Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Scalability Limitations of CESM Simulation on Juqueen

Mads R. B. Kristensen[a] and Roman Nuterman[a]

[a]*Niels Bohr Institute, Universityt of Copenhagen, Denmark*

Abstract

In this paper, we explore the challenges of running the current version (v1.2.2) of Community Earth System Model (CESM) Simulation on Juqueen**.** We present a set of workarounds for the Juqueen supercomputer that enables massively parallel executions and demonstrate scalability up to 3024 CPU-cores.

Introduction

The Community Earth System Model (CESM)[1] is a fully coupled, global climate model that provides state-of-the-art computer simulations of the Earth's past, present, and future climate states. CESM supports a broad range of architecture including the Blue Gene series B, P, and Q thus at first glance it should be straightforward to run CESM on the Blue Gene/Q supercomputer called Juqueen hosted at the Jülich Supercomputing Centre[2]. However, it turns out that running *vanilla* CESM v1.2.2 fails when scaling above 128 CPU-cores on Juqueen – the execution crashes with an out-of-memory error in the communication layer. The problem arises when performing ultra high-resolution (1/4 deg. for atmosphere and 1/10 for ocean model) fully coupled simulations because of massive model output. For this purpose, CESM uses a parallel NetCDF library. Unfortunately, CESM using the parallel NetCDF library does not work properly for more than 22000 CPU-cores, which high-resolution simulations require. The problem appeared due to wrong parallel topology (hardcoded in the wrapper) of Blue Gene/Q I/O nodes specific to parallel I/O on Juqueen. The main contribution of this paper is a set of workarounds that makes it possible to scale CESM v1.2.2 above 128 CPU-cores when running on Juqueen.

Workarounds

In order to run CESM on Juqueen, we have to use a number of workarounds. Because the recent CESM v1.2.2 release is not supported on Juqueen, one has to back port the machine specific files from a CESM v1.3 development version. Our colleague, J. Dennis and A. Baker, at National Center for Atmospheric Research (NCAR) did a great job and came up with the following modifications:

(1) We needed to obtain all of the CESM input data for the various resolutions from NCAR (since Juqueen is not yet supported, the input data has not been installed on the file system). The automated procedure for obtaining input data did not retrieve everything, so a number of files had to be obtained manually (e.g., ice_ic, fsurdat, all relevant/cpl/gridmaps files, and the pophdr file).

(2) We used the mpixlf95_r compiler (instead of mpixlf2003_r), adding "-qxlf2003=polymorphic" to the FFLAGS, "-DNO_C_SIZEOF" to the CPPDEFS, and "--enable-filesystem-hints=gpfs" to PIO_CONFIG_OPTS.

(3) Parallel NetCDF I/O (PIO) tuning: The high-resolution case (1/4 deg. atm and 1/10 ocean: ne120_t12) must use PNETCDF, and requires the following settings:
<entry id="PIO_NUMTASKS" value="256"/>
<entry id="PIO_TYPENAME" value="pnetcdf"/>

(4) In order to handle out-of-memory errors, a patch for the topology.c file was required (see appendix).

(5) We successfully ran a high-resolution case from default initial conditions for 4 days. Using 22324 MPI-tasks and 4 tasks per node (bg_size=5581), the simulation at a rate of 0.43 simulated years per day. This is quite slow, but subsequent attempts to incorporate threading have resulted in jobs crashing in different locations for unclear reasons.

Finally, for job submission on Juqueen the *runjob* command was used with the options that are presented here:

runjob --envs XLSMPOPTS=stack=64000000 --envs OMP_NUM_THREADS=${mthrds} --envs LOGNAME=${USER} --ranks-per-node ${MAX_TASKS_PER_NODE} --np ${ntasks} --exe \${EXEROOT}/cesm.exe >&! cesm.log.\$LID

Benchmark

With the presented workarounds, it is possible to achieve scalable performance of medium-resolution (1 deg. for atmosphere and ocean model) fully coupled CESM simulations on Juqueen. Figure 1 shows strong scale speedup with 128 CPU-cores as the baseline, i.e. the work size is fixed throughout all runs. At 256 and 512 CPU-cores, the speedup compared to the baseline is close to linear with a utilization of 95% and 90%, respectively. At 1024 and 3024 CPU-cores, the scalability is limited to a utilization of 76% and 62%, respectively.

Figure 2, shows the scalability of the individual CESM components. Most of the components scale fairly well but two components, River Transport Model and the Parallel Ocean Program, show no speedup when going from 1024 to 3024 CPU-cores. However, the scalability of ultra high-resolution (1/4 deg. for atmosphere and 1/10 for ocean model) fully coupled CESM simulations is limited -- utilizing 22324 CPU-cores only achieves a rate of 0.43 simulated years per 24 hours of wall-time.

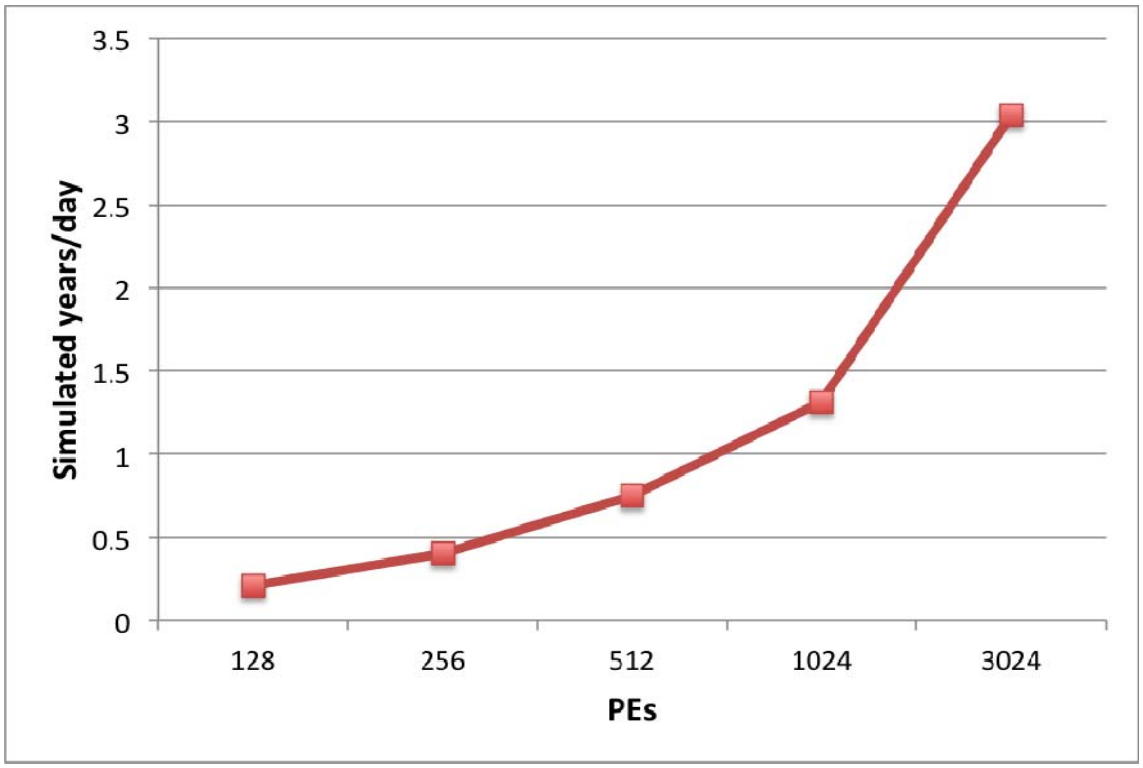

Figure 1: Fully coupled CESM comp-set with 1 x 1 deg. horizontal grid resolution for both atmosphere and ocean. Simulated years per day (24 hours) vs. number of processor elements

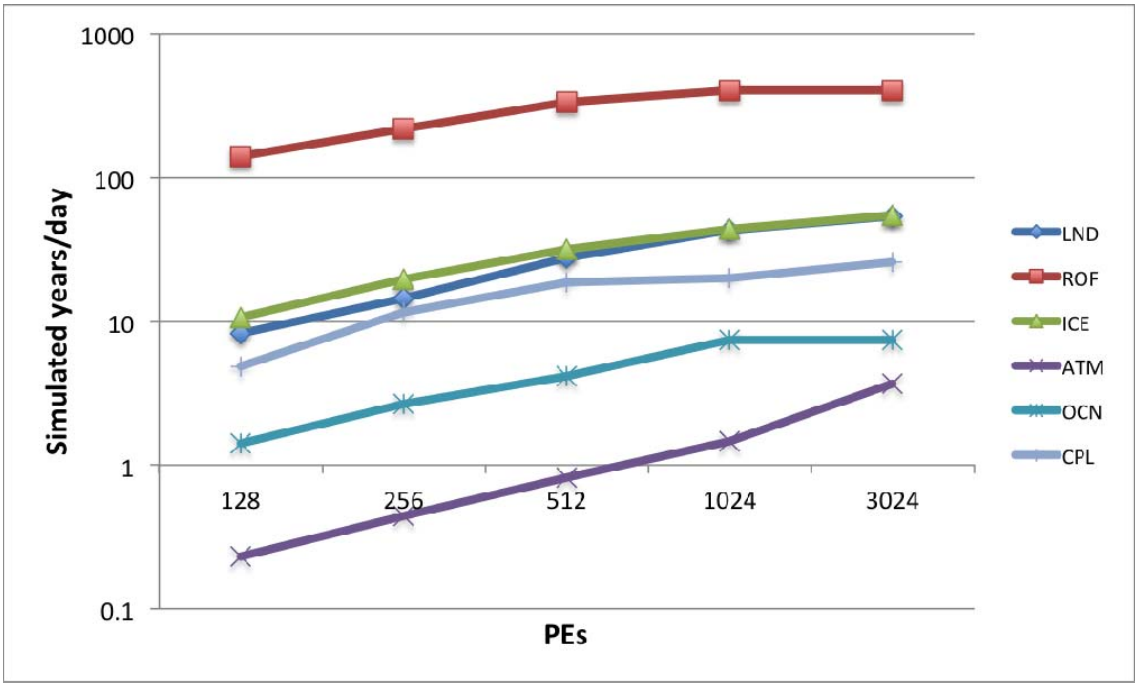

Figure 2: Fully coupled CESM comp-set with 1 x 1 deg. horizontal grid resolution for both atmosphere and ocean. Simulated years per day (24 hours) for each package vs. number of processor elements, where LND – Community Land Model (CLM), ROF – River Transport Model (RTM), ICE – sea-ice component (CICE), ATM – Community Atmosphere Model (CAM), OCN – Parallel Ocean Program (POP) and CPL - coupler package.

Summary

The presented work shows that it is possible to achieve good performance, up to 3024 CPU-cores, when running a medium-resolution fully coupled CESM simulation on Juqueen. However, ultra high-resolution CESM simulations that require significantly more CPU-cores (minimum 22324 CPU-cores) do not scale well. Furthermore, in order to scale CESM simulations above 128 CPU-cores, a set of mandatory workarounds have been presented for the Juqueen supercomputer.

References

[1] James W. Hurrell , M. M. Holland, P. R. Gent, S. Ghan, Jennifer E. Kay, P. J. Kushner, J.-F. Lamarque, W. G. Large, D. Lawrence, K. Lindsay, W. H. Lipscomb, M. C. Long, N. Mahowald, D. R. Marsh, R. B. Neale, P. Rasch, S. Vavrus, M. Vertenstein, D. Bader, W. D. Collins, J. J. Hack, J. Kiehl, and S. Marshall, 2013: The community earth system model: a framework for collaborative research. Bull. Amer. Meteor. Soc., 94, 1339–1360.

[2] Jülich Supercomputing Centre (JSC), http://www.fz-juelich.de/ias/jsc/EN/Home

Acknowledgements

This work was financially supported by the PRACE-3IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763.

Appendix – topology.c patch

The code in the appendix can be found on the PRACE web accompanying this White Papers as

`WP200_Appendix_topology.c.patch`

```
--- topology.c_old     2014-09-12 10:20:35.710277314 +0200
+++ topology.c_new.c   2014-09-12 10:20:26.286277062 +0200
@@ -3,7 +3,7 @@
 #include <string.h>


-#if defined(BGL) || defined(BGP)
+#if defined(BGL) || defined(BGP) || defined(BGQ)

 #include <mpi.h>
 #include <math.h>
@@ -23,8 +23,8 @@
 #define   Personality_rankInPset         BGLPersonality_rankInPset
 #define   Personality_psetNum            BGLPersonality_psetNum

-
-#else
+#endif
+#ifdef BGP

 #include <spi/kernel_interface.h>
 #include <common/bgp_personality.h>
@@ -39,10 +39,30 @@
 #define   Personality_rankInPset         BGP_Personality_rankInPset
 #define   Personality_psetNum            BGP_Personality_psetNum

+#endif
+
+#ifdef BGQ
+
+#include <kernel/process.h>
+#include <kernel/location.h>
+#include <firmware/include/personality.h>
+#include <mpix.h>
+
+#define   get_personality                Kernel_GetPersonality
+#define   get_processor_id               Kernel_PhysicalProcessorID
+#define   Personality                    Personality_t

 #endif

-#define min(a,b) a<b?a:b
+#define max(a,b) \
+   ({ __typeof__ (a) _a = (a); \
+       __typeof__ (b) _b = (b); \
+     _a > _b ? _a : _b; })
+
+#define min(a,b) \
+   ({ __typeof__ (a) _a = (a); \
+       __typeof__ (b) _b = (b); \
+     _a < _b ? _a : _b; })


 int rank;
@@ -60,6 +80,11 @@
    MPI_Comm_size(comm2,&np);
    MPI_Get_processor_name(my_name, &my_name_len);

+#ifdef BGQ
+    MPIX_Hardware_t hw;
+    MPIX_Hardware(&hw);
+#endif
+
    /*  Get the personality  */
    Personality pers;
    char message[100];
@@ -70,29 +95,51 @@
    int *tmp;
    int i,ierr;

+#ifdef BGQ
+  Personality personality;
+  Kernel_GetPersonality(&pers, sizeof(pers));
+#else
    get_personality (&pers, sizeof(pers));
-   Personality_getLocationString (&pers, message);
-
+#endif
    int numIONodes,numPsets,numNodesInPset,rankInPset;
+#if defined(BGL) || defined(BGP)
+   Personality_getLocationString (&pers, message);
    numIONodes = Personality_numIONodes (&pers);
    numNodesInPset = Personality_numNodesInPset (&pers);
    rankInPset = Personality_rankInPset (&pers);
+#endif
+#ifdef BGQ
+   int numpsets, psetID, psetsize, psetrank;
```

```
   +
   +   bgq_pset_info (comm2, &numpsets, &psetID, &psetsize, &psetrank);
   +
   +   numIONodes = numpsets;
   +   numNodesInPset = psetsize;
   +   rankInPset = rank;
   +#endif

    #ifdef BGL
       numPsets = Personality_numPsets (&pers);
   -#else
   +#endif
   +#ifdef BGP
       rankInPset --;
   numPsets = BGP_Personality_numComputeNodes(&pers)/numNodesInPset;
    #endif
   +#ifdef BGQ
   +   numPsets = numpsets;
   +#endif

   -
       if(rank == 0) { printf("number of IO nodes in block: %i \n",numIONodes);}
       if(rank == 0) { printf("number of Psets in block : %i \n",numPsets);}
       if(rank == 0) { printf("number of compute nodes in Pset: %i \n",numNodesInPset);}

   -
       int psetNum;
   +#ifdef BGQ
   +   psetNum = psetID;
   +#else
       psetNum = Personality_psetNum (&pers);
   +#endif
   +
    #ifdef DEBUG
       if((*iotask)>0) {
          printf( "%04i (%-50s %s) %i yes\n", rank, my_name, message, psetNum );
   @@ -143,9 +190,14 @@
       int task_count;
       MPI_Comm comm2;

   +#ifdef BGQ
   +   MPIX_Hardware_t hw;
   +   MPIX_Hardware(&hw);
   +#endif

       comm2 = MPI_Comm_f2c(*comm);

   +
       MPI_Comm_rank(comm2, &rank);

       MPI_Comm_size(comm2, &np);
   @@ -160,19 +212,31 @@
       /*  printf("Determine  io  tasks:  proc  %i:  tasks=  %i  numiotasks=%i  stride=  %i  \n",  rank,  np,
(*numiotasks), (*stride)); */

       if((*rearr) > 0) {
   +
   +#ifdef BGQ
   +    Personality personality;
   +    Kernel_GetPersonality(&pers, sizeof(pers));
   +#else
         get_personality (&pers, sizeof(pers));
   +#endif

         int numIONodes,numPsets,numNodesInPset,rankInPset;
         int numiotasks_per_node,remainder,numIONodes_per_pset;
         int lstride;

   +     /* Number of computational nodes in processor set */
   +     #ifdef BGQ
   +     int numpsets, psetID, psetsize, psetrank;
   +    bgq_pset_info (comm2,&numpsets, &psetID, &psetsize, &psetrank);
   +     numIONodes = numpsets;
   +     numNodesInPset = psetsize;
   +     #else
         /* total number of IO-nodes */
         numIONodes = Personality_numIONodes (&pers);
   -
   -     /* Number of computational nodes in processor set */
         numNodesInPset = Personality_numNodesInPset (&pers);
   -
   -      /*printf("Determine io tasks: me %i : nodes in pset= %i ionodes = %i\n", rank, numNodesInPset,
numIONodes); */
   +     #endif
   +
   +     /*     printf("Determine io tasks: me %i : nodes in pset= %i ionodes = %i\n", rank, numNodesInPset,
numIONodes); */
```

```
       if((*numiotasks) < 0 ) {
@@ -221,9 +285,13 @@
       /* Number of processor sets */
 #ifdef BGL
       numPsets = Personality_numPsets (&pers);
-#else
+#endif
+#ifdef BGP
       numPsets = BGP_Personality_numComputeNodes(&pers)/numNodesInPset;
 #endif
+#ifdef BGQ
+     numPsets = numpsets;
+#endif

       /* number of IO nodes in processor set (I need to add
      code to deal with the case where numIONodes_per_pset != 1 works
@@ -234,12 +302,17 @@
       coreId = get_processor_id ();

       /* What is the rank of this node in the processor set */
+#ifdef BGQ
+     psetNum = psetID;
+     rankInPset = psetrank;
+#else
+     /* determine the processor set that this node belongs to */
+     psetNum = Personality_psetNum (&pers);
       rankInPset = Personality_rankInPset (&pers);
-#ifdef BGP
+#endif
 #ifdef BGP
       rankInPset--;
 #endif
-     /* determine the processor set that this node belongs to */
-     psetNum = Personality_psetNum (&pers);

       /* printf("Pset #: %i has %i nodes in Pset; base = %i\n",psetNum,numNodesInPset, *base); */

@@ -249,10 +322,10 @@


       /* start stridding MPI tasks from base task */
-     iam = rankInPset-(*base);
+     iam = max(0,rankInPset-(*base));
       if (iam >= 0)  {
         /* mark tasks that will be IO-tasks  or IO-clients */
-       /*       printf("iam = %d lstride = %d coreID = %d\n",iam,lstride,coreId);*/
+       /*    printf("iam = %d lstride = %d coreID = %d\n",iam,lstride,coreId); */
         if((iam % lstride == 0) && (coreId == 0) ) {  /* only io tasks indicated by stride and coreId = 0
*/
        if((iam/lstride) < numiotasks_per_node) {
          /* only set the first (numiotasks_per_node - 1) tasks */
@@ -261,18 +334,18 @@
          /*  If there is an uneven number of io-clients to io-nodes
              allocate the first remainder - 1 processor sets to
              have a total of numiotasks_per_node */
-       if(psetNum < remainder) {(*iamIOtask) = 1;};
+       if(psetNum < remainder) {(*iamIOtask) = 1;
+       };
       }
        }
+         /*         printf("comm = %d iam = %d lstride = %d coreID = %d iamIOtask = %i \n",comm2,
iam,lstride,coreId,(*iamIOtask)); */
     }
   }else{
-     /* We are not doing rearrangement.... so all tasks are io-tasks */
-     (*iamIOtask) = 1;
+     /* We are not doing rearrangement.... so all tasks are io-tasks */
+     (*iamIOtask) = 1;
     }
-  }
-  else
-  {
+
     /* We are not doing rearrangement.... so all tasks are io-tasks */
     (*iamIOtask) = 1;
   }

-  /* printf("myrank = %i iotask = %i \n", rank, (*iamIOtask)); */
+  /*printf("comm = %d myrank = %i iotask = %i \n", comm2, rank, (*iamIOtask));*/

     /* now we need to correctly determine the numiotasks */
     MPI_Allreduce(iamIOtask, &task_count, 1, MPI_INT, MPI_SUM, comm2);
@@ -282,4 +355,129 @@

 }

+int bgq_ion_id (void)
+{
```

```
+  int iA,  iB,  iC,  iD,  iE;               /* The local node's coordinates  */
+  int nA,  nB,  nC,  nD,  nE;               /* Size of each torus dimension  */
+  int brA, brB, brC, brD, brE;              /* The bridge node's coordinates */
+  int io_node_route_id;
+
+  Personality_t personality;
+
+  Kernel_GetPersonality(&personality, sizeof(personality));
+
+  iA  = personality.Network_Config.Acoord;
+  iB  = personality.Network_Config.Bcoord;
+  iC  = personality.Network_Config.Ccoord;
+  iD  = personality.Network_Config.Dcoord;
+  iE  = personality.Network_Config.Ecoord;
+
+  nA  = personality.Network_Config.Anodes;
+  nB  = personality.Network_Config.Bnodes;
+  nC  = personality.Network_Config.Cnodes;
+  nD  = personality.Network_Config.Dnodes;
+  nE  = personality.Network_Config.Enodes;
+
+  brA = personality.Network_Config.cnBridge_A;
+  brB = personality.Network_Config.cnBridge_B;
+  brC = personality.Network_Config.cnBridge_C;
+  brD = personality.Network_Config.cnBridge_D;
+  brE = personality.Network_Config.cnBridge_E;
+
+/*
+* This is the bridge node, numbered in ABCDE order, E increments first.
+* It is considered the unique "io node route identifer" because each
+* bridge node only has one torus link to one io node.
+*/
+
+  io_node_route_id = brE + brD*nE + brC*nD*nE + brB*nC*nD*nE + brA*nB*nC*nD*nE;
+
+  return io_node_route_id;
+
+}
+
+
+
+int bgq_pset_info (MPI_Fint comm2, int* tot_pset, int* psetID, int* pset_size, int* rank_in_pset)
+{
+        MPI_Comm comp_comm2, pset_comm, bridge_comm;
+        int comp_rank, status, key, bridge_root, tot_bridges, cur_pset, itr, t_buf;
+        int temp_id, rem_psets;
+        MPI_Status mpi_status;
+
+        MPI_Comm_rank (comm2, &comp_rank);
+
+        status = MPI_Comm_dup ( comm2, &comp_comm2);
+    if ( MPI_SUCCESS != status)
+    {
+        printf(" Error duplicating communicator \n");
+        MPI_Abort(comm2, status);
+    }
+
+        // Compute the ION BridgeNode ID
+           key = bgq_ion_id ();
+
+        // Create the pset_comm per bridge node
+           status = MPI_Comm_split ( comp_comm2, key, comp_rank, &pset_comm);
+           if ( MPI_SUCCESS != status)
+           {
+               printf(" Error splitting communicator \n");
+               MPI_Abort(comm2, status);
+           }
+
+        // Calculate the rank in pset and pset size
+           MPI_Comm_rank (pset_comm, rank_in_pset);
+           MPI_Comm_size (pset_comm, pset_size);
+
+        // Create the Bridge root nodes communicator
+           bridge_root = 0;
+           if (0 == *rank_in_pset)
+                bridge_root = 1;
+
+        // Calculate the total number of bridge nodes / psets
+           tot_bridges = 0;
+           MPI_Allreduce (&bridge_root, &tot_bridges, 1, MPI_INT, MPI_SUM, comm2);
+
+           *tot_pset = tot_bridges;
+
+        // Calculate the Pset ID
+        cur_pset = 0;
+        rem_psets = tot_bridges;
+        if ((0 == comp_rank) && (bridge_root ==1))
```

```
+       {
+               *psetID = 0;
+               rem_psets = tot_bridges-1;
+               cur_pset++;
+       }
+
+       t_buf = 0; // Dummy value
+       if (0 == comp_rank)
+       {
+               for (itr = 0; itr < rem_psets; itr++)
+               {
+                       MPI_Recv (&t_buf,1, MPI_INT,  MPI_ANY_SOURCE, MPI_ANY_TAG,comm2, &mpi_status);
+                       MPI_Send (&cur_pset, 1, MPI_INT, mpi_status.MPI_SOURCE, 0, comm2);
+                       cur_pset++;
+               }
+       }
+
+       if ((1 == bridge_root) && ( 0 != comp_rank))
+       {
+               MPI_Send (&t_buf, 1, MPI_INT, 0, 0, comm2);
+               MPI_Recv (&temp_id,1, MPI_INT, 0, 0, comm2, &mpi_status);
+
+               *psetID = temp_id;
+               /*printf (" Pset ID is %d \n", *psetID);*/
+       }
+       // Broadcast the PSET ID to all ranks in the psetcomm
+       MPI_Bcast ( psetID, 1, MPI_INT, 0, pset_comm);
+
+       // Free the split comm
+               MPI_Comm_free (&pset_comm);
+
+       MPI_Barrier (comm2);
+
+       return 0;
+}
+
 #endif
```