Best Practice Guides

Dimitris Dellis

GRNET

Athens, 1 Dec. 2017

Outline

- Profilers
- Examples of profiling in real applications
- Best Practice Guides
- Hands On on supplied codes or your own code
- Discussion

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Ταμείο Περιφερειακής Ανάπτυξης
ψηφιακήελλάδα
Όλα είναι δυνατά
Επιχειρησιακό Πρόγραμμα "Ψηφιακή Σύγκλιση"
ΠΕΠ
ΕΣΠΑ 2007-2013

- Profiler is software that gets metrics on source execution, without addition of timers in source code.
- Serial Profilers
  - One can find detailed time spent in code procedures, i.e. How many times a procedure was called, average time per call, total time spent in procedure, from which point in source was called etc.
  - Standard Unix profiler gprof and its variants, for example sprof.
  - Compiler specific profilers, like vtune for Intel compilers or pgprof for PGI.

- MPI
  - mpiP : Traces MPI calls and gives performance indicators, possible bottlenecks etc. OpenSource, Works with any compiler and MPI implementation.
  - MPI implementations profilers, for example OpeMPI VampirTrace.
- Hybrid MPI/OpenMP/Threads Profilers
  - scalasca : Traces MPI calls, as well as OpenMP calls, provides detailed information timing information per thread, task, node, code line. Graphical Interface to explore profile information.
  - Other mainly commercial profilers/debuggers, for example DDT

In Practice

- Serial Applications : gprof
  - At Compile time use the flags : -pg
  - It is suggested to use -O0 for optimization to avoid any inlining that may result to missing functions timing.
  - Example : `00_profiling1.f` : Matrix Matrix Maltiplication.

```
module load binutils
gcc 00_profiling1.f -pg -O0 -o 00_profiling1.x
./00_profiling1.x
gprof 00_profiling1.x
```

  - You'll se something like

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.30     10.82    10.82        1    10.82    10.82  mymm_
  0.09     10.83     0.01        1     0.01     0.01  initializearrays_
  0.00     10.83     0.00        1     0.00    10.83  MAIN__
```

- In Brief :
  - mymm is executed 1 times, need 10.82 seconds for each call, it is the main time consuming procedure.
  - initializearrays is executed 1 times, need 0.01 secs per call.
  - Main is executed 1 times, it needs less than 0.005 seconds to complete.
  - We have a good estimation where the execution time is spent. In real serial applications output is more interesting.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Ταμείο Περιφερειακής Ανάπτυξης
ψηφιακήελλάδα
Όλα είναι δυνατά
Επιχειρησιακό Πρόγραμμα "Ψηφιακή Σύγκλιση"
ΠΕΠ
ΕΣΠΑ 2007-2013

In Practice

- Pure MPI Applications : mpiP
- If you compile your application using : mpif90 mycode.f -o mycode.x
  do

```
module load mpiP
mpif90 mycode.f -g -L$MPIPROOT/lib -lmpiP -lbfd -lunwind -o mycode.x
```

- -g (debug) flag is needed to include source code information in executable.
- If (that is usually the case) you have a makefile to compile, use in the linking stage mpiP, example :

```
$LD $(OBJECTFILES) -g -L$MPIPROOT/lib -lmpiP -lbfd -lunwind -o mycode.x
```

- Run it : srun mycode.x in slurm

- or mpiexec.hydra -n 8 mycode.x (interactively on login node with 8 procs)
- After completion you'll find a report file called mycode.x.NPROCS.PID.mpiP
- Have a look in the provided information.
- You'll se something like

```
@ mpiP
@ Command : ./06.x
@ Version                    : 3.4.1
@ MPIP Build date            : Sep  7 2015, 16:33:51
@ Start time                 : 2017 11 29 21:45:28
@ Stop time                  : 2017 11 29 21:45:31
@ Timer Used                 : PMPI_Wtime
@ MPIP env var               : [null]
@ Collector Rank             : 0
@ Collector PID              : 29284
@ Final Output Dir           : .
@ Report generation          : Collective
@ MPI Task Assignment        : 0 login01
```

```
......
@--- MPI Time (seconds) ---------------------------------------------
---------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       2.72        0.7    25.69
   1       2.72       1.16    42.52
   2       2.72       1.07    39.11
   3       2.72       1.06    38.98
   4       2.72       1.04    38.24
   5       2.72       1.32    48.29
.....
 31       2.72       1.13    41.51
  *       87.1       34.9    40.05
......
@--- Callsites: 11 --------------------------------------------------
---------------------------------------------------------------------
 ID Lev File/Address                 Line Parent_Funct           MPI_Call
  1   0 06_md_inhomegeneous_reduce.f  115 md                     Bcast
  2   0 06_md_inhomegeneous_reduce.f  137 md                     Bcast
  3   0 06_md_inhomegeneous_reduce.f  202 md                     Reduce
.....
@--- Aggregate Time (top twenty, descending, milliseconds) ----------
---------------------------------------------------------------------
Call                     Site       Time    App%    MPI%     COV
Reduce                      4   1.27e+04   14.57   36.37    0.94
```

```
Barrier                  8   1.03e+04   11.78   29.41    1.09
Bcast                    6    8.8e+03   10.10   25.22    0.18
......
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site       Count       Total       Avrg  Sent%
Reduce                  3          32     3.2e+07      1e+06  11.11
Reduce                  4          32     3.2e+07      1e+06  11.11
Reduce                  9          32     3.2e+07      1e+06  11.11
Bcast                  11          32     3.2e+07      1e+06  11.11
......
@--- Callsite Time statistics (all, milliseconds): 352 --------------------
---------------------------------------------------------------------------
Name              Site Rank  Count      Max     Mean      Min   App%   MPI%
Barrier              8    0      1    0.048    0.048    0.048   0.00   0.01
Barrier              8    1      1      774      774      774  28.42  66.83
Barrier              8    2      1      769      769      769  28.23  72.17
```

....

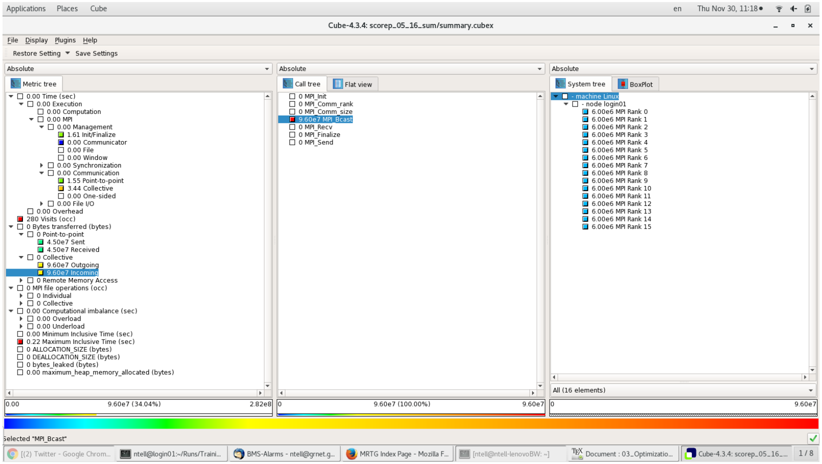
and more.

In Practice

- Hybrid Applications : Scalasca
- If you compile your application using : mpif90 mycode.f -o mycode.x
  do

```
module load binutils qt/5.6.0 cuda/7.5.18
scalasca -instrument mpif90 mycode.f -o mycode.x
scalasca -analyze mpiexec.hydra -n 8 ./mycode.x
scalasca -examine scorep_mycode.x_8_sum
```

- You'll see something like (You need X11 at your Desktop)
- https://sourceforge.net/projects/xming/
- If not X11 available, instead of scalasca -examine use : square -s `scorep_mycode.x_8_sum`. A report will be in `scorep_mycode.x_8_sum/scorep.score` text file.



Cube-4.3.4: scorep_05_16_sum/summary.cubex

```
0.00 Time (sec)
  0.00 Execution
    0.00 Computation
    0.00 MPI
      0.00 Management
        1.61 Init/Finalize
        0.00 Communicator
        0.00 File
        0.00 Window
      0.00 Synchronization
      0.00 Communication
        1.55 Point-to-point
        3.44 Collective
        0.00 One-sided
      0.00 File I/O
  0.00 Overhead
280 Visits (occ)
0 Bytes transferred (bytes)
  0 Point-to-point
    4.50e7 Sent
    4.50e7 Received
  0 Collective
    9.60e7 Outgoing
    9.60e7 Incoming
  0 Remote Memory Access
0 MPI file operations (occ)
  0 Individual
  0 Collective
0.00 Computational imbalance (sec)
  0.00 Overload
  0.00 Underload
0.00 Minimum Inclusive Time (sec)
0.22 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
```

```
0 MPI_Init
0 MPI_Comm_rank
0 MPI_Comm_size
9.60e7 MPI_Bcast
0 MPI_Recv
0 MPI_Finalize
0 MPI_Send
```

```
- machine Linux
  - node login01
    6.00e6 MPI Rank 0
    6.00e6 MPI Rank 1
    6.00e6 MPI Rank 2
    6.00e6 MPI Rank 3
    6.00e6 MPI Rank 4
    6.00e6 MPI Rank 5
    6.00e6 MPI Rank 6
    6.00e6 MPI Rank 7
    6.00e6 MPI Rank 8
    6.00e6 MPI Rank 9
    6.00e6 MPI Rank 10
    6.00e6 MPI Rank 11
    6.00e6 MPI Rank 12
    6.00e6 MPI Rank 13
    6.00e6 MPI Rank 14
    6.00e6 MPI Rank 15
```

```
0.00    9.60e7 (34.04%)    2.82e8
0    9.60e7 (100.00%)    9.60e7
0    9.60e7
```

Selected "MPI_Bcast"

[(2) Twitter - Google Chrom...
ntell@login01:~/Runs/Traini...
BMS-Alarms - ntell@grnet.g...
MRTG Index Page - Mozilla F...
[ntell@ntell-lenovoBW: ~]
Document : 03_Optimizatio...
Cube-4.3.4: scorep_05_16_...
1 / 8

Efficient use I

- ARIS compute nodes have 20 or 40 cores. Use if possible full nodes, i.e. 20/40 cores/node.
- If it is not the case, limit the required nodes.

cores	Nodes	tasks/node	Unused cores
64	4	20	16 on 1 node
128	7	20	12 on 1 node
256	13	20	4 on 1 node
512	26	20	8 on 1 node

Efficient use II

- Common mistake

cores	Nodes	tasks/node	Unused cores
64	8	8	12 cores/node on 8 nodes=96
64	4	16	4 cores/node on 4 nodes = 16
90	6	15	5 cores/node on 6 nodes = 30
128	8	16	4 cores/node on 8 nodes = 32
480	40	12	8 cores/node on 40 nodes = 320
512	32	16	4 cores/node on 32 nodes = 128

- Do not use mpirun/mpiexec nor typical desktop arguments like -np. It happens to forget to change the really needed resources, for example :

Efficient use III

```
#SBATCH --nodes=10
#SBATCH --ntasks=200
mpirun -np 8
or
srun -n 8
```

You allocate (and charged for) 200 cores while you use only 8.

- Try to use the correct combination of tasks and threads with Hybrid applications. Check that the `OMP_NUM_THREADS` is set. In SLURM script template there is code that checks for this.

Efficient use IV

- Surprisingly, this piece of code is frequently removed.

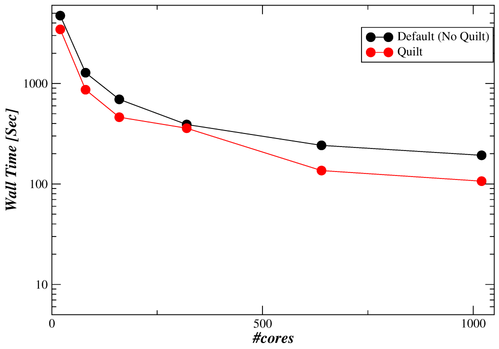
Efficient use I

- Explore the capabilities of your application. With some options in input file(s) you may see much better performance.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Ταμείο Περιφερειακής Ανάπτυξης
ψηφιακήελλάδα
Όλα είναι δυνατά
Επιχειρησιακό Πρόγραμμα "Ψηφιακή Σύγκλιση"
ΠΕΠ
ΕΣΠΑ 2007-2013

Efficient use II

- Example : WRF quilting

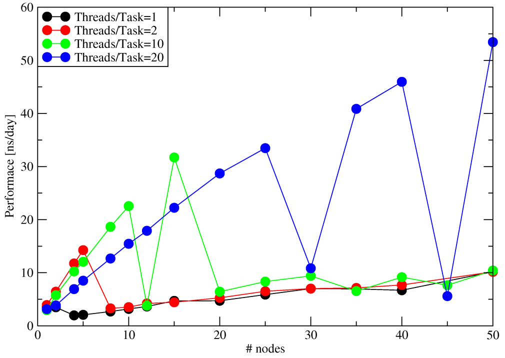


Efficient use

- It depends on the algorithm
- ..and mainly on data
- The same algorithm may exhibit diffierent efficiency with different data
- There are ”gold” rules for algorithms, but :
- Measure performance with your data before decision

Efficient use

- Example : MD of an inhomogeneous system



Efficient use I

- If you can use save/restart and need very long time, use it. Instead of a job of 10 days, use 10 jobs of 1 day (propability of a HW failure in 10 days much higher - especially with multinode runs).
- Request from the Resource Manager wall time slightly higher than the expected. NOT the typical 2 days.
- Example : Submit 100 jobs requesting 2 days each. Scheduler will arrange to run them in $\sim$ 1 week. If each run takes 5 minutes, requesting 6 minutes, all runs will finish in $\sim$ 1 hour instead of $\sim$ 1 week.

Efficient use II

- Even better, submit few jobs with multiple srun, for example 10 jobs with 10 srun.
- Stats : Sept. 2017
  65% of jobs took up to 5% of requested time
  9% between 5 and 10%.
  11% more than 50%

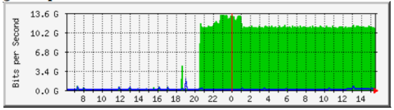
Efficient use III

- Avoid to use .bashrc. Especially when more than 1 versions of package are available. Use modules instead. For example, OpenFOAM :
  `module load openfoam/3.0.1`
  `source $FOAM_BASHRC` instead of put in `.bashrc` all OpenFOAM variables, specific to a certain version.
- Avoid no necessary parameters in input, especially those that affect load balance, grids, methods etc. if it is possible to specify them at runtime, for example, `NPROC_X/Y` in WRF, `processors` or `pair_style lj/cut/gpu` vs `pair_style lj/cut` and `-sf gpu` with LAMMPS.

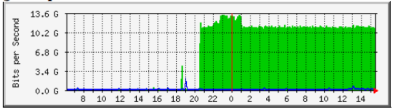
Efficient use IV

- Heavy use of scratch : Read from files with rate 12.6 GBytes/sec for 2 days = 2.12 PBytes for a 100 cores job!!.

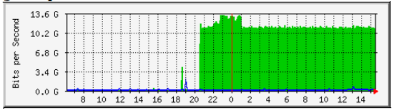
gss01-port1



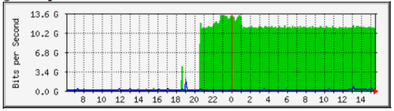
gss01-port2



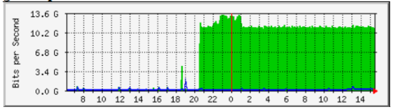
gss01-port3



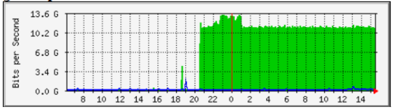
gss01-port4



gss02-port1



gss02-port2



Hands On

- Profile Serial, MPI, Hybrid MPI/OpenMP applications with gprof, mpiP, scalasca.
- For those who have their own Code, try to profile your own code.
- Those who are familiar with vtune, try also vtune, especially with OpenMP only codes.
- Discuss Findings, Suggestions to improve performance.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Ταμείο Περιφερειακής Ανάπτυξης
ψηφιακήελλάδα Όλα είναι δυνατά
Επιχειρησιακό Πρόγραμμα "Ψηφιακή Σύγκλιση"
ΠΕΠ πραγματική ανάπτυξη
ΕΣΠΑ 2007-2013 Ποιότητα ζωής για όλους