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)

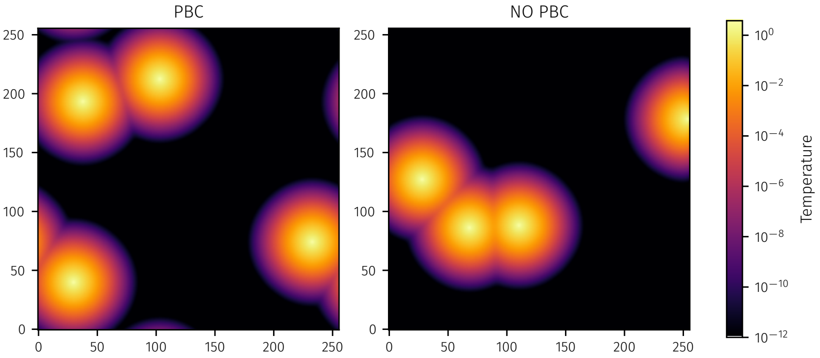
$$\partial_t u = \alpha(\partial_x^2 u + \partial_y^2 u)$$

Finite difference integration (5-point stencil method)

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

$$x \in [0, L_x] \to i \in \{1, \dots, N_x - 1\}$$
$$y \in [0, L_y] \to j \in \{1, \dots, N_y - 1\}$$

Code Correctness



Build Configuration

```
CC       = mpicc
CFLAGS   = -O3 -Wall -fopenmp -march=native
TARGET   = stencil_mpi

ENABLE_OUTPUT ?= 0
```

Parallelization: shared memory

Implementation

```
#pragma omp parallel for schedule(static)
for (uint j = 1; j <= ysize; j++){
    for ( uint i = 1; i <= xsize; i++){

        // update rule

    }
}
```

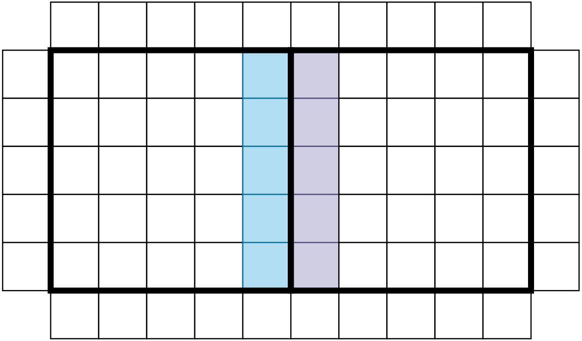
Thread placement

```
export OMP_PLACES=cores
export OMP_PROC_BIND=close
```

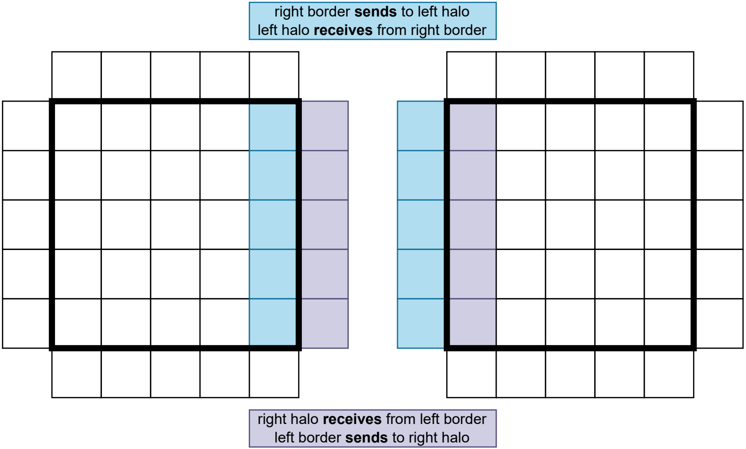
First-touch policy

```c
int memory_allocate ( ... ){

    planes_ptr[OLD].data = (double*) malloc(frame_elems * sizeof
        (double));
    planes_ptr[NEW].data = (double*) malloc(frame_elems * sizeof
        (double));

    #pragma omp parallel for schedule(static)
    for (int j = 0; j < Ny + 2; ++j){
        for (int i = 0; i < Nx + 2; ++i) {
            size_t idx = (size_t)j * (Nx + 2) + i;
            planes_ptr[OLD].data[idx] = 0.0;
            planes_ptr[NEW].data[idx] = 0.0;
        }
    }
}
```

Parallelization: distributed memory



Parallelization: distributed memory



Parallelization: distributed memory

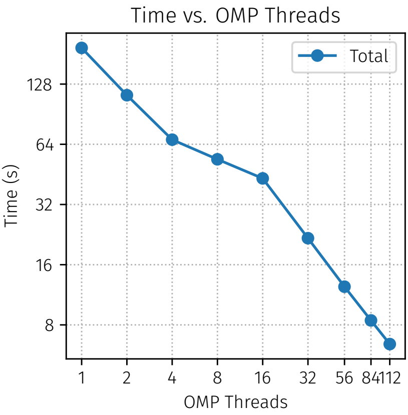
For each task:

```
// pack buffers

MPI_Irecv(...);

MPI_Isend(...);

update_internal();

MPI_Waitall();

// unpack buffers

update_border();
```

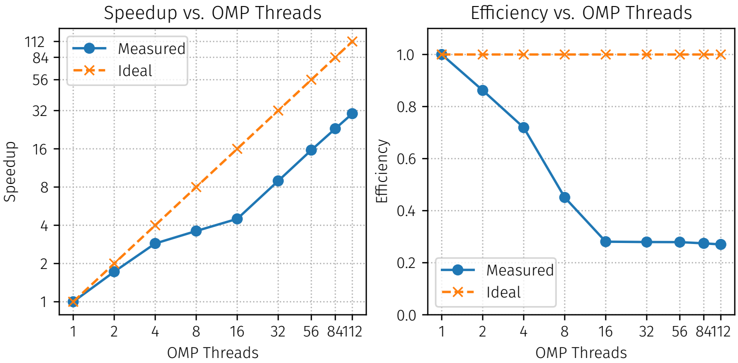
Results

Thread Scaling

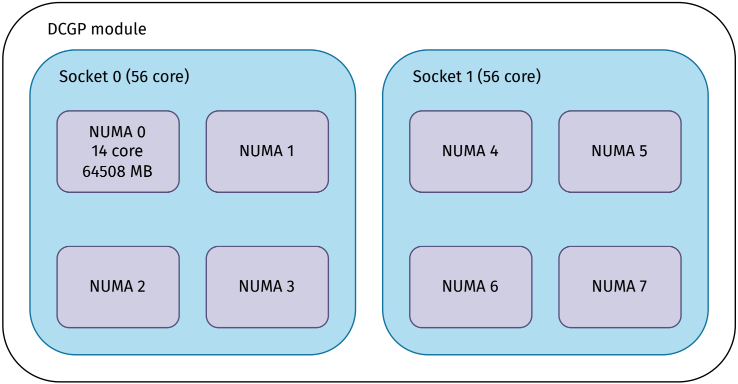
```
GRID_SIZE_X=16384
GRID_SIZE_Y=16384
N_STEPS=500

NODES=1
N_TASKS_PER_NODE=1
THREADS="1 2 4 8 16 32 56 84 112"
```



Thread Scaling



Node Architecture

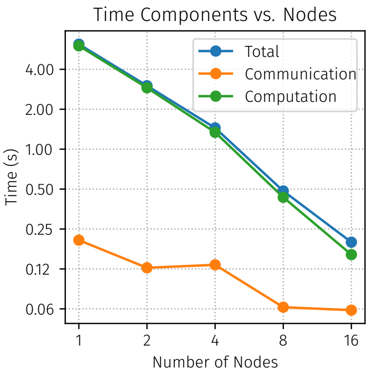


Node Architecture: distance matrix

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

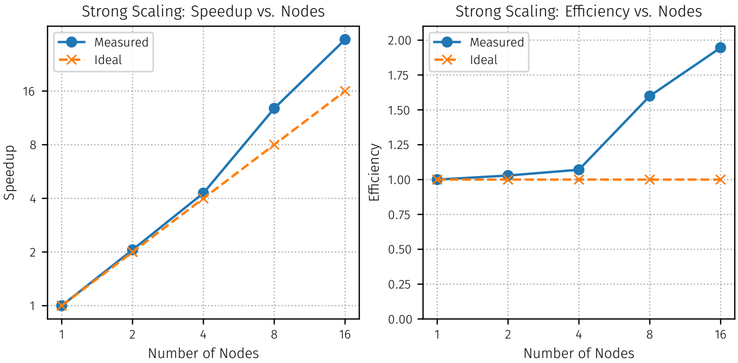
Strong Scaling (1/4)

```
GRID_SIZE_X=16384
GRID_SIZE_Y=16384
N_STEPS=500

OMP_THREADS=14
N_TASKS_PER_NODE=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 ($8 \to 128$) → smaller (local) grid size
- for 16 nodes (128 tasks as 16×8), size $N_x = N_y = 2^{14}$

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

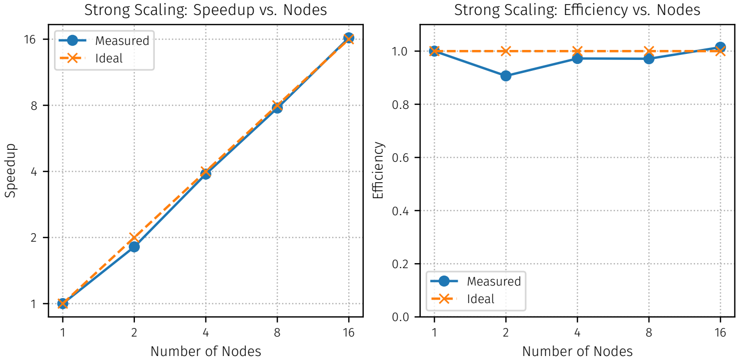
```
[glucarel@lrdn4293 HPC-leonardo]$ lscpu | egrep 'L1d|L1i|L2|L3'
    L1d cache:                   48K
    L1i cache:                   32K
    L2 cache:                    2048K
    L3 cache:                    107520K
```

Superlinearity from cache effects expoitation!?[1]

[1] If the grid is small enough

Strong Scaling (2/4)

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



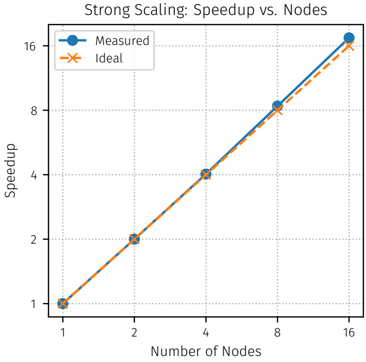
Strong Scaling (2/4): Analysis

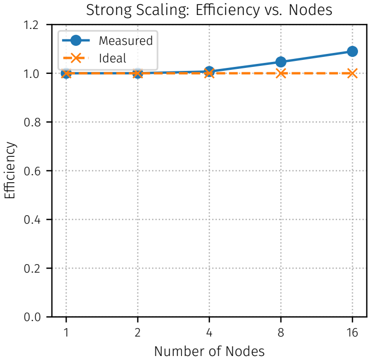
- 1 MPI tasks per node $\rightarrow$ lower number of tasks $(1 \rightarrow 16)$
- for 16 nodes (16 tasks as 4×4), size $N_x = X_y = 2^{14}$

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

Strong Scaling (3/4)

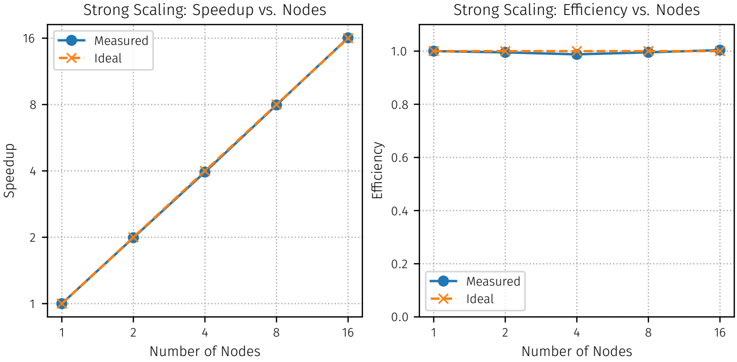
```
GRID_SIZE_X=32768
OMP_THREADS=14
N_TASKS_PER_NODE=8
```





Strong Scaling (4/4)

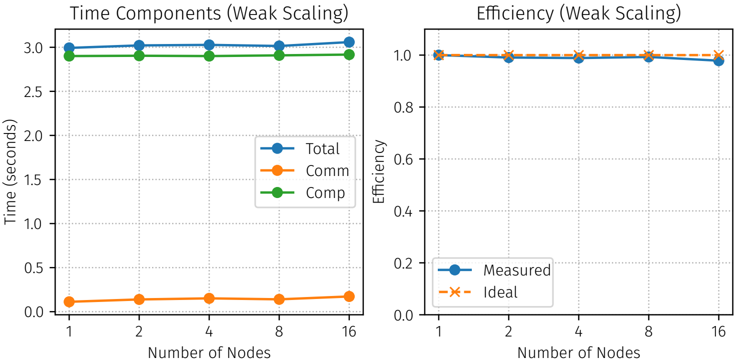
```
GRID_SIZE_X=65536
OMP_THREADS=14
N_TASKS_PER_NODE=8
```



Weak Scaling

```bash
LOCAL_X=4096
LOCAL_Y=4096
OMP_THREADS=14
TASKS_PER_NODE=8

for NODES in "1 2 4 8 16"; do
    TOTAL_TASKS=$(( NODES * TASKS_PER_NODE ))

    case "${TOTAL_TASKS}" in
    8)    PX=4;   PY=2  ;;    # 1 node  (8 ranks)
    ...
    esac

    GRID_SIZE_X=$(( LOCAL_X * PX ))
    GRID_SIZE_Y=$(( LOCAL_Y * PY ))
    ...
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!