Can the compiler make the difference?

R. El Khatib (Météo-France)
Reasons for this talk

- ECMWF said that in IFS benchmark, the Cray compiler was the best performing one.
- Intel is used to saying that its Intel compiler is the best performing one in benchmarks in general.
- HIRLAM reported that for AROME, Gfortran compiler is better performing than the Cray compiler.

... Who is right???
A unique opportunity

- ECMWF has got a Cray computer with both 3 compilers installed:
  - Cray
  - Intel
  - Gfortran

- Cycle 41 has been ported on Cray by ECMWF

There is a unique opportunity to make the comparison of compilers from the point of view of operational forecasting, not from the point of view of vendors benchmarks.
Plan

- Conditions of the comparisons
- Results on cycle 41
- Explanations on the differences of performances
  - What is happening
  - What to do
- New results after optimizations (cycle 41T1+)
- Conclusion
Background conditions for a fair comparison

- Make sure the scientific results are valid with respect to a reference set of spectral norms (like for vendors benchmarks)
- Make sure the scientific results are bitwise repeatable and reproducible
- Same MPI library: MPI Cray 6.3.1, compiled for each compiler
- Same MPI driver: mpiauto, by Ph. Marguinaud (*)
- Scientific library: OpenBLAS for Intel and GNU compilers, Cray SCI library for Cray(**)
- No use of a Cray paging module for the Cray compiler
- No hyperthreading
- Same striping (=4) for the file system Lustre

(*) mpiauto performs as well, or better (with hyperthreading on Intel) than aprun
(**) I was too lazy to recompile OpenBlas for Cray, but the impact is neutral
Selection of compiler versions and options

- Use the most recent versions of compilers:
  - Cray 8.2.7
  - Intel 15.0
  - GCC 4.9

- (Try to) use equal and optimal compiler options:
  - Cray (*) : -hflex_mp=conservative -hadd_paren -hfp1
  - Intel (**) : -align array64byte -fp-model source -ftz -O2 -xAVX -finline-functions -finline-limit=500
  - Gfortran : -fstack-arrays -O2 -fno-vectorize
    with -ffast-math at link time

(*) Reference : ECMWF
(**) Reference : Météo-France
Applications tested

- **AROME forecast:**
  - 1.3 km L90 H12
  - + I/O server (3 nodes)
  - + post-processing on-line
  - + reduced coupling frequency (for commodity, mostly)
  - 200 nodes for the model part: 400 tasks x 12 threads
Troubleshooting with Lustre

- The file system Lustre at ECMWF is not reliable for small files, it makes non-repeatable elapse time (5 % to 10% difference from one run to another)

- Several developments to reduce this issue with small files:
  - No output listing
  - ifs.stat file, atcp file, DDH files & IO-server xml files written out to specific directories on NFS
  - pp-server control files read from a specific directory on NFS

- Elapse time is now reliable +/- 2 %

- Still there are sporadic issues of repeatable elapse time due to the file system (whatever the size of the file is)
  - Solution: rerun...
AND THE WINNER OF THE COMPARISON IS …
Comparisons on AROME cycle 41

Relative performance of compilers (relatively to the mean performance of them)

AROME - Cycle 41

Cray is almost 30 % faster than Intel

Gfortran is 7 % faster than Intel
Explanation n° 1

- **Array syntax formulations (*)**: 
  - **Cray** can make loop fusion on that statements
  - **Gfortran** doesn't make loop fusion on that statements
  - **Intel** hardly make loop fusion on that statements
    (even not really with -O3)

- **What to do**: 
  - Rewrite and use the good old fortran 77 syntax!
    *Indeed: array syntax can hardly make good re-use of the memory cache*
  - It will even benefits a bit to Cray

(*) coding style >> \((A) :,:) = B( :, :) * C( :, :) - D( :, :) / Z( :, :)\)
Explanation n° 2

- **Memory allocation:**
  - **Cray** uses a compiler built-in tcmalloc (thread-caching malloc)
    - Faster for allocations/deallocations cycles in parallel regions
  - **Gfortran** and **Intel** use malloc
    - Poor performance difficult to diagnose because the allocation takes place only when the memory is « touched »

- **What to do:**
  - Replace allocations/deallocations cycles by one of these:
    - Automatic arrays
    - Unique allocation
    - Deallocate/reallocate only if size has increased (or changed)
Vectorization skillness (*) and efficiency (**) :

- **Cray** is skillful and efficient
- **Gfortran** is skillful but less efficient
- **Intel** is efficient but looks less skillful
- Each compiler may decide to vectorize or not, on different criteria

What to do :

- Report vectorization failures to Vendors
- Try to help the compilers with directives
  (mainly « !DEC$ IVDEP » and « !DEC$ VECTOR ALWAYS » for Intel)
- If a compiler is faster than the other, try to guess how it does it

(*) ability to vectorize a given loop
(**) efficiency of the vectorization
Benefits of optimizations

- **Removal of an allocation/deallocation cycle:**
  - ≥ 200 %

- **Vectorization of a loop:**
  - from 20 % to 300 %, depending on:
    - occurrence of array syntax
    - occurrence of deferred-shape dummy arrays

- **Replacement of array syntax by a f77-style loop:**
  - 10-15 % at least
  - Should increase with the number of lines
Impact of such optimizations

Performance of cycle 41T1+(*) against cycle 41 for each compiler:

Optimization from cycle 41 to cycle 41T1+

AROME

Speedup for each compiler

- Cray: 3%
- Intel: 30%
- Gfortran: 25%

(*) Cycle 41T1 + supplementary optimizations
AND
AFTER SUCH OPTIMIZATIONS,
THE WINNER IS …
Comparisons on AROME cycle 41T1+(*)

Relative performance of compilers (relatively to the mean performance of them)

AROME - Cycle 41T1+

All 3 compilers performs at the same speed

(*) Cycle 41T1 + supplementary optimizations
Conclusions

- Yes, the compiler can make the difference:
  - Buy a Cray compiler if you have money and you don't want to worry about optimization
  - Consider the trade-off between Intel compiler licences + potential benefits after optimization work (with a little help from Gfortran or Cray), and just Gfortran for free

- The comparison of compilers tells us:
  - Where there can be room for optimizations, anyway
  - How to code in a more « competitive » way (or is it a more « portable » way?)
Thank you for your attention!
Optimization modset available on git@merou:
khatib_CY41T1_main.01%opt1