GPU Programming Platforms and Running

Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu
http://geco.mines.edu/tesla
Where to run?

• Cuda is supported on systems with newer Nvidia graphics cards

• Examples:
  • Current generation of Apple Mac laptops
  • Mac Pro with NVIDIA GeForce GT 120 or aftermarket GTX 285
  • Linux boxes is Green Center 297 (coming)

• Can run in emulation mode on machines that don’t have a NVIDIA card
cudal.mines.edu

- Purchased using TechFee money
- Front end
  - Dell 1950 node
  - 24 Gbytes of Ram
- PCI 8x connection to Tesla S1070
<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Tesla GPUs</td>
<td>4</td>
</tr>
<tr>
<td># of Streaming Processor Cores</td>
<td>960 (240 per processor)</td>
</tr>
<tr>
<td>Frequency of processor cores</td>
<td>1.296 to 1.44 GHz</td>
</tr>
<tr>
<td>Single Precision floating point performance (peak)</td>
<td>3.73 to 4.14 TFlops</td>
</tr>
<tr>
<td>Double Precision floating point performance (peak)</td>
<td>311 to 345 GFlops</td>
</tr>
<tr>
<td>Floating Point Precision</td>
<td>IEEE 754 single &amp; double</td>
</tr>
<tr>
<td>Total Dedicated Memory</td>
<td>16 GB</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>512-bit</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>408 GB/sec</td>
</tr>
<tr>
<td>Max Power Consumption</td>
<td>800 W</td>
</tr>
<tr>
<td>System Interface</td>
<td>PCIe x16 or x8</td>
</tr>
<tr>
<td>Software Development Tools</td>
<td>C-based CUDA Toolkit</td>
</tr>
</tbody>
</table>

deviceQuery - example

[tkaiser@cuda1 release]$ ./deviceQuery
CUDA Device Query (Runtime API) version (CUDART static linking)
There are 4 devices supporting CUDA

Device 0: "Tesla C1060"
  CUDA Capability Major revision number: 1
  CUDA Capability Minor revision number: 3
  Total amount of global memory: 4294705152 bytes
  Number of multiprocessors: 30
  Number of cores: 240
  Total amount of constant memory: 65536 bytes
  Total amount of shared memory per block: 16384 bytes
  Total number of registers available per block: 16384
  Warp size: 32
  Maximum number of threads per block: 512
  Maximum sizes of each dimension of a block: 512 x 512 x 64
  Maximum sizes of each dimension of a grid: 65535 x 65535 x 1
  Maximum memory pitch: 262144 bytes
  Texture alignment: 256 bytes
  Clock rate: 1.44 GHz
  Concurrent copy and execution: Yes
  Run time limit on kernels: No
  Integrated: No
  Support host page-locked memory mapping: Yes
  Compute mode: Default (multiple host threads can use this device simultaneously)
Running on cuda1

[floydl@ra ~]$ ssh cuda1
Last login: Tue Jul 14 10:06:34 2009 from ra.mines.edu
[floydl@cuda1 ~]$

You will want to add the following lines to your .bash_profile file on cuda1 to enable you to access the cuda compilers.

```bash
export LD_LIBRARY_PATH=/usr/local/cuda/lib:/usr/local/cuda/cudaprof/bin:$LD_LIBRARY_PATH
export PATH=$PATH:/usr/local/cuda/bin
export PATH=$PATH:$HOME/cuda/bin/linux/release
export MANPATH=$MANPATH:/usr/local/cuda/man
```

Next copy the example programs to your home directory.

[floydl@cuda1 ~]$ cp -r /usr/local/cuda_sdk cuda

Finally, log out and log back in to set your path correctly.
Running on cuda1

After you log back in go to your cuda directory and type make. This will build the cuda example programs.

The examples will be in the $HOME/cuda/bin/linux/release directory.

[floyd1@cuda1 ~]$ cd cuda
[floyd1@cuda1 cuda]$ ls
bin common doc lib Makefile projects releaseNotesData ReleaseNotes.html tools
[floyd1@cuda1 cuda]$ make
make[1]: Entering directory `/home/floyd1/cuda/common'
a - obj/release/bank_checker.cpp.o
a - obj/release/cmd_arg_reader.cpp.o
a - obj/release/cutil.cpp.o
a - obj/release/stopwatch.cpp.o
a - obj/release/stopwatch_linux.cpp.o
...
...
[floyd1@cuda1 ~]$ cd $HOME/cuda/bin/linux/release
[floyd1@cuda1 release]$ ls
alignedTypes          dwtHaar1D           nbody                   simpleStreams
asyncAPI              dxtc                oceanFFT                simpleTemplates
bandwidthTest         eigenvalues         particles               simpleTexture
...
...
deviceQueryDrv        MonteCarloMultiGPU  simpleMultiGPU          volumeRender
[floyd1@cuda1 release]$

Note: Procedure on Green Center 279 Machines should be the same
nvcc Compiler

[tkaiser@cudal release]$ nvcc --help

Usage : nvcc [options] <inputfile>

Options for specifying the compilation phase
============================================

More exactly, this option specifies up to which stage the input files must be compiled, according to the following compilation trajectories for different input file types:

- .c/.cc/.cpp/.cxx : preprocess, compile, link
- .i/.ii : compile, link
- .cu : preprocess, cuda frontend, ptxassemble, merge with host C code, compile, link
- .gpu : nvopencc compile into cubin
- .ptx : ptxassemble into cubin.

--device-emulation (-deviceemu)
Generate code for the GPGPU emulation library.
“hello world” example

• Cuda Kernel routines can not print unless compiled in emulation mode
• Reads in dimension of the grid \((gx,gy)\)
• Reads in dimension of block within grid \((bx,by,bz)\)
• Number of threads \(= gx*gy*bx*by*bz\)
• Allocates a local and device array \(nthreads*6\)
• Calls kernel
Kernel Uses:

- `gridDim.x` `gridDim.y`
- Size of grid
- `blockIdx.x` `blockIdx.y`
- Which block for thread
- `blockDim.x` `blockDim.y` `blockDim.z`
- Size of each block
- `threadIdx.x` `threadIdx.y` `threadIdx.z`
- Thread within block

Calculates a “natural” thread id and stuffs everything in an array
```c
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>

void checkCUDAError(const char *msg);

__global__ void Kernel(int *dat);

main() {
    int *dat_local, *dat_remote;
    int gx, gy;
    int bx, by, bz;
    int size;
    int numthreads, j;

    scanf("%d %d", &gx, &gy);
    scanf("%d %d %d", &bx, &by, &bz);

    dim3 dimGrid(gx, gy);
    dim3 dimBlock(bx, by, bz);

    numthreads = gx * gy * bx * by * bz;

    size = 6 * sizeof(int) * numthreads;
    cudaMalloc((void**) &dat_remote, size);
    checkCUDAError("cudaMalloc");
    dat_local = (int*)malloc(size);
    checkCUDAError("malloc");
    Kernel<<<dimGrid, dimBlock>>>(dat_remote);
    checkCUDAError("Kernel");
    ...
    ...
```
__global__ void Kernel(int *dat) {
    /* get my block within a grid */
    int myblock=blockIdx.x+blockIdx.y*gridDim.x;
    /* how big is each block within a grid */
    int blocksize=blockDim.x*blockDim.y*blockDim.z;
    /* get thread within a block */
    int subthread=threadIdx.z*(blockDim.x*blockDim.y)+threadIdx.y*blockDim.x+threadIdx.x;
    /* find my thread */
    int thread=myblock*blocksize+subthread;
    #if __DEVICE_EMULATION__
        printf("gridDim=\(\%3d \%3d\) blockIdx=\(\%3d \%3d\) blockDim=\(\%3d \%3d \%3d\) threadIdx=\(\%3d \%3d \%3d\) \%6d\n",
            blockDim.x,blockDim.y,
            blockIdx.x,blockIdx.y,
            blockDim.x,blockDim.y,blockDim.z,
            threadIdx.x,threadIdx.y,threadIdx.z,thread);
    #endif
    /* starting index into array */
    int index=thread*6;
    dat[index]=thread;
    dat[index+1]=blockIdx.x;
    dat[index+2]=blockIdx.y;
    dat[index+3]=threadIdx.x;
    dat[index+4]=threadIdx.y;
    dat[index+5]=threadIdx.z;
}
cudaMemcpy(dat_local, dat_remote, size, cudaMemcpyDeviceToHost);
for(int i=0;i<numthreads;i++) {
  j=i*6;
  printf("%6d %3d %3d %3d %3d %3d\n",
         dat_local[j],
         dat_local[j+1],dat_local[j+2],
         dat_local[j+3],dat_local[j+4],dat_local[j+5]);
}

void checkCUDAError(const char *msg)
{
  cudaError_t err = cudaGetLastError();
  if( cudaSuccess != err)
  {
    fprintf(stderr, "Cuda error: %s: %s.\n", msg,
             cudaGetErrorString(err));
    exit(EXIT_FAILURE);
  }
}
void checkCUDAError(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if( cudaSuccess != err )
    {
        fprintf(stderr, "Cuda error: %s: %s.\n", msg,
                       cudaGetErrorString( err ) );
        exit(EXIT_FAILURE);
    }
}
[peloton:~/wave3d] tkaiser% nvcc testinput.cu  -o testinput
[peloton:~/wave3d] tkaiser% ./testinput

2 3
2 3 4

0 0 0 0 0 0
1 0 0 1 0 0
2 0 0 0 1 0
3 0 0 1 1 0
4 0 0 0 2 0
5 0 0 1 2 0
6 0 0 0 0 1
7 0 0 1 0 1
8 0 0 0 1 1

...

134 1 2 0 1 2
135 1 2 1 1 2
136 1 2 0 2 2
137 1 2 1 2 2
138 1 2 0 0 3
139 1 2 1 0 3
140 1 2 0 1 3
141 1 2 1 1 3
142 1 2 0 2 3
143 1 2 1 2 3

[peloton:~/wave3d] tkaiser%