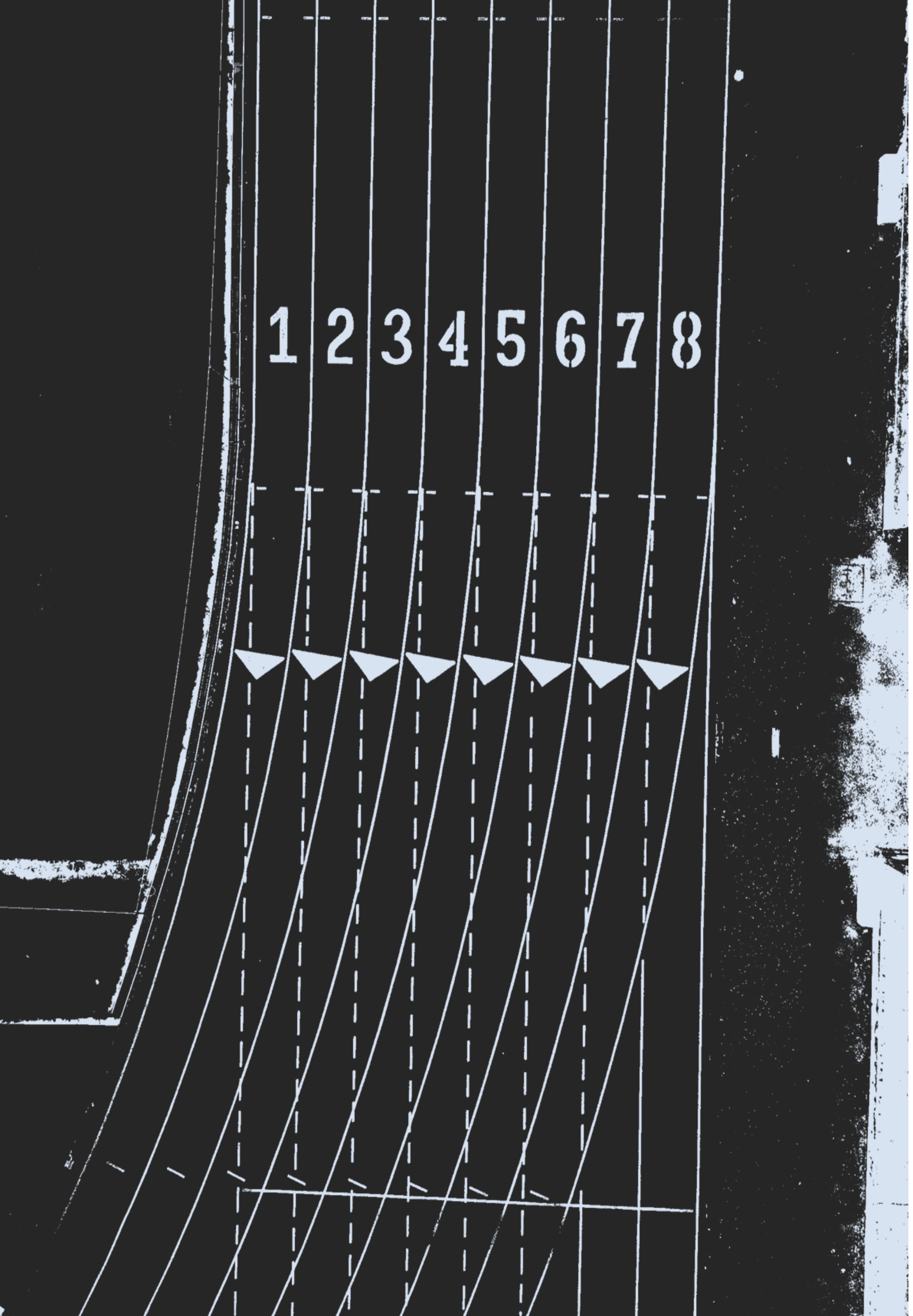


HPC 24/25

Solving Heat Equation in parallel: the 5-Point Stencil method

Invernici Marta

The numerical problem

The ***heat equation*** is a PDE modelling physical transfer of heat in a region over time:

$$\frac{\partial u(t,x,y)}{\partial t} = \alpha\left(\frac{\partial^2 u(t,x,y)}{\partial x^2} + \frac{\partial^2 u(t,x,y)}{\partial y^2}\right)$$

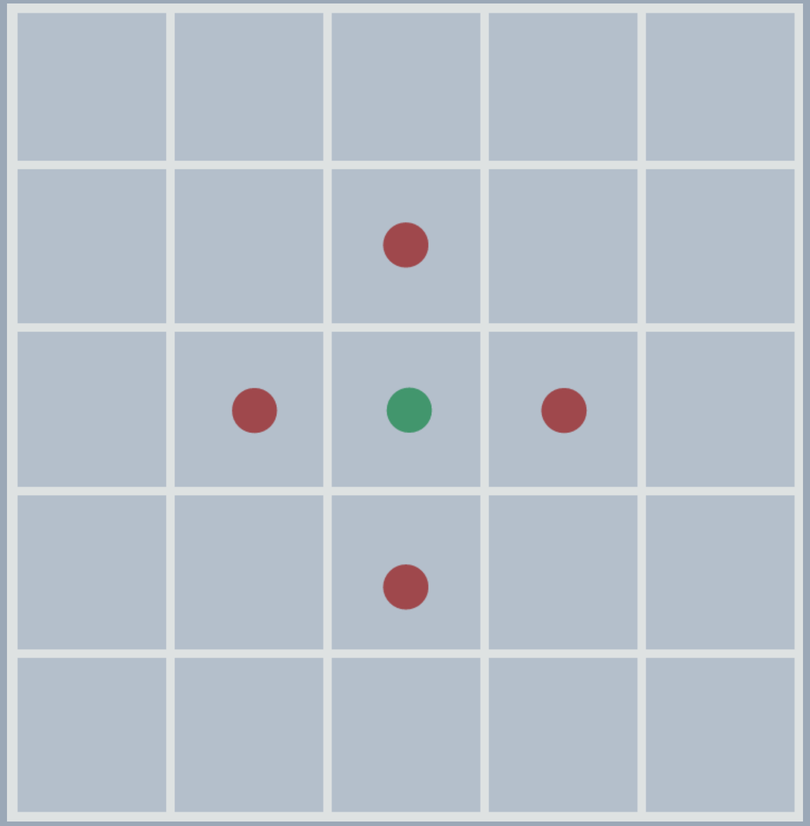
Numerically, the solution is approximated as a discretized space-time function using Finite-difference methods to discretize the derivatives.

The ***update equation***, approximates function $\boldsymbol{u}$, at time $\boldsymbol{t}$ and place $\boldsymbol{x}$ as $U_{m.l}^{n} \approx u(n\Delta t, m\Delta x, l\Delta y)$:

$$U_{m,l}^{n+1} = [1 - 4(\alpha\Delta t/\Delta x^2)]U_{m,l}^{n} + (\alpha\Delta t/\Delta x^2)\left(U_{m-1,l}{}^{n} + U_{m+1,l}{}^{n} + U_{m,l-1}{}^{n} + U_{m,l+1}{}^{n}\right)$$

This approximation scheme falls into the general category of a ***stencil computation :*** value at a point (m,ℓ) in space at time step n requires only values from neighboring points of (m,ℓ) from a few previous time steps.

We relied on a ***5-point stencil*** access pattern, but there are many alternative approximation schemes and related stencil computations.



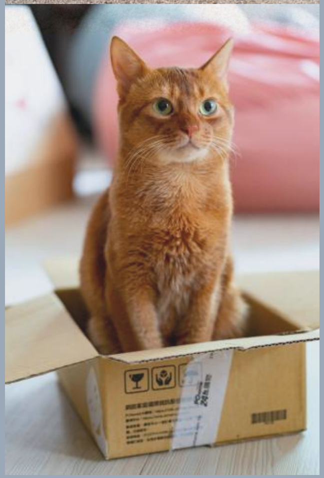
Constraints of serial programming

- Time:

Ts $= \Theta(N^2)$

- Space:

Ss $= \Theta(N^2)$



Perks of parallel programming & HPC

- Fixed problem size, decreased time

  (Strong scaling)

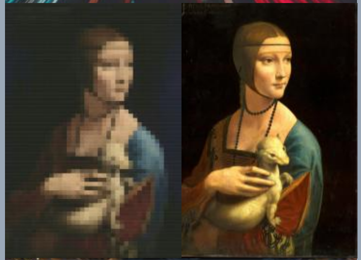
- Bigger problem size, constant time

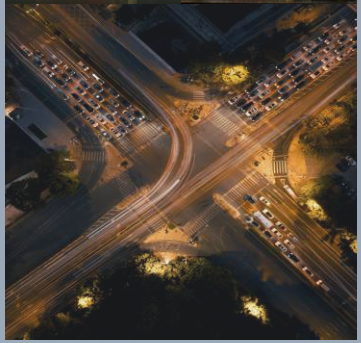
  (Weak scaling)

- Beyond single-node limits

BUT:

- Overhead





Hybrid Parallel Implementation

OpenMP

Shared Memory regime

- **#pragma omp parallel for schedule(static)**

  plane_update

  memory_allocate (grid initialization)

- **#pragma omp parallel for reduction(+: totenergy)**

  get_total_energy

MPI

Distributed Memory regime

- **MPI_Init_thread, MPI_Comm_rank, MPI_Comm_size, MPI_Comm_dup, MPI_Finalize**

  initialization and finalization of MPI env

- **MPI_Isend, MPI_Irecv, MPI_Waitall, MPI_Test**

  non-blocking P-2-P communications and assessment

- **MPI_Bcast, MPI_Reduce, MPI_Gather**

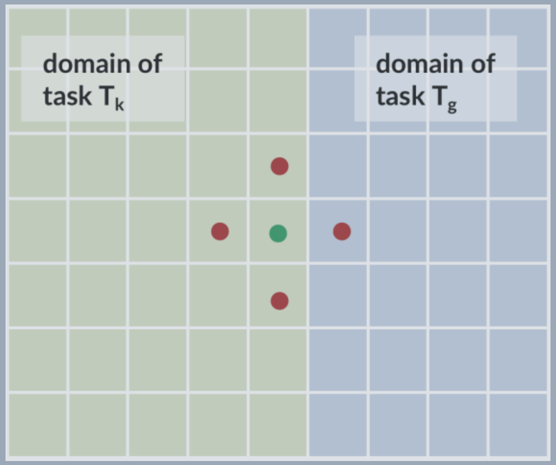
  initilalize_sources

  output_energy_stat

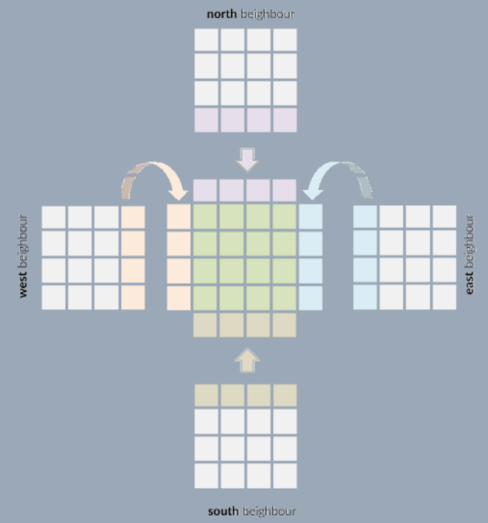
  time results

- **MPI_Wtime()**

Domain Decomposition & Communications

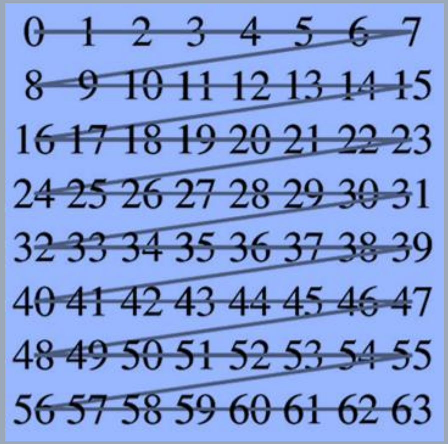


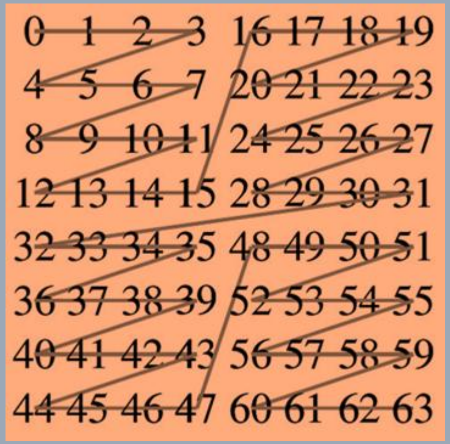
- **Hybrid model**: MPI_Init_thread requesting FUNNELED; OpenMP does the intra-rank work.
- **Domain decomposition**: neighbor array (N/E/S/W); edges handled with MPI_PROC_NULL.
- **Halo buffers**: N/S halos as contiguous rows; E/W halos via temporary column buffers.
- **Non-blocking halo exchange**: MPI_Isend / MPI_Irecv with consistent per-direction tags.
- Comm/comp overlap: update the inner grid while messages are in flight; finalize with MPI_Waitall.
- **Border completion**: after receives complete, copy halos into the plane, then update border cells.



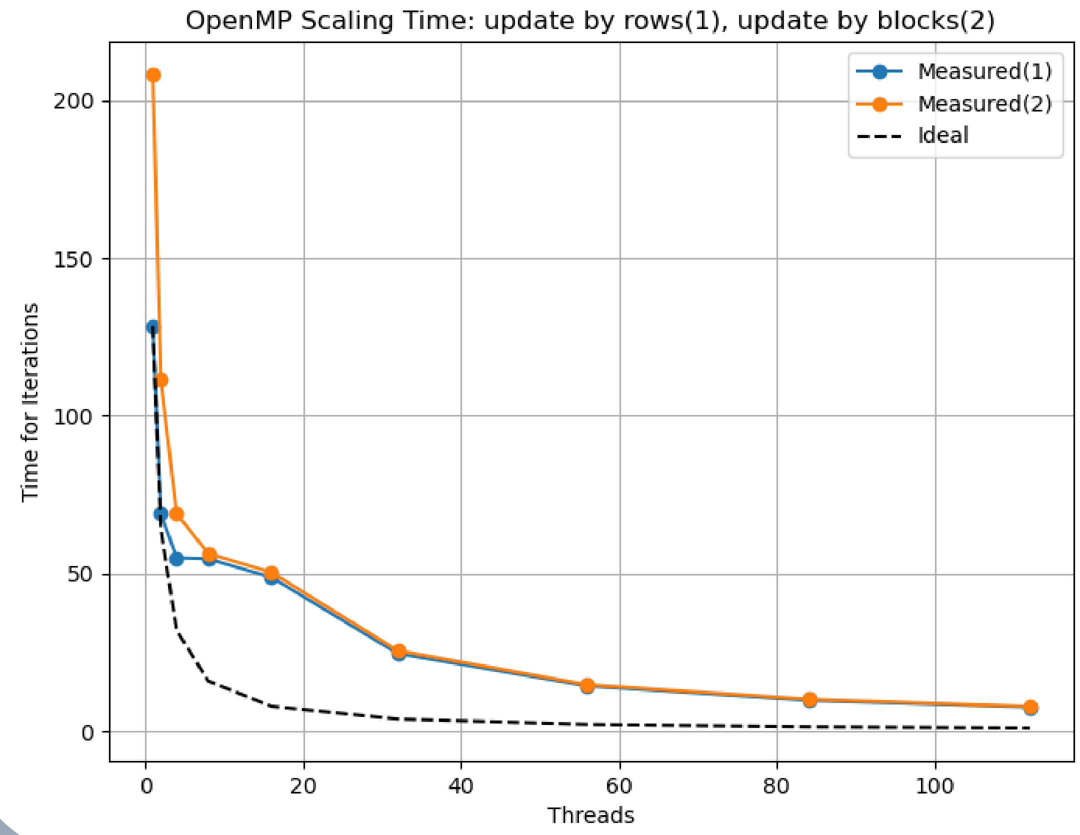
Computing Grid Updates

- **Row-order (1)** or **Block-order(2)** scan for locality enhancement
- **Touch-by-all** with parallel malloc initialization to place pages locally
- **Static scheduling** for predictable locality
- **Comm/comp overlap** : Inner grid update + Borders update
- **OMP_PLACES**=cores, **OMP_PROC_BIND**=close for memory affinity



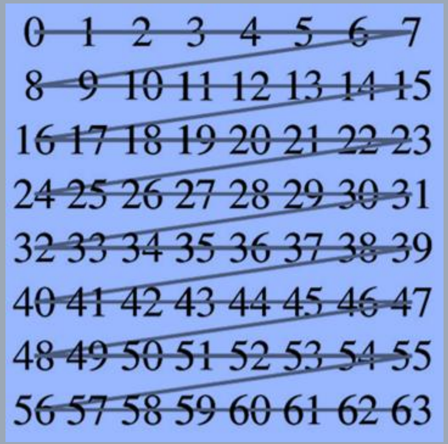


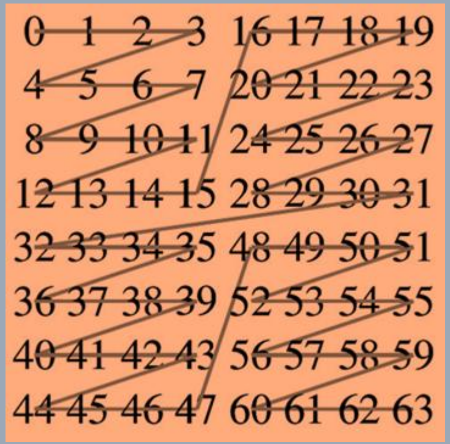
Orfeo EPYC: 2x(16x4) cores, 512 GB



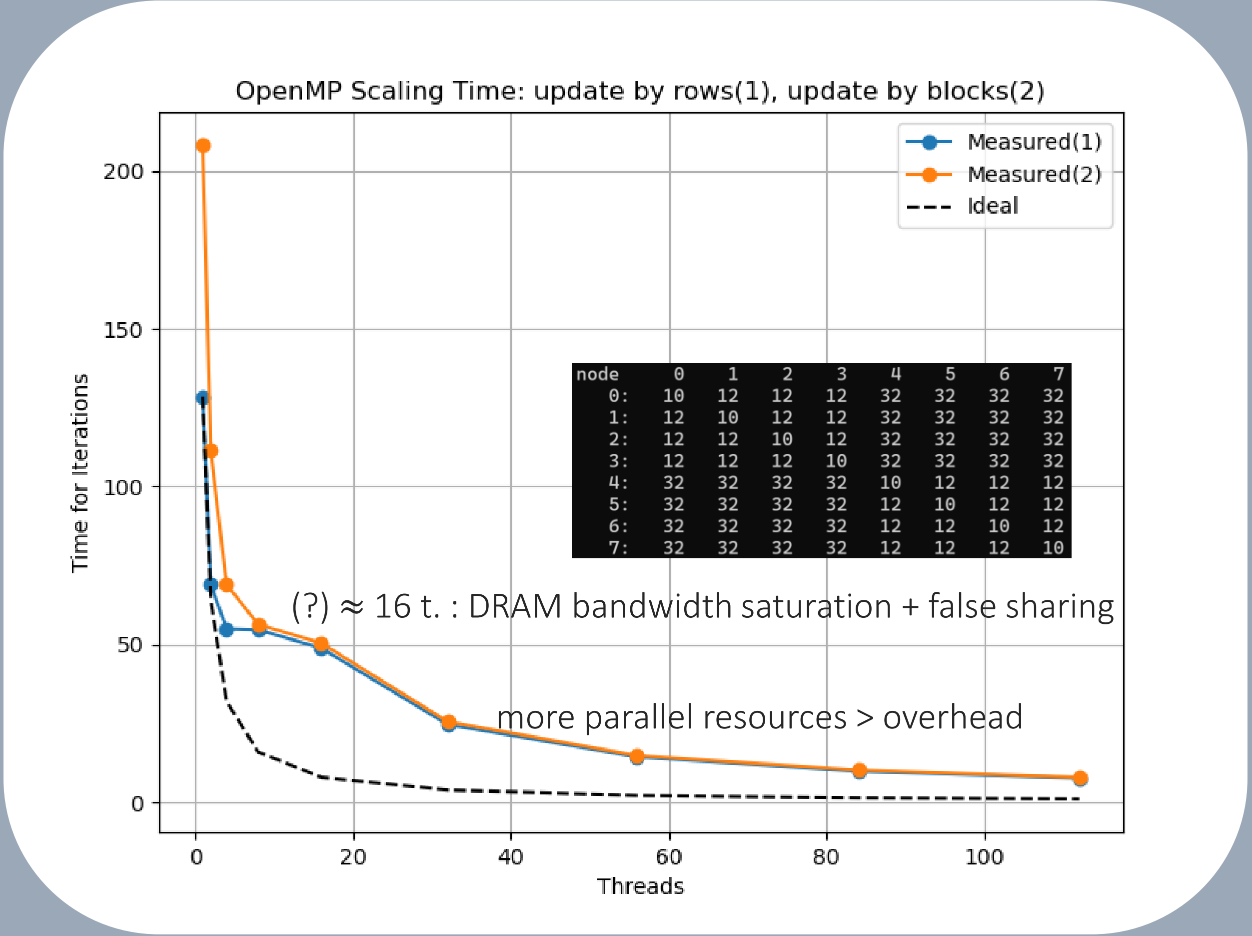
Computing Grid Updates

- **Row-order (1)** or **Block-order(2)** scan for locality enhancement
- **Touch-by-all** with parallel malloc initialization to place pages locally
- **Static scheduling** for predictable locality
- **Comm/comp overlap** : Inner grid update + Borders update
- **OMP_PLACES**=cores, **OMP_PROC_BIND**=close for memory affinity





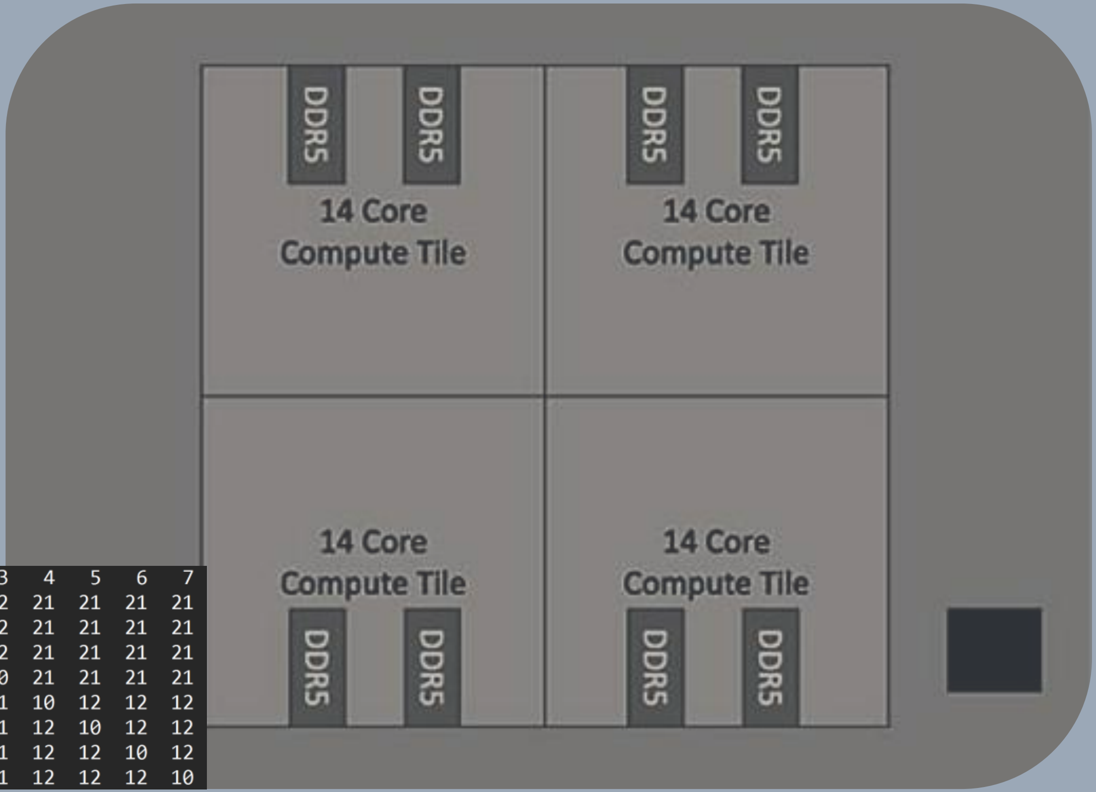
Orfeo EPYC: 2x(16x4) cores, 512 GB



Leonardo – DCGP Module

<u>Data Centric General Purpose (DCGP) module</u>
1536 nodes , 172032 CPU cores
https://leonardo-supercomputer.cineca.eu/hpc-system/#jump-partition

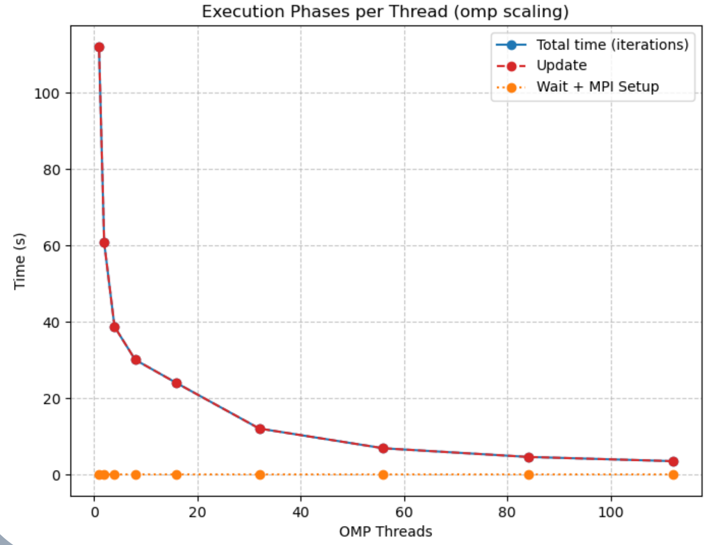
Three-node BullSequana X2140 blade.
Each **NODE** has:
- **2x 56-core** *Intel Xeon Platinum 8480+* CPU
  (side: conceptual diagram)
- **2x 8x 32 GB** DDR5-4800 **(512 GB)**

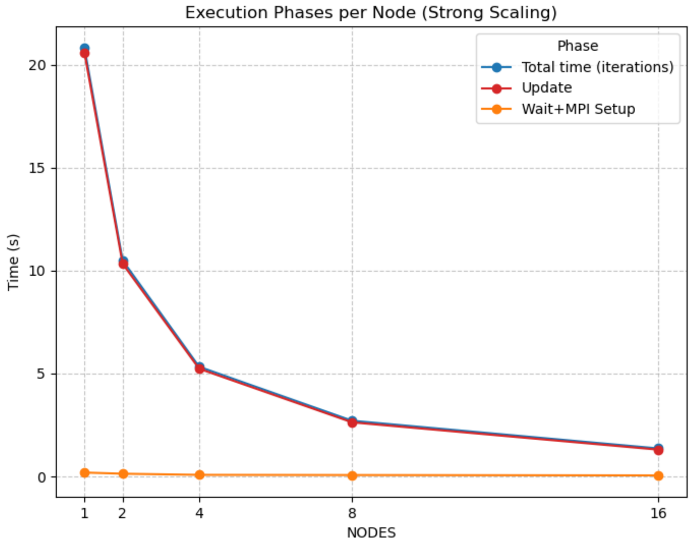


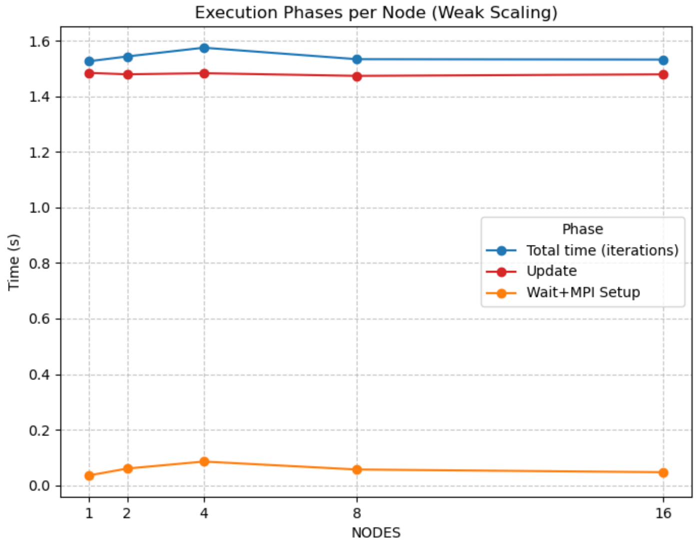
node	0	1	2	3	4	5	6	7
0:	10	12	12	12	21	21	21	21
1:	12	10	12	12	21	21	21	21
2:	12	12	10	12	21	21	21	21
3:	12	12	12	10	21	21	21	21
4:	21	21	21	21	10	12	12	12
5:	21	21	21	21	12	10	12	12
6:	21	21	21	21	12	12	10	12
7:	21	21	21	21	12	12	12	10

Scalability tests

Communication and Computation TIMES

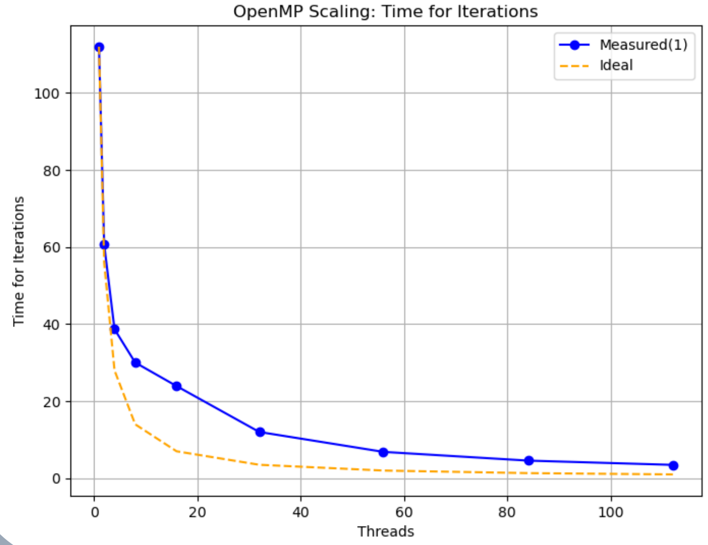


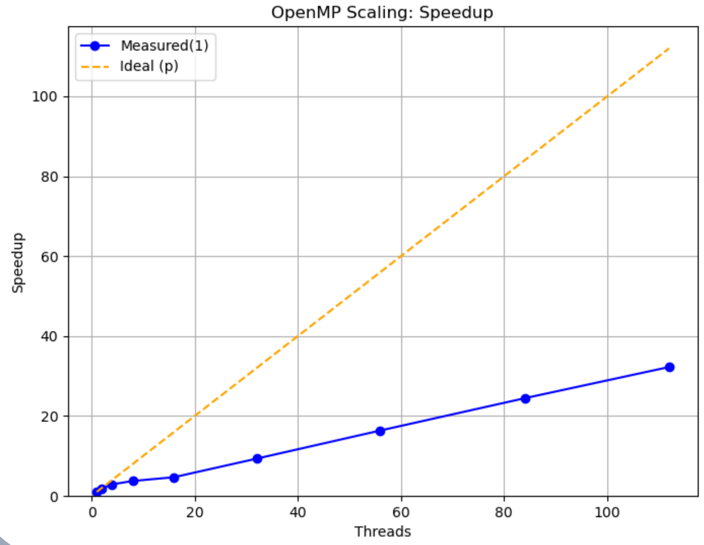


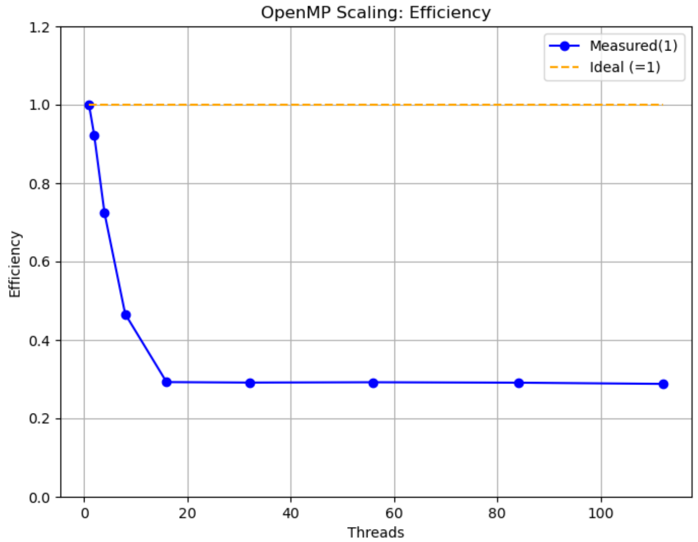


Test threads: 1 2 4 8 16 32 56 84 112; Test nodes: 1 2 4 8 16 nodes

OPENMP SCALING

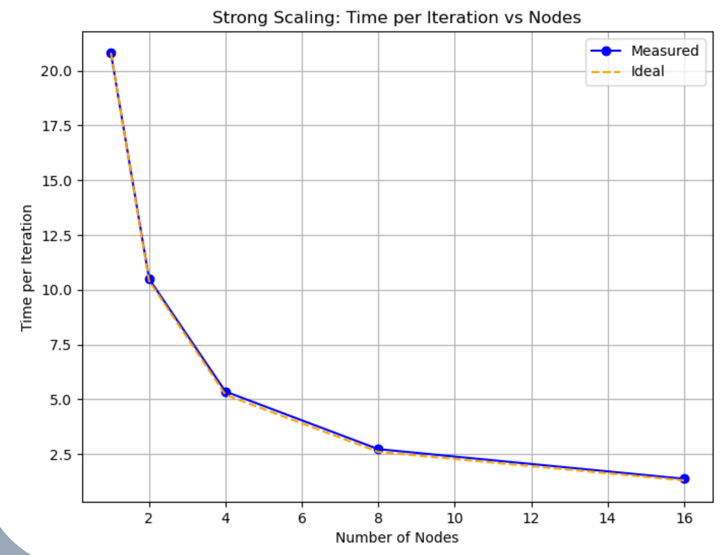


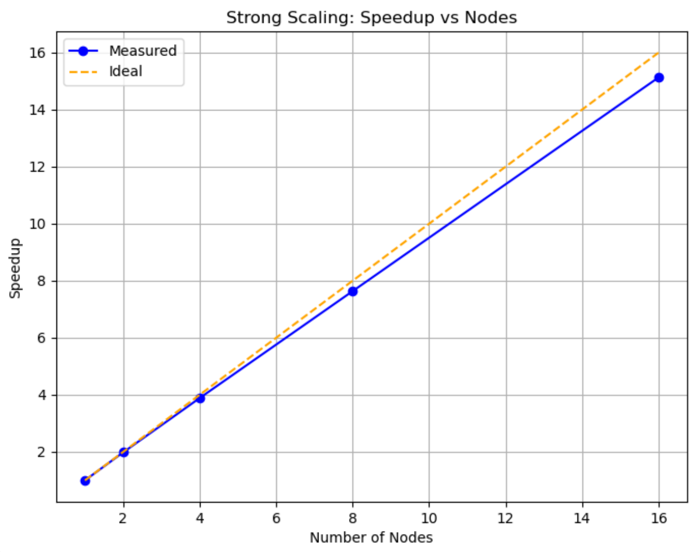


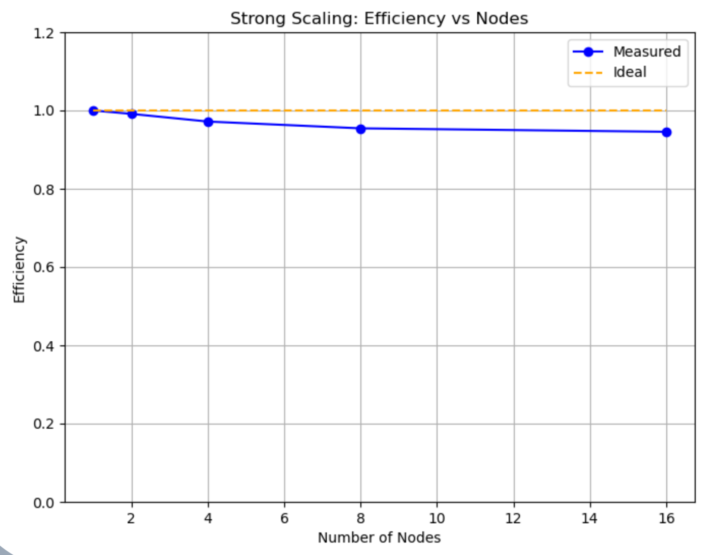


Test points: 1 2 4 8 16 32 56 84 112 threads; Grid Size: 10000x10000; Iterations: 750

STRONG SCALING



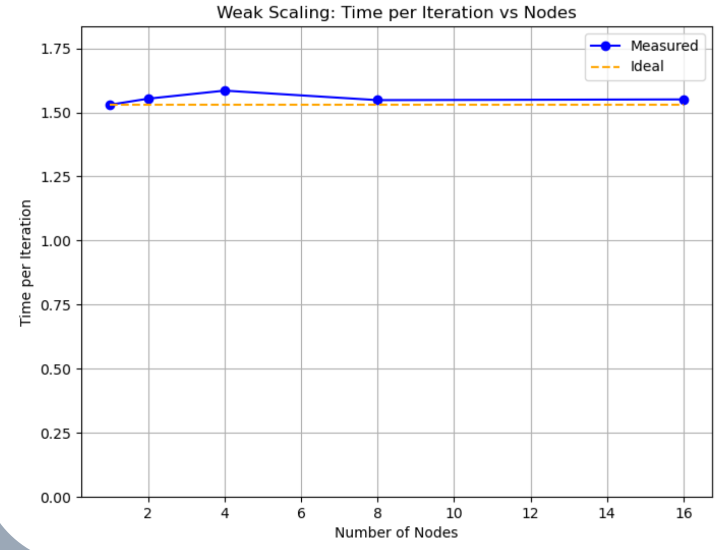


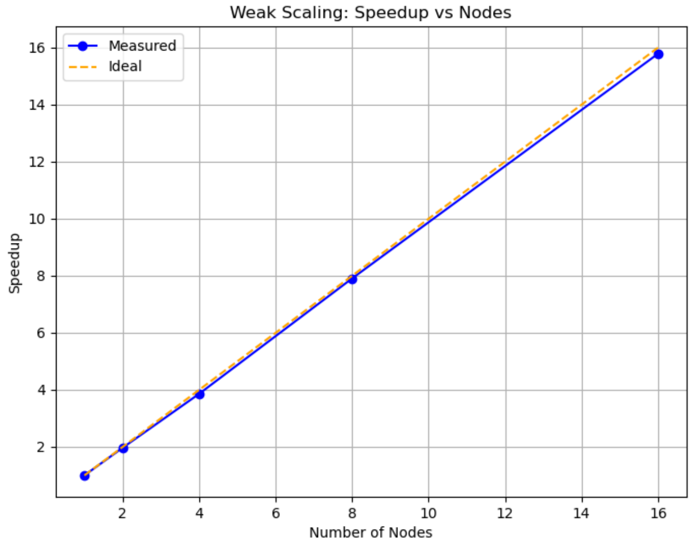


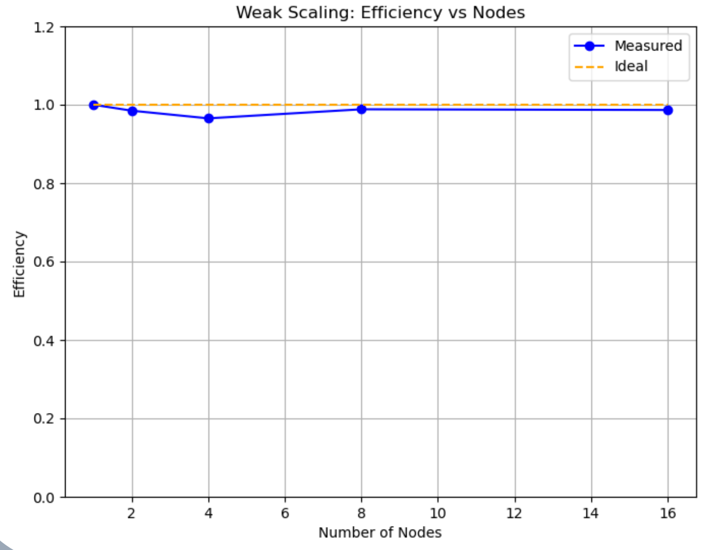
Node: 14 threads-per-task * 8 tasks; Test points: 1 2 4 8 16 nodes; Grid Size: 15000x15000; Iterations: 500

*All timings are wall-time of the slowest rank

WEAK SCALING







Node: 14 threads-per-task * 8 tasks; Test points: 1 2 4 8 16 nodes; Grid Size: 4000^2 8000x4000 8000^2 16000x8000 16000^2 ;
Iterations: 500

*All timings are wall-time of the slowest rank

To infinity and beyond, i.e. possible enhancements

- Goals achieved. **Hybrid** MPI+OpenMP 5-point stencil with **compute/comm overlap**; timings are wall-time of the slowest rank; scaling is solid (OMP up to the DRAM/NUMA knee; strong/weak as expected); communications are not the bottleneck—the grid update dominates.

- **Where time goes:** The **grid update** dominates the runtime; **communication** is largely hidden by overlap and does not set the pace.

Top priority now. Strengthen the update kernel with more update parallelism and cache locality to reduce memory traffic.

- **Why:** the stencil is likely **memory-bandwidth bound**. Each update touches several neighbors and does a store; if data isn't reused in cache, we re-fetch lines from DRAM and even remote NUMA, wasting bandwidth. Possible working direction to follow to push performance:

    Tiling to keep working sets in L1/L2/L3 caches;

    SIMD to update several cells with one fetch;

    Local allocation, sticking threads to their cores to keep accesses local;

    Padding/alignment lowers false sharing pushing performance toward the bandwidth roofline.

Thank you