Программирование для систем с несколькими GPU

Романенко А.А.
arom@ccfit.nsu.ru

Новосибирский государственный университет

GPU и поток исполнения

- Поток ассоциирован с одним GPU
- Выбор GPU или явно (cudaSetDevice()) или неявно - по-умолчанию.
- По умолчанию выбирается GPU с номером «0»
- Если в потоке выполнена какая-либо операций над GPU, то попытка сменить GPU на другой приведет к ошибке.

Многопоточное программирование

- POSIX Threads
- OpenMP
- пр.

OpenMP

```
#pragma omp parallel sections
{
#pragma omp section
 {
  cudaSetDevice(0);
  ...
 }
#pragma omp section
 {
  cudaSetDevice(1);
  ...
 }
}
```

OpenMP

```
int nElem = 1024;
cudaGetDeviceCount(&nGPUs);
if(nGPUs >= 1){
     omp_set_num_threads(nGPUs);

#pragma omp parallel
     {
          unsigned int cpu_thread_id = omp_get_thread_num();
          unsigned int num_cpu_threads = omp_get_num_threads();
          cudaSetDevice(cpu_thread_id % nGPUSs); //set device

          dim3 BS(128);
          dim3 GS(nElem / (gpu_threads.x * num_cpu_threads));

          // memory allocation and initialization
          int startIdx = cpu_thread_id * nElem / num_cpu_threads;
          int threadNum = nElem / num_cpu_threads;
          kernelAddConstant<<<GS, BS>>>(pData, startIdx, threadNum);
          // memory copying
     }
```

OpenMP. Сборка программ

- gcc 4.3
- Command line
  - `$ nvcc -Xcompiler \-fopenmp -Xlinker \-lgomp cudaOpenMP.cu`
- Makefile
  - ```
    EXECUTABLE := cudaOpenMP
    CUFILES := cudaOpenMP.cu
    CUDACCFLAGS := -Xcompiler -openmp
    LIB := -Xlinker -lgomp
    include ../../common/common.mk
    ```
    ```

CUDA Utillity Library

```
static CUT_THREADPROC solverThread(SomeType *plan){
    // Init GPU
    cutilSafeCall( cudaSetDevice(plan->device) );
    // start kernel
    SomeKernel<<<GS, BS>>>(some parameters);
    cudaThreadSynchronize();

    cudaThreadExit();
    CUT_THREADEND;
}
```

- Макросы используются для переносимости программы с Unix на Windows и обратно.

CUDA Utillity Library

```
SomeType solverOpt[MAX_GPU_COUNT];
CUTThread threadID[MAX_GPU_COUNT];

for(i = 0; i < GPU_N; i++){
   solverOpt[i].device = i; …
}

//Start CPU thread for each GPU
for(gpuIndex = 0; gpuIndex < GPU_N; gpuIndex++){
    threadID[gpuIndex] =
          cutStartThread((CUT_THREADROUTINE)solverThread,
                         &SolverOpt[gpuIndex]);
}


//waiting for GPU results
cutWaitForThreads(threadID, GPU_N);
```