Introduction to PSC’s Bridges

Anirban Jana, PhD
Computational Scientist
Pittsburgh Supercomputing Center

January 9, 2018
Agenda

Overview of Bridges
Bridges Programming Environment
Large Shared Memory
Big Data on Bridges with Hadoop (will not cover today)
GPU computing on Bridges
Virtualization on Bridges
Agenda

Overview of Bridges
Bridges Programming Environment
Large Shared Memory
Big Data on Bridges with Hadoop
GPU computing on Bridges
Virtualization on Bridges
Bridges Overview

• The $9.65M Bridges acquisition is made possible National Science Foundation (NSF) award #ACI-1445606: Bridges: From Communities and Data to Workflows and Insight

• Overarching goal is to converge HPC and Big Data to enable new areas of research

• Delivered by Hewlett Packard Enterprise

• First deployment of Intel Omnipath Architecture (OPA)
# Bridges Overview

<table>
<thead>
<tr>
<th>Type</th>
<th>RAM</th>
<th>Ph</th>
<th>n</th>
<th>CPU / GPU / other</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESM</td>
<td>12TB</td>
<td>1</td>
<td>2</td>
<td>16 × Intel Xeon E7-8880 v3 (18c, 2.3/3.1 GHz, 45MB LLC)</td>
<td>HPE Integrity Superdome X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>16 × TBA</td>
<td></td>
</tr>
<tr>
<td>LSM</td>
<td>3TB</td>
<td>1</td>
<td>8</td>
<td>4 × Intel Xeon E5-8860 v3 (16c, 2.2/3.2 GHz, 40 MB LLC)</td>
<td>HPE ProLiant DL580</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>34</td>
<td>4 × TBA</td>
<td></td>
</tr>
<tr>
<td>RSM</td>
<td>128GB</td>
<td>1</td>
<td>752</td>
<td>2 × Intel Xeon E5-2695 v3 (14c, 2.3/3.3 GHz, 35MB LLC)</td>
<td>HPE Apollo 2000</td>
</tr>
<tr>
<td>RSM-GPU</td>
<td>128GB</td>
<td>1</td>
<td>16</td>
<td>2 × Intel Xeon E5-2695 v3 + 2 × NVIDIA K80</td>
<td>HPE ProLiant DL360</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32</td>
<td>2 × Intel Xeon E5-2695 v3 + 2 × NVIDIA next-generation GPU</td>
<td></td>
</tr>
<tr>
<td>DB-s</td>
<td>128GB</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3 + SSD</td>
<td>HPE ProLiant DL360</td>
</tr>
<tr>
<td>DB-h</td>
<td>128GB</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3 + HDDs</td>
<td>HPE ProLiant DL380</td>
</tr>
<tr>
<td>Web</td>
<td>128GB</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3</td>
<td>HPE ProLiant DL360</td>
</tr>
<tr>
<td>Otherb</td>
<td>128GB</td>
<td>1</td>
<td>14</td>
<td>2 × Intel Xeon E5-2695 v3</td>
<td>HPE ProLiant DL360, DL380</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. All RAM in these nodes is DDR4-2133
b. Other nodes = front end (2) + management/log (8) + boot (2) + MDS (4)
Bridges Overview

http://staff.psc.edu/nystrom/bvt

- 20 Storage Building Blocks, implementing the parallel Pylon filesystem (~10PB) using PSC’s SLASH2 filesystem
- 4 MDS nodes
- 2 front-end nodes
- 2 boot nodes
- 8 management nodes
- 6 “core” Intel® OPA edge switches: fully interconnected, 2 links per switch
- Intel® OPA cables
- 4 HPE Integrity Superdome X (12TB) compute nodes
- 42 HPE ProLiant DL580 (3TB) compute nodes
- 12 HPE ProLiant DL380 database nodes
- 6 HPE ProLiant DL360 web server nodes
- 20 “leaf” Intel® OPA edge switches
- 32 RSM nodes with NVIDIA next-generation GPUs
- 16 RSM nodes with NVIDIA K80 GPUs

Purpose-built Intel® Omni-Path topology for data-intensive HPC
Bridges Overview

Database and Web Server Nodes

• Dedicated database nodes will power persistent relational and NoSQL databases
  – Support data management and data-driven workflows
  – SSDs for high IOPs; RAIDed HDDs for high capacity

• Dedicated web server nodes
  – Enable distributed, service-oriented architectures
  – High-bandwidth connections to XSEDE and the Internet
Agenda

Overview of Bridges

**Bridges Programming Environment**

Large Shared Memory

Big Data on Bridges with Hadoop

GPU computing on Bridges

Virtualization on Bridges
Bridges Programming Environment
The module command

- Sets up the environment for using a package

$ module av gcc

--------------------------------------------------- /opt/modulefiles ---------------------------------------------------
gcc/4.7.2  gcc/4.8.4  gcc/5.3.0(default) gcc/6.3.0  gcc/7.2.0

$ module list
Currently Loaded Modulefiles:
  1) psc_path/1.1  2) slurm/17.02.5  3) intel/17.4

$ module unload intel
$ module load gcc
$ module list
Currently Loaded Modulefiles:
  1) psc_path/1.1  2) slurm/17.02.5  3) gcc/5.3.0

$ module swap gcc gcc/7.2.0
$ module list
Currently Loaded Modulefiles:
  1) psc_path/1.1  2) slurm/17.02.5  3) gcc/7.2.0
Bridges Programming Environment
The *module* command

• **Some additional useful *module* commands**

  $ module help gcc/7.2.0
  
  To display a brief help about the package

  $ module show gcc/7.2.0
  
  To list the specific actions that will be taken to set up the package environment
# Bridges Programming Environment

## Compilers and MPI

<table>
<thead>
<tr>
<th>Compilers</th>
<th>Intel</th>
<th>OpenMPI</th>
<th>MVAPICH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GNU</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PGI</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **Recommended 1\(^{st}\) choice** – Intel compilers/MPI
- **Recommended 2\(^{nd}\) choice** – GNU/OpenMPI
- **Code for GPU using OpenACC directives** – PGI
Bridges Programming Environment
Compilers and MPI

• Intel Parallel Studio
  • Intel compilers (icc, icpc, ifort, mpiicc, mpiicpc, mpiifort)
  • Intel MPI
  • Intel tools (VTune, Advisor, Trace analyzer etc)

$ module av intel

-------------------------------- /opt/modulefiles --------------------------------
intel/17.4(default) intel/compilers intelpython/3.5.2
intel/18.0.0.128 intel/compilers-2017 intelpython/2.7.12
Bridges Programming Environment
Compilers and MPI

• Just the GNU compilers

$ module av gcc

--------------------------------- /opt/modulefiles ---------------------------------
gcc/4.7.2 gcc/4.8.4 gcc/5.3.0(default) gcc/6.3.0 gcc/7.2.0
Bridges Programming Environment
Compilers and MPI

• Just the PGI compilers

$ module av pgi

--------------------------------------------- /opt/modulefiles ---------------------------------------------
pgi/16.10     pgi/16.3     pgi/17.5(default)
Bridges Programming Environment
Compilers and MPI

• Other compiler/MPI combinations

$ module av mpi

--------------------------------- /opt/modulefiles ---------------------------------
mpi/gcc_mvapich mpi/intel_mpi mpi/intel_openmpi mpi/pgi_openmpi
mpi/gcc_openmpi mpi/intel_mvapich mpi/pgi_mvapich mpi-caffe/git_bf17d5a4

Note: To use a particular combination, you have to load modules for both the compiler and the MPI (exception: to use Intel compiler and Intel MPI, just load the intel module).

Example: To use GNU 5.3.0 compilers and OpenMPI, do

$ module load gcc mpi/gcc_openmpi
Bridges Programming Environment
Other languages

**Python**

```bash
$ module av python2
```

```
----------------------------- /opt/modulefiles -----------------------------
python2/2.7.11_gcc(default) python2/2.7.11_gcc_np1.11 python2/intel_2.7.12
python2/intel_2.7.13
```

```bash
$ module av python3
```

```
----------------------------- /opt/modulefiles -----------------------------
python3/3.4.2 python3/3.5.2_gcc_mkl(default) python3/intel_3.5.2
python3/intel_3.6.2
```
Bridges Programming Environment

Other languages

• Python

• Rich collection of add-on packages in each Python

Example: A subset of python2 packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>appdirs</td>
<td>1.4.0</td>
</tr>
<tr>
<td>biopython</td>
<td>1.66</td>
</tr>
<tr>
<td>bz2file</td>
<td>0.98</td>
</tr>
<tr>
<td>cutadapt</td>
<td>1.10</td>
</tr>
<tr>
<td>cycler</td>
<td>0.10.0</td>
</tr>
<tr>
<td>Cython</td>
<td>0.23.4</td>
</tr>
<tr>
<td>decorator</td>
<td>4.0.9</td>
</tr>
<tr>
<td>funcsigsn</td>
<td>1.0.2</td>
</tr>
<tr>
<td>functools</td>
<td>3.2.3.post2</td>
</tr>
<tr>
<td>h5py</td>
<td>2.5.0</td>
</tr>
<tr>
<td>khmer</td>
<td>2.0</td>
</tr>
<tr>
<td>matplotlib</td>
<td>1.5.1+1597.g3cd9c2d</td>
</tr>
<tr>
<td>mock</td>
<td>2.0.0</td>
</tr>
<tr>
<td>mpi4py</td>
<td>2.0.0</td>
</tr>
<tr>
<td>networkx</td>
<td>1.11</td>
</tr>
<tr>
<td>nltk</td>
<td>3.2.1</td>
</tr>
<tr>
<td>nose</td>
<td>1.3.7</td>
</tr>
<tr>
<td>numpy</td>
<td>1.11.0</td>
</tr>
<tr>
<td>pbr</td>
<td>1.9.1</td>
</tr>
<tr>
<td>PIL</td>
<td>1.1.7</td>
</tr>
<tr>
<td>pbr</td>
<td>1.9.1</td>
</tr>
<tr>
<td>Ply</td>
<td>3.8</td>
</tr>
<tr>
<td>py</td>
<td>1.4.31</td>
</tr>
<tr>
<td>pyfaidx</td>
<td>0.4.7.1</td>
</tr>
<tr>
<td>pyparsing</td>
<td>2.1.1</td>
</tr>
<tr>
<td>pysam</td>
<td>0.9.0</td>
</tr>
<tr>
<td>pytest</td>
<td>2.9.1</td>
</tr>
<tr>
<td>python-dateutil</td>
<td>2.5.3</td>
</tr>
<tr>
<td>pytools</td>
<td>2016.1</td>
</tr>
<tr>
<td>pytz</td>
<td>2016.4</td>
</tr>
<tr>
<td>PyVCF</td>
<td>0.6.8</td>
</tr>
<tr>
<td>scikit-learn</td>
<td>0.17.1</td>
</tr>
<tr>
<td>scipy</td>
<td>0.17.0</td>
</tr>
<tr>
<td>screed</td>
<td>0.9</td>
</tr>
<tr>
<td>six</td>
<td>1.10.0</td>
</tr>
<tr>
<td>skflow</td>
<td>0.1.0</td>
</tr>
<tr>
<td>sklearn</td>
<td>0.0</td>
</tr>
<tr>
<td>virtualenv</td>
<td>14.0.6</td>
</tr>
</tbody>
</table>
Bridges Programming Environment
Other languages

• **MATLAB**

```bash
$ module av matlab
```

-------------- /opt/modulefiles ---------------------------------------
matlab/MCR_R2013a matlab/R2016a  matlab/R2017a
Bridges Programming Environment
Other languages

• R

$ module av R

------------------------------------- /opt/modulefiles ---------------------------------
R/3.2.3-mkl R/3.3.1-mkl R/3.3.3-mkl R/3.4.1-mkl
Bridges Programming Environment

Other languages

• Java

$ module av java

-------------------------------- /opt/modulefiles --------------------------------
java/jdk8u73(default) java7/jdk7u80
Bridges Programming Environment

File systems

$HOME
10GB quota per user; for codes, scripts, small datasets etc

$SCRATCH
Lustre; primarily to be used as scratch; files older than 30 days will be deleted

$PROJECT
SLASH2; primarily to be used as persistent storage of simulation outputs, large datasets etc

$LOCAL
Local storage on each node available to every running job; fast; will be wiped after the job finishes

$RAMDISK
Memory file system available to every running job; fastest – especially for many small read/writes; will be wiped after the job finishes

Note: Total SCRATCH+PROJECT quota for a group equals storage awarded for the grant
Bridges Programming Environment
Job scheduler

• **SLURM**
  - The 2 most useful commands are *sbatch* and *squeue*
  - See the Bridges User Guide for how to typically use them
  - See the manpages for more advanced usage; for example *man sbatch*
  - Use *sbatch* options **--nodes** (-N for short) and **--ntasks-per-node** to specify the right number of cores (alternatively **--ntasks** for total number of cores)
  - MPI jobs requesting up to 42 nodes (1176 cores) can use the *sbatch* option **--switch=1[@maxtime]** to ensure all nodes are allocated on a single Omnipath leaf switch; this will likely improve performance due to faster inter-processor communication, but also may incur higher wait times in queue; optional *maxtime* specifies how long scheduler tries to find a single switch, after which scheduler ignores the single switch restriction
  - To submit many jobs together (e.g., parameter sweep) use **--array** (or -a for short)
Bridges Programming Environment
Job scheduler

• **SLURM partitions**
  - **RM** - for jobs running on the RSM (128GB) nodes, have exclusive use of allocated nodes
  - **RM-shared** - for jobs running on the RSM nodes, but share nodes with other jobs.
  - **RM-small** - for small jobs needing <= 2 RSM nodes and <= 8 hrs
  - **LM** - for jobs that will run on Bridges' LSM and ESM (3TB and 12TB) nodes; 1 node per job; share node with other jobs
  - **GPU** - for jobs that will run on GPU nodes, have exclusive use of allocated nodes
  - **GPU-shared** - for jobs that will run on the GPU nodes and share nodes with other jobs
  - For more queue details, see user guide

• For LM and LM-shared, need to ask for memory
  - --mem=GBs
  - Max is 3000GB for a LSM node, 12000GB for an ESM node

• For GPU and GPU-shared, need to ask for GPUs
  - --gres=gpu:type:#gpus (where type should be either k80 or p100)
  - You will also get a certain number of cpu cores automatically
Bridges Programming Environment
Charging

• On RM nodes: \( \text{SUs} = \#\text{CPU-cores} \times \#\text{hours} \)

• On k80 GPU nodes: \( \text{SUs} = \#\text{GPUs} \times \#\text{hours} \)

• On p100 GPU nodes: \( \text{SUs} = \#\text{GPUs} \times \#\text{hours} / 2.5 \)

• On LM nodes: \( \text{SUs} = \#\text{TBs} \times \#\text{hours} \)

• To check usage, log onto Bridges and type

\[ \text{xdusage} \ -u \ <\text{username}> \]
Bridges Programming Environment
Interactive usage

• Launch an interactive session on compute nodes using the `interact` command
• Options for the `interact` command include `-p`, `-t`, `-N`, `--ntasks-per-node`, `-A`, `-R`, `--mem`, `--gres` etc
• `interact -h` will give you more details of its usage
Bridges Programming Environment
Interactive usage

• Example: Get interactive access to 4 CPU-cores of a node in the RM-shared partition for 1.5hrs:

```bash
$ interact -p RM-shared -N 1 --ntasks-per-node=4 -t 1:30:00
```
Bridges Programming Environment
Interactive usage

• Example: Get interactive access to 1 p100 GPU (and 16 CPU-cores) in the GPU-shared partition for 15 minutes:

$ interact -p GPU-shared -N 1 --gres=gpu:p100:1 -t 15:00
Bridges Programming Environment
Interactive usage

• Example: Get interactive access to a large memory node for 1hr and request 1TB of memory:

  $ interact -p LM --mem=1000GB

• Note: You will different number of CPU-cores depending on which large memory node you land on; to get exact # check $SLURM_JOB_CPUS_PER_NODE or $SLURM_CPUS_ON_NODE

• Note: Default walltime for interactive jobs is 1 hr
Bridges Programming Environment
Tools: Debugger

• DDT from Allinea
  • Powerful Graphical User Interface (GUI)
  • Access via DDT client instead of XWindows; will result in better performance
  • Instructions for download, installation and initial configuration of the client can be found at
    https://www.psc.edu/index.php/user-resources/software/ddt#install
Bridges Programming Environment
Tools: Debugger DDT

RUN
Run and debug a program.

ATTACH
Attach to an already running

OPEN CORE
Open a core file from a previous run.

MANUAL LAUNCH (ADVANCED)
Manually launch the backend

OPTIONS

Remote Launch:
Off
Off
Configure ...
bridges
Bridges Programming Environment
Tools: Debugger DDT
Bridges Programming Environment
Tools: Debugger DDT

Run and debug an executable on Bridges
Bridges Programming Environment
Tools: Debugger DDT

Full path to executable
Bridges Programming Environment

Tools: Debugger DDT
Bridges Programming Environment

Tools: Debugger DDT
Bridges Programming Environment
Tools: Performance analyzer

- VTune from Intel (for serial, multithreaded and MPI jobs)

Running example: Step 1
- Run and collect measurements

```bash
$ module load vtune
$ cd <directory containing laplace_mpi_c>
$ interact -p RM -N 1 --ntasks-per-node=28
$ mpirun -n 4 amplxe-cl -result-dir laplace_mpi_c_vtune -quiet -collect \ hotspots ./laplace_mpi_c
```

Note: Although we may use just 4 cores, we request all 28 cores of a node to avoid node-sharing, as that may cloud the performance measurements
Bridges Programming Environment

Tools: Performance analyzer

• VTune from Intel

Running example: Step 2

• Visualize and analyze the measurements

First make sure some XWindows program such as XMing is running

$ cd laplace_mpi_c_vtune.r501.pvt.bridges.psc.edu
$ amplxe-gui laplace_mpi_c_vtune.r501.pvt.bridges.psc.edu.amplxe
Bridges Programming Environment

Tools: Performance analyzer

- VTune from Intel
Bridges Programming Environment
More Intel

• **Math Kernel Library (MKL)**
  C and FORTRAN APIs

• **Data Analytics Acceleration Library (DAAL)**
  C++ and Java APIs (and Python API coming)
Agenda

Overview of Bridges
Bridges Programming Environment
Large Shared Memory
Big Data on Bridges with Hadoop
GPU computing on Bridges
Virtualization on Bridges
Large Shared Memory on Bridges

Hardware

- **Large Shared memory (LSM) nodes**
  - 8 Phase 1 LSM nodes, with 64 CPU-cores per node
  - 34 Phase 2 LSM nodes, with 80 CPU-cores per node
  - 3TB RAM per node
  - 16TB on-node storage

- **Extreme Shared Memory (ESM) nodes**
  - 2 Phase 1 ESM nodes, with 288 CPU-cores per node
  - 2 Phase 2 ESM nodes, with 352 CPU-cores per node
  - 12TB RAM per node
  - 64TB on-node storage
Large Shared Memory on Bridges

Target applications

- Applications that need a lot of shared memory
  - Graph analytics, Machine learning, Genome sequence assembly

- Applications that need a lot of threads, e.g., OpenMP codes that scales well beyond 28 threads, MATLAB jobs with > 28 workers
  - Only 28 threads on each RSM node
  - Upto 64 and 80 threads on each Phase 1 and 2 LSM node respectively
  - Upto 288 and 352 threads on each Phase 1 and 2 ESM node respectively

- Applications that benefit from using of memory file system ($RAMDISK)
  - Applications that open and close a large number of files
  - Consider using $LOCAL vs using $RAMDISK
Large Shared Memory on Bridges
How to write code for Large Memory?

- Bridges programming environment supports all the well-established shared memory programming models. You (or your package) are most likely to be using:
  - OpenMP
  - Java
  - MATLAB
Large Shared Memory on Bridges

How to run jobs?

- Jobs must be submitted to the LM partition
  
  `sbatch -p LM myjob.slurm`
  
  Or `#SBATCH -p LM` (in SLURM script)

- Jobs must request “amount of memory”
  - Max 12000 GB
  - If requested memory ≤ 3000GB, job may run on an LSM or an ESM node
  - If requested memory > 3000 GB, job will run on an ESM node

- Jobs cannot span more than one node

- Multiple jobs may share one node
Large Memory Nodes
Measuring memory usage

- **Linux utility time**
  
  ```
  $/usr/bin/time -f "%Es %MkB %PCPU" mpirun
  ```

- **top (in the background)**
  
  ```
  # Starting to take snapshots using the TOP command in batch mode (-b);
  # snapshots are taken every 60 seconds (-d 60) based on the processes
  # ${USER} is running currently.
  top -b -u ${USER} -d 60 > top_snapshots.mem &
  toppid=$!
  
  #Execute processes
  
  #Killing the TOP command at the end...
  kill -9 $toppid
  ```
Agenda

Overview of Bridges
Bridges Programming Environment
Large Shared Memory
Big Data on Bridges with Hadoop
GPU computing on Bridges
Virtualization on Bridges
GPU Computing on Bridges
Hardware at present

- 16 Phase 1 GPU nodes
- Each has the foll specs:
  - 28 CPU cores per node
  - 128GB RAM
  - 8TB on-node storage
  - 2 NVIDIA K80 (Kepler) dual GPUs (4 GPUs total)

- 32 Phase 2 GPU nodes
- Each has the foll specs:
  - 32 CPU cores per node
  - 128GB RAM
  - 8TB on-node storage
  - 2 NVIDIA P100 (Pascal) GPUs (2 GPUs total)
GPU Computing on Bridges
Programming paradigms

3 Ways to Accelerate Applications on GPUs

Applications

- Libraries
  
  "Drop-in" Acceleration

- OpenACC Directives
  
  Easily Accelerate Applications

- Programming Languages
  (CUDA, OpenCL)
  
  Maximum Flexibility
GPU Computing on Bridges
CUDA Toolkit

$ module av cuda

--------------------------------------- /opt/modulefiles ----------------------------------------
cuda/7.5    cuda/8.0(default)    cuda/8.0RC    cuda/9.0    cuda/9.0RC
GPU Computing on Bridges
GPU accelerated libraries

- cuBLAS
- cuDNN
- cuFFT
- cuRAND
- NPP
- cuSolver
- cuSparse

In /opt/packages/cuda/7.5/lib64
If you want something that is not yet on Bridges, email bridges@psc.edu
GPU Computing on Bridges

Debuggers

- allinea DDT
- CUDA-GDB
- CUDA-MEMCHECK
- NVIDIA NSIGHT
GPU Computing on Bridges
Performance Analyzers

NVIDIA Visual Profiler

NVIDIA NSIGHT
GPU Computing on Bridges
OpenACC
Will need PGI compilers

$ module av pgi
--------------------------------------------- /opt/modulefiles ---------------------------------------------
pgi/16.3

$ module unload intel
$ module load pgi

$ module list
Currently Loaded Modulefiles:
  1) psc_path/1.0  2) slurm/15.08.8  3) pgi/16.3
Let us compile the Laplace example (C version)

$ pgcc -fast -acc -Minfo=accel laplace_acc.c -o laplace_acc_c

main:
56, Generating copy(Temperature_last[:,:])
   Generating create(Temperature[:,:])
61, Loop is parallelizable
62, Loop is parallelizable
   Accelerator kernel generated
   Generating Tesla code
   61, #pragma acc loop gang, vector(4) /* blockIdx.y threadIdx.y */
   62, #pragma acc loop gang, vector(32) /* blockIdx.x threadIdx.x */
72, Loop is parallelizable
73, Loop is parallelizable
   Accelerator kernel generated
   Generating Tesla code
   72, #pragma acc loop gang, vector(4) /* blockIdx.y threadIdx.y */
   73, #pragma acc loop gang, vector(32) /* blockIdx.x threadIdx.x */
   74, Max reduction generated for dt
82, Generating update host(Temperature[:,:,])
Compiling the Laplace example (Fortran version)

[anirban@br006 acc_only]$ pgf90 -fast -acc -Minfo=accel laplace_acc.f90 -o laplace_acc_f90

serial:

   41, Generating copy(temperature_last(:,:,))
       Generating create(temperature(:,:,))

   45, Loop is parallelizable

   46, Loop is parallelizable
       Accelerator kernel generated
       Generating Tesla code

       45, !$acc loop gang, vector(4) ! blockIdx%y threadIdx%y

       46, !$acc loop gang, vector(32) ! blockIdx%x threadIdx%x

       57, Loop is parallelizable

       58, Loop is parallelizable
           Accelerator kernel generated
           Generating Tesla code

           57, !$acc loop gang, vector(4) ! blockIdx%y threadIdx%y

           58, !$acc loop gang, vector(32) ! blockIdx%x threadIdx%x

           59, Max reduction generated for dt

       67, Generating update host(temperature(:,:,))
SLURM script to submit the job – laplace_acc_c.slurm

#!/bin/bash

#SBATCH -p GPU-shared
#SBATCH -N 1 #Number of nodes
#SBATCH -n 7  #Number of cpu cores to be allocated per node, set it to be 7*(number of gpus)
#SBATCH --gres=gpu:k80:1
#SBATCH -t 10:00

date
set -x

export PGI_ACC_TIME=1 #To output profiling information for GPU execution

./laplace_acc_c

For FORTRAN version, use laplace_acc_f90.slurm
GPU Computing on Bridges
OpenACC

Submit the job

$ sbatch laplace_acc_c.slurm
Submitted batch job 185769

$ squeue -u anirban

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>185769</td>
<td>GPU-share</td>
<td>laplace_</td>
<td>anirban</td>
<td>R</td>
<td>0:08</td>
<td>1</td>
<td>gpu015</td>
</tr>
</tbody>
</table>
---------- Iteration number: 2000 ----------
[9995,9995]: 97.64  [9996,9996]: 98.34  [9997,9997]: 98.93  [9998,9998]: 99.38  [9999,9999]: 99.72  [10000,10000]: 99.93

Max error at iteration 2000 was 0.017918
Total time was 65.139827 seconds.

Accelerator Kernel Timing data
/home/anirban/xsede2016/tutorial/anirbantests/gpu/acc_only/laplace_acc.c
main  NVIDIA  devicenum=0
time(us): 62,133,563
56: data region reached 2 times
  56: data copyin transfers: 48
    device time(us): total=129,929 max=2,742 min=1,927 avg=2,706
  88: data copyout transfers: 48
    device time(us): total=120,777 max=2,578 min=1,780 avg=2,516
60: compute region reached 2000 times
62: kernel launched 2000 times
  grid: [313x2500]  block: [32x4]
    device time(us): total=22,263,681 max=11,199 min=11,108 avg=11,131
    elapsed time(us): total=22,310,931 max=11,310 min=11,132 avg=11,155
71: compute region reached 2000 times
71: data copyin transfers: 2000
  device time(us): total=8,952 max=16 min=3 avg=4
73: kernel launched 2000 times
  grid: [313x2500]  block: [32x4]
    device time(us): total=34,212,284 max=17,232 min=16,925 avg=17,106
    elapsed time(us): total=34,254,876 max=17,252 min=16,946 avg=17,127
73: reduction kernel launched 2000 times
  grid: [1]  block: [256]
    device time(us): total=2,953,531 max=1,507 min=1,476 avg=1,476

$ tail -36 slurm-185769.out

Job output – includes performance data for the GPU

GPU Computing on Bridges
OpenACC

BRIDGES
PITTSBURGH SUPERCOMPUTING CENTER
Agenda

Overview of Bridges
Bridges Programming Environment
Large Shared Memory
Big Data on Bridges with Hadoop
GPU computing on Bridges

Virtualization on Bridges
Virtualization on Bridges

VMs and Containers

• Why may anybody need one?
  • Really need very customized environment
  • Host a webserver or a database node
  • Combination of these leading to a very complex workflow
Virtualization on Bridges

Request VMs via form on PSC webpage

https://www.psc.edu/bridges/user-guide/virtual-machines/request-a-virtual-machine
Questions ???

• **Bridges User Guide**
  • [https://www.psc.edu/bridges/user-guide](https://www.psc.edu/bridges/user-guide)
  • Your feedback welcome.

• **Real People**
  • [https://www.psc.edu/homepage/contact-us](https://www.psc.edu/homepage/contact-us)
  • Mail / Phone. Backed by expertise. Not a black hole.

• **Training/workshops**
  • [https://www.xsede.org/for-users/training](https://www.xsede.org/for-users/training)
  • E.g., workshops on MPI/OpenMP, GPU/OpenACC, Big Data