CUDA
Оптимизация программ

Романенко А.А.
arom@ccfit.nsu.ru

Новосибирский государственный университет

Сборка программы

- Компилятор — nvcc
- Исходные коды - *.cu или *.cuh
- Рекомендуется собирать с помощью **make**
  - Скопировать к себе из примеров Makefile и common.mk
  - Поправить пути для выходных файлов
- Сборка
  - **make emu=1** — в режиме эмуляции определен макрос __DEVICE_EMULATION__
  - **make dbg=1** — с отладочной информацией
  - **make** — финальной версии программы

Профилирование

- clock_t clock() - счетчик, который увеличивается с каждым тактом GPU
  - Разность значений в начале и конце ядра — количество тактов, затраченных GPU на выполнение потока. НЕ ПРОВЕДЕННОЕ в исполнении потока.
- Переменные окружения
  - CUDA_PROFILE=1
  - CUDA_PROFILE_LOG
  - CUDA_PROFILE_CONFIG

Профилирование. Файл конфигурации

- timestamp
- gridsize
- threadblocksize
- dynsmemperblock
- stasmemperblock
- regperthread
- memtransferdir
- memtransfersize
- streamid

Профилирование. Файл конфигурации

- gld_incoherent
- gld_coherent
- gld_32b / gst_32b
- gld_64b / gst_62b
- gld_128b / gst_128b
- gld_request
- gst_incoherent
- gst_coherent
- gst_request

- local_load
- local_store
- branch
- divergent_branch
- instructions
- warp_serialize
- cta_launched
  - gputime
  - cputime
  - occupancy

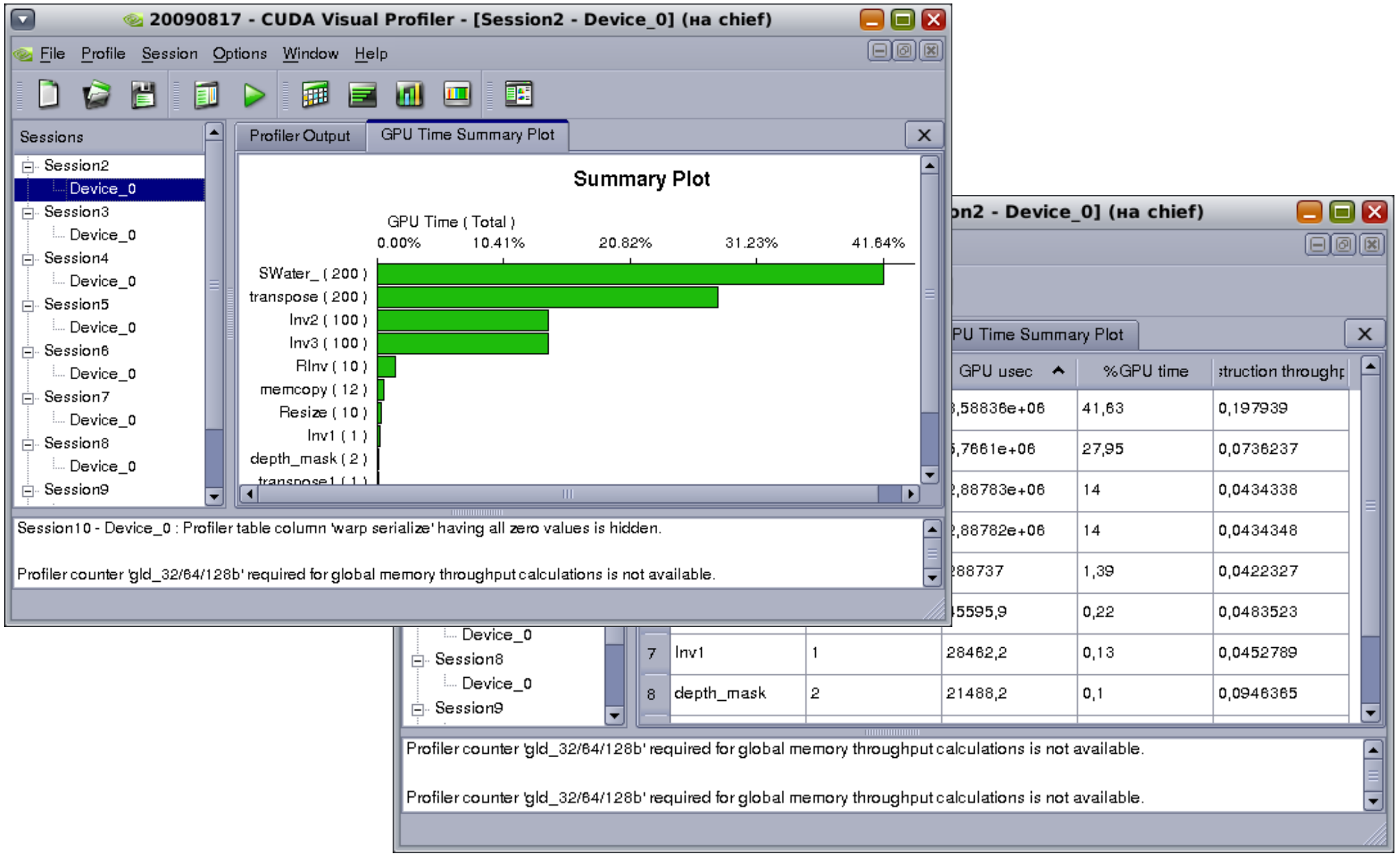
Профилирование
Вывод

```
method,gputime,cputime,occupancy
method=[ memcopy ] gputime=[ 5519.680 ]
method=[ memcopy ] gputime=[ 5516.992 ]
method=[ memcopy ] gputime=[ 5517.728 ]
method=[ memcopy ] gputime=[ 5508.288 ]
method=[ __globfunc__Z10depth_maskPfS_fii ] gputime=[ 1791.424 ] cputime=[ 14.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z10depth_maskPfS_fii ] gputime=[ 1769.920 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1598.336 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z12Invariants_XPfS_S_S_S_S_S_ii ] gputime=[ 5061.920 ] cputime=[ 3.000 ]
                                                                  occupancy=[ 0.812 ]
method=[ __globfunc__Z7SWater_PfS_S_S_S_S_fS_S_S_ii ] gputime=[ 10506.752 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.406 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1599.776 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1613.472 ] cputime=[ 1.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z9transposePfS_ii ] gputime=[ 1609.056 ] cputime=[ 2.000 ] occupancy=[ 0.812 ]
method=[ __globfunc__Z7SWater_PfS_S_S_S_S_fS_S_S_ii ] gputime=[ 10758.784 ] cputime=[ 2.000 ]
                                                                  occupancy=[ 0.406 ]
method=[ __globfunc__Z13RInvariants_YPfS_S_S_S_S_S_ii ] gputime=[ 5423.456 ] cputime=[ 3.000 ]
                                                                  occupancy=[ 0.812 ]
```

Анализ отчета о профилировании

- Значение имеет не цифры, а их приращение
- Для ядер надо стремиться чтобы стремились к нулю непоследовательное обращение к памяти (gld_incoherent, gld_coherent, gst_incoherent, gst_coherent)

Профилировщик cudaprof

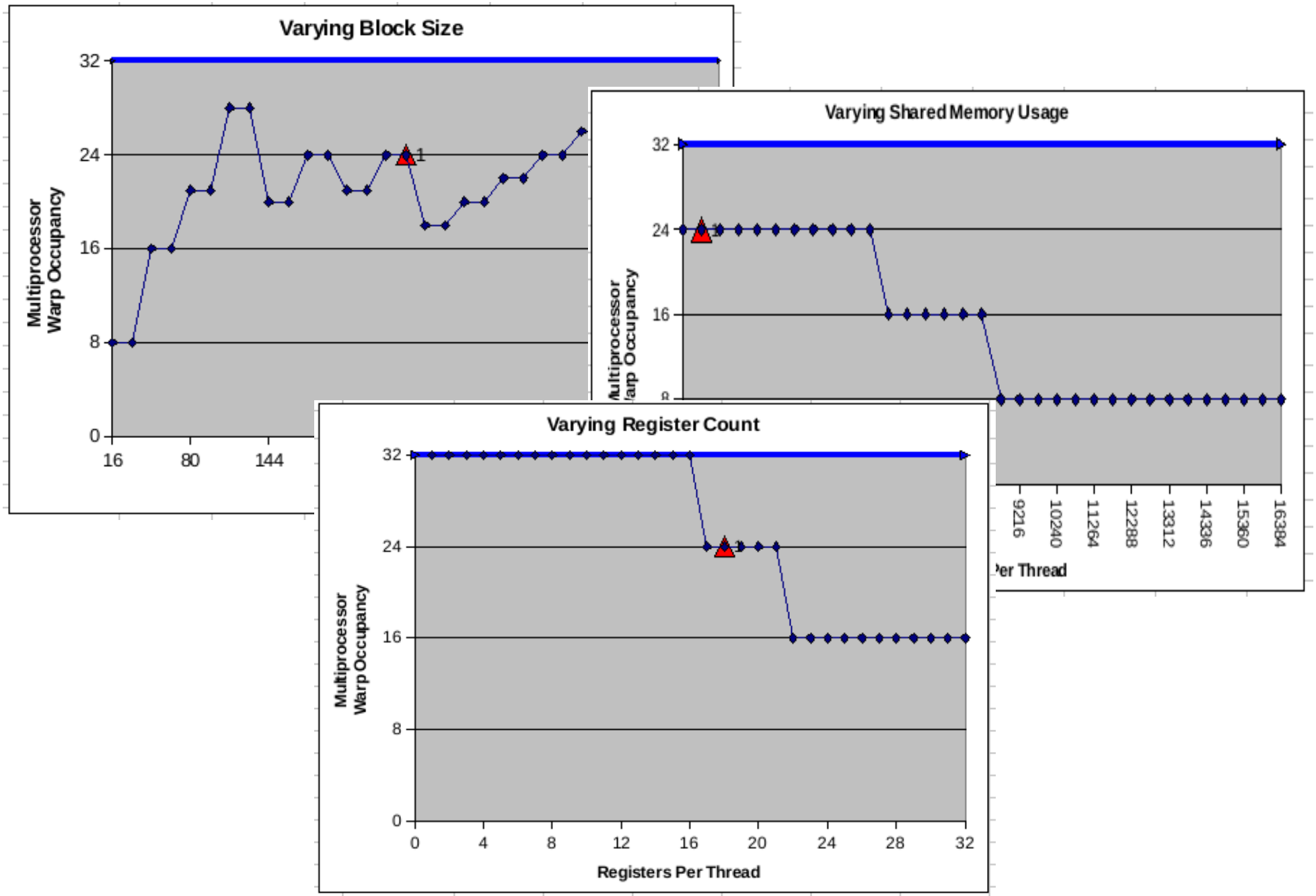


CUDA Occupancy calculator

Just follow steps 1, 2, and 3 below! (or click here for help)	
1.) Select Compute Capability (click):	**1,2**
2.) Enter your resource usage:	
Threads Per Block	256
Registers Per Thread	18
Shared Memory Per Block (bytes)	512
(Don't edit anything below this line)	
3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	**768**
Active Warps per Multiprocessor	**24**
Active Thread Blocks per Multiprocessor	**3**
Occupancy of each Multiprocessor	**75%**

Physical Limits for GPU:	
Threads / Warp	
Warps / Multiprocessor	
Threads / Multiprocessor	
Thread Blocks / Multiprocessor	
Total # of 32-bit registers / Multiprocess	
Shared Memory / Multiprocessor (bytes)	
Allocation Per Thread Block	
Warps	
Registers	
Shared Memory	512
These data are used in computing the occupancy data in blue	
Maximum Thread Blocks Per Multiprocessor	Blocks
Limited by Max Warps / Multiprocessor	4
Limited by Registers / Multiprocessor	**3**
Limited by Shared Memory / Multiprocessor	32
Thread Block Limit Per Multiprocessor highlighted	**RED**

CUDA Occupancy calculator



Оптимизация

- Обработка инструкций
  - Чтение операндов
  - Выполнение инструкции
  - Сохранение результата
- Для оптимизации
  - Использовать более быстрые инструкции
  - Минимизировать задержки на обращение к памяти
  - По максимуму использовать пропускную способность шин данных

Исполнение инструкций

- Арифметические операции
  - 4 такта для FMUL, FADD, FMAD IADD, бинарные операции, сравнение, MIN, MAX, приведение типов
  - 16 тактов для __log, 1/sqrt, IMUL (1.x), 1/(float)x
  - 32 такта для sqrt, __sin, __cos, __exp
  - 36 такта для FDIV
  - 20 тактов для __fdividef(x, y)

Условные переходы

- Если в ворпе есть две ветви исполнения (условный переход), то сначала исполняются потоки, которые проходят одну ветвь, затем потоки, которые проходят вторую.
- Минимизировать количество ветвлений. В частности внутри ворпа. Например за счет предвычислений.

Доступ к памяти

- 4 такта на обработку одной инструкции по работе с памятью (разделяемая, константная*, текстуры*).
- 400-600 тактов задержка по доступу к глобальной памяти
- Метод работы:
  - Загрузить данные из глобальной памяти в разделяемую (через текстуры)
  - Обработать данные
  - Выгрузить в глобальную

* - если нет промахов по кэшу

Доступ к памяти

- Используйте cudaMalloPitch вместо cudaMalloc, если двумерный массив
- Используйте cudaMallocArray для 2D и 3D массивов
- Используйте текстуры

Передача данных на/с устройства

- Скорость копирования на устройстве выше, чем между Хостом и Устройством
- Рекомендуется не выгружать данные, а запустить ядро с малым уровнем параллелизма, если это возможно.
- На Хосте выделять память с помощью cudaMallocHost()
- Совмещать передачу данных с вычислениями.

Оптимизация

- Количество регистров на поток и разделяемой памяти на блок
  - --ptxas-options=-v
  - CUDA Occupancy calculator
-

Back up

Отладка

- make dbg=1 emu=1
- nvcc … -deviceemu
- Макрос __DEVICE_EMULATION__
  - Можно использовать операции ввода/вывода в файл и на экран внутри __device__ и __global__ функций (printf()).
- Эмуляция а не симуляция
- Точность вычислений над операциями с числами с плавающей точкой может отличаться
- Используйте любой отладчик, который вам нравиться

Отладка

- cudaGetLastError();
- cudaGetErrorString(error);
-