Программное обеспечение высокопроизводительных вычислительных систем

ЛАБОРАТОРНАЯ РАБОТА №4 (для А-10)

«Программирование графических процессоров средствами NVIDIA CUDA. Размещение данных в глобальной, разделяемой и константной памяти, время доступа к данным»

Автор: Филатов А.В.

Предыдущая лабораторная работа была посвящена знакомству с основами программирования графических ускорителей *NVIDIA* с технологией *CUDA*. Данная лабораторная работа посвящена знакомству с тремя видами памяти графического ускорителя – глобальной, разделяемой и константной.

Глобальная память самая ёмкая из всех, она доступна всем потоковым мультипроцессорам (*SMP*), т.е. всем потокам задачи, выполняющимся на потоковых процессорах (*SP*) – ядрах мультипроцессоров. При этом глобальная память является и самой медленной. В отличие от глобальной (назовём её *Global MEM*) памяти, разделяемая (назовём её *Shared MEM*) память разбита на модули, доступные только потокам одного блока задачи. Это происходит потому, что каждый блок потоков выполняется на одном *SMP и* доступ к встроенному модулю *Shared MEM* имеют только его потоки, которые выполняются на ядрах SP данного мультипроцессора. *Shared MEM* достаточно быстрая, но небольшая память. Типичная её ёмкость составляет всего 48-64 Кбайт. Главной особенностью константной (назовём её *Constant MEM*) памяти является то, что её содержимое не может быть изменено потоками задачи. Она тоже разделена на модули, но они хранят идентичное, доступное потокам всех блоков содержимое.

Схема доступа к рассматриваемой нами памяти графического ускорителя показана на рис. 1. Здесь собственные процессоры и память основного модуля компьютера обозначены как *host*, а графический ускоритель как *device*. Чтобы загрузить данные в *Global MEM* или выгрузить из неё результат, используется знакомая нам функция *cudaMemcpy*. Чтобы загрузить данные в *Constant MEM*, используется функция *cudaMemcpyToSymbol*. Прямого доступа с *host*-а в *Shared MEM* нет, поэтому данные сначала загружаются в глобальную память, а потом переносятся в разделяемую простым копированием.

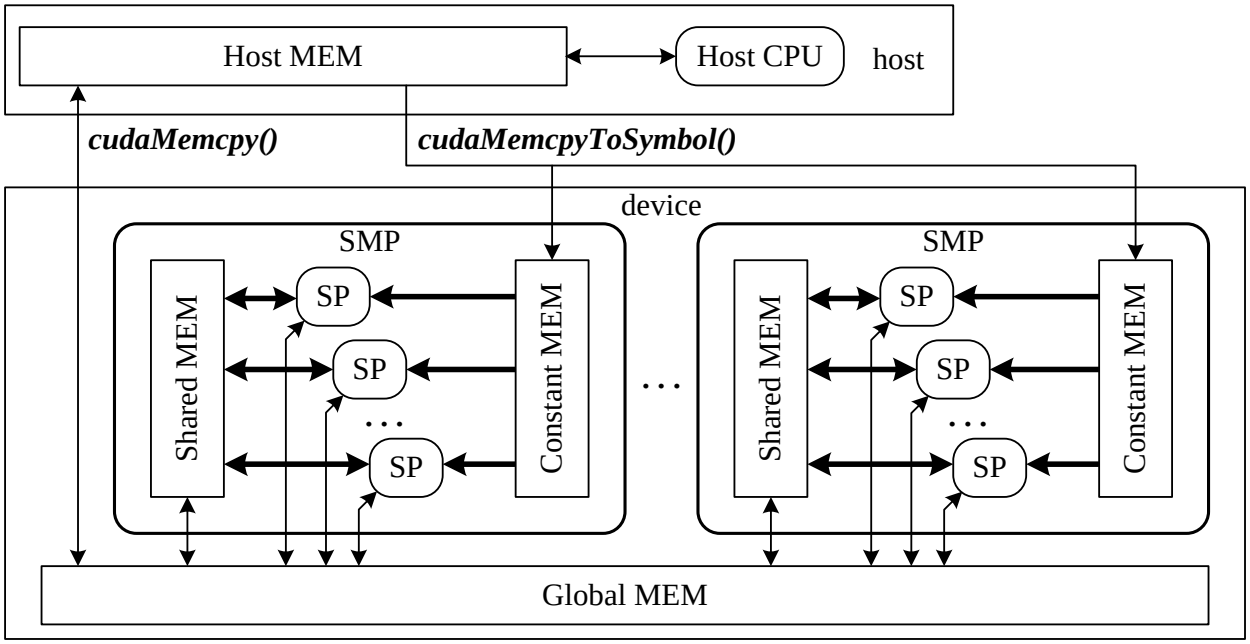

Рис. 1. Схема доступа к памяти

Ниже, приведён фрагмент ядра *CUDA* – программы с простым поэлементным копированием потоком *threadIdx.x* своих *s*-элементов данных из глобальной памяти (массив *A*) в распределённую (массив *as*) и наоборот.

```
__global__ void fun_kernel( float * a, int s)
{
        __shared__ float as [NS];  //где NS*sizeof(float) < 48 Кб

for ( int k = s*threadIdx.x; k < (s+1)*threadIdx.x; k++ )
   as [k]=A[k];

        __syncthreads();
        // Работа с данными в разделяемой памяти
        __syncthreads();

for ( int k = s*threadIdx.x; k < (s+1)*threadIdx.x; k++ )
   A[k]=as[k];
}
```

Важно отметить две детали. Чтобы массив *as* был создан в разделяемой памяти, его объявление необходимо снабдить префиксом *__shared__*. Чтобы не

было рассогласований между потоками при работе с разделяемой памятью, часто бывает необходимо производить синхронизацию потоков функцией *__syncthreads*.

Выполняя данную лабораторную работу, будем считать, что нам доступно только 16 Кб в каждом модуле *Shared MEM*, поэтому, работая с большими массивами, необходимо будет их делить на части, чтобы они вместились в указанную ёмкость. При этом, чтобы выборка из памяти была наиболее эффективна, необходимо соблюдать правила блочного запроса – *coalescing* (см. лекционный материал).

Чтобы загрузить данные в константную память, надо сначала напрямую (глобально, вне функций) в программе объявить массив, например *ac,* с прафексом *__constant__* , а затем с помощью функции *cudaMemcpyToSymbol* скопировать данные из памяти *host*-а (*Host MEM*) в *Constant MEM*.

```
__constant__ float ac[N];
…
__global__ void fun_kernel() // Функция-ядро
…
int main()
{
   float Ahost[N]; //массив данных на host-е
   …
   cudaMemcpyToSymbol(ac, Ahost, sizeof(float)*N);
   …
  <вызов ядра fun_kernel() >
   …
}
```

Задание на лабораторную работу

1. Написать на языке *C* и отладить исходную программу для *host*-а, реализующую обработку данных согласно варианту из таблицы 1.
2. Задать число *N* = 512. Для записи результатов работы подготовить таблицу протокола по шаблону (см. таблицу 2).

3. Переписать исходную программу в программу для *CUDA* так, чтобы в процессе вычислений на *device* все массивы хранились в глобальной памяти.
4. Выполнить вычисления программы п.3 одним, двумя и четырьмя блоками, по 1, 4, 32, 128 потоков в каждом. Замерить времена выполнения и занести их в таблицу протокола.
5. Переписать исходную программу для *CUDA* так, чтобы в процессе вычислений на *device* все массивы хранились в разделяемой (*shared*) памяти. (Не забывайте про *coalescing*).
6. Выполнить вычисления программы п.5 одним, двумя и четырьмя блоками, по 1, 4, 32, 128 потоков в каждом. Замерить времена выполнения и занести их в таблицу протокола.
7. Переписать исходную программу для *CUDA* так, чтобы в процессе вычислений на *device* все массивы кроме X хранились в константной (*constant*) памяти, а сам X в разделяемой (*shared*) памяти.
8. Выполнить вычисления программы п.7 одним, двумя и четырьмя блоками, по 1, 4, 32, 128 потоков в каждом. Замерить времена выполнения и занести их в таблицу протокола.
9. Задать число $N = 65536$. Для записи результатов работы подготовить ещё одну таблицу протокола по шаблону (см. таблицу 2).
10. Переписать программы п. 5 и 7 таким образом, чтобы, считая доступной для хранения массивов ёмкость разделяемой (*shared*) памяти по 16 Кб на блок, можно было разместить и эффективно использовать данные из массивов. Для этого надо рассчитать, сколько элементов массивов одновременно может храниться в памяти и организовать загрузку и выгрузку данных и результатов между глобальной памятью и разделяемой памятью.
11. Выполнить программы п. 3 и 10 (при $N = 65536$) одним, двумя и четырьмя блоками, по 1, 4, 32, 256 потоков в каждом. Замерить времена выполнения и занести их в таблицу протокола.
12. Составить сравнительную таблицу времён выполнения вычислений (см. таблицу 3).

Таблица 1. Варианты заданий

№ бригады	**Вариант задания**

1	#define M=**45*< номер вашей студенческой группы >**
	
	int A[N];
	float B[N],X[N]; char C[N];
	. . .
	for (i=0; i<M; i++)
	  for (j=0; j<N; j++)
	  X[j]=*(float)* A[j]*X[j]+(B[j]-X[j])/C[j];

2	#define M=**60*< номер вашей студенческой группы >**
	
	int A[N], B[N];
	double X[N], C[N];
	. . .
	for (i=0; i<N; i++)
	  for (j=0; j<M; j++)
	  X[i]=*(double)* A[i]*(X[i]+B[i])/C[i];

3	#define M=**31*< номер вашей студенческой группы >**
	
	char A[N], B[N];
	float X[N], C[N], D[N];
	. . .
	for (i=0; i<N; i++)
	  for (j=0; j<M; j++)
	  X[i]=*(float)* A[i]*(X[i]+B[i])+X[i]/(C[i]+ D[i]);

4	#define M=**< номер вашей студенческой группы >+879**
	
	int A[N], B[N], C[N];
	double X[N];
	. . .
	for (i=0; i<3*M; i++)
	  for (j=0; j<N; j++)
	  X[j]=*(double)* (A[j]*X[j]+C[j]*X[j])/B[j];

5	#define M=**35*< номер вашей студенческой группы >-12**
	
	int X[N], B[N];
	char A[N], C[N], D[N];
	. . .
	for (i=0; i<N; i++)
	  for (j=0; j<M; j++)
	  X[i]=B[i]*(X[i]+A[i]+C[i]) + X[i]*D[i]);

6	#define M=**21*< номер вашей студенческой группы >+234**
	
	char A[N], B[N], C[N];
	double X[N], D[N];
	. . .
	for (i=0; i<N; i++)
	  for (j=0; j<2*M; j++)
	  X[i]=*(double)* A[i]*X[i]*(X[i]*C[i]+B[i])/D[i];

7	#define M=**< номер вашей студенческой группы >+956**
	
	int A[N], B[N], X[N];
	float C[N]; char D[N];
	. . .
	for (i=0; i<N; i++)
	  for (j=0; j<4*M; j++)
	  X[i]=*(int)* X[i]*(*(float)* (C[i]+B[i])/(X[i]*A[i]+D[i]));

8	#define M=**87*< номер вашей студенческой группы >**
	
	int B[N], C[N];
	float X[N], A[N]; char D[N];
	. . .
	for (i=0; i<M; i++)
	  for (j=0; j<N; j++)
	  X[j]=*(float)* (A[j]*X[j]+C[j]*X[j] /(B[j]- D[j]);

Таблица 2. Шаблон таблицы протокола

Времена выполнения вычислений на *GPU* при *N*=...												
	(число блоков / число потоков в блоке)											
	(1/1)	(1/4)	(1/32)	(1/256)	(2/1)	(2/4)	(2/32)	(2/256)	(4/1)	(4/4)	(4/32)	(4/256)
Только глобальная память												
Разделяемая память												
Разделяемая и константная память												

Таблица 3. Шаблон сравнительной таблицы

		(число блоков / число потоков в блоке)											
N	Память	(1/1)	(1/4)	(1/32)	(1/256)	(2/1)	(2/4)	(2/32)	(2/256)	(4/1)	(4/4)	(4/32)	(4/256)
N=512	Glob	1	1	1	1	1	1	1	1	1	1	1	1
	Shar												
	S&C												
N = 65536	Glob	1	1	1	1	1	1	1	1	1	1	1	1
	Shar												
	S&C												