

HPC Exam Project

Hybrid MPI/OpenMP 5-Point Stencil:
Performance and Scalability Study

Giovanni Lucarelli

January 23, 2026



UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Introduction

Goals

1. **Parallelize** using hybrid (MPI, OpenMP) approach the stencil method for the 2d heat equation
2. Perform **scalability** study:
 - 2.1 Thread scaling
 - 2.2 Strong scaling
 - 2.3 Weak scaling

Algorithm

Heat equation (2d)

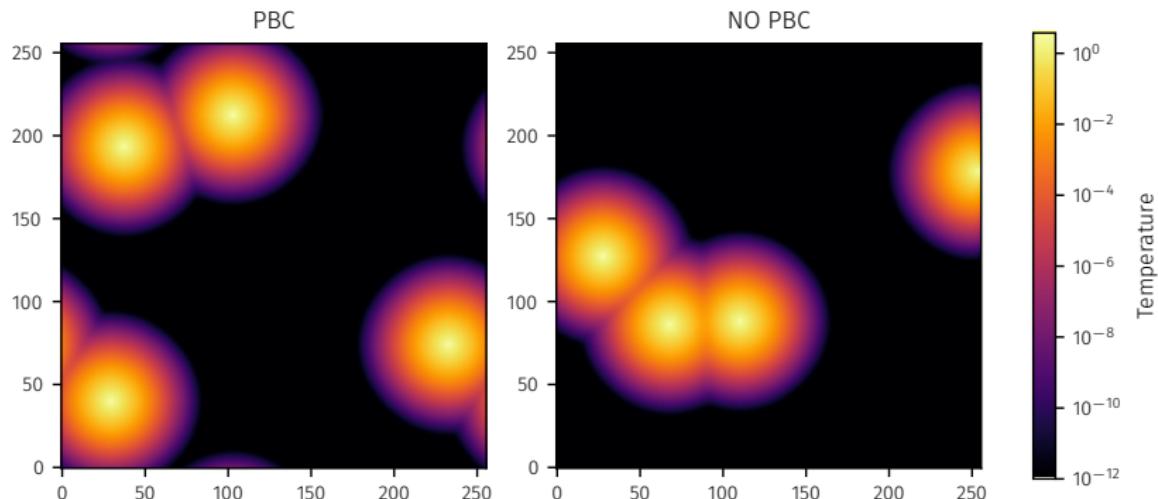
$$\partial_t u = \alpha(\partial_x^2 u + \partial_y^2 u)$$

Finite difference integration (5-point stencil method)

$$u_{i,j}^{(t+1)} = (1 - 4\alpha)u_{i,j}^{(t)} + \alpha \sum_{\langle i,j \rangle} u_{i,j}^{(t)}$$

$$\begin{aligned} x \in [0, L_x] &\rightarrow i \in \{1, \dots, N_x - 1\} \\ y \in [0, L_y] &\rightarrow j \in \{1, \dots, N_y - 1\} \end{aligned}$$

Code Correctness



Build Configuration

```
1 CC      = mpicc
2 CFLAGS = -O3 -Wall -fopenmp -march=native
3 TARGET = stencil_mpi
4
5 ENABLE_OUTPUT ?= 0
```

Parallelization: shared memory

Implementation

```
1 #pragma omp parallel for schedule(static)
2 for (uint j = 1; j <= ysize; j++){
3     for ( uint i = 1; i <= xsize; i++){
4
5         // update rule
6
7     }
8 }
```

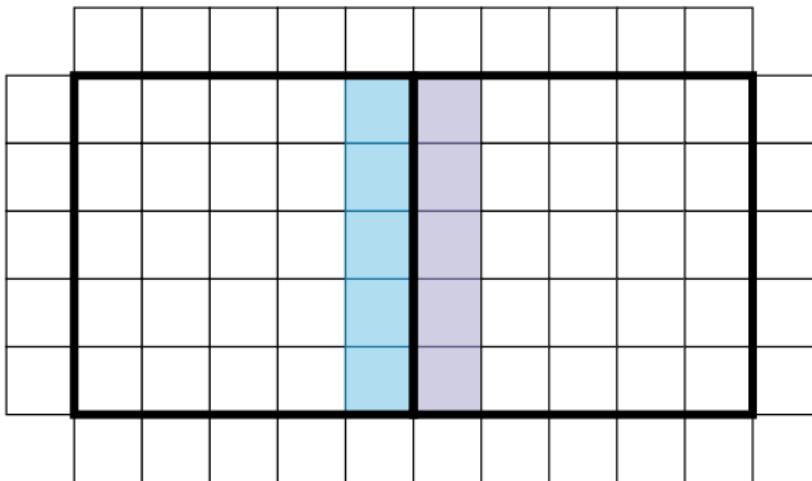
Thread placement

```
1 export OMP_PLACES=cores
2 export OMP_PROC_BIND=close
```

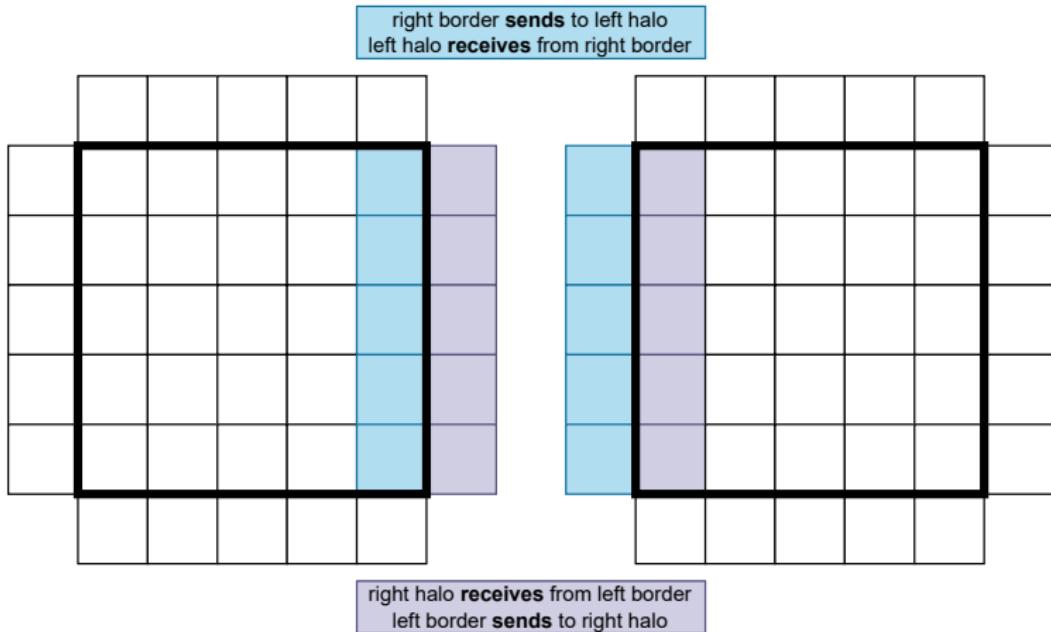
Touch-by-all policy

```
1 int memory_allocate ( ... ){
2
3     planes_ptr[OLD].data = (double*) malloc(frame_elems * sizeof
4                                     (double));
5     planes_ptr[NEW].data = (double*) malloc(frame_elems * sizeof
6                                     (double));
7
8     #pragma omp parallel for schedule(static)
9     for (int j = 0; j < Ny + 2; ++j){
10         for (int i = 0; i < Nx + 2; ++i) {
11             size_t idx = (size_t)j * (Nx + 2) + i;
12             planes_ptr[OLD].data[idx] = 0.0;
13             planes_ptr[NEW].data[idx] = 0.0;
14         }
15     }
16 }
```

Parallelization: distributed memory



Parallelization: distributed memory



Parallelization: distributed memory

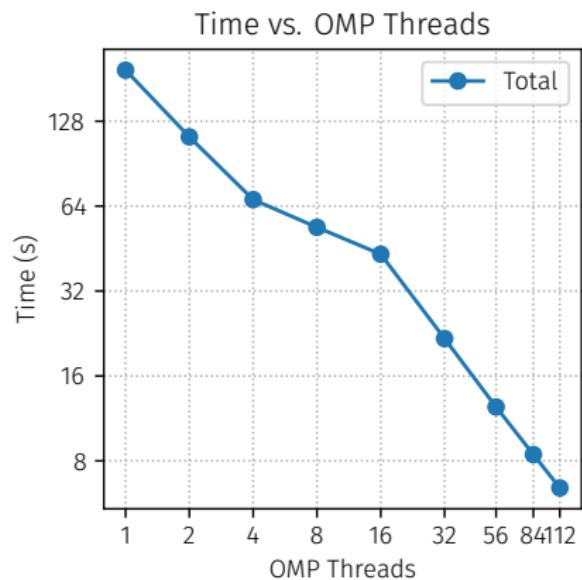
For each task:

```
1 // pack buffers
2
3 MPI_Irecv(...);
4
5 MPI_Isend(...);
6
7 update_internal();
8
9 MPI_Waitall();
10
11 // unpack buffers
12
13 update_border();
```

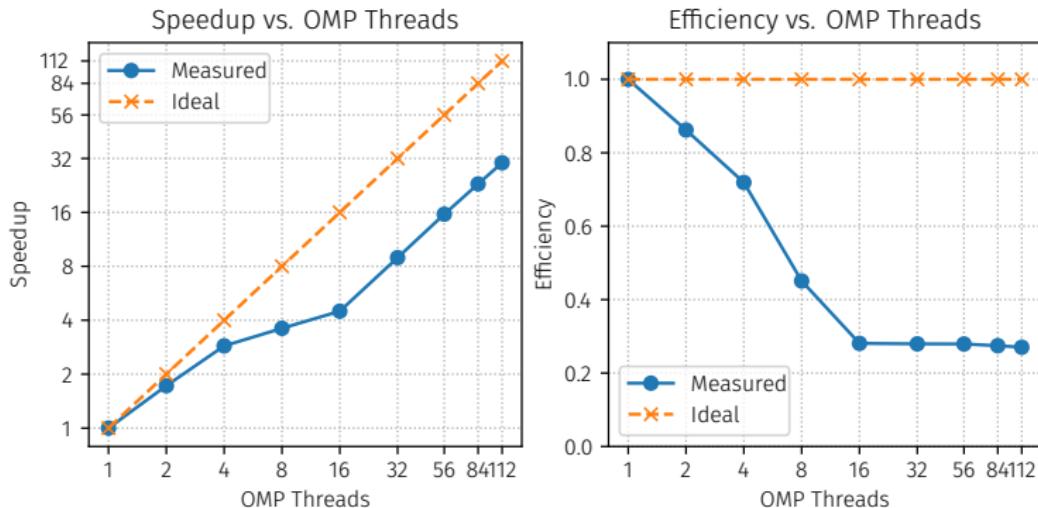
Results

Thread Scaling

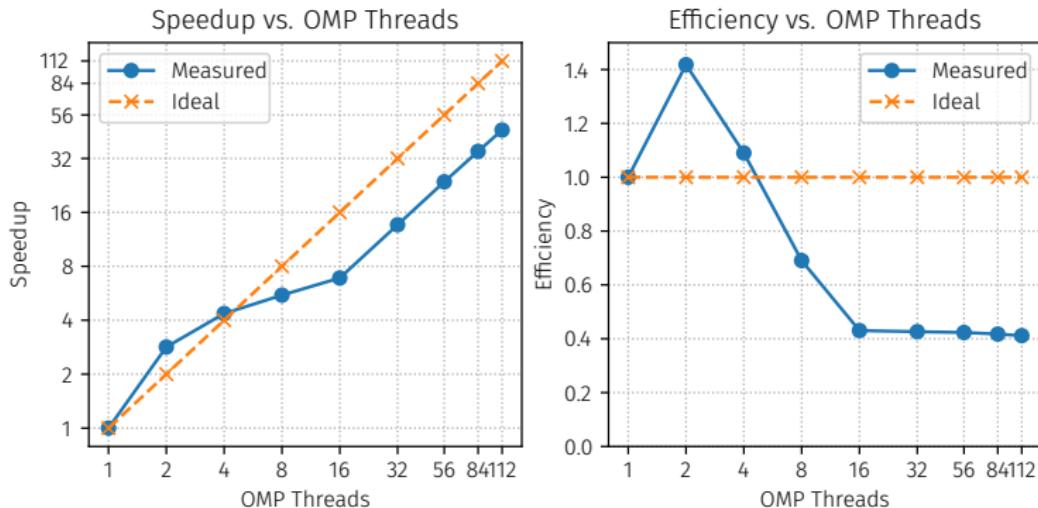
```
1 GRID_SIZE_X=16384  
2 GRID_SIZE_Y=16384  
3 N_STEPS=500  
4  
5 NODES=1  
6 N_TASKS_PER_NODE=1  
7 THREADS="1 2 4 8  
16 32 56 84  
112"
```



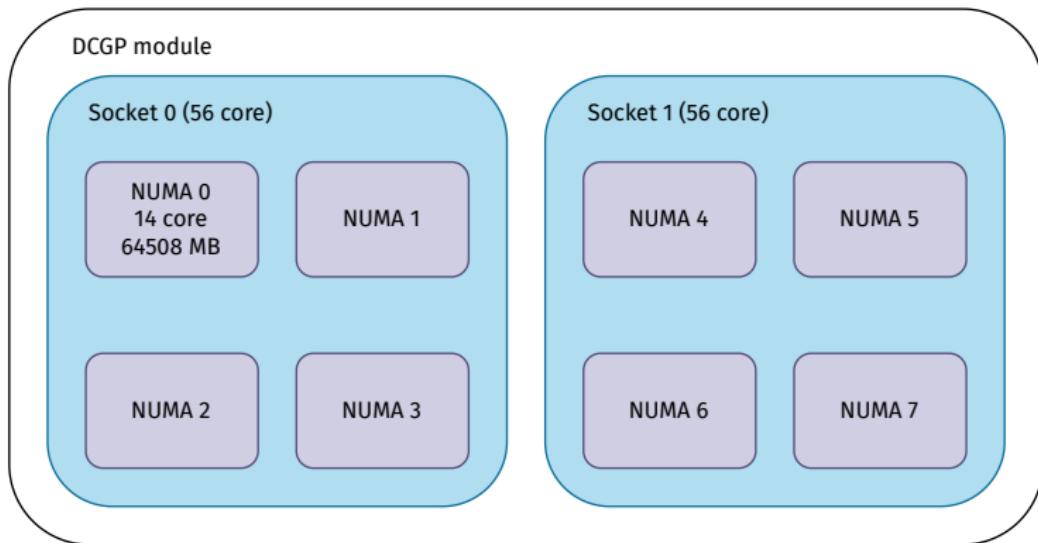
Thread Scaling: touch-first policy



Thread Scaling: touch-by-all policy



Node Architecture

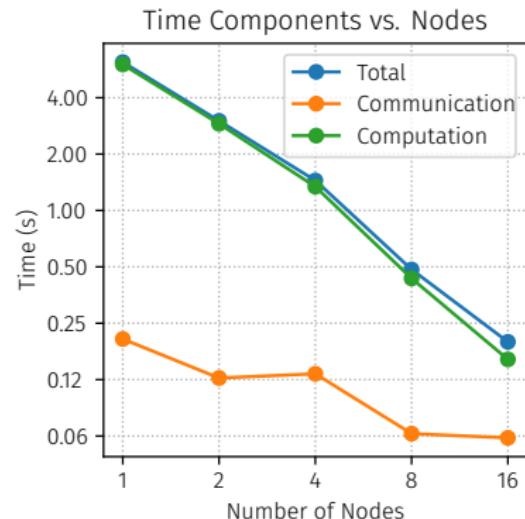


Node Architecture: distance matrix

```
1 [glucarel@lrdn4293 HPC-leonardo]$ numactl --hardware
2     available: 8 nodes (0-7)
3
4     node   0    1    2    3    4    5    6    7
5       0:  10   12   12   12   21   21   21   21
6       1:  12   10   12   12   21   21   21   21
7       2:  12   12   10   12   21   21   21   21
8       3:  12   12   12   10   21   21   21   21
9       4:  21   21   21   21   10   12   12   12
10      5:  21   21   21   21   12   10   12   12
11      6:  21   21   21   21   12   12   10   12
12      7:  21   21   21   21   12   12   12   10
```

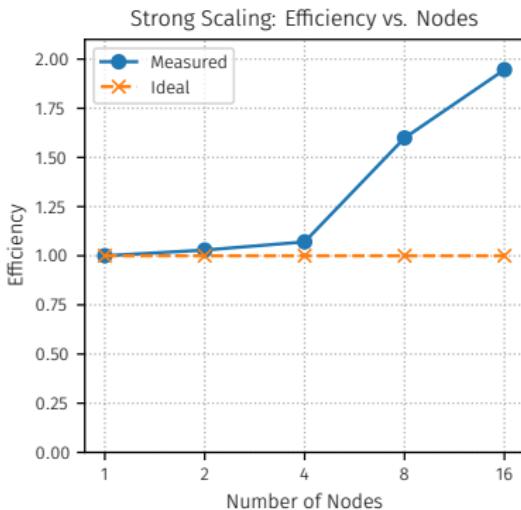
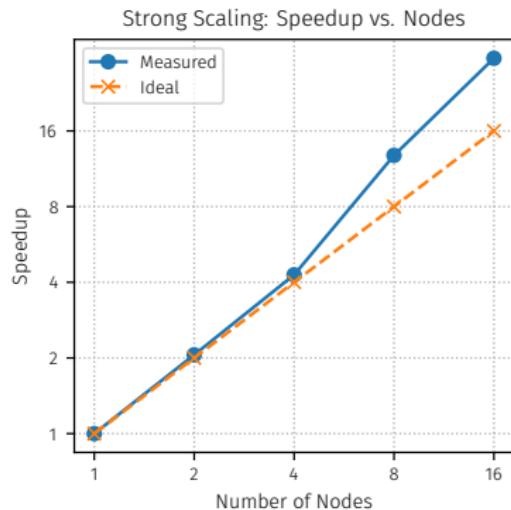
Strong Scaling (1/4)

```
1 GRID_SIZE_X=16384  
2 GRID_SIZE_Y=16384  
3 N_STEPS=500  
4  
5 OMP_THREADS=14  
6 N_TASKS_PER_NODE=8  
7  
8 NODES="1 2 4 8 16"
```



Computation \gg Communication !

Strong Scaling (1/4)



Strong Scaling (1/4): Analysis

- 8 MPI tasks per node → one per **NUMA region**
- larger number of nodes → larger number of tasks
(8 → 128) → smaller (local) grid size
- for 16 nodes (128 tasks as 16×8), size $N_x = N_y = 2^{14}$

$$mem_{128} = \frac{2^{14}}{16} \times \frac{2^{14}}{8} \times 16B \approx 33.5\text{MB}$$

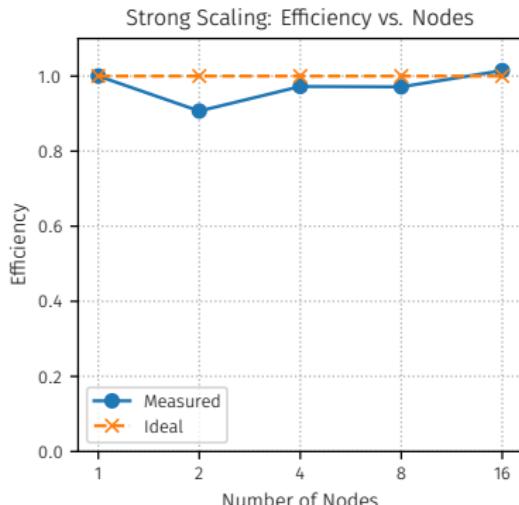
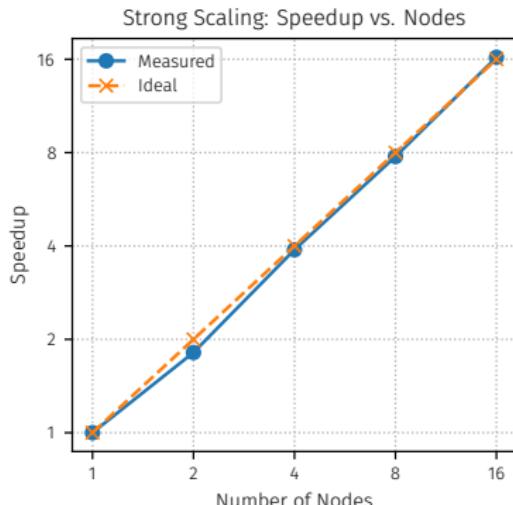
```
1 [glucarel@lrdn4293 HPC-leonardo]$ lscpu | egrep 'L1d|L1i|L2|L3'
2     L1d cache:          48K
3     L1i cache:          32K
4     L2 cache:         2048K
5     L3 cache:        107520K
```

Superlinearity from cache effects exploitation!¹

¹If the grid is small enough

Strong Scaling (2/4)

```
1 GRID_SIZE_X=16384  
2 OMP_THREADS=112  
3 N_TASKS_PER_NODE=1
```



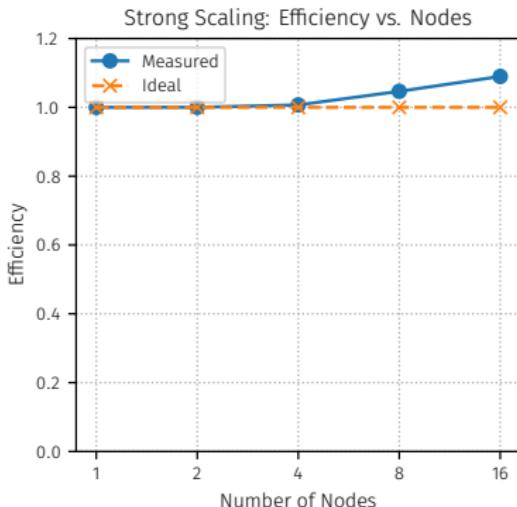
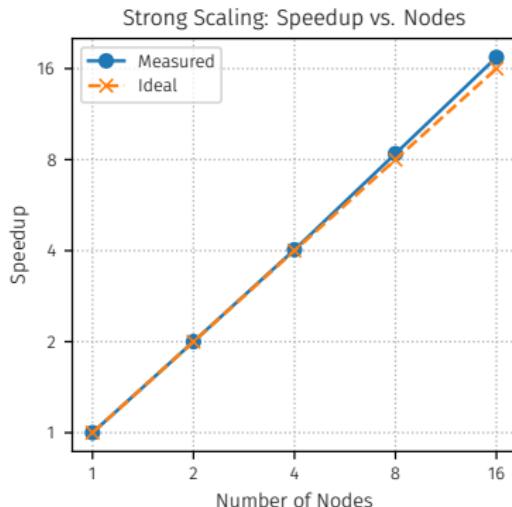
Strong Scaling (2/4): Analysis

- 1 MPI tasks per node → lower number of tasks
(1 → 16)
- for 16 nodes (16 tasks as 4×4), size $N_x = X_y = 2^{14}$

$$mem_{16} = \frac{2^{14}}{4} \times \frac{2^{14}}{4} \times 16B \approx 268\text{MB}$$

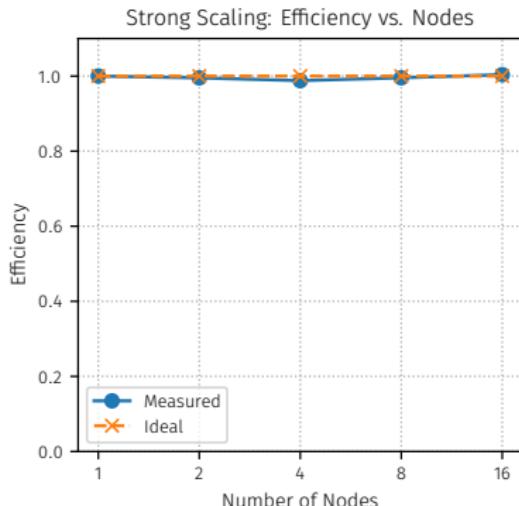
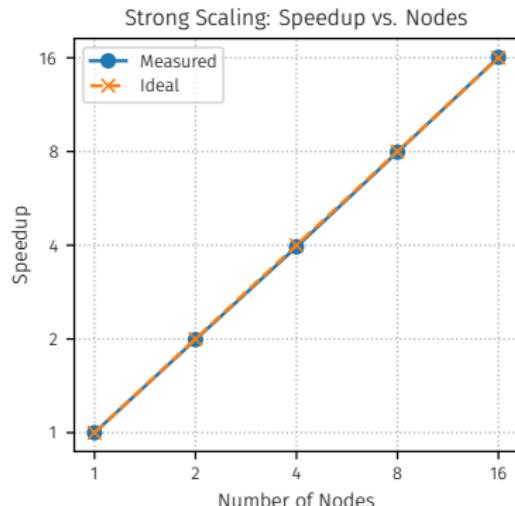
Strong Scaling (3/4)

```
1 GRID_SIZE_X=32768  
2 OMP_THREADS=14  
3 N_TASKS_PER_NODE=8
```



Strong Scaling (4/4)

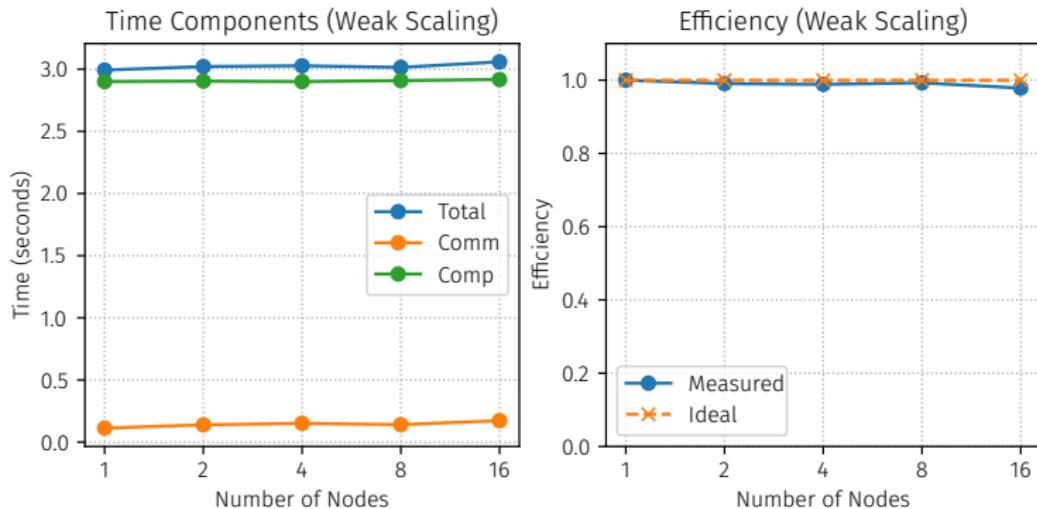
```
1 GRID_SIZE_X=65536  
2 OMP_THREADS=14  
3 N_TASKS_PER_NODE=8
```



Weak Scaling

```
1      LOCAL_X=4096
2      LOCAL_Y=4096
3      OMP_THREADS=14
4      TASKS_PER_NODE=8
5
6      for NODES in "1 2 4 8 16"; do
7          TOTAL_TASKS=$(( NODES * TASKS_PER_NODE ))
8
9          case "${TOTAL_TASKS}" in
10             8)    PX=4;   PY=2   ;;    # 1 node (8 ranks)
11             ...
12         esac
13
14         GRID_SIZE_X=$(( LOCAL_X * PX ))
15         GRID_SIZE_Y=$(( LOCAL_Y * PY ))
16         ...
17     done
```

Weak Scaling



Conclusion

Conclusion

Key findings

- MPI communication time is small compared to the stencil update
- OpenMP efficiency decreases at high thread counts (likely memory-bandwidth / NUMA limited)

Possible improvements

- Enable/verify SIMD vectorization of the stencil kernel
- Use MPI derived datatypes or optimized packing/unpacking
- Explore alternative schemes (e.g., higher-order stencils) and their performance/accuracy trade-offs

Thank You!