A Hybrid Synchronization Mechanism for Parallel Sparse Triangular Solve

Prabhjot Sandhu, Clark Verbrugge, and Laurie Hendren

Sable Research Group
McGill University

14 October 2021

Solving System of Linear Equations : An Example

Solve for $\mathbf{y}$ in the equation $\mathbf{Ay = b}$

$$
\begin{aligned}
y_1 + y_2 + y_3 &= 1 \\
4y_1 + 3y_2 - y_3 &= 6 \\
3y_1 + 5y_2 + 3y_3 &= 4
\end{aligned}
\quad \Rightarrow \quad
\overset{\mathbf{A}}{\begin{pmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{pmatrix}}
\overset{\mathbf{y}}{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}}
=
\overset{\mathbf{b}}{\begin{pmatrix} 1 \\ 6 \\ 4 \end{pmatrix}}
$$

Solving System of Linear Equations : An Example

To Solve **Ay = b**, Decompose **A = LU**

$$\underset{\mathbf{A}}{\begin{pmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{pmatrix}} = \underset{\mathbf{L}}{\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{pmatrix}} \underset{\mathbf{U}}{\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{pmatrix}}$$

Solving System of Linear Equations : An Example

To Solve **Ay = b**, Decompose **A = LU** and solve **LUy = b**

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{pmatrix}}_{\mathbf{L}} \underbrace{\overbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{pmatrix}}^{\mathbf{U}} \overbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}}^{\mathbf{y}}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 1 \\ 6 \\ 4 \end{pmatrix}}_{\mathbf{b}}$$

Solve for **x** in **Lx = b**, and then Solve for **y** in **Uy = x**,
where L is a lower triangular matrix, and
U is an upper triangular matrix.

Dense Triangular Solve : Inherent Sequential Execution

Solve for **x** in **Lx = b**

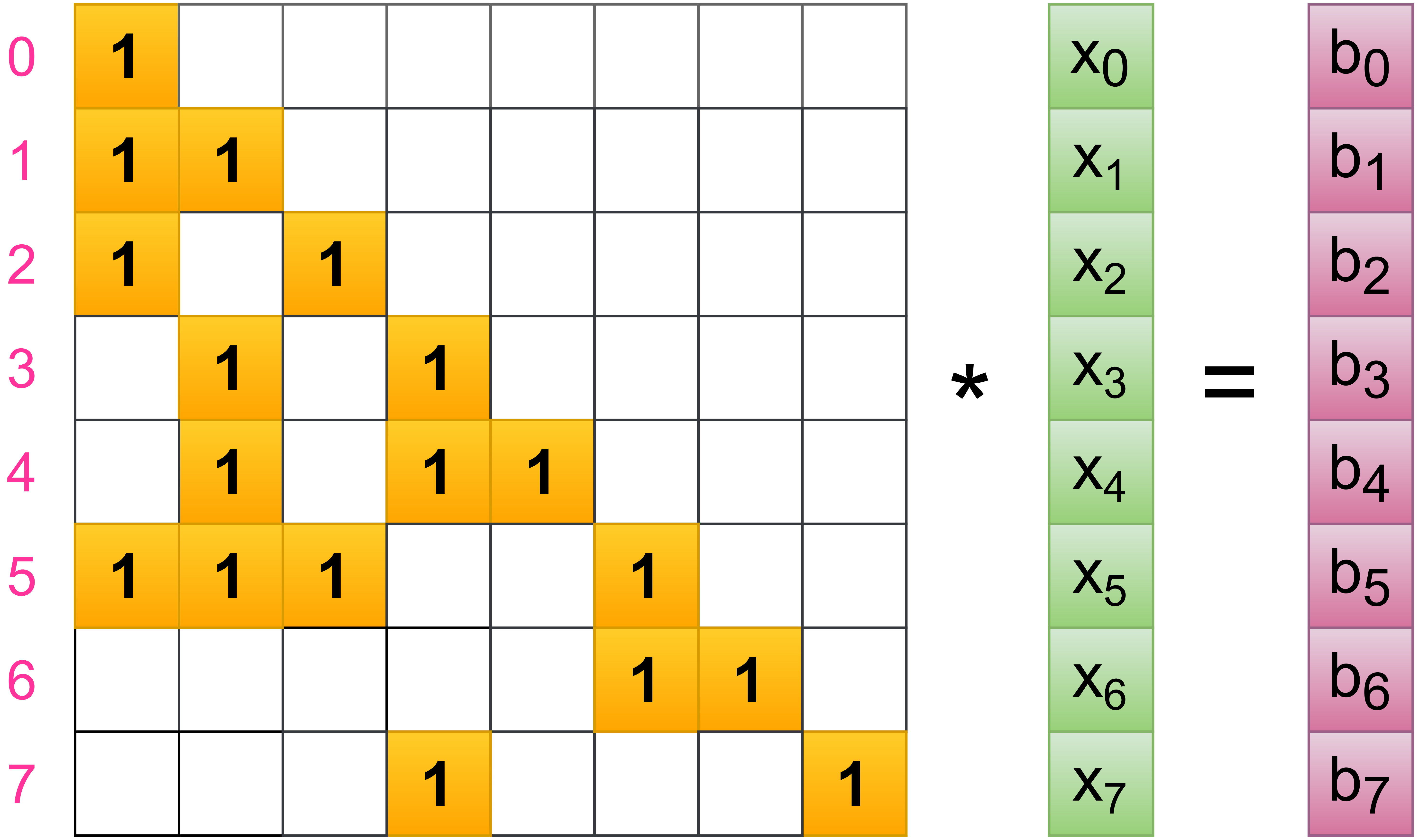
$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{pmatrix}}_{\mathbf{L}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 1 \\ 6 \\ 4 \end{pmatrix}}_{\mathbf{b}}$$

Solving x_i requires all the values from x_o to x_{i-1}

$$\begin{aligned} x_1 + 0 + 0 &= 1 \\ 4x_1 + x_2 + 0 &= 6 \\ 3x_1 - 2x_2 + x_3 &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} x_1 &= 1 \\ x_2 &= 6 - 4(1) = 2 \\ x_3 &= 4 - 3(1) + 2(2) = 5 \end{aligned}$$

Sparse Triangular Solve (SpTS)

Solve for **x** in the equation **Lx = b**

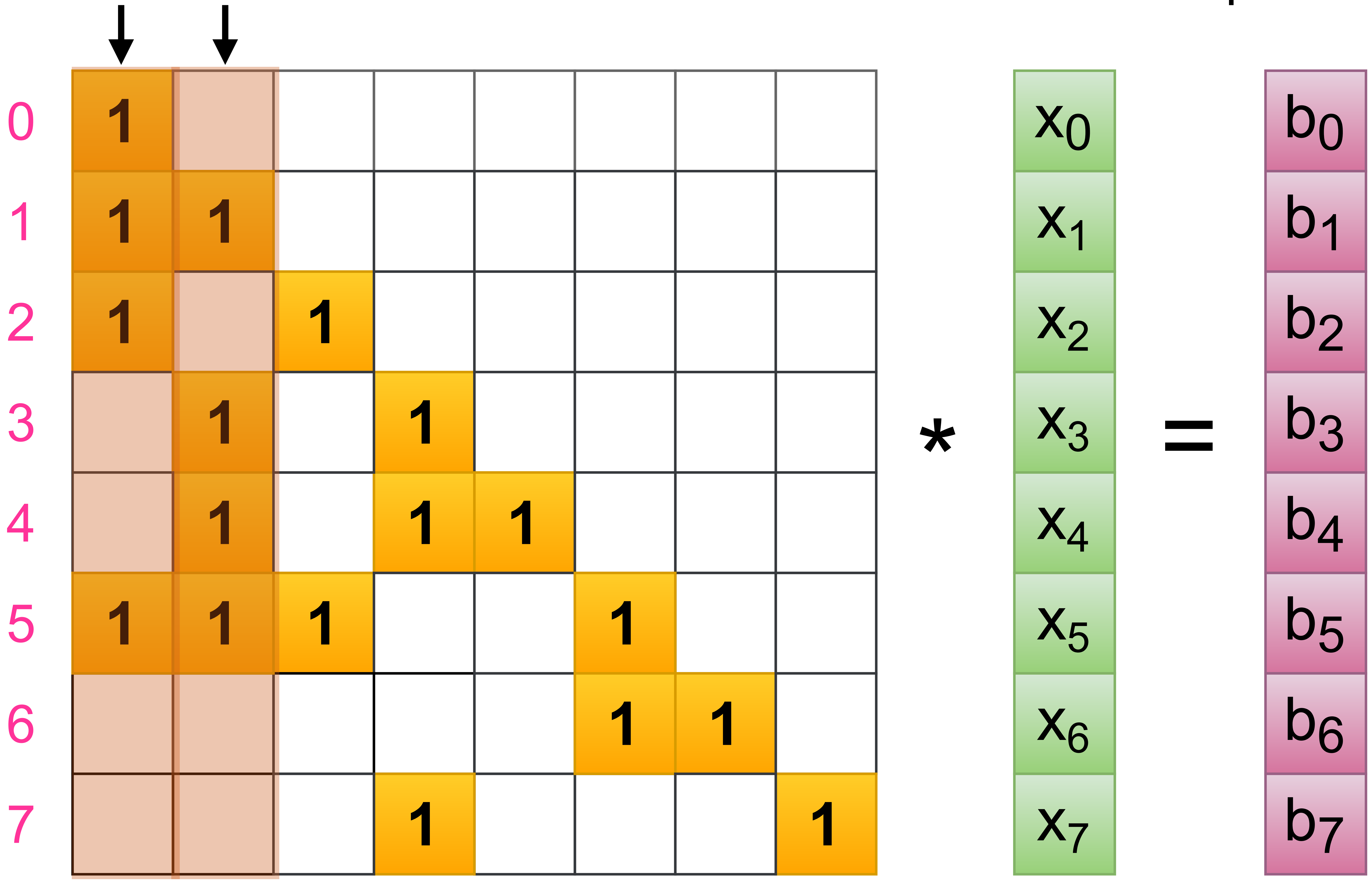


Lower Triangular Sparse Matrix

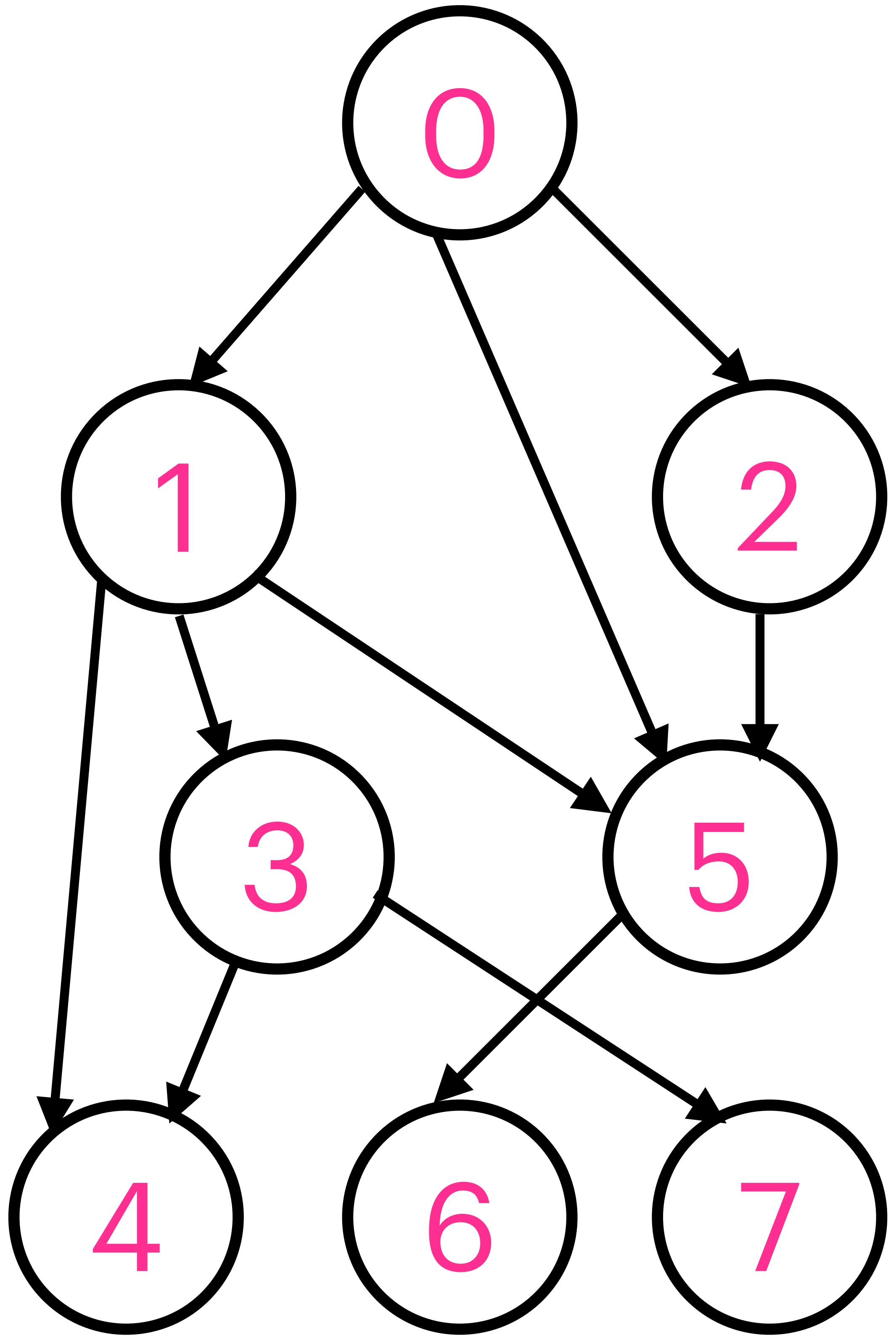
Solving x_i may not require all the values from x_o to x_{i-1}

Sparse Triangular Solve (SpTS) : Task Dependency Graph

Solve for **x** in the equation **Lx = b**



Lower Triangular Sparse Matrix



Therefore, SpTS has the potential to be computed in parallel.

Parallel SpTS : Existing Synchronization Methods

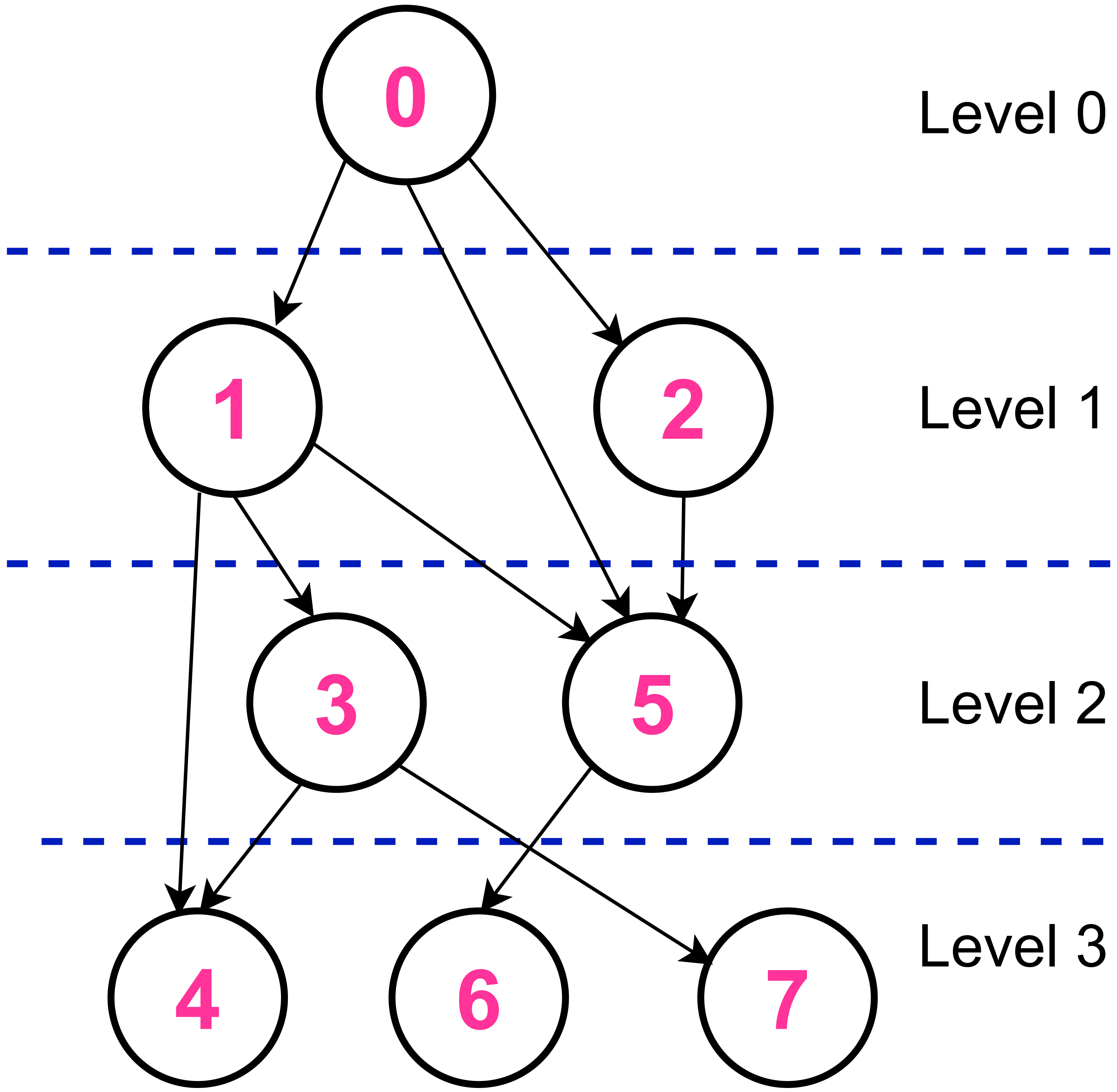
Parallel SpTS : Level-set Method

Solve for **x** in the equation **Lx = b**

- Make sets of the matrix-rows which can be solved **independently** and **simultaneously**.
- Dependency graph represents the level-set formation.
- Uses barrier synchronization.

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

Level-set formation:

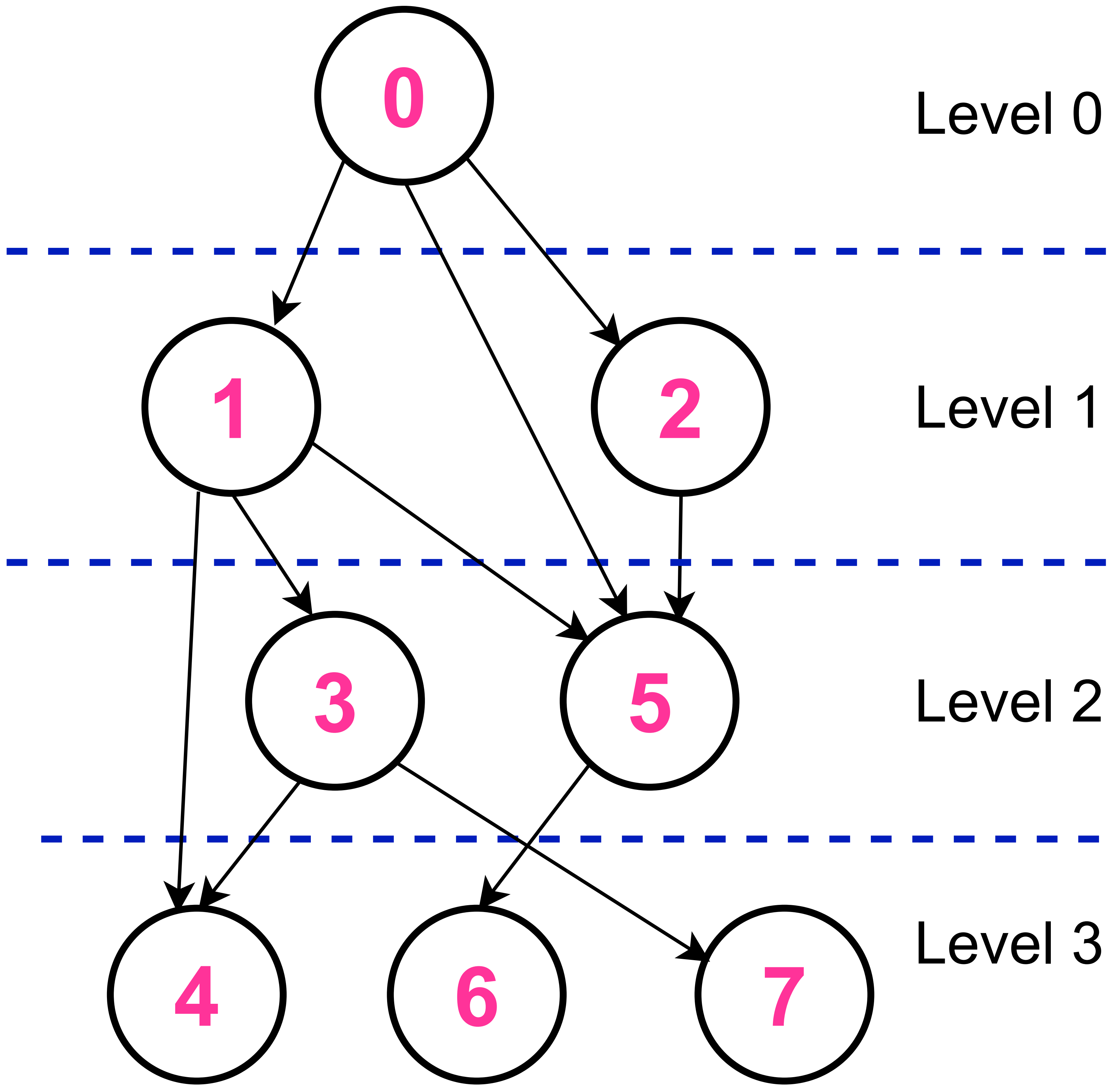


Parallel SpTS : Level-set Method

Solve for **x** in the equation **Lx = b**

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

Level-set formation:



Works well when :

- Balanced workload among the workers at each level.
- A small number of levels

```
for each level
  for each row i inside the level in parallel
    Solve x[i]
  end for
// barrier synchronization
end for
```

Parallel SpTS : Synchronization-free Method

Solve for **x** in the equation **Lx = b**

- Eliminate the pre-processing step.
- Uses atomic operations for busy-waiting.
- Effective for GPUs.

```
for each row i in parallel
  for each dependent row j
    while atomic_read(flag[j]) != 1
     // busy-wait
    end while
    Solve x[i]
  end for
  Solve x[i]
  atomic_write(flag[i], 1)
end for
```

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

Limitations of the existing methods

Level-set method

- Large number of level-sets -> costly barrier synchronization.
- Small and varied number of components per level -> waste the assigned CPU resources.
- Uneven distribution of non-zeros among the rows -> load imbalance.

Synchronization-free method

- Highly impractical for CPUs due to the heavy use of expensive atomic and busy-waiting operations on the limited number of threads.

Our Objective

Improve the performance of parallel SpTS for WebAssembly on CPUs

How?

1. Avoid synchronization barriers !

2. Minimize the use of atomic operations as much as possible !

Why WebAssembly?

1. A new low-level target language for the web.

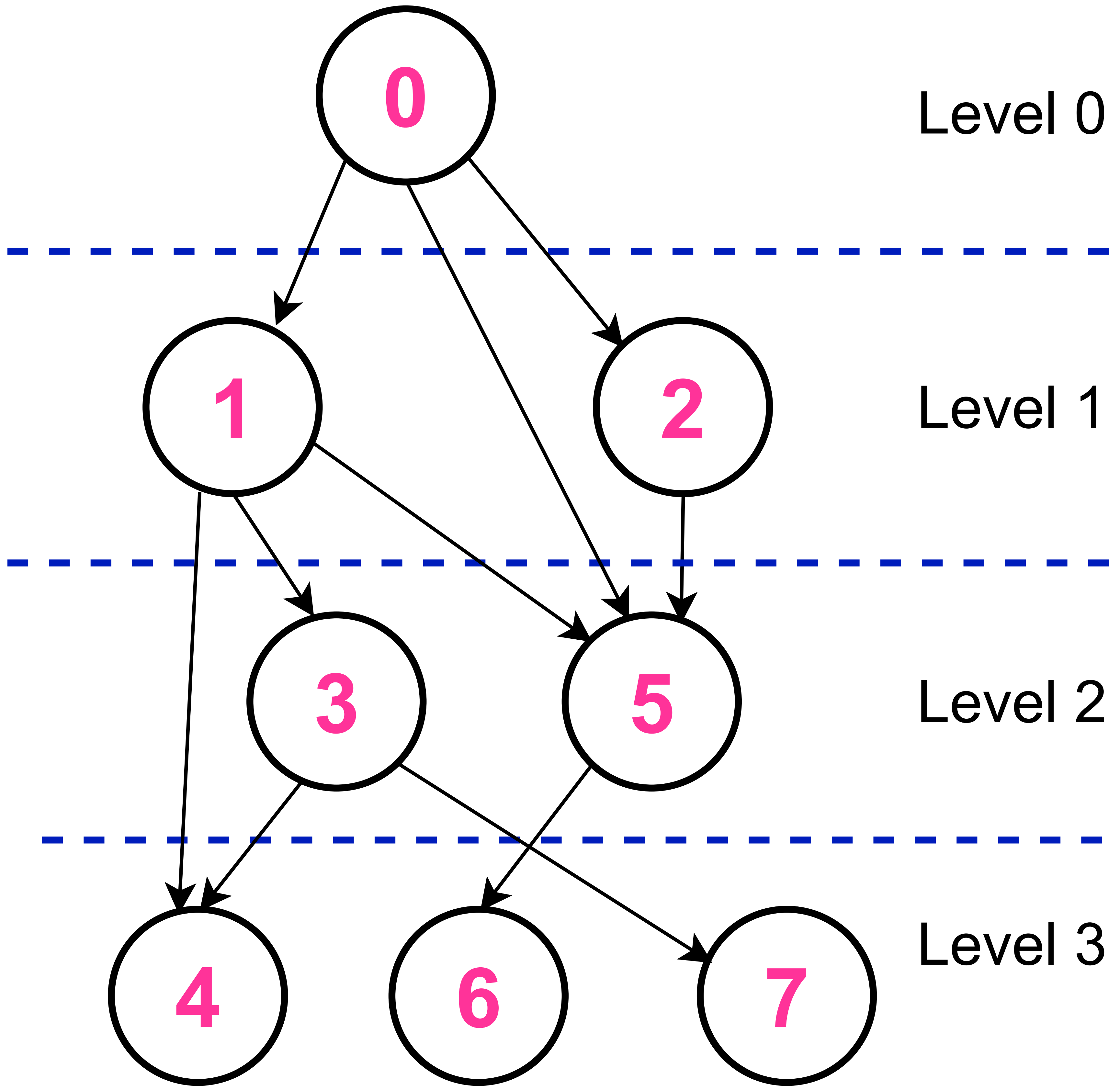
2. Building efficient web-based sparse matrix kernels for ML.

No synchronization barriers

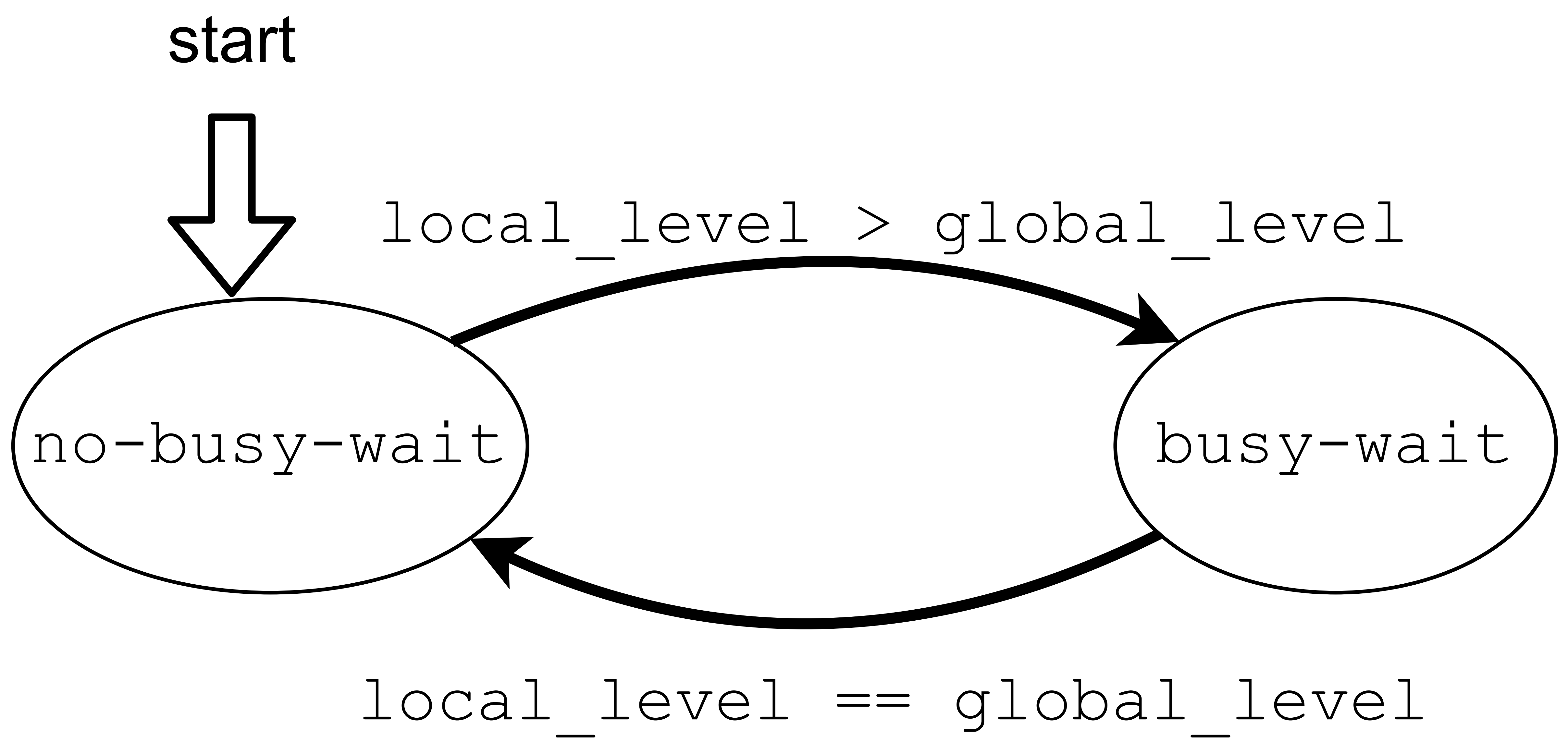
But with level-set formation

- Keep the pre-processing step of level-set method.
- Why? A systematic way to guarantee: the threads at the same level can make progress independently and simultaneously.
- Spatial locality benefits from the level-set formation.
- Additionally, allow each thread to immediately process the next level after the completion of its work at the current level.

Level-set formation:



Our Technique : Two Synchronization Modes



- **no-busy-wait (default) :** when the *current working level of the thread* (***local_level***) is equal to the *maximum working level achieved by all the threads* (***global_level***).
- **busy-wait :** when *local_level* is greater than *global_level,* indicating that the thread is presently working in the advanced levels.
- Each thread can dynamically switch between the two modes as many times as required.

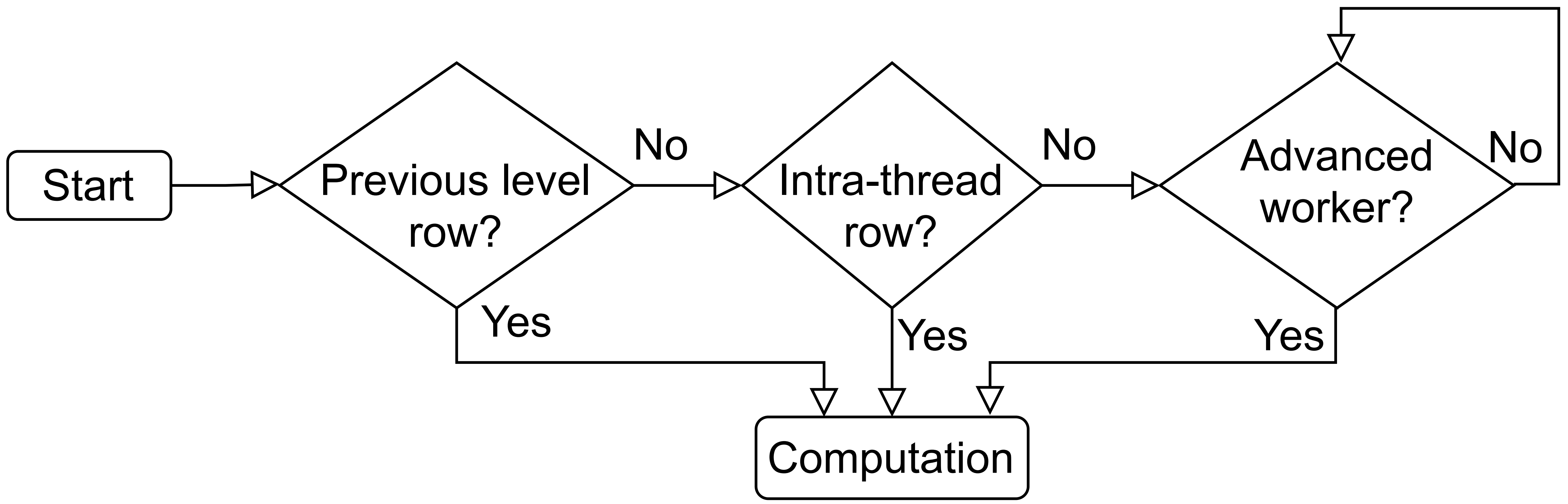
An example to contrast the parallel SpTS workflow

A tuple **(current_level, row, column)** represents the values of these parameters for each thread.

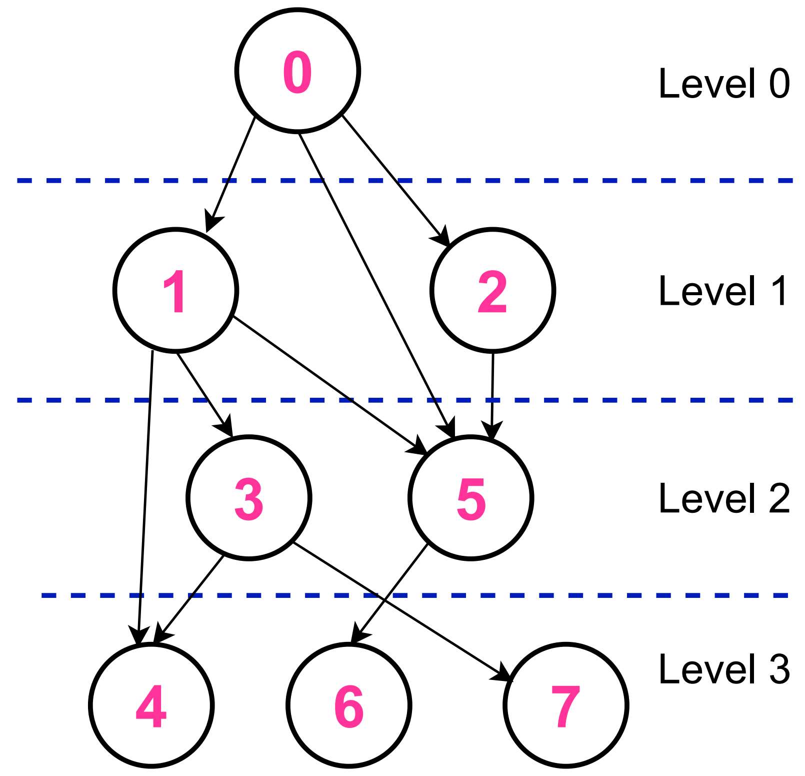
busy-wait synchronization mode

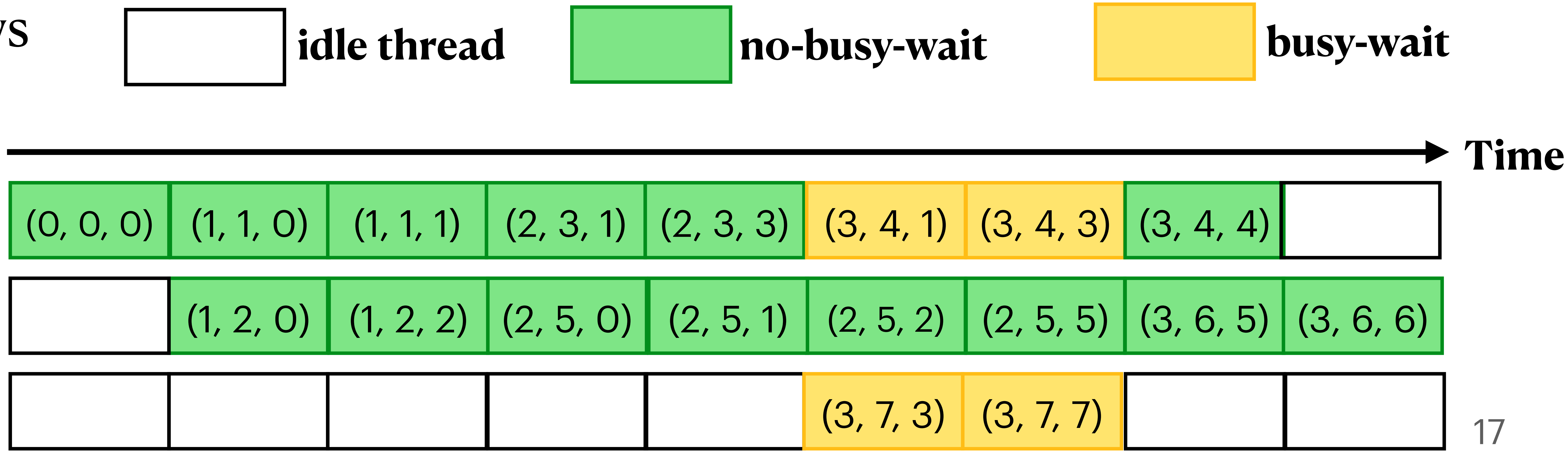
Classify required-rows into 4 exhaustive categories : allows progress

- Previous-level rows
- Intra-thread rows
- Advanced-worker inter-thread rows
- Inter-thread rows



Level-set formation:

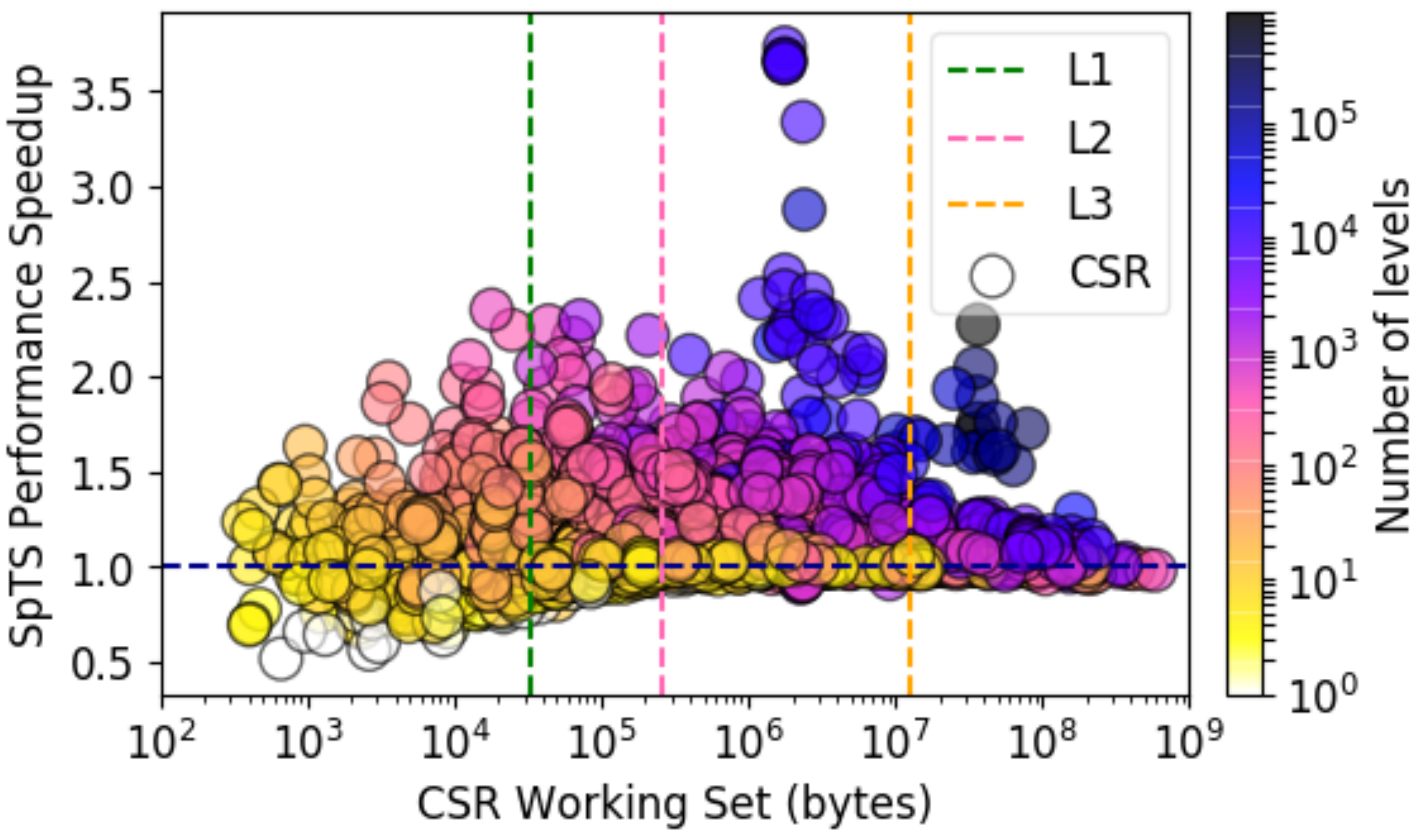




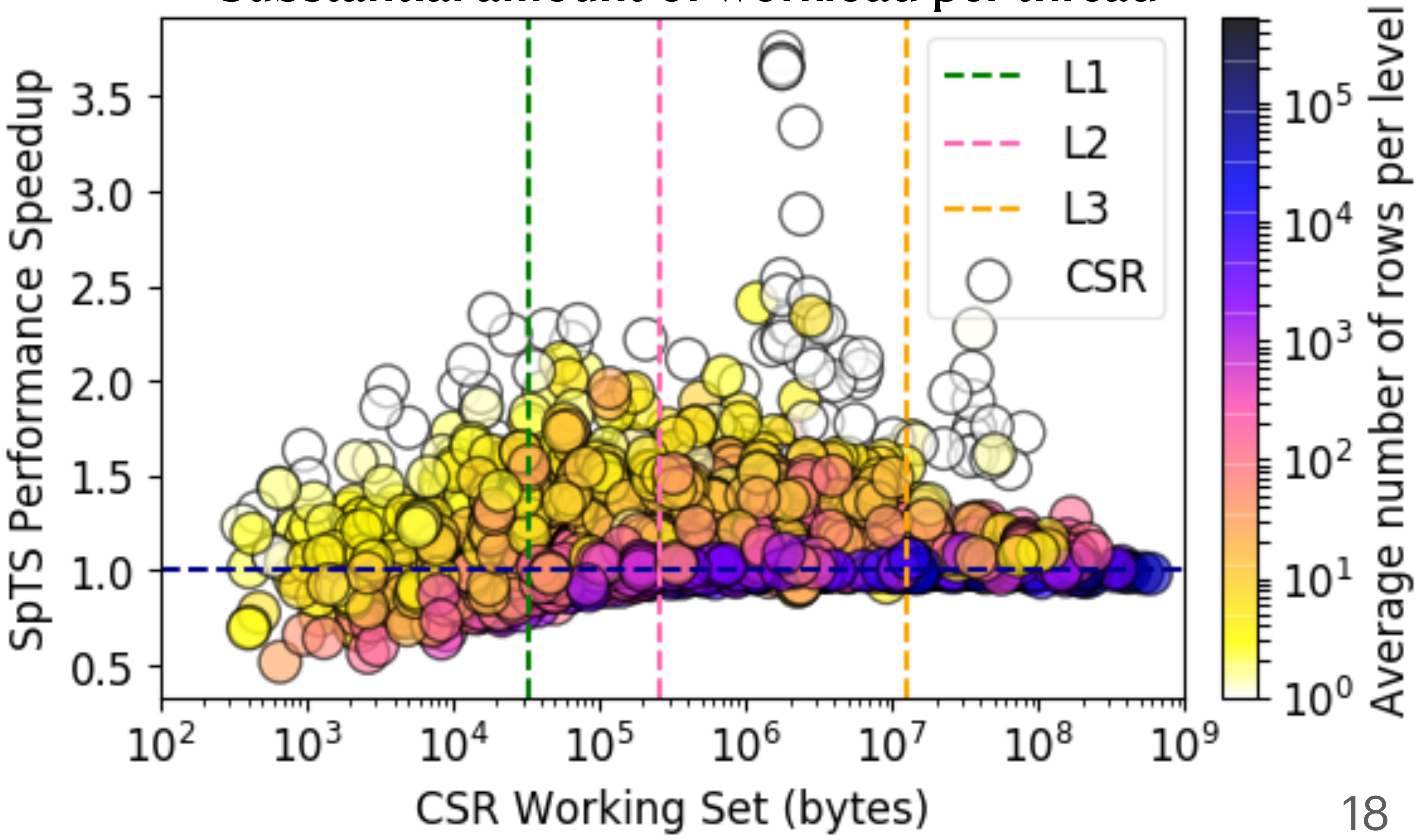
SpTS Performance Comparison

Our Hybrid Method vs Level-set Method : higher is better

Machine : Intel Core i7-3930K with 6 3.20GHz cores, 12MB last-level cache and 16GB memory, running Ubuntu Linux 18.04.5
Input : 1957 real-life sparse matrices from The SuiteSparse Matrix Collection.
Storage Format : CSR (Compressed Sparse Row)
Target Language : WebAssembly & JavaScript
Execution Environment : Chrome 92 headless browser



Substantial amount of workload per thread



Below 1

Matrices where level-set method performs better than hybrid method

Matrix	N	nnz	nlevels	N/nlevels	Level-set Performance (GFLOPS)	Hybrid Performance (GFLOPS)	Speedup
t2dal_e	4257	4257	1	4257	1.70	1.52	0.89x
bcspwr08	1624	3837	14	116	0.89	0.74	0.83x
t3dl_a	20360	265113	633	32.2	1.32	1.22	0.92x
exdata_1	6001	1137751	1501	3.99	1.47	1.37	0.93x

NS : not shown in the table, investigated separately

- Large number of rows per level -> substantial amount of workload for each thread.
- Nearly balanced workload among the threads at each level (NS) -> insignificant cost of barrier synchronization.
- Small number of rows per level with large number of non-zeros per row (NS) -> minimal parallelism but likely reduced the cost of barrier synchronization.

Above 1

Matrices where hybrid method performs better than level-set method

Matrix	N	nnz	nlevels	N/nlevels	Level-set Performance (GFLOPS)	Hybrid Performance (GFLOPS)	Speedup
lung2	109460	273647	479	228.5	1.47	2.28	1.55x
delaunay_n17	131072	524248	910	144	1.36	1.82	1.34x
e40r0100	17281	257727	512	33.7	1.49	2.06	1.38x
smt	25710	1887646	4646	5.5	1.27	1.72	1.35x

NS : not shown in the table, investigated separately

- Large number of levels -> increased the cost of barrier synchronization for level-set method.
- Small to moderate number of rows per level -> limited amount of workload for each thread.
- Uneven distribution of rows among the levels (NS) -> limits the amount of workload per thread and waste CPU resources at the barriers.
- Hybrid method benefits by allowing the threads to move to further levels to perform some feasible computation.

Close to 1

Matrices where hybrid method performs similar to level-set method

Matrix	N	nnz	nlevels	N/nlevels	Level-set Performance (GFLOPS)	Hybrid Performance (GFLOPS)	Speedup
t3dl_e	20360	20360	1	20360	1.87	1.83	0.98x
mbeacxc	496	30309	214	2.3	0.76	0.76	1.00x
coPapersCiteseer	434102	16470822	8087	53.7	2.26	2.24	0.99x
kron-g500-logn18	262144	10844830	1820	144	1.21	1.19	0.98x

NS : not shown in the table, investigated separately

- Presence of diagonal matrices in both Below 1 and Close to 1 categories -> overhead of our method becomes insignificant for the large matrices with small number of levels.
- Large number of levels with little imbalanced workload (NS) -> overhead cancels out the performance gain.

Summary

- We employ the level-set formation without barrier synchronization, and make minimal use of expensive atomic operations by dynamically switching between the two synchronization modes as required.
- We evaluate the performance of hybrid method over level-set method using our WebAssembly implementations on around 2000 sparse matrices.
- Our evaluations show the potential of our method to support the adaptive synchronization techniques in the future.

Future Directions

- Explore more sparse storage formats and apply optimization techniques like SIMD.
- Employ the upcoming synchronization constructs like floating-point atomics from the rapidly evolving WebAssembly instruction set.
- Investigate pertinent matrix structure features to develop an adaptive synchronization method (I mean build a "sorting hat"!) in the future (and perhaps call our strategy to be "Ravenclaw").

Contact us :
Prabhjot Sandhu, **prabhjot.sandhu@mail.mcgill.ca**
PhD student advised by Prof. Clark Verbrugge and previously by Laurie Hendren,
School of Computer Science, McGill University, Montreal.
Webpage : **www.cs.mcgill.ca/~psandh3**

Extras

busy-wait synchronization mode

```
(f32.load (local.get $csr_val))
(i32.load (local.get $csr_col))
(local.set $required_row)
(i32.atomic.load (local.get $global_row_index))
(local.get $required_row)
(i32.le_s)
if
  (i32.load (i32.add (local.get $row_worker_index) (i32.shl (local.get $required_row)
        (i32.const 2))))
  (local.set $worker)
  (local.get $worker)
  (local.get $current_worker)
  (i32.ne)
  if
    (i32.load (i32.add (local.get $row_level_index) (i32.shl (local.get $required_row
         ) (i32.const 2))))
    (local.set $required_level)
    (loop $busy_wait_loop
      (local.get $required_level)
      (i32.atomic.load (i32.add (local.get $worker_level_index) (i32.shl (local.get
           $worker) (i32.const 2))))
      (i32.gt_s)
      (br_if $busy_wait_loop)
    )
  end
end
(f32.load (i32.add (local.get $x) (i32.shl (local.get $required_row) (i32.const 2))))
(f32.mul)
```