Playbook with templates for Intel tools usage on JURECA

Version 0.1, 09.01.2017

Please send feedback to heinrich.bockhorst@intel.com

Note: Module defaults on FZJ Jureca from 3.1.2017

0. Environment

Load the necessary modules for each tool:

A. Intel Compiler:

```
$ module load Intel
```

loads by default: Intel/2017.0.098-GCC-5.4.0
containing: 2017 Version of icc,ifort,icpc and GCC 5.4.0

B. Intel Advisor

```
$ module load Advisor
```

loads by default: Advisor/2017_update1

This tool analyzes SIMD vectorization of programs compiled with Intel compiler. Please add "-g" to the compile line.

C. Intel VTune Amplifier

```
$ module load VTune
```

loads by default: VTune/2017

This tools profiles single node applications including threading. It processes many performance counters and displays the results in many ways.
Please compile your program with "-g". Gcc can be also used.

D. Intel Inspector

```
$ module load Inspector
```

loads by default: Inspector/2017_update1

This tool performs several memory and threading correctness checks.
Please compile your app with "-g".

E. Intel MPI

$ module load IntelMPI

loads by default: IntelMPI/2017.0.098

Intel MPI version based on MPICH2. Contains many optimizations especially for collective operations.

F. Intel Trace Analyzer and Collector (ITAC)

$ module load itac

loads by default itac/2017.1.024

ITAC is similar to Vampir and provids many graphical displays of MPI execution.
ITAC package also includes the Correctness Checker for checking MPI correctness and the
MPI Performance Snapshot (MPS) for a low overhead high scaling analysis.

1. Affinity Settings
==========

OpenMP applications or Hybrid MPI/OpenMP applications will be running faster in most cases by imposing an affinity. In this context affinity means the binding of threads to cores/ logical cores/ caches etc.

For apps compiled with the Intel compiler you may use the Intel OpenMP environments variables. You may also set affinity masks by using the environment variables provided by the OpenMP standard

```
$ export PRG=prg.x : is the program
$ export FLAGS=" " : program flags (optional)
$ export CORES=`cpuinfo -g | grep Cores | grep -v package | cut -d":" -f 2`
```

(last line needs Intel MPI)

Work-flow
---

i) run without affinity

```
$ KMP_AFFINITY=verbose $PRG $FLAGS > out_0.txt  2> out_0.err
```

KMP_AFFINITY=verbose just reports the current affinity settings on stderr.

ii) run with affinity scatter (max distance between thread n , n+1)

```
$ KMP_AFFINITY=verbose,scatter $PRG $FLAGS > out_1.txt  2> out_1.err
```

iii) run with affinity compact (min distance between thread n, n+1)

```
$ KMP_AFFINITY=verbose,compact $PRG $FLAGS > out_2.txt  2> out_2.err
```

iv) limit to one thread per core

```
$ export OMP_NUM_THREADS=$CORES

$ KMP_AFFINITY=verbose,scatter $PRG $FLAGS > out_3.txt  2> out_3.err
```

v) now with compact - watch performance

```
$ KMP_AFFINITY=verbose,compact $PRG $FLAGS > out_4.txt  2> out_4.err
```

the above line will put 2 threads per core on half of the cores!
the setting below will change the granularity to core level.

```
$ KMP_AFFINITY=verbose,compact,1 $PRG $FLAGS > out_5.txt  2>
out_5.err
```

The same effect can be gained by NOT setting OMP_NUM_THREADS and limit the
thread number to a single thread per core

```
$ unset OMP_NUM_THREADS
$ export KMP_HW_SUBSET=1T
```

run with affinity scatter

```
$ KMP_AFFINITY=verbose,scatter $PRG $FLAGS > out_6.txt  2> out_6.err
```

run with affinity compact

```
$ KMP_AFFINITY=verbose,compact $PRG $FLAGS > out_7.txt  2> out_7.err
```

unset KMP_HW_SUBSET=1T
export KMP_AFFINITY=verbose

unsetting KMP affinity but keep the verbose printing (Intel Compiler)
using OpenMP variables
use granularity threads

```
$ OMP_PLACES=threads $PRG $FLAGS > out_8.txt  2> out_8.err

use granularity core

$ OMP_PLACES=cores $PRG $FLAGS > out_9.txt  2> out_9.err


2. Advisor XE (Vectorization analysis)
======================================

Analysis is done in several steps. The results of each step are
accumulated into the current display. The reason is that some steps
generate more overhead like the trip count analysis. Using a singel
run would spoil the timing results.

all steps can be done using the Advisor GUI: $ advixe-gui
For analysis on clusters it may be good to do these steps using the
command line interface because there might be no X connection.

A. getting Help
-----------------------------------------------

$ advixe-cl -help > advisor_help.txt

getting help on collection

$ advixe-cl -help collect > advisor_help_collect.txt


B. survey analysis (light weight profiling, first step)
-----------------------------------------------

$ advixe-cl --collect survey --project-dir ADV -- $PRG $FLAGS

analyze survey results - subset of loops printed in csv format

advixe-cl -report=survey -format=csv   --project-dir ADV | cut -d
"," -f 1,2,3,5,6 > survey_short.txt

shows all loops. Loops that are not vectorized are marked "SCALAR".
Note the IDs (first column) of the top scalar loops
for later use

can be also viewed by the GUI

$ $advixe-gui &


C. trip count and flops analysis
-----------------------------------------------

$ advixe-cl --collect tripcounts -flops-and-masks --project-dir ADV
-- $PRG $FLAGS
```

```
dependency analysis for Loop ID=5 shows to be scalar in survey above
(for example, you may have other IDs)

$ advixe-cl --collect dependencies --mark-up-list=5 --project-dir
ADV -- $PRG $FLAGS

map (memory) analyis for the above loop showing stride

$ advixe-cl --collect map --mark-up-list=5 --project-dir ADV -- $PRG
$FLAGS

view results with GUI or ASCII report

D. generate snapshot (compressed result file). Can be copied to
another computer and analyzed.
-------------------------------------------

$ advixe-cl --snapshot --project-dir ADV --pack --cache-sources --
cache-binaries -- snapshot_01

open snapshot by starting the GUI and select "open result"

$ advixe-gui &



 3. VTune
=====================================

A. getting Help
-------------------------------------------

$ amplxe-cl -help > basic_VTune_help.txt

more details on collection

$ amplxe-cl -help collect > VTune_help_collect.txt


B. work with GUI - sometimes better for starters
-------------------------------------------

$ amplxe-gui &

alternative: work with command line:
i) generate results on cluster
ii) view results with amplxe-gui & + "open result" or  transfer
result + binary + source
to different computer with (same) VTune version


C. Basic Hotspots
-------------------------------------------
```

simplest analysis is Basic Hotspots (works without special drivers)
resolution : 10ms
uses results dir BH (for example)

```
$ amplxe-cl -c hotspots -r BH -- $PRG $FLAGS
```

shows already ascii summary output
more info by opening the GUI

```
$ amplxe-gui BH &
```

D. Advanced Hotspots

better resulution (1 ms by default) given by advanced-hotspots
(drivers have to be installed!)

```
$ amplxe-cl -c advanced-hotspots -r AH -- $PRG $FLAGS
```

shows counts for 3 Events covering clockticks and instructions
more options shown by

```
$ amplxe-cl -help collect advanced-hotspots
```

switch on more omp analysis

```
$ amplxe-cl -c advanced-hotspots -knob analyze-openmp=true -r AH-omp
-- $PRG $FLAGS
```

E. HPC Performance Analysis

```
$ amplxe-cl -help collect hpc-performance
```

shows some additional knobs

```
amplxe-cl --collect hpc-performance -r HPC -- $PRG $FLAGS
```

view data with GUI

```
$ amplxe-gui HPC
```

F. Memory Analysis

```
$ amplxe-cl --collect memory-access -r ME -- $PRG $FLAGS
```

shows if app is memory or compute bound. Shows if memory bandwidth
is sufficient.

further knobs for the analyis available (more overhead)

```
$ amplxe-cl -help collect hpc-performance
```

G. General Exploration

This is the most fundamental analyis pointing to further investigations.
First level shows the stage (back end, front end) where execution is running sub optimal (pink background)
Next level(s) will show individual steps of each execution stage.

```
$ amplxe-cl --collect general-exploration -r GE -- $PRG $FLAGS
```

==========

Playbook ends here! Next version will show more details about how to use the above tools in combination with MPI on Clusters.
Other tools like inspector and MPS (MPI Performance Snapshot) may also be covered.