BLUE WATERS

SUSTAINED PETASCALE COMPUTING

December 3, 2013

XK Programming Environment and Tools

Blue Waters Workshop

December, 2013

NCSA

NSF

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

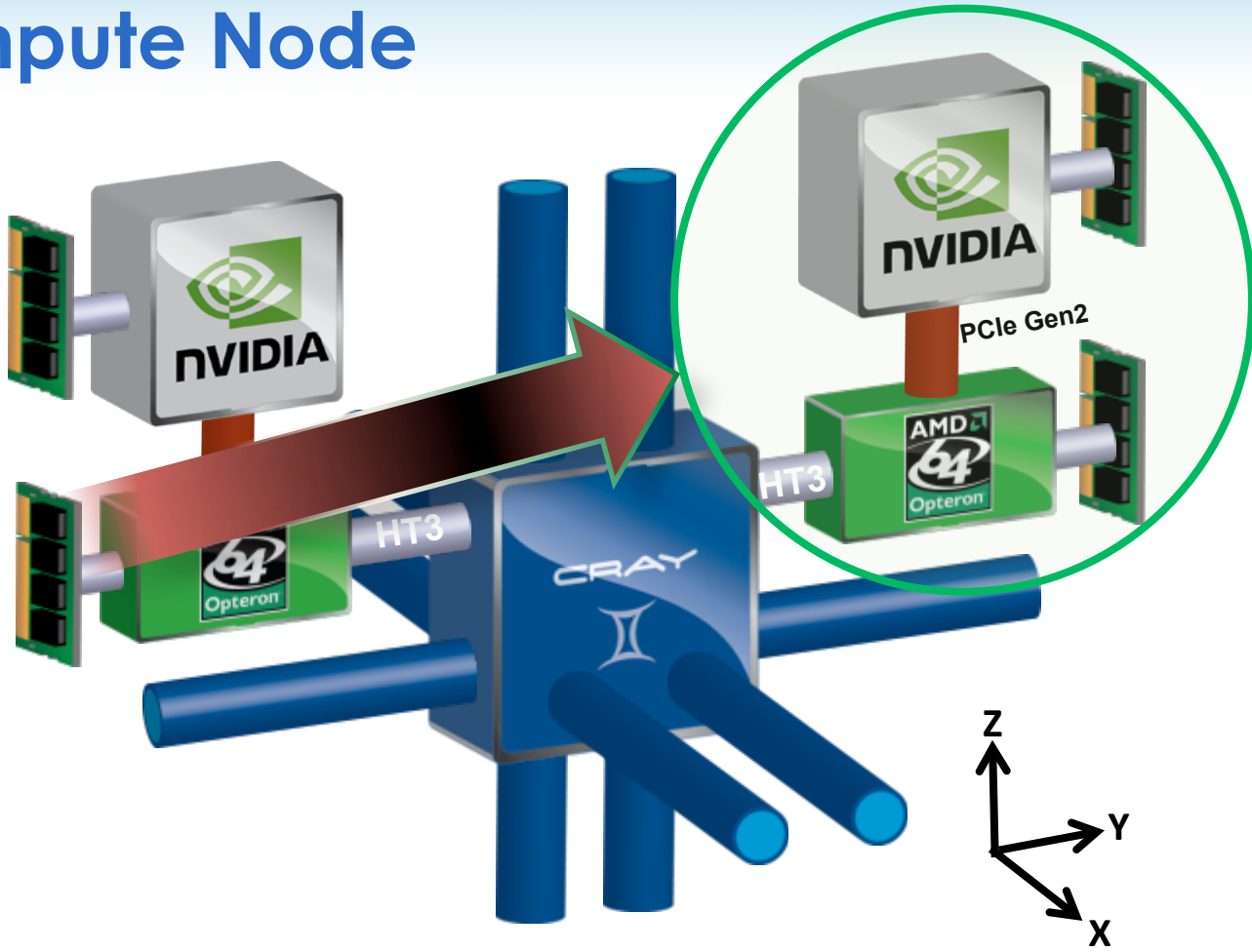
CRAY

XK7 Programming Environment and Tools

- XK7 Hardware Overview
- Interactive session on XK nodes
- CUDA Programming Environment
- CUDA RDMA Support
- CUDA FORTRAN
- OpenACC
- Cray Accelerated Scientific Libraries
- Examples /u/staff/anisimov/training/xk.tgz

Cray XK7 Compute Node

XK7 Compute Node Characteristics
AMD Series 6200 (Interlagos) Core Module
NVIDIA Kepler
Host Memory - 32GB 1600 MHz DDR3
NVIDIA Memory 6GB GDDR5 capacity
Gemini High Speed Interconnect
Upgradeable to future GPUs



XK7 Hardware Characteristics

- GPU: NVIDIA K20X (1 Kepler GK110)
  - 2688 processor cores
  - Processor core clock: 732 MHz
  - Memory clock: 2.6 GHz
  - Memory bandwidth (ECC off): 250 GB/sec
  - 6 GB ECC RAM
  - Peak double precision performance: 1311 GFLOPS
  - Peak single precision performance: 3950 GFLOPS

- CPU: AMD 6276 Interlagos
  - 8 Bulldozer Cores, 32 GB RAM
  - 156 GFLOPS peak aggregate CPU performance

Software Environment

- Cray Compilers - Cray Compiling Environment (CCE)
  - Cray OpenACC (C, C++, FORTRAN)
- GNU Compiler Collection (GCC)
  - CUDA C, C++
- Portland Group Inc (PGI) Compilers
  - PGI OpenACC
  - CUDA, PGI OpenACC (C, C++, FORTRAN)
- CUDA Toolkit 5.5
- Cray LibSci

Direct Access to XK node in CCM mode

- qsub -I -l gres=ccm,nodes=1:ppn=16:xk,walltime=01:00:00
- module add ccm
- ccmlogin
- module load craype-accel-nvidia35
- module swap PrgEnv-cray PrgEnv-pgi
- pgaccelinfo

GPU Accelerator Info

CUDA Driver Version: 5050
NVRM version: NVIDIA UNIX x86_64 Kernel Module 319.37 Wed Jul 3 17:08:50 PDT 2013
CUDA Device Number: 0
Device Name: Tesla K20X
Device Revision Number: 3.5
Global Memory Size: 6039339008
Number of Multiprocessors: 14
Number of SP Cores: 2688
Number of DP Cores: 896
Concurrent Copy and Execution: Yes
Total Constant Memory: 65536
Total Shared Memory per Block: 49152
Registers per Block: 65536
Warp Size: 32
Maximum Threads per Block: 1024
Maximum Block Dimensions: 1024, 1024, 64
Maximum Grid Dimensions: 2147483647 x 65535 x 65535
Maximum Memory Pitch: 2147483647B
Texture Alignment: 512B

Clock Rate: 732 MHz
Execution Timeout: No
Integrated Device: No
Can Map Host Memory: Yes
Compute Mode: exclusive-process
Concurrent Kernels: Yes
ECC Enabled: Yes
Memory Clock Rate: 2600 MHz
Memory Bus Width: 384 bits
L2 Cache Size: 1572864 bytes
Max Threads Per SMP: 2048
Async Engines: 2
Unified Addressing: Yes
Initialization time: 24738 microseconds
Current free memory: 5928378368
Upload time (4MB): 1823 microseconds (740 ms pinned)
Download time: 1253 microseconds (696 ms pinned)
Upload bandwidth: 2300 MB/sec (5667 MB/sec pinned)
Download bandwidth: 3347 MB/sec (6026 MB/sec pinned)
PGI Compiler Option: -ta=nvidia,cc35`

CUDA Support on Blue Waters

- CUDA compute capability 3.5
- CUDA C code should be compiled with nvcc
- Cray provides cc and CC wrappers for C/C++ that include support for MPI and OpenMP (use cc and CC instead of mpicc)
- Dynamic linking (static linking is not supported)

Building CUDA Application

- Download NVIDIA example

  http://docs.nvidia.com/cuda/cuda-samples/index.html

  simpleMPI.cpp simpleMPI.cu simpleMPI.h

- Setup CUDA programming environment

  module load cudatoolkit

- Build CUDA code using GNU, PGI, or Cray compiler

  module swap PrgEnv-pgi PrgEnv-cray

  module list

Building CUDA Application, continued

```
nvcc -c -gencode arch=compute_35,code=compute_35 -o simpleMPIcuda.o simpleMPI.cu
CC -o simpleMPI.x simpleMPI.cpp simpleMPIcuda.o
```

Rule: keep all MPI staff in C files and CUDA kernels in CU-files

Run the application

```
#PBS -l nodes=2:ppn=16:xk
cd $PBS_O_WORKDIR
aprun -n2 -N1 ./simpleMPI.x > job.out
```

Examine output

```
cat job.out
  Running on 2 nodes
  Average of square roots is: 0.667279
  PASSED
```

RDMA Support in Kepler K20x GPUs

- **CRAY_CUDA_MPS=[0|1]** default=1 (On) - CUDA proxy - multiple MPI tasks accessing same GPU on the node; turn to off (0) - single MPI task accessing GPU

- **LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH**

- **MPICH_RDMA_ENABLED_CUDA=[0|1]**

  Allows the MPI application to pass GPU pointers directly to point to-point and collective communication functions. If the send or receive buffer for a point-to-point or collective communication is on the GPU, the network transfer and the transfer between the host CPU and the GPU are pipelined to improve performance.

Building GPU-to-GPU Application

OSU micro-benchmarks: **osu_latency_39.c**

Download link http://mvapich.cse.ohio-state.edu/benchmarks/

module swap PrgEnv-cray PrgEnv-gnu (won't work with PGI and Cray)
module load cudatoolkit
cc -D_ENABLE_CUDA_ -o osu_latency39.x osu_latency_39.c

#PBS -l nodes=2:ppn=16:xk
export MPICH_RDMA_ENABLED_CUDA=1
cd $PBS_O_WORKDIR
aprun -n 2 -N 1 ./osu_latency39.x D D > job39.out

Building GPU-to-GPU Application, output

```
# OSU MPI-CUDA Latency Test
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size            Latency (CUDA 5.5)    Latency (CUDA 5.0)
0                         1.75                  1.67
1                        37.80                 88.96
2                        37.70                 88.71
4                        37.60                 88.53
8                        37.59                 88.44
16                       37.64                 88.45
32                       37.57                 88.54
64                       37.62                 88.52
128                      37.64                 88.58
...
32768                    51.52                103.61
65536                    66.79                117.98
131072                   96.20                147.42
262144                  156.03                207.31
524288                  275.45                327.89
1048576                 383.42                479.94
2097152                 604.72                801.52
4194304                1048.64               1435.69
```

PGI CUDA Fortran

Extension of F90 standard by CUDA language constructs

CUDA Fortran file has extension .CUF (compare to .F90)

Building CUDA Fortran application on Cray

wget http://www.pgroup.com/lit/samples/matmul.CUF

module swap PrgEnv-gnu PrgEnv-pgi

module add cudatoolkit

ftn matmul.CUF -o matmul.x

PGI CUDA Fortran, output

```
#PBS -l nodes=1:ppn=16:xk
#PBS -l walltime=0:05:00
cd $PBS_O_WORKDIR
aprun -n1 ./matmul.x > job.out
```

```
cat job.out
   arrays sized          512  by         1024  by          512
   calling mmul
   Kernel time excluding data xfer:    3673.000      microseconds
   Megaflops excluding data xfer:      73083.44
   Total time including data xfer:     421161.0      microseconds
   Megaflops including data xfer:      637.3702
   C(1,1) =    3.5791874E+11
   C(2,2) =    3.5739933E+11
   No errors found
```

OpenACC Programming Model

What is OpenACC

- Inspired by OpenMP
- Implemented by Cray, PGI, CAPS
- Includes functions to query device(s)

How to get started

- http://openacc.org
  - Quick reference guide (OpenMP programmers)
  - Specifications: 1.0 and 2.0a draft
- OpenACC Training
  - Blue Waters joint workshop with PSC, XSEDE, NVIDIA

https://www.psc.edu/index.php/training/xsede-hpc-workshop-november-2013

Cray Compiler OpenACC Support

module load PrgEnv-cray craype-accel-nvidia35

- Fortran
  - -h acc, noomp ! openmp is default, be careful mixing
  - -rm ! include a .lst listing file to show the loop markup
  - -G2 ! -g breaks Cray OpenACC code
- C
  - -h pragma=acc -h nopragma=omp
  - -h msgs # show loop markup in stdout/stderr
  - -Gp # partial optimization

PGI Compiler OpenACC Support

PGI

module load PrgEnv-pgi cudatoolkit

- PGI creates CUDA code as intermediate
  - -ta=nvidia,keepgpu,keepptx
- Fortran , C
  - -acc -ta=nvidia
  - -mcmodel=medium
  - -Minfo=accel

GNU

- Not implemented

OpenACC, example

```
module add craype-accel-nvidia35
module switch PrgEnv-cray PrgEnv-pgi

ftn -acc -Minfo vecAdd-reduction.f90

main:
    31, Generating present_or_copyout(c(1:100000))
        Generating present_or_copyin(a(1:100000))
        Generating present_or_copyin(b(1:100000))
        Generating NVIDIA code
        Generating compute capability 1.3 binary
        Generating compute capability 2.0 binary
        Generating compute capability 3.0 binary
    32, Loop is parallelizable
        Accelerator kernel generated
        32, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
    34, Sum reduction generated for sum
```

OpenACC, output

```
#PBS -l nodes=1:ppn=16:xk
#PBS -l walltime=0:05:00
cd $PBS_O_WORKDIR
aprun -n1 ./a.out > job.out

cat job.out
final result:     1.000000000000000
```

Cray Scientific Libraries

Cray LibSci accelerated BLAS, LAPACK, and ScaLAPACK libraries

PrgEnv-cray or PrgEnv-gnu Programming Environment
module add craype-accel-nvidia35

```
call libsci_acc_init()
… (your code) …
call libsci_acc_finalize()
```

The Library interface automatically initiates appropriate execution mode (CPU, GPU, Hybrid).

When BLAS or LAPACK routines are called from applications built with the Cray or GNU compilers, Cray LibSci automatically loads and links libsci_acc libraries upon execution if it determines performance will be enhanced.

Execution control from source code:

routine_name	invokes automated method
routine_name_cpu	executed on host CPU only
routine_name_acc	executed on GPU only

See “man intro_linsci_acc” for more information.

Cray Scientific Libraries, continued

Libsci_acc is not thread safe. It will fail when called concurrently from OpenMP threads.

ENVIRONMENT VARIABLES

CRAY_LIBSCI_ACC_MODE

Specifies execution mode for libsci_acc routinee:

0 Use automated mode. Adds slight overhead. This is the default.

1 Forces all supported auto-tuned routines to execute on the accelerator.

2 Forces all supported routines to execute on the CPU if the data is located within host processor address space.

LIBSCI_ACC_BYPASS_FUNCTION

Specifies the execution mode for FUNCTION.

0 Use automated mode. Adds slight overhead. This is the default.

1 Call the version that handles data addresses resident on the GPU.

2 Call the version that handles data addresses resident on the CPU.

3 For SGEMM, DGEMM, CGEMM, ZGEMM, this will call accelerated version.

Warning: Passing a wrong CPU / GPU address will cause the program to crash.

Cray Scientific Libraries, examples

module swap PrgEnv-pgi PrgEnv-gnu (or PrgEnv-cray)
module add craype-accel-nvidia35

Download examples:
echo $LIBSCI_ACC_EXAMPLES_DIR

ftn ${LIBSCI_ACC_EXAMPLES_DIR}/fortran_simple/f_dgemm_simple.f90

```
#PBS -l nodes=1:ppn=16:xk
#PBS -l walltime=0:05:00
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=16
aprun -n1 -cc none ./a.out > job.out
```

Cray recommends “-cc none” that allows OpenMP threads to migrate.

Cray Scientific Libraries, output

	Cray PE		GNU PE		
N	HYBRID GF/s	CPU GF/s	HYBRID GF/s	CPU GF/s	ERROR
512	94.949	54.362	94.853	68.356	3.9E-13
1024	252.288	93.548	251.198	98.459	1.0E-12
1536	360.780	101.800	359.211	108.308	2.1E-12
2048	428.374	109.323	417.169	115.251	1.5E-12
2560	532.272	113.391	529.767	110.312	1.7E-12
3072	664.947	116.206	667.482	115.191	1.9E-12
3584	672.476	117.848	669.742	116.186	2.1E-12
4096	723.276	112.808	721.233	112.025	2.2E-12
4608	646.523	115.176	646.781	114.086	2.7E-12
5120	724.432	116.251	722.819	115.456	2.7E-12
5632	729.633	116.475	727.661	114.683	2.9E-12
6144	836.011	117.397	829.653	116.332	3.1E-12
6656	759.293	118.324	744.176	117.134	3.1E-12
7168	728.308	119.651	746.932	118.111	3.4E-12
7680	701.924	116.264	749.823	115.137	3.8E-12
8192	727.964	109.861	759.400	110.145	3.6E-12

Final Remarks

Further Reading about GPU Programming:

- http://bluewaters.ncsa.illinois.edu
- http://www.xsede.org
- http://docs.cray.com
- http://developer.nvidia.com
- http://openacc.org

Questions, Comments, Suggestions

help+bw@ncsa.illinois.edu