49th International Conference on Parallel Processing - ICPP

CapelliniSpTRSV: A Thread-Level Synchronization-Free Sparse Triangular Solve on GPUs

Jiya Su◇ ‡, Feng Zhang◇, Weifeng Liu★, Bingsheng He+,

Ruofan Wu◇, Xiaoyong Du◇, Rujia Wang‡

◇Renmin University of China
★ China University of Petroleum
+National University of Singapore
‡ Illinois Institute of Technology

Outline

1. Background
2. Motivation
3. Challenges
4. CapelliniSpTRSV
5. Evaluation
6. Source Code at Github
7. Conclusion

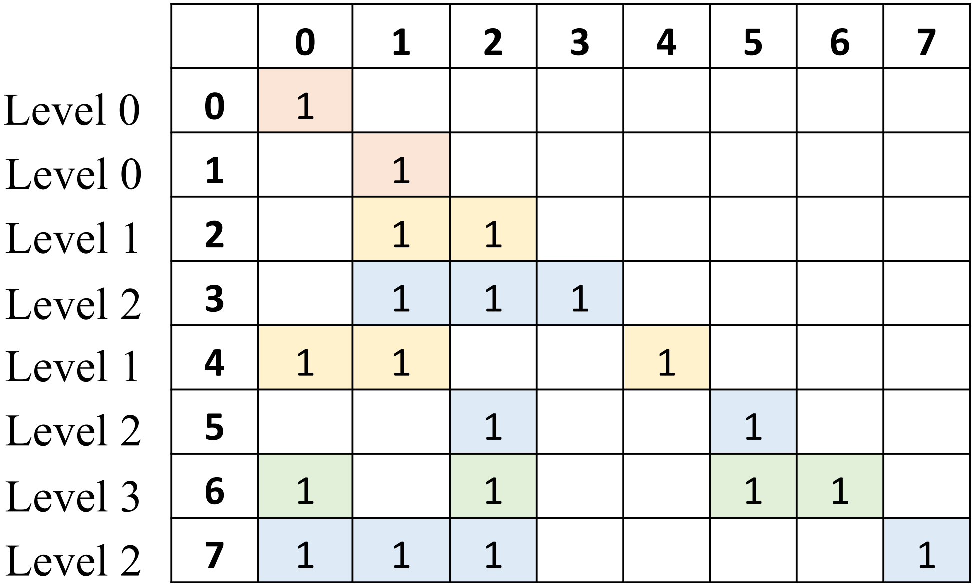
Outline

1. **Background**
2. Motivation
3. Challenges
4. CapelliniSpTRSV
5. Evaluation
6. Source Code at Github
7. Conclusion

1. Background

Sparse Matrix
in CSR format

Lower Triangular Matrix *L*



		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

(a) Matrix *L*.

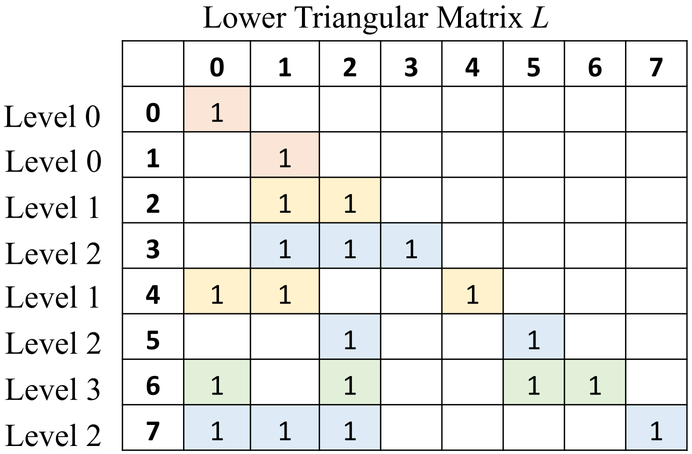
csrRowPtr = (0, 1, 2, 4, 7, 10, 12, 16, 20)
csrColIdx = (0, 1, 1, 2, 1, 2, 3, 0, 1, 4, 2, 5, 0, 2, 5, 6, 0, 1, 2, 7)
csrVal = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

(b) CSR representation.

1. Background

Sparse Triangular Solve

Example: **Lx = b**



Lower Triangular Matrix *L*

		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

Matrix *L*

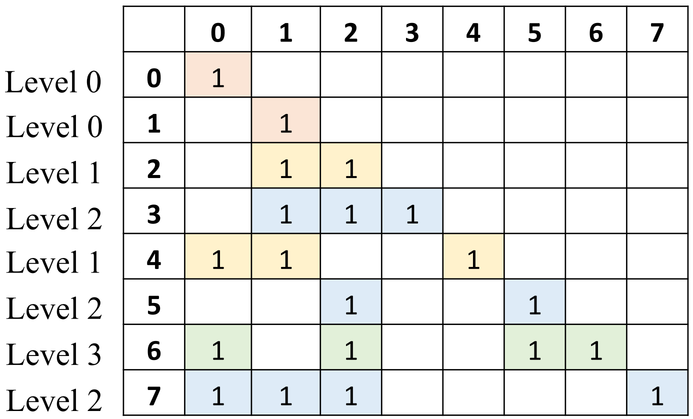
$$\underbrace{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{pmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}_{b}$$

1. Background

Sparse Triangular Solve

Example: **Lx = b**

Lower Triangular Matrix *L*



		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

Matrix *L*

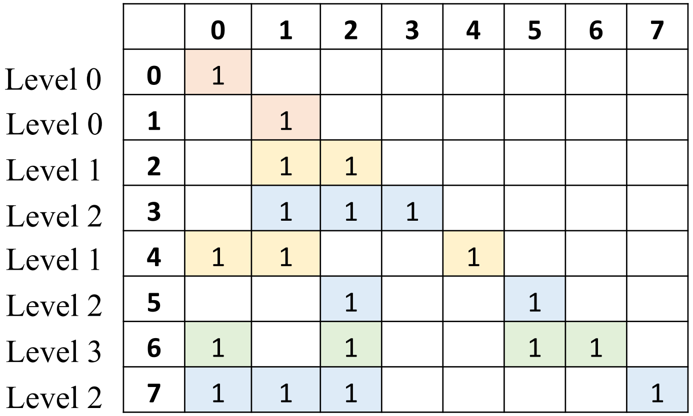
$$\underset{L}{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}} \times \underset{x}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \underset{b}{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}$$

1. Background

Concepts :

· Component

Lower Triangular Matrix *L*



		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

Matrix *L*

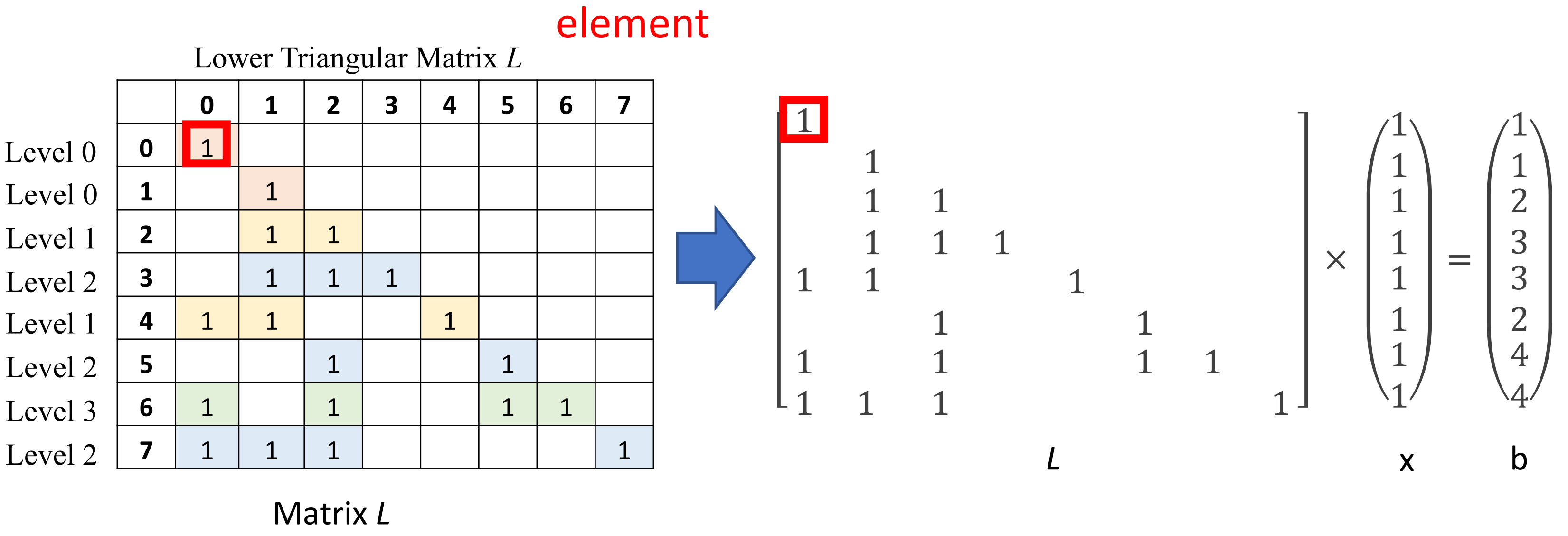
component

$$\underset{L}{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}} \times \underset{\mathrm{x}}{\begin{pmatrix} \boxed{1} \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \underset{\mathrm{b}}{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}$$

1. Background

Concepts :

- Component
- Element



Lower Triangular Matrix *L*

		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

Matrix *L*

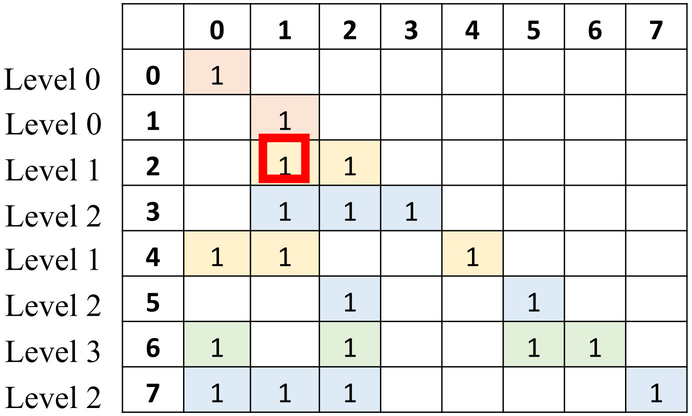
$$\underset{L}{\begin{bmatrix} \boxed{1} & & & & & & & \\ & 1 & & & & & & \\ & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}} \times \underset{x}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \underset{b}{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}$$

1. Background

Concepts :

- Component
- Element
- Dependency

Lower Triangular Matrix *L*



		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

Matrix *L*

dependency

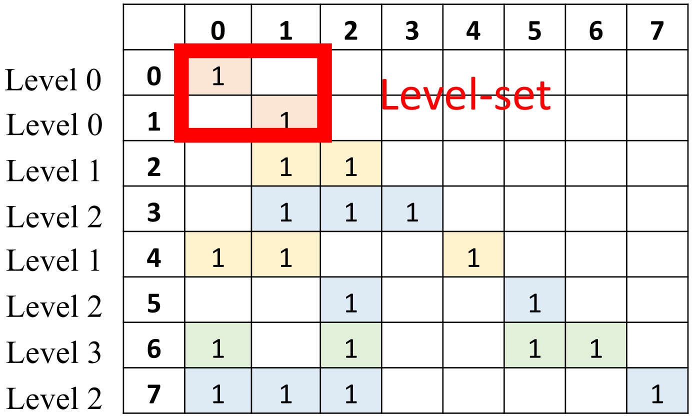
$$\underbrace{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & \boxed{1} & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{pmatrix} 1 \\ 1 \\ \boxed{\begin{matrix} 1 \\ 1 \end{matrix}} \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\mathrm{x}} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}_{\mathrm{b}}$$

1. Background

Concepts :

· Component · Element · Dependency · **Level**

Lower Triangular Matrix *L*



		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

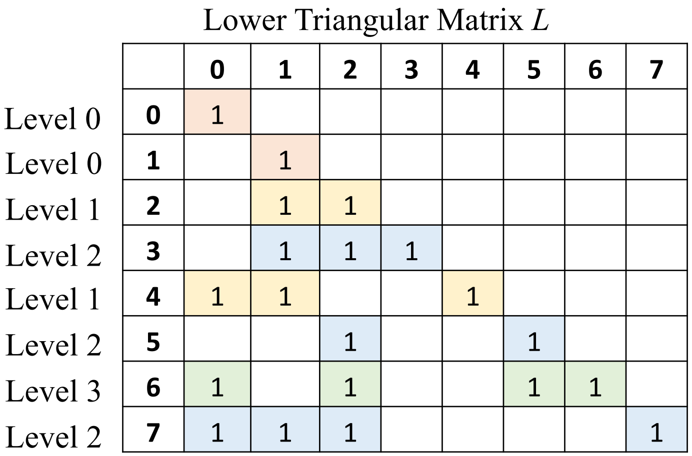
Matrix *L*

$$\underbrace{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ 1 & 1 & & & 1 & & & \\ & & 1 & & & 1 & & \\ 1 & & 1 & & & 1 & 1 & \\ 1 & 1 & 1 & & & & & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 4 \\ 4 \end{pmatrix}}_{b}$$

1. Background

Level-set SpTRSV

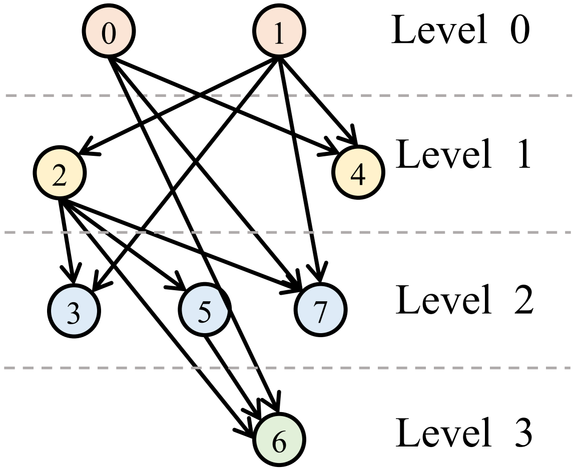
The level-set method has two phases: (1) grouping nodes (rows or columns) that can be consumed in parallel, and (2) solving nodes group by group with barriers between.



Lower Triangular Matrix *L*

		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

(a) Matrix *L*.

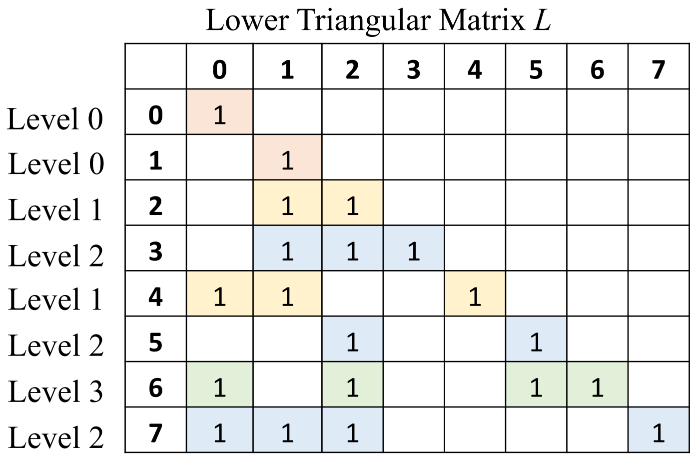


(b) Components *x* in the level-sets.

1. Background

Level-set SpTRSV

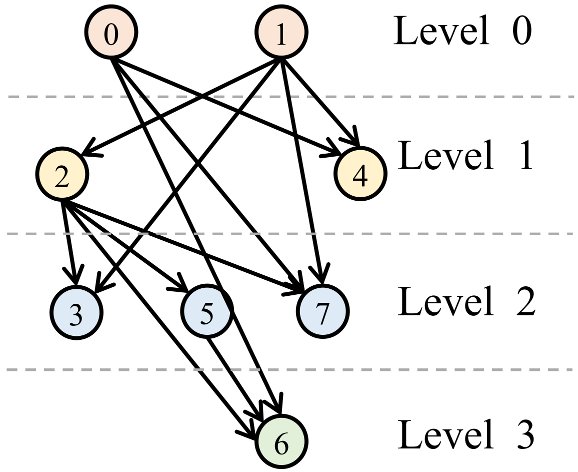
The level-set method has two phases: **(1) grouping nodes (rows or columns) that can be consumed in parallel**, and (2) solving nodes group by group with barriers between.



Lower Triangular Matrix *L*

		0	1	2	3	4	5	6	7
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

(a) Matrix *L*.

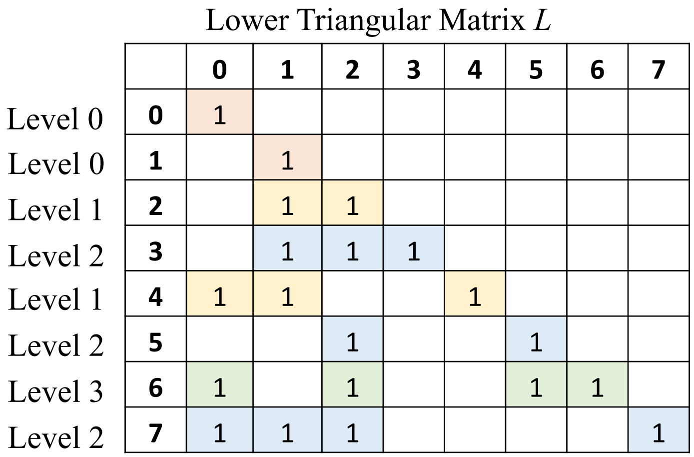


(b) Components *x* in the level-sets.

1. Background

Level-set SpTRSV

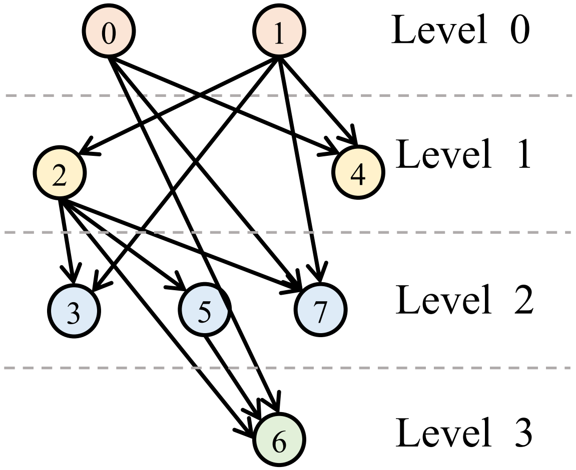
The level-set method has two phases: (1) grouping nodes (rows or columns) that can be consumed in parallel, and **(2) solving nodes group by group with barriers between**.



Lower Triangular Matrix *L*

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Level 0	**0**	1							
Level 0	**1**		1						
Level 1	**2**		1	1					
Level 2	**3**		1	1	1				
Level 1	**4**	1	1			1			
Level 2	**5**			1			1		
Level 3	**6**	1		1			1	1	
Level 2	**7**	1	1	1					1

(a) Matrix *L*.



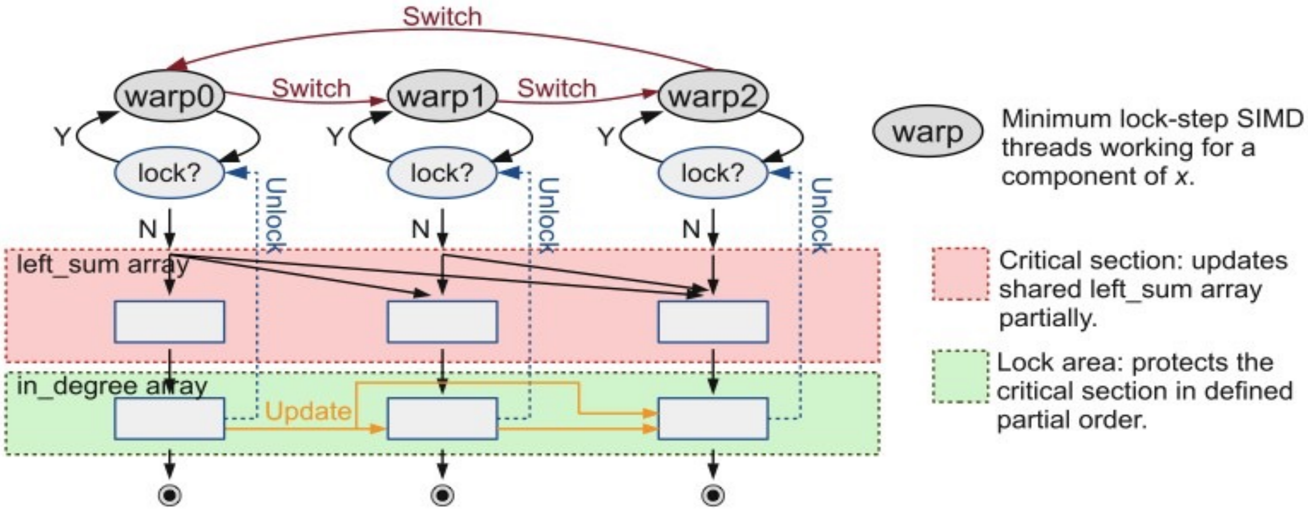
(b) Components *x* in the level-sets.

1. Background

Synchronization-Free SpTRSV (warp-level)

The algorithm computes components **x** in the original row order of the input matrix and uses one warp to compute one row.

It uses a new flag array ***in_degree*** to show whether the component **x** is solved, which avoids the synchronization and greatly reduces the processing time.



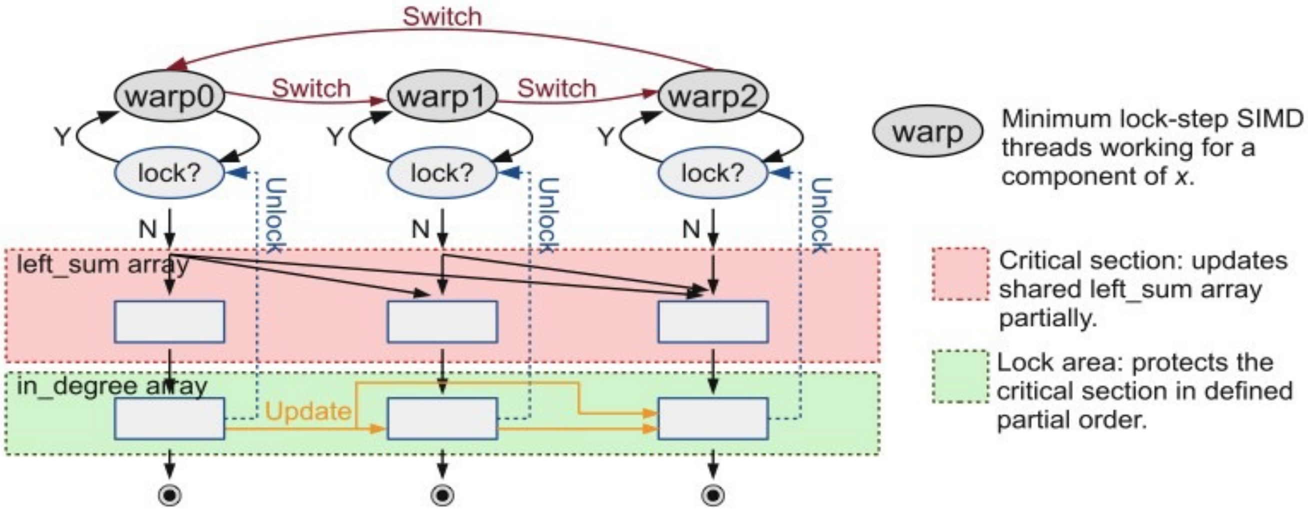
Liu W, Li A, Hogg J, et al. A synchronization-free algorithm for parallel sparse triangular solves[C]//European Conference on Parallel Processing. Springer, Cham, 2016: 617-630.

1. Background

Synchronization-Free SpTRSV (warp-level)

The algorithm computes components x in the original row order of the input matrix and uses one warp to compute one row.

It uses a new flag array ***in_degree*** to show whether the component **x** is solved, which avoids the synchronization and greatly reduces the processing time.



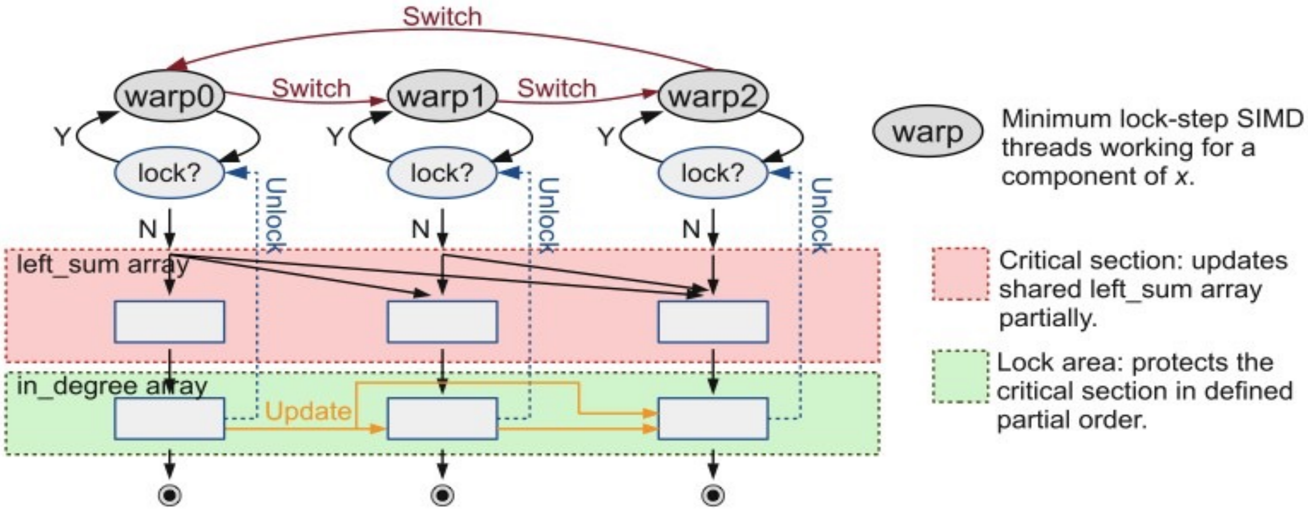
Liu W, Li A, Hogg J, et al. A synchronization-free algorithm for parallel sparse triangular solves[C]//European Conference on Parallel Processing. Springer, Cham, 2016: 617-630.

1. Background

Synchronization-Free SpTRSV (warp-level)

The algorithm computes components **x** in the original row order of the input matrix and uses one warp to compute one row.

It uses a new flag array *in_degree* to show whether the component x is solved, which avoids the synchronization and greatly reduces the processing time.



Liu W, Li A, Hogg J, et al. A synchronization-free algorithm for parallel sparse triangular solves[C]//European Conference on Parallel Processing. Springer, Cham, 2016: 617-630.

1. Background

Case study for preprocessing time and execution time of different SpTRSV algorithms

Algorithm	time (ms)	nlpkkt160	wiki-Talk	cant
Level-Set	preprocessing execution	310.07 28.07	31.09 12.89	4.81 28.79
cuSPARSE	preprocessing execution	16.24 37.98	1.99 11.88	0.28 7.69
Sync-Free ☺	preprocessing execution	8.07 27.73	0.42 10.02	0.28 5.02

2. Motivation

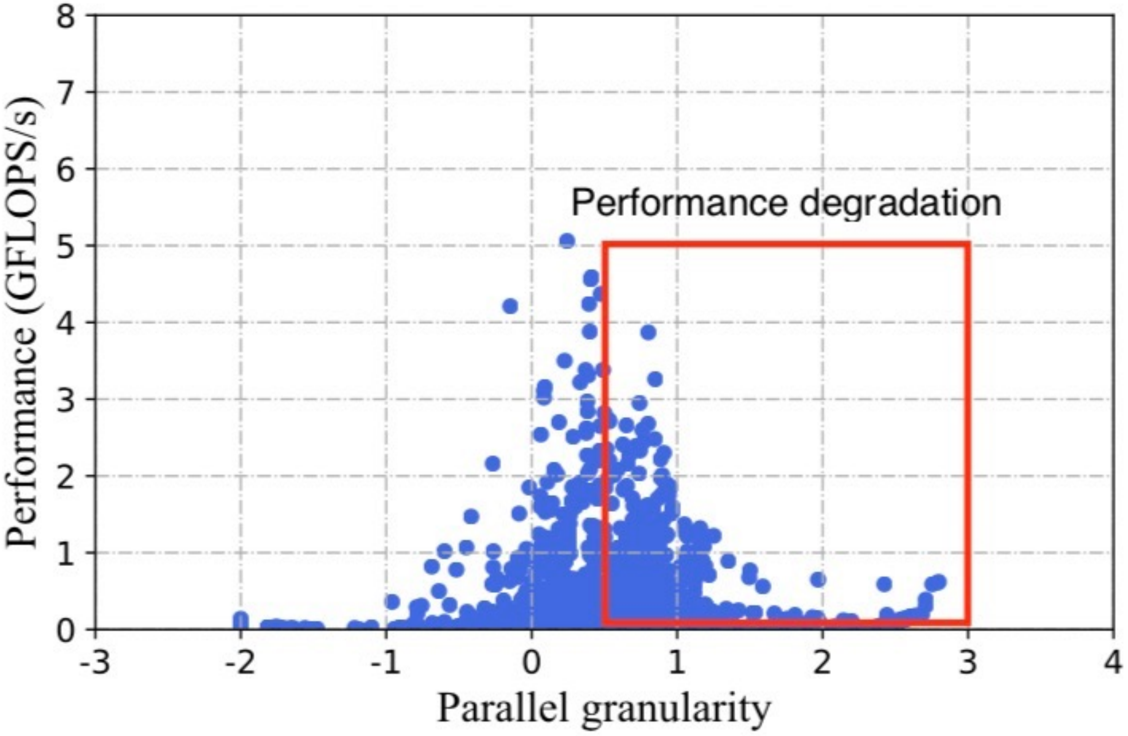
Performance trend of warp-level synchronization-free SpTRSV.

$$parallel\_granularity = log_{c_1}(\frac{log_{c_2}(n_{level})}{log_{c_3}(nnz_{row} + b_1)} + b_2)$$

2. Motivation

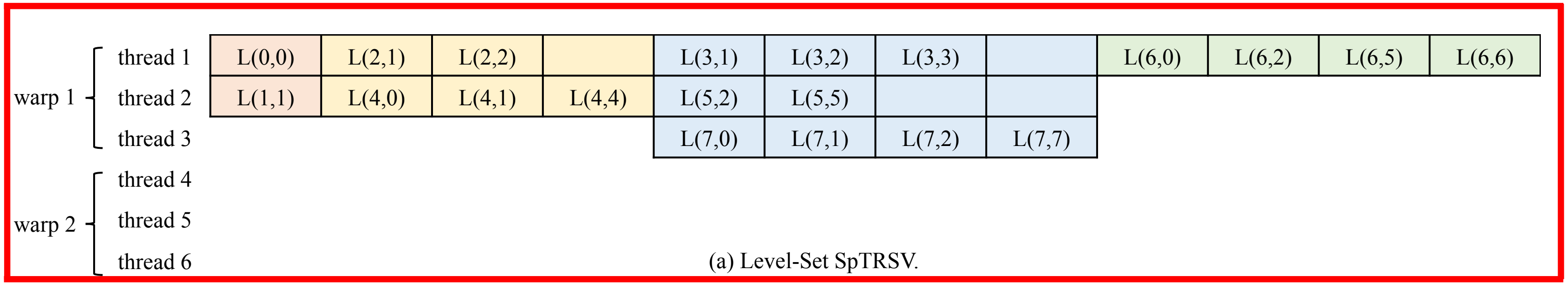
Performance trend of warp-level synchronization-free SpTRSV.

$$parallel\_granularity = log_{c_1}\left(\frac{log_{c_2}(n_{level})}{log_{c_3}(nnz_{row} + b_1)} + b_2\right)$$



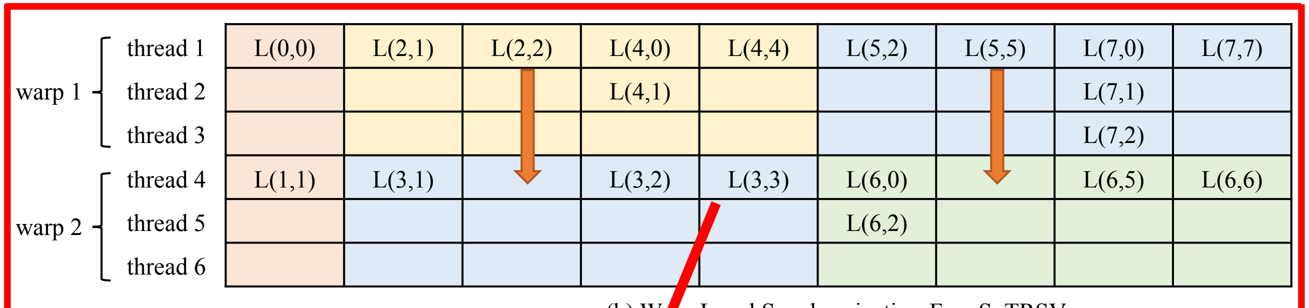
The performance declines after reaching the peak state.

2. Motivation



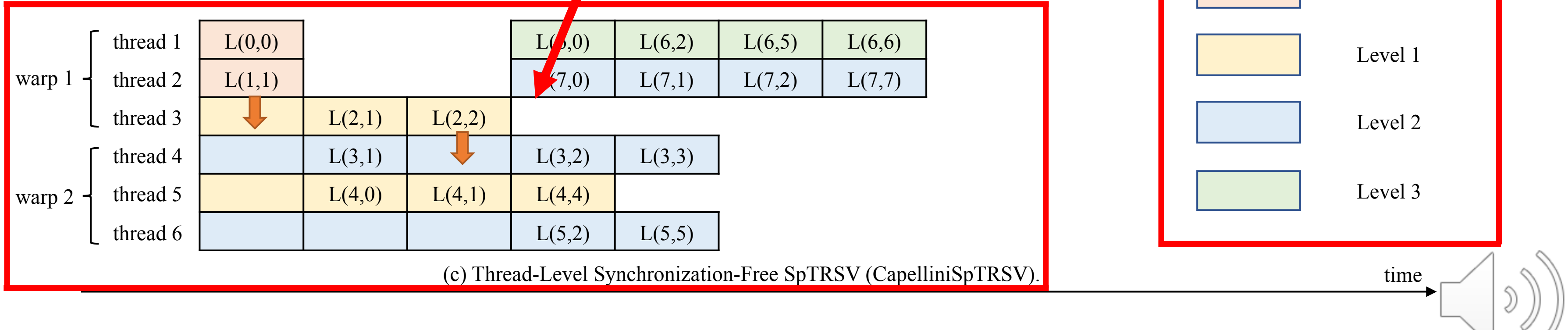
warp 1	thread 1	L(0,0)	L(2,1)	L(2,2)		L(3,1)	L(3,2)	L(3,3)		L(6,0)	L(6,2)	L(6,5)	L(6,6)
	thread 2	L(1,1)	L(4,0)	L(4,1)	L(4,4)	L(5,2)	L(5,5)						
	thread 3					L(7,0)	L(7,1)	L(7,2)	L(7,7)				
warp 2	thread 4												
	thread 5												
	thread 6												

(a) Level-Set SpTRSV.



warp 1	thread 1	L(0,0)	L(2,1)	L(2,2)	L(4,0)	L(4,4)	L(5,2)	L(5,5)	L(7,0)	L(7,7)
	thread 2				L(4,1)				L(7,1)	
	thread 3								L(7,2)	
warp 2	thread 4	L(1,1)	L(3,1)		L(3,2)	L(3,3)	L(6,0)		L(6,5)	L(6,6)
	thread 5						L(6,2)			
	thread 6									

(b) Warp-Level Synchronization-Free SpTRSV.



warp 1	thread 1	L(0,0)			L(6,0)	L(6,2)	L(6,5)	L(6,6)
	thread 2	L(1,1)			L(7,0)	L(7,1)	L(7,2)	L(7,7)
	thread 3		L(2,1)	L(2,2)				
warp 2	thread 4		L(3,1)		L(3,2)	L(3,3)		
	thread 5		L(4,0)	L(4,1)	L(4,4)			
	thread 6				L(5,2)	L(5,5)		

(c) Thread-Level Synchronization-Free SpTRSV (CapelliniSpTRSV).

➡ Data transmission

🟧 Level 0

🟨 Level 1

🟦 Level 2

🟩 Level 3

2. Motivation

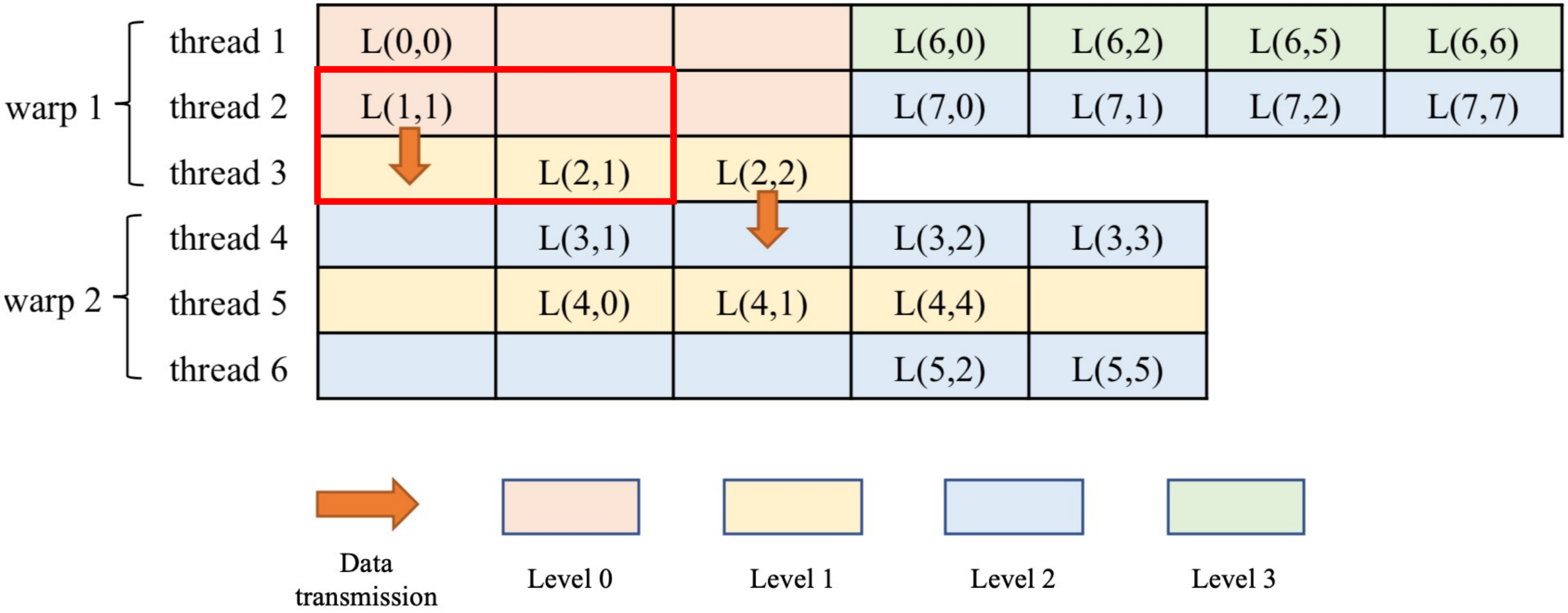
- Observation: Warp-level synchronization-free SpTRSV algorithm cannot fully utilize GPU resources when parallel granularity is large.
- Insight:

Capellini

fine-grained

3. Challenges

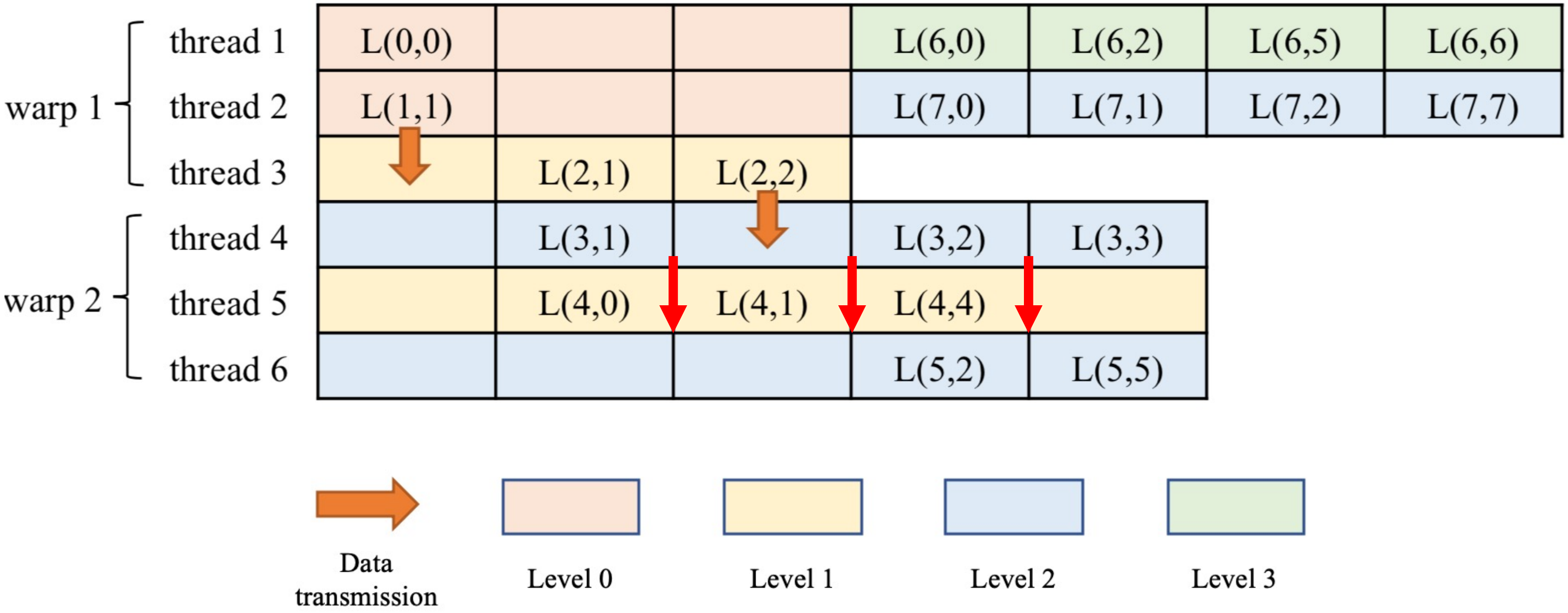
- Challenge 1: avoiding deadlocks
  - In thread-level design, the threads in one warp may have dependencies.



warp 1	thread 1	L(0,0)			L(6,0)	L(6,2)	L(6,5)	L(6,6)
	thread 2	L(1,1)			L(7,0)	L(7,1)	L(7,2)	L(7,7)
	thread 3		L(2,1)	L(2,2)				
warp 2	thread 4		L(3,1)		L(3,2)	L(3,3)		
	thread 5		L(4,0)	L(4,1)	L(4,4)			
	thread 6				L(5,2)	L(5,5)		

3. Challenges

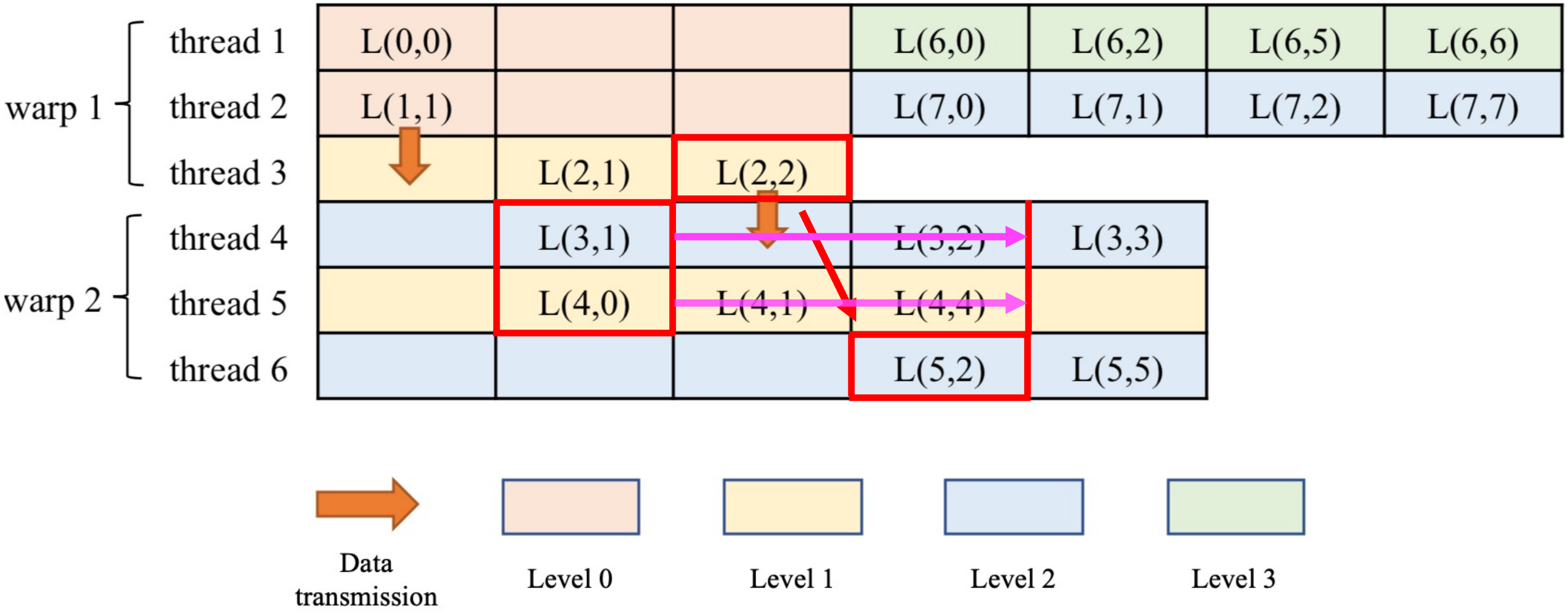
- Challenge 2: last element checking
  - We need to verify whether the processed element is on the diagonal, which causes time overhead.



warp 1	thread 1	L(0,0)			L(6,0)	L(6,2)	L(6,5)	L(6,6)
	thread 2	L(1,1)			L(7,0)	L(7,1)	L(7,2)	L(7,7)
	thread 3		L(2,1)	L(2,2)				
warp 2	thread 4		L(3,1)		L(3,2)	L(3,3)		
	thread 5		L(4,0)	L(4,1)	L(4,4)			
	thread 6				L(5,2)	L(5,5)		

3. Challenges

- Challenge 3: thread execution model
  - Although we use a thread to handle one component, the GPUs are still executed in the warp execution mode.



warp 1	thread 1	L(0,0)			L(6,0)	L(6,2)	L(6,5)	L(6,6)
	thread 2	L(1,1)			L(7,0)	L(7,1)	L(7,2)	L(7,7)
	thread 3		L(2,1)	L(2,2)				
warp 2	thread 4		L(3,1)		L(3,2)	L(3,3)		
	thread 5		L(4,0)	L(4,1)	L(4,4)			
	thread 6				L(5,2)	L(5,5)		

4. CapelliniSpTRSV

- Design to avoid deadlocks
  - A two-phase mechanism to avoid the deadlocks in CapelliniSpTRSV

4. CapelliniSpTRSV

```
main() {   //host code
           InputMatrix(L); // Rows = L.row_number
           InitiateVector(x, b, get_value); // x = 0, get_value = 0
           launchKernel(Rows); // create Rows threads
}


kernel(L, x, b, get_value) {   // GPU kernel
           rowID = globalID;
           sum = 0;
           B = getBoundary(L, rowID);
           processWhileLoop(L, b, B, rowID, sum, get_value);
           processWrtFst(L, b, B, rowID, sum, get_value, x);
}
```

(a) Two-Phase CapelliniSpTRSV

4. CapelliniSpTRSV

- Design to avoid deadlocks
  - A two-phase mechanism to avoid the deadlocks in CapelliniSpTRSV.
- Efficient last element checking
  - A novel design to reduce the number of last element checking.

4. CapelliniSpTRSV

```
processWhileLoop(L, b, B, rowID, sum, get_value){
        For id = L.rowID.start to B{
                While !checkSolve(L, id, get_value);
                recordValue(L, id, b, sum);
        }
}
processWrtFst(L, b, B, rowID, sum, get_value, x){
        id = B;
        While id < L.rowID.end{
                While checkSolve(L, id, get_value){
                        recordValue(L, id, b, sum);
                        id ++;
                }
                If id == (L.rowID.end -1){
                        computeXValue(L, x, b, sum, rowID);
                        setValue_get(rowID, get_value);
                        id ++;
                }
        }
}
```

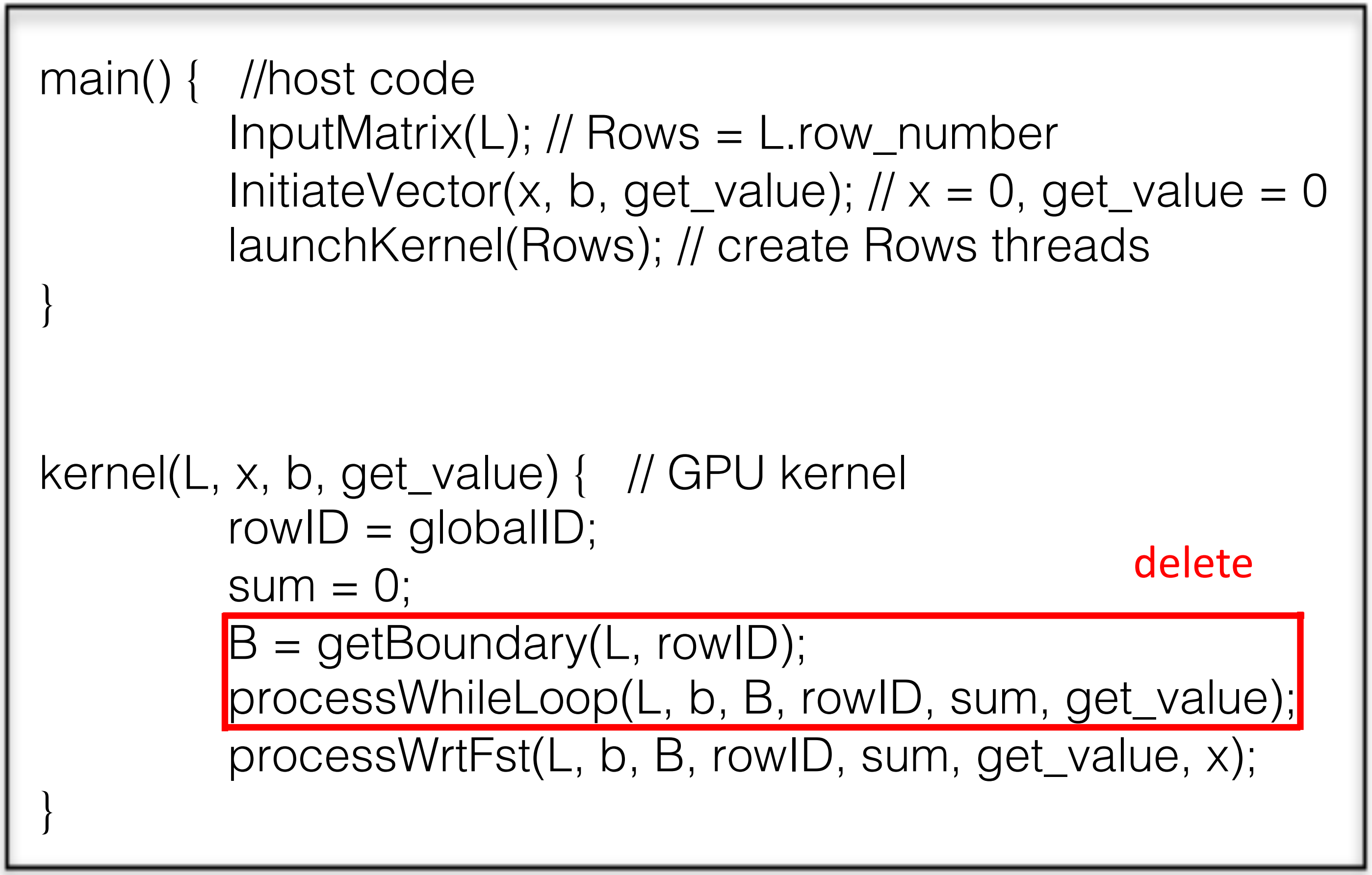
4. CapelliniSpTRSV

```
processWhileLoop(L, b, B, rowID, sum, get_value){
        For id = L.rowID.start to B{
                While !checkSolve(L, id, get_value);
                recordValue(L, id, b, sum);
        }
}
processWrtFst(L, b, B, rowID, sum, get_value, x){
        id = B;
        While id < L.rowID.end{
                While checkSolve(L, id, get_value){
                        recordValue(L, id, b, sum);
                        id ++;
                }
                If id == (L.rowID.end -1){
                        computeXValue(L, x, b, sum, rowID);
                        setValue_get(rowID, get_value);
                        id ++;
                }
        }
}
```

4. CapelliniSpTRSV

- Design to avoid deadlocks
  - A two-phase mechanism to avoid the deadlocks in CapelliniSpTRSV.
- Efficient last element checking
  - A novel design to reduce the number of such last element checkings.
- Adaptation to GPU thread execution
  - A Writing-First optimization that threads can compute the elements and write the partial results first without waiting for the other threads.

4. CapelliniSpTRSV



```
main() {   //host code
        InputMatrix(L); // Rows = L.row_number
        InitiateVector(x, b, get_value); // x = 0, get_value = 0
        launchKernel(Rows); // create Rows threads
}


kernel(L, x, b, get_value) {   // GPU kernel
        rowID = globalID;
        sum = 0;
        B = getBoundary(L, rowID);
        processWhileLoop(L, b, B, rowID, sum, get_value);
        processWrtFst(L, b, B, rowID, sum, get_value, x);
}
```

delete

(a) Two-Phase CapelliniSpTRSV

4. CapelliniSpTRSV

```
main() {   //host code
        InputMatrix(L); // Rows = L.row_number
        InitiateVector(x, b, get_value); // x = 0, get_value = 0
        launchKernel(Rows); // create Rows threads
}


kernel(L, x, b, get_value) {   // GPU kernel
        rowID = globalID;
        sum = 0;
        B = getBoundary(L, rowID);
        processWhileLoop(L, b, B, rowID, sum, get_value);
        processWrtFst(L, b, B, rowID, sum, get_value, x);
}
```

delete

(a) Two-Phase CapelliniSpTRSV

4. CapelliniSpTRSV

```
main() {   //host code
        InputMatrix(L); // Rows = L.row_number
        InitiateVector(x, b, get_value); // x = 0, get_value = 0
        LaunchKernel(Rows); // create Rows threads
}


kernel(L, x, b, get_value) {   // GPU kernel
        rowID = globalID;
        sum = 0;
        processWrtFst(L, b, L.rowID.start, rowID, sum, get_value, x);
}
```

(b)Writing-First CapelliniSpTRSV

4. CapelliniSpTRSV

Features:

- No preprocessing
  - Our algorithm can be easily applied to various situations.
- Strong effectiveness
  - Our algorithm completes the current synchronization-free SpTRSV design.
- CSR format
  - The most popular CSR format.

4. CapelliniSpTRSV

Features:

- No preprocessing
  - Our algorithm can be easily applied to various situations.
- Strong effectiveness
  - Our algorithm completes the current synchronization-free SpTRSV design.
- CSR format
  - The most popular CSR format.

4. CapelliniSpTRSV

Features:

- No preprocessing
  - Our algorithm can be easily applied to various situations.
- Strong effectiveness
  - Our algorithm completes the current synchronization-free SpTRSV design.
- CSR format
  - The most popular CSR format.

4. CapelliniSpTRSV

Features:

- No preprocessing
  - Our algorithm can be easily applied to various situations.
- Strong effectiveness
  - Our algorithm completes the current synchronization-free SpTRSV design.
- CSR format
  - The most popular CSR format.

5. Evaluation

Experimental Setup

- Methods
  - Capellini
  - SyncFree
  - cuSPARSE

- Platforms
  - Pascal: GTX 1080
  - Volta: V100
  - Turing: RTX 2080 ti
- Datasets
  - 245 matrices from University of Florida Sparse Matrix Collection

5. Evaluation

Experimental Setup

- Methods
  - Capellini
  - SyncFree
  - cuSPARSE

- Platforms
  - Pascal: GTX 1080
  - Volta: V100
  - Turing: RTX 2080 ti
- Datasets
  - 245 matrices from University of Florida Sparse Matrix Collection

5. Evaluation

Experimental Setup

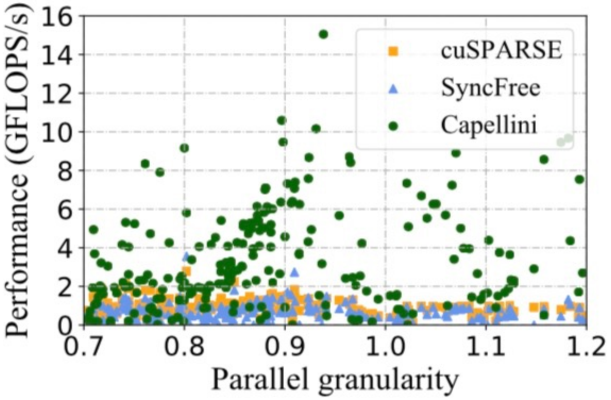
- Methods
  - Capellini
  - SyncFree
  - cuSPARSE

- Platforms
  - Pascal: GTX 1080
  - Volta: V100
  - Turing: RTX 2080 ti
- Datasets
  - 245 matrices from University of Florida Sparse Matrix Collection

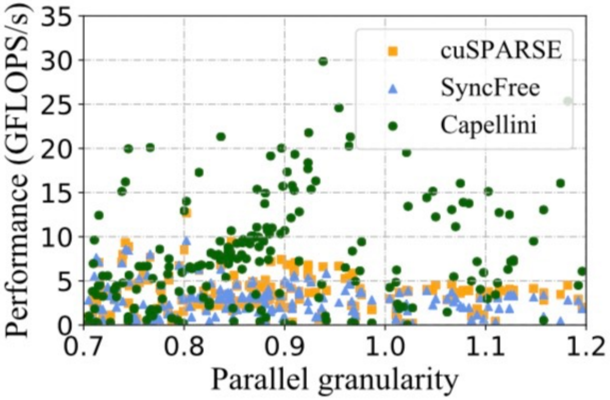
5. Evaluation

Performance (GFLOPS/s) average :

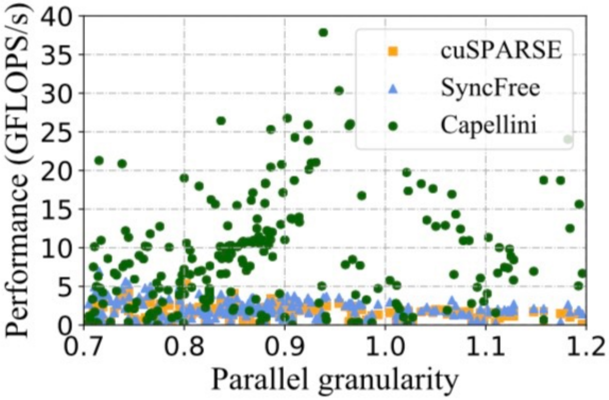
- cuSPARSE : 1.92 GFLOPS/s
- SyncFree : 1.78 GFLOPS/s
- CapelliniSpTRSV : 6.84 GFLOPS/s



(a) Pascal (GeForce GTX 1080)



(b) Volta (Tesla V100)



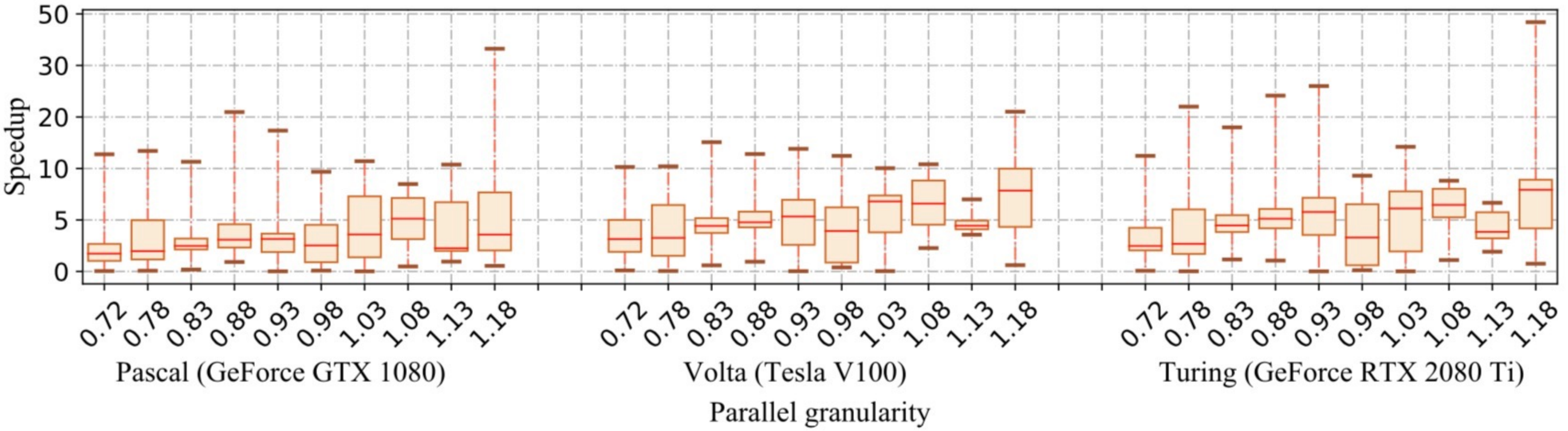
(c) Turing (GeForce RTX 2080 Ti)

5. Evaluation

Speedup average :

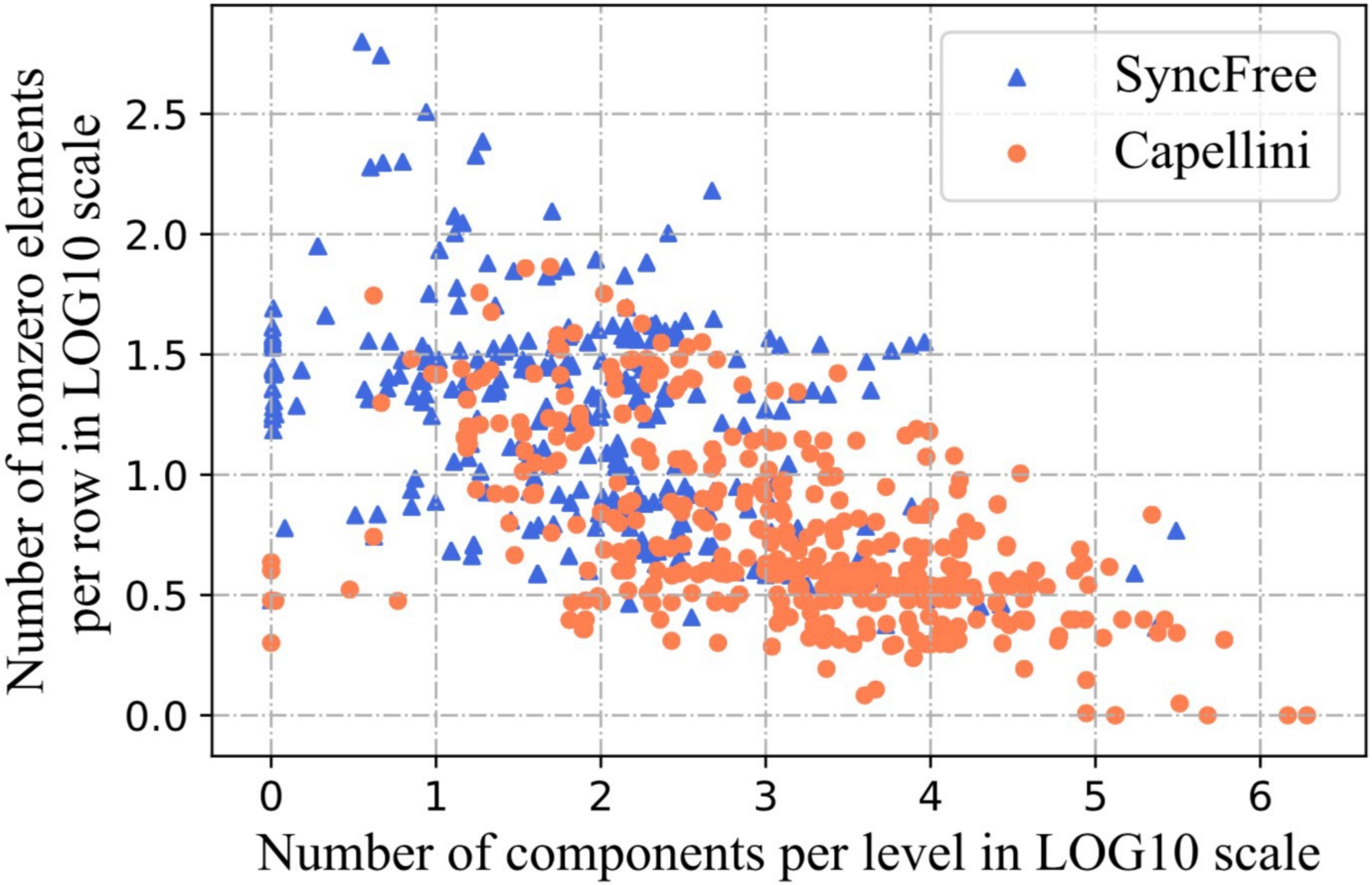
SyncFree : 4.97x

cuSPARSE: 4.74x



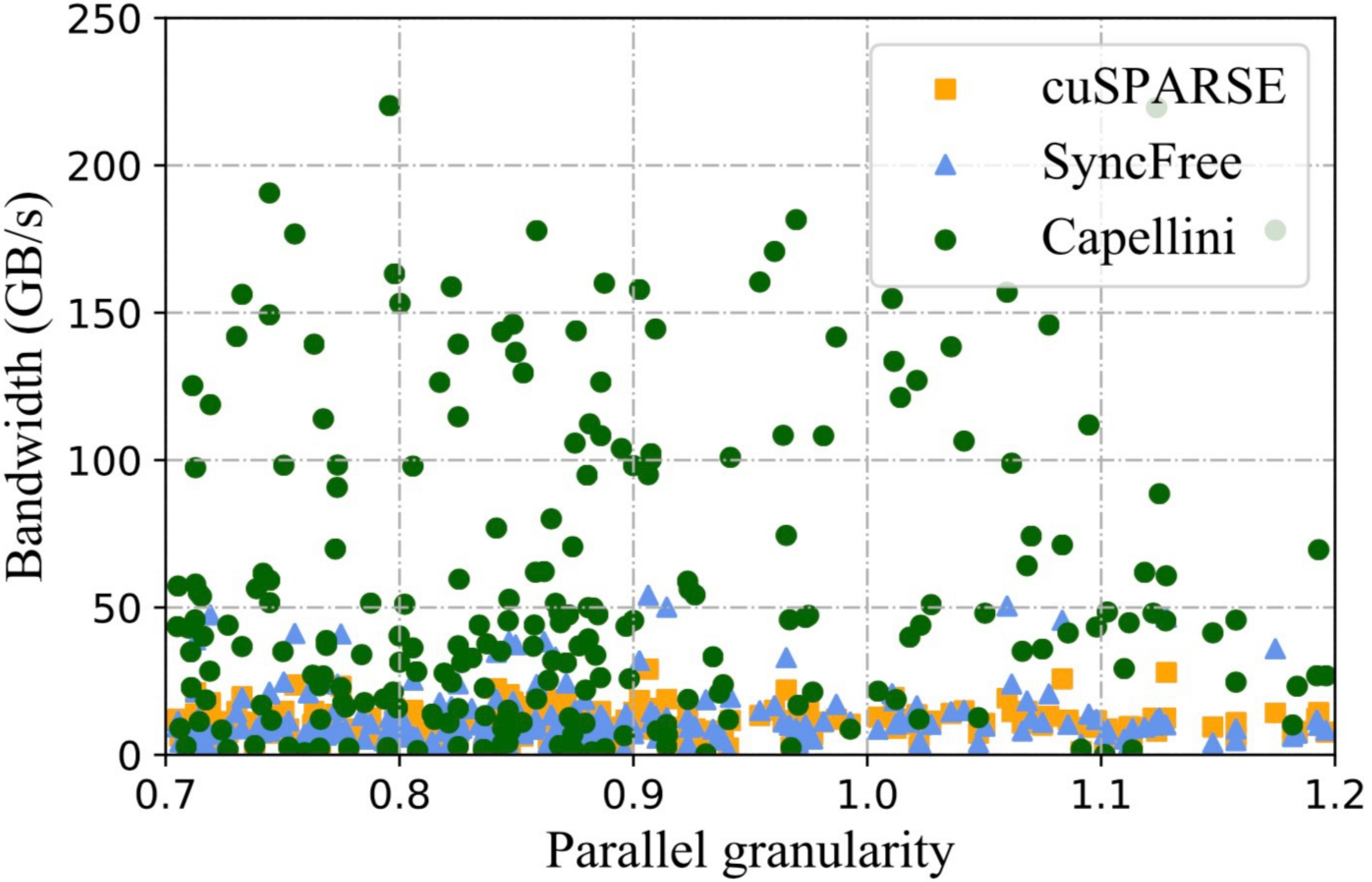
5. Evaluation

Algorithm preference distribution



5. Evaluation

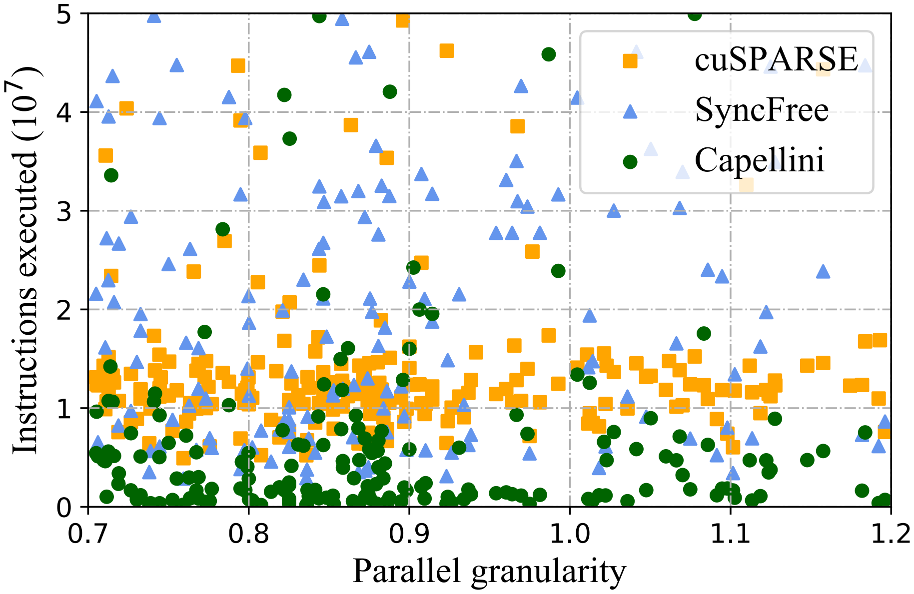
Detailed Analysis



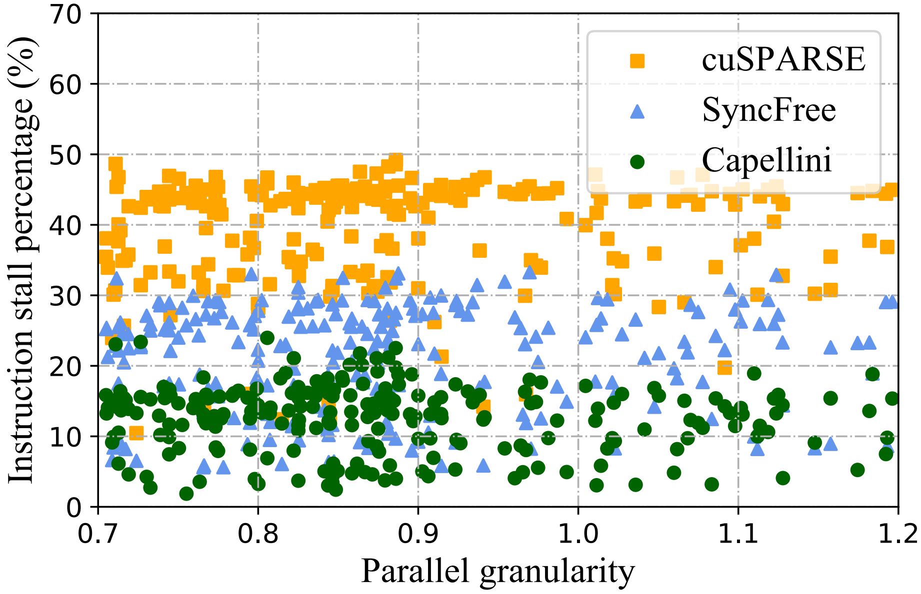
Bandwidth utilization (sum of read and write bandwidth)

5. Evaluation

- Detailed Analysis



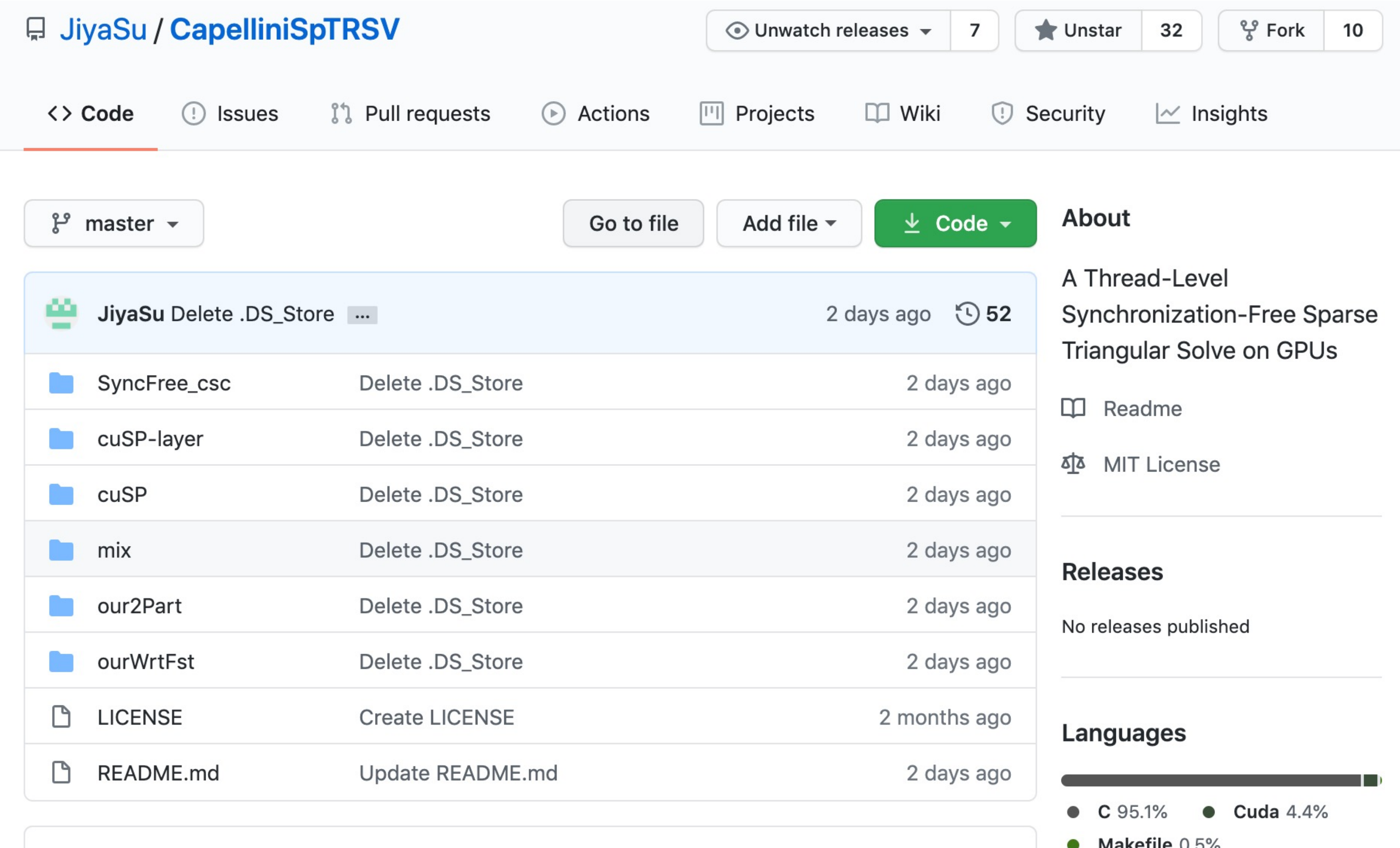
(a) Number of GPU instructions executed.



(b) Percentage of instruction dependency stalls.

6. Source Code at GitHub

- https://github.com/JiyaSu/CapelliniSpTRSV



JiyaSu / CapelliniSpTRSV

JiyaSu Delete .DS_Store … 2 days ago 52

SyncFree_csc	Delete .DS_Store	2 days ago
cuSP-layer	Delete .DS_Store	2 days ago
cuSP	Delete .DS_Store	2 days ago
mix	Delete .DS_Store	2 days ago
our2Part	Delete .DS_Store	2 days ago
ourWrtFst	Delete .DS_Store	2 days ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	2 days ago

About

A Thread-Level Synchronization-Free Sparse Triangular Solve on GPUs

Readme

MIT License

Releases

No releases published

Languages

● **C** 95.1% ● **Cuda** 4.4%
● **Makefile** 0.5%

7. Conclusion

- We show our insights in current SpTRSV algorithms and propose parallel granularity to describe sparse matrices.
- We develop CapelliniSpTRSV to process sparse matrices that previous SpTRSV algorithms cannot handle efficiently.
- We evaluate CapelliniSpTRSV with 245 matrices, and demonstrate its benefits over the state-of-the-art SpTRSV.

7. Conclusion

- We show our insights in current SpTRSV algorithms and propose parallel granularity to describe sparse matrices.
- We develop CapelliniSpTRSV to process sparse matrices that previous SpTRSV algorithms cannot handle efficiently.
- We evaluate CapelliniSpTRSV with 245 matrices, and demonstrate its benefits over the state-of-the-art SpTRSV.

7. Conclusion

- We show our insights in current SpTRSV algorithms and propose parallel granularity to describe sparse matrices.
- We develop CapelliniSpTRSV to process sparse matrices that previous SpTRSV algorithms cannot handle efficiently.
- We evaluate CapelliniSpTRSV with 245 matrices, and demonstrate its benefits over the state-of-the-art SpTRSV.

Thank you!

- Any questions?

CapelliniSpTRSV: A Thread-Level Synchronization-Free Sparse Triangular Solve on GPUs

Jiya Su◊, Feng Zhang◊, Weifeng Liu★, Bingsheng He+,

Ruofan Wu◊, Xiaoyong Du◊, Rujia Wang‡

◊Renmin University of China
⋆ China University of Petroleum
+National University of Singapore
‡ Illinois Institute of Technology

Jiya_Su@ruc.edu.cn, fengzhang@ruc.edu.cn, weifeng.liu@cup.edu.cn, hebs@comp.nus.edu.sg,
2017202106@ruc.edu.cn, duyong@ruc.edu.cn, rwang67@iit.edu