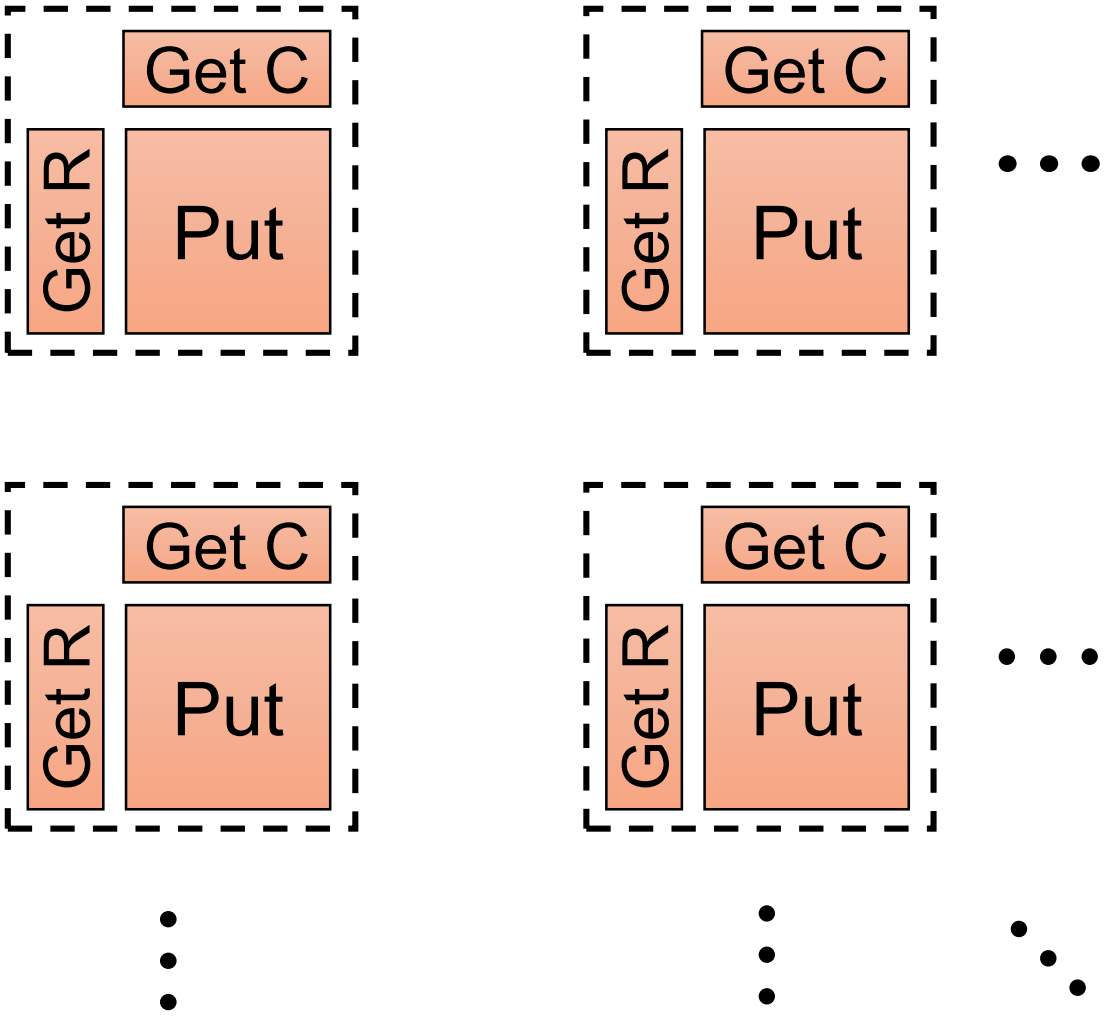
SW26010 Processor

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

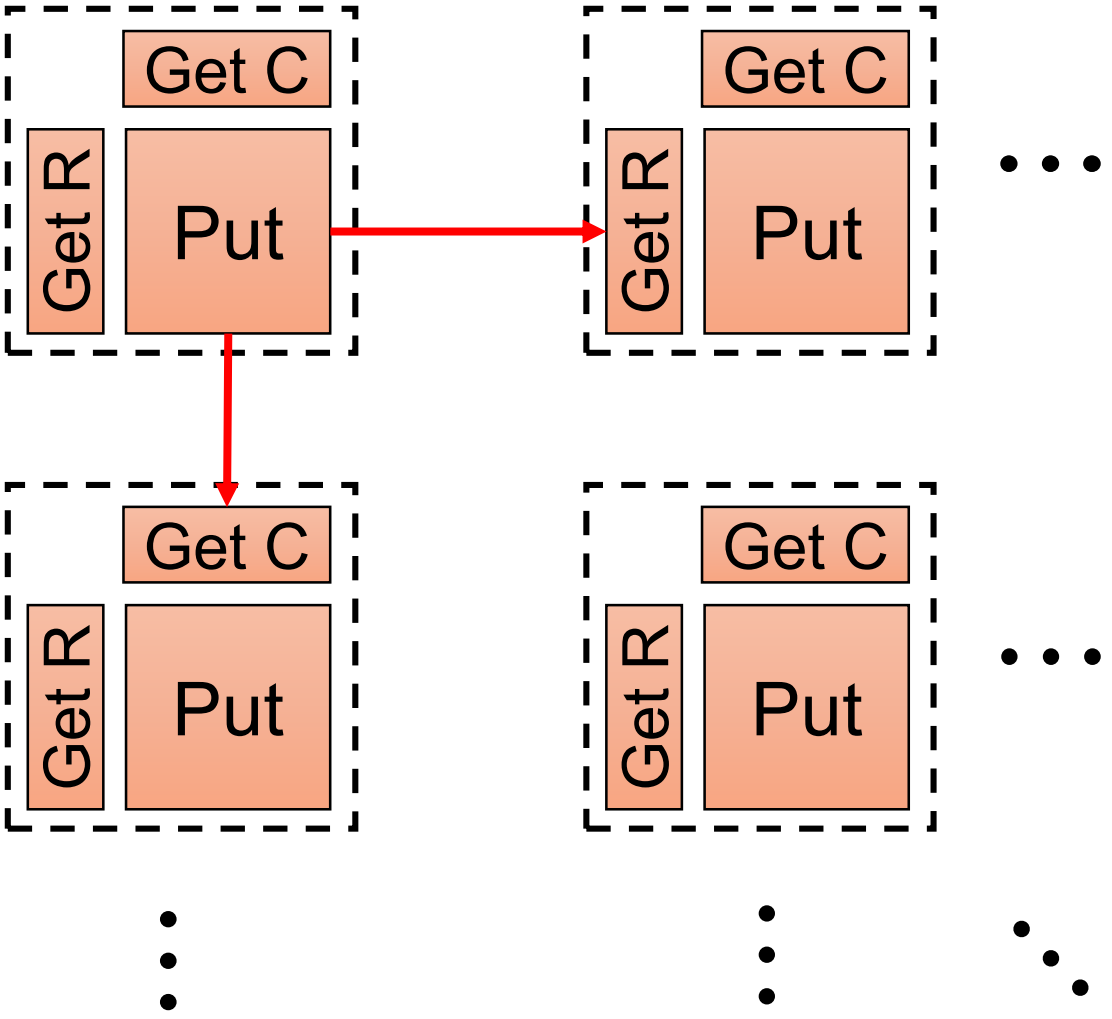


Name	MPE	CPE
Frequency (GHz)	1.45	1.45
Cores	1	64
SIMD width	4 double / 8 integer	4 double / 8 integer
Peak Gflop/s in D.P.	23.2	742.4
Data L1 / L2 cache	32KB / 256KB	64KB (SPM) / -
Instruction L1 / L2 cache	32KB / +	16KB / 64KB
Memory capacity (GB)	32 (shared)	

Register Communication



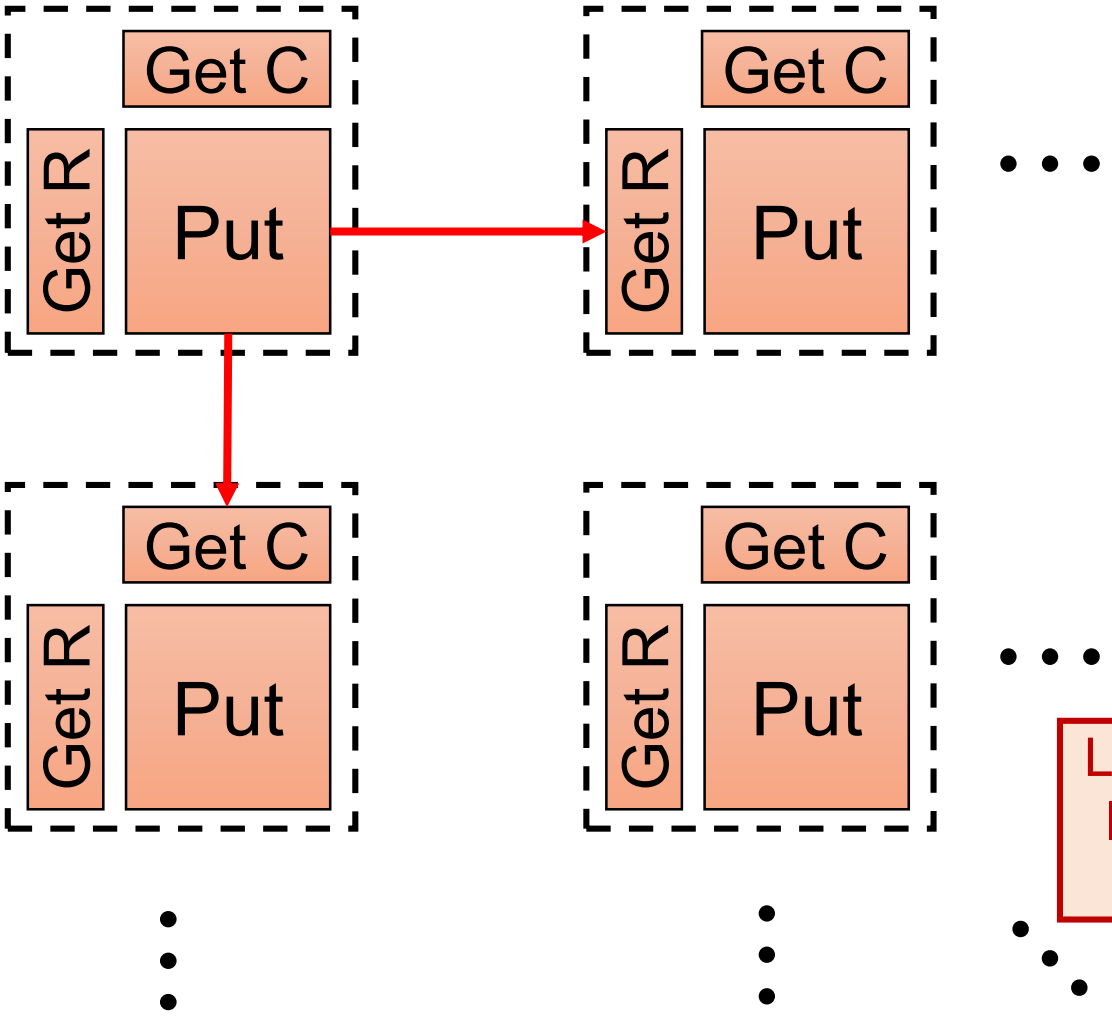
Register Communication



putr ←→ getr

putc ←→ getc

Register Communication




```
//P2P Test
if (id%2 == 0)
    while(1)
        putr(data, id+1);
else
    while(1)
        getr(&data);
```

Latency: less than **11** cycles
Integrated Bandwidth: **637** GB/s

Xu, Zhigeng, James Lin, and Satoshi Matsuoka. "Benchmarking SW26010 Many-Core Processor." *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 2017.

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

Input: $*col\_ptr$, $*row\_idx$, $*val$, $*b$;
Output: $*x$;

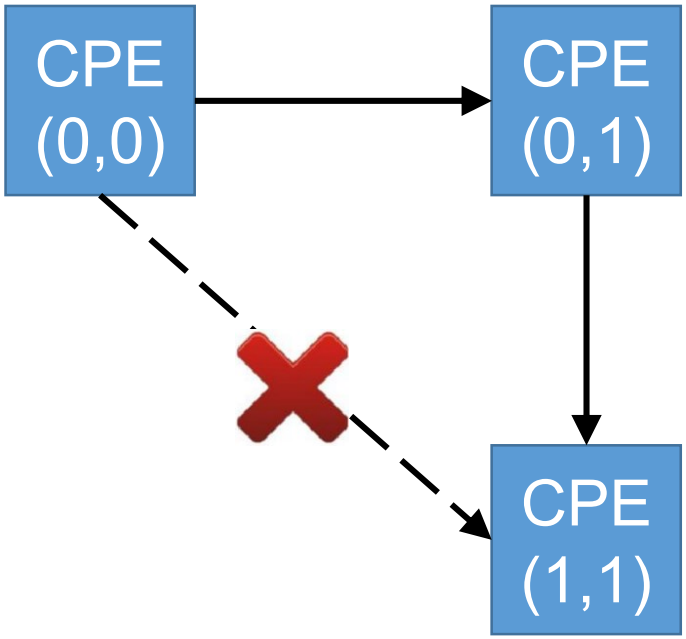
1: **for** $i = 0$ to $n - 1$ **do**
2: $x[i] = b[i]/val[col\_ptr[i]]$;
3: **for** $j = col\_ptr[i] + 1$ to $col\_ptr[i + 1] - 1$ **do**
4: **if** $b[row\_idx[j]]$ is not in SPM **then**
5: **Swap** $b[row\_idx[j]]$ into SPM
6: **end if**
7: $b[row\_idx[j]] = b[row\_idx[j]] - val[j] * x[i]$;
8: **end for**
9: **end for**

- **Manual cache system**
- **Direct memory access**
- Register communication

- Cost much even cache hit
- Hurt the instruction pipeline
- Difficult to prefetch

Mismatch between SpTRSV and Sunway

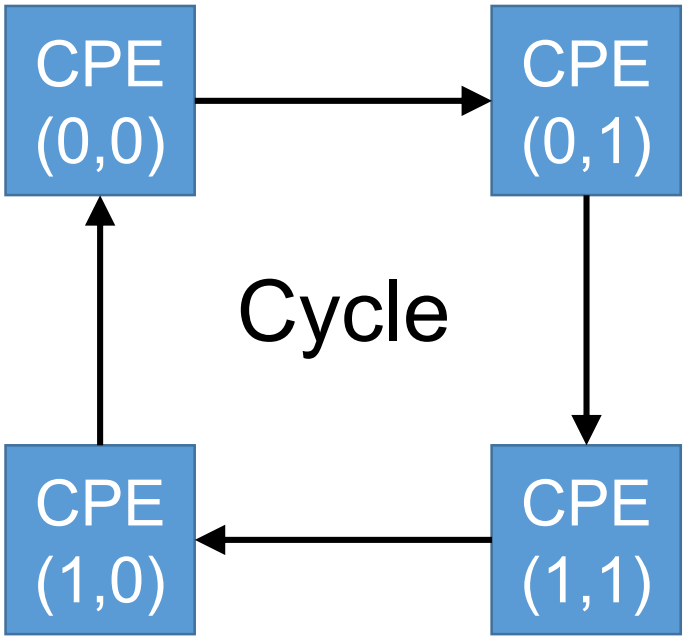
Limitation of register communication: only happen in the same column or row



- Manual cache system
- Direct memory access
- **Register communication**

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**

Lin H, et al. Scalable Graph Traversal on Sunway TaihuLight with Ten Million Cores[C] Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International. IEEE, 2017: 635-645.

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)

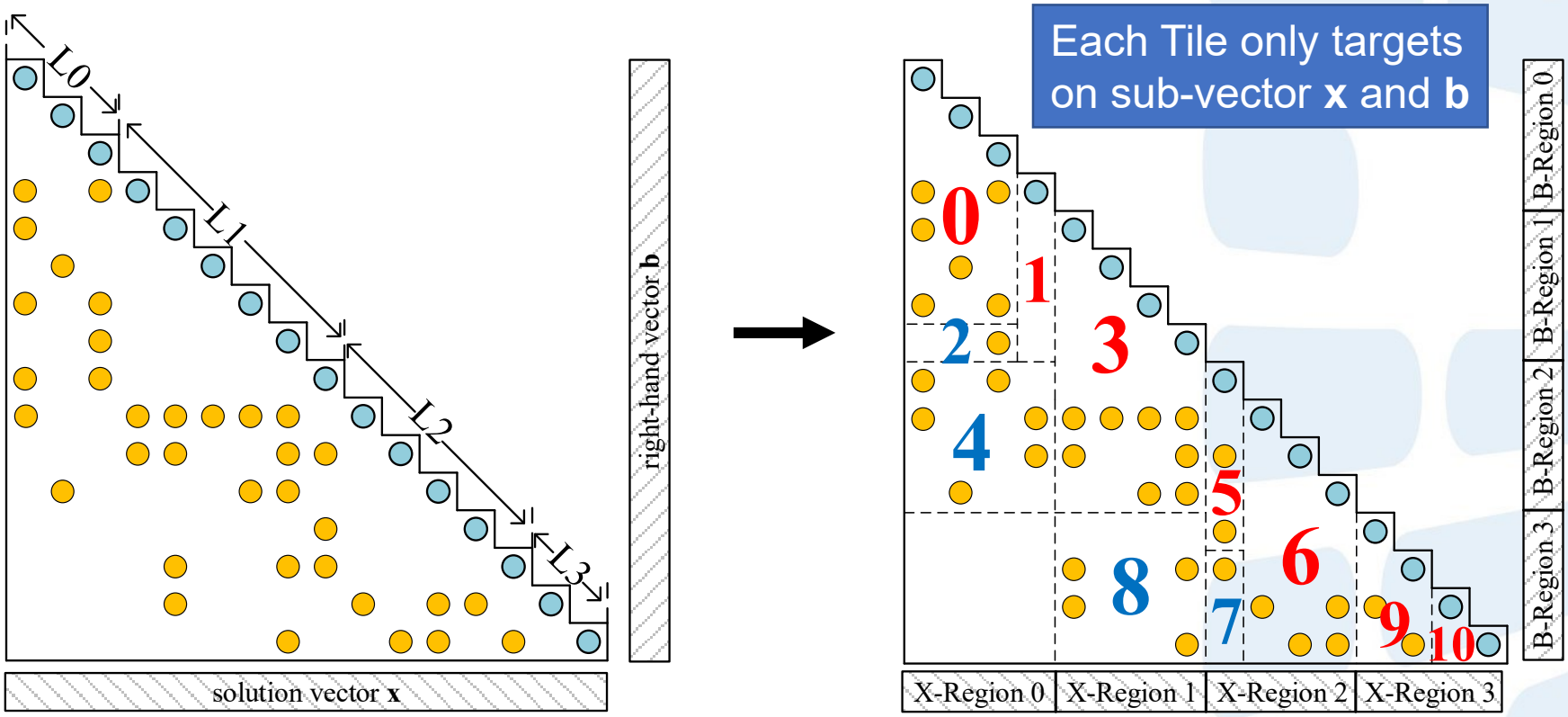
- Sparse Level Tile layout
  - Make sure all the computation is cache-hit;
  - Replace fine-grained, random and unprefetchable memory access with course-grained, predictable and prefetchable memory access;

Contributions

- Register Communication

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

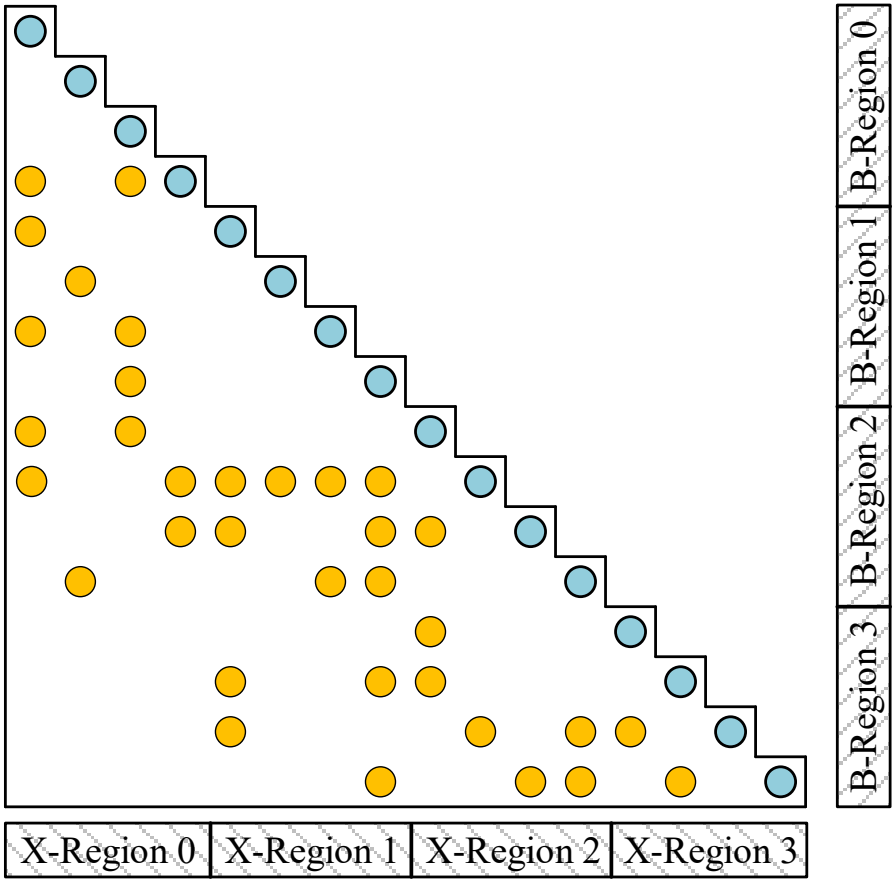
Sparse Level Tile (SLT) Layout



fine-grained, random, unprefetchable

course-grained, predictable, prefetchable

Sparse Level Tile (SLT) Layout: Step 1

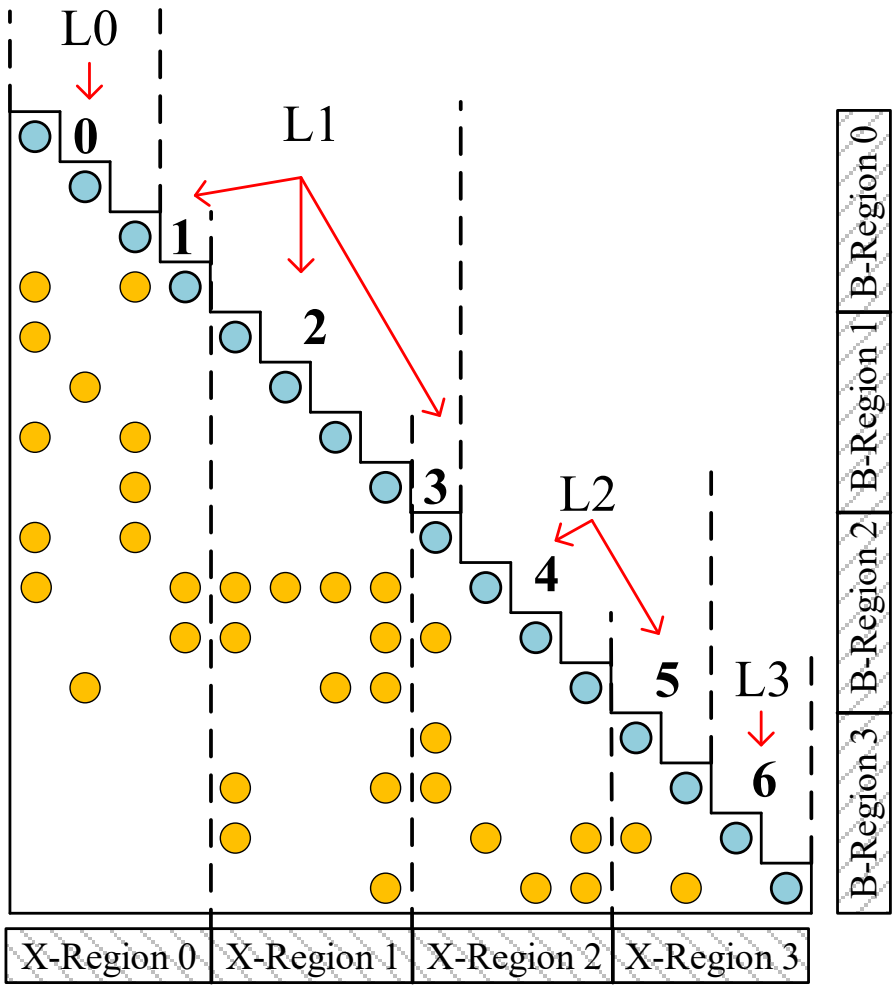


Step 1:

- 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

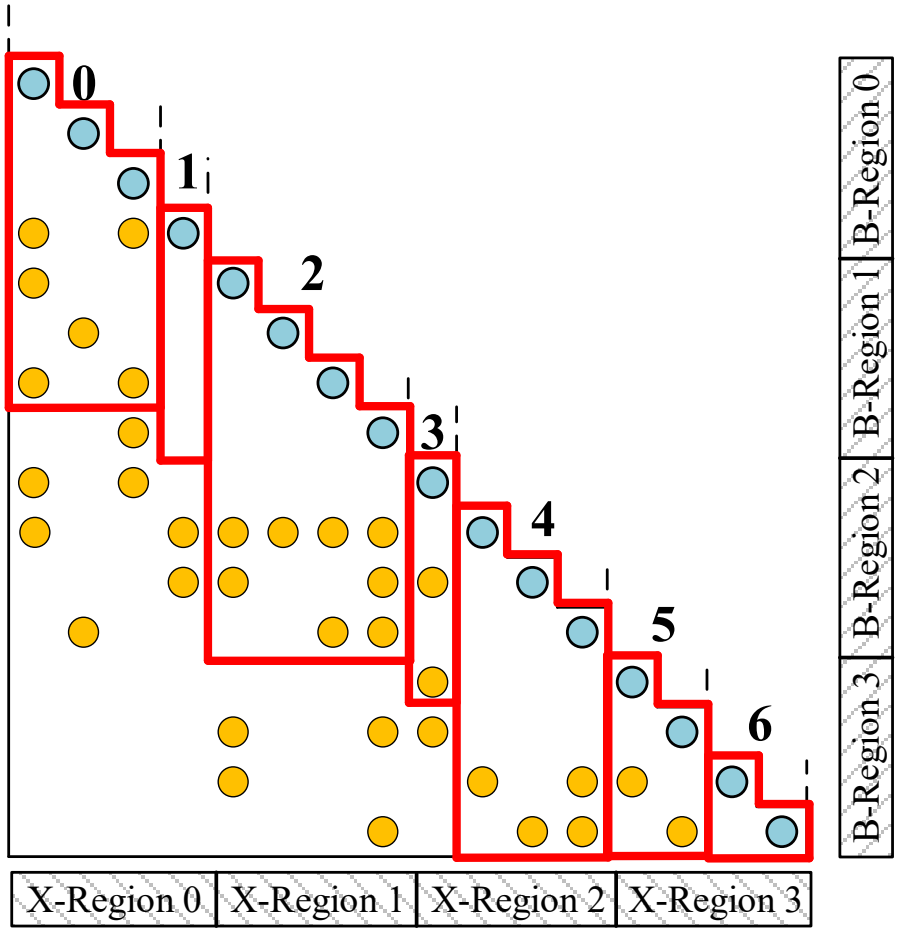
Sparse Level Tile (SLT) Layout: Step 2



Step 2:

- Separate the original levels into multiple levels if crossing multiple X-Regions.

Sparse Level Tile (SLT) Layout: Step 3



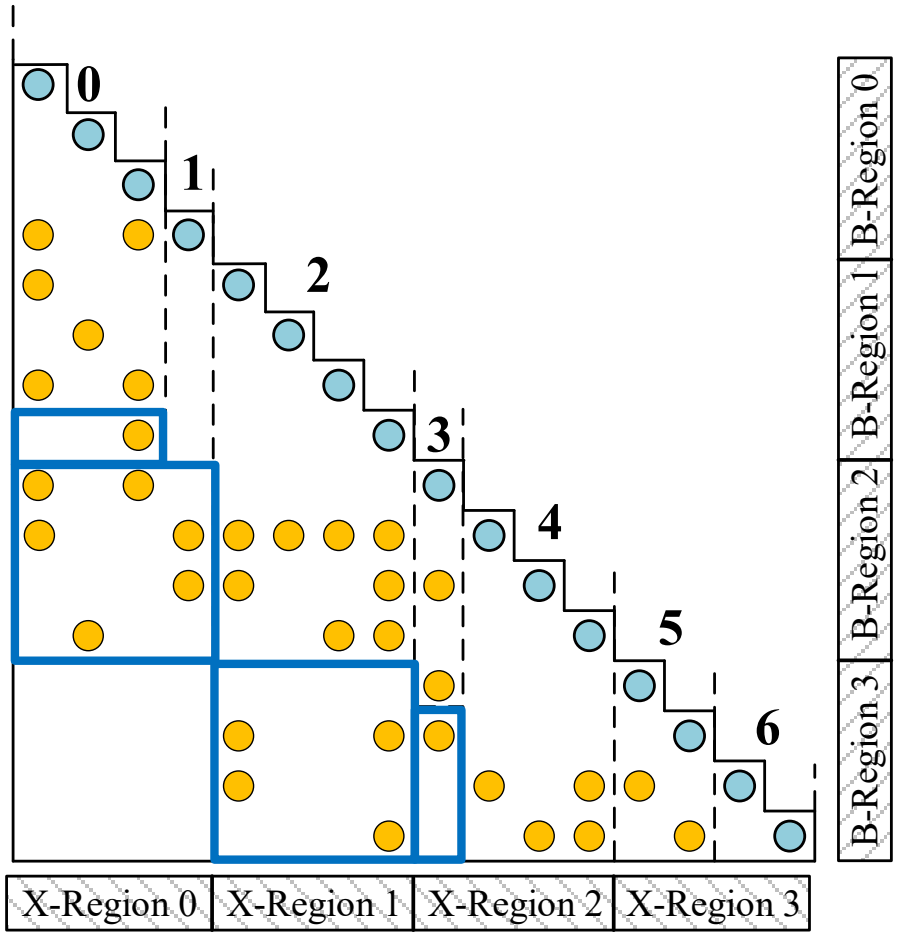
Step 3:

- 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.

Sparse Level Tile (SLT) Layout: Step 4



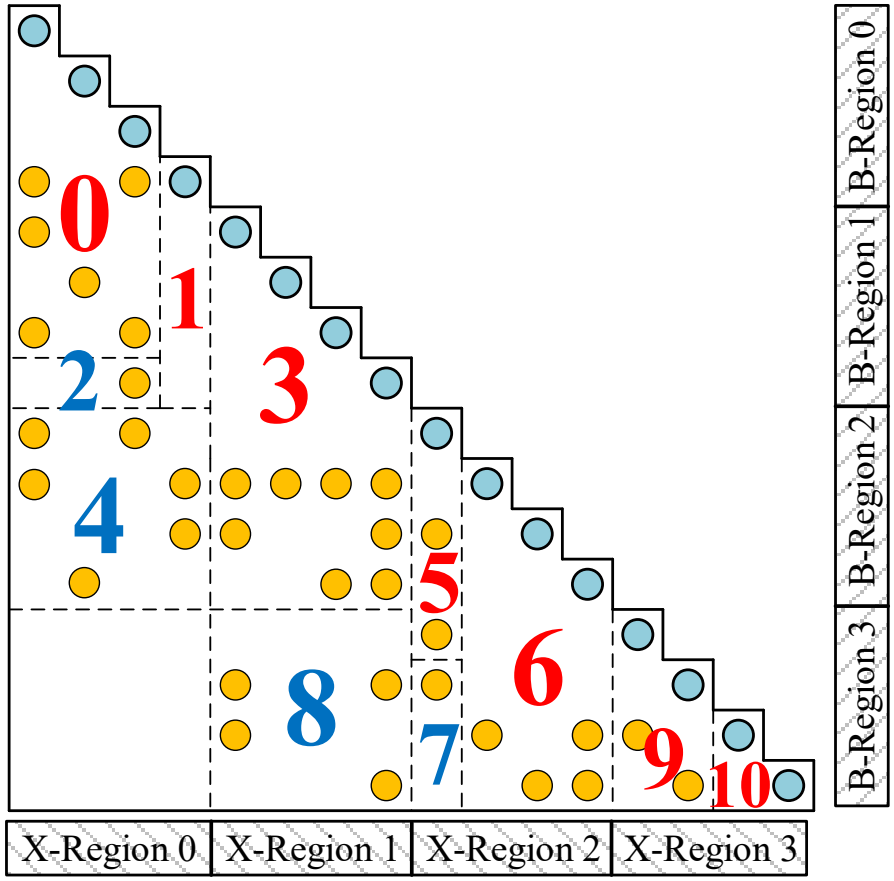
Step 4:

- 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



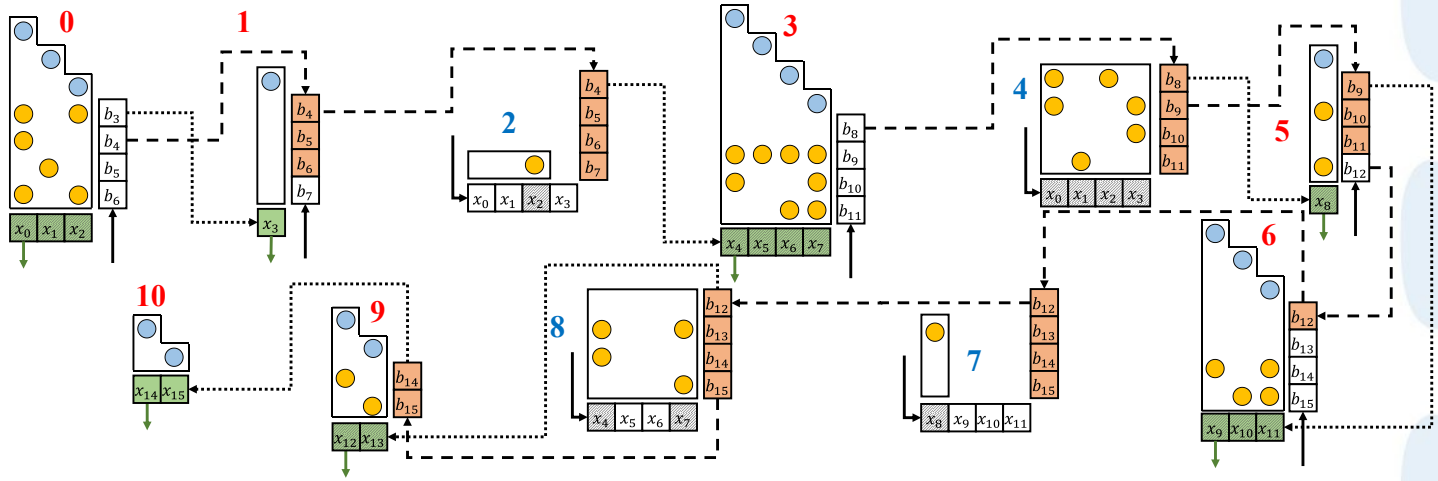
Step 5:

- 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

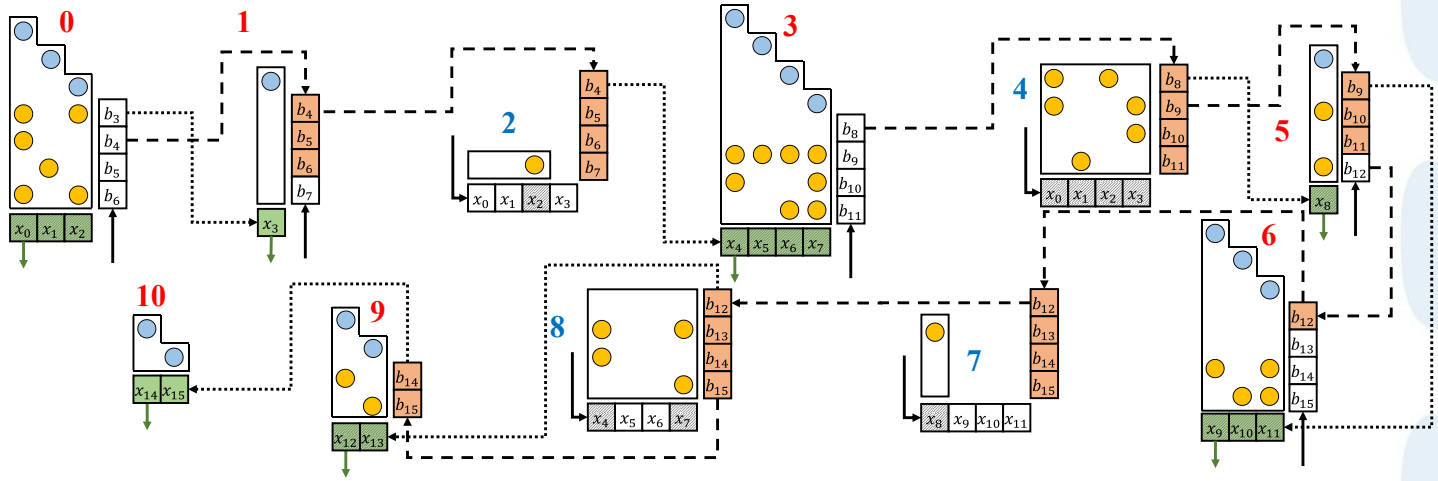
The SpTRSV process based on SLT layout



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

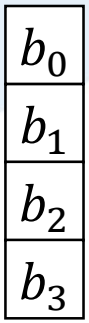
1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[REGION_SIZE] // cache for **x**
3: CACHE cb[REGION_SIZE] // cache for **b**
4: Replenish cb from **b** for REGION_SIZE
5: **for** $t = 0 \rightarrow tiles - 1$ **do**
6: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
7: **if** 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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**

The SpTRSV process based on SLT layout

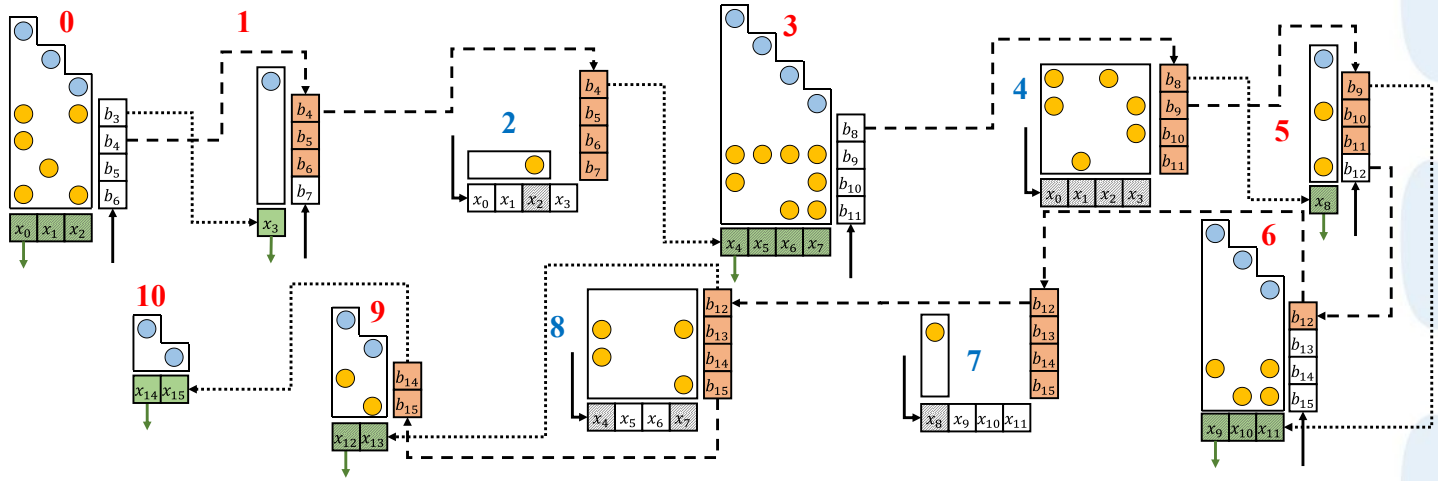


Input: l, li, lj, $tiles$, $sizes$, idx, **b**
Output: **x**

1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[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: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**

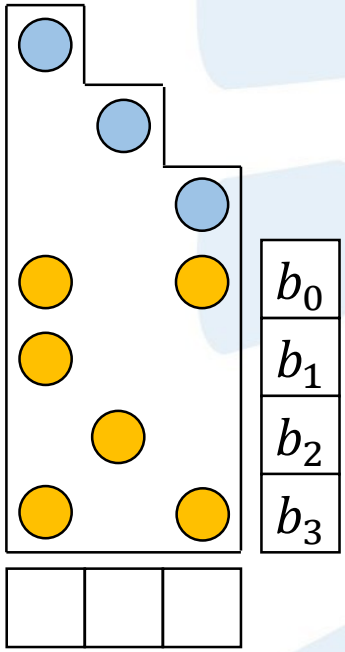


The SpTRSV process based on SLT layout

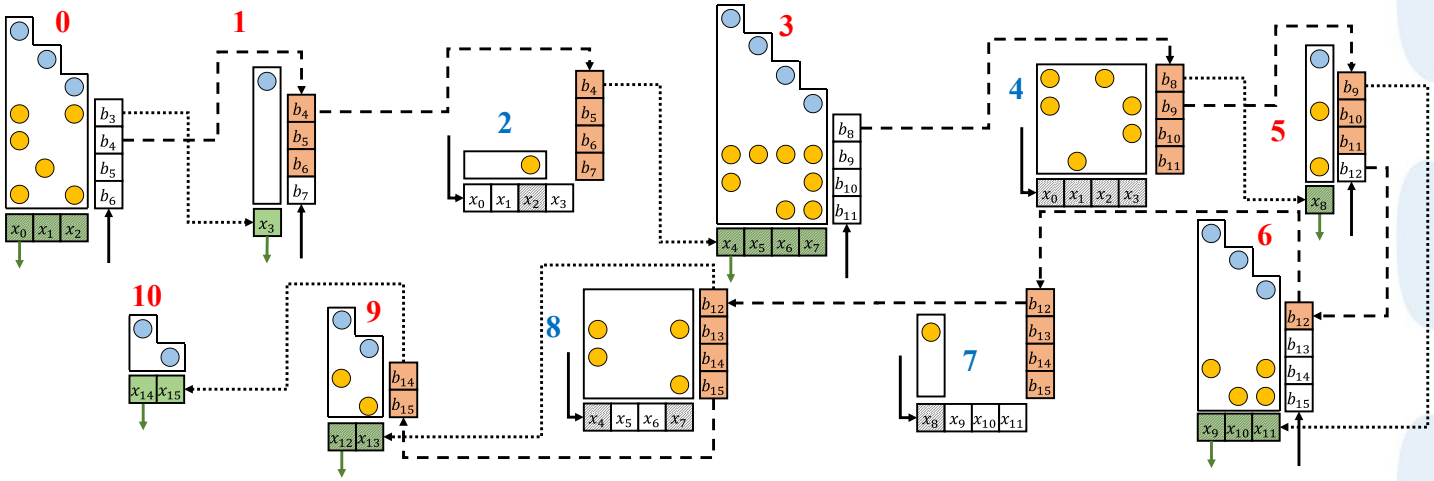


Input: l, li, lj, $tiles$, $sizes$, idx, **b**
Output: **x**

1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[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: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**

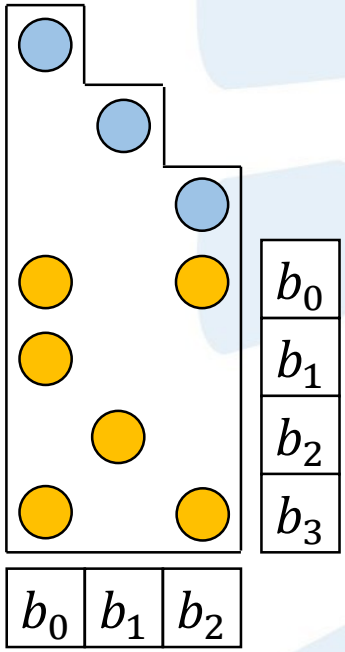


The SpTRSV process based on SLT layout

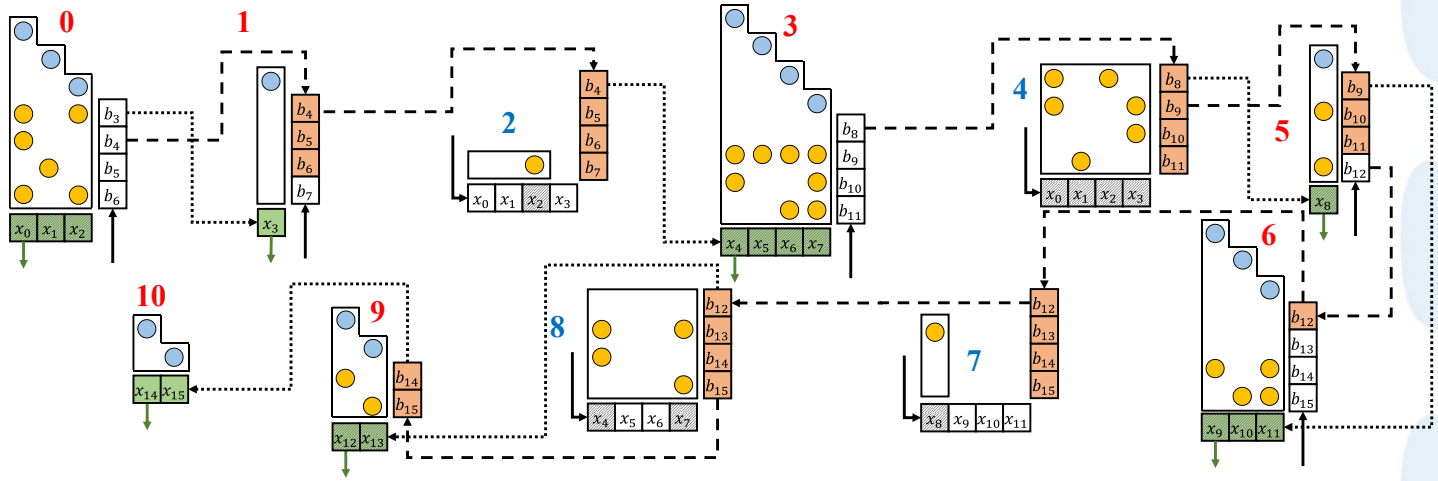


Input: l, li, lj, $tiles$, $sizes$, idx, **b**
Output: **x**

1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[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: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**

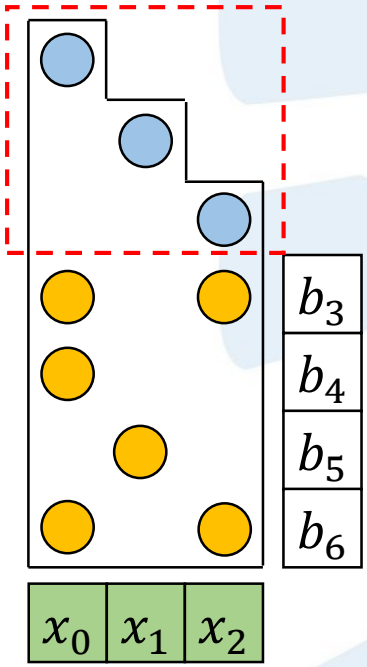


The SpTRSV process based on SLT layout

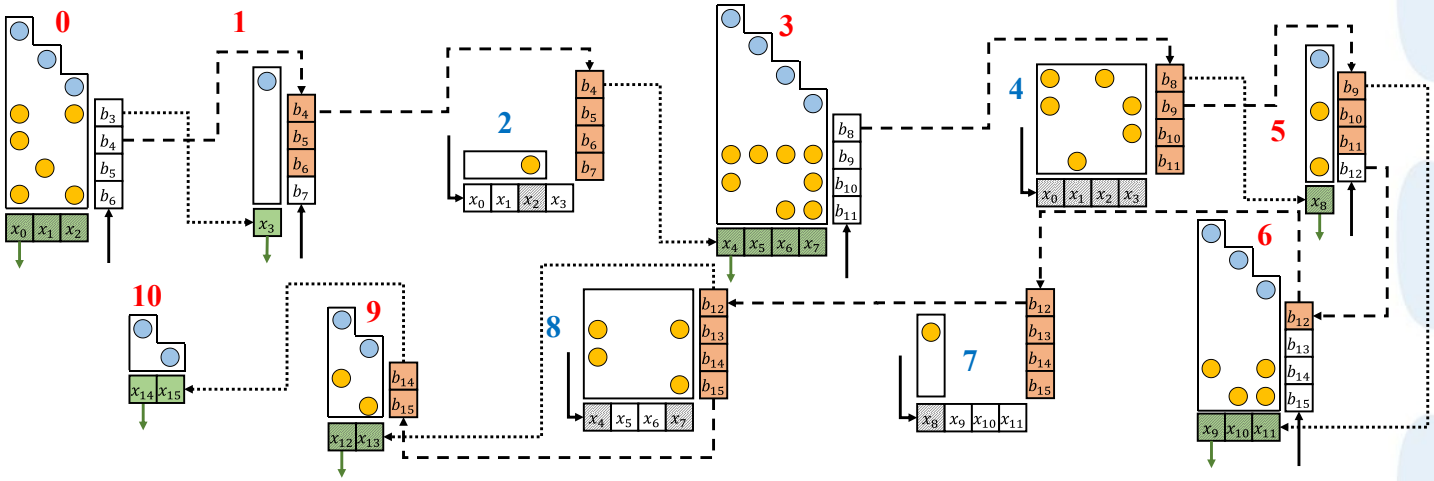


Input: l, li, lj, $tiles$, $sizes$, idx, **b**
Output: **x**

1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[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: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**

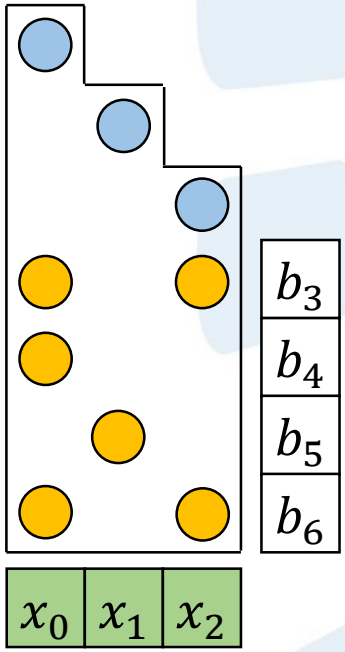


The SpTRSV process based on SLT layout

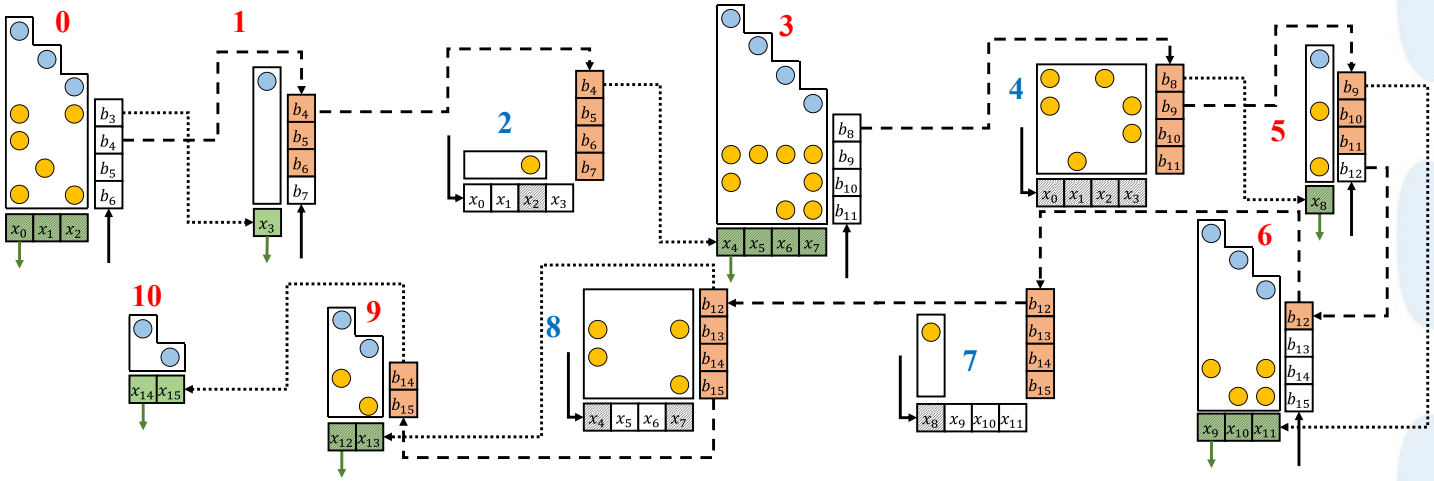


Input: l, li, lj, $tiles$, $sizes$, idx, **b**
Output: **x**

```
1:  function SpTRSV(x, b, l, li, lj, tiles, sizes, idx)
2:      CACHE cx[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_ij x_j, Tile_t
    means the submatrix consist of the nonzeros in Tile t
16:     end for
17: end function
```

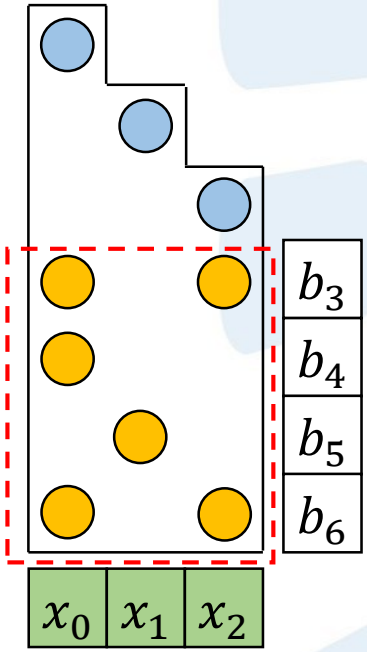


The SpTRSV process based on SLT layout



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

1: **function** SpTRSV(x, b, l, li, lj, $tiles$, $sizes$, idx)
2: CACHE cx[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: $nodnz \leftarrow idx[t+1] - idx[t] - sizes[t]$
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 REGION_SIZE
14: **end if**
15: $cb = cb - Tile_t \times cx$ for $nodnz$ //$b_i = b_i - l_{ij}x_j$, $Tile_t$ means the submatrix consist of the nonzeros in Tile t
16: **end for**
17: **end function**



Sparse Level Tile (SLT) Layout

- Manual Cache System
- Direct Memory Access (DMA)

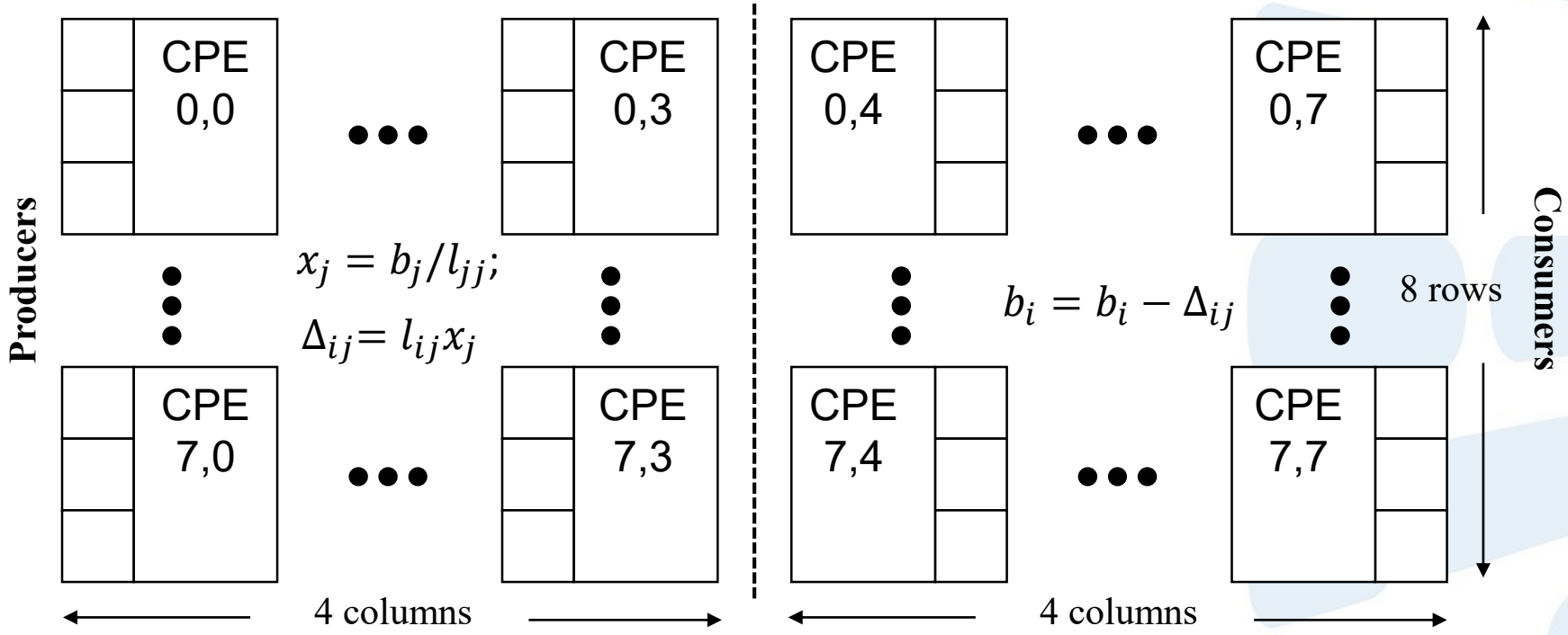
- Sparse Level Tile layout
  - Make sure all the computation is cache-hit;
  - Replace fine-grained, random and unprefetchable memory access with course-grained, predictable and prefetchable memory access;

Producer-Consumer pairing method

- Register Communication

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

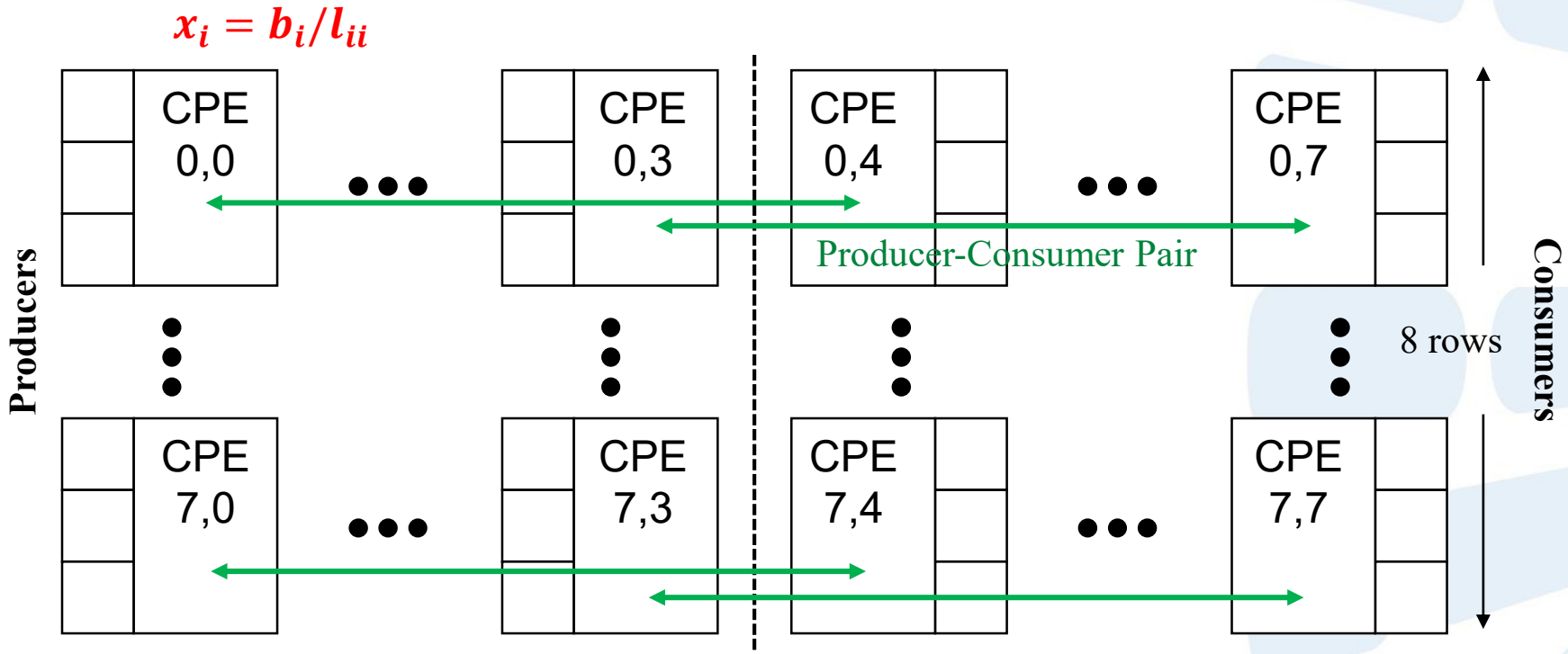
Producer-Consumer pairing method



$$x_j = b_j/l_{jj};$$
$$b_i = b_i - l_{ij}x_j$$

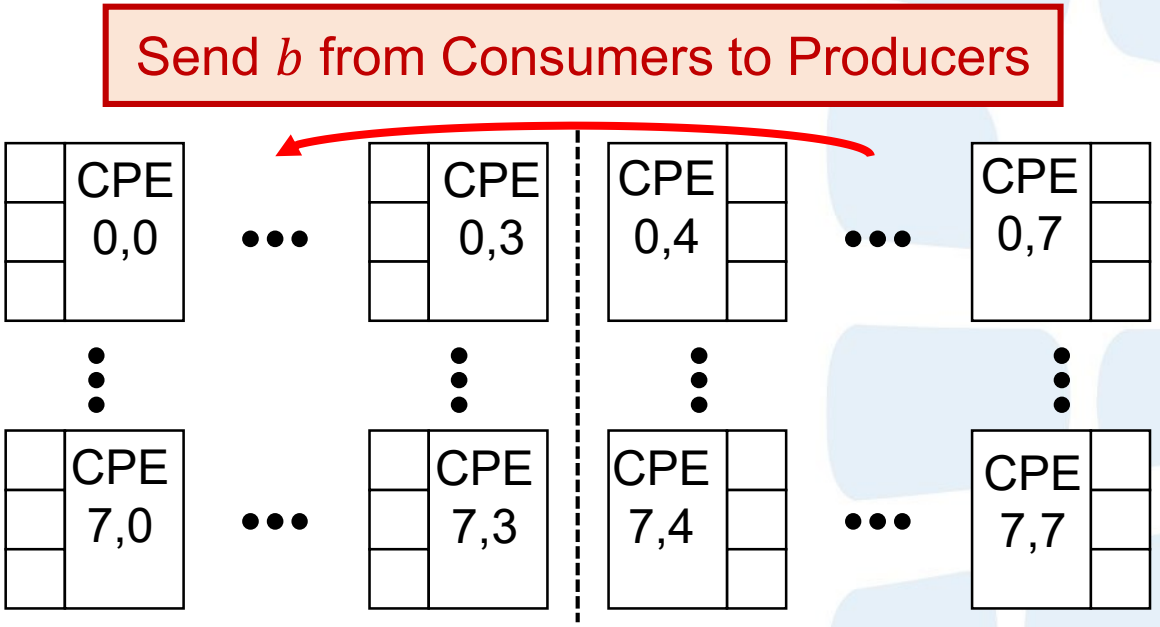
$$x_j = b_j/l_{jj};$$
$$\Delta_{ij} = l_{ij}x_j$$
$$b_i = b_i - \Delta_{ij}$$

Producer-Consumer pairing method



Producer-Consumer pairing method

$$x_j = b_j / l_{jj};$$

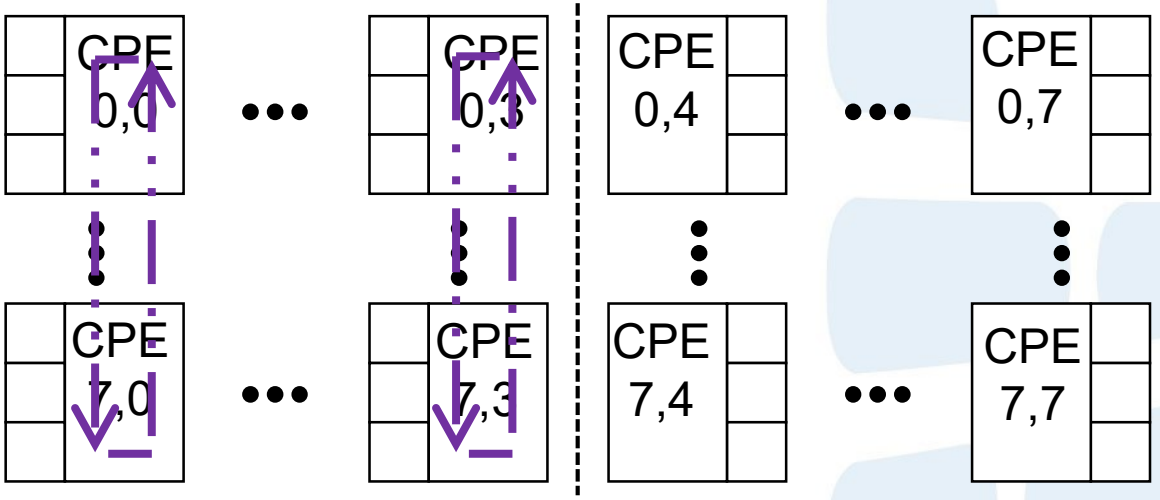


1. Acyclic communication
2. Pre-known communication size

Producer-Consumer pairing method

Share x across the same column

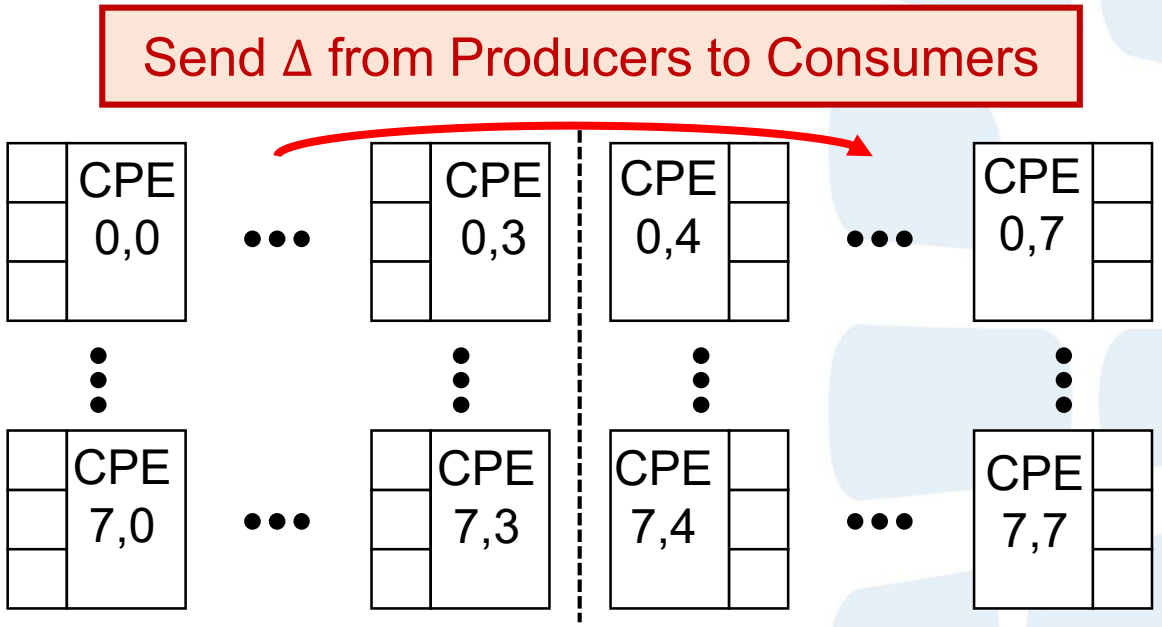
$$\Delta_{ij} = l_{ij} x_j$$



1. Acyclic communication
2. Pre-known communication size

Producer-Consumer pairing method

$$b_i = b_i - \Delta_{ij}$$



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

Testbeds
A single CG of SW26010 @ **34** GB/s
A single chip of NVIDIA K80 @ **240** GB/s
An Intel Xeon Phi 7120 KNC @ **352** GB/s

- 3 platforms

Experimental Setup

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

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

[1] Liu W, et al. A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves[C] European Conference on Parallel Processing. Springer International Publishing, 2016: 617-630.

[2] Park J, et al. Sparsifying synchronization for high-performance shared-memory sparse triangular solver[C] International Supercomputing Conference. Springer, Cham, 2014: 124-140.

Experimental Setup

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

Group	Parallelism		#Matrices
	Range	Average	
A	$[2^0, 2^5)$	15.97	249
B	$[2^5, 2^{10})$	287.59	1015
C	$[2^{10}, 2^{15})$	7064.68	634
D	$[2^{15}, 2^{20})$	358216.55	159
Total	$[2^0, 2^{20})$	30010.37	2057

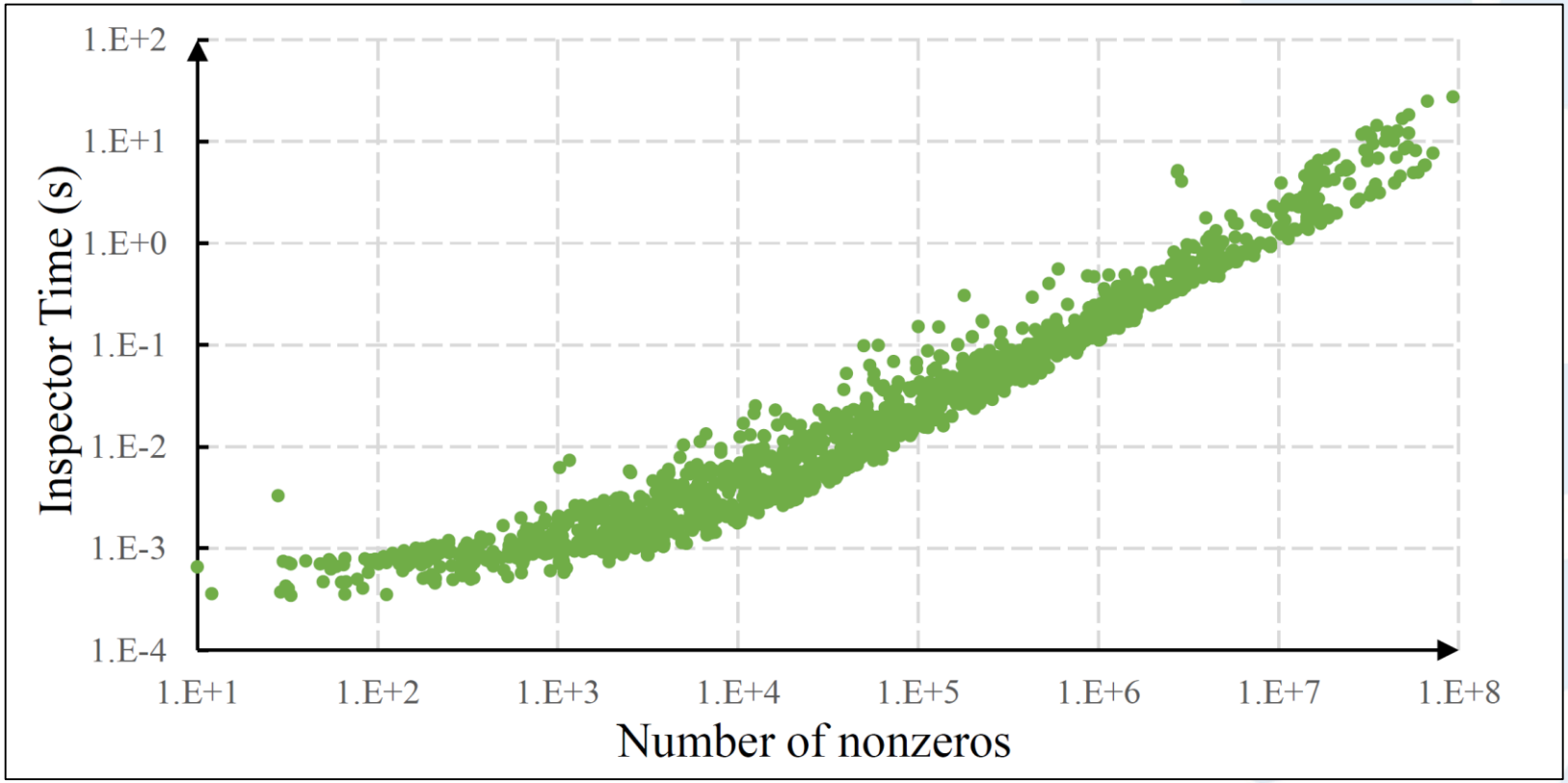
Parallelism = #nonzeros/#levels

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

[1] Liu W, et al. A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves[C] European Conference on Parallel Processing. Springer International Publishing, 2016: 617-630.

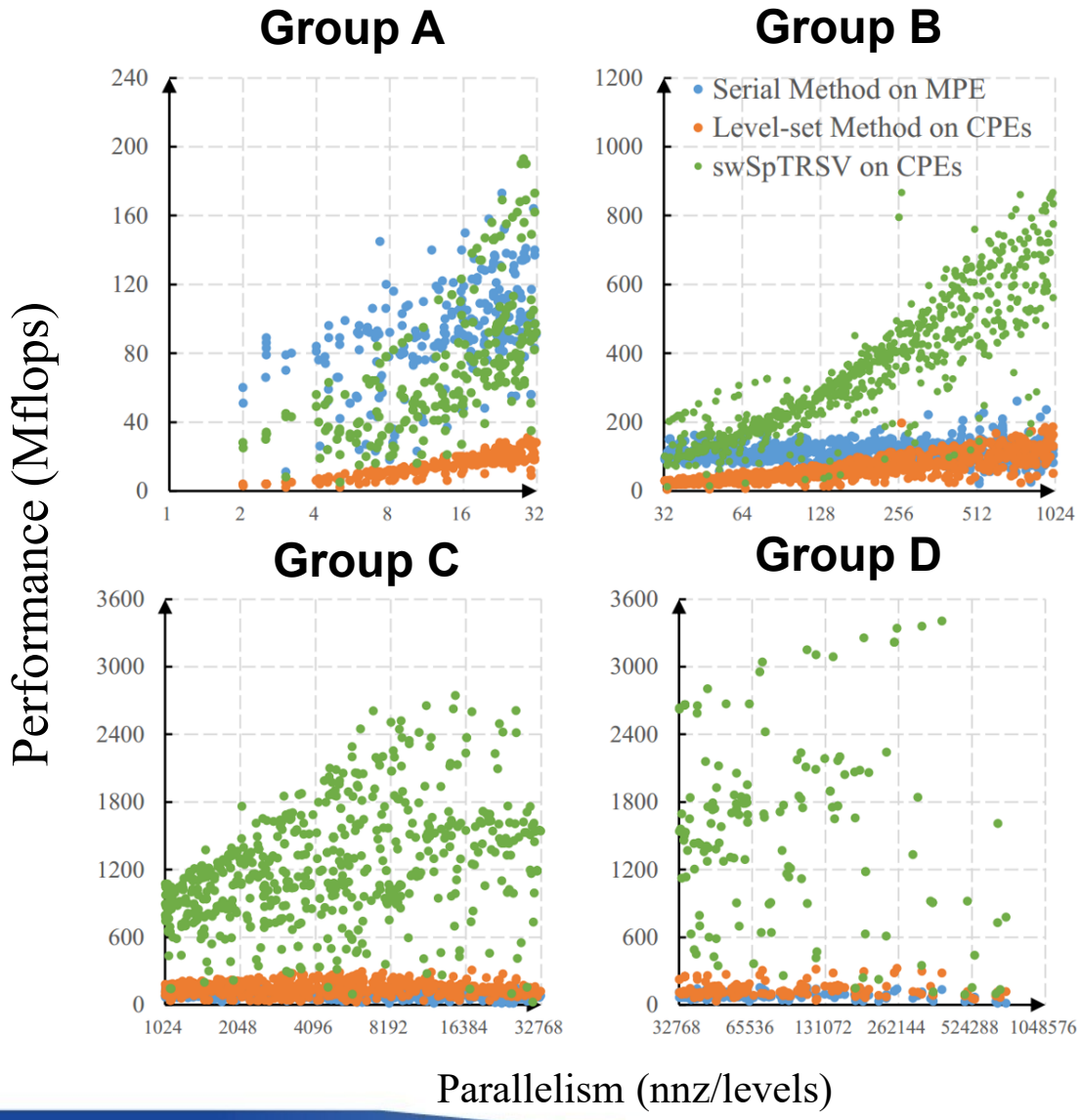
[2] Park J, et al. Sparsifying synchronization for high-performance shared-memory sparse triangular solver[C] International Supercomputing Conference. Springer, Cham, 2014: 124-140.

Pre-processing cost



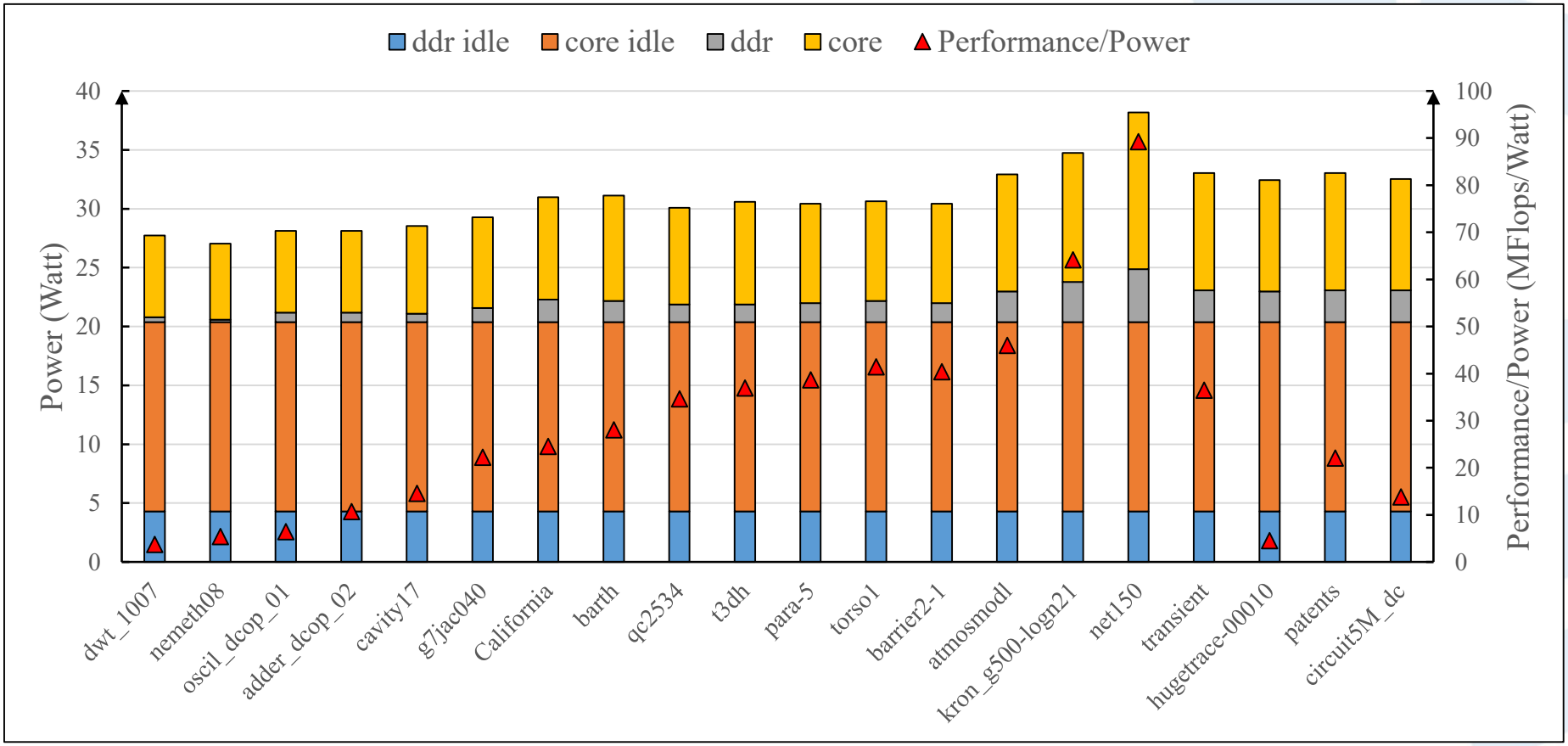
The cheapest cost is **0.3ms**, and the most expensive cost is **28.2s**. The harmonic average is **3.3ms**.

Methods on Sunway Processor



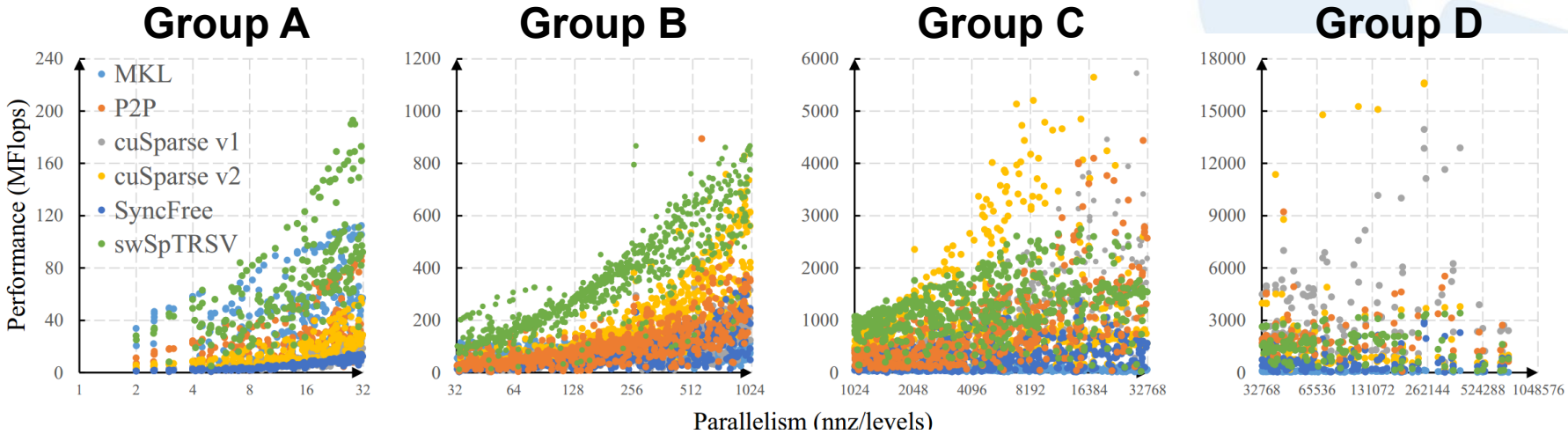
1. Best performance:
   3406 MFlops
2. Compared with sequential method:
   - Average speedup: **7.8**;
   - Best speedup: **117.3**;
3. Compared with level-set method:
   - Average speedup: **6.9**;
   - Best speedup: **38.5**;

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



	Group A	Group B	Group C	Group D	Sum
MKL	24	0	0	0	24
P2P	0	3	49	25	77
cuSparse v1	0	0	66	**107**	173
cuSparse v2	0	10	143	6	159
SyncFree	0	0	0	0	0
swSpTRSV	**225**	**1002**	**376**	21	**1624**
Total	249	1015	634	159	2057

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

- 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

Conclusion

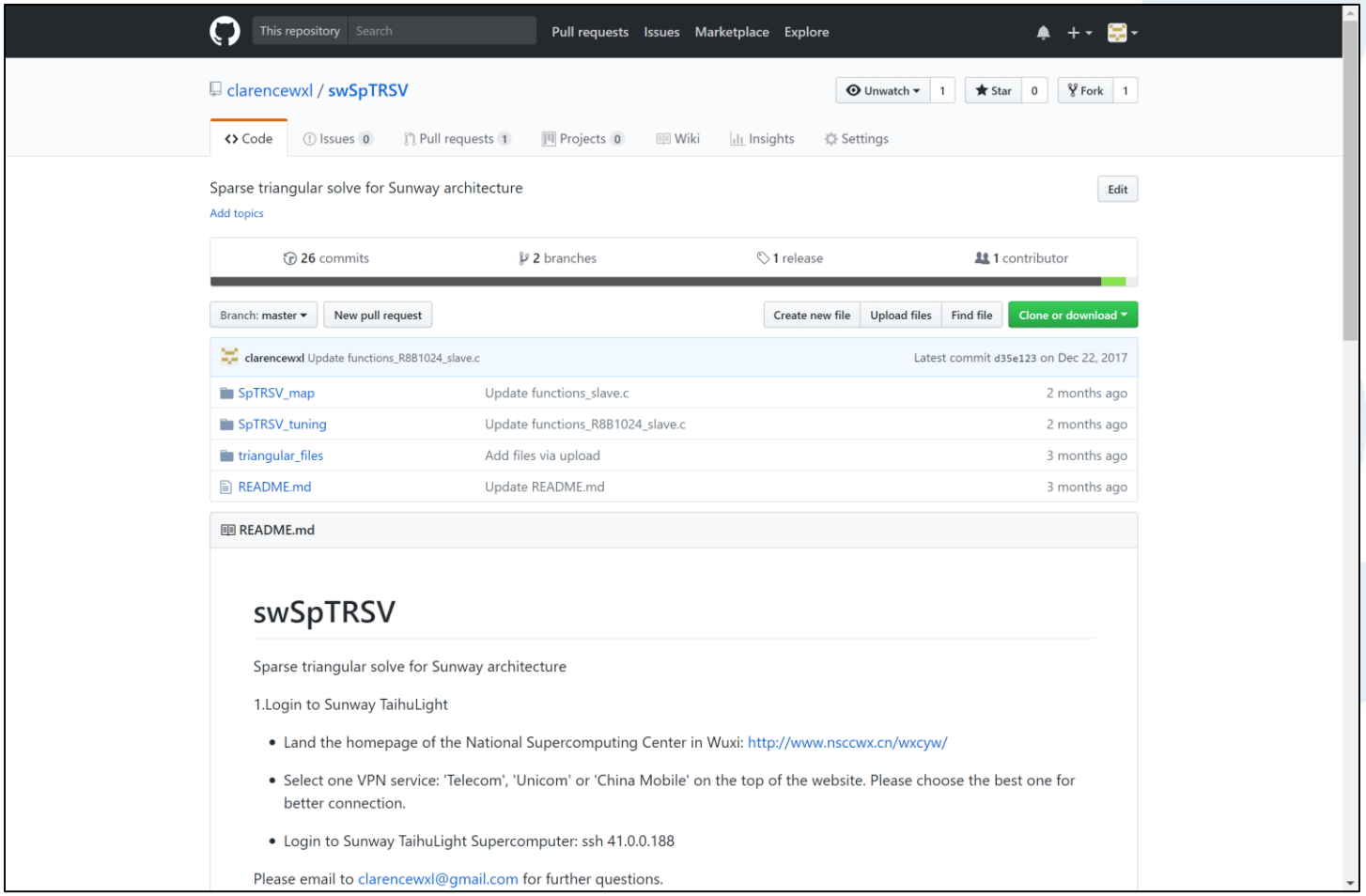
Method:

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

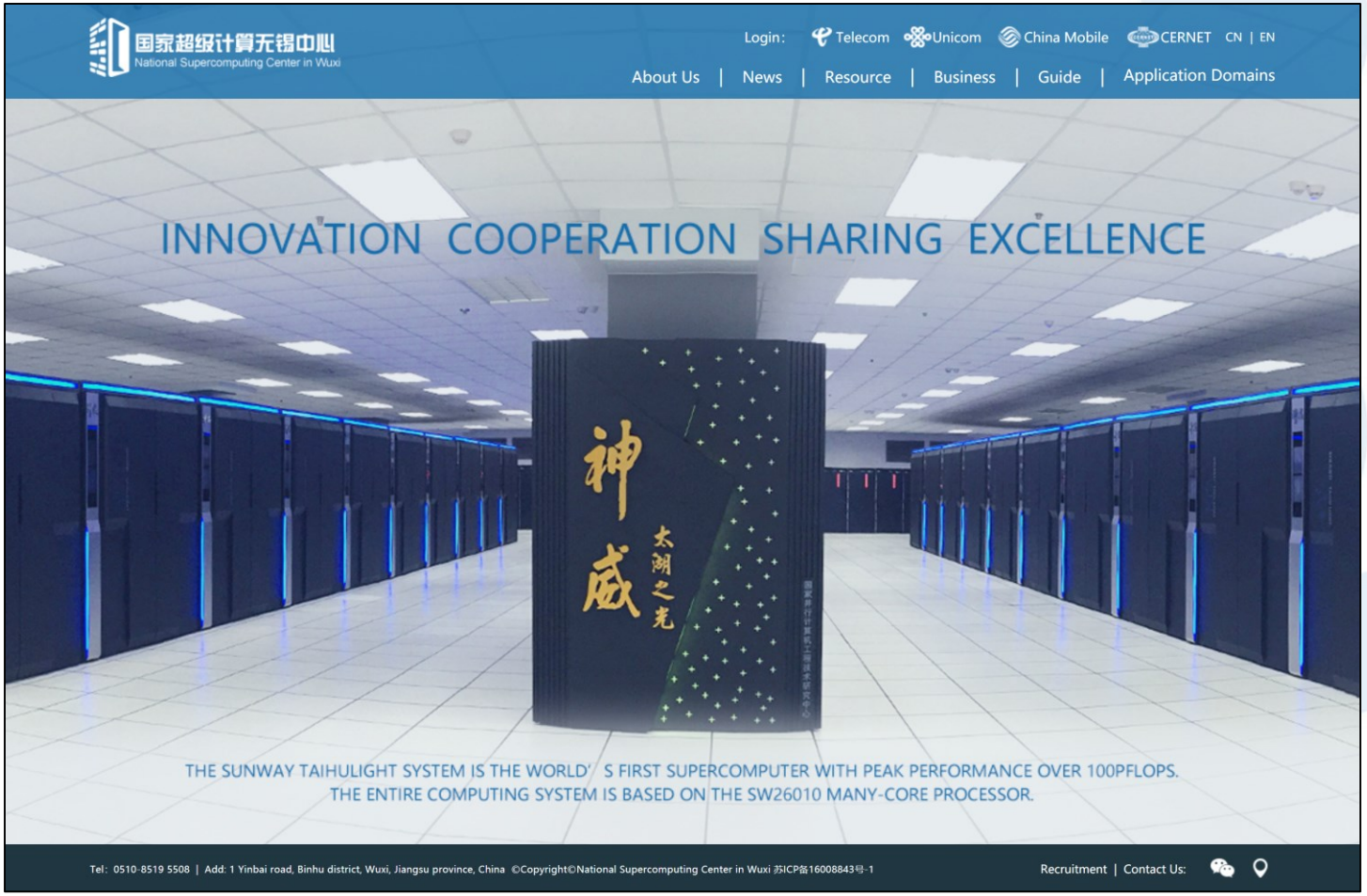
Performance:

- 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



Login: http://www.nsccwx.cn/wxcyw/



Thanks, Q&A

Welcome to Wuxi !
Welcome to TaihuLight !