Implementation of Parallel Bitonic Sort using MPI and Intel TBB

Presented for CSE702 Fall 2021

Instructor: Dr. Russ Miller

Presenter: Zainul Abideen Sayed

University at Buffalo The State University of New York

University at Buffalo The State University of New York

Why care about parallel sorting algorithms ?

Why parallel sorting?

- Sorting is one of the most common operations in computation.
- The advancement in parallel hardware.
- Increasing nodes in a cluster and cores in a processor.
- Efficient utilization of resources.
- Therefore, good parallel sorting algorithms are needed.

Popular parallel sorting algorithms

- Bitonic sort
- Sample sort
- Merge sort
- Quick sort
- Radix sort

Bitonic sorting algorithm is based on bitonic sorting network. The key operation is based on the sorting network which converts a given sequence into a bitonic sequence and finally bitonic merge can produce a monotonically increasing or decreasing sequency.

Bitonic Sort Principle

- Bitonic sequence <1, 2, 4, 7, 6, 0> <8, 9, 2, 1, 0, 4> <0, 4, 8, 9, 2, 1> <3, 4, 7, 8, 6, 5, 2, 1>
- Let s = <a0, a1, ..., an-1>
- $s_1 = \{ \min(a_o, a_{n/2}), \min(a_1, a_{n/2+1}),\ldots\ldots,\min(_{an/2\text{-}1}, a_{n\text{-}1}) \}$
- $s_2 = \{ \max(a_o, a_{n/2}), \max(a_1, a_{n/2+1}),\ldots\ldots,\max(_{an/2\text{-}1}, a_{n\text{-}1}) \}$
- In sequence s_1 , there is an element $\mathbf{b_i = \min\{ ai , a_{n/2+i} \}}$ such that all the elements before b_i are from the increasing part of the original sequence and all the elements after b_i are from the decreasing part.
- Opposite case for $\mathbf{b_i` = \max\{ ai , a_{n/2+i} \}}$

*Introduction to Parallel Computing 2nd Edition, Ananth Grama

Example

Log_2N

3 4 7 8 | 6 5 2 1

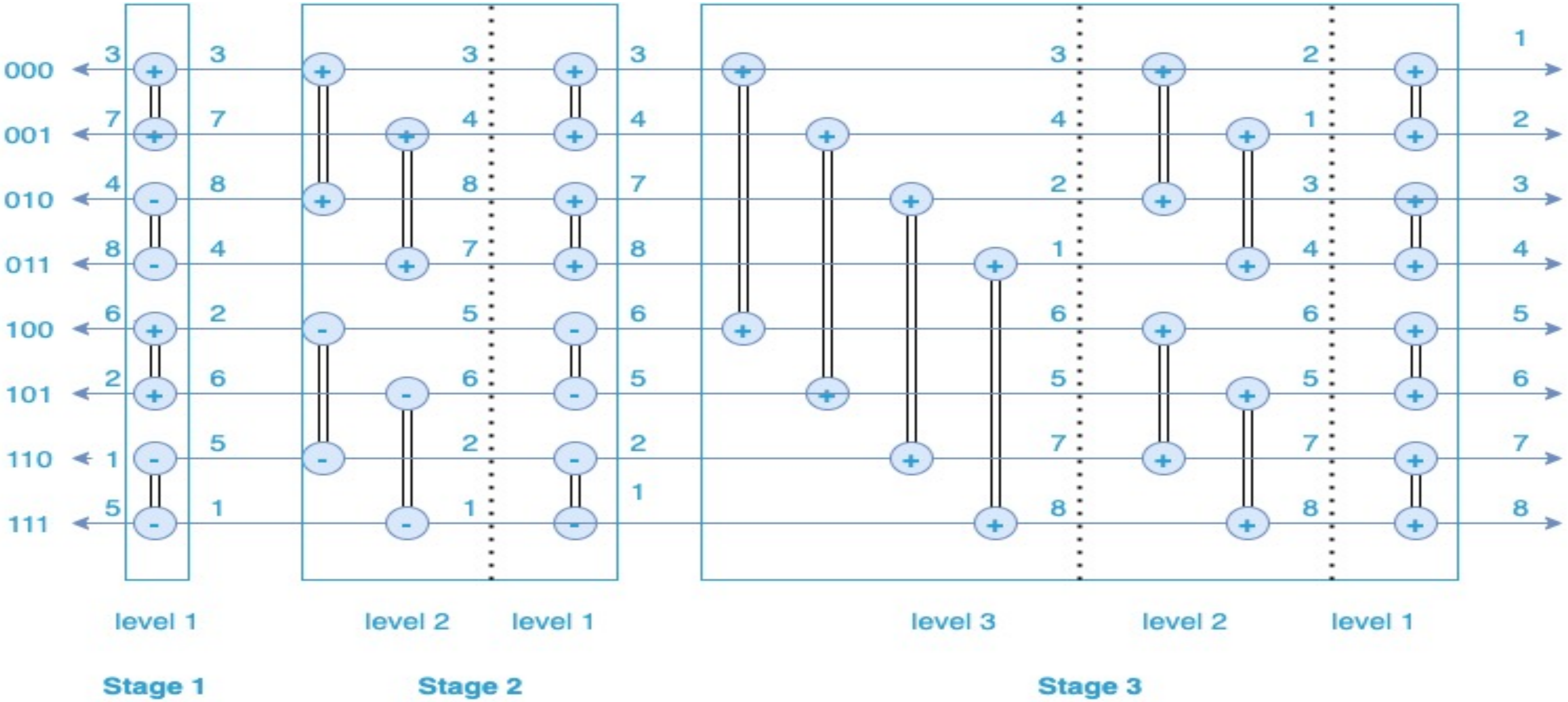
3 4 2 1 | 6 5 7 8

2 1 | 3 4 | 6 5 | 7 8

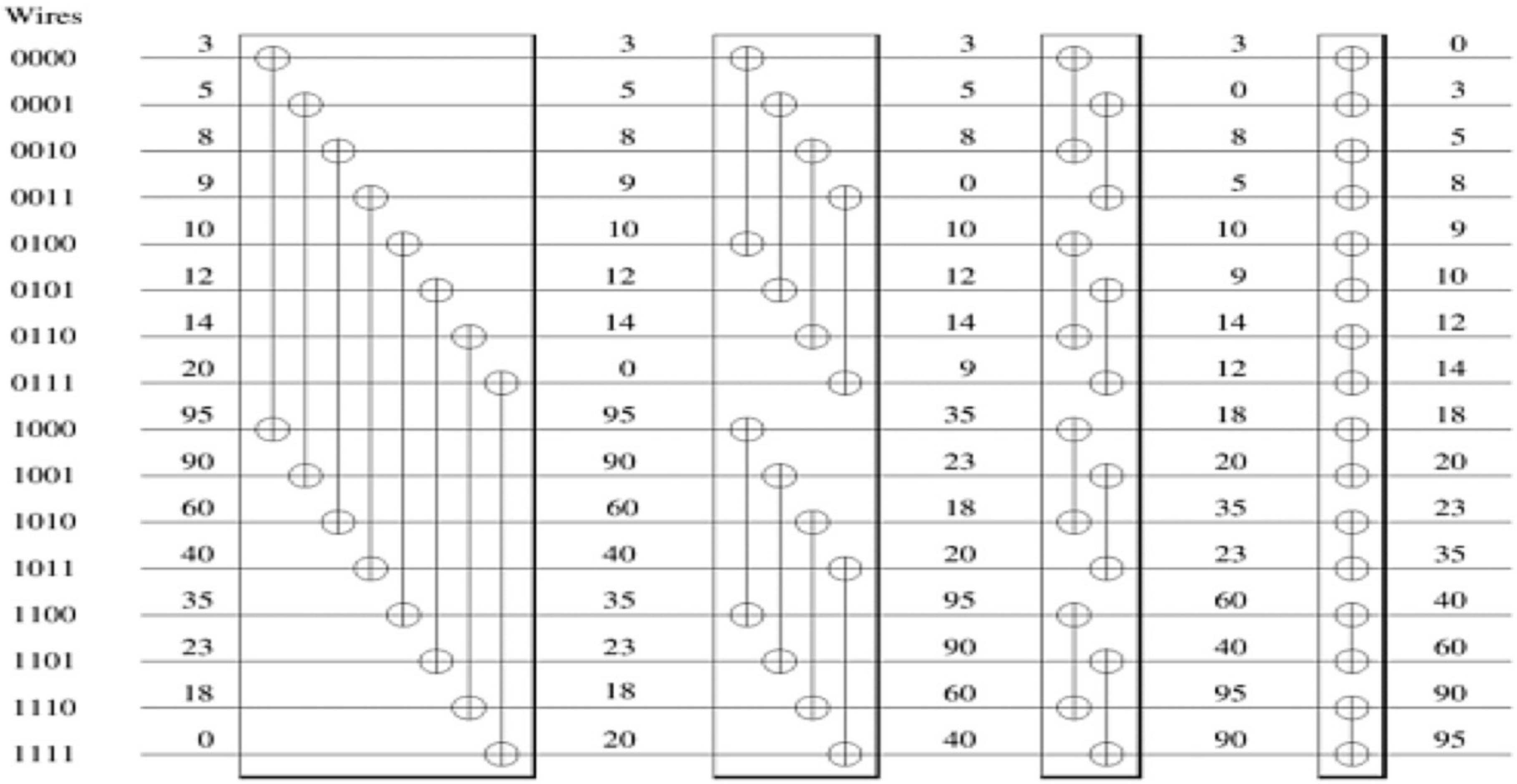
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

All elements in second sequence is greater than the first

Example: Sorting Network (Sort + Merge)



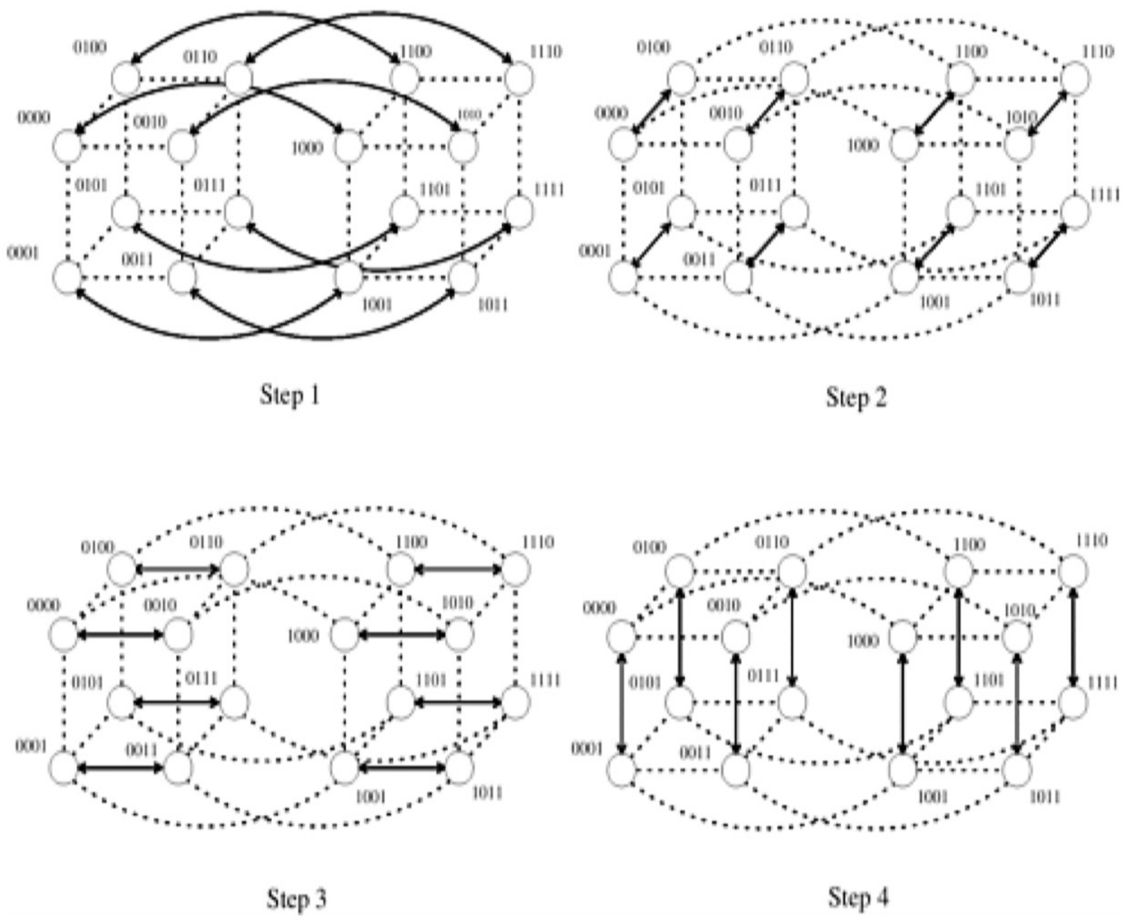
Example: 16 lines



*Introduction to Parallel Computing 2nd Edition, Ananth Grama

Example: 16 lines

Hypercube



*Introduction to Parallel Computing 2nd Edition, Ananth Grama

Algorithm

COMPARE SECTION

```
procedure BITONIC_SORT(label, d)
begin
    for i := 0 to d - 1 do
        for j := i downto 0 do
            if (i + 1) st bit of label != j th bit of label then
                comp_exchange_max(j);
            else
                comp_exchange_min(j);
end BITONIC_SORT
```

Complexity

- (1 + log n) (log n) / 2
- $T_p = \theta(n \log^2 n))$

i	j	0	1	0	0
1	0	comp_exchange_min(0);			
2	1	comp_exchange_max(1);			
2	0	comp_exchange_max(0);			
3	2	comp_exchange_max(2);			
3	1	comp_exchange_min(1);			
3	0	comp_exchange_min(0);			

*Introduction to Parallel Computing 2nd Edition, Ananth Grama

More on complexity…

- N/P Block of data per processor
- Fast sequential sort
  - Merge sort $\theta((n/p)\log(n/p))$
- Bitonic Merge
  - $\theta(\log 2\ p)$

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}^{\text{comparisons}} + \overbrace{\Theta\left(\frac{n}{p}\log^2 p\right)}^{\text{communication}}.$$

Results for 1 Billion keys

Key Observation: Inter-node parallelism give better performance than Intra-node parallelism.

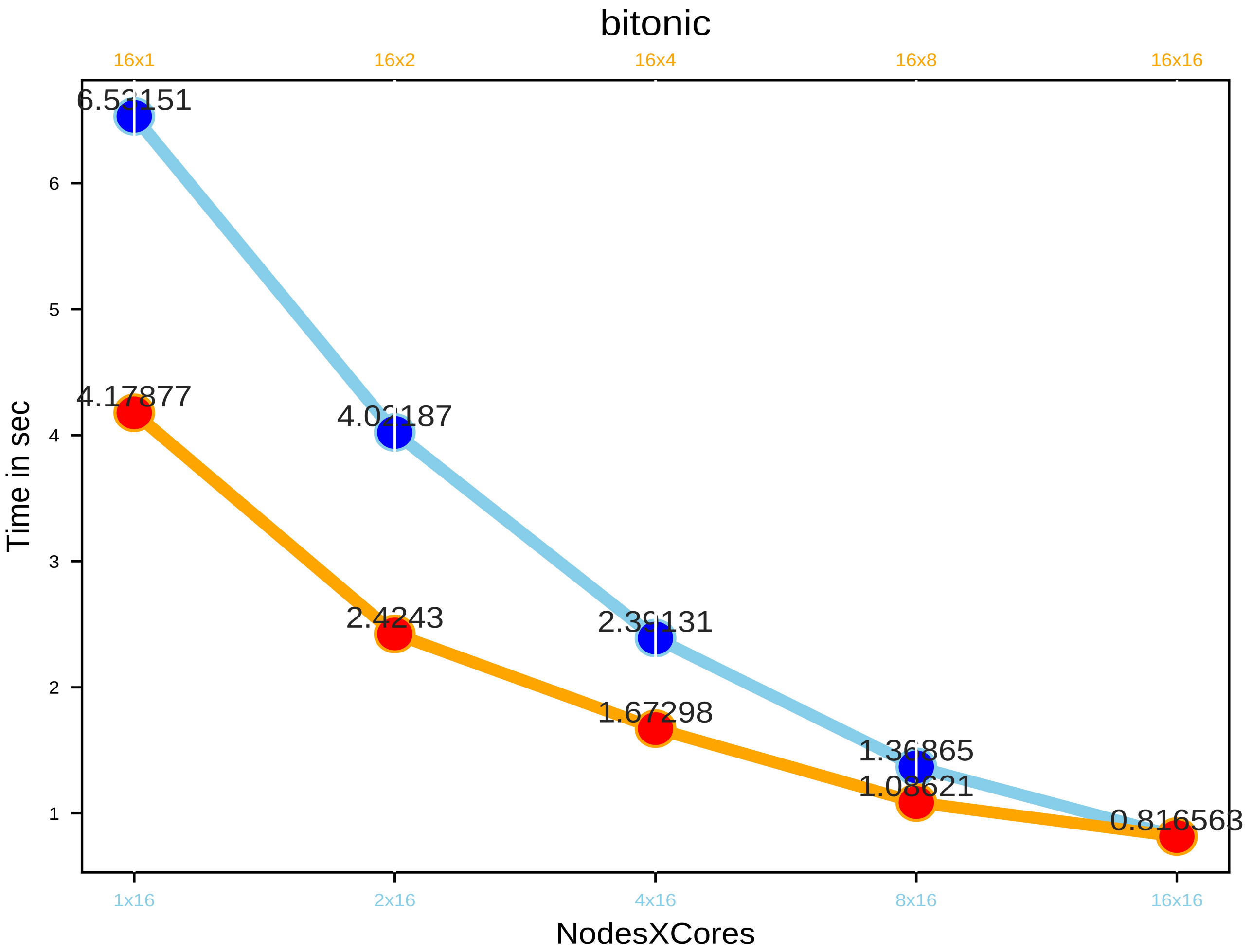
Relative speed-up:

$T_1 = 6.53$; $T_{16} = 0.8$

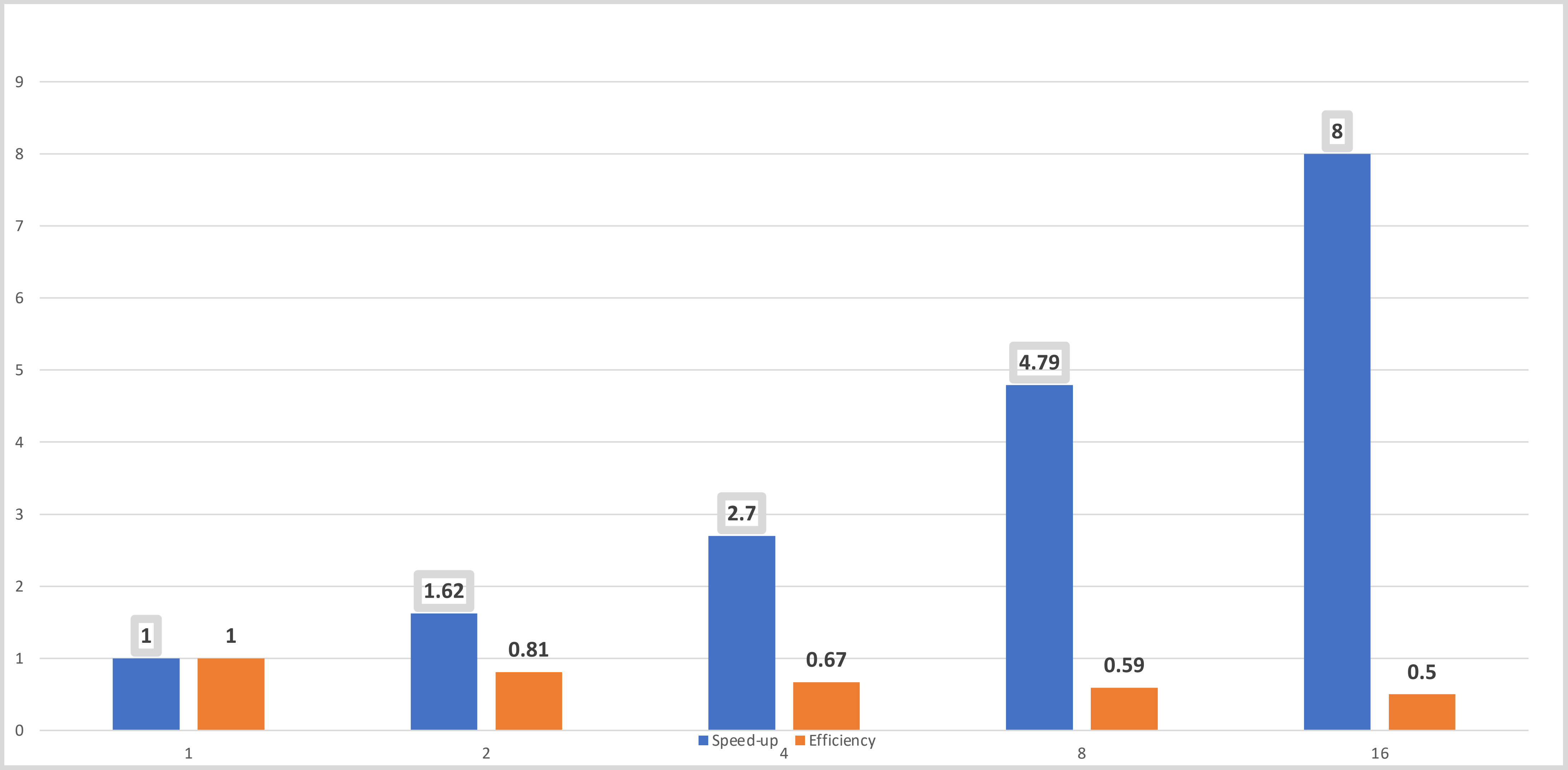
$\mathbf{S_p = 8}$

$\mathbf{E_p = 0.5}$

- Was able to sort 100 Billion keys on 512 ranks in 44.7524 sec
- Input Data size : 383G



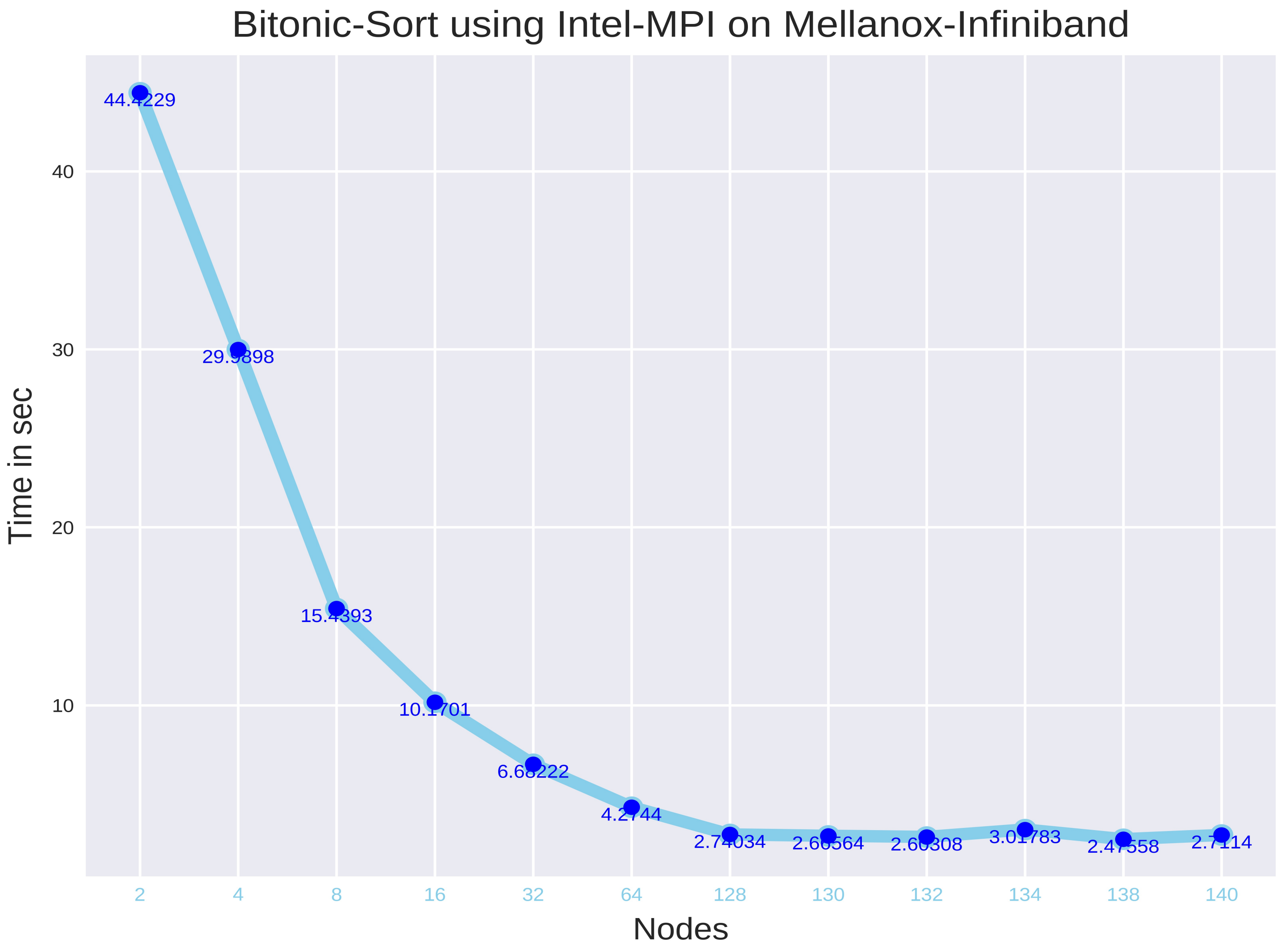
Speed-up and Efficiency

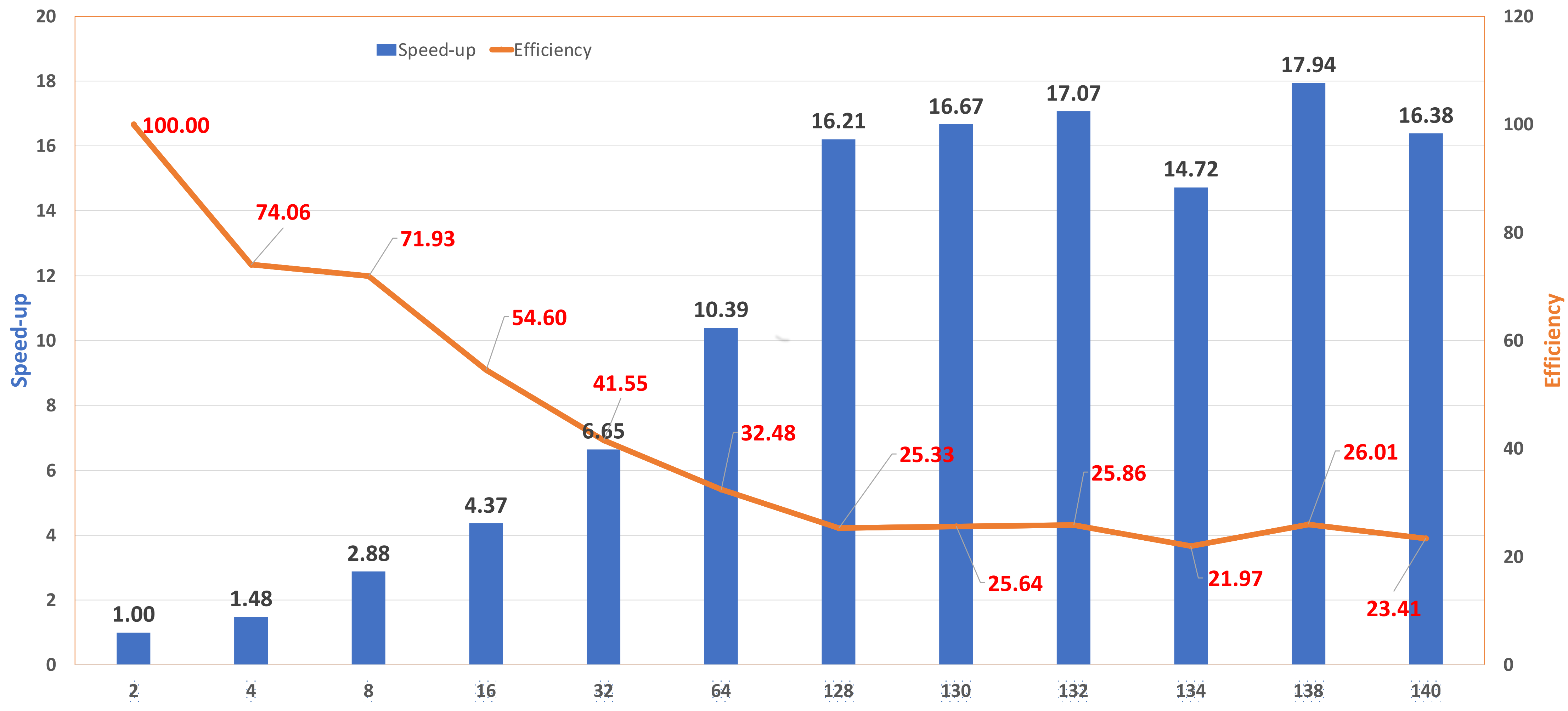


Results for 1 Billion keys

Observation:

- The code stops scaling after 128 Nodes.
- The execution time increases in some cases.
- However, It will be interesting to see the performance at 256 nodes.





Speed-up
Efficiency
20
18
16
14
12
10
8
6
4
2
0
120
100
80
60
40
20
0
1.00
1.48
2.88
4.37
6.65
10.39
16.21
16.67
17.07
14.72
17.94
16.38
100.00
74.06
71.93
54.60
41.55
32.48
25.33
25.64
25.86
21.97
26.01
23.41
2
4
8
16
32
64
128
130
132
134
138
140

Amdahl's Law

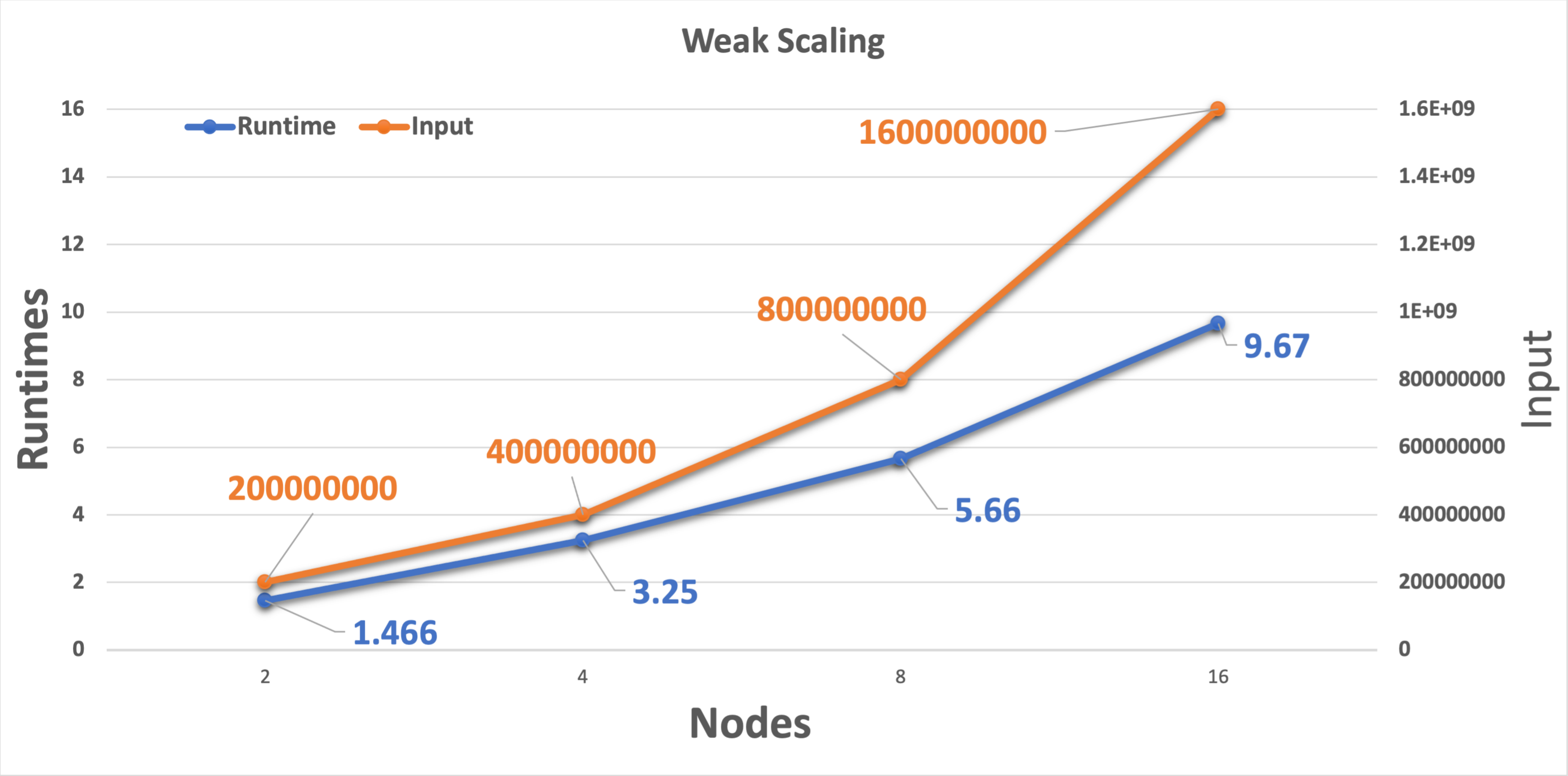
The Law focuses on strong scaling where input remains constant, and we increase the processors expecting the runtime to reduce in proportion to number of processors added maintaining reasonable efficiency.

$$S_p = \frac{1}{\beta + \frac{(1-\beta)}{p}}$$

where β is the serial part of the code which cannot be parallelized.

- In the above experiment: (S_p = 16, p = 140) Therefore, β comes to = 5.5% .
- So according the Amdahl's Law 5.5% of code will never be parallelized.
- For Input size of 1 Billion keys we are able to strongly scale upto 32 nodes. After that the efficiency decrease dramatically.

Gustafson's law



Gustafson's law

As we increase the processor, we are able to solve bigger and bigger problems thus, achieving weak scaling

$$S_p = p - \alpha (p - 1)$$

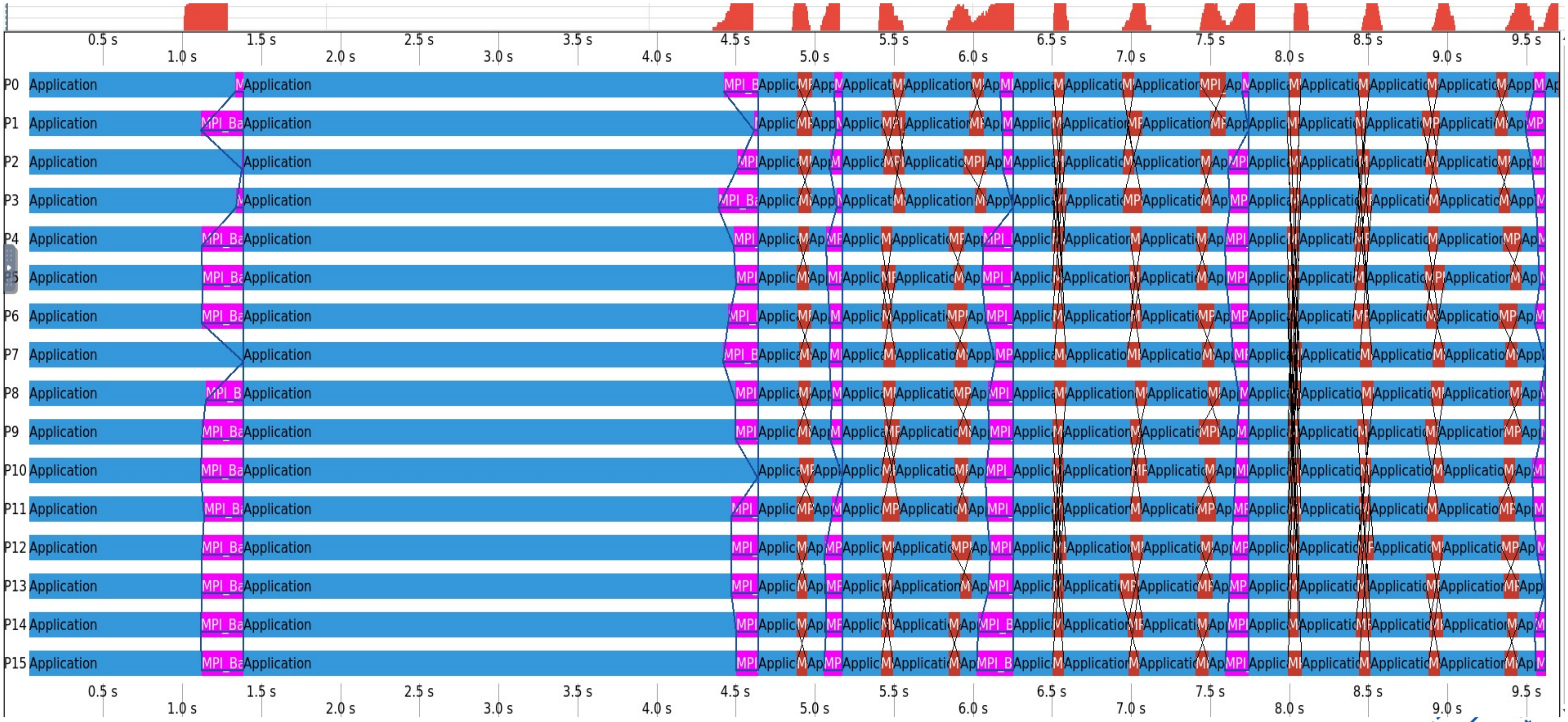
$$\alpha = \frac{T_{seq}}{T_{seq} + Tpar}$$

Substituting the values from above experiment : (T_{seq} = 1.4, T_{par} = 9.6, $\alpha = 0.127$)

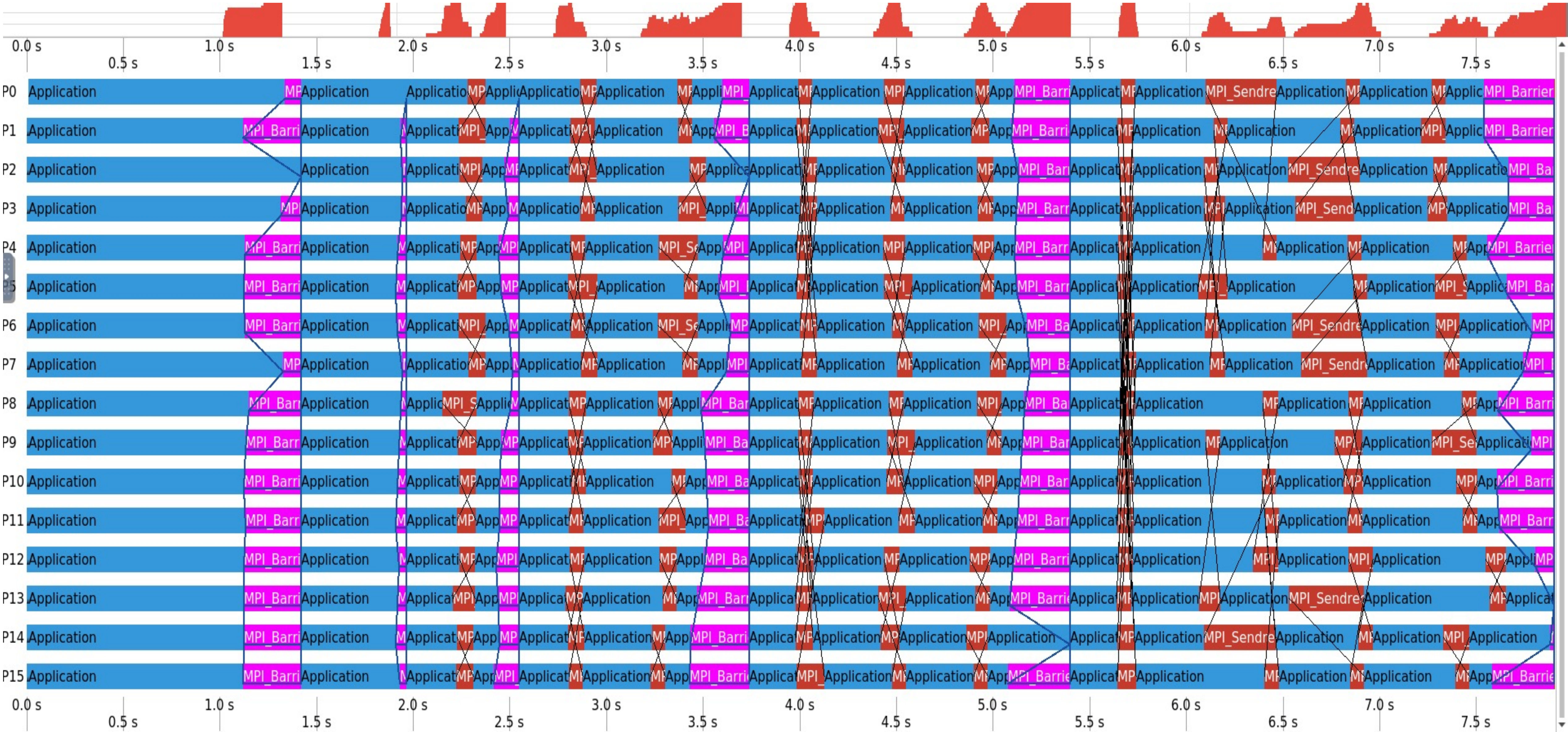
$$S_p = \mathbf{14.05}$$

Therefore, the algorithm is weakly scalable at 16 nodes as N:P are in ratio.

Running without TBB (Inter MPI Tracer)



Running with TBB (Inter MPI Tracer)



Slurm.sh

```bash
#!/bin/bash


# SBATCH --mem=64000

# SBATCH --exclusive
# SBATCH --constraint=IB

# SBATCH --job-name="702"

# SBATCH --partition=general-compute
# SBATCH --qos=general-compute
# SBATCH --account=zsayed

# SBATCH --output=~/panasas/logs/bitonic_%x_%j.stdout
# SBATCH --error=~/panasas/logs/bitonic_%x_%j.stderr

# SBATCH --time=00:10:00

# SBATCH --nodes=16
# SBATCH --ntasks-per-node=1

module load intel-oneapi-2021.3
module load intel-oneapi-mpi/2021.3.0
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

module load  intel-tbb/2019.3
source /util/academic/intel/19.3/compilers_and_libraries/linux/tbb/bin/tbbvars.


srun --mpi=pmi2  /user/zsayed/projects/bitonic-sort/bin/bitonic 1000000000

#mpirun -trace  /user/zsayed/projects/bitonic-sort/bin/bitonic  1000000000
```

Readme

````markdown
### Bitonic sort
```bash

./build.sh


mpirun -np 8 ./bin/bitonic 1024

mpirun -np 8 <tau_exec> ./bin/bitonic 1024

mpirun -trace ./bin/bitonic 1024

sbatch slurm.sh

```


- install tau pdt to instrument and profile

```bash

export TAU_TRACE=1; export TAU_PROFILE=1; export TAU_COMM_MATRIX=1;
export  TRACEDIR=./tau_trace; export PROFILEDIR=./tau_trace;

tau_treemerge.pl;

tau2slog2 tau.trc tau.edf -o tau.slog2;

jumpshot tau.slog2

```
````

Repo: https://gitlab.com/zain_s/bitonic-sort

References

- Introduction to Parallel Computing Solutions Manual on the Web
  Grama, Gupta, Karypis & Kumar
- R. Miller and L. Boxer, *Algorithms Sequential and Parallel: A Unified Approach*, Third Edition, Cengage Learning, Boston, Mass., 2013.

Thank you!