Refactoring the memory access pattern to improve computational performance in NEMO


OS4.8 - Numerical modelling of the ocean: new scientific advances in ocean models to foster exchanges within NEMO community and contribute to future developments
NEMO Ocean Model

Single core performance of the NEMO model is limited by memory access and poor exploitation of vector processing units on modern HPC architectures.

The analysis of the memory access pattern shows that many repeated accesses occur for reading values not available in cache.

-> high rate of cache miss!!!

Goal of the work: enhance the exploitation of the cache memory of the modern parallel architectures through the loop fusion approach.
Single core performance - Loop fusion

- **Loop fusion technique** aims at better exploiting the cache memory by fusing DO loops together

  ```
  DO j=1, n-1
      DO i=1, n
          b(i,j) = in(i,j+1) - in(i,j)
      END DO
  END DO
  DO j=2, n-1
      DO i=1, n
          out(i,j) = b(i,j) - b(i,j-1)
      END DO
  END DO
  ```

- advantages: reduction of cache misses and reduction of the memory footprint
- disadvantage: due to data dependencies redundant operations are needed
Loop fusion approach has been applied on the NEMO MUSCL advection kernel

Three different levels of fusion have been implemented

- **prototype1**: has the maximum level of fusion with redundant operations
- **prototype2**: introduces the buffers rotation in the outer loop
- **prototype3**: uses the buffers rotation in the outer and middle loop

buffers rotation technique avoid redundant operations by adopting pointers to arrays and implementing a rotation at each loop iteration as shown in the figure
First approach – prototype 1

- The halo exchange is moved before all DO-loops (at the beginning of the routine)
  - The halo region needs to be extended up to two halo lines
- The advective trend is computed for each single \((ji, jj, jk, jn)\) grid cell within a single big DO-loop
  - This approach implies also a duplication of the calculus up to a factor 3
First approach – prototype 1

BaseLine

Prototype 1

# shareEGU20 - Refactoring the memory access pattern to improve computational performance in NEMO
Along the vertical direction:

- The advective flux at level $jk=1$ is computed before the loop over $jk$ (we call this flux $F_{jk-1}$)
- Inside the $jk$ loop, the flux at level $jk$ (which we call $F_{jk}$) is computed
- We use $F_{jk-1}$ and $F_{jk}$ to calculate the advective trend at level $jk$ and to update the RHS variable at level $jk$
- Before incrementing the $jk$ level, we update the flux at level $jk-1$: $F_{jk} \rightarrow F_{jk-1}$

This approach reduces the number of redundant operations, but it introduces a data dependencies in the $jk$ loop, hence the $jk$ loop cannot be vectorized, neither executed in parallel.
Second approach – prototype2

Prototype 1

Prototype 2

# shareEGU20 - Refactoring the memory access pattern to improve computational performance in NEMO
Along the horizontal direction:

- In the \(jk\) loop, before updating the RHS variable with the advective trend, the fluxes for the whole horizontal domain are computed.
- The fluxes are hence used to compute the advective trend and to update the RHS variable.
- This approach further reduces the number of redundant operations.
Third approach – prototype 3

Prototype 2

Prototype 3

# shareEGU20 - Refactoring the memory access pattern to improve computational performance in NEMO
Preliminary Performance Analysis

- Global domain: 2240 x 1500 x 31 points
- Smallest sub-domain (with 1024 cores): 74 x 51 x 31 points

Test executed on Intel Xeon based Architecture
16 cores per node

Elapsed Time

Prototypes Speedup
Preliminary Performance Analysis

- Global domain: 2240 x 1500 x 31 points
- Smallest sub-domain (with 1024 cores): 74 x 51 x 31 points

Parallel Speedup

Parallel Efficiency

70%
Preliminary Performance Analysis

- Global domain: 2240 x 1500 x 31 points
- Smallest sub-domain (with 1024 cores): 74 x 51 x 31 points
Preliminary Performance Analysis

- Global domain: 70 x 46 x 19 points
- Sub-domain with 64 cores: 13 x 10 x 19 points

![Elapsed Time Graph](chart1.png)

<table>
<thead>
<tr>
<th>number of processes</th>
<th>Elapsed time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>4</td>
<td>0.004</td>
</tr>
<tr>
<td>16</td>
<td>0.016</td>
</tr>
<tr>
<td>64</td>
<td>0.064</td>
</tr>
<tr>
<td>256</td>
<td>0.256</td>
</tr>
<tr>
<td>1024</td>
<td>1.024</td>
</tr>
</tbody>
</table>

![Prototypes Speedup Graph](chart2.png)

<table>
<thead>
<tr>
<th>number of processes</th>
<th>Prototypes Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.06</td>
</tr>
<tr>
<td>4</td>
<td>1.12</td>
</tr>
<tr>
<td>8</td>
<td>1.16</td>
</tr>
<tr>
<td>16</td>
<td>1.20</td>
</tr>
<tr>
<td>32</td>
<td>1.24</td>
</tr>
<tr>
<td>64</td>
<td>1.30</td>
</tr>
<tr>
<td>128</td>
<td>1.36</td>
</tr>
<tr>
<td>256</td>
<td>1.44</td>
</tr>
<tr>
<td>512</td>
<td>1.60</td>
</tr>
<tr>
<td>1024</td>
<td>1.80</td>
</tr>
</tbody>
</table>

# shareEGU20 - Refactoring the memory access pattern to improve computational performance in NEMO
Preliminary Performance Analysis

- Global domain: 70 x 46 x 19 points
- Sub-domain with 64 cores: 13 x 10 x 19 points

Parallel Speedup

Parallel Efficiency
• Prototypes 1 and 2 provide a good improvement up to 256 cores then the redundant operations lead to a loss of performance
• Prototypes 3 improves parallel efficiency by ~30% on 1024 cores
• This approach enhanced the vectorization level and the cache reuse, reducing L3 Total Cache Misses by ~80% on 1024 cores
• Loop-fusion is strictly linked to the computing architecture → A fully portable performance improvement can be ensured by the adoption of a DSL.