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 Quantitative & Computational Biology
University of Southern California

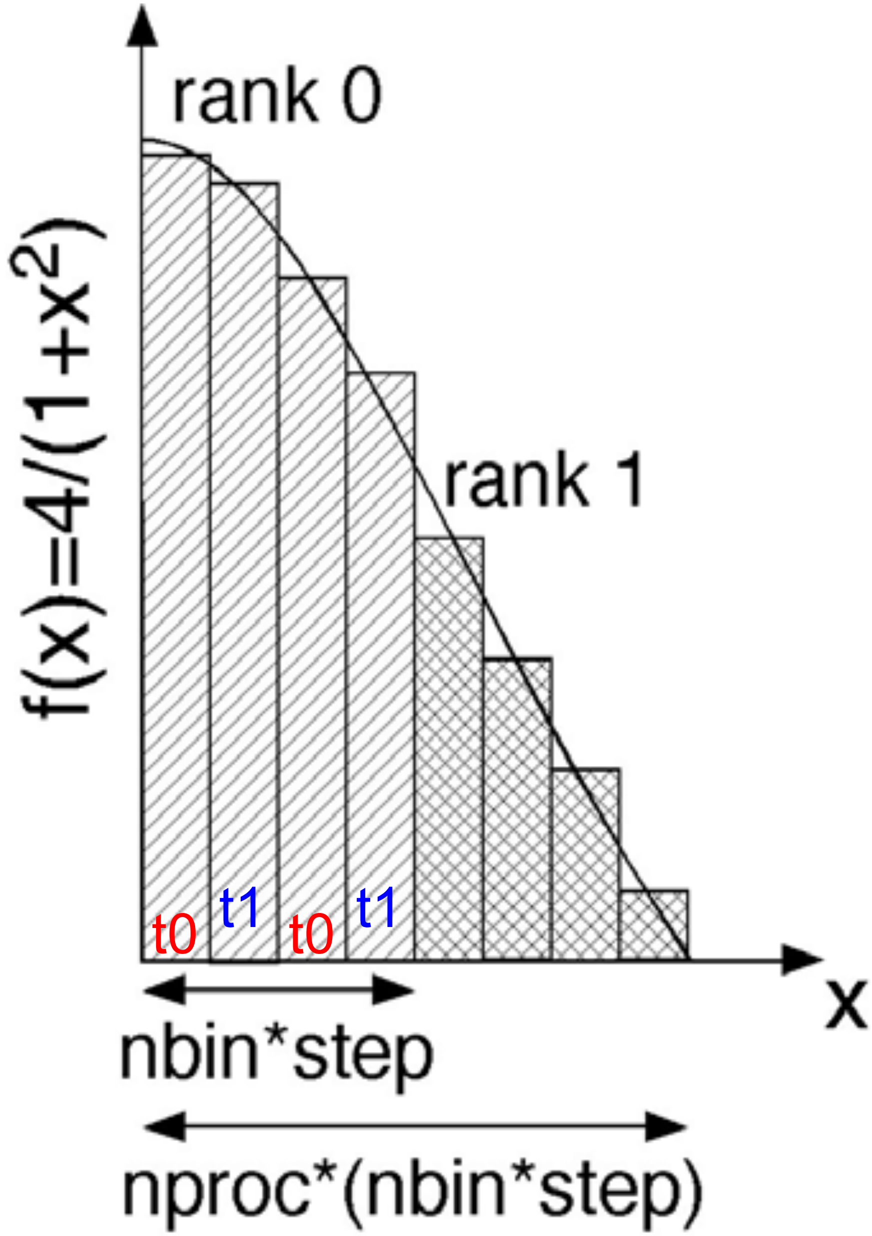
Email: anakano@usc.edu

Objective: Hands-on experience in MPI+OpenMP+CUDA programming for hybrid parallel computing on a cluster of GPU-accelerated multicore computing nodes

MPI+CUDA Calculation of π

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

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



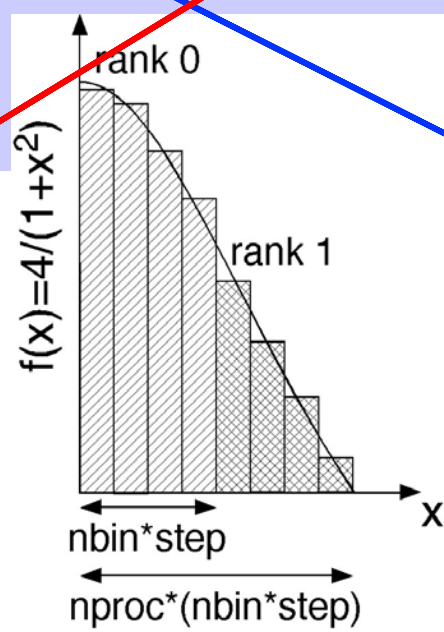
cf. Hybrid MPI+OpenMP program, https://aiichironakano.github.io/cs596/src/hybrid/hpi.c

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 *via* offset (how many bins have been computed before me?)

CUDA thread interleaving (give way bins to the other threads)

cf. Kernel in https://aiichironakano.github.io/cs596/src/cuda/pi.cu

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 Discovery

- **Set an environment (add the following lines in your `.bashrc`)**

```
module purge
module load usc
module load cuda/10.1.243
```

- **Compilation (this also works for MPI+OpenMP+CUDA programs) — this should be typed all in one line:**

```
nvcc -Xcompiler -fopenmp hypi.cu -o hypi -
I${OPENMPI_ROOT}/include -L${OPENMPI_ROOT}/lib -lmpi -lgomp
```

This should be all in one line

Running MPI+CUDA on Discovery

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

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:1
#SBATCH --time=00:00:59
#SBATCH --output=hypi.out
#SBATCH -A anakano_429
srun -n 2 ./hypi
```

srun is a Slurm version of mpirun

- **Output**

```
myid = 1: partial pi = 1.287001
myid = 0: partial pi = 1.854596
PI = 3.141597
```

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);
  ...
}
```

https://aiichironakano.github.io/src/cuda/hypi_setdevice.cu

Variation: Using 2 GPUs per Node (2)

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

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:2
#SBATCH --time=00:00:59
#SBATCH --output=hypi_setdevice.out
#SBATCH -A anakano_429
srun -n 4 ./hypi_setdevice
```

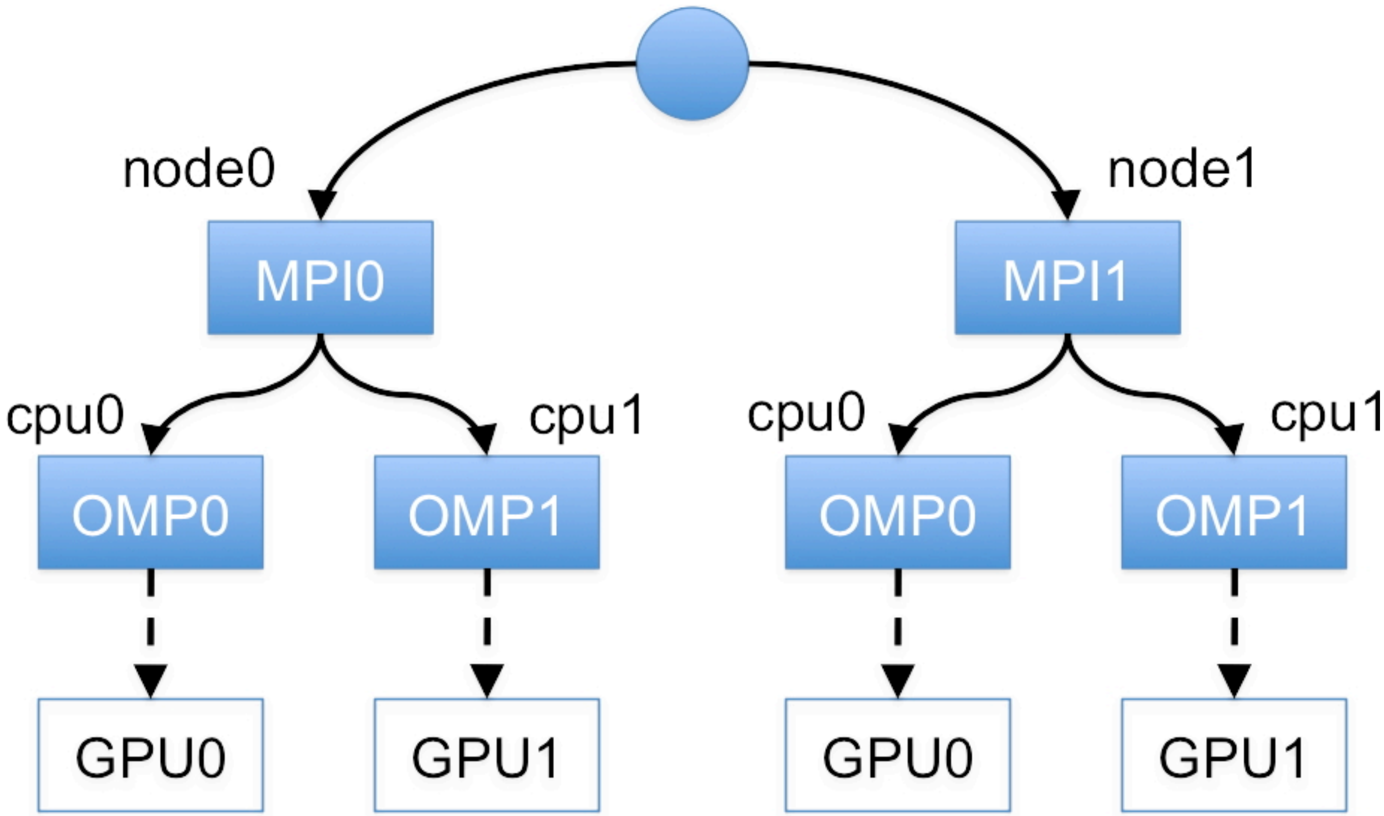
Slurm way of mpirun

- **Output**

```
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**



https://aiichironakano.github.io/cs596/src/cuda/hypi_setdevice.cu

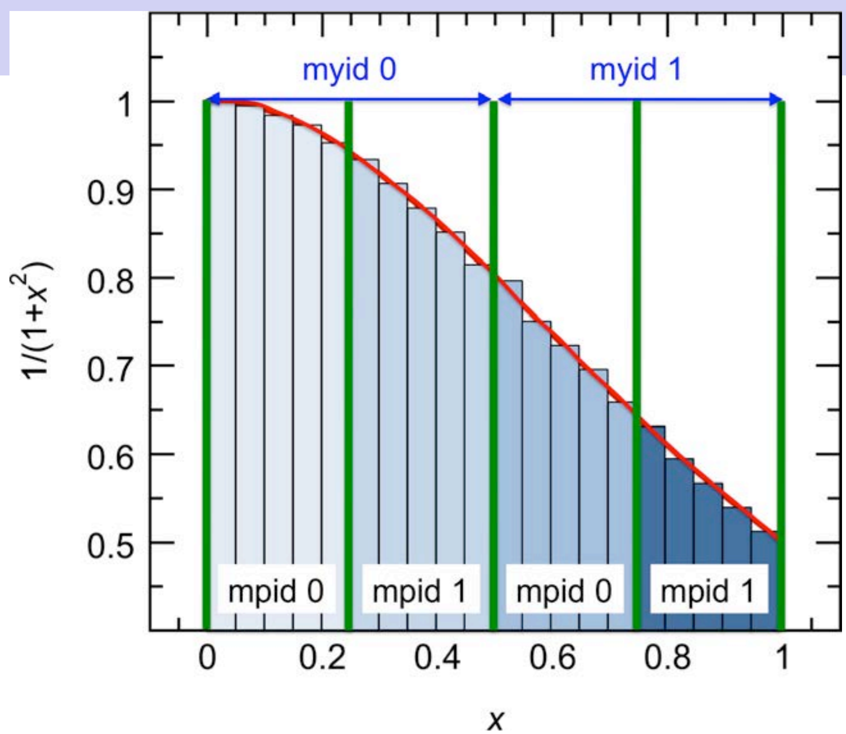
MPI+OpenMP Spatial Decompositions

```
#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)
{
  int mpid = omp_get_thread_num();
  offset = (NUM_DEVICE*myid+mpid)*step*nbin;  // Quadrature-point offset
  cudaSetDevice(mpid%2);
  ...
}
```

$$offset = \left(\overbrace{\overbrace{\overbrace{NUM\_DEVICE}^{\text{omp threads/rank}} \times \overbrace{myid}^{\text{my rank}}}^{\text{how many threads before this rank}} + \overbrace{mpid}^{\text{my thread ID in this rank}}}^{\text{how many threads before me}} \right) \times \overbrace{\overbrace{nbin}^{\text{per thread}} \times \overbrace{step}^{\text{bin width}}}^{\text{integration rage per thread}}$$

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

Hierarchical MPI+OpenMP spatial decomposition



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

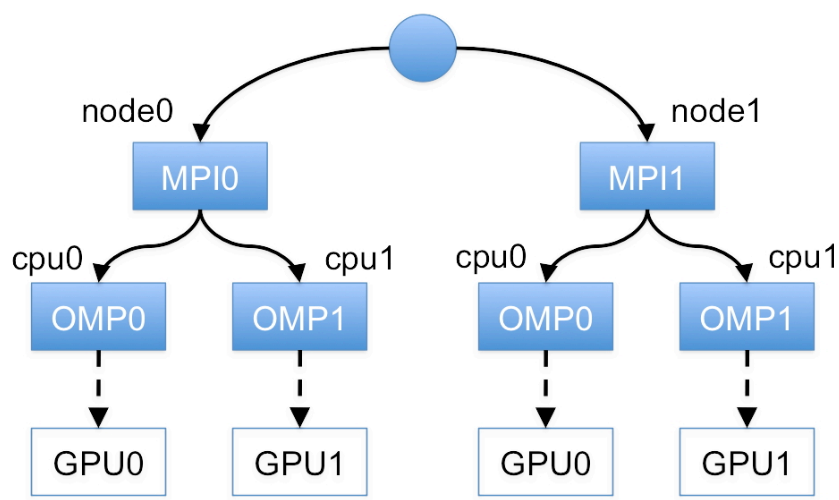
- **To report which of the two GPUs has been used for the run, insert the following lines within the OpenMP parallel block:**

```
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

- **Set an environment (add the following lines in your `.bashrc`)**

```
module purge
module load usc
module load cuda/10.1.243
```

- **Compilation**

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

```
nvcc -Xcompiler -fopenmp pi3.cu -o pi3 -
I${OPENMPI_ROOT}/include -L${OPENMPI_ROOT}/lib -lmpi -
lgomp
```

Running MPI+OpenMP+CUDA

- **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 anakano_429
srun --cpu-bind=none -n 2 ./pi3
```

- **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
```