ARCHER Champions 2 workshop

Mike Giles

Mathematical Institute & OeRC, University of Oxford

Sept 5th, 2016

Tier 2 bids

Out of the 8 bids, I know only about 4:

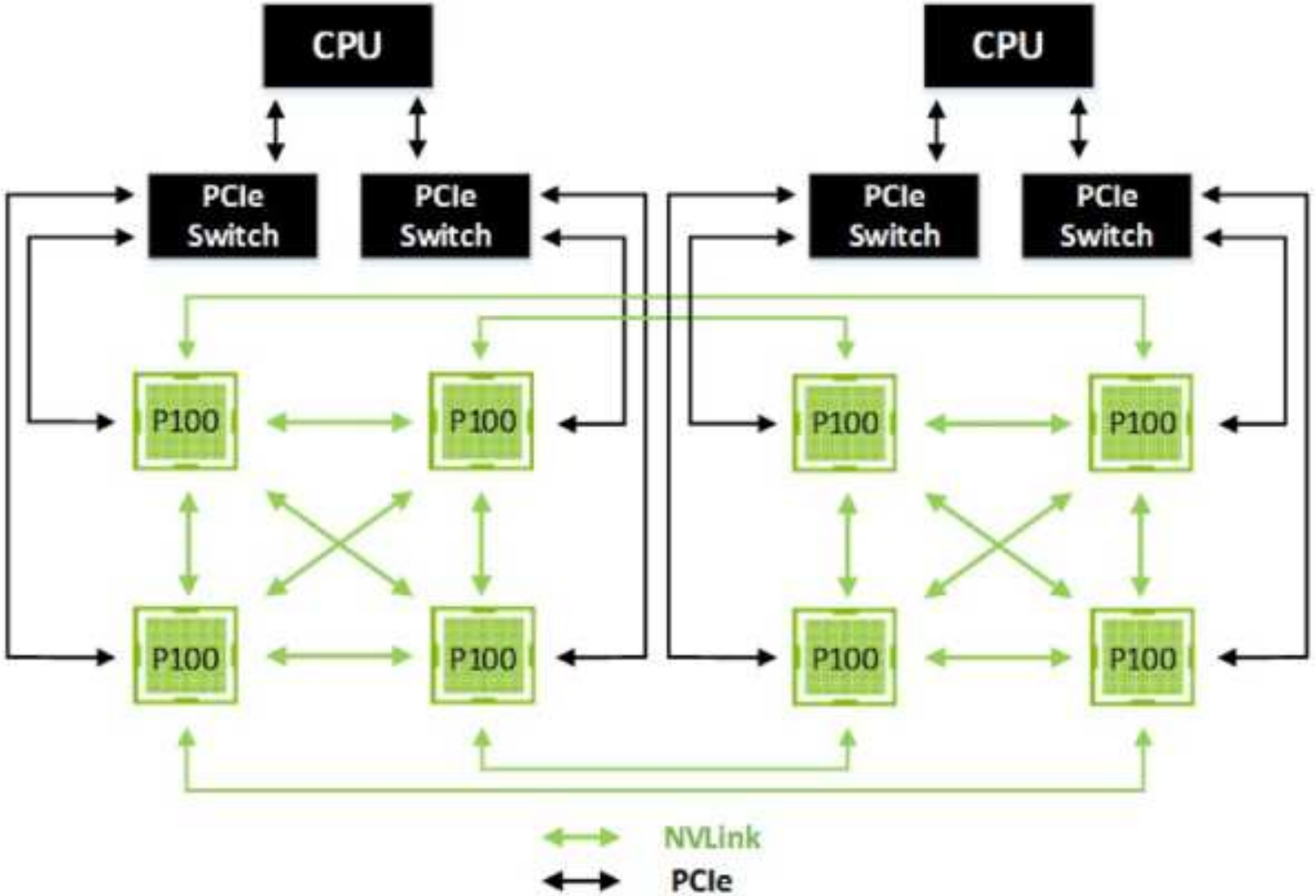
- Cambridge: £5M
  - mainly x86, but also some GPUs and Xeon Phi (KNL)
  - national facility, free for all
- UCL: £4M
  - x86 system for materials researchers
  - restricted to materials researchers, with usage charges
- Bristol: £3M
  - CRAY system with ARM processors
  - national facility, free for all?
- Oxford: £3M
  - GPU system
  - national facility, free for all

JADE

- Joint Academic Data science Endeavour
- I am the PI, but really a national facility with Co-Is from Bristol, Edinburgh, KCL, Oxford, QMUL, Sheffield, Southampton, UCL
- primary motivation is needs of machine learning groups in Alan Turing Institute, Edinburgh, KCL, Oxford, QMUL, UCL
- NVIDIA GPU system intended for
  - machine learning (50%)
  - molecular dynamics (30%)
  - other applications (20%)

JADE

Based on NVIDIA's DGX-1 Deep Learning server



JADE

Each NVIDIA P100 GPU has:

- 3584 cores
- 16 GB of 720GB/s HBM2 memory
- 4 20GB/s bi-directional NVlink inter-connections

and is capable of 20/10/5 TFlops in half/single/double precision

In addition, each node has:

- 2 20-core Intel Xeons (E5-2698 v4)
- 512 GB DDR4 memory and 8TB SSD
- 4 single-port Infiniband EDR cards connected to the PCIe switches

JADE

Systems come pre-loaded with highly optimised versions of machine learning packages Caffe, Torch, Theano – benchmarking shows the P100 is twice as fast as new Intel Xeon Phi

We will also install optimised versions of key molecular dynamics packages GROMACS, Amber and NAMD.

In addition they will have the usual NVIDIA compilers, libraries, etc, for those developing their own applications.

JADE

There will be new/existing RSE support at

- Edinburgh (Alan Gray)
- KCL
- Oxford (Ian Bush + 1)
- QMUL
- Sheffield (Paul Richmond)
- Southampton

In addition there will be training at

- Oxford (my CUDA course)
- Edinburgh (Alan Gray)
- Sheffield (Paul Richmond)

JADE

Operationally, system will be procured from a prime contractor who will cover all operating costs by selling some cycles to industry

System might be hosted at STFC Hartree or at EPCC

Access will be free through a Resource Allocation Panel, similar to ARCHER

If over-subscribed, priority will be given to groups involved in Tier 2 bid

GPU Programming

Various options, from no effort to a little:

- use existing third-party software
  e.g. Caffe, Torch, Theano, GROMACS, Amber, NAMD
- use CPU libraries like BLAS which automatically off-load computations to the GPU
- use Thrust, a CPU-level C++ package which has GPU-based "objects" and off-loads computations
- use OpenACC / OpenMP 4.0 to write CPU-level code which off-loads computations for certain parts
- use NVIDIA's CUDA (or OpenCL) to write GPU-specific code with explicit or implicit data transfer from/to CPU

GPU Programming

Using CUDA is fairly straightforward:

- some code runs on host – mainly command-and-control, launching "kernel" code on GPU, and handling data movement to/from GPU
- "kernel" code is executed on the GPU – execution involves multiple blocks of threads, each block executing on one GPU Streaming Multiprocessor functional unit
- main thing to note is that kernel code is executed from point of view of a single GPU thread – special variables identify which block it is in, and which thread within the block (similar to MPI)

Kernel code

```
#include <helper_cuda.h>

__global__ void my_first_kernel(float *x)
{
 int tid = threadIdx.x + blockDim.x*blockIdx.x;

 x[tid] = (float) threadIdx.x;
}
```

- `__global__` identifier says it's a kernel function
- each thread sets one element of `x` array
- within each block of threads, `threadIdx.x` ranges from 0 to `blockDim.x-1`, so each thread has a unique value for `tid`

Host code

```
int main(int argc, char **argv) {
 float *h_x, *d_x;          //  h=host, d=device
 int   nblocks=2, nthreads=8, nsize=2*8;

 h_x = (float *)malloc(nsize*sizeof(float));
 cudaMalloc((void **)&d_x,nsize*sizeof(float));

 my_first_kernel<<<nblocks,nthreads>>>(d_x);

 cudaMemcpy(h_x,d_x,nsize*sizeof(float),
            cudaMemcpyDeviceToHost);

 for (int n=0; n<nsize; n++)
   printf(" n,  x  =  %d  %f \n",n,h_x[n]);

 cudaFree(d_x); free(h_x);
}
```

Host code

New version with managed memory

```
int main(int argc, char **argv) {
 float *x;
 int   nblocks=2, nthreads=8, nsize=2*8;

 cudaMallocManaged(&x, nsize*sizeof(float));

 my_first_kernel<<<nblocks,nthreads>>>(x);

 for (int n=0; n<nsize; n++)
   printf(" n,  x  =  %d  %f \n",n,x[n]);

 cudaFree(x);
}
```

More information

For extensive lectures and many practicals, see my course page:
`http://people.maths.ox.ac.uk/gilesm/cuda/`

CUDA also comes with

- debugger
- performance profiler
- integration in Eclipse / Visual Studio NSight IDE
- lots of numerical libraries
  - cuBLAS
  - cuFFT
  - cuSPARSE
  - cuRAND