Проект комиссии Президента
по модернизации и технологическому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров
в области суперкомпьютерных технологий и
специализированного программного обеспечения»

УТВЕРЖДАЮ
Председатель экспертного совета
системы НОЦ СКТ, член-корр. РАН
В.В. Воеводин
_____
"_____"_____201__г.

Конспект лекций дисциплины

«Параллельное программирование для многопроцессорных систем
с общей и распределенной памятью»

«**010400.62 – Прикладная математика и информатика**»

Разработчики: Лаева В.И., Трунов А.А.
Рецензент: проф. Вшивков В.А.

Москва

CUDA. Лекция № 1

Архитектура графических процессорных устройств (ГПУ)

ГПУ представляет собой набор слабо связанных мультипроцессоров (несколько десятков). В свою очередь каждый мультипроцессор состоит из нескольких потоковых ядер (~ 10), регистров (~ 10 кб) и разделяемой памяти (~ 10 кб). Разделяемая память аналогична кэш-памяти центральных процессорных устройств (ЦПУ). При этом мультипроцессоры не синхронизированы между собой непосредственно.

Каждое потоковое ядро в пределах мультипроцессора выполняет одну и ту же инструкцию (SIMD-архитектура).

Чаще всего ГПУ имеет собственную оперативную память (~ 10 Гб) с большим времени доступа.

Ограничения, присущие ГПУ:

- ориентация на задачи с мелкозернистым параллелизмом;
- для получения значительного ускорения требуется глубокое знание архитектуры ГПУ и относительно низкоуровневое программирование;
- реальная производительность ограничена скоростью передачи данных между CPU и GPU.

В настоящее время разработано значительное количество различных технологий, позволяющих проводить вычисления на ГПУ

- CUDA C (nVidia) - расширение языка C + библиотеки организации вычислений + стандартные библиотеки cuBLAS, cuFFT и др.;
- CUDA Fortran (PGI) - расширение языка Fortran 2003;
- OpenCL – открытый стандарт, поддержанный Apple, AMD, ARM, IBM, Intel, Motorola, nVidia и др. производителями. Представляет собой расширение языка C и библиотеки организации вычислений;
- Accelerator (PGI) - директивы компилятору (аналог OpenMP). Имеется поддержка C99 и Fortran95/2003, используется неявная модель программирования;
- Ct (Intel) - библиотека (framework) для C++. Ориентация на параллельность по данным, технология основана на RapidMind.

Решаемые задачи на ГПУ:

- Вычислительная гидродинамика / электродинамика / теплопередача / астрофизика / молекулярная динамика …
  - сеточные методы
  - методы частиц
  - гибридные методы (частицы-в-ячейках)
- Методы Монте-Карло
- Клеточные автоматы

- Линейная алгебра и др.

Рассмотрим типовой алгоритм взаимодействия с ГПУ:
- подготовка данных в памяти ЦПУ;
- пересылка данных в память ГПУ;
- вычисления на ГПУ;
- пересылка результатов в память ЦПУ;
- обработка результатов на ЦПУ.

В настоящем курсе лекций будет рассмотрена технология CUDA, как получившая значительное распространение и предоставляющая достаточно высокий уровень параллельного программирования относительно многих существующих средств.

CUDA является акронимом от Compute Unified Device Architecture, что можно перевести как «унифицированная архитектура вычислительного устройства».

Особенности CUDA^
- поддержка языков C/C++/Fortran/OpenCL/DirectCompute/Java/Python;
- первоначальная ориентация на ГПУ, однако есть реализации для ЦПУ (например, от PGI);
- библиотеки: cuBLAS, cuFFT, Video, PhysX, и др.;
- библиотека времени выполнения (run-time);
- драйвер ГПУ.

Модель параллельной программы CUDA

Иерархия потоков. Блоки

Все параллельные потоки (нити, треды, threads) группируются в блоки (blocks). Все потоки в пределах блока гарантированно выполняются на одном мультипроцессоре. На одном мультипроцессоре могут выполняться несколько блоков одновременно. Также на одном мультипроцессоре могут выполняться несколько блоков последовательно.

Мультипроцессор выполняет потоки в пределах блока группами (варпами, ворпами, warps).

Потоки внутри блока имеют многомерную индексацию: 1D / 2D / 3D. Поток в пределах блока нумеруется при помощи структуры threadIdx, состоящей из полей threadIdx.x, threadIdx.y, threadIdx.z. Некоторые из полей могут быть равны 1, что позволяет говорить об 1D / 2D / 3D индексации. Размерности блока потоков определяется структурой blockDim, состоящей из полей blockDim.x, blockDim.y, blockDim.z.

Можно перейти от многомерной индексации потоков (в пределах блока) к линейной, используя простые формулы:

n_{2D} = blockDim.x * threadIdx.y + threadIdx.x

n_{3D} = blockDim.x * blockDim.y * threadIdx.z + blockDim.x * threadIdx.y + threadIdx.x

Иерархия потоков. Грид

Блоки потоков группируются в грид (grid) - множество потоков, решающих общую задачу (выполняющих одно ядро, kernel). Блоки внутри грида имеют многомерную индексацию: 1D / 2D / 3D. Блок в пределах грида нумеруется при помощи структуры blockIdx, состоящей из полей blockIdx.x, blockIdx.y, blockIdx.z. Некоторые из полей индекса блока могут быть равны 1, что позволяет говорить об 1D / 2D / 3D индексации.

Расширения CUDA для языков C/C++/Fortran

ГПУ программируются на языках высокого уровня, основанных на традиционных последовательных языках. В языки C/C++/Fortran вводятся новые сущности, позволяющие учитывать архитектуру (неполный список):

- спецификаторы типа памяти объекта;
- спецификаторы типа устройства, на котором выполняется функция;
- встроенные векторные типы данных;
- встроенные переменные;
- функции синхронизации потоков;
- функции синхронизации памяти;
- доп. математические функции;
- функции для работы с текстурами;
- функции для работы с поверхностями;
- атомарные операции (реализуются как функции);
- синтаксис вызова ЦПУ функции, выполняющейся на ГПУ
- библиотека времени выполнения (работа с памятью ГПУ, получение информации о ГПУ, управление ГПУ).

Запуск вычислений на ГПУ

```
/* вызов ядра - код выполняется на ЦПУ */
dim3 dimBlock(thread_block_size);
dim3 dimGrid(N / dimBlock.x);
matvecmul_kernel <<<dimGrid, dimBlock>>> (A_d, x_d, y_d, N);


/* определение ядра - код выполняется на ГПУ */
__global__ void matvecmul_kernel(...)
{
...
}

/* Шаблон запуска ядра */
kernel <<<dimGrid, dimBlock, [Nshared, Stream]>>> (...);
```

dim3 dimGrid - структура, определяющая размерность и размер грида
dim3 dimBlock - структура, определяющая размерность и размер каждого блока
size_t Nshared [= 0] - определяет кол-во байт разделяемой (shared) памяти, динамически выделяемой на блок (в добавок к статически выделяемой памяти)
cudaStream_t Stream [= 0] - определяет ассоциированный с ядром стрим (stream)

Компиляция и запуск CUDA-программ

Компиляция программ осуществляется при помощи nvcc - драйвера компилятора. nvcc анализирует исходные тексты программ и разделяет код, выполняемый на ЦПУ и ГПУ. Код для ЦПУ модифицируется для компиляции с использованием традиционных компиляторов (<<< ... >>> заменяется на функцию времени выполнения CUDA). Код для ЦПУ может быть скомпилирован отдельно пользователем или nvcc может вызвать соответствующий компилятор, имеющийся в системе. Код для ГПУ компилируется в бинарную форму (cubin) или в инструкции ассемблера (PTX).

Пример компиляции:

nvcc --ptxas-options=-v -code=sm_21 file1.cu file2.c -o prog

Пример запуска в ОС GNU\Linux:

./prog

Пример CUDA-программы: код ЦПУ

```
#define BLOCK_SIZE 256
int main(int argc, char** argv)
{
float *d_A, *d_B, *d_C;
int N = BLOCK_SIZE*100;
size_t size = N * sizeof(float);
float   *h_A   =   (float*)malloc(size);   float   *h_B   =
(float*)malloc(size);
float *h_C = (float*)malloc(size);
Init(h_A, N); Init(h_B, N);
cudaMalloc((void**)&d_A,    size);    cudaMalloc((void**)&d_B,
size);
cudaMalloc((void**)&d_C, size);
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

int threadsPerBlock = BLOCK_SIZE; int blocksPerGrid = N /
threadsPerBlock;
VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    ...
```

```
}
```

Пример CUDA-программы: код ГПУ

```
__global__ void VecAdd(const float *A, const float *B, float
*C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    C[i] = A[i] + B[i];
}
```

Спецификаторы функций

Спецификатор	Выполняется на	Вызывается с
__device__	ГПУ	ГПУ
__global__	ГПУ	ЦПУ
__host__	ЦПУ	ЦПУ

__device__ и __host__ могут быть использованы совместно

Особенности функции-ядра (kernel):

- определяется спецификатором __global__;
- не возвращает значения (void);
- не поддерживает рекурсию;
- не поддерживает статических переменных (static);
- не поддерживает переменное кол-во аргументов;
- вызывается только с ЦПУ;
- при вызове требует специального синтаксиса <<<...>>>;
- вызывается асинхронно.

Особенности функции, вызываемой с ГПУ

- определяется спецификатором __device__;
- не поддерживает рекурсию;
- не поддерживает статических переменных (static);
- не поддерживает переменное кол-во аргументов;
- не поддерживает взятие указателя на функцию.