Parallel Quicksort Algorithms in MPI and OpenMP

Marco Tallone

May 13, 2024

DATA SCIENCE & ARTIFICIAL INTELLIGENCE

SCIENTIFIC & DATA-INTENSIVE COMPUTING

Introduction

Different parallel versions of the Quicksort algorithm in MPI and OpenMP have been implemented and compared:

- **Task Quicksort** (OpenMP only)
- **Simple Parallel Quicksort**
- **Hyperquicksort**
- **PSRS** (Parallel Sorting by Regular Sampling)

Serial Quicksort

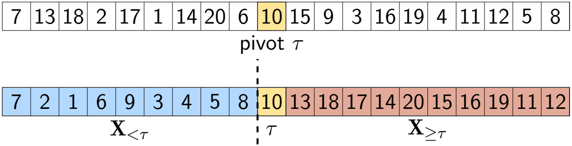
1. Choose a pivot τ from the input array $\mathbf{X}$;

7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

pivot τ

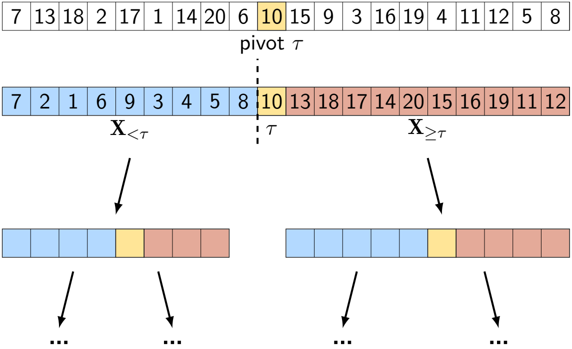
Serial Quicksort

1. Choose a pivot τ from the input array $\mathbf{X}$;
2. Partition X into two subarrays $\mathbf{X}_{<\tau}$ and $\mathbf{X}_{\geq\tau}$;



Serial Quicksort

1. Choose a pivot τ from the input array $\mathbf{X}$;
2. Partition X into two subarrays $\mathbf{X}_{<\tau}$ and $\mathbf{X}_{\geq\tau}$;
3. Recursively sort $\mathbf{X}_{<\tau}$ and $\mathbf{X}_{\geq\tau}$ until $\text{len}(\mathbf{X}) \leq 2$.



Serial Quicksort

Serial Quicksort

```
function serial_qsort(X):
    if len(X) > 2:
        τ ← choose_pivot(X)
        X_{<τ}, X_{≥τ} ← partition(X, τ)


        serial_qsort(X_{<τ})


        serial_qsort(X_{≥τ})
    else:
        if len(X) == 2 and X[0] > X[1]:
            swap(X[0], X[1])
```

Task Quicksort

OpenMP Task Quicksort

```
function omp_task_qsort(X):
    if len(X) > 2:
        τ ← choose_pivot(X)
        X_{<τ}, X_{≥τ} ← partition(X, τ)

        #pragma omp task
        omp_task_qsort(X_{<τ})

        #pragma omp task
        omp_task_qsort(X_{≥τ})
    else:
        if len(X) == 2 and X[0] > X[1]:
            swap(X[0], X[1])
```

⚠️ This function needs to be called **inside a** `#pragma omp parallel` **parallel region** to work properly.

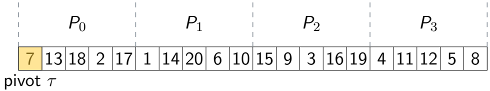
Simple Parallel Quicksort: MPI implementation

1. Split $\mathbf{X}$ in P chunks;

P_0					P_1					P_2					P_3				
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

Simple Parallel Quicksort: MPI implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ and broadcast it;



Simple Parallel Quicksort: MPI implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ and broadcast it;
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;

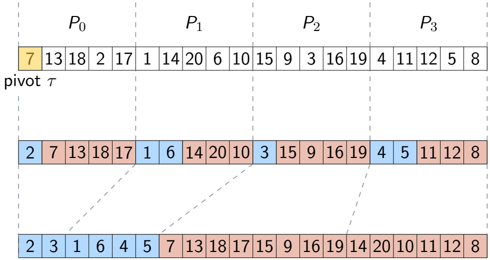
P_0					P_1					P_2					P_3				
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

pivot τ

P_0					P_1					P_2					P_3				
2	7	13	18	17	1	6	14	20	10	3	15	9	16	19	4	5	11	12	8

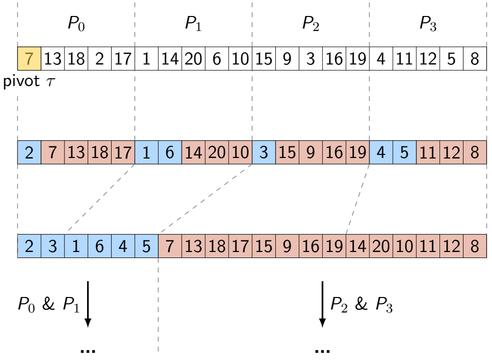
Simple Parallel Quicksort: MPI implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ and broadcast it;
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. "Low" partitions $\rightarrow$ rank $< P/2$, "High" partitions $\rightarrow$ rank $\geq P/2$;



Simple Parallel Quicksort: MPI implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ and broadcast it;
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. "Low" partitions $\rightarrow$ rank $< P/2$, "High" partitions $\rightarrow$ rank $\geq P/2$;
5. Recursively sort the chunks.



Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;

P_0				P_1				P_2				P_3				P_4			
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);

P_0				P_1				P_2				P_3				P_4			
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

pivot τ

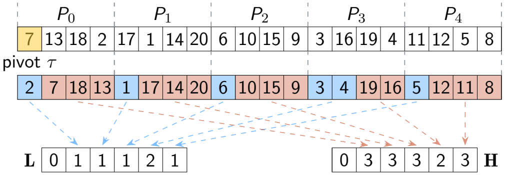
Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;

P_0				P_1				P_2				P_3				P_4			
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
pivot τ																			
2	7	18	13	1	17	14	20	6	10	15	9	3	4	19	16	5	12	11	8

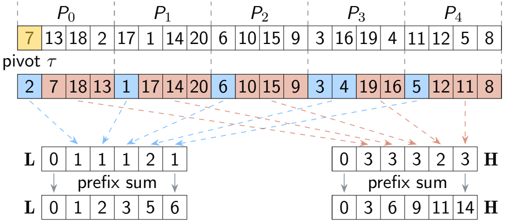
Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. $\mathbf{L} \leftarrow \#$ elements in low partitions, $\mathbf{H} \leftarrow \#$ elements in high partitions;



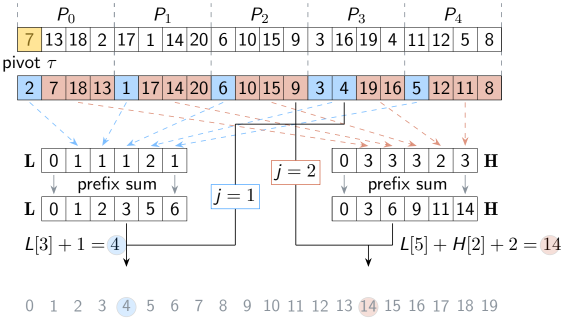
Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. $\mathbf{L} \leftarrow \#$ elements in low partitions, $\mathbf{H} \leftarrow \#$ elements in high partitions;
5. $\mathbf{L}$ and $\mathbf{H}$ prefix sums;



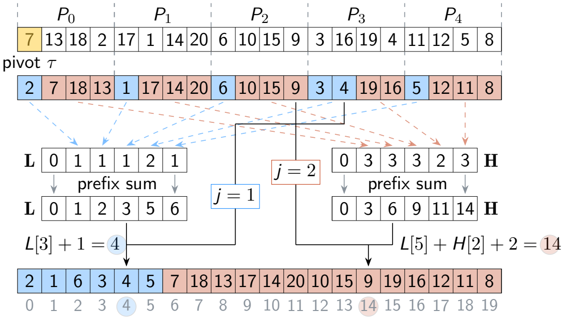
Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. $\mathbf{L} \leftarrow \#$ elements in low partitions, $\mathbf{H} \leftarrow \#$ elements in high partitions;
5. $\mathbf{L}$ and $\mathbf{H}$ prefix sums;
6. Update a **shared** `indexes` array;



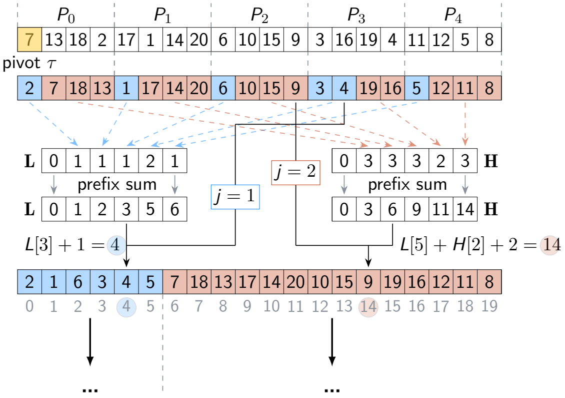
Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. $\mathbf{L} \leftarrow \#$ elements in low partitions, $\mathbf{H} \leftarrow \#$ elements in high partitions;
5. $\mathbf{L}$ and $\mathbf{H}$ prefix sums;
6. Update a **shared** `indexes` array;
7. **Serially reorder** according to `indexes`;



Simple Parallel Quicksort: OpenMP implementation

1. Split $\mathbf{X}$ in P chunks;
2. Choose **one** pivot τ (**shared** variable);
3. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
4. $\mathbf{L} \leftarrow \#$ elements in low partitions, $\mathbf{H} \leftarrow \#$ elements in high partitions;
5. $\mathbf{L}$ and $\mathbf{H}$ prefix sums;
6. Update a **shared** `indexes` array;
7. **Serially reorder** according to `indexes`;
8. Repeat recursively.



Hyperquicksort

1. Split $\mathbf{X}$ in P chunks;
2. **Each process locally sorts its chunk**
3. **One process chooses the median of its chunk as pivot τ** and broadcasts it;
4. Local partitioning in $\mathbf{X}^i_{<\tau}$ and $\mathbf{X}^i_{\geq\tau}$;
5. "Low" partitions $\rightarrow$ rank $< P/2$,
   "High" partitions $\rightarrow$ rank $\geq P/2$;
6. Recursively sort the chunks.

PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:

$$\frac{i \cdot n}{P^2}\ ,\ i = 0, 1, \ldots, P-1$$

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	**7**	13	**14**	17	18	**3**	6	**9**	10	**15**	16	20	**4**	5	**8**	11	**12**	19

PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:

$$\frac{i \cdot n}{P^2}\ ,\ i = 0, 1, \ldots, P - 1$$

4. One process:
   - Gathers and sorts all samples;
   - Regularly picks $P - 1$ pivots;
   - Broadcasts pivots.

P_0							P_1							P_2						
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8	
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19	

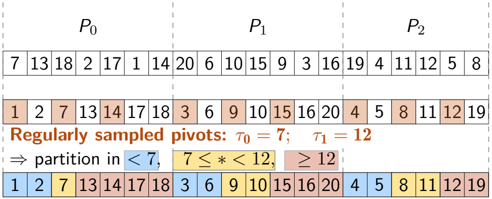
Regularly sampled pivots: $\tau_0 = 7$; $\tau_1 = 12$

PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:

$$\frac{i \cdot n}{P^2}\ ,\ i = 0, 1, \ldots, P-1$$

4. One process:
   - Gathers and sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Broadcasts pivots.
5. Local partition according to pivots;

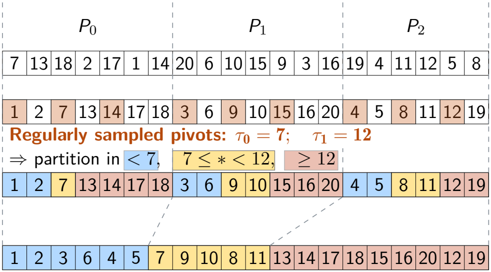


PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:

$$\frac{i \cdot n}{P^2}\ ,\ i = 0, 1, \ldots, P-1$$

4. One process:
   - Gathers and sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Broadcasts pivots.
5. Local partition according to pivots;
6. **All-to-all** exchange:
   P_i sends j^{th} partition to P_j;

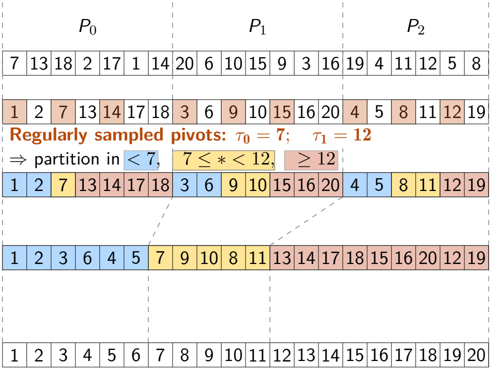


PSRS: MPI Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:

$$\frac{i \cdot n}{P^2}\ ,\ i = 0, 1, \ldots, P-1$$

4. One process:
   - Gathers and sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Broadcasts pivots.
5. Local partition according to pivots;
6. **All-to-all** exchange: P_i sends j^{th} partition to P_j;
7. Local sorting.



PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.

P_0							P_1							P_2						
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8	

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19	

Regularly sampled pivots: $\tau_0 = \mathbf{7}$; $\tau_1 = \mathbf{12}$

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

Regularly sampled pivots: $\tau_0 = 7; \quad \tau_1 = 12$

$\Rightarrow$ partition in < 7, $7 \leq * < 12$, ≥ 12

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1) \times (P+1)$ matrix $\mathbf{M}$;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

Regularly sampled pivots: $\tau_0 = 7; \quad \tau_1 = 12$

$\Rightarrow$ partition in < 7, $7 \leq * < 12$, ≥ 12

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

$$\begin{array}{|c|c|c|c|} \hline 0 & 0 & 0 & 0 \\ \hline 0 & 2 & 2 & 2 \\ \hline 0 & 1 & 2 & 2 \\ \hline 0 & 4 & 3 & 2 \\ \hline \end{array} \quad \mathbf{M}$$

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1) \times (P+1)$ matrix $\mathbf{M}$;

P_0							P_1							P_2						
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8	

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19	

Regularly sampled pivots: $\tau_0 = 7$; $\tau_1 = 12$

$\Rightarrow$ partition in < 7, $7 \leq * < 12$, ≥ 12

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19	

$\mathbf{M}$

0	0	0	0
0	2	4	6
0	1	3	5
0	7	4	9

$\longrightarrow$ prefix sum

PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1) \times (P+1)$ matrix $\mathbf{M}$;
7. Prefix sum $\mathbf{S}$ of last column;

P_0							P_1							P_2					
7	13	18	2	17	1	14	20	6	10	15	9	3	16	19	4	11	12	5	8
1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

Regularly sampled pivots: $\tau_0 = 7$; $\tau_1 = 12$

$\Rightarrow$ partition in < 7, $7 \leq * < 12$, ≥ 12

1	2	7	13	14	17	18	3	6	9	10	15	16	20	4	5	8	11	12	19

0	0	0	0
0	2	4	6
0	1	3	5
0	7	4	9

$\mathbf{M}$

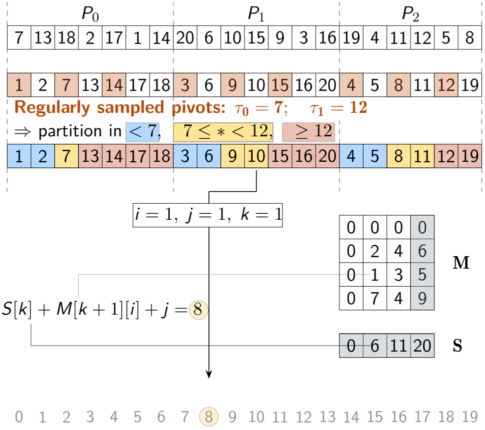
prefix sum

0	6	11	20

$\mathbf{S}$

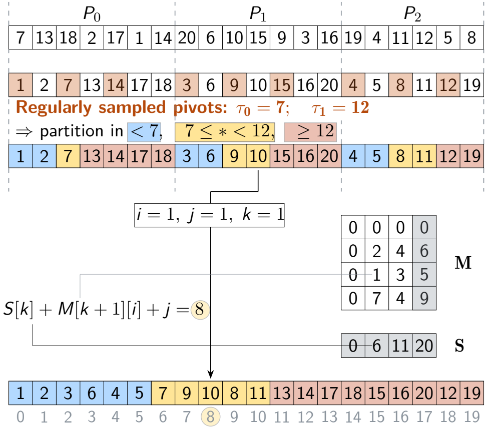
PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1) \times (P+1)$ matrix $\mathbf{M}$;
7. Prefix sum $\mathbf{S}$ of last column;
8. Update the `indexes` array;



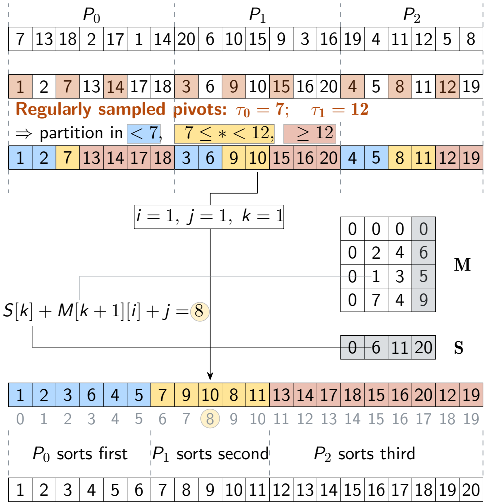
PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples:
   $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1) \times (P+1)$ matrix $\mathbf{M}$;
7. Prefix sum $\mathbf{S}$ of last column;
8. Update the `indexes` array;
9. **Serially reorder** elements according to `indexes`;



PSRS: OpenMP Implementation

1. Split $\mathbf{X}$ in P chunks;
2. Local chunk **sorting**;
3. Each selects P samples: $\rightarrow$ **shared** variable;
4. Master thread:
   - Sorts all samples;
   - Regularly picks $P-1$ pivots;
   - Writes pivots in shared array.
5. Local partition according to pivots;
6. $(P+1)\times(P+1)$ matrix $\mathbf{M}$;
7. Prefix sum $\mathbf{S}$ of last column;
8. Update the `indexes` array;
9. **Serially reorder** elements according to `indexes`;
10. Each thread sorts from $\mathbf{S}[i]$ to $\mathbf{M}[i+1][P]$



Scaling Analysis

Strong Scaling

Measuring how the execution time t varies with the number of processors P for a fixed total workload.

Weak Scaling

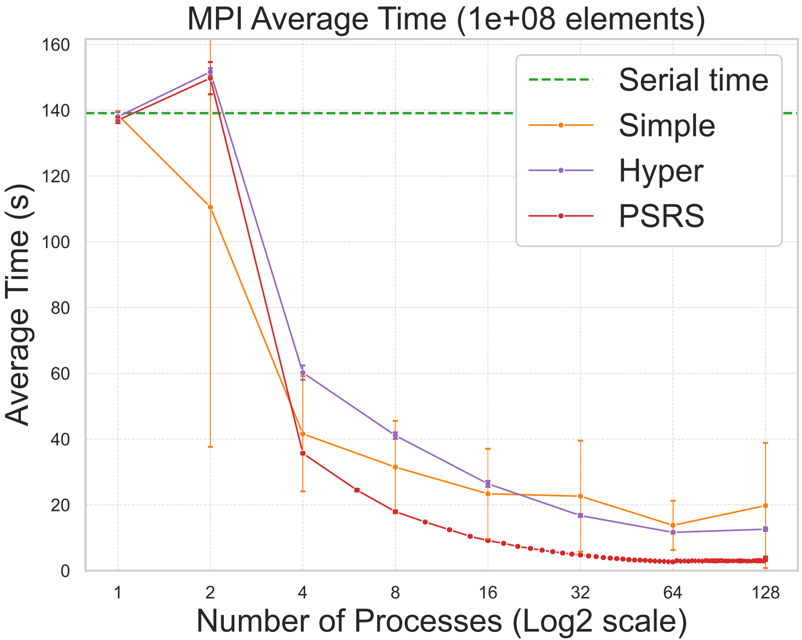
Measuring how the execution time t varies with the number of processors P for a fixed workload per processor.

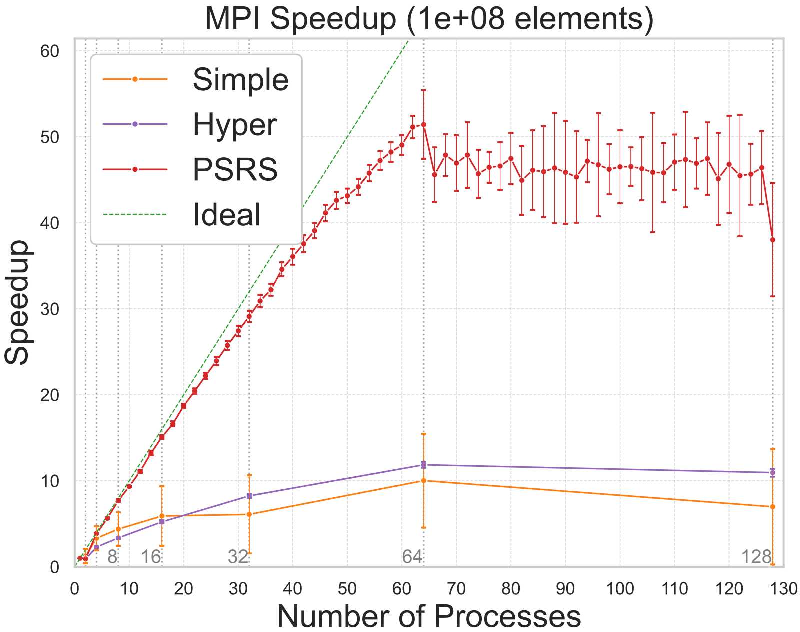
MPI Implementation

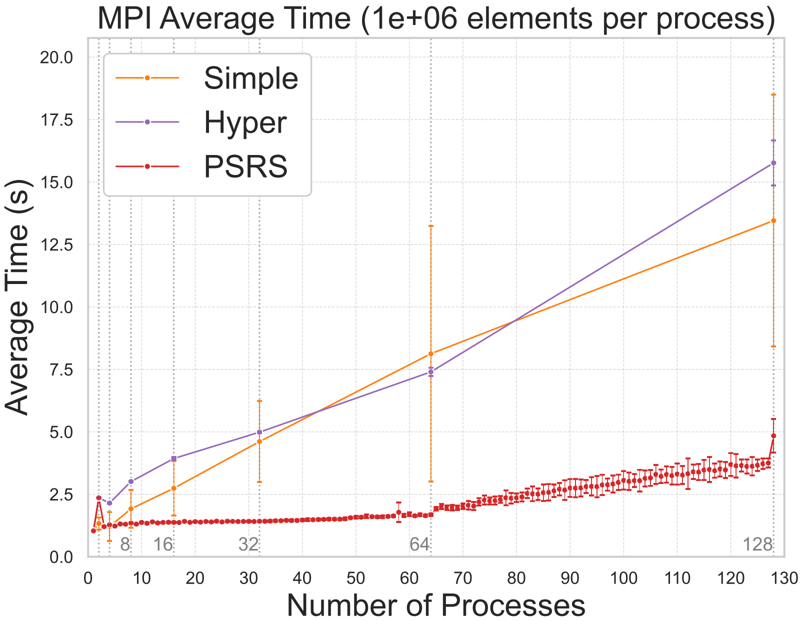
- Strong Scaling $t - P$;
- Strong Scaling $S_p - P$;
- Weak Scaling $t - P$;
- Weak Scaling $\varepsilon - P$.

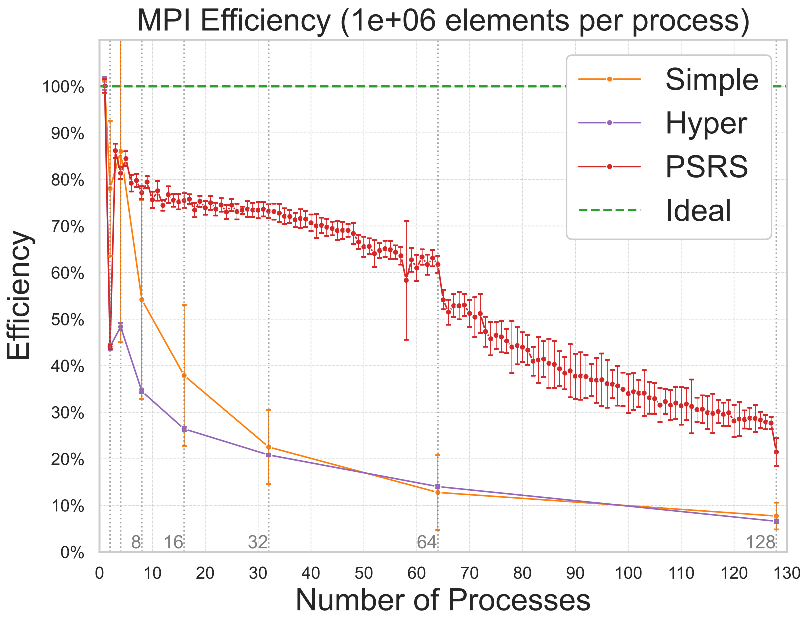
OpenMP Implementation

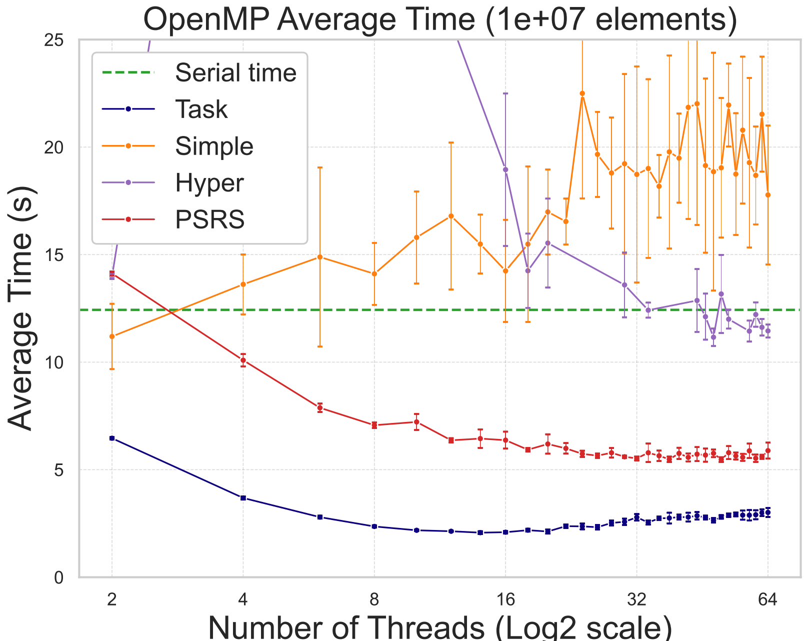
- Strong Scaling $t - P$;
- Strong Scaling $S_p - P$;
- Weak Scaling $t - P$;
- Weak Scaling $\varepsilon - P$.

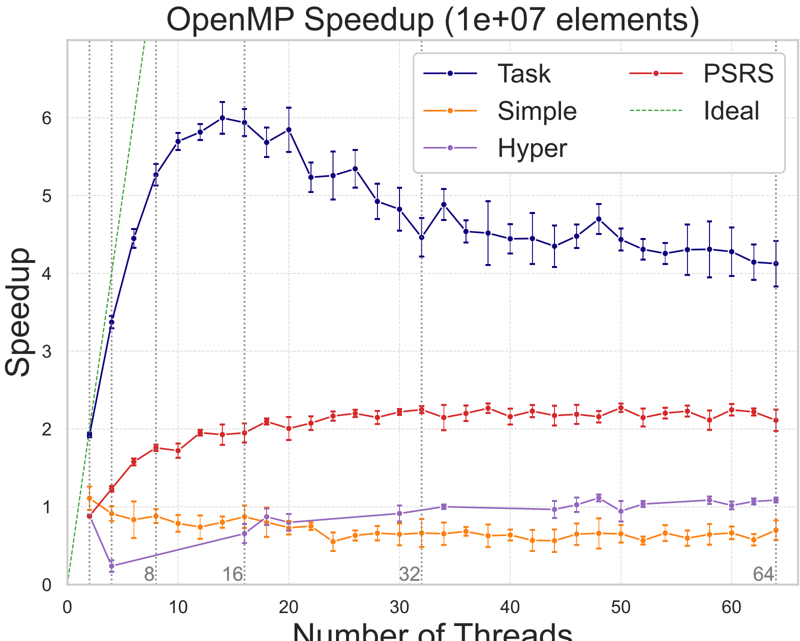

MPI Average Time (1e+08 elements)
Serial time
Simple
Hyper
PSRS
Average Time (s)
0
20
40
60
80
100
120
140
160
Number of Processes (Log2 scale)
1
2
4
8
16
32
64
128

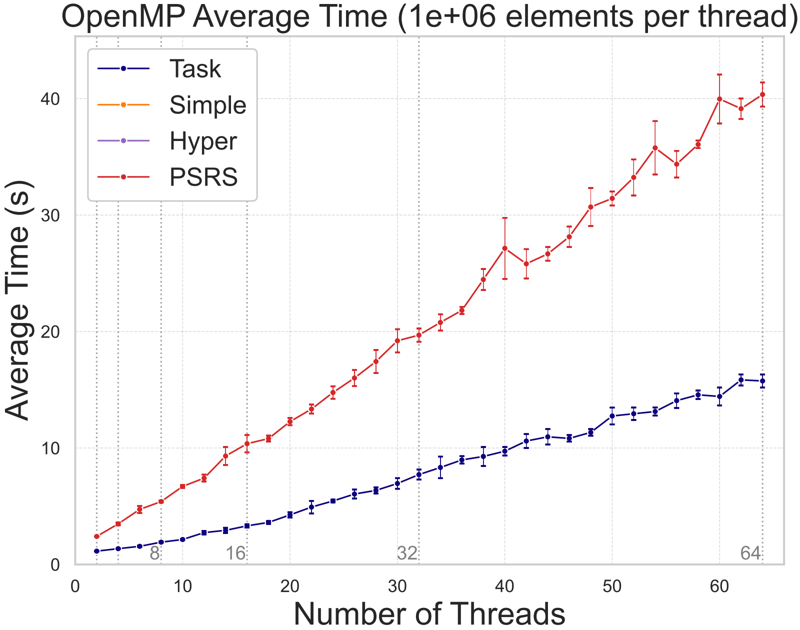

MPI Speedup (1e+08 elements)
Simple
Hyper
PSRS
Ideal
Speedup
Number of Processes
0
10
20
30
40
50
60
70
80
90
100
110
120
130
8
16
32
64
128

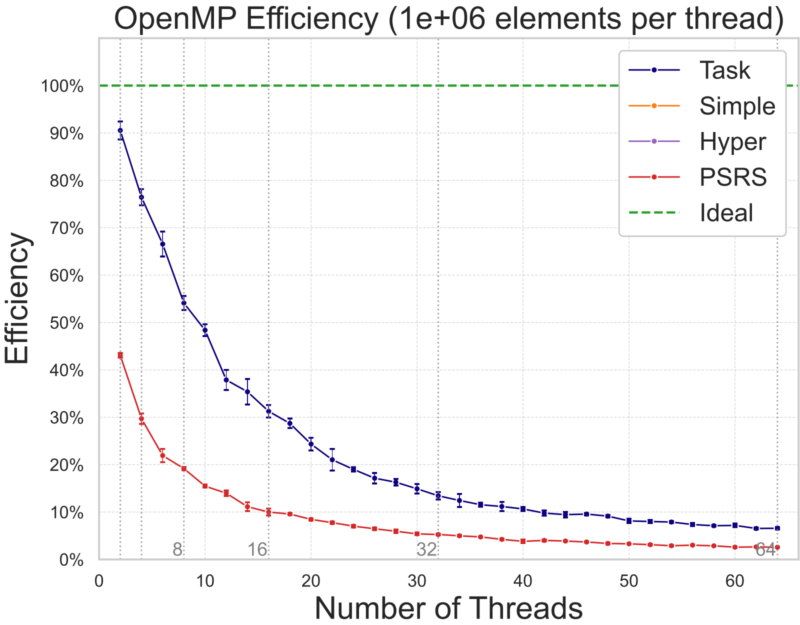

MPI Average Time (1e+06 elements per process)
Simple
Hyper
PSRS
Average Time (s)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
8
16
32
64
128
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Number of Processes


MPI Efficiency (1e+06 elements per process)
Efficiency
Number of Processes
Simple
Hyper
PSRS
Ideal
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
0
10
20
30
40
50
60
70
80
90
100
110
120
130
8
16
32
64
128


OpenMP Average Time (1e+07 elements)
Serial time
Task
Simple
Hyper
PSRS
Average Time (s)
0
5
10
15
20
25
2
4
8
16
32
64
Number of Threads (Log2 scale)


OpenMP Speedup (1e+07 elements)
Task
Simple
Hyper
PSRS
Ideal
Speedup
Number of Threads
0
1
2
3
4
5
6
0
10
20
30
40
50
60
8
16
32
64


OpenMP Average Time (1e+06 elements per thread)
Task
Simple
Hyper
PSRS
Average Time (s)
0
10
20
30
40
Number of Threads
0
10
20
30
40
50
60
8
16
32
64


OpenMP Efficiency (1e+06 elements per thread)
Efficiency
Number of Threads
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
10
20
30
40
50
60
8
16
32
64
Task
Simple
Hyper
PSRS
Ideal

References

AMD.
Epyc 7h12 specifications, 2023.

OpenMP Architecture Review Board.
Openmp application program interface, 2023.

MPI Forum.
Mpi: A message-passing interface standard, 2023.

Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar.
Introduction to Parallel Computing.
Addison-Wesley, 2nd edition, 2003.

Frank Nielsen.
Introduction to HPC with MPI for Data Science.
Springer, 2016.

AREA Science Park.
Orfeo documentation, 2023.