Hybrid MPI+OpenMP+CUDA Programming

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Biological Sciences
University of Southern California

Email: anakano@usc.edu

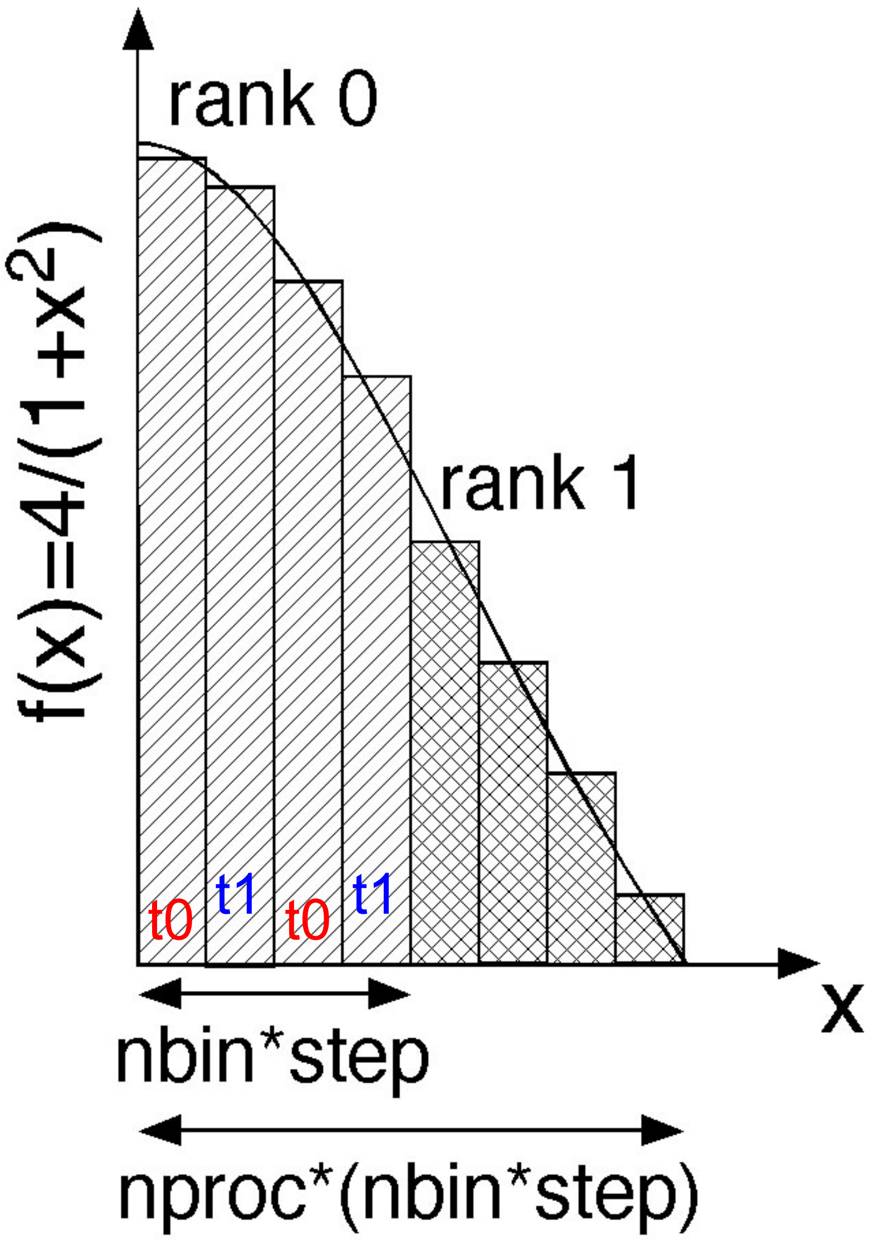
Standard programming on GPU-accelerated clusters

https://hpcc.usc.edu/support/documentation/gpucluster/

MPI+CUDA Calculation of π

- **Spatial decomposition: Each MPI process integrates over a range of width `1/nproc`, as a discrete sum of `nbin` bins each of width `step`**
- **Interleaving: Within each MPI process, `NUM_BLOCK*NUM_THREAD` CUDA threads perform part of the sum**

$$\pi = \int_0^1 \frac{4}{1+x^2}dx \cong \Delta \sum_{i=0}^{N-1} \frac{4}{1+x_i^2}$$

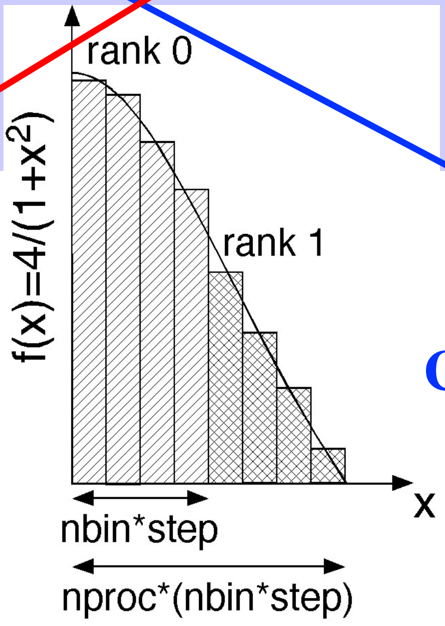


Calculate Pi with MPI+CUDA: `hypi.cu` (1)

```
#include <stdio.h>
#include <mpi.h>
#include <cuda.h>


#define NBIN  10000000  // Number of bins
#define NUM_BLOCK   13  // Number of thread blocks
#define NUM_THREAD 192  // Number of threads per block


// Kernel that executes on the CUDA device
__global__ void cal_pi(float *sum,int nbin,float step,float offset,int nthreads,int nblocks)
{
   int i;
   float x;
   int idx = blockIdx.x*blockDim.x+threadIdx.x;  // Sequential thread index across blocks
   for (i=idx; i< nbin; i+=nthreads*nblocks) {    // Interleaved bin assignment to threads
      x = offset+(i+0.5)*step;
      sum[idx] += 4.0/(1.0+x*x);
   }
}
```



MPI spatial decomposition

CUDA thread interleaving

Calculate Pi with MPI+CUDA: `hypi.cu` (2)

```
int main(int argc,char **argv) {
   int myid,nproc,nbin,tid;
   float step,offset,pi=0.0,pig;
   dim3 dimGrid(NUM_BLOCK,1,1);  // Grid dimensions (only use 1D)
   dim3 dimBlock(NUM_THREAD,1,1);  // Block dimensions (only use 1D)
   float *sumHost,*sumDev;  // Pointers to host & device arrays
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);  // My MPI rank
   MPI_Comm_size(MPI_COMM_WORLD,&nproc);  // Number of MPI processes
   nbin = NBIN/nproc;  // Number of bins per MPI process
   step = 1.0/(float)(nbin*nproc);  // Step size with redefined number of bins
   offset = myid*step*nbin;  // Quadrature-point offset
   size_t size = NUM_BLOCK*NUM_THREAD*sizeof(float);  //Array memory size
   sumHost = (float *)malloc(size);  //  Allocate array on host
   cudaMalloc((void **) &sumDev,size);  // Allocate array on device
   cudaMemset(sumDev,0,size);  // Reset array in device to 0
   // Calculate on device (call CUDA kernel)
   cal_pi <<<dimGrid,dimBlock>>> (sumDev,nbin,step,offset,NUM_THREAD,NUM_BLOCK);
   // Retrieve result from device and store it in host array
   cudaMemcpy(sumHost,sumDev,size,cudaMemcpyDeviceToHost);
   // Reduction over CUDA threads
   for(tid=0; tid<NUM_THREAD*NUM_BLOCK; tid++) pi += sumHost[tid];
   pi *= step;
   // CUDA cleanup
   free(sumHost);
   cudaFree(sumDev);
   printf("myid = %d: partial pi = %f\n",myid,pi);
   // Reduction over MPI processes
   MPI_Allreduce(&pi,&pig,1,MPI_FLOAT,MPI_SUM,MPI_COMM_WORLD);
   if (myid==0) printf("PI = %f\n",pig);
   MPI_Finalize();
   return 0;}
```

Compiling MPI+CUDA on HPC

- **Set an environment on the front-end (ssh to `hpc-login3.usc.edu`)**

```
source /usr/usc/openmpi/default/setup.sh
```
(if bash)
```
source /usr/usc/cuda/default/setup.sh
```
or
```
source /usr/usc/openmpi/default/setup.csh
```
(if tcsh)
```
source /usr/usc/cuda/default/setup.csh
```

- **Compilation (in fact, this is for MPI+OpenMP+CUDA)**

```
nvcc -Xcompiler -fopenmp hypi.cu -o hypi
  -I/usr/usc/openmpi/default/include
  -L/usr/usc/openmpi/default/lib -lmpi -lgomp
```

Interactive Run on HPC

Here, we assume that you have included the source commands (in the previous slide) to set up interoperable OpenMPI & CUDA environments within your .bashrc or .cshrc in your home directory

```
[anakano@hpc-login3 ~]$ salloc --nodes=2 --ntasks-per-node=1
                        --cpus-per-task=1 --gres=gpu:1 -t 29
salloc: Pending job allocation 2140476
salloc: job 2140476 queued and waiting for resources
salloc: job 2140476 has been allocated resources
salloc: Granted job allocation 2140476
salloc: Waiting for resource configuration
salloc: Nodes hpc[3820,3823] are ready for job
[anakano@hpc3820 anakano]$ srun -n 2 ./hypi
myid = 1: partial pi = 1.287001
myid = 0: partial pi = 1.854596
PI = 3.141597
```

large main quick	hpc3817 – hpc3834, hpc3852	19	64 GB	16	2.6	xeon	2	k40	avx avx2	nx360m5	1.79 TB	IB

Variation: Using 2 GPUs per Node (1)

- **Run multiple MPI processes on each node, and assign different GPUs to different processes**

hypi_setdevice.cu

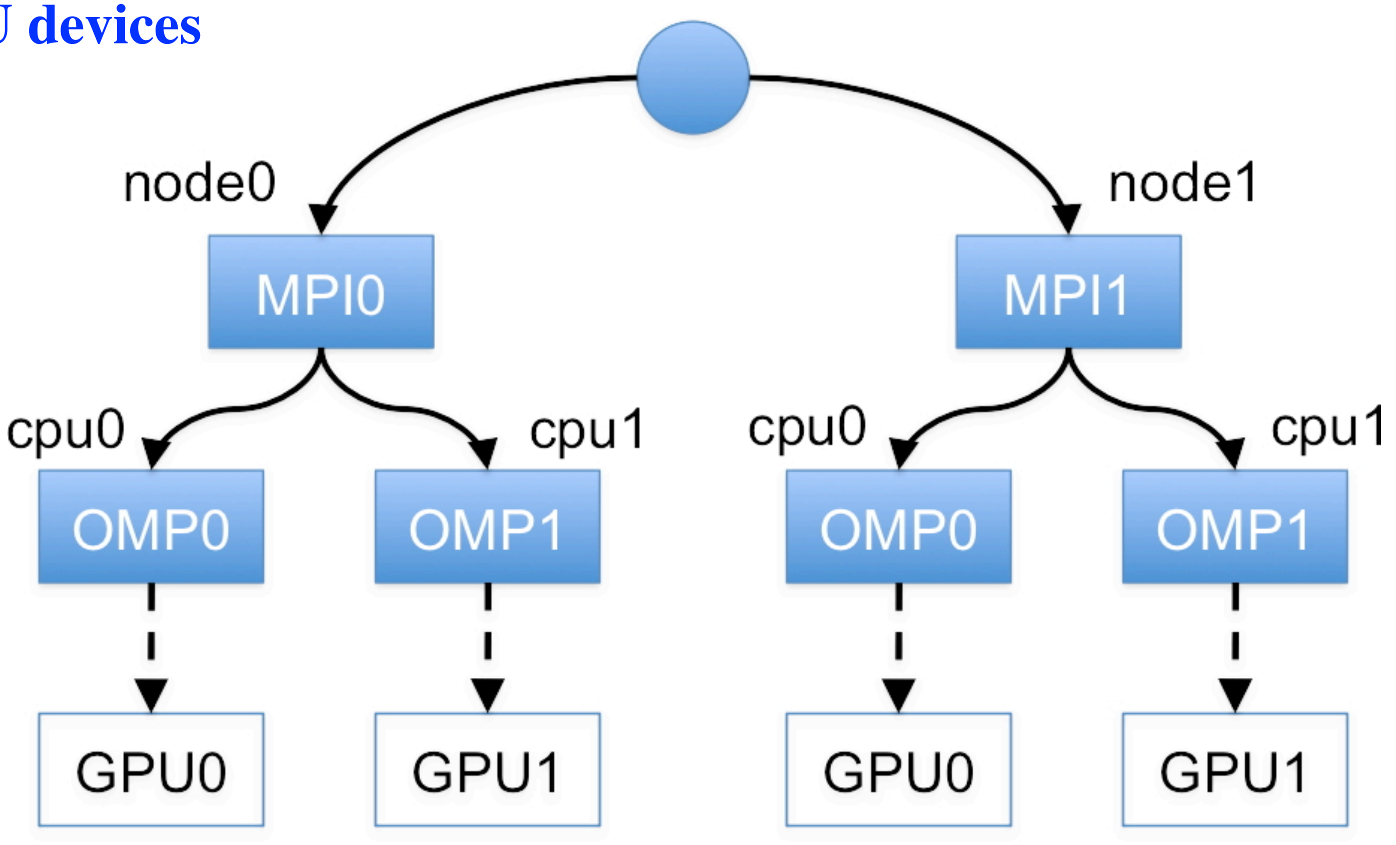
```
int main(int argc,char **argv) {
  int dev_used;
  ...
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);  // My MPI rank
  cudaSetDevice(myid%2);  // Pick one of the 2 GPUs (0 or 1)
  ...
  cudaGetDevice(&dev_used);  // Find which GPU is being used
  printf("myid = %d: device used = %d; partial pi = %f\n”,myid,dev_used,pi);
  ...
}
```

Variation: Using 2 GPUs per Node (2)

```
[anakano@hpc-login3 ~/work596]$ salloc --nodes=2 --ntasks-per-node=2
--cpus-per-task=1 --gres=gpu:2 -t 29
salloc: Pending job allocation 2140495
salloc: job 2140495 queued and waiting for resources
salloc: job 2140495 has been allocated resources
salloc: Granted job allocation 2140495
salloc: Waiting for resource configuration
salloc: Nodes hpc[3820-3821] are ready for job
[anakano@hpc3820 anakano]$ srun -n 4 ./hypi_setdevice
myid = 0: device used = 0; partial pi = 0.979926
myid = 1: device used = 1; partial pi = 0.874671
myid = 2: device used = 0; partial pi = 0.719409
myid = 3: device used = 1; partial pi = 0.567582
PI = 3.141588
```

MPI+OpenMP+CUDA Computation of π

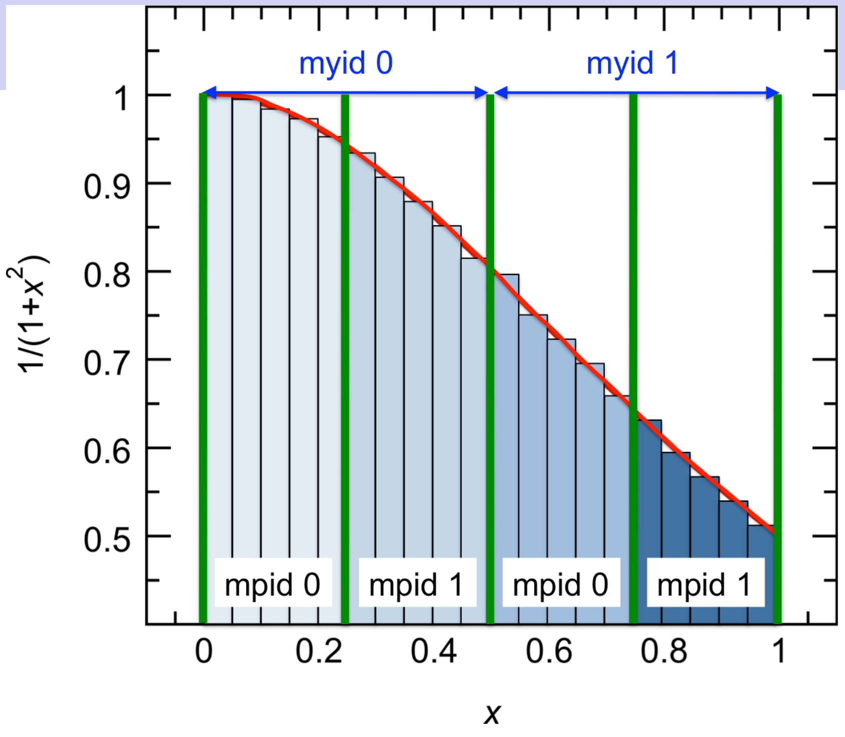
- Write a triple-decker MPI+OpenMP+CUDA program, `pi3.cu`, by inserting an OpenMP layer to the double-decker MPI+CUDA program, `hypi_setdevice.cu`
- Launch one MPI rank per node, where each rank spawns two OpenMP threads that run on different CPU cores & use different GPU devices



MPI+OpenMP Spatial Decompositions

```
#include <omp.h>
#define NUM_DEVICE   2  // # of GPU devices = # of OpenMP threads
...
// In main()
MPI_Comm_rank(MPI_COMM_WORLD,&myid);    // My MPI rank
MPI_Comm_size(MPI_COMM_WORLD,&nproc);   // # of MPI processes
omp_set_num_threads(NUM_DEVICE);        // One OpenMP thread per GPU device
nbin = NBIN/(nproc*NUM_DEVICE);         // # of bins per OpenMP thread
step = 1.0/(float)(nbin*nproc*NUM_DEVICE);
#pragma omp parallel private(list the variables that need private copies)
{
  mpid = omp_get_thread_num();
  offset = (NUM_DEVICE*myid+mpid)*step*nbin;  // Quadrature-point offset
  cudaSetDevice(mpid%2);
  ...
}
```

- **For the CUDA layer, leave the interleaved assignment of quadrature points to CUDA threads in `hypi_setdevice.cu` as it is**



Data Privatization

- Circumvent the race condition for variable `pi`, by defining a private accumulator per OepnMP thread (or GPU device):

```
float pid[NUM_DEVICE];
```

- Use the array elements as dedicated accumulators for the OepnMP threads
- Upon exiting from the OpenMP parallel section, perform reduction over the elements of `pid[]` to obtain the partial sum, `pi`, per MPI rank

- Alternatively use (recall false sharing)

```
#pragma omp parallel reduction(+:pi)
```

Output

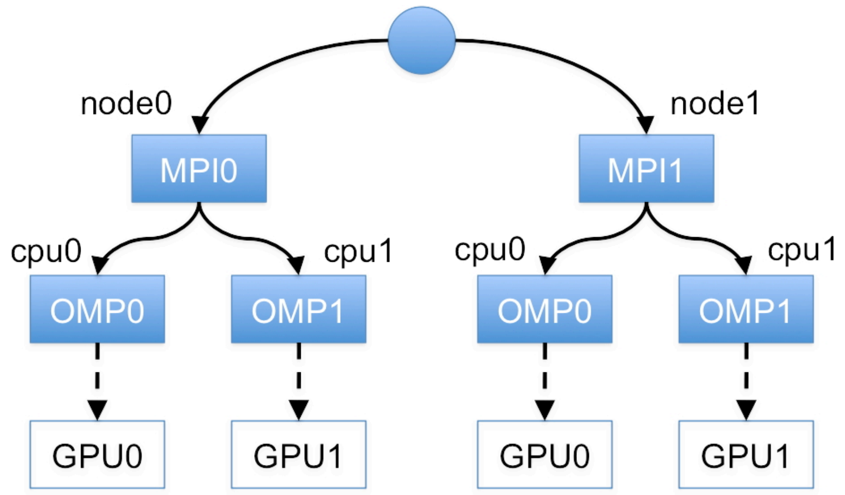
- **To report which of the two GPUs has been used for the run, insert the following lines within the OpenMP parallel block:**

```
cudaGetDevice(&dev_used);
printf("myid = %d; mpid = %d: device used = %d; partial pi =
%f\n", myid, mpid, dev_used, pi);
```

MPI rank — **OpenMP thread ID** — **ID of the GPU device (0 or 1) that was used** — **Partial sum per OpenMP thread or `pid[mpid]` if data privatized manually**

- **Output**

```
myid = 0; mpid = 0: device used = 0; partial pi = 0.979926
myid = 0; mpid = 1: device used = 1; partial pi = 0.874671
myid = 1; mpid = 0: device used = 0; partial pi = 0.719409
myid = 1; mpid = 1: device used = 1; partial pi = 0.567582
PI = 3.141588
```



Compiling MPI+OpenMP+CUDA on HPC

- Set an environment on the front-end (ssh to `hpc-login3.usc.edu`)

```
source /usr/usc/openmpi/default/setup.sh
```
(if bash)
```
source /usr/usc/cuda/default/setup.sh
```
or
```
source /usr/usc/openmpi/default/setup.csh
```
(if tcsh)
```
source /usr/usc/cuda/default/setup.csh
```

- Compilation

nvcc option to pass the following option (-fopenmpi) to gcc

```
nvcc -Xcompiler -fopenmp pi3.cu -o pi3
  -I/usr/usc/openmpi/default/include
  -L/usr/usc/openmpi/default/lib -lmpi -lgomp
```

Running MPI+OpenMP+CUDA on HPC

- **Submit the following Slurm script using the sbatch command**

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu:2
#SBATCH --time=00:00:59
#SBATCH --output=pi3.out
#SBATCH -A lc_an2
source /usr/usc/openmpi/default/setup.sh
source /usr/usc/cuda/default/setup.sh
WORK_HOME=/home/rcf-proj/an2/YourID
cd $WORK_HOME
srun -n 2 ./pi3
```

No need if in .bashrc

- **Output**

```
myid = 1; mpid = 1: device used = 1; partial pi = 0.567582
myid = 1; mpid = 0: device used = 0; partial pi = 0.719409
myid = 0; mpid = 0: device used = 0; partial pi = 0.979926
myid = 0; mpid = 1: device used = 1; partial pi = 0.874671
PI = 3.141588
```

Q: Why MPI+OpenMP+CUDA?

A: All US supercomputers will be GPU-accelerated.

Q: Why calculus (quantum dynamics) on GPU?

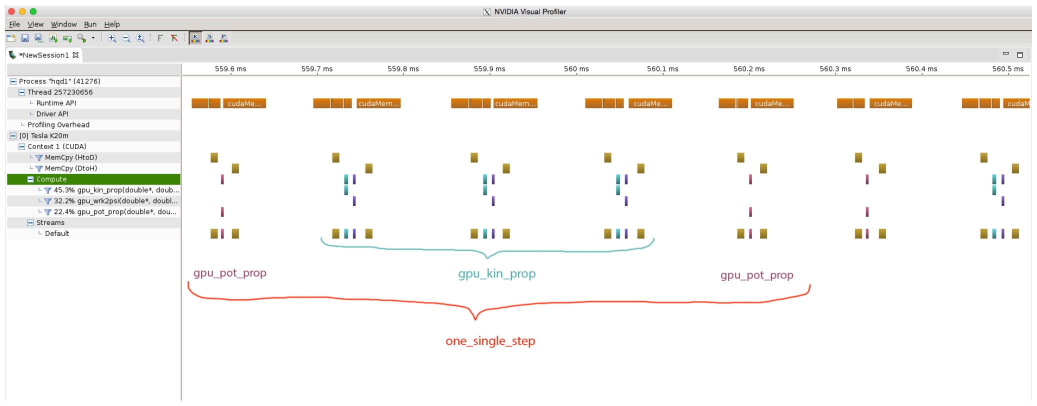
A: Differentiable machine learning for all.
(Example) Natural language processing (NLP)
DP (dynamic programming) → DL (deep learning) → DiffL (differentiable learning)

Take-home lessons:

- GPU-offload basics: host2device → kernel → device2host
- Multiple GPUs per node (6 on Summit):
  cudaSetDevice(OMP thread ID%NUM_DEVICE) ← cudaGetDeviceCount(int *)

Where to go from here = make it orders-of-magnitude faster:

- Overlap CPU+GPU computations: Persistent & asynchronous kernels
- Minimize CPU-GPU communication


NVIDIA Visual Profiler
File View Window Run Help
*NewSession1
559.6 ms
559.7 ms
559.8 ms
559.9 ms
560 ms
560.1 ms
560.2 ms
560.3 ms
560.4 ms
560.5 ms
Process "hqd1" (41276)
Thread 257230656
Runtime API
Driver API
Profiling Overhead
[0] Tesla K20m
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
45.3% gpu_kin_prop(double*, doub...
32.2% gpu_wrk2psi(double*, doubl...
22.4% gpu_pot_prop(double*, dou...
Streams
Default
cudaMe...
cudaMem...
gpu_pot_prop
gpu_kin_prop
gpu_pot_prop
one_single_step