North Fold optimization

I.Epicoco, S.Mocavero, G.Aloisio
Scientific Computing and Operation (SCO) Division
CMCC, Italy

NEMO version: v3.5
Date: May 2013

Abstract
In the following we report the optimization of the north-fold algorithm carried out at CMCC/SCO (Scientific Computing & Operations Division) with some preliminary insights on Vesta (BG/Q @ANL) and Athena (iDataplex system@CMCC).

Introduction
The folding of points at north of the domain handles the north boundary of a three-polar ORCA grid. Such a grid has two poles in the northern hemisphere (see Fig.1).

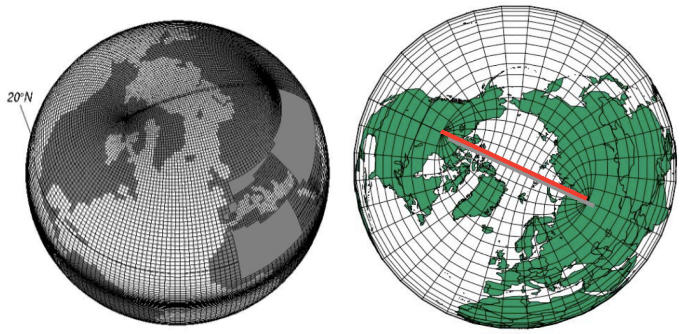

Fig.1: tripolar ORCA grid

The folding slightly differs depending on the nature of the points (T-, U-, V-, F-point) to be folded and on the pivoting point (T- or F-point pivot). More in general the folding is achieved considering only the last 4 rows on the top of the global domain and by applying a rotation pivoting on the point in the middle. During the folding, the point on the top left is updated with the value of the point on bottom right and so on (see Fig. 2).

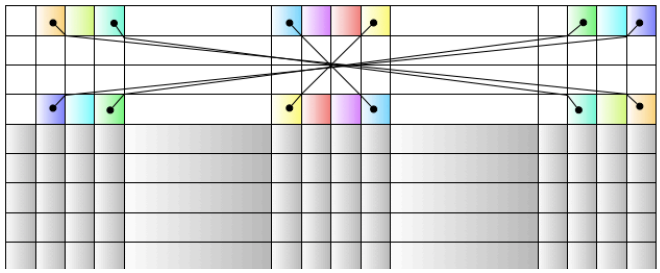
Fig. 2: north-fold algorithm

If we consider *jpiglo* the total number of column in the global domain, the sequential time is proportional to *jpiglo*.
The current version of the parallel algorithm is based on the domain decomposition. Each MPI process takes care of a block of points. Each process can update its points using values belonging to the symmetric process. An MPI_Sendrecv communication is used for sending *jpi* point to the corresponding process. *jpi* is exactly the number of column assigned to each process. If we consider the MPI processes disposed in a grid of *jpni x jpnj*, the following relation can be used to identify the pairs of processes that must exchange its north points for the north fold.

$$P(i, jpnj) \leftrightarrow P(jpni - i + 1, jpnj) \qquad \forall i,\ 1 \le i \le jpni$$

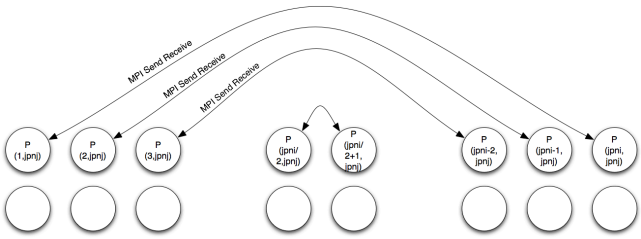

Fig. 3: communication pattern for the north-fold

In the current implementation, each received message is placed in a buffer with a number of element equal to the total dimension of the global domain (*jpiglo* elements).
Each process sweeps the entire buffer, but only a part of that computation is really useful for the process' sub-domain, resulting in a bunch of redundant operations that can be avoided. Since each process sweeps a buffer with *jpiglo* elements the parallel time of the algorithm is still proportional to the dimension of the global domain.

Optimization
In our optimization we reduced the buffer length only to the number of elements actually needed by the receiving process. The number of elements needed are exactly the dimension of the sub-domain. We avoid redundant operations and the parallel time is now proportional to the problem size and inverse proportional with the number of processes.

Performance Evaluation
We evaluated the benefit coming from the optimization, using a high resolution configuration named GLOB16 on two architecture: BG/Q named Vesta located at the ALCF – Argonne National Laboratory (ANL); iDataPlex with SandyBridge named Athena located at CMCC Supercomputing Center.

MODEL SETUP	**GLOB16**
Resolution	1/16° ~7Km
Grid dimension	5762 x 3133 x 100
Number of grid points	1800M
Time Step	2.5 min
Memory usage	~ 1.2 TB

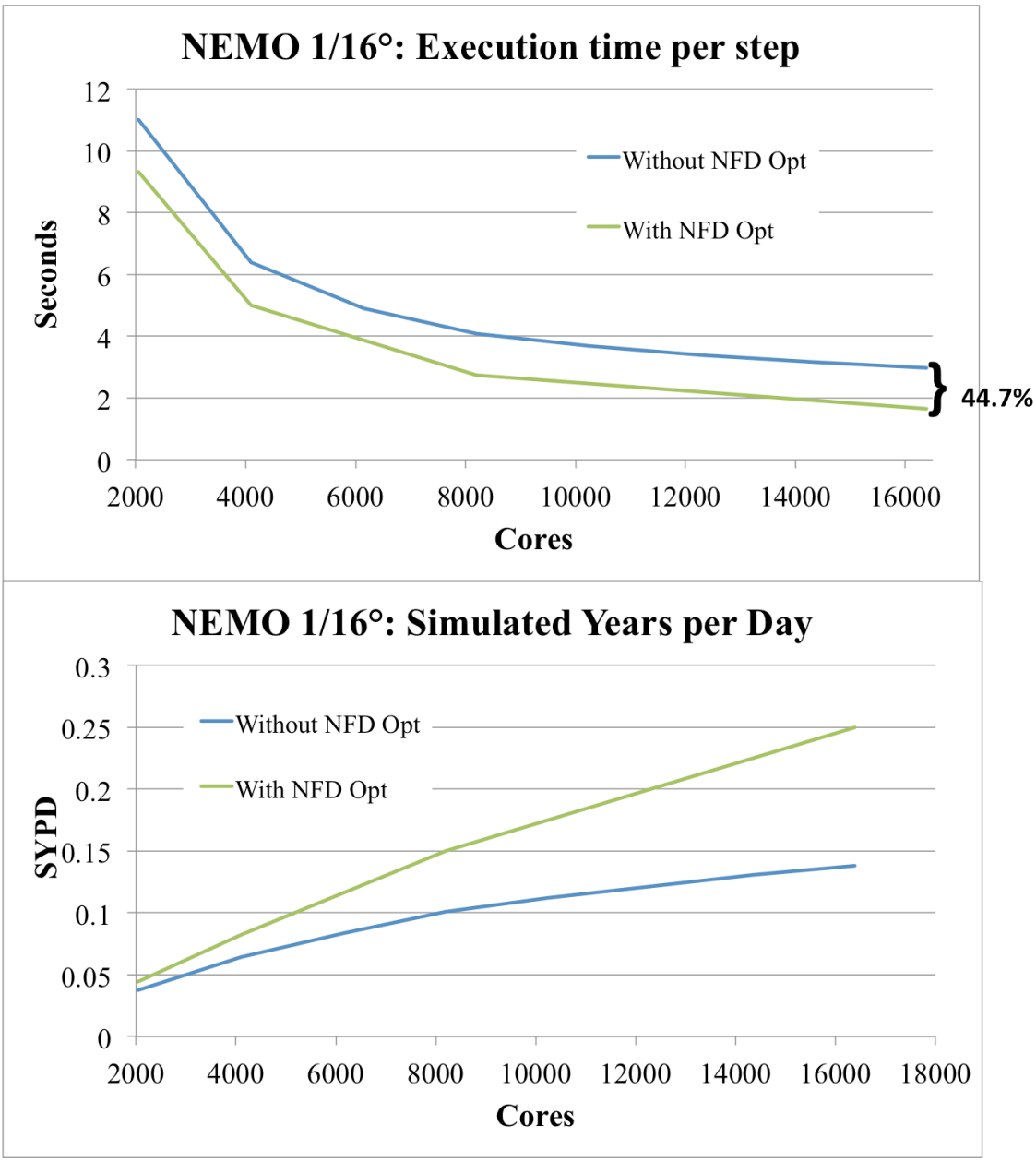

Fig. 4: Execution time and SYPD of NEMO in configuration GLOB16 on BG/Q Vesta (ANL)

On BG/Q, the optimization produced an improvement of 44.7% of in the execution time per step on 16K cores. The code scales well up to 16K cores with a parallel efficiency of 70.9%. We could scale up to 32K cores.

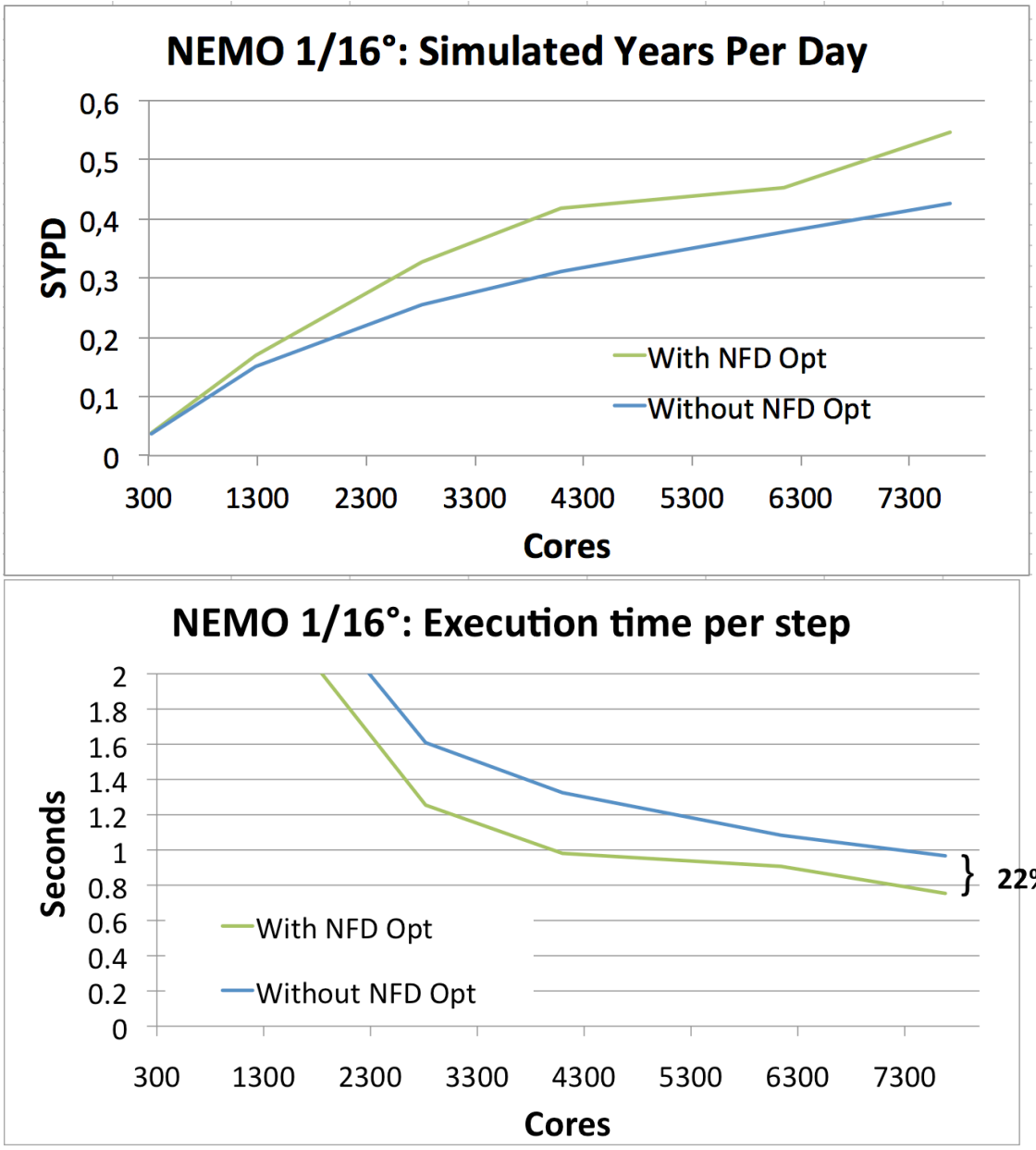

Fig. 5: Execution time and SYPD of NEMO in configuration GLOB16 on Athena (CMCC)

On iDataPlex, the optimization produced an improvement of 22% in the execution time per step on more than 7K cores. The code scales well up to 7K cores with a parallel efficiency of 59.5%. The SYPD is still very low, half a year per day.

A brief comparison between BG/Q and iDataPlex shows that on Athena we can simulate half year per day using 7Kcores while on Vesta we got a maximum of 0.25 years per day with 16K core. The main reason is due to the SMT exploitation: since the code is pure MPI, SMT is not exploited at all. A hybrid parallelization of NEMO will better suite on many core architectures.