Scalability of BATOR:

A problem of strong scalability?

Ryad El Khatib (CNRM/GMAP)
INTRODUCTION
- Presentation of BATOR software
- Characteristics of BATOR compared to AROME

STUDIES
- Improvements and limits of the scalability of today
- Software performance
- Other parallelisations algorithms

CONCLUSIONS
- Recommendations for Bator and softwares in general
- Perspectives for Bator in particular
Application to transform the collected observations over the planet into a database of the « ODB » format, suitable for ARPEGE, ALADIN, AROME

First task on the critical path of an assimilation suite 3DVar (AROME) or 4DVar (ARPEGE)

Mechanism: several executions of the applications in order to transform sets of observations files delivered in different formats (BUFR mainly) and different sizes
### « Anatomy » of BATOR as used for AROME (3DVar)

<table>
<thead>
<tr>
<th>Kind of observations or instrument</th>
<th>Number of files</th>
<th>Format</th>
<th>Size (Mb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>1</td>
<td>OBSOUL</td>
<td>1</td>
</tr>
<tr>
<td>Wind profilers + GPS</td>
<td>2</td>
<td>OBSOUL</td>
<td>1</td>
</tr>
<tr>
<td>Conventional</td>
<td>1</td>
<td>OBSOUL</td>
<td>7</td>
</tr>
<tr>
<td>SEVIRI</td>
<td>1</td>
<td>GRIB</td>
<td>18</td>
</tr>
<tr>
<td>HIRS</td>
<td>1</td>
<td>BUFR</td>
<td>2</td>
</tr>
<tr>
<td>AMSUA</td>
<td>1</td>
<td>BUFR</td>
<td>1</td>
</tr>
<tr>
<td>AMSUB</td>
<td>1</td>
<td>BUFR</td>
<td>4</td>
</tr>
<tr>
<td>SSMI</td>
<td>1</td>
<td>BUFR</td>
<td>3</td>
</tr>
<tr>
<td>IASI</td>
<td>1</td>
<td>BUFR</td>
<td>13</td>
</tr>
<tr>
<td>Geowind</td>
<td>1</td>
<td>BUFR</td>
<td>2</td>
</tr>
<tr>
<td>ERS + ASCAT</td>
<td>2</td>
<td>BUFR</td>
<td>1</td>
</tr>
<tr>
<td>AIRS</td>
<td>1</td>
<td>BUFR</td>
<td>0</td>
</tr>
<tr>
<td>RADAR</td>
<td>24</td>
<td>BUFR</td>
<td>200</td>
</tr>
</tbody>
</table>
## Characteristics of BATOR

<table>
<thead>
<tr>
<th>BATOR vs AROME</th>
<th>BATOR (without ODB)</th>
<th>AROME forecast (3h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lines of code</td>
<td>≈ 7 000</td>
<td>≈ 1 600 000</td>
</tr>
<tr>
<td>MPI parallelisation</td>
<td>Oui mais inefficace</td>
<td>Oui</td>
</tr>
<tr>
<td>Open-MP parallélisation</td>
<td>Non</td>
<td>Oui</td>
</tr>
<tr>
<td>CPUs used in operations</td>
<td>1</td>
<td>16 (SX9)</td>
</tr>
<tr>
<td>Elapse time</td>
<td>≈ 500 s.</td>
<td>≈ 500 s.</td>
</tr>
<tr>
<td>Memory per CPU</td>
<td>15 Go</td>
<td>11.5 Go (SX9)</td>
</tr>
<tr>
<td>Static memory allocated</td>
<td>≈ 600 Mo</td>
<td>≈ 400 Mo</td>
</tr>
</tbody>
</table>

### Impact of the hardware architecture

<table>
<thead>
<tr>
<th></th>
<th>NEC SX9 Vector machine</th>
<th>Intel Xeon Scalar machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapse time</td>
<td>≈ 500 s.</td>
<td>≈ 180 s.</td>
</tr>
</tbody>
</table>
Load balancing of the BATOR tasks

The task devoted to the 24 radars files is dramatically proeminent

=> In such conditions, the scalability is near to zero
What if we maximize the number of tasks? (1 task per file)

BATOR / AROME temps d'exécution par tâche (s.)

BATOR / AROME - Coût mémoire (Mo)

=> Will a external dynamic load balancing be enough?
Limits of the scalability with a dynamic load balancing

- Limited because the number of observations files is limited ('36' wall)
- **Scalability loss because of residual load imbalance**
  (we can't run faster than the slowest task : red line)

*And also :*
- Relatively high memory cost per task
- Memory-anti-scalable parallelisation scheme

Practically : beyond 16 processors, the ressources at disposal is critical
How to cross this scalability barrier?

Jump over the obstacle?

- Increase the number of observations files??
  - In 4DVar: slice the files into shorter time slots
  - Cut the files per geographical sub-area?
  - Define a better-adapted file format?

- However, handling many small files may not be the best solution

System time vs / Real time

![System time vs / Real time chart]

- | Number of tasks | 1 | 4 | 8 | 12 | 16 |
  - | system | 100% | 80% | 60% | 40% | 20% |
  - | real | 0% | 20% | 40% | 60% | 80% | 100% |
How to cross this scalability barrier?

Turn around the obstacle, looking for better performance?

- **Efficiency** may contribute to improve the **Scalability**
  - Are the files read/written efficiently?
  - Does the algorithm fit parallel machines?
  - Is the code performant?

<table>
<thead>
<tr>
<th>Avg-%</th>
<th>Avg.time</th>
<th># of calls</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.65%</td>
<td>9.697</td>
<td>13312</td>
<td>BUEXS3</td>
</tr>
<tr>
<td>26.40%</td>
<td>6.002</td>
<td>7</td>
<td>Bator_lbufr_radar</td>
</tr>
<tr>
<td>7.89%</td>
<td>1.795</td>
<td>1</td>
<td>BATOR</td>
</tr>
<tr>
<td>4.87%</td>
<td>1.107</td>
<td>1</td>
<td>BATOR_ELIM</td>
</tr>
</tbody>
</table>

=> An obvious efficiency issue in decoding radar BUFR files
=> Subroutine Bator_lbufr_radar to be further examined
How to cross this scalability barrier?

Turn around the obstacle
Using other directions in parallelism?

- BUFR decoding library uses **global variables**
  - => To use **Open-MP** one should modify the software
- Bator algorithm is **intrinsincally sequential**
  - => To use **Open-MP** one should revisit the algorithm
- Bator contains a lot of **loops left by GOTO instructions**
  - => Difficult to analyse the code performance and implement **Open-MP**. The code has to be modified.
- **MPI parallelisation** in dans Bator: it exists but:
  - Parallelism based on the distribution of a set of input observations files
  - => No treatment of **memory load balancing**
  - => No treatment of **CPU load balancing**
  - => finally less efficient than the external dynamic parallelisation
Another unexpected issue

The number of observations pools should be a multiple of the number of MPI tasks in the subsequent applications (Screening, Minimization)

- ODB_IO_METHOD=1
  - 1 file per table and per pool
  - => would lead to much small files on many-processors machine. Is the file system ready to support this?

- ODB_IO_METHOD=4
  - Less files of fixed size
  - => Requires (much) more memory. May easily break the memory limit of a node with Bator on a scalar machine

- Alternative # 1 : ODB_IO_METHOD=1 + tool « Odb1to4 »
- Alternative # 2 : ODB_IO_METHOD=4 + « reshuffling » (needs a specific ODB recompilation)
Conclusions

- Bator exhibits strong scalability issues than, could be overcome:
  - Better I/O conditionning (format, number of files)
  - Parallelisation methods (MPI, threads) using algorithms adapted to the problem
  - Playing with ODB tools

- The search for **scalability** should not mask the **performance issue**

- Softwares should **evolve** permanently according to its **context of execution**, not its own being:
  - « High Performance Computation » => *batch processing* (« vectorization »)
  - Evolution of programing languages, hardware architectures
  - Software context (3DVar, 4DVar for Bator, OOPS later on)
Perspectives for Bator

- Scalability and performance issues for Bator/AROME could be solved for short or mid term:
  - Thanks to a sufficient external parallelisation
  - Because the enhancement of performance (Bufrdc) seems feasible

- Bator/AROME-3DVar solution is extensible to 4DVar

- Ongoing: Fusion of ECMWF Bufr2odb with Bator
  - Full parallelisation support from Bufr2Odb
  - Get the software out of the critical path thanks to an earlier upstream execution
  - Object-oriented context for 3DVar/4DVar?