HPC Exam Project

Scaling Study of the Stencil Method

Giovanni Lucarelli

October 20, 2025

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Introduction

Goals

1. **Optimize** the stencil method for the 2d heat equation
2. **Parallelize** using hybrid approach
3. Perform **scalability** study:
    3.1 Thread scaling
    3.2 Strong scaling
    3.3 Weak scaling

Algorithm

Heat equation (2d)

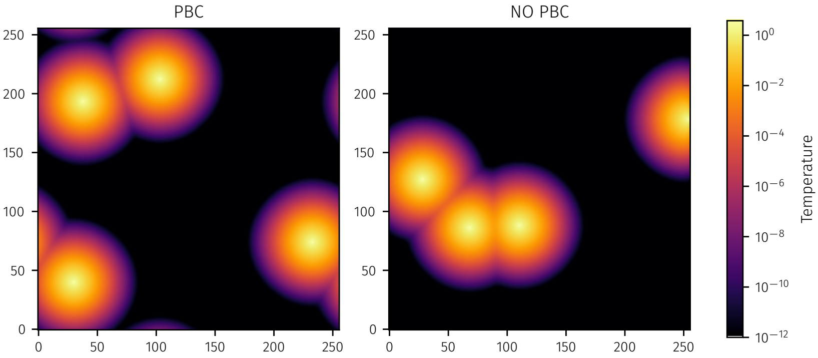
$$\partial_t u = \alpha(\partial_x^2 u + \partial_y^2 u)$$

Finite difference integration

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

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

Code Correctness



Optimization

- Compiler flags:
  `-O3 -Wall -march=native`
- Preprocessor directive:
  `#pragma GCC unroll`

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

    }
}
```

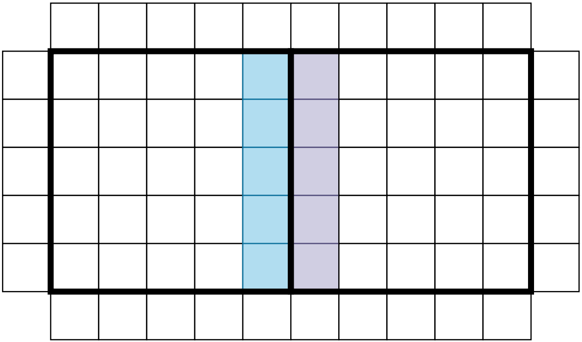
Thread placement and affinity

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

First-touch

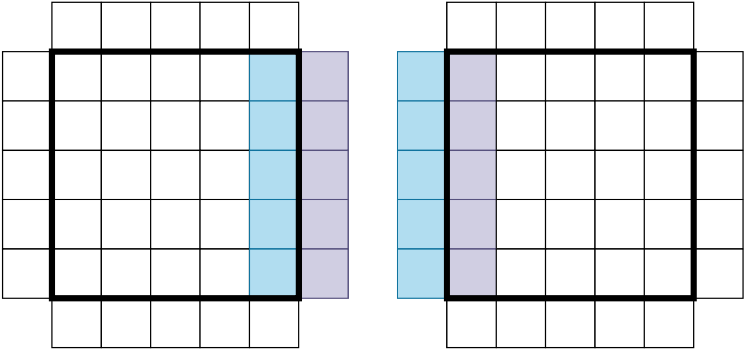
```c
int memory_allocate ( ... ){

    #pragma omp parallel for collapse(2) 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

right border **sends** to left halo
left halo **receives** from right border



right halo **receives** from left border
left border **sends** to right halo

Parallelization: distributed memory

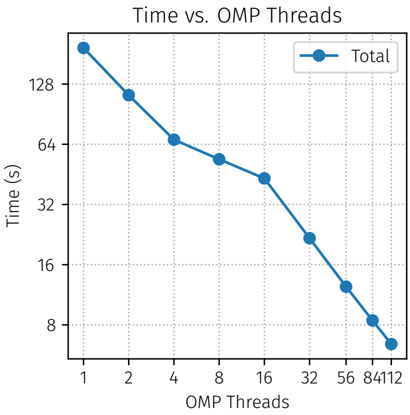
For each task:

```
// pack buffers

MPI_Irecv(...);

MPI_Isend(...);

update_internal();

MPI_Waitall();

// unpack buffers

update_border();
```

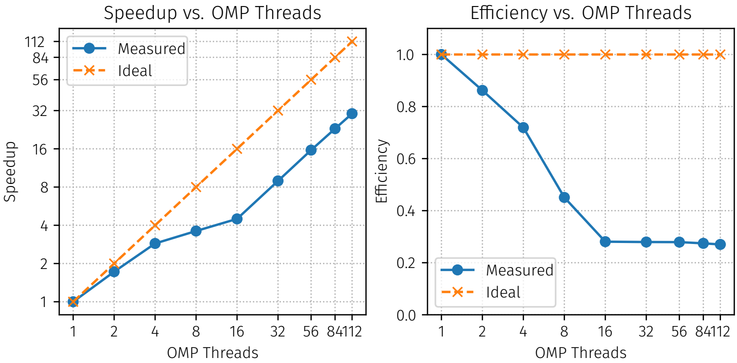
Results

Thread Scaling

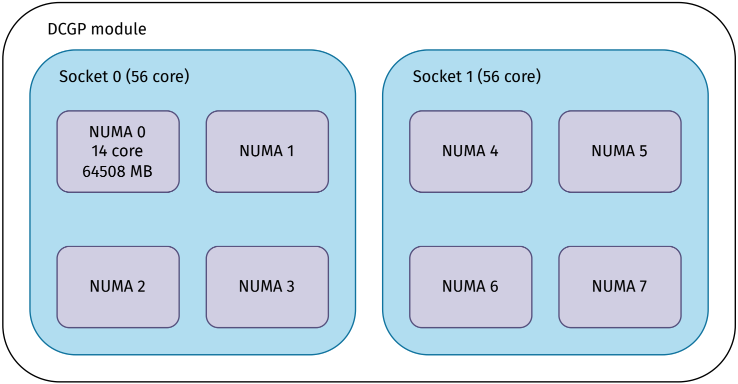
```
GRID_SIZE_X=16384
GRID_SIZE_Y=16384
N_STEPS=500

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



Thread Scaling



Node Architecture

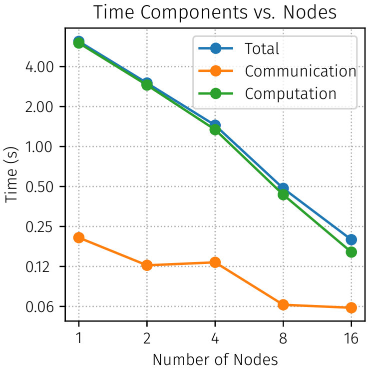


Node Architecture: distance matrix

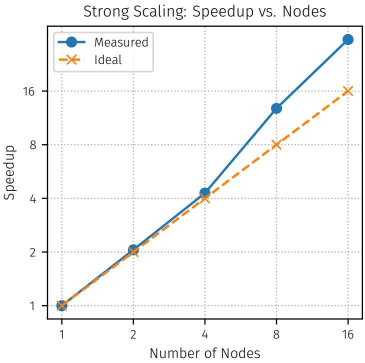
```
[glucarel@lrdn4293 HPC-leonardo]$ numactl --hardware
    available: 8 nodes (0-7)
      node   0   1   2   3   4   5   6   7
        0:  10  12  12  12  21  21  21  21
        1:  12  10  12  12  21  21  21  21
        2:  12  12  10  12  21  21  21  21
        3:  12  12  12  10  21  21  21  21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

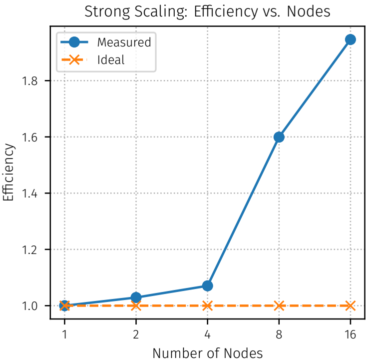
Strong Scaling (1/2)

```
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"
```



Strong Scaling (1/2)





Strong Scaling (1/2): Analysis

- 8 MPI tasks per node $\rightarrow$ one per **NUMA region**
- smaller grid as nodes increase ($8 \rightarrow 128$)
- for nodes=16 (grid $N = 2^{14}$, 128 tasks = 16 x 8)

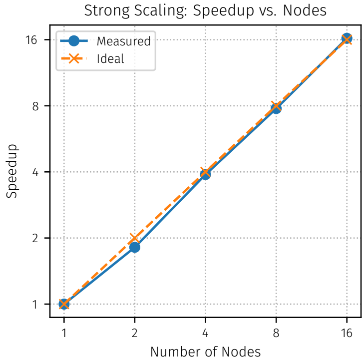
$$mem_{128} = 2^{10} \times 2^{11} \times 16\text{B} = 2^{25}\text{B} \approx 33.5\text{MB}$$

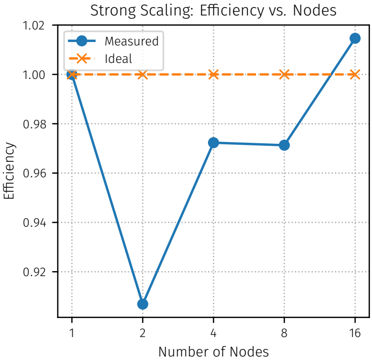
```
[glucarel@lrdn4293 HPC-leonardo]$ lscpu | egrep 'L1d|L1i|L2|L3'
    L1d cache:              48K
    L1i cache:              32K
    L2 cache:               2048K
    L3 cache:               107520K
```

Superlinearity comes from better cache effects expoitation!

Strong Scaling (2/2)

```
OMP_THREADS=112
N_TASKS_PER_NODE=1
```





Strong Scaling (2/2): Analysis

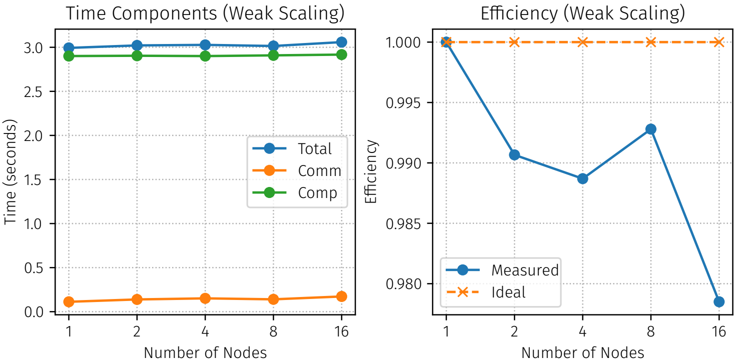
- 1 MPI tasks per node
- smaller grid as nodes increase ($1 \rightarrow 16$)
- for nodes=16 (grid $N = 2^{14}$, 16 tasks = 4 x 4)

$$mem_{16} = \frac{2^{14}}{4} \times \frac{2^{14}}{4} \times 16\text{B} = 2^{28}\text{B} \approx 268\text{MB}$$

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

About stencil method:

- computation » communication
- data locality and effective cache usage improve the performance

Possible improvements:

- Use MPI derived datatypes
-

Thank You!