INFO0939:

HIGH PERFORMANCE SCIENTIFIC COMPUTING

Julien HANSEN
s201806

Clément SMAGGHE
s203332

January 2024

LIÈGE université

Outline

- Serial Code
- Message Passing Interface (MPI)
- Open Multi-Processing (OMP)
- Scalability
- Profiling
- Graphics Processing Units (GPU)
- Extended physical model

Serial code

Modification of default code and memory locality

In the given code, the GET and SET macros use an indexing scheme that places elements with consecutive i (columns) next to each other in memory.

Therefore, the loop order was changed to j, i (rows, columns) to increase performance in row-major order and minimize the number of cache miss.

We now complete around 518 MUpdates/s (speedup factor $S = 6.16$) on the login node of NIC5.

Bilinear interpolation

We have implemented Bilinear interpolation for bathymetry evaluation instead of nearest neighbors

Arithmetic intensity

	Floating point ops	memory accesses (double)
Update velocities	14	7
Update eta	9	7
Total	25	14

Executing 25 flop per update on a machine capable of 2.8 TFLOPS/s,

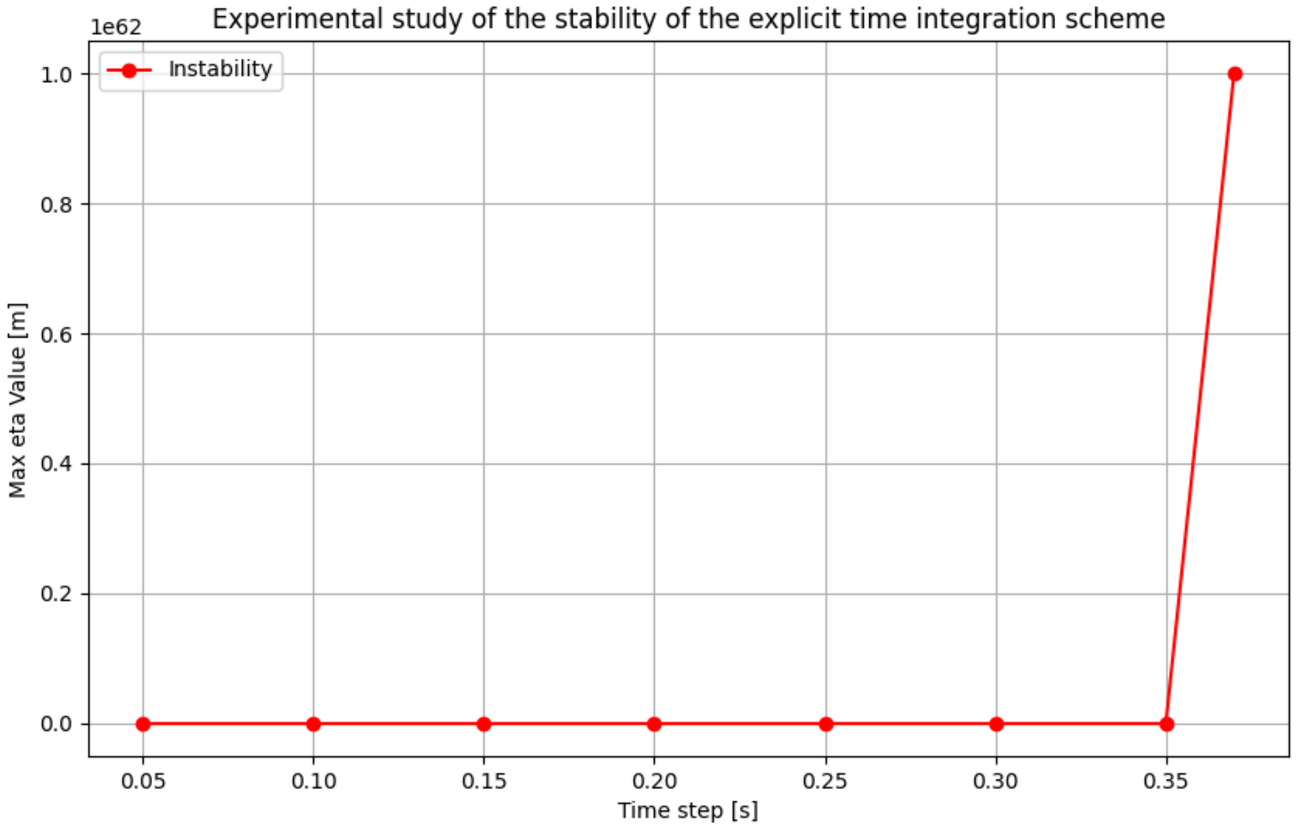
2800 / 25 = 112 GUpdates/s

Accessing the memory 14 times per update at a 200 GB/s rate,

200 / (14 . 8) = 1,785 GUpdates/s

Stability Analysis

This figure shows the presence of the instability, varying the time step for a fix value of dx leads to it, of course to stay in stable value of dt and dx/dy, one should a study of stability using Von-Neuman method which requires a form for the h(x,y) function.



Message Passing Interface (MPI)

Domain partitioning

As in the tutorials, we use **MPI_Dims_create()** on the bathymetry data (before interpolation) to perfectly split the domain in parts according to the number of available ranks (and no resources wasted).

To reduce the amount of data exchanged between ranks, we make sure to use fewer threads to the smaller dimension and to split the most evenly as possible the bathymetry map.

At the end, all ranks subdomains are displayed in the manifest VTK file with different origins.

Number of MPI ranks	Bathymetry dimensions X×Y	MPI_Dims_create() X×Y ranks	Corrected X×Y ranks
6	40×60	3×2	2×3
7	100×100	7×1	7×1
7	100×900	7×1	1×7

Communication

MPI exchanges data between ranks using non-blocking communication.

Borders of the subdomain are computed first, then sent and received (in the background). During that time, the inner part is computed and finally all transactions are waited before next time step.

Ghost cells are **MPI_Type_vector()**, to limit the number of calls to **MPI_Isend()**.

Open Multi-Processing (OMP)

Collapse clause

- We used **#pragma omp parallel for**, on our main loop such as for updating eta, velocities and the interpolation in which we also used **private** clause to ensure that each thread has its own copy of the variables.
- Our testing showed that using **collapse** clause slow down our program, this is due to the spatial locality being disrupted leading to more cache miss and thus worse performance.

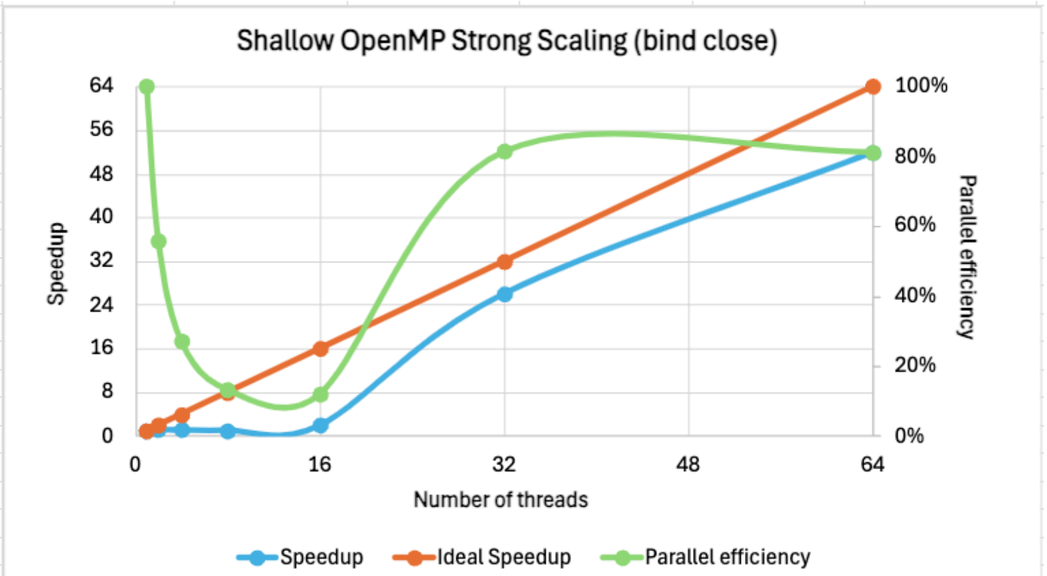
Comparison

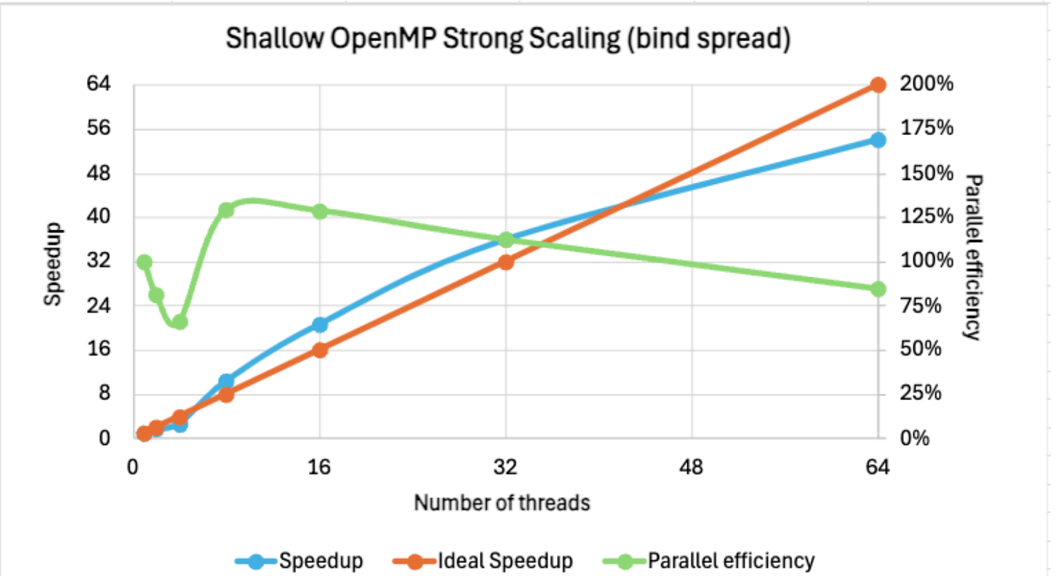
	4M points problem
OMP 8 threads	365.853 MUpdates/s
MPI 8 ranks	620.672 MUpdates/s
Hybrid 4 ranks, 2 threads per rank	404.197 MUpdates/s

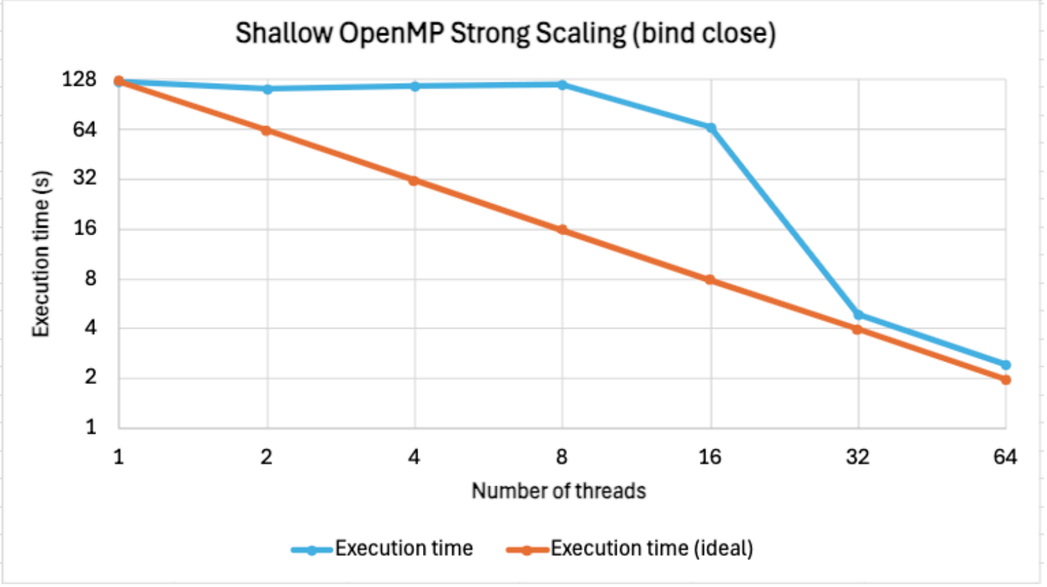
Scalability

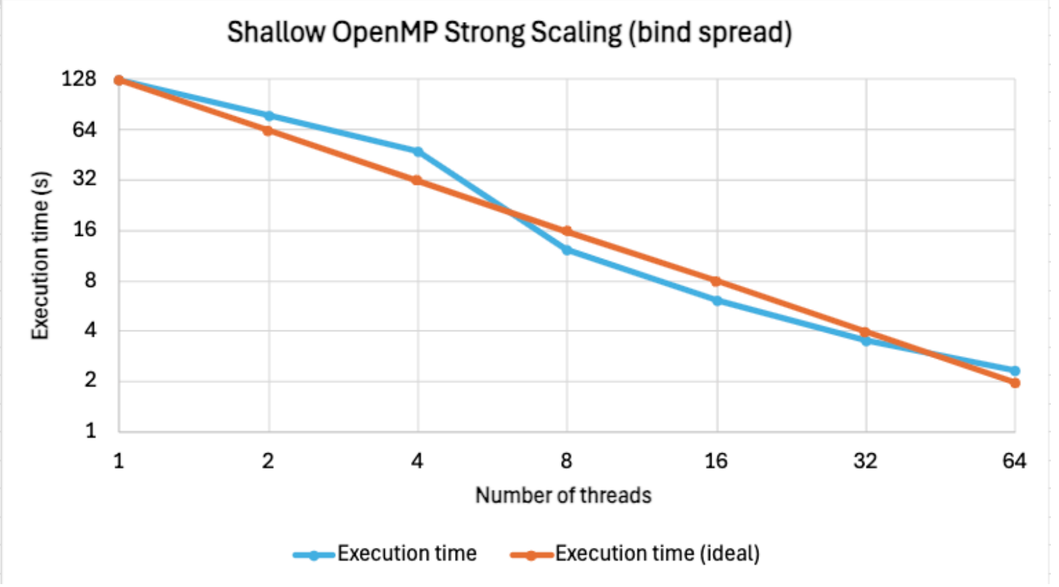
Strong Analysis

OMP Strong Analysis





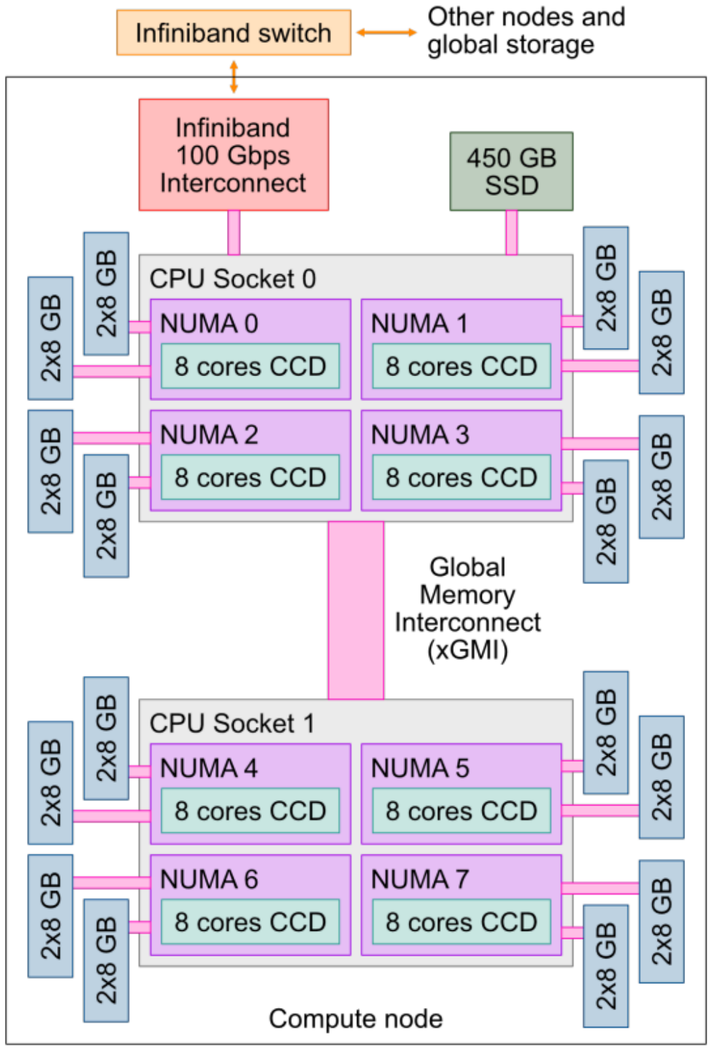




Nic5 Architecture

Reminder

- The NIC5 architecture consists of compute nodes providing each a total of 64 cores.
- These CPUs are connected through a Global Memory Interconnect (GMI). Each node has 256 GB of memory, split into 8 NUMA nodes.



Overview of a NIC5 compute nodes

Spread vs Close options

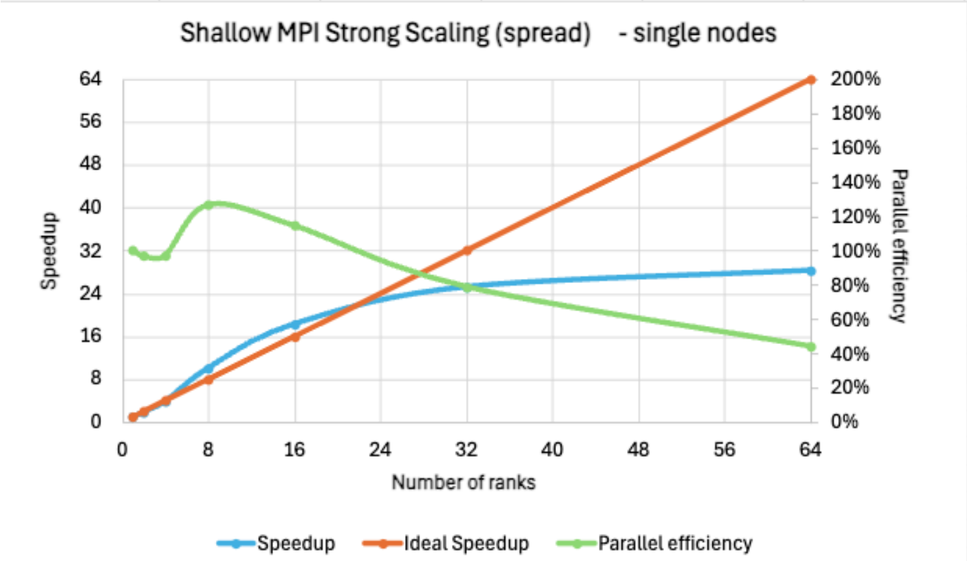
Bind Close

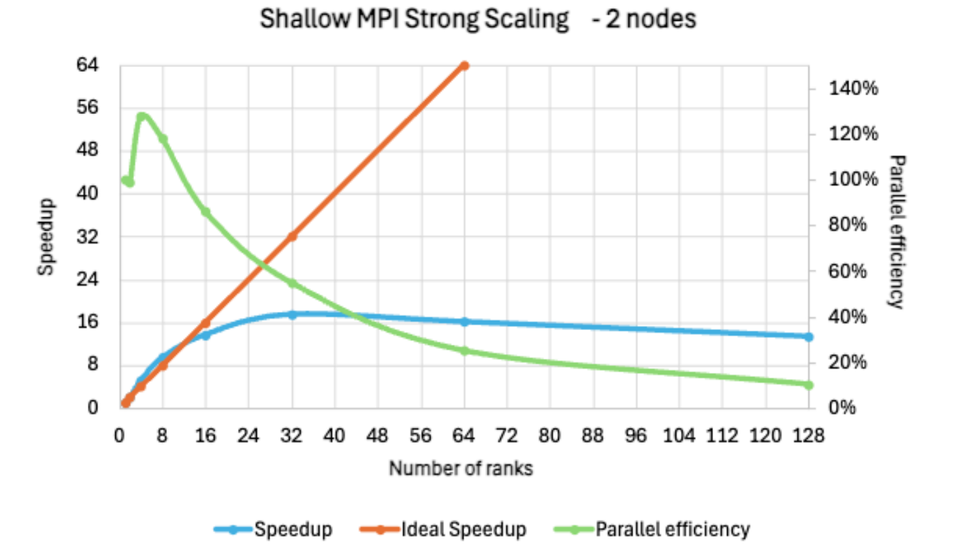
- Threads are bound to specific CPU cores within the same NUMA node
- Threads on the same NUMA node share access to the same memory channels, which can lead to memory contention when like in this case, the code is memory-bounded
- Result in increased overhead due to competition for memory access, causing slower performance as we can see in the graphs

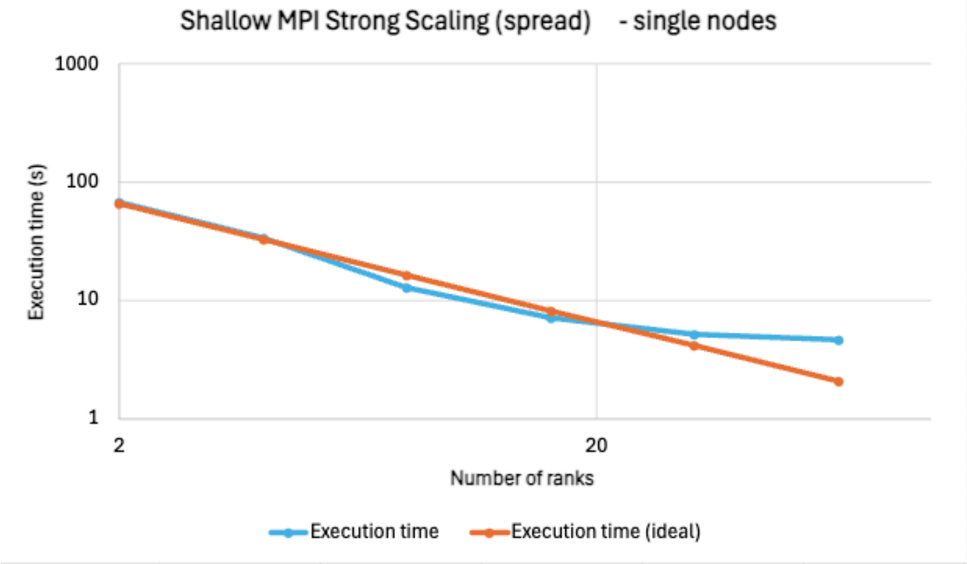
Bind Spread

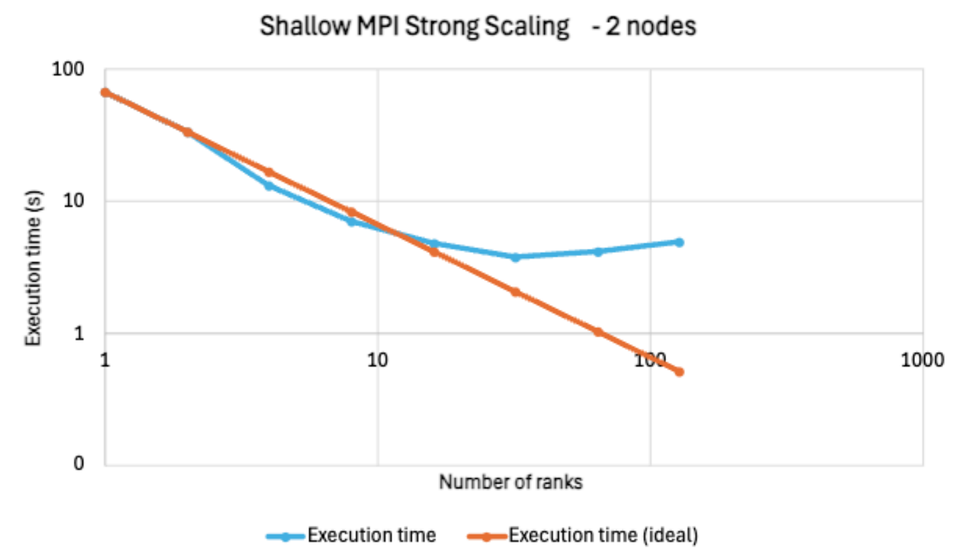
- Threads are distributed across different NUMA nodes, they are not bound to a specific core
- This reduces memory contention by allowing threads to access separate memory resources, improving overall memory bandwidth and reducing overhead
- This leads to less competition among threads and thus better overall results on the execution time.

MPI Strong Analysis

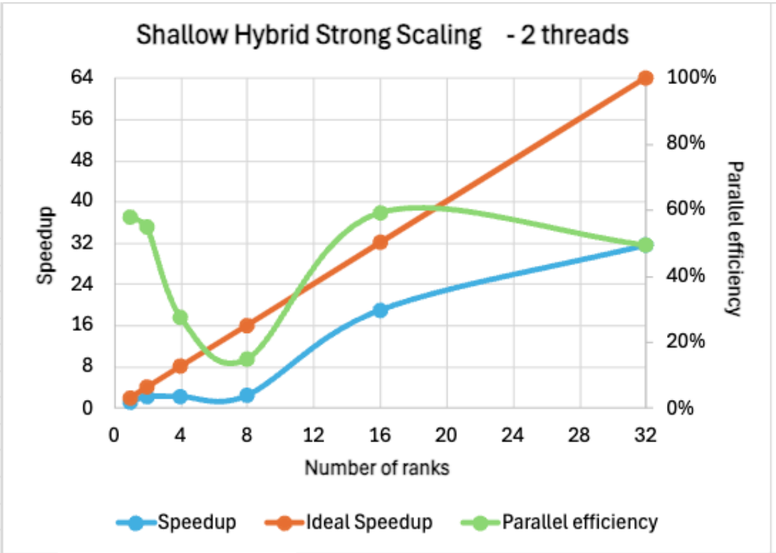


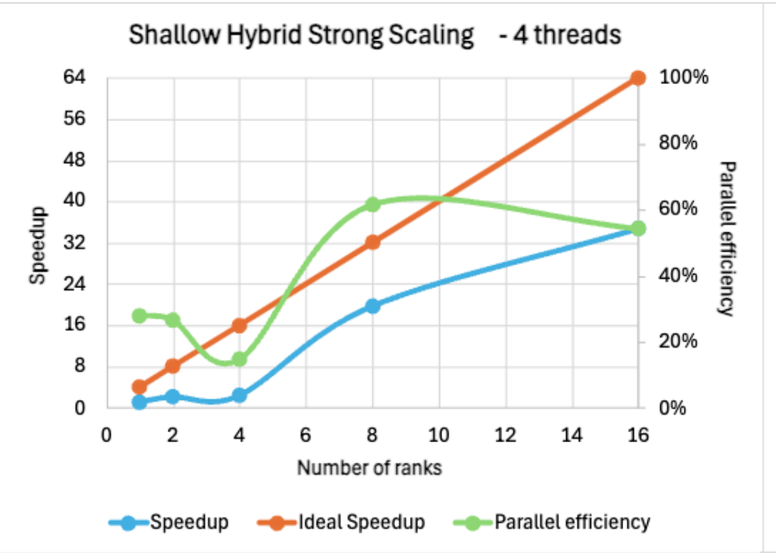


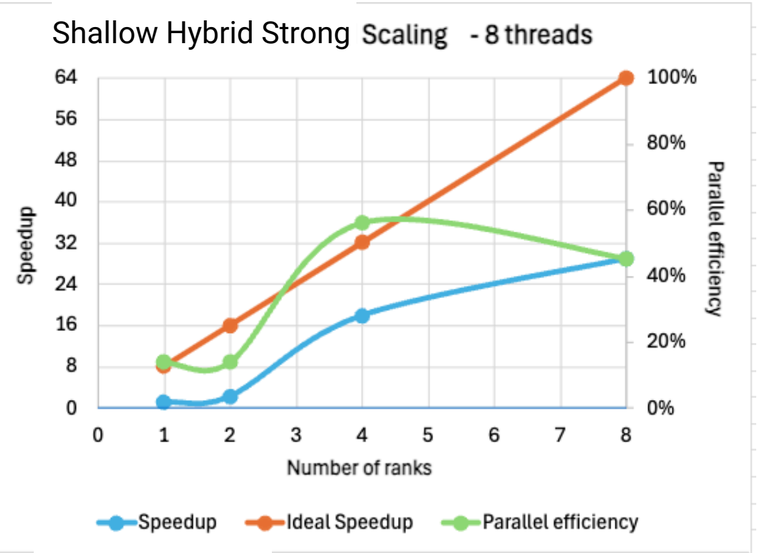


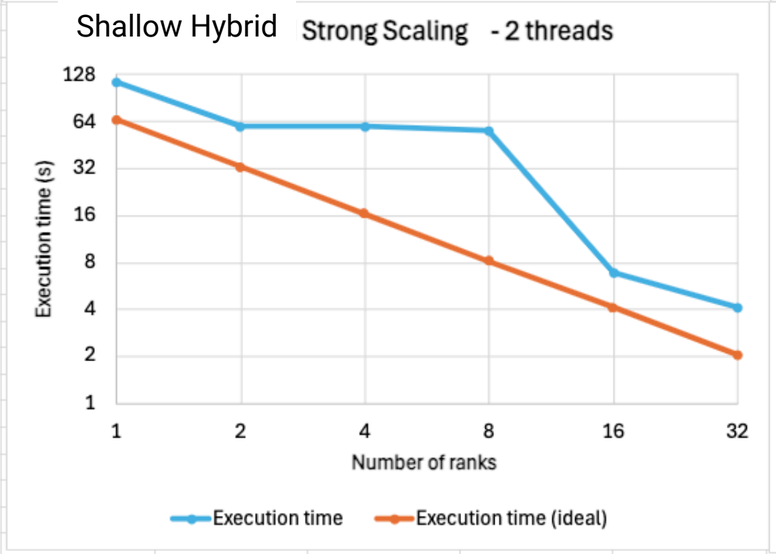


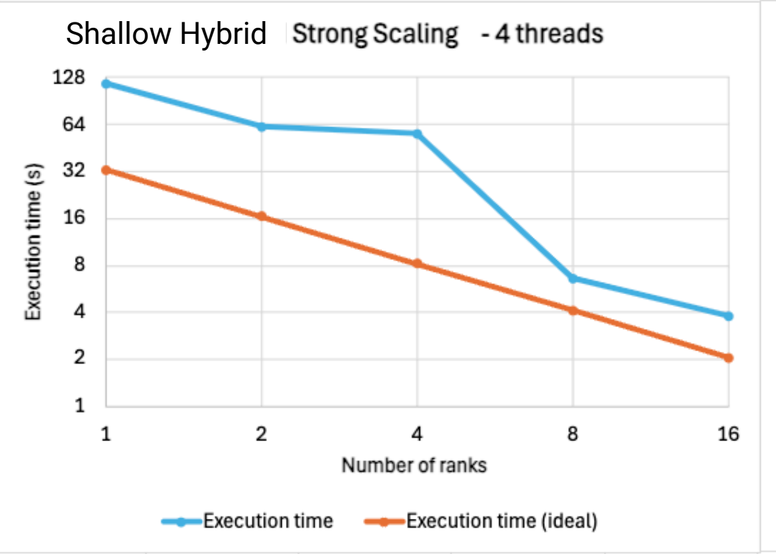
Hybrid Strong Analysis

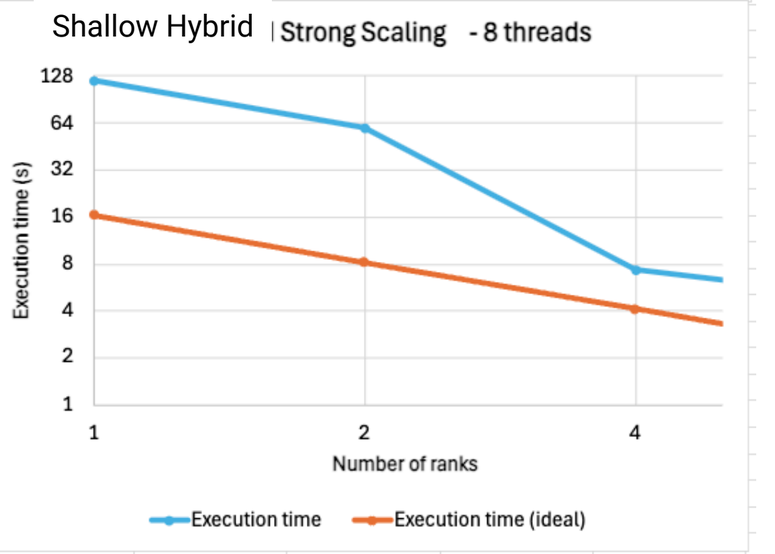






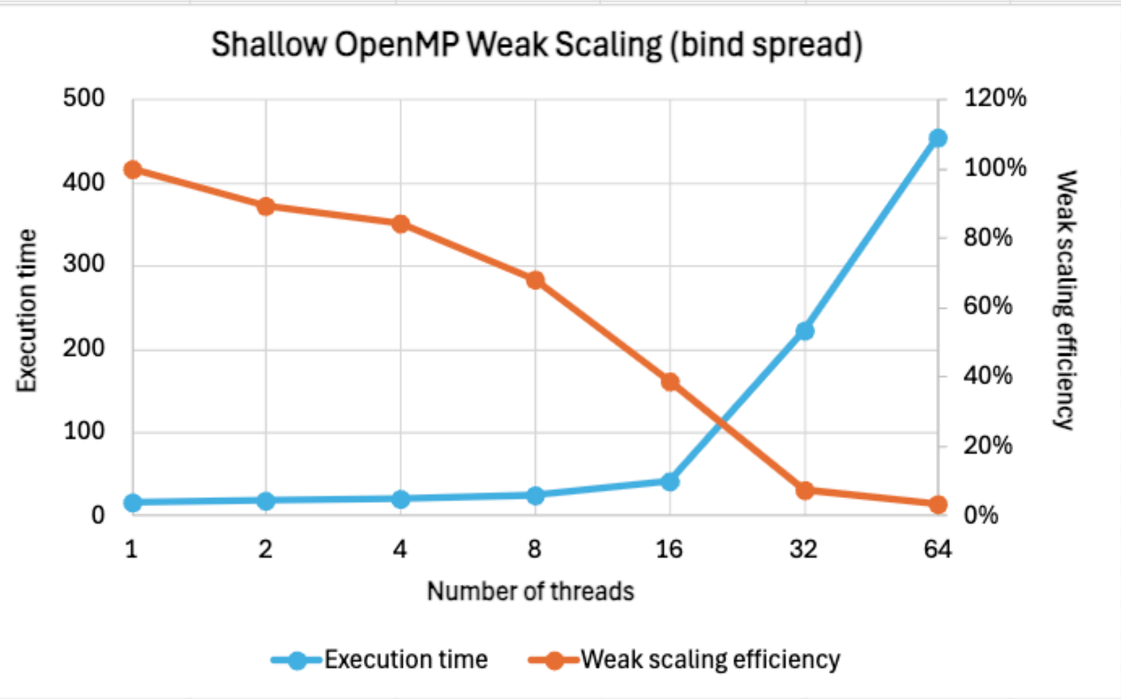


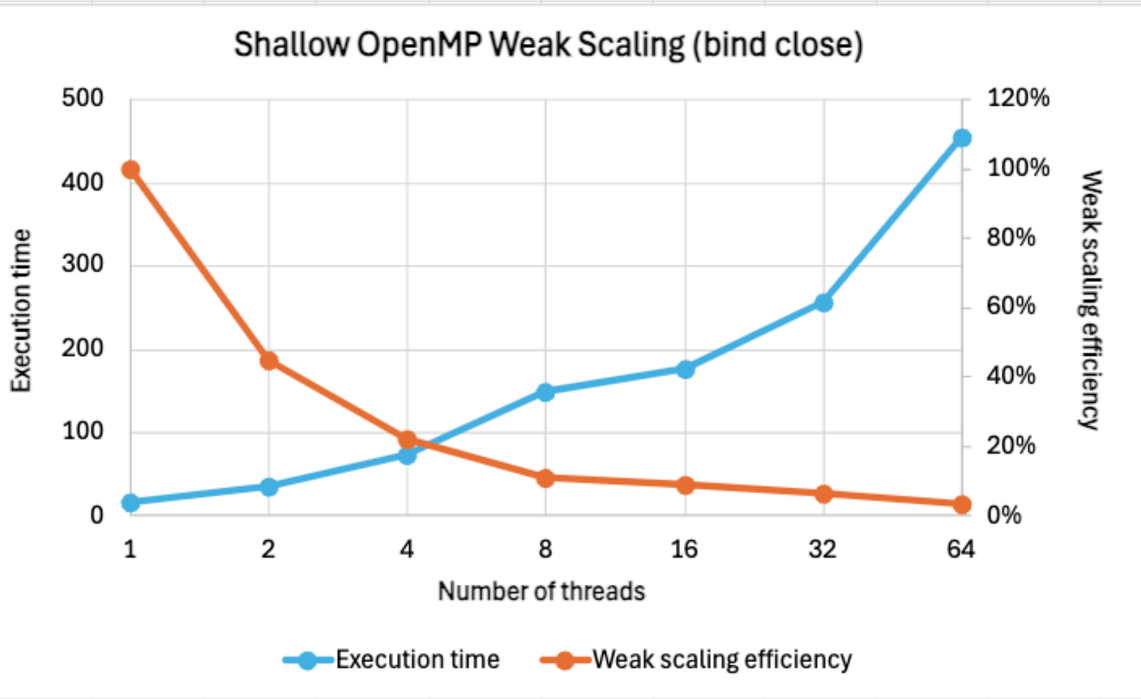




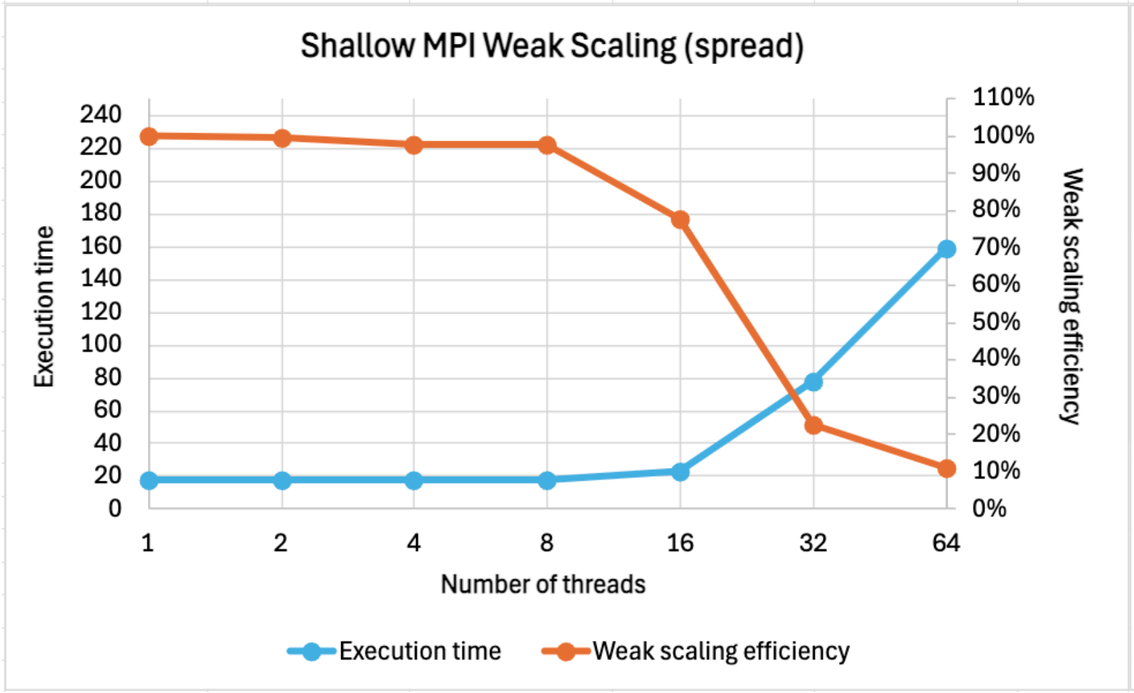
Weak Analysis

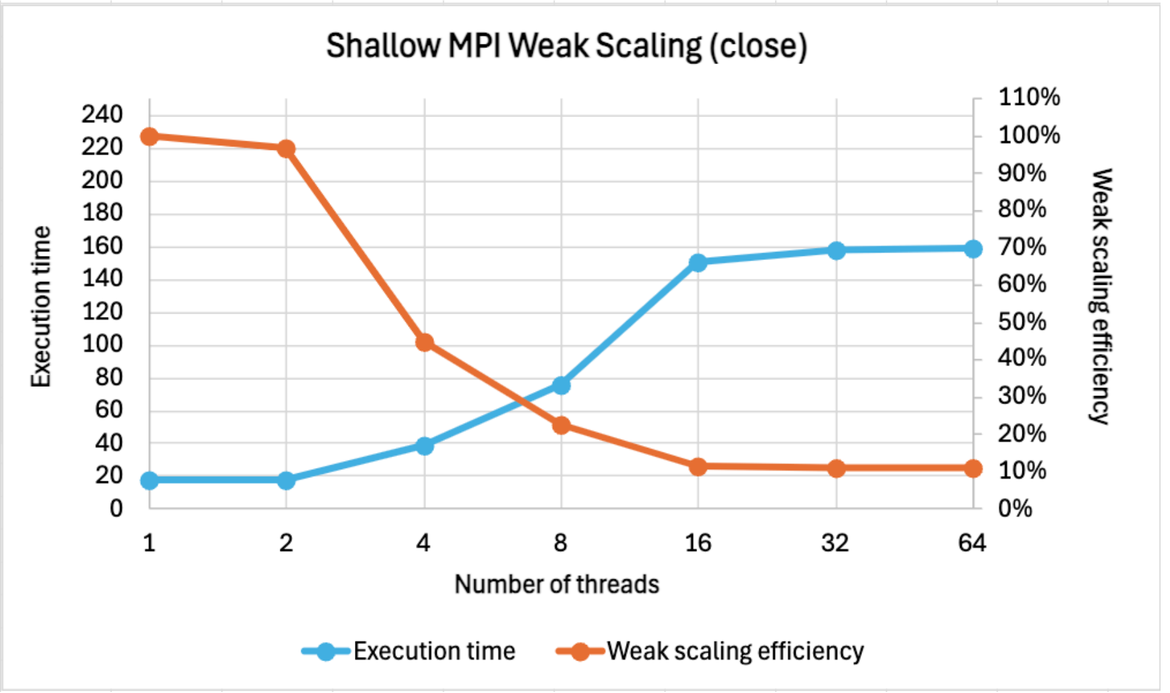
OMP Weak Analysis



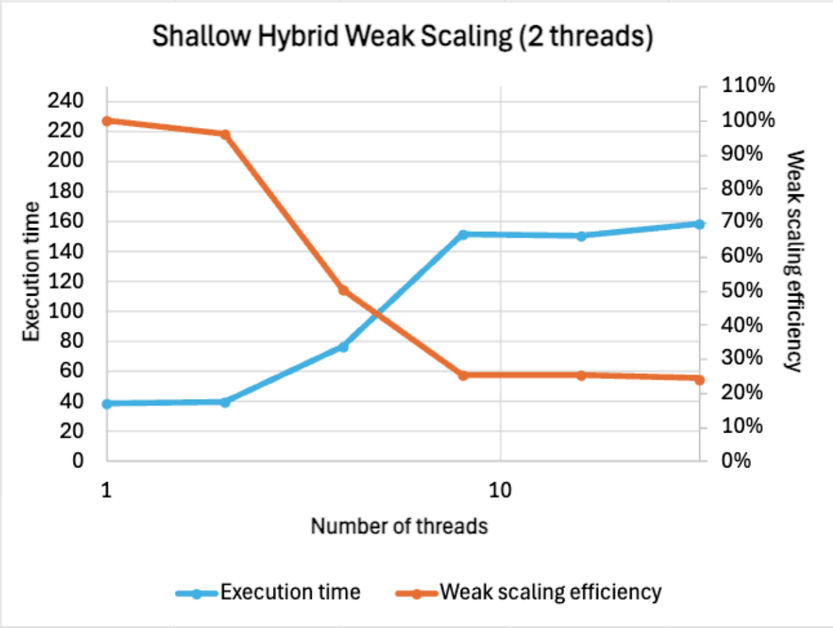


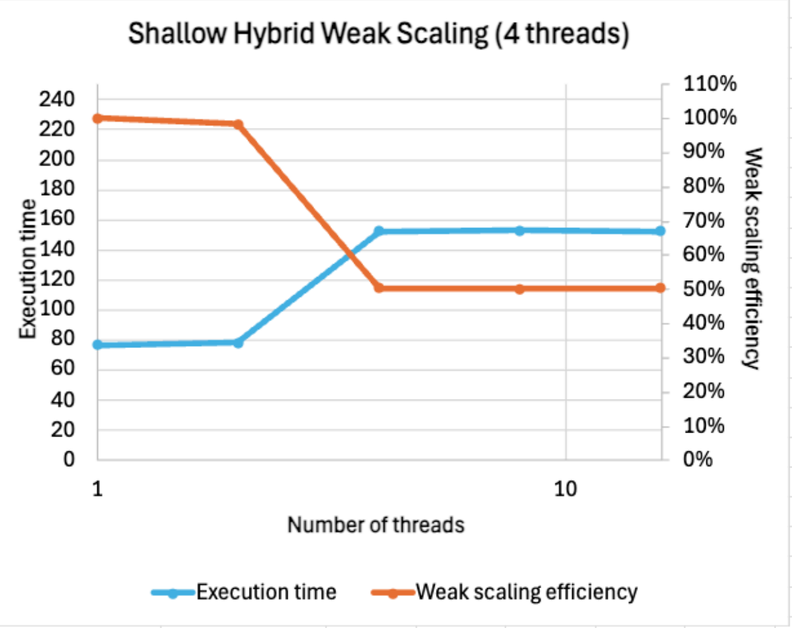
MPI Weak Analysis

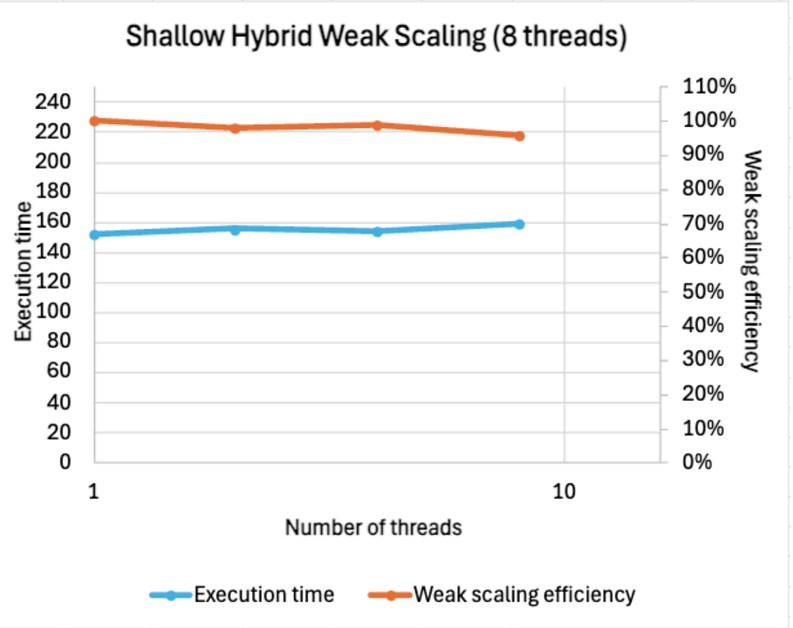




Hybrid Weak Analysis

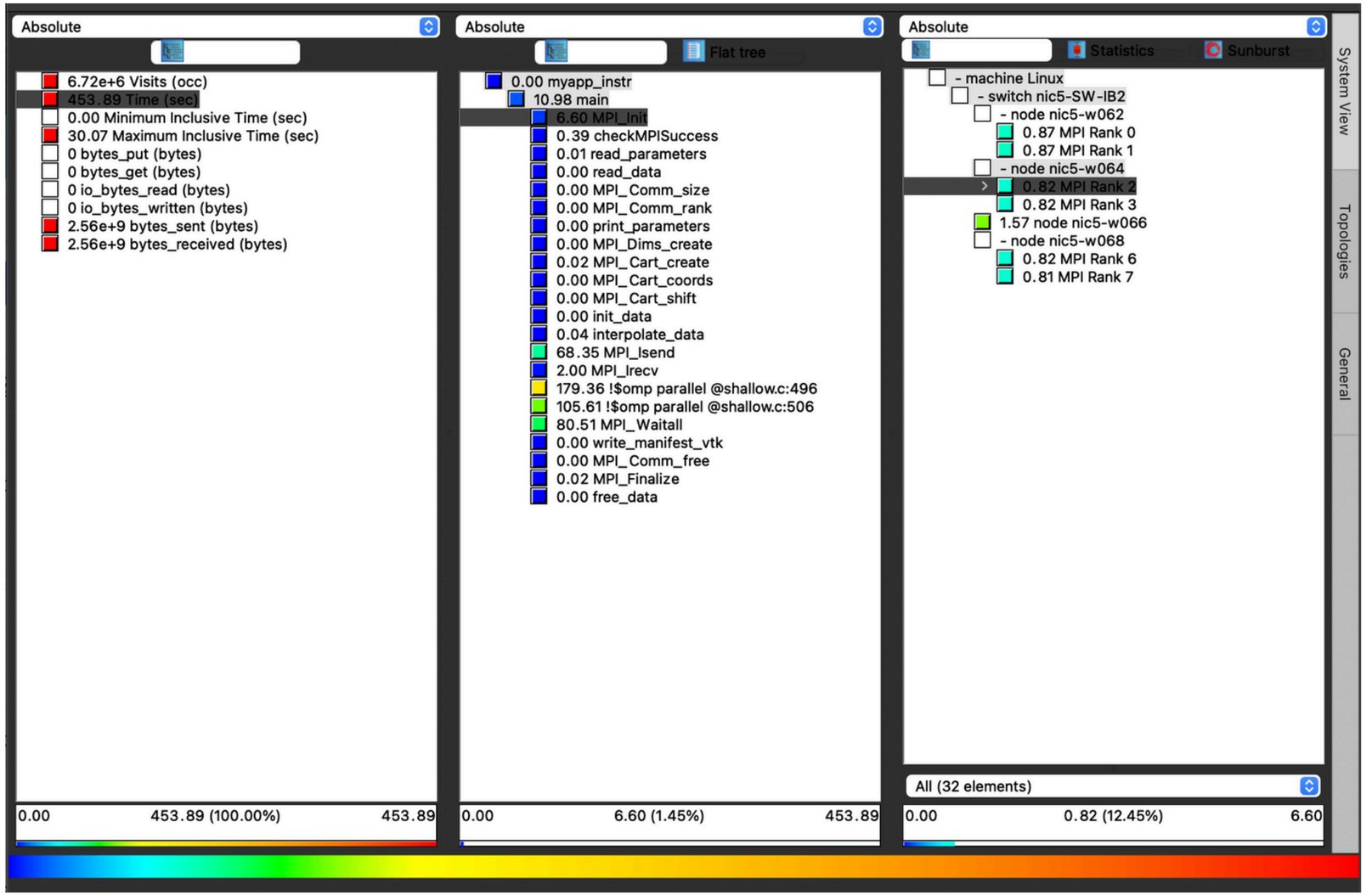






Profiling

Profiling demonstration



Graphics Processing Units (GPUs)

Mapper

- We defined a mapper for our data structure to specify the mapping existing between the CPU and GPU and how data arrays needs to be moved to/from them.

- Similar to OpenMP cpu we use **#pragma omp target teams distribute**, on our main loop to parallelize the work into the different teams on the GPU, and we don’t copied back the values computed by the GPU at each timesteps as communication between devices is a lot slower than computation.

- We now complete around 2294.25 MUpdates/s (speedup factor S = **27.28**)

Optional Enhancements

Extend physical model with Coriolis forces

Modified Explicit Scheme

Modifications of the explicit scheme ⟶ new coriolis term added

$$\frac{\eta_{i,j}^{n+1} - \eta_{i,j}^{n}}{\Delta t} = -\frac{h(\mathbf{x}^{u_{i+1,j}})u_{i+1,j}^{n} - h(\mathbf{x}^{u_{i,j}})u_{i,j}^{n}}{\Delta x} - \frac{h(\mathbf{x}^{v_{i,j+1}})v_{i,j+1}^{n} - h(\mathbf{x}^{v_{i,j}})v_{i,j}^{n}}{\Delta y}$$

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = -g\frac{\eta_{i,j}^{n+1} - \eta_{i-1,j}^{n+1}}{\Delta x} - \gamma u_{i,j}^{n} - f v_{i,j}^{n}$$

$$\frac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} = -g\frac{\eta_{i,j}^{n+1} - \eta_{i,j-1}^{n+1}}{\Delta y} - \gamma v_{i,j}^{n} + f u_{i,j}^{n}$$

Where f represents the coriolis coefficient and is computed as:

$$\mathrm{f} = 2\ \Omega sin(\phi)$$

in which the first term represents the rate of angular rotation of the earth and the second the latitude of the location

Transparent boundaries condition

In order to simulate transparent boundaries, the boundaries condition were change to **Absorbing boundaries condition**

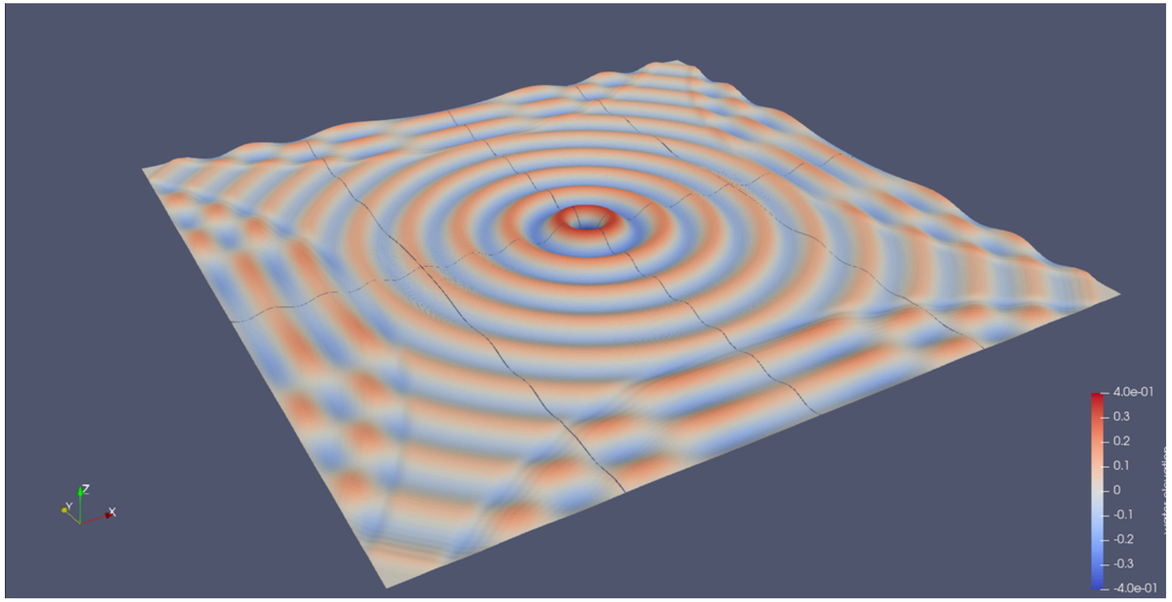
These conditions are based on the **Sommerfeld radiation condition** :

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial \eta} = 0$$

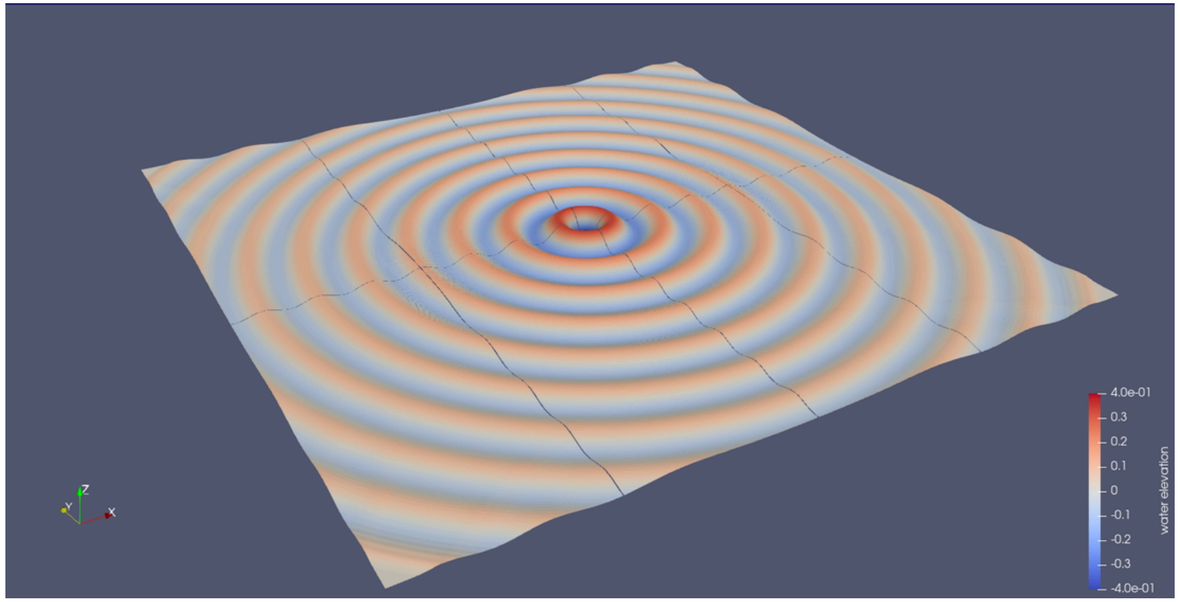
Discretizing these equations leads to :

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t.c.\frac{u_{i,j}^{n} - u_{i-1,j}^{n}}{\Delta x}$$

Visualization



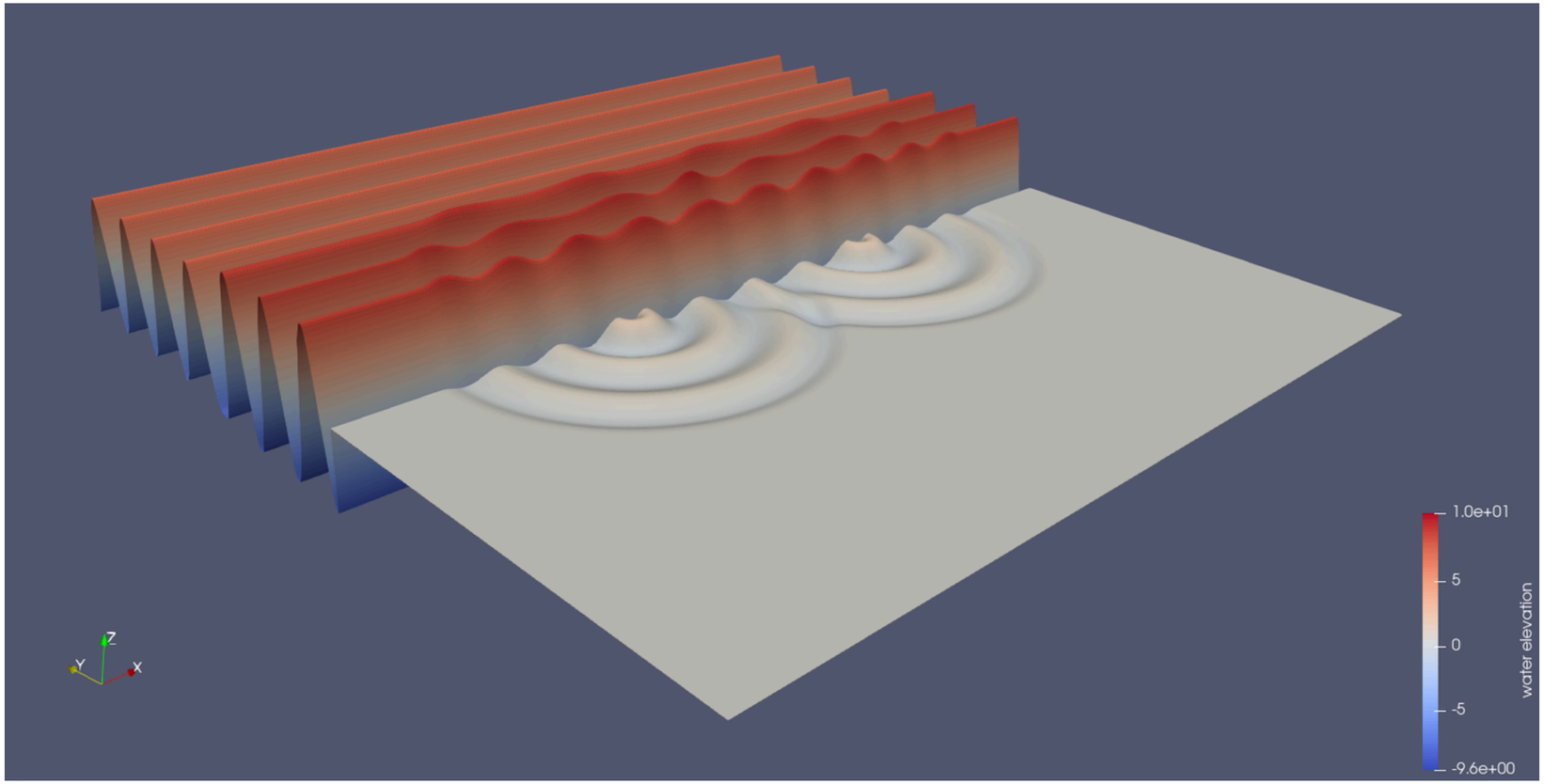
Non-transparent boundary (source 2)



Transparent boundary (source 3)

Custom implementations

Visualization



Young's interferences (source 4)