1.1 HPCC

1.1.1 *Summary*

The HPC Challenge benchmark [1] is a combined benchmark set which consists of 7 subbenchmarks, such as HPL (the Linpack TPP benchmark), DGEMM (matrix-matrix multiplication execution rate), STREAM (sustainable memory bandwidth and the corresponding computation rate), PTRANS (communication of large arrays), RandomAccess (the rate of integer random updates of memory), FFT (complex one-dimensional Discrete Fourier Transform), b_eff Latency/Bandwidth (latency and bandwidth of a number of simultaneous communication patterns). Detailed description is found on [2].
Source code and xml files for the operation on JuBE framework are available under the SVN (https://prace.osd.sara.nl/svn/trunk/pracewp74a/PABS/applications/hpcc) whose TRAC version is 220.

1.1.2 *TestCase*

-- On CURIE --
Of 7 available benchmarks, we perform HPL, STREAM, and Star Random Access, for the consistency with the prior synthetic benchmarks in the PRACE PP [3].

a) Compilation

We use the Intel compiler Ver. 12.1.0 with Intel MKL library Ver. 10.3 for BLAS functions and the Bullxmpi Ver. 1.1.14.1. We refer to
http://software.intel.com/en-us/articles/performance-tools-for-software-developers-use-of-intel-mkl-in-hpcc-benchmark/
and
http://software.intel.com/en-us/articles/performance-tools-for-software-developers-hpl-application-note/
as a guideline for the tuning of HPCC runs under the Intel compiler.

Compilation and linker flags are as follows:

Compilation Flag: -O3 –ip –ftz -openmp

Linker Flag: -nofor-main -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core –lm -lpthread

-nofor-main flag is added to resolve the linking error caused by a fortran executable referring to the c-based library
(http://software.intel.com/en-us/articles/error-undefined-reference-to-main/).

For running specific part of the whole HPCC package, you can edit individual parameter (e.g., RunHPL, RunStarStream, ...) at the start of the source code (hpcc.c). We explicitly edited the code for operating interesting benchmarks only. See Appendix 1.

For job submission on CURIE via JuBE benchmark environemt[4], a number of input files are required according to its format. Details are given in Appendix 2.

b) Parameters and Experiments

Ns (Problem Size), NBs (Block Size), P and Q (Process Grid Ratio) are four important parameters for the better performance. We refer to
http://lab.advancedclustering.com/hpl.html
for setting these input parameters optimal for CURIE fat nodes (32 Cores per node with 4 GB RAM per core). According to the recommendation from HPCC [1], Ns is designed to create a 2-dimensional double-precision array which consumes 80% of the total memory, i.e., $Ns = \sqrt{Total\_Memory / 8(byte) * 0.8}$; NBs is recommended to be in the range of 32 – 256 and it shall be the factor of Ns; Q is designed to be eual to or slightly larger than P. Also, Q and P are factors of the number of blocks in each direction, i.e., (Ns / NBs).

We perform four distinct measurements as follows:

Test Case A: LINPACK Sustained Flop/s
(HPL banchmark from 1 to 1024 cores)
Test Case B: Sustained Memory Bandwidth
(StarSTREAM benchmark from 1 to 32 cores)
Test Case C: T1.6 Cache Miss Performance
(RandomAccess benchmark from 1 to 32 cores)
Test Case D: T2.1 Memory Bandwidth Compared to Flop/s
(HPL and StarSTREAM benchmark with 32 cores)

Each measurement has been performend separately, by turning off all other subbenchmarks for an individual measurement.

c) Test Case A

LINPACK Sustained Flop/s has been measured via the pure MPI job allocation. We compared the performance via a MPI (32 processors) and a hybrid (4 threads * 8 processors) task and find that
MPI: N=115712,NB=128,P=4,Q=8 → HPL=0.257304
Hybrid: N=115712,NB=256,P=2,Q=4 → HPL=0.245906
which verifies the MPI task is preferred for the better Linpack performance.

9 distinct tests have been performed with the following conditions:

- 1 (core) * 1 (node), 4 (cores) * 1 (node), 16 (cores) * 1 (node),
  32 (cores) * 1 (node), 32 (cores) * 2 (nodes), 32 (cores) * 4 (nodes),
  32 (cores) * 8 (nodes), 32 (cores) * 16 (nodes), 32 (cores) * 32 (nodes)

We vary Ns, NBs, or P * Q and measure the performance for determining these parameters. We use 1 full node resource (32 Cores with 4 GB memory per core) for this instrument.

For determining Ns, we initially fix NBs = 256 and P*Q = 4*8. We vary Ns to 57856 (equivalent to 1 GB memory usage), 81920 (2 GB), 99840 (3 GB) and 115712 (4 GB).

	1 GB	2 GB	3 GB	4 GB	(Unit)
HPL	0.235275	0.242752	0.248511	0.253234	(Tflops)

It verifies that larger memory usage (within the limit of system's capacity) will increase the sustained Flop/s.

NBs is determined by fixing Ns equivalent to 4 GB and P*Q = 4*8. We vary NBs to 128, 256, 384 and 512.

	128	256	384	512	(Unit)
HPL	0.256534	0.252713	0.247194	0.239978	(Tflops)

We observe that the best NBs is 128 for CURIE fat nodes.

Finally P and Q are determined on a condition that Ns is 115712 (equivalent to 4 GB memory per core) and NBs is 128. We vary P from 1 to 32.

P	1	2	4	8	16	32
Q	32	16	8	4	2	1
HPL	0.236568	0.247805	0.257304	0.257147	0.246444	0.229048

For the clarity in case of inter-node communication, we also performed the same test on 4 nodes.

P	1	2	4	8	16	32	64	128
Q	128	64	32	16	8	4	2	1
HPL	0.817384	0.889192	0.932069	0.968265	0.94026	0.758805	0.578785	0.411013

It verifies that Q≥P condition satisfies the best performance.

d) Test Case B

Memory banwithd

Sustained Memory Bandwidth test runs the StarSTREAM under the condition that the problem is set to use the full memory. Thus, we apply the same Ns condition as the HPL test. Other parameters are not much relevant to this test.
We initially compared 32-core jobs via pure MPI, pure OpenMP and hybrid (cores under a single socket are threaded) runs to figure out which way is optimal for the memory bandwidth.

	MPI	Hybrid	OpenMP	(Unit)
StarSTREAM_Copy	1.83408	14.5708	58.0668	(MB/s)
StarSTREAM_Scale	1.81728	14.5426	57.8167	(MB/s)
StarSTREAM_Add	2.0457	16.3238	64.9911	(MB/s)
StarSTREAM_Triad	2.07442	16.409	65.3644	(MB/s)

We sense that the measured bandwith is the average-per-process. If it is rewritten via per-core,

	MPI	Hybrid	OpenMP	(Unit)
StarSTREAM_Copy	1.83408	1.82135	1.814588	(MB/s)
StarSTREAM_Scale	1.81728	1.817825	1.806772	(MB/s)
StarSTREAM_Add	2.0457	2.040475	2.030972	(MB/s)
StarSTREAM_Triad	2.07442	2.051125	2.042638	(MB/s)

We observe that the pure MPI job provides the better bandwidth.

e) Test Case C

Cache Miss Performance test runs StarRandomAccess and MPIRandomAccess benchmark tasks. In accordance to PP D 5.2 [3], we set Ns to consume half of total available memory. Other parameters are not much relevant to this test.
Again we compared 32-core jobs via pure MPI, pure OpenMP and hybrid (cores under a single socket are threaded) runs to figure out which way is optimal.

	MPI	Hybrid	OpenMP	(Unit)
MPIRandomAccess	0.068468	0.008286	0.001817	(GUPs)
StarRandomAccess	0.012019	0.03536	0.006504	(GUPs)

We find that the hybrid operation provides a better update rate compared to MPI job. Thus, we perform the MPI run for MPIRandomAccess and a hybrid run for the StarRandomAccess.

f) Test Case D

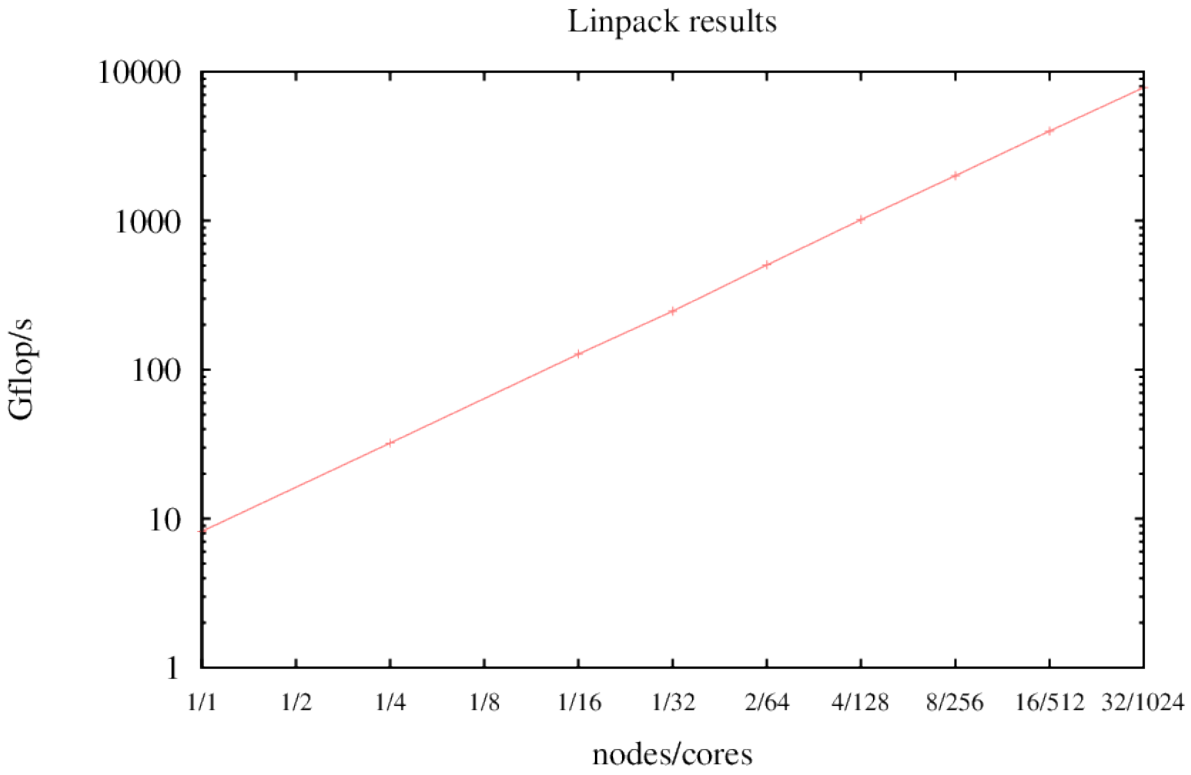
This test computes the Memory Bandwidth to Flop/s ratio. It can be acquired by the simple arithmetics from Test A and B, while we repeat this test to match the same local condition between HPL and STREAM tests. It is done for a single node.

1.1.3 *Results and Analysis*

a) LINPACK Sustained Flop/s

This is a measure of sustained double precision (DP) floating-point operations per second (flop/s) using the LINPACK benchmark which solves a dense linear system of equations. It is used to provide a flop calculation rate measure close to peak performance. The benchmark is included here because of the popularity of LINPACK in ranking floating-point performance in the TOP500 list. In total, 9 jobs have been run, combining 1 up till 32 nodes. The highest measured rate was 7829.00 Gflop/s on 32 nodes with 1024 cores. The measured data points form an almost perfect straight line

	Input Parameters				
nodes/cores	**N**	**NB**	**P**	**Q**	**Gflop/s**
1/1	20480	128	1	1	8.22
1/4	40960	128	2	2	32.20
1/16	81920	128	4	4	127.30
1/32	115712	128	4	8	247.40
2/64	163840	128	8	8	505.30
4/128	231680	128	8	16	1018.00
8/256	327680	128	16	16	2006.00
16/512	463360	128	16	32	3998.00
32/1024	655360	128	32	32	7829.00

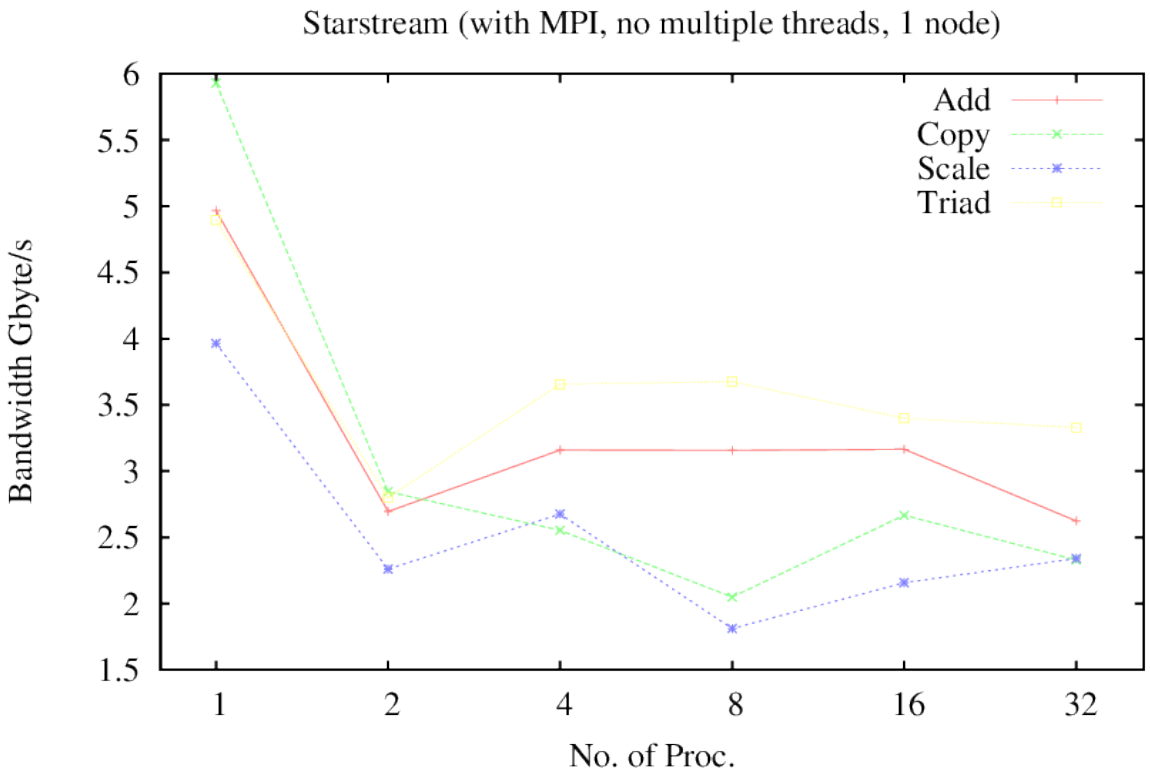


One thing to mention is, the solution at 64 cores / 256 cores (when P=Q) result in residual check (in other words, solution diverges). That should be related with the P and Q setup (setting so that P < Q) or the system-optimal compilation option. Considering that P<Q cases returns the converged solution, P<Q seems a mandatory condition for HPL benchmark.

b) Sustained Memory Bandwidth

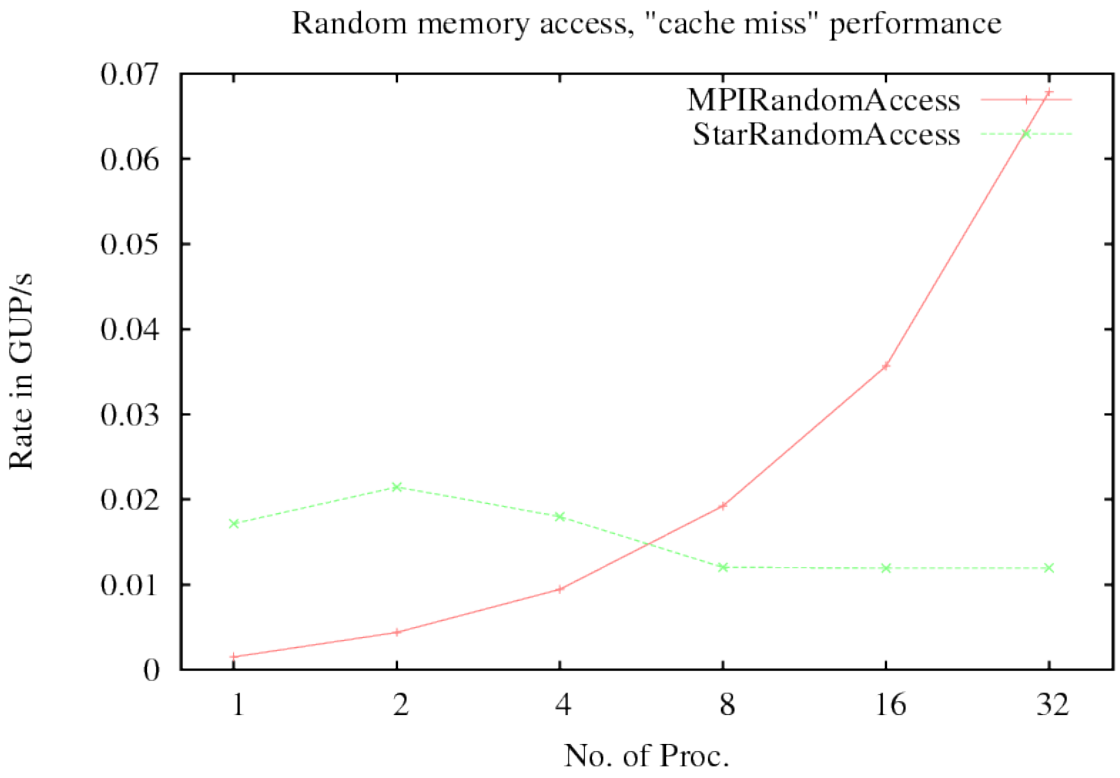
This is a measure of sustained memory bandwidth using the Stream benchmark from HPCC. The purpose of this benchmark is to stress main memory.
The bandwidth drops roughly a factor 2 going from one core to multiple cores. Adding more cores does not decrease the bandwidth less.

Cores	Add (GB/s)	Copy (GB/s)	Scale (GB/s)	Triad (GB/s)
1	4.9680	5.9307	3.9643	4.8958
2	2.6948	2.8451	2.2604	2.8002
4	3.1589	2.5527	2.6755	3.6559
8	3.1554	2.0491	1.8113	3.6758
16	3.1656	2.6675	2.1563	3.3987
32	2.6239	2.3287	2.3406	3.3284



c) Cache Miss Performance

The RandomAccess benchmark measures cache-miss performance by identifying the number of memory locations that can be randomly updated in one second. It reports one figure, Giga updates per second (GUP/s) and can be run on multiple cores and nodes. In this case, the tests were performed on a single node only.



d) Memory Bandwidth Compared to Flops

Memory access is commonly a restriction on computation speeds and this ratio quantifies the theoretical maximum bytes that can be delivered from memory for each flop. This is a derived ratio using previous results.

The run settings are provided where these results are first stated in this document, T1.1 for the LINPACK assessment and T1.4 for the STREAM memory bandwidth assessment, and so they are not repeated here.

Cores/nodes	**Processes**	**STREAM average bandwidth of 1 node (GB/s)**	**LINPACK (GFlop/s)**	**Byte/Flop**
32/1	32	84.9728	247.400	0.34

The result slightly differs according to the system's condition. For more clarity, we run both benchmarks at the same time. The result is as follows.

- (N, NB, P, Q): 115712, 128, 4, 8
- (HPL_Tflops): 0.257356
- (STREAM Geometric Mean): 1.97008
  - StarSTREAM_Copy=1.86597
  - StarSTREAM_Scale=1.82477
  - StarSTREAM_Add=2.05364
  - StarSTREAM_Triad=2.13594

⇨ Byte/Flop = (1.97008 * 1000) MByte/s * (32) Procs / (0.257356 * 1000000) Mflop/s = 0.244962

References

[1] HPCC: http://icl.cs.utk.edu/hpcc/
[2] Report on available performance analysis and benchmark tools, PRACE Preparatory Phase Deliverable D6.3.1, November 2008.
[3] Technical Assessment Report of Prototype Systems, PRACE Preparatory Phase Deliverable D5.2, December 2009
[4] JuBE framework, http://www2.fz-juelich.de/jsc/jube/
[5] P-SNAP homepage, http://www.ccs3.lanl.gov/pal/software/psnap/
[6] mixedMode, http://www2.epcc.ed.ac.uk/~markb/mpiopenmpbench/intro.html

Appendix 1: HPCC Source Code Change

To perform interesting benchmark only, we introduce a new variable called "BenchTest“ which is provided from the command line argument.

```
/* A parameter for the HPCC Benchmarks
  BenchTest == 0       : Perform all benchmarks
  BenchTest == 11      : Perform T1.1 LINPACK Sustained Flops
  BenchTest == 14      : Perform T1.4 Sustained Memory Bandwidth
  BenchTest == 16      : Perform T1.6 Cache Miss Performance
  BenchTest == 21      : Perform T2.1 Memory Bandwidth Compared to Flops
*/
```

It handles whether each benchmark is performed in the following way:

e.g., StarDGEMM, which is not run unless the full package is executed

```
  if(BenchTest == 0) {
  if (params.RunStarDGEMM) HPCC_StarDGEMM( &params );
  } // End of StarDGEMM
```

e.g., HPL, which is run for LINPACK and Memory Bandwidth VS Flops

```
  if(BenchTest == 0 || BenchTest == 11 || BenchTest == 21) {
  if (params.RunHPL) HPL_main( argc, argv, &params.HPLrdata, &params.Failure );
  } // End of HPL
```

It must be possible to easily handle by setting the parameter (RunHPL, ...) from input file.

Appendix 2: Rung HPCC Tests on JuBE Environment

JuBE benchmark provides the integrated job compilation/execution/analysis environment. To make use of it,

1) Install (download and unpack) JuBE framework
2) Locate the source code (to run) on relevant place. Usually codes are located under /application/package_name.
3) Set system-specific environments.
    a. /platform/platform.xml
       It contains compilation-specific configurations. Example on CURIE given below.
    b. /platform/System_Name/job_submission_script.sh
       It contains default job submission script format. Example on CURIE given below.
4) Set code-specific environments.
    a. /applications/code_name/compile.xml
       It contains specific compilation option for this benchmark. Example of HPCC on CURIE given below.
    b. /applications/code_name/execute.xml
       It contains specific job submission option for this benchmark. Example of HPCC on CURIE given below.
    c. /applications/code_name/ (analyse.xml, result.xml, verify.xml)
       Those files are also provided if other tests are needed. In principle, only above two files are enough for running the task.
5) Set problem-specific environments.
    a. /applications/code_name/prepare.xml
       Have the parameter for individual problem set. Example of HPL with 256 code in HPCC on CURIE given below.
    b. /applications/code_name/submission_task_name.xml
       In principle, JuBE supports multiple independent job submissions at the same time. However, on CURIE, it was not working fine (only the first resource request is submitted and all simulation runs are assigned on that allocation in tandem). So we provide separate execution sets per task. Example of HPL with 256 code in HPCC on CURIE given below.
6) Run the task. To run it,
   On the banchmark code's location, type
   ../../bench/jube submission_task_name.xml

```
===== /platform/platform.xml =====
   <platform name="Intel-Nehalem-CURIE">
    <params
     make          = "gmake"
        rm           = "rm -f"
        ar           = "ar"
        arflags      = "-rs"
        ranlib       = "ranlib"
```

```
        cpp           = "cpp"
        cppflags      = "-P"

      cc            = "icc"
      cflags        = "-O3 -ftz -ip -ipo"

      cxx           = "icpc"
        cxxflags      = "-DMPICH_IGNORE_CXX_SEEK"

      f77           = "ifort"
      f90           = "ifort"

        mpi_cc        = "mpicc"
      mpi_cxx       = "mpicxx"

      mpi_f77       = "mpif77"
      f77flags      = "-O3 -ftz -ip -ipo"

      mpi_f90       = "mpif90"
      f90flags      = "-O3 -ftz -ip -ipo"

      ldflags       = ""

        mpi_dir       = ""
        mpi_lib       = ""
        mpi_inc       = ""
        mpi_bin       = ""

        blas_dir      = "-L/usr/local/Intel_compilers/c/composerxe-
2011.3.174/mkl/lib/intel64"
        blas_lib      = "-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lm"

        lapack_dir    = "-L/usr/local/Intel_compilers/c/composerxe-
2011.3.174/mkl/lib/intel64"
        lapack_lib    = "-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lm"

        fftw3_dir     = "-L/usr/local/fftw3-3.2.2/lib"
        fftw3_lib     = "-lfftw3 -lm"
        fftw3_inc     = "-I/usr/local/fftw3-3.2.2/include"

        fftw2_dir     = "-L/usr/local/fftw2-2.1.5/lib"
        fftw2_lib     = "-ldfftw -ldfftw_mpi -ldrfftw -ldrfftw_mpi"
        fftw2_inc     = "-I/usr/local/fftw2-2.1.5/include"
        netcdf3_dir   = ""
        netcdf3_lib   = ""
        netcdf3_inc   = ""

      module_cmd    = "module load"
     />
    </platform>
```

```
===== /platform/Intel-Nehalem-CURIE/intel_PBSsubmit.job.in =====
#!/bin/bash -x

#MSUB -r #BENCHNAME#
#MSUB -N #NODES#
#MSUB -n #TASKS#
#MSUB -c #THREADSPERTASK#
#MSUB -T #TIME_LIMIT#
#MSUB -o #STDOUTLOGFILE#
#MSUB -e #STDERRLOGFILE#
set -x
cd ${BRIDGE_MSUB_PWD}
echo "<jobstart at=\"`date`\" />" >> #OUTDIR#/start_info.xml
#ENV#
#PREPROCESS#
#MEASUREMENT# #STARTER# #ARGS_STARTER# #EXECUTABLE#
#ARGS_EXECUTABLE#
#POSTPROCESS#
echo "<jobend at=\"`date`\" />" >> #OUTDIR#/end_info.xml


===== /applications/hpcc/compile.xml =====
<compile cname="Intel-Nehalem-CURIE">
   <src directory="./src" files="*" />

   <substitute infile="hpl/Make.PRACE.jube" outfile="hpl/Make.PRACE">
   <sub from="#MPI_DIR#"      to="$mpi_dir" />
      <sub from="#MPI_INC#"      to="$mpi_inc" />
      <sub from="#MPI_LIB#"      to="$mpi_lib" />
      <sub from="#BLAS_DIR#"      to="$blas_dir" />
      <sub from="#BLAS_LIB#"      to="$blas_dir -lmkl_intel_lp64 -lmkl_intel_thread -
lmkl_core -lm -lpthread" />
      <!--sub from="#BLAS_LIB#"      to="$blas_dir $blas_lib" /-->


      <sub from="#BLAS_INC#"      to="$blas_dir" />
      <sub from="#F2CDEFS#"      to="" />
      <sub from="#CFLAGS#"      to="$cflags -O3 -ftz -ip -openmp" />
      <sub from="#LDFLAGS#"      to="$ldflags -nofor-main" />

      <sub from="#MPI_CC#"      to="$mpi_cc" />
      <sub from="#MPI_F90#"      to="$mpi_f90" />

      <sub from="#AR#"      to="$ar" />
      <sub from="#ARFLAGS#"      to="$arflags" />
      <sub from="#RANLIB#"      to="$ranlib" />

   </substitute>

   <substitute infile="hpl/lib/arch/build/Makefile.intelSTREAM"
```

```
            outfile="hpl/lib/arch/build/Makefile.hpcc">
   </substitute>

      <command>make arch=PRACE;cp hpcc $execname</command>

</compile>


===== /applications/hpcc/execute.xml =====
<execute cname="Intel-Nehalem-CURIE">

   <input files="../../platform/Intel-Nehalem-CURIE/intel_PBSsubmit.job.in" />

   <substitute infile="intel_PBSsubmit.job.in" outfile="intel_PBSsubmit.job">
      <sub from="#OUTDIR#"           to="$outdir" />
      <sub from="#BENCHNAME#"        to="$benchname" />
      <sub from="#STDOUTLOGFILE#"    to="$stdoutlogfile" />
      <sub from="#STDERRLOGFILE#"    to="$stderrlogfile" />
      <sub from="#NODES#"            to="$nodes" />
      <sub from="#TASKS#"            to="$tasks" />
      <sub from="#TASKSPERNODE#"     to="$taskspernode" />
      <sub from="#THREADSPERTASK#"   to="$threadspertask" />
      <sub from="#TIME_LIMIT#"       to="$timelimit" />
      <sub from="#NOTIFICATION#"     to="n" />
      <sub from="#NOTIFY_EMAIL#"     to="" />
      <sub from="#EXECUTABLE#"       to="$executable" />
      <sub from="#ENV#"              to="$env" />
      <sub from="#PREPROCESS#"       to="export
OMP_NUM_THREADS=${BRIDGE_MSUB_NCORE}" />
      <sub from="#POSTPROCESS#"      to="cat hpccoutf.txt" />
      <sub from="#STARTER#"          to="ccc_mprun" />
      <sub from="#ARGS_STARTER#"     to="" />

<!--
      <sub from="#ARGS_STARTER#"     to="-n `$nodes * $taskpernode`" />
-->

      <sub from="#MEASUREMENT#"      to="time" />
      <sub from="#ARGS_EXECUTABLE#"  to="" />
      <sub from="#MEMORYPERTASK#"    to="" />
   </substitute>

   <command>ccc_msub intel_PBSsubmit.job</command>
</execute>


===== /applications/hpcc/prepare.xml =====
%%% Case for HPL benchmark with 256 cores
<prepare cname="HPL256">
  <input files="src/hpccinf.txt.jube" />
  <substitute infile="hpccinf.txt.jube" outfile="hpccinf.txt">
```

```
<sub from="#HPLNN#" to="1" /> <!-- basic -->
<sub from="#HPLN#" to="327680" /> <!-- basic -->
<sub from="#HPLNNBS#" to="1" /> <!-- basic -->
<sub from="#HPLNBS#" to="128" /> <!-- basic -->
<sub from="#HPLNGRID#" to="1" /> <!-- basic -->
<sub from="#HPLPS#" to="16" /> <!-- basic -->
<sub from="#HPLQS#" to="16" /> <!-- basic -->
<sub from="#HPLTHRESHOLD#" to="16.0" /> <!-- advanced -->
<sub from="#HPLNPFACTS#" to="1" /> <!-- advanced -->
<sub from="#HPLPFACTS#" to="2" /> <!-- advanced -->
<sub from="#HPLNNBMINS#" to="1" /> <!-- advanced -->
<sub from="#HPLNBMINS#" to="4" /> <!-- advanced -->
<sub from="#HPLNNDIVS#" to="1" /> <!-- advanced -->
<sub from="#HPLNDIVS#" to="2" /> <!-- advanced -->
<sub from="#HPLNRFACTS#" to="1" /> <!-- advanced -->
<sub from="#HPLRFACTS#" to="1" /> <!-- advanced -->
<sub from="#HPLNBROADCAST#" to="1" /> <!-- advanced -->
<sub from="#HPLBROADCAST#" to="1" /> <!-- advanced -->
<sub from="#HPLNDEPTHS#" to="1" /> <!-- advanced -->
<sub from="#HPLDEPTHS#" to="1" /> <!-- advanced -->
<sub from="#HPLSWAP#" to="2" /> <!-- advanced -->
<sub from="#HPLSWAPTHRESHOLD#" to="64" /> <!-- advanced -->
<sub from="#HPLNADDPTRANSN#" to="0" /> <!-- more for PTRANS -->
<sub from="#HPLNADDPTRANSNB#" to="0" /> <!-- more for PTRANS -->
<sub from="#HPLADDPTRANSN#" to="" /> <!-- more for PTRANS -->
<sub from="#HPLADDPTRANSNB#" to="" /> <!-- more for PTRANS -->
  </substitute>
</prepare>


===== /applications/hpcc/specific_task_name.xml =====
%%% Case for HPL benchmark with 256 cores
<bench name="HPCC" platform="Intel-Nehalem-CURIE" >
<benchmark name="HPCC-T1.1-HPL" active="1">

   <!-- version="reuse|new" -->
   <compile     cname="$platform" version="reuse" />

   <tasks       threadspertask="1" tasks pernode="32" nodes="8"  />
   <prepare     cname="HPL256" />

   <execution   iteration="1" cname="$platform" timelimit="14400" />
   <verify      cname="Generic" />
   <analyse     cname="Generic" />
</benchmark>
</bench>
```