

Hybrid Strategies for the NEMO Ocean Model on Many-core Processors

I. Epicoco, S. Mocavero, G. Aloisio
University of Salento & CMCC, Italy

*2012 Programming weather, climate, and earth-system models on
heterogeneous multi-core platforms
Boulder (CO) Sept. 12-13, 2012*



Introduction

NEMO uses

- **Time discretization** based on a three level scheme where the tendency term is evaluated centered in time, forward or backward depending on the nature of the term
- **Space discretization** using the Arakawa C grid
 - vertical discretization is based on full step or partial step z- or s-coordinates
 - for tracers and momentum explicit, split-explicit and filtered free surface formulations are implemented

NEMO is characterized by:

- Intensive memory usage
- 2D MPI parallelization
- Cross communication pattern (using point-to-point calls).



Goal

Evaluating different hybrid parallel approaches for multi-core architectures:

- Reducing the communication time
- Exploiting the shared memory
- Exploiting the memory hierarchy



Test case: Advection schema for tracers

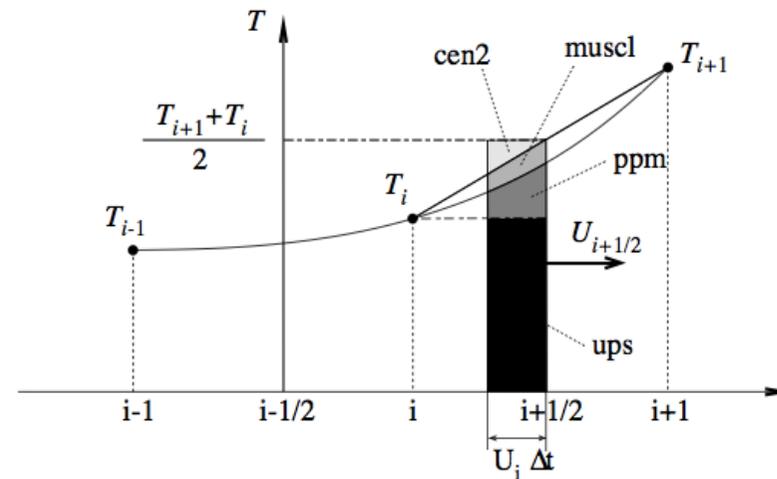
In the MUSCL formulation (traadv_muscl.f90), the tracer is evaluated at velocity points assuming a linear tracer variation between two adjacent T-points

The implementation follows this steps:

- Horizontal advective fluxes
 - First guess of the slopes
 - Boundaries exchange among processes
 - Evaluation of the slopes
 - Evaluation of the MUSCL fluxes
 - Boundaries exchange among processes
- Vertical advective fluxes
 - Evaluation of the slopes
 - Evaluation of the MUSCL fluxes

The implementation uses 3 nested loops along i, j and k directions

The MPI communication uses a cross pattern (with point-to-point calls)



Outer loop parallelization

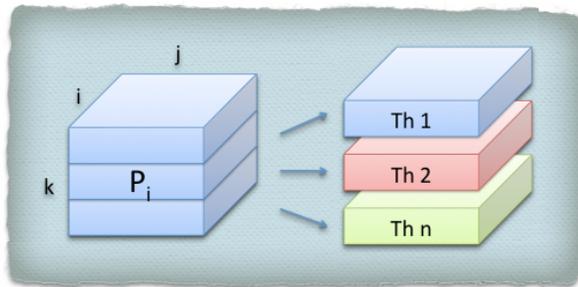
Only the loops along k are distributed among the parallel threads:

PROs:

- It is extremely simple to implement
- It introduces minimum amount of parallel overhead

CONs:

- The parallelism is limited to the number of vertical levels ($O(10^2)$)



Original code	Outer loop parallelization
<pre>DO jk = startk, endk DO jj = startj, endj DO ji = starti, endi Body (Data independency along k must be satisfied) END DO END DO END DO</pre>	<pre>!\$OMP DO DO jk = startk, endk DO jj = startj, endj DO ji = starti, endi Body (Data independency along k must be satisfied) END DO END DO END DO</pre>



Tile based parallelization

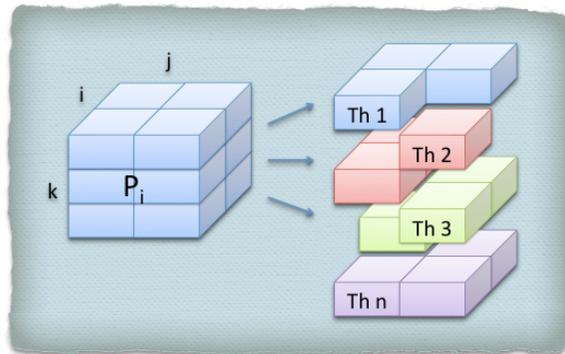
Each MPI sub-domain is divided into 3D tiles; the available tiles are distributed among the parallel threads

PROs:

- It allows a fine grained parallelism (up to a single grid point)

CONs:

- The cache misses are greater since the approach does not preserve a contiguous memory access



Original code	Tile Based Parallelization
	!definition of the total number of tiles jpb = <total number of tiles> !tile initialization DO jb = 1, jpb tile(jb)%si = <first pos. along i> tile(jb)%ei = <last pos. along i> tile(jb)%sj = <first pos. along j> tile(jb)%ej = <last pos. along j> tile(jb)%sk = <first pos. along k> tile(jb)%ek = <last pos. along k> END DO
	!\$OMP PARALLEL
	!\$OMP DO
DO jk = startk, endk DO jj = startj, endj DO ji = starti, endi	DO jb = 1, jpb DO jk = tile(jb)%sk, tile(jb)%ek DO jj = tile(jb)%sj, tile(jb)%ej DO ji = tile(jb)%si, tile(jb)%ei
<i>Body</i> <i>(Data independency along i,j,k</i> <i>must be satisfied)</i>	<i>Body</i> <i>(Data independency along i,j,k</i> <i>must be satisfied)</i>
END DO END DO END DO	END DO !ji loop END DO !jj loop END DO !jk loop END DO !jb loop



Loops Merge parallelization

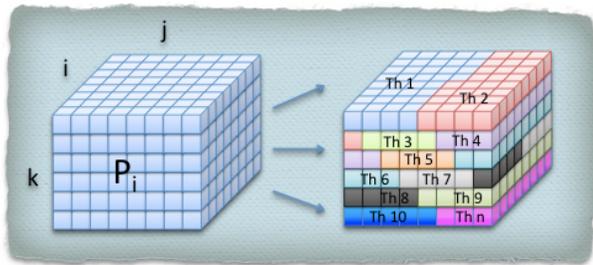
The 3 nested loops are merged together and all of the iteration are distributed among the threads

PROs:

- It allows a fine grained parallelism (up to a single grid point)

CONs:

- It requires a deep revision of the arrays indexes



Original code	Loop Merge
<pre>DO jk = startk, endk DO jj = startj, endj DO ji = starti, endi a(ji,jj,jk) = b(ji,jj,jk) - c(ji,jj,jk) (Data independency along i,j,k must be satisfied) END DO END DO END DO</pre>	<pre>jpelemi = (endi - starti + 1) jpelemj = (endj - startj + 1) jpelemk = (endk - startk + 1) jpelem = jpelemi * jpelemj * jpelemk !\$OMP DO DO jx = 1, jpelem a(jx,1,1) = b(jx,1,1) - c(jx,1,1) END DO</pre>

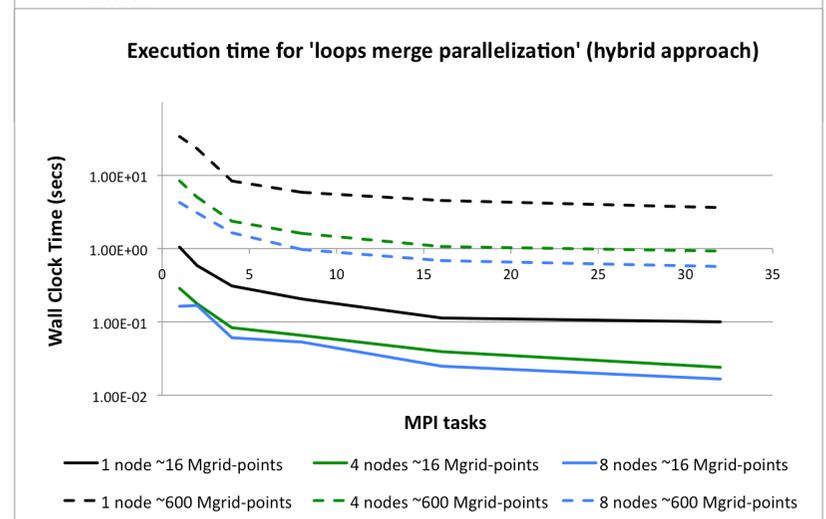
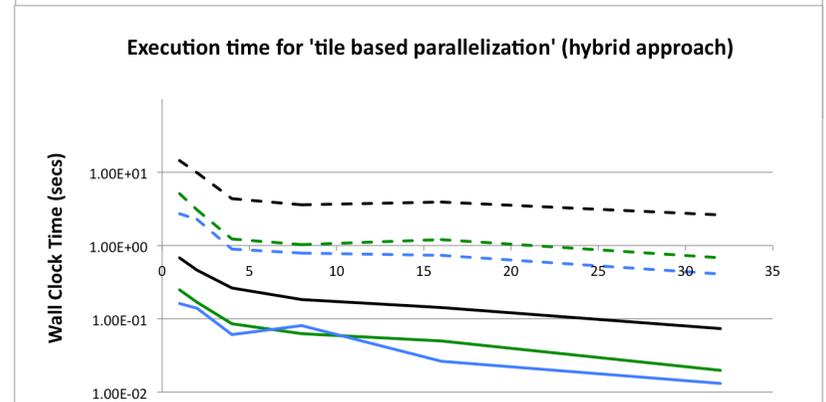
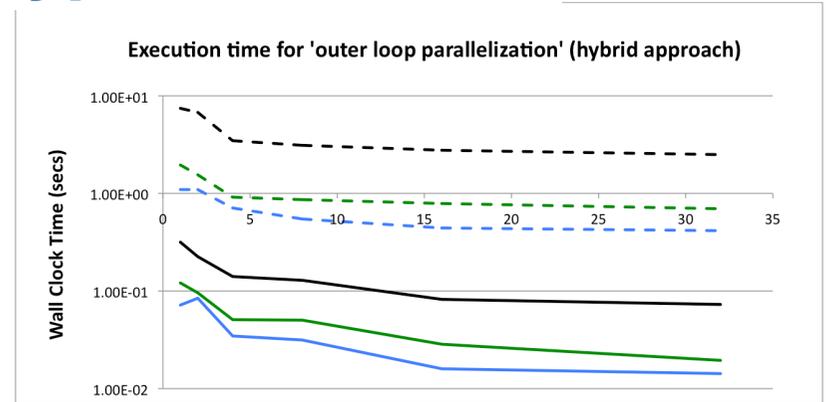
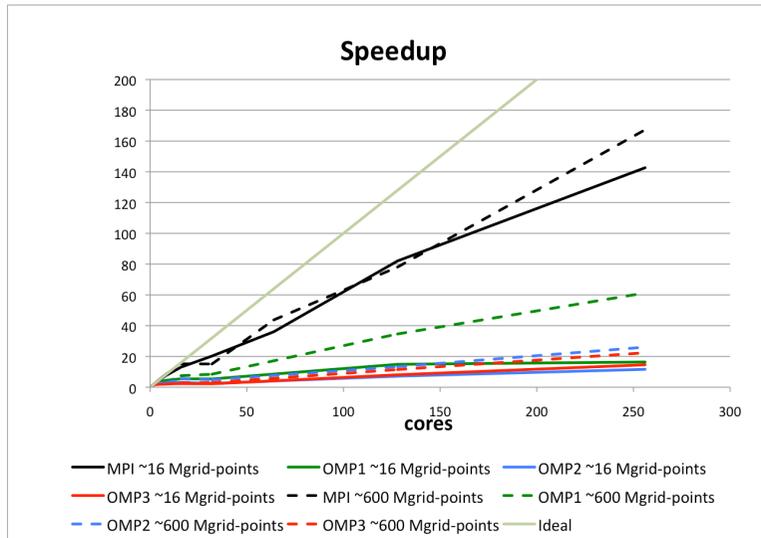
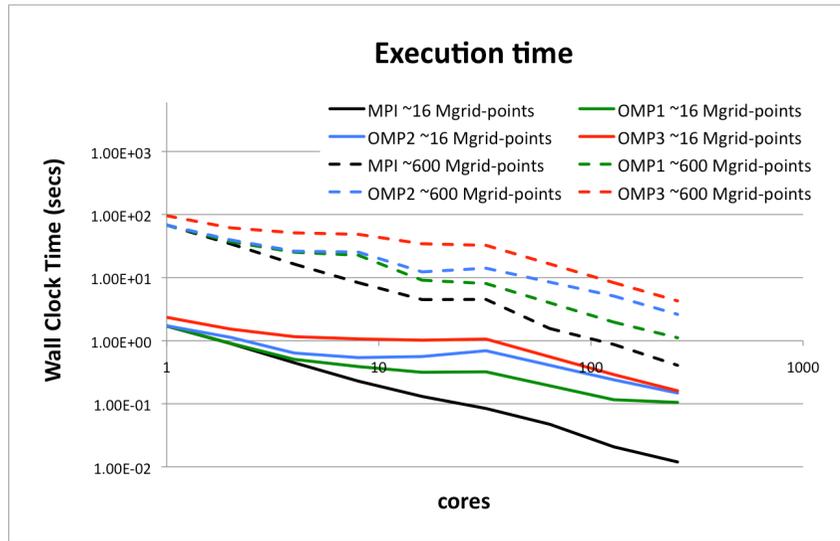


Architectures

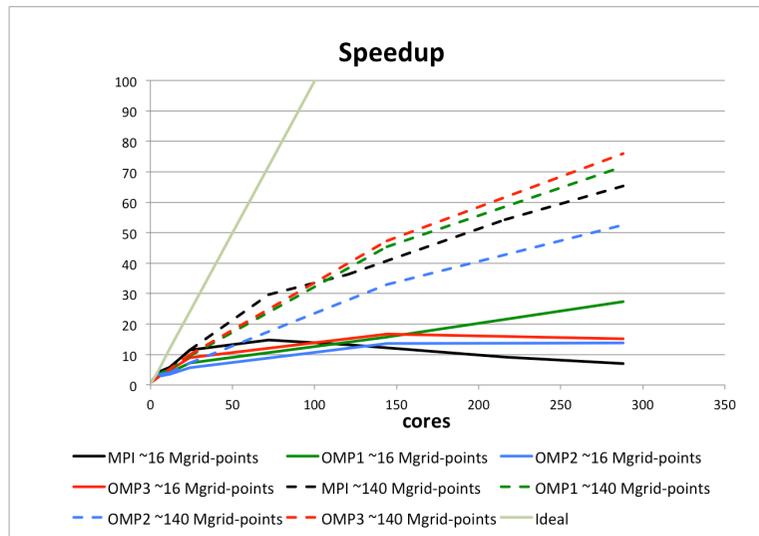
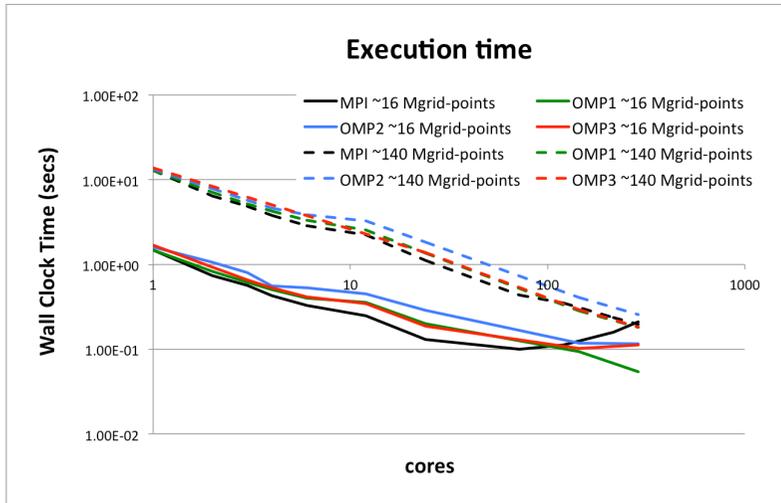
- IBM P6 (Calypso at CMCC)
 - P575 nodes with 16 dual-core chips
 - 4 QUAD with 4 processors each
 - Memory: 128GB per node
 - Infiniband 4x DDR interconnection
- IBM iDataPlex (PLX at Cineca)
 - iDataPlex M3 nodes with 2 6-core chips
 - Memory: 48GB per node
 - Interconnection: Infiniband with 4x QDR switches
- IBM PPC970 (MareNostrum at BSC)
 - P615 nodes with 2 dual-core processor
 - Memory: 8GB per node
 - Interconnection: Myrinet Network
- CRAY XE6 (Hector at STFC)
 - XE6 nodes with 2 16-core chips
 - 4 NUMA regions with 8 cores each
 - Memory: 32GB per node
 - Interconnection: Cray Gemini communication chips



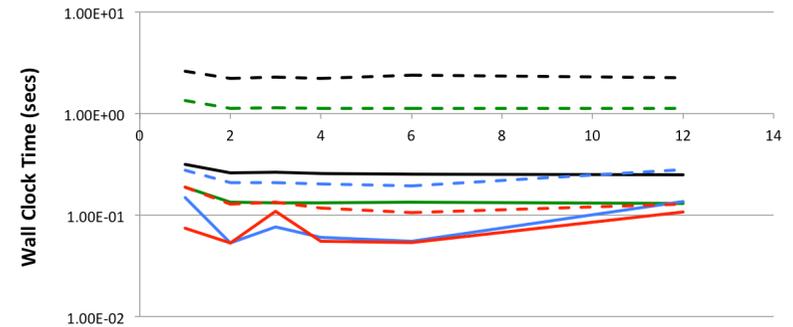
Results on IBM P6 – Calypso at CMCC



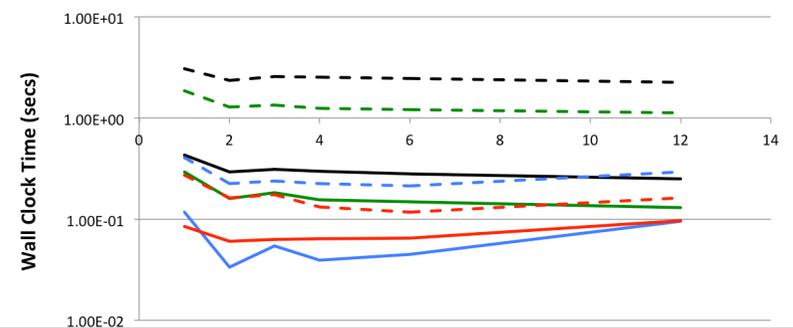
Results on IBM iDataPlex – PLX at CINECA



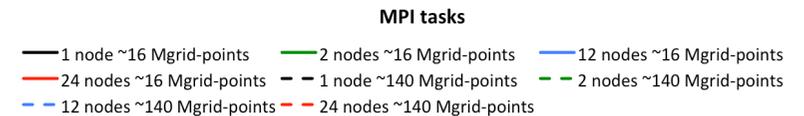
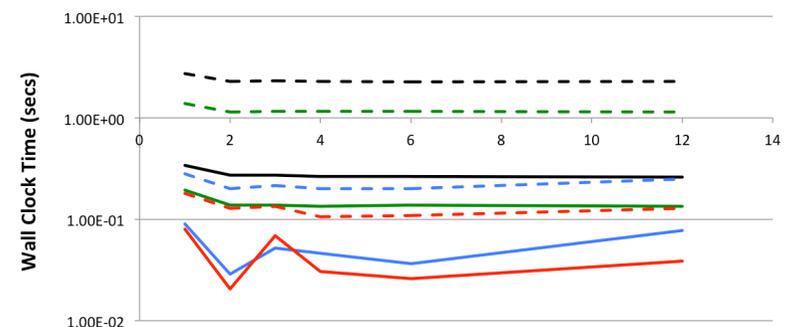
Execution time for 'outer loop parallelization' (hybrid approach)



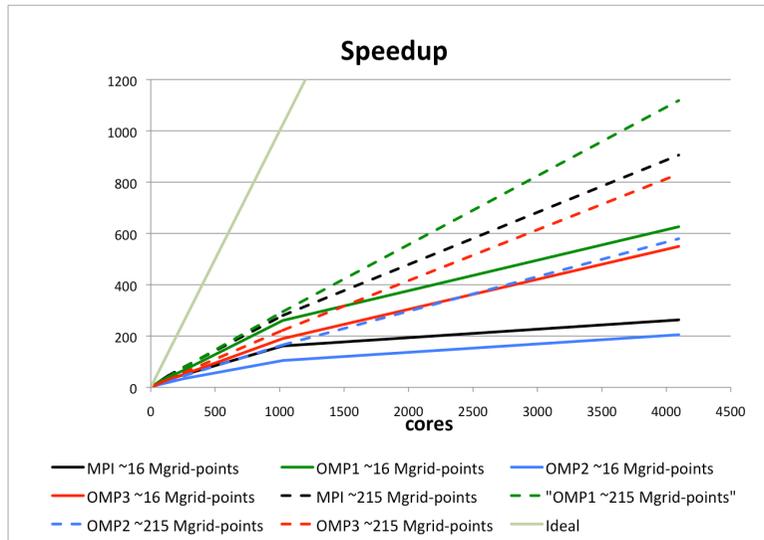
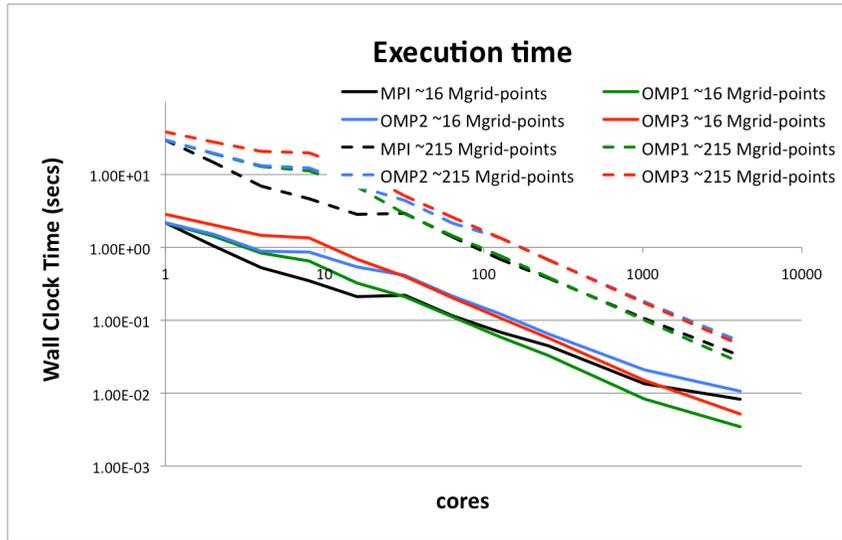
Execution time for 'tile based parallelization' (hybrid approach)



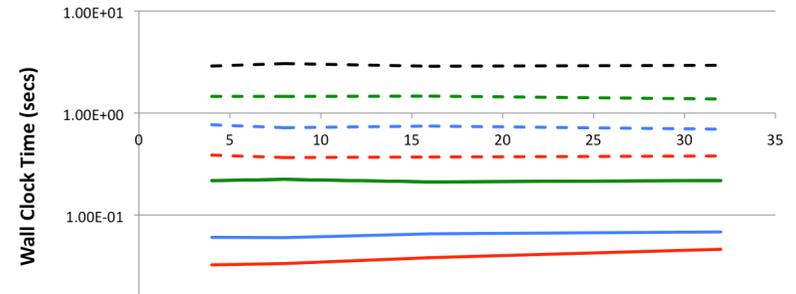
Execution time for 'loops merge parallelization' (hybrid approach)



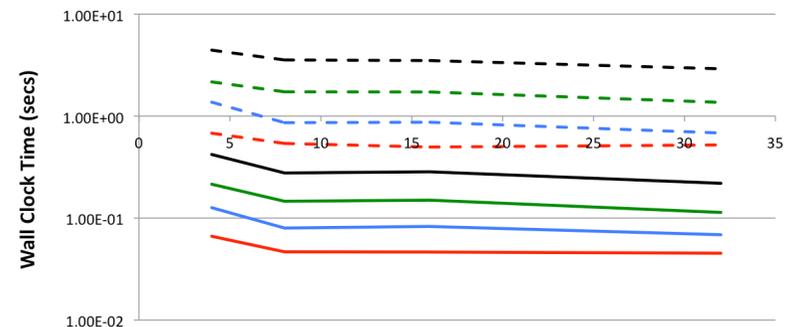
Results on CRAY XE6 – HECToR at STFC



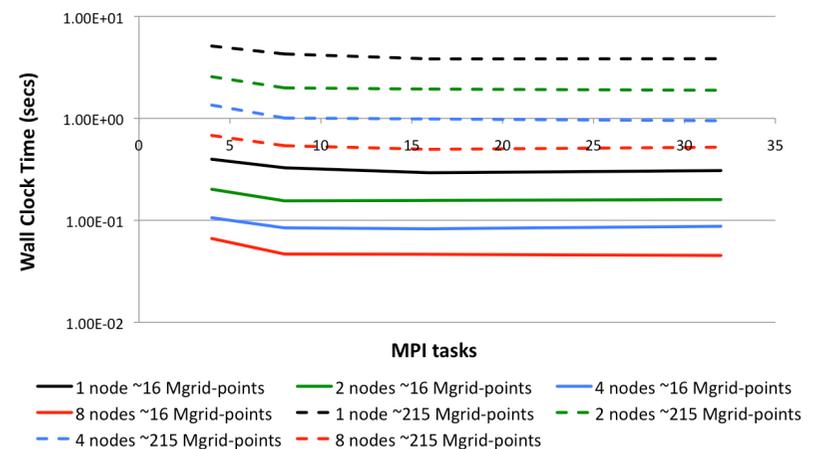
Execution time for 'outer loop parallelization' (hybrid approach)



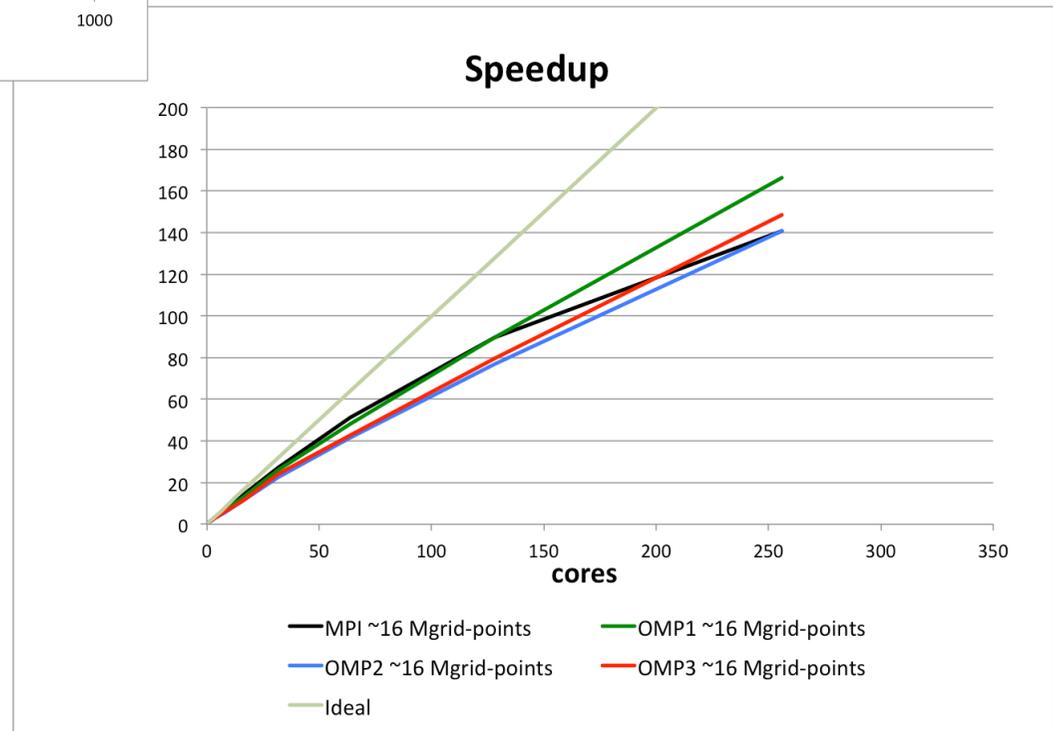
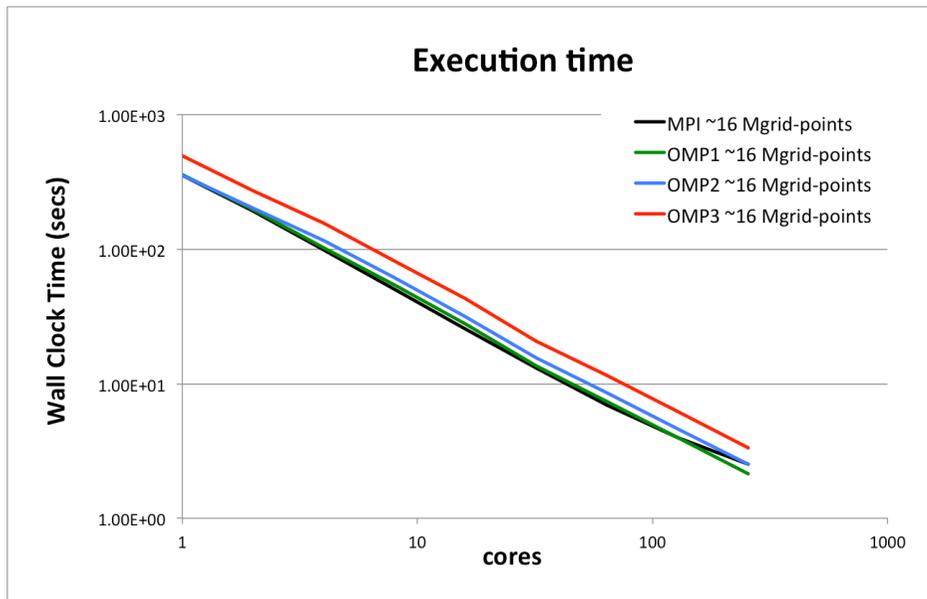
Execution time for 'tile based parallelization' (hybrid approach)



Execution time for 'loops merge parallelization' (hybrid approach)



Results on IBM PPC970 – MareNostrum at BSC



Conclusion

- The OpenMP parallelization provides a further level of parallelism (along vertical levels) in a relatively easy way
- The benefit of OpenMP parallelization can be appreciated only for high level of parallelism
- The benefit given by the hybrid parallelization strongly depends by the architecture and by the MPI implementation
- The bottleneck for the analyzed advection schema is due to the weight of the communication time over the computing time. The communication time decreases linearly with the dimension of the sub-domain, the computing time decrease squarely

Next Steps

- To optimize the memory access



Credits

- Italo Epicoco - CMCC
- Silvia Mocavero - CMCC
- Andrew Porter - STFC
- Stephen Pickles - STFC
- Mike Ashworth - STFC
- Giovanni Aloisio - CMCC



Thanks

