Технология CUDA для высокопроизводительных вычислений на кластерах с GPU

Лихогруд Николай
n.lihogrud@gmail.com

Часть пятая

Профилирование и отладка

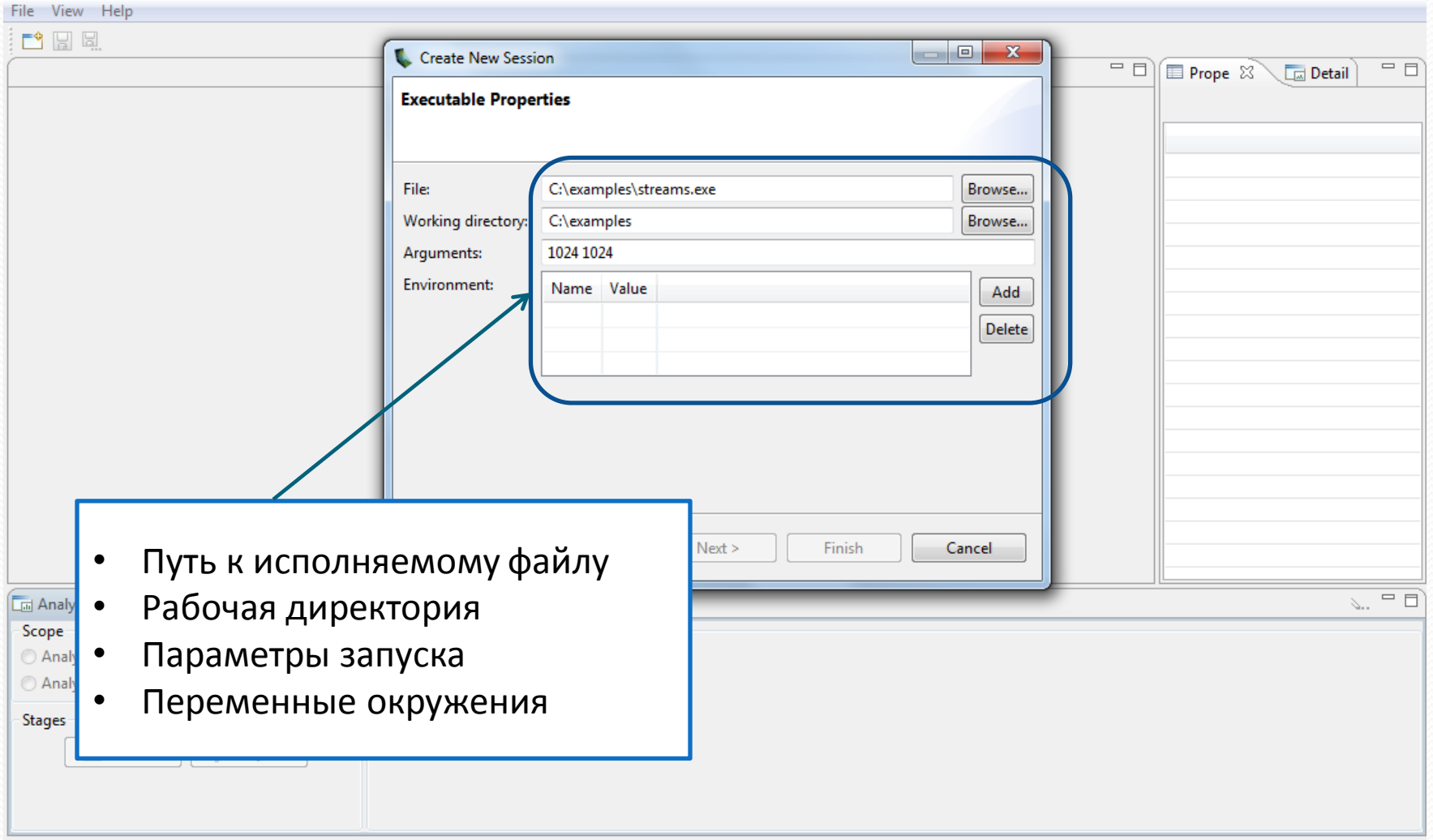
nvvp
nvprof
Удаленное профилирование
cuda-gdb

Visual Profiler

- NVidia Visual Profiler, NVVP
  - Поставляется вместе с toolkit-ом
  - Linux/Windows

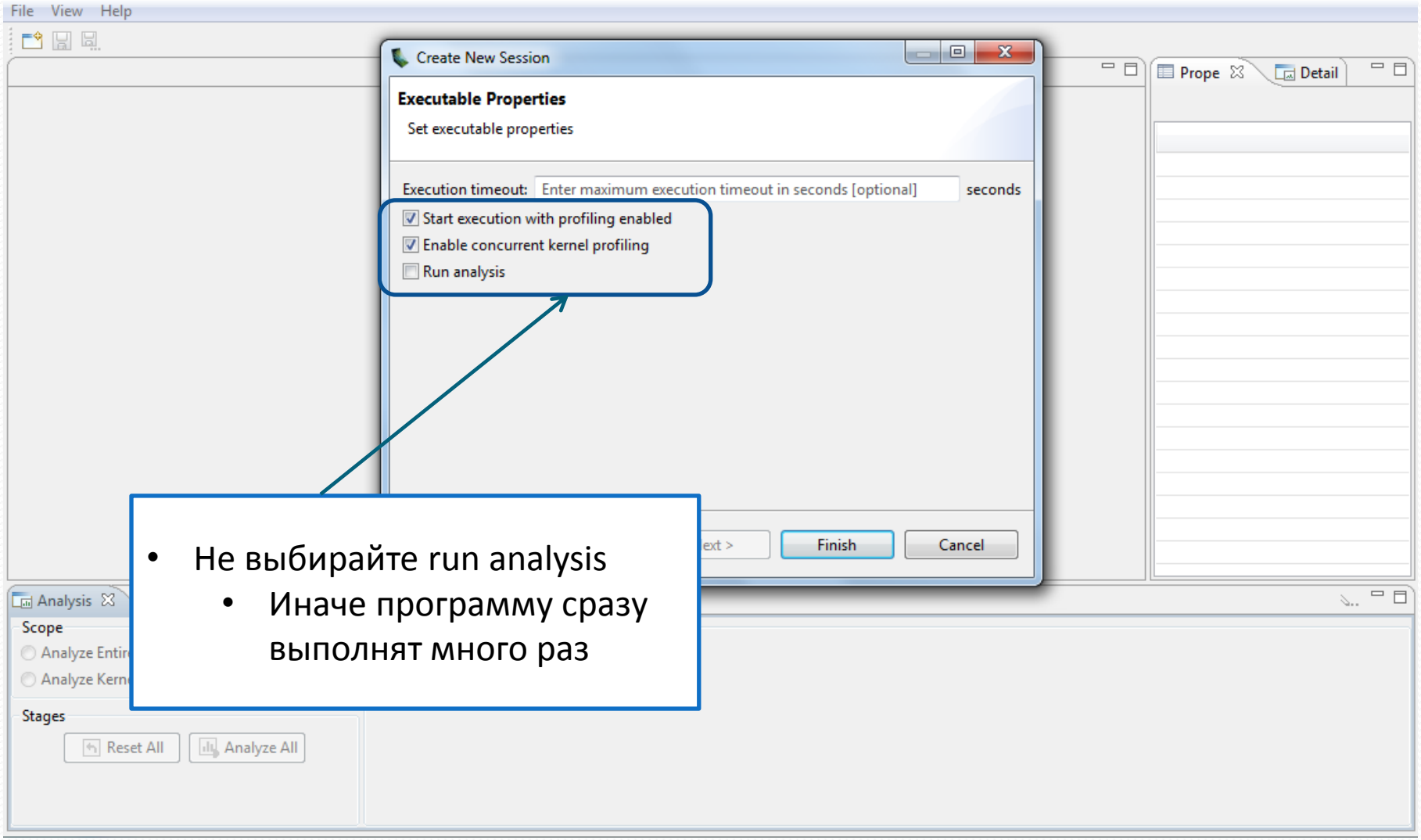
- timeline выполнения команд на GPU
- Показания счетчиков различных событий
- Расчёт метрик
- Автоматический анализ эффективности

File -> new Session



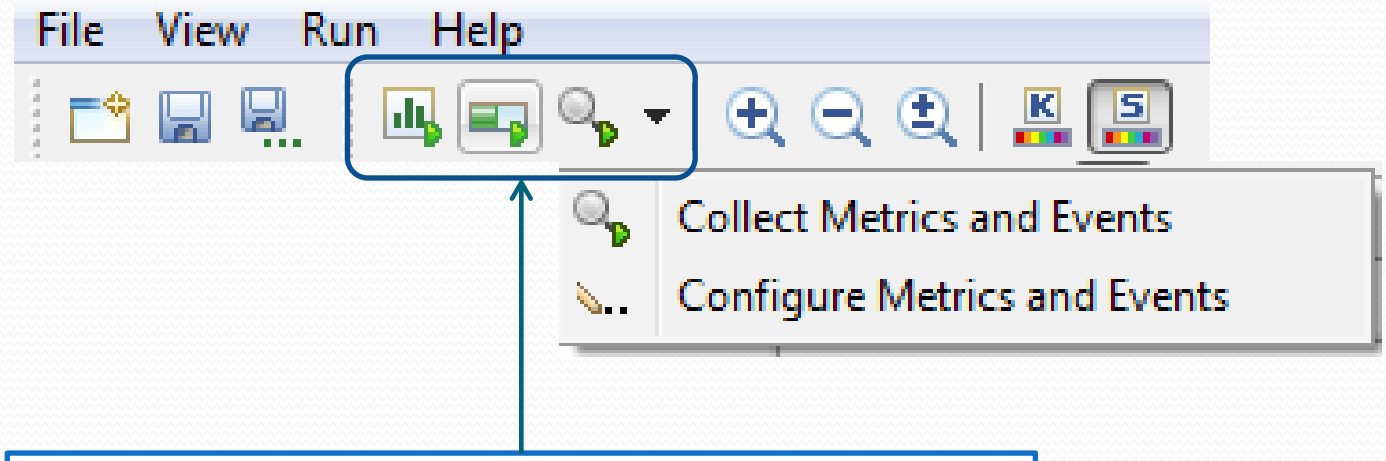
- Путь к исполняемому файлу
- Рабочая директория
- Параметры запуска
- Переменные окружения

File -> new Session -> next



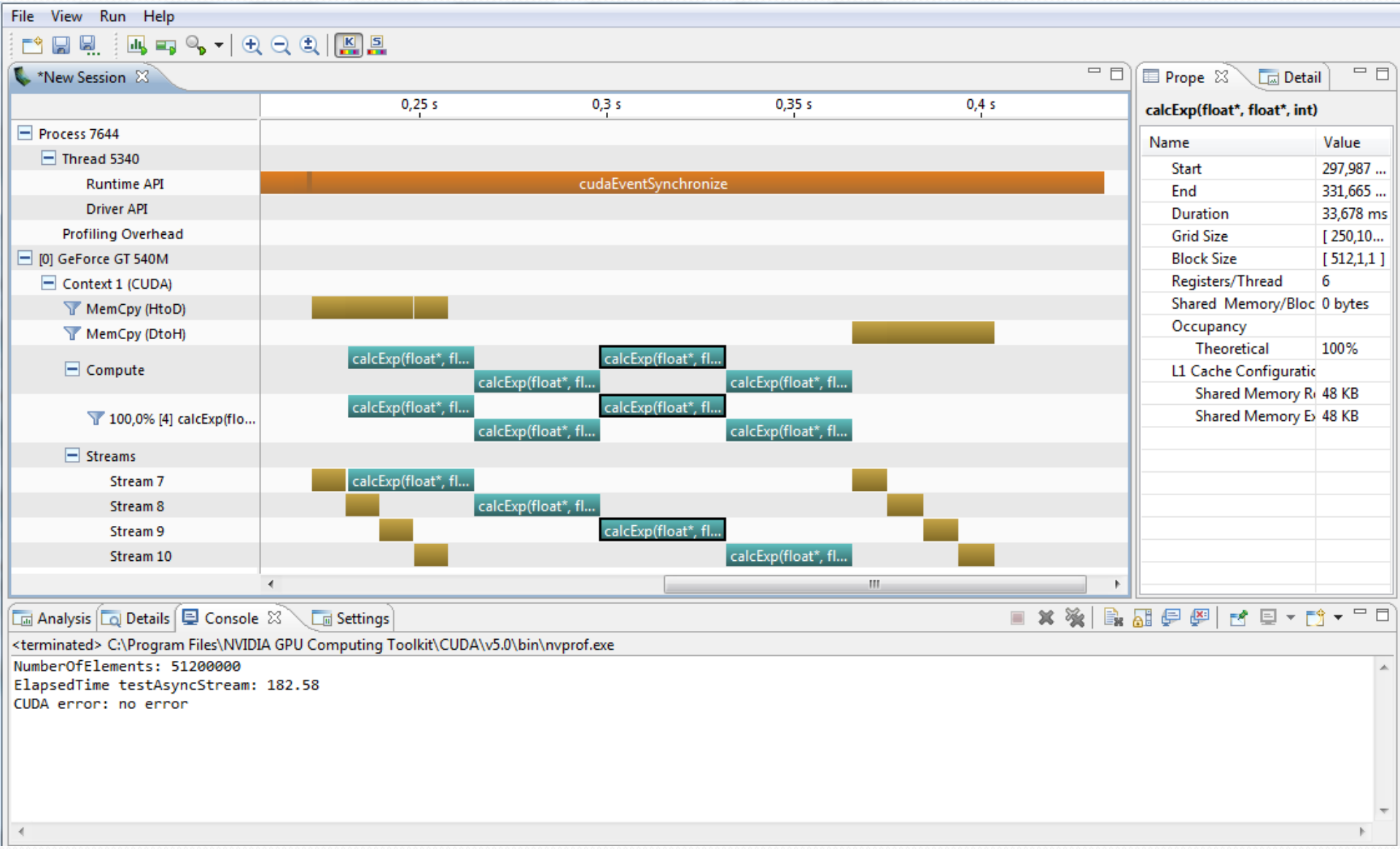
- Не выбирайте run analysis
  - Иначе программу сразу выполнят много раз

Toolbar

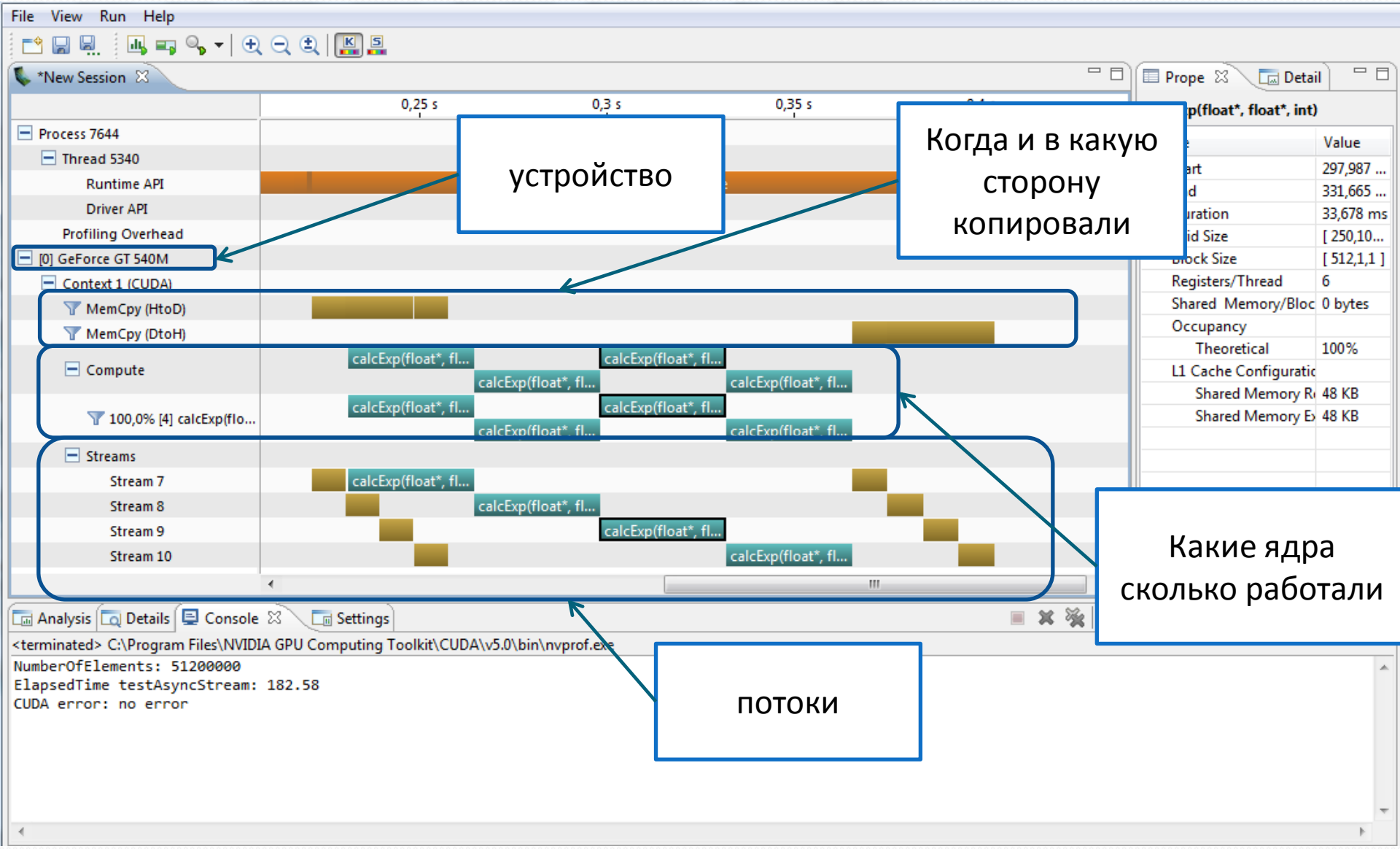


- Автоматический анализ
- Рассчитать timeline
- Собрать показания счетчиков и метрик
  - Выбрать интересующие счетчики и метрики

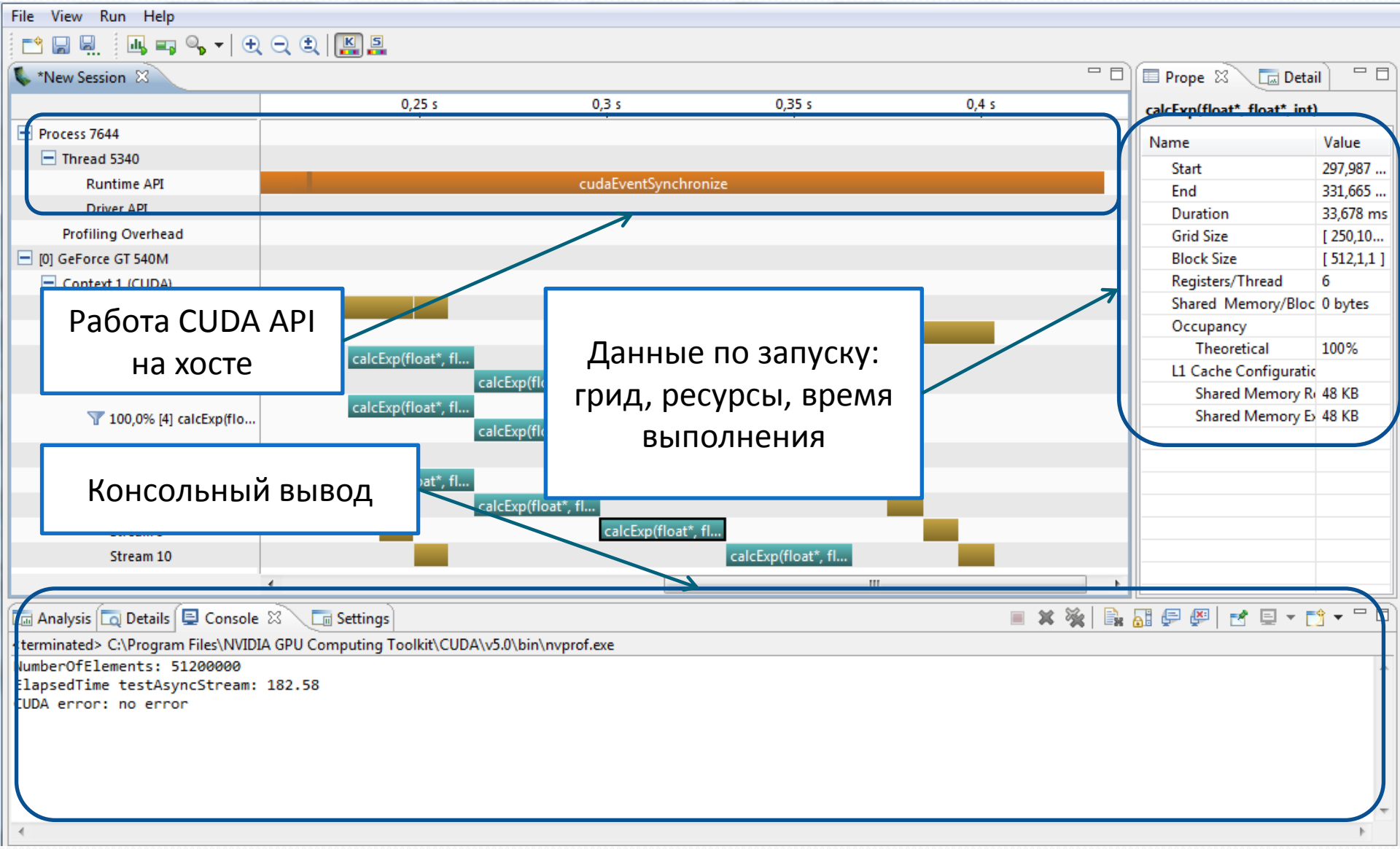
Основное окно



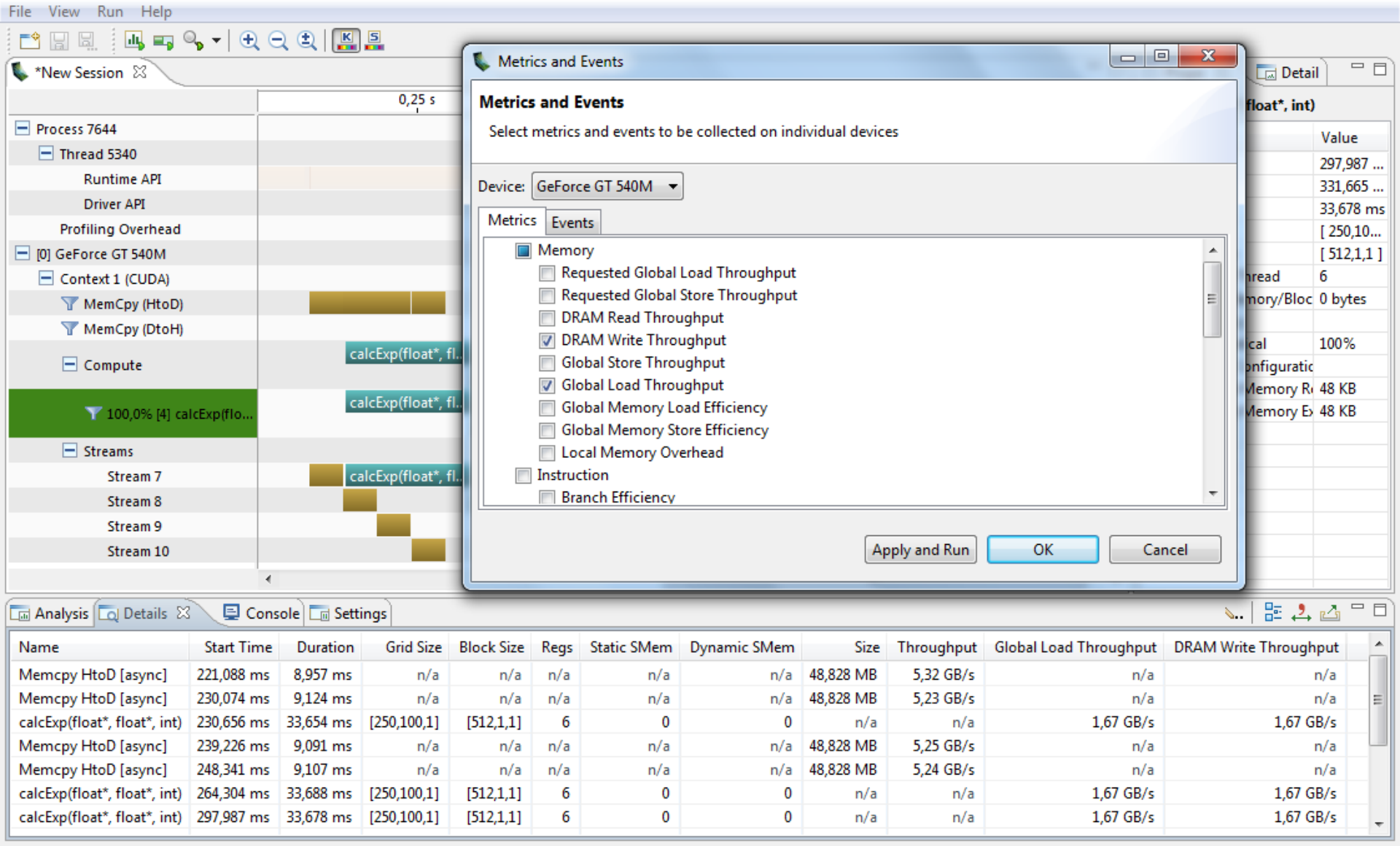
Основное окно



Основное окно

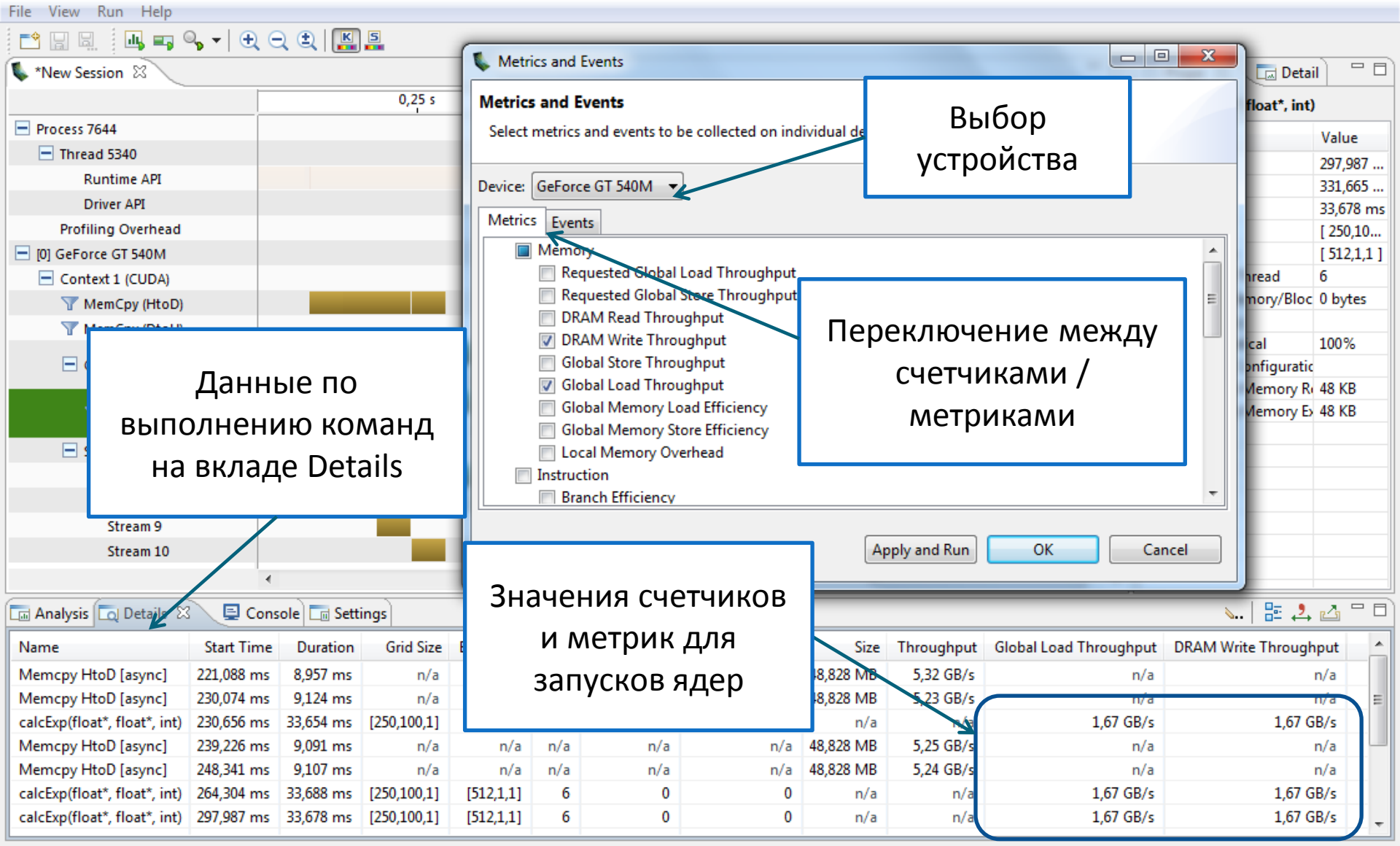


Счетчики и метрики



Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput	Global Load Throughput	DRAM Write Throughput
Memcpy HtoD [async]	221,088 ms	8,957 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,32 GB/s	n/a	n/a
Memcpy HtoD [async]	230,074 ms	9,124 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,23 GB/s	n/a	n/a
calcExp(float*, float*, int)	230,656 ms	33,654 ms	[250,100,1]	[512,1,1]	6	0	0	n/a	n/a	1,67 GB/s	1,67 GB/s
Memcpy HtoD [async]	239,226 ms	9,091 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,25 GB/s	n/a	n/a
Memcpy HtoD [async]	248,341 ms	9,107 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,24 GB/s	n/a	n/a
calcExp(float*, float*, int)	264,304 ms	33,688 ms	[250,100,1]	[512,1,1]	6	0	0	n/a	n/a	1,67 GB/s	1,67 GB/s
calcExp(float*, float*, int)	297,987 ms	33,678 ms	[250,100,1]	[512,1,1]	6	0	0	n/a	n/a	1,67 GB/s	1,67 GB/s

Счетчики и метрики



Счетчики и метрики

- Event
  - Некоторое низкоуровневое событие, например кеш-промах, банк-конфликт, бранчинг, завершение выполнения варпа и т.д.
  - В специальных регистрах-счетчиках на мультипроцессорах накапливаются количества таких событий
- Metric
  - Более высокоуровневая информация, рассчитываемая на основе счетчиков, например, % кеш промахов
- Примеры счетчиков и метрик в следующий раз

Полезные метрики

- Метрики рассчитываются на основе значений аппаратных счетчиков
- Описания метрик - http://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference
- Точные формулы расчета можно найти в CUDA_Profiler_User_Guide.pdf

Полезные метрики

- **achieved_occupancy**
- **branch_efficiency** – доля варпов, выполненных без бранчей
- **warp_execution_efficiency** – средняя доля активных нитей в варпах
- **inst_replay_overhead** – повторы инструкций (из-за сериализации) / общее число выполненных устройством инструкций
- **shared_replay_overhead, global_cache_replay_overhead, local_replay_overhead** – разделение предыдущего по причинам

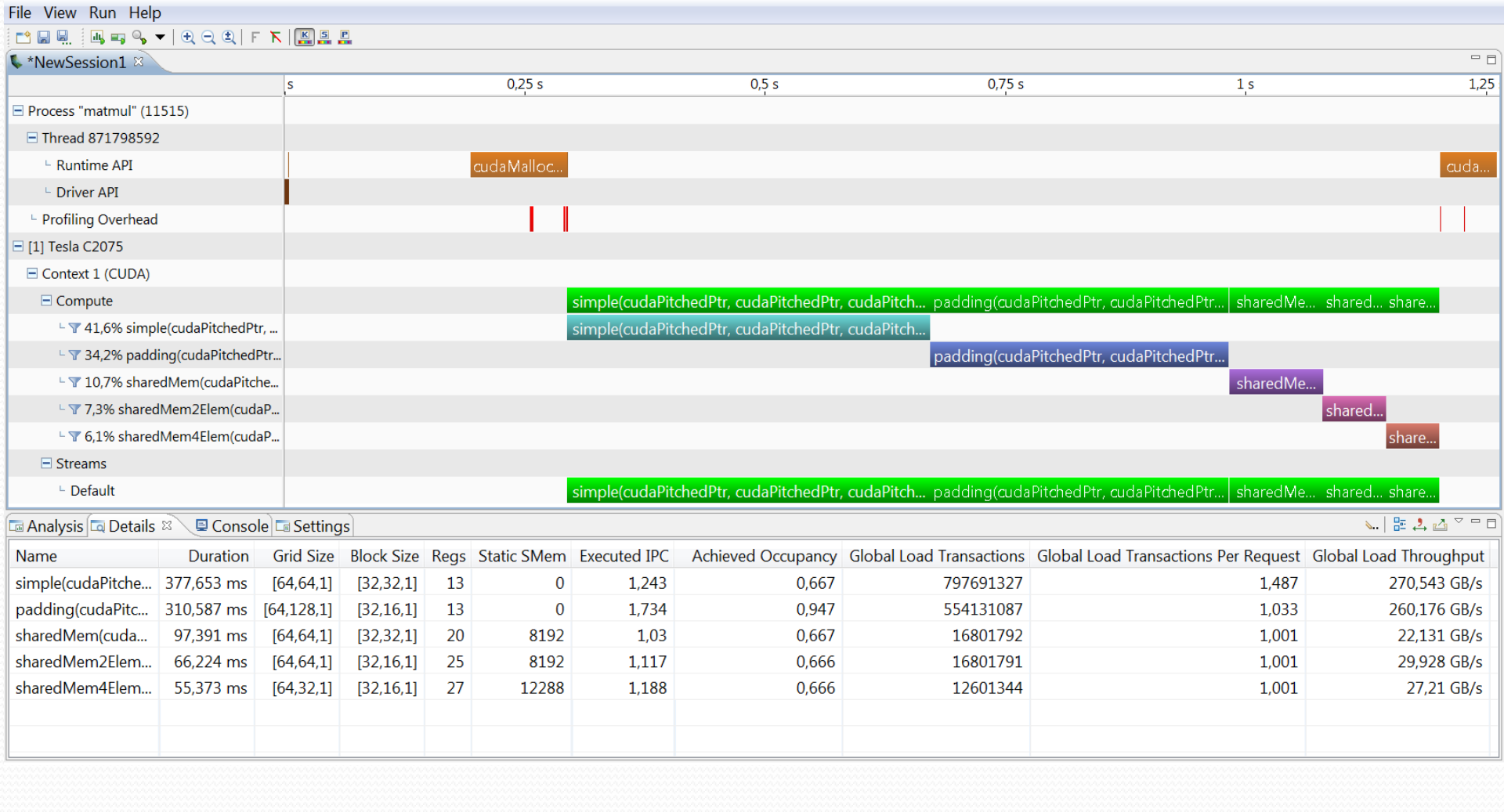
Полезные метрики

- **dram_read_throughput**– объем памяти, отданной ядру / время выполнения
- **gld_throughput** - объем глобальной памяти, отданной ядру / время выполнения
- **gld_requested_throughput** - объем глобальной памяти, запрошенный ядром / время выполнения
- **gld_efficiency** – gld_requested_throughput / gld_throughput
- **dram_write_throughput, gst_throughput, gst_requested_throughput, gst_efficiency** - аналогично для записи

Полезные метрики

- **l1_cache_global_hit_rate** – доля кеш-попаданий при обращении к глобальной памяти
- **l1_cache_local_hit_rate** – доля кеш-попаданий при обращении к локальной памяти
- **l2_l1_read_hit_rate –**доля кеш попаданий при обращении кеша l1 к кешу l2
- **local_memory_overhead** - доля локальной памяти в обменах между l1 и l2
- **IPC** – число инструкций, запускаемых за цикл работы warp scheduler

Пример: перемножение матриц



Name	Duration	Grid Size	Block Size	Regs	Static SMem	Executed IPC	Achieved Occupancy	Global Load Transactions	Global Load Transactions Per Request	Global Load Throughput
simple(cudaPitche...	377,653 ms	[64,64,1]	[32,32,1]	13	0	1,243	0,667	797691327	1,487	270,543 GB/s
padding(cudaPitc...	310,587 ms	[64,128,1]	[32,16,1]	13	0	1,734	0,947	554131087	1,033	260,176 GB/s
sharedMem(cuda...	97,391 ms	[64,64,1]	[32,32,1]	20	8192	1,03	0,667	16801792	1,001	22,131 GB/s
sharedMem2Elem...	66,224 ms	[64,64,1]	[32,16,1]	25	8192	1,117	0,666	16801791	1,001	29,928 GB/s
sharedMem4Elem...	55,373 ms	[64,32,1]	[32,16,1]	27	12288	1,188	0,666	12601344	1,001	27,21 GB/s

command-line profiler

Command-line profiler

Включается/отключается через переменную окружения COMPUTE_PROFILE:

```
$export COMPUTE_PROFILE=1
$./a.out
```

Или

```
$COMPUTE_PROFILE=1 ./a.out
```

Файл лога

- По умолчанию пишет лог профилировки в файлы cuda_profile_%d.log в текущей директории (pwd), где %d заменяется на номер контекста
  - Для каждого GPU свой лог профилировки - cuda_profile_0.log, cuda_profile_1.log и т.д.

- Можно изменить через переменную COMPUTE_PROFILE_LOG:

```
$export COMPUTE_PROFILE_LOG=example.%d.log
$COMPUTE_PROFILE=1 ./a.out
```

Поддерживаемые форматы

- Key-Value-Pairs (по-умолчанию)

```
method,gputime,cputime,occupancy
_Z6kernelPfi,1724792.500,17.270,1.000
_Z6kernelPfi,1724766.000,5.320,1.000
_Z6kernelPfi,1724765.625,8.675,1.000
```

- Comma-Separated Values

```
method,gputime,cputime,occupancy
method=[ _Z6kernelPfi ] gputime=[ 1724745.250 ] cputime=[ 17.638 ] occupancy=[ 1.000 ]
method=[ _Z6kernelPfi ] gputime=[ 1724793.750 ] cputime=[ 5.287 ] occupancy=[ 1.000 ]
method=[ _Z6kernelPfi ] gputime=[ 1724740.500 ] cputime=[ 8.878 ] occupancy=[ 1.000 ]
```

Поддерживаемые форматы

- Comma-Separated Values включается через переменную COMPUTE_PROFILE_CSV:

```
$export COMPUTE_PROFILE_CSV=1
$COMPUTE_PROFILE=1 ./a.out
```

- Или строчкой в файле конфигурации:

```
profilelogformat CSV
```

Стандартные счетчики

- Command-line profiler по умолчанию для каждой команды, выполненной на GPU, выводит в отдельной строке:
  - **method** – имя ядра | memcpyHtoD | memcpyDtoH
  - **gputime** – сколько времени команда выполнялась на GPU
  - **cputime** – сколько времени команда выполнялась на хосте
    - Для неблокирующих команд – время запуска, затраченное хостом на запуск
    - Для блокирующих – полное время, на которое заблокировалась хост-нить
  - **occupancy**

Файл конфигурации

- Файл, в котором указаны идентификаторы интересующих опций/счетчиков, по одному в строке + строки с комментариями, начинающиеся с #
- Доступные опции:

  http://docs.nvidia.com/cuda/profiler-users-guide/index.html#command-line-profiler-options

- Доступные счетчики:

```
$nvprof --query-events
```

Файл конфигурации

- Указываем файл в переменной окружения COMPUTE_PROFILE_CONFIG:

```
$ cat profiler.config
#some comment
gpustarttimestamp
streamid
$export COMPUTE_PROFILE_CONFIG=profiler.config
$COMPUTE_PROFILE=1 ./a.out
```

Пример

```
$cat profiler.config
gpustarttimestamp
streamid
conckerneltrace
gridsize
$export COMPUTE_PROFILE_CSV=1
$export COMPUTE_PROFILE_CONFIG=profiler.config
$export COMPUTE_PROFILE_LOG=example.%d.log
$export COMPUTE_PROFILE=1
$./a.out
```

Пример

```
$cat example.0.log
# CUDA_PROFILE_LOG_VERSION 2.0
# CUDA_DEVICE 3 Tesla C2075
# CUDA_CONTEXT 1
# CUDA_PROFILE_CSV 1
# TIMESTAMPFACTOR 13a27e35a487845e
gpustarttimestamp,method,gputime,cputime,gridsizeX,gridsizeY,occupancy,streamid
13a71b99010f2020,memcpyHtoD,397217.531,1501.800,,,,1
13a71b9930a33d40,memcpyHtoD,393306.625,1338.747,,,,1
13a71b99483ee600,_Z6matmul14cudaPitchedPtrS_S_,12667.392,5.871,32,32,0.667,1
13a71b99490c0bc0,memcpyDtoH,2406.752,4200.187,,,,1
```

- Файл example.0.log можно открыть в Visual Profiler

Visual Profiler & Command-line

- Лог command-line профилировщика можно открыть в NVVP, если
  - В конфигурации указаны gpustarttimestamp и streamid - для построения timeline
  - Формат CSV
- Можно открыть сразу несколько логов – их timeline будут склеены
  - Если несколько логов получилось в результате прогонов с различными наборами счетчиков – лучше вручную объединить столбцы

Несовместимость счетчиков

- Некоторые счетчики не могут быть собраны за один прогон
  - Например, gld_inst_32bit и sm_cta_launched

- В этом случае профилировщик выберет совместимый набор счетчиков, а в лог профилировки добавит «NV_Warning: Counter 'sm_cta_launched' is not compatible with other selected counters and it cannot be profiled in this run»

- Visual Profiler автоматически делает нужное число прогонов, но с command-line profiler это приходится делать вручную

Command Line Profiler & метрики

Command-line profiler не поддерживает метрики!

nvprof

nvprof

- «Бэкенд» nvvp
- Позволяет быстро посмотреть что-сколько считается, собрать события и метрики
- Данные профилировки nvprof могут быть экспортированы в nvvp

Использование

- Компиляция остается без изменений
- Профилировка:

```
$nvprof [опции] [исполняемый файл] [параметры]
```

- Режимы:
  - Summary
  - Trace
  - Events/metrics

Режимы

- *Summary*

  Быстрый способ узнать на что ушла большая часть времени исполнения

- *Trace*

  Трасса вызовов CUDA API и GPU-операций

- *Events/metrics*

  События и метрики

Summary

```
$nvprof ./matmul 1024 1024 1024
```

- Вывести список использованных функций тулкита, ядер, пересылок данных.
- Для каждой операции:
  - Доля от общего времени выполнения, затраченное на все выполнения операции
  - Суммарное время всех выполнений операции
  - Число выполнений
  - Время выполнения (среднее, минимальное, максимальное)

Пример Summary

```
$nvprof ./matmul 1024 1024 1024
```

```
Elapsed time: 5.10317
CUDA error: no error
==12718== NVPROF is profiling process 12718, command: ./matmul 1024 1024 1024
==12718== Profiling application: ./matmul 1024 1024 1024
==12718== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 58.87%  5.0759ms         1  5.0759ms  5.0759ms  5.0759ms  matmul(cudaPitchedPtr, cudaPitchedPtr, cudaPitchedPtr)
 21.73%  1.8737ms         2  936.83us  923.18us  950.47us  [CUDA memcpy HtoD]
 19.40%  1.6731ms         1  1.6731ms  1.6731ms  1.6731ms  [CUDA memcpy DtoH]

==12718== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 59.33%  104.54ms         3  34.847ms  153.32us  104.23ms  cudaMallocPitch
 31.37%  55.274ms         1  55.274ms  55.274ms  55.274ms  cudaDeviceReset
  2.95%  5.2055ms         1  5.2055ms  5.2055ms  5.2055ms  cudaEventSynchronize
  2.84%  5.0000ms         3  1.6667ms  1.0558ms  2.7587ms  cudaMemcpy2D
  1.55%  2.7308ms       332  8.2250us     270ns  425.65us  cuDeviceGetAttribute
  1.29%  2.2730ms         4  568.24us  549.16us  579.66us  cudaGetDeviceProperties
  0.33%  577.77us         3  192.59us  161.06us  252.17us  cudaFree
  0.17%  308.30us         4  77.076us  64.779us  90.876us  cuDeviceTotalMem
  0.13%  221.07us         4  55.267us  52.939us  59.994us  cuDeviceGetName
…
```

Пример Summary

```
$nvprof ./matmul 1024 1024 1024
```

```
Elapsed time: 5.10317
CUDA error: no error
==12718== NVPROF is profiling process 12718, command: ./matmul 1024 1024 1024
==12718== Profiling application: ./matmul 1024 1024 1024
==12718== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 58.87%  5.0759ms         1  5.0759ms  5.0759ms  5.0759ms  matmul(cudaPitchedPtr, cudaPitchedPtr, cudaPitchedPtr)
 21.73%  1.8737ms         2  936.83us  923.18us  950.47us  [CUDA memcpy HtoD]
 19.40%  1.6731ms         1  1.6731ms  1.6731ms  1.6731ms  [CUDA memcpy DtoH]

==12718== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 59.33%  104.54ms         3  34.847ms  153.32us  104.23ms  cudaMallocPitch
 31.37%  55.274ms         1  55.274ms  55.274ms  55.274ms  cudaDeviceReset
  2.95%  5.2055ms         1  5.2055ms  5.2055ms  5.2055ms  cudaEventSynchronize
  2.84%  5.0000ms         3  1.6667ms  1.0558ms  2.7587ms  cudaMemcpy2D
  1.55%  2.7308ms       332  8.2250us     270ns  425.65us  cuDeviceGetAttribute
  1.29%  2.2730ms         4  568.24us  549.16us  579.66us  cudaGetDeviceProperties
  0.33%  577.77us         3  192.59us  161.06us  252.17us  cudaFree
  0.17%  308.30us         4  77.076us  64.779us  90.876us  cuDeviceTotalMem
  0.13%  221.07us         4  55.267us  52.939us  59.994us  cuDeviceGetName
…
```

Пример Summary

```
$nvprof ./matmul 1024 1024 1024
```

```
Elapsed time: 5.10317
CUDA error: no error
==12718== NVPROF is profiling process 12718, command: ./matmul 1024 1024 1024
==12718== Profiling application: ./matmul 1024 1024 1024
==12718== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 58.87%  5.0759ms         1  5.0759ms  5.0759ms  5.0759ms  matmul(cudaPitchedPtr, cudaPitchedPtr, cudaPitchedPtr)
 21.73%  1.8737ms         2  936.83us  923.18us  950.47us  [CUDA memcpy HtoD]
 19.40%  1.6731ms         1  1.6731ms  1.6731ms  1.6731ms  [CUDA memcpy DtoH]

==12718== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 59.33%  104.54ms         3  34.847ms  153.32us  104.23ms  cudaMallocPitch
 31.37%  55.274ms         1  55.274ms  55.274ms  55.274ms  cudaDeviceReset
  2.95%  5.2055ms         1  5.2055ms  5.2055ms  5.2055ms  cudaEventSynchronize
  2.84%  5.0000ms         3  1.6667ms  1.0558ms  2.7587ms  cudaMemcpy2D
  1.55%  2.7308ms       332  8.2250us     270ns  425.65us  cuDeviceGetAttribute
  1.29%  2.2730ms         4  568.24us  549.16us  579.66us  cudaGetDeviceProperties
  0.33%  577.77us         3  192.59us  161.06us  252.17us  cudaFree
  0.17%  308.30us         4  77.076us  64.779us  90.876us  cuDeviceTotalMem
  0.13%  221.07us         4  55.267us  52.939us  59.994us  cuDeviceGetName
  …
```

CPU Trace

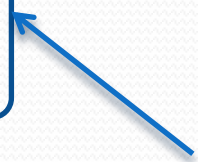
```
$nvprof --print-api-trace ./matmul 1024 1024 1024
```

- Трасса вызовов CUDA API
- Для каждого вызова в отдельной строке время старта и время выполнения
- Не только CUDA runtime API, но и CUDA driver
  - CUDA driver API – низкоуровневое API, на котором построен CUDA runtime

CPU Trace

```
$nvprof --print-api-trace ./matmul 1024 1024 1024
```

```
==13358== NVPROF is profiling process 13358, command: ./a.out 32768 1 2 2
==13358== Profiling application: ./a.out 32768 1 2 2
==13358== Profiling result:
   Start  Duration  Name
126.71ms  1.3140us  cuDeviceGetCount
126.72ms    402ns  cuDeviceGet
126.73ms    404ns  cuDeviceGet
. . .
450.57ms  206.81us  cudaMallocPitch
450.78ms  1.2243ms  cudaMemcpy2D
452.01ms  1.4435ms  cudaMemcpy2D
453.46ms  9.1320us  cudaEventCreate
```



Очень много неявных вызовов cuDeviceGet для получения параметров устройства

GPU Trace

```
$nvprof --print-gpu-trace ./matmul 1024 1024 1024
```

- Трасса GPU-операций – запуски ядер и копирования памяти
- Для каждой операции в отдельной строке
  - Старт, продолжительность
  - Для запусков ядер:
    - Грид
    - Ресурсы (регистры, общая память)
  - Для копирований памяти:
    - Объем
    - Пропускная способность GB/s
  - Устройство
  - Поток

GPU Trace

```
$nvprof --print-gpu-trace ./matmul 1024 1024 1024

==13406== NVPROF is profiling process 13406, command: ./a.out 32768 1 2 2
==13406== Profiling application: ./a.out 32768 1 2 2
==13406== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*
289.04ms  13.981ms                    -               -        -         -         -
303.02ms  15.931ms                    -               -        -         -         -
303.03ms  1.8371ms          (16384 1 1)       (512 1 1)       16        0B        0B
304.87ms  15.973ms                    -               -        -         -         -
318.98ms  1.8346ms          (16384 1 1)       (512 1 1)       16        0B        0B
320.85ms  11.025ms                    -               -        -         -         -
```

Первая половина столбцов

GPU Trace

```
$nvprof --print-gpu-trace ./matmul 1024 1024 1024
```

```
    Size  Throughput           Device   Context    Stream  Name
67.109MB  4.8001GB/s  Tesla C2075 (2)         1        13  [CUDA memcpy HtoD]
67.109MB  4.2124GB/s  Tesla C2075 (2)         1        14  [CUDA memcpy HtoD]
-           -         Tesla C2075 (2)         1        13  kernel(double*, int) [363]
67.109MB  4.2013GB/s  Tesla C2075 (2)         1        13  [CUDA memcpy DtoH]
-           -         Tesla C2075 (2)         1        14  kernel(double*, int) [369]
67.109MB  6.0872GB/s  Tesla C2075 (2)         1        14  [CUDA memcpy DtoH]
```

Вторая половина столбцов

События/метрики

```
$nvprof --devices <индекс> --query-metrics
$nvprof --devices <индекс> --query-events
```

- Запросить доступные для устройства метрики/события

```
$nvprof -m <метрика>,[<метрика>] ./matmul 1024 1024 1024
$nvprof -e <событие>,[<событие>] ./matmul 1024 1024 1024
```

- Посчитать значения метрик/событий (среднее, максимальное, минимальное) для всех запусков каждого ядра

События/метрики

```
$nvprof  -m ipc,gld_efficiency,flop_sp_efficiency ./matmul 1024 1024 1024
==13638== NVPROF is profiling process 13638, command: ./matmul 1024 1024 1024
==13638== Warning: Some kernel(s) will be replayed on device 1 in order to collect all
events/metrics.
==13638== Profiling application: ./matmul 1024 1024 1024
==13638== Profiling result:
==13638== Metric result:
Invocations  Metric Name   Metric Description             Min       Max       Avg
Device "Tesla C2075 (1)"
         Kernel: matmul(cudaPitchedPtr, cudaPitchedPtr, cudaPitchedPtr)
1       ipc                Executed IPC                   1.484029  1.484029  1.484029
1       gld_efficiency     Global Memory Load Efficiency  100.20%   100.20%   100.20%
1       flop_sp_efficiency FLOP Efficiency(Peak Single)   5.87%     5.87%     5.87%
```

Перенаправление вывода

```
$nvprof --log-file nvprof.log ./matmul 1024 1024 1024
```

- Перенаправить вывод в файл

```
$nvprof --profile-api-trace none ./matmul 1024 1024 512
```

- Не профилировать вызовы CUDA API методов на хосте

```
$nvprof -o nvprof.trace ./matmul 1024 1024 512
```

- Записать результаты профилирования в файл со специальным форматом, которые можно экспортировать в **nvvp**

Удаленное профилирование

Удаленное профилирование

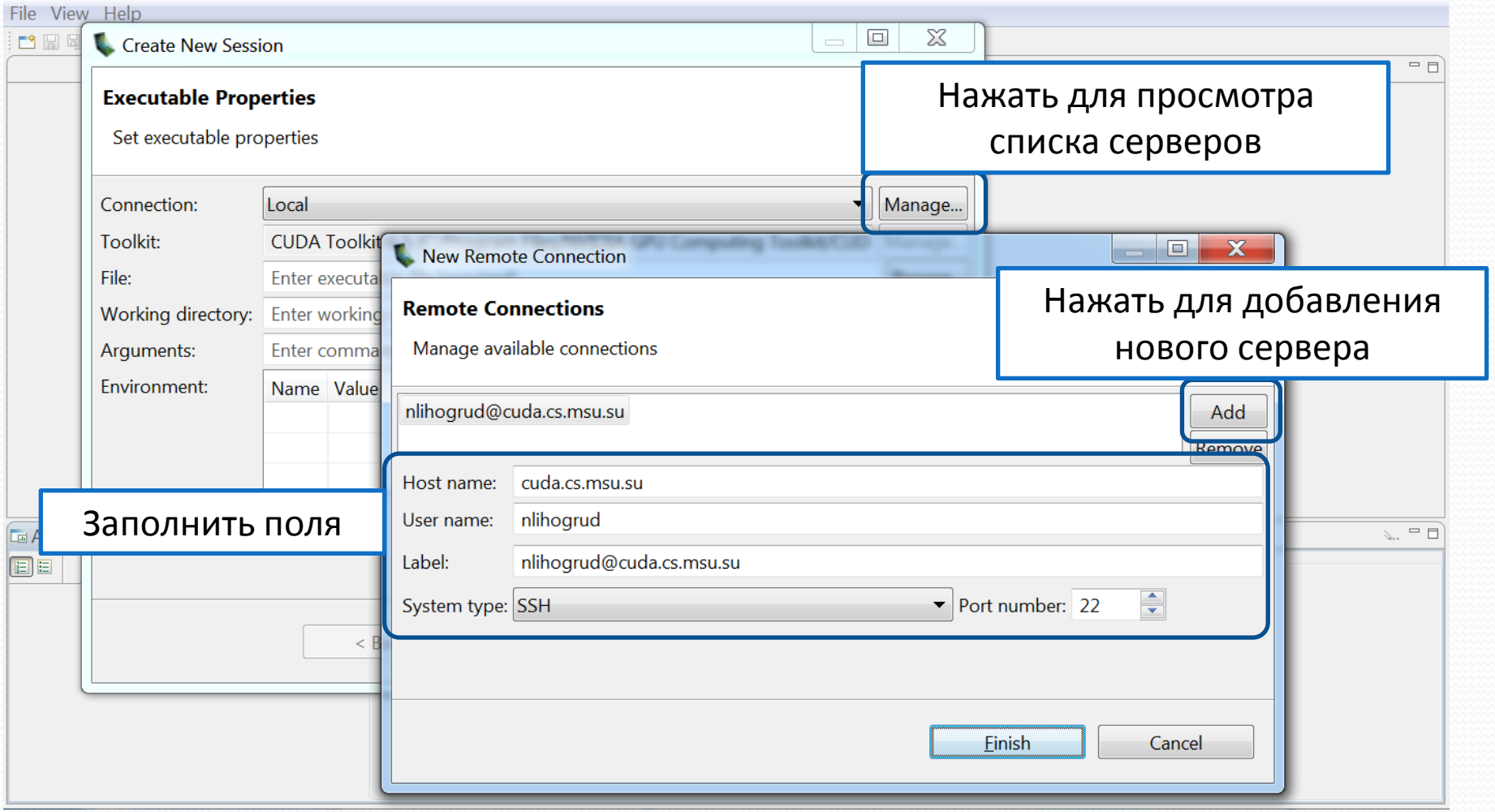
- Если на локальной машине стоит тулкит:
  - Экспорт логов **Command Line Profiler** / **nvprof** в **nvvp**
  - Удаленное профилирование прямо из **nvvp**
- Если на локальной машине не стоит тулкит:
  - Используем **nvprof**

Экспорт логов

- Получаем экспортируемые логи
  - Для command line profiler
    - В конфигурации, как минимум, указаны `gpustarttimestamp` и `streamid` - для построения timeline
    - Формат CSV
  - Для nvvp
    - `$nvprof -o nvprof.log ./matmul 1024 1024 512`
- `Открываем логи в nvvp`
  - `File->Open`

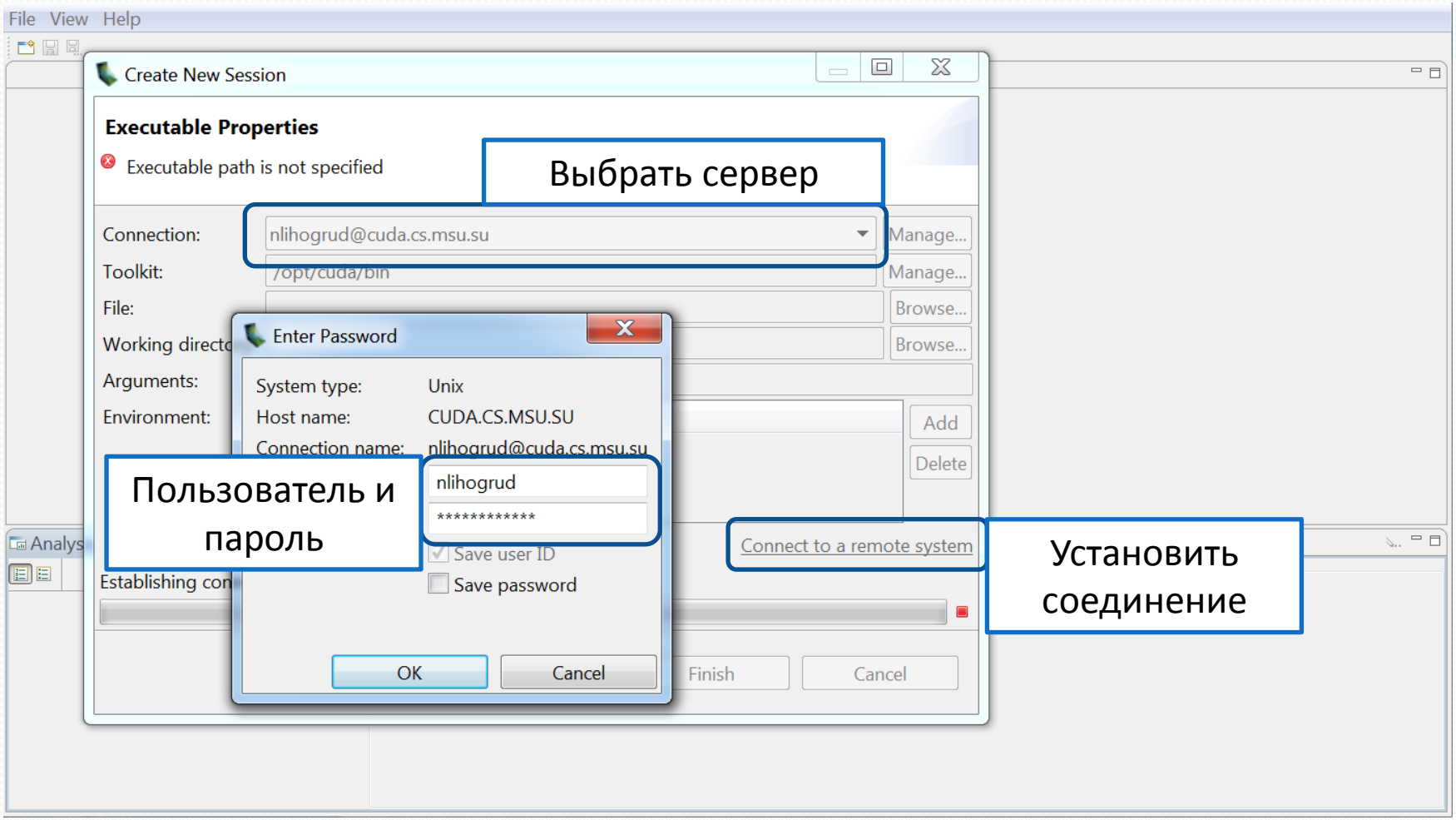
Удаленное профилирование в nvvp

- Добавление удаленного сервера



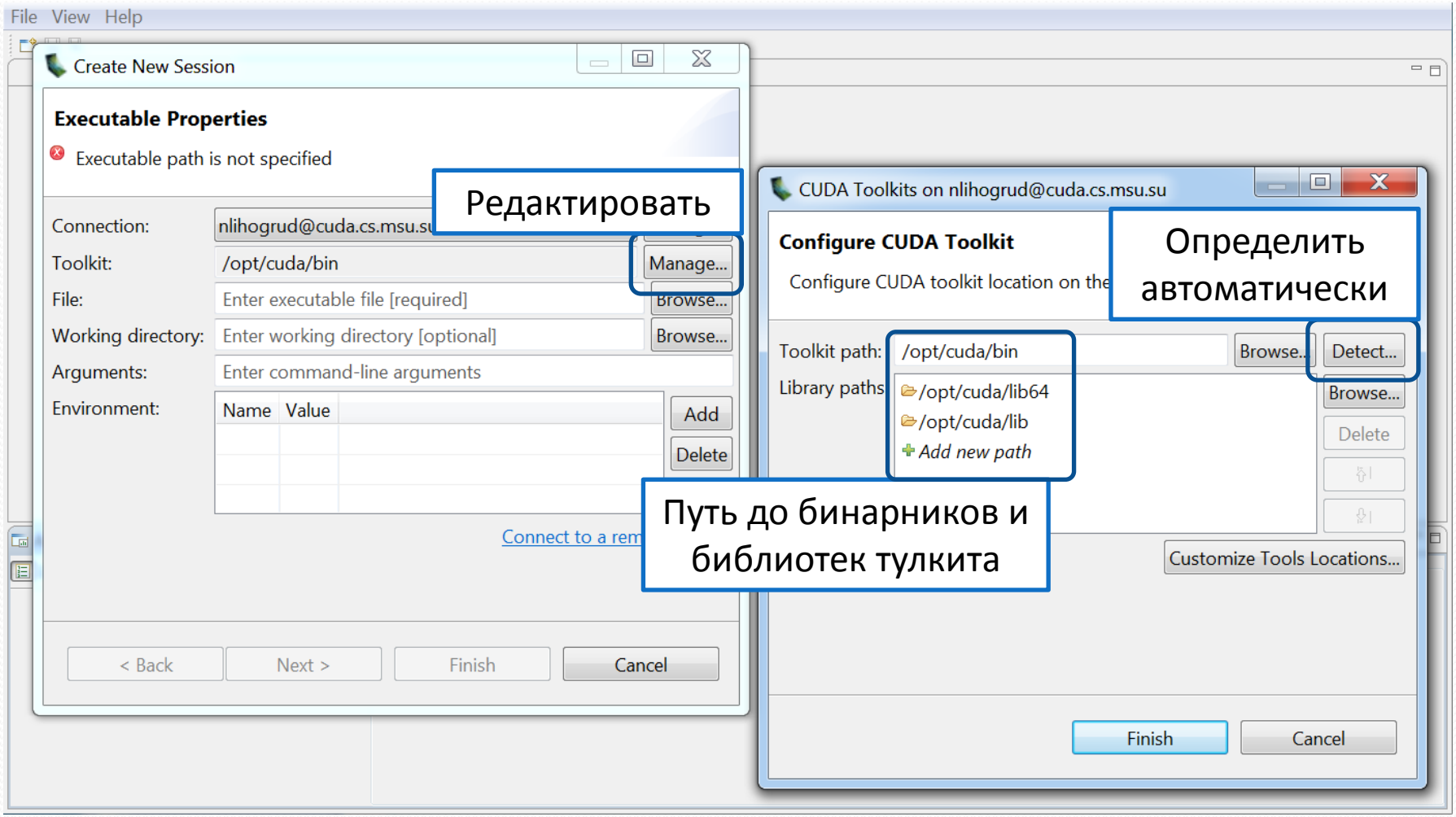
Удаленное профилирование в nvvp

- Выбор сервера, соединение



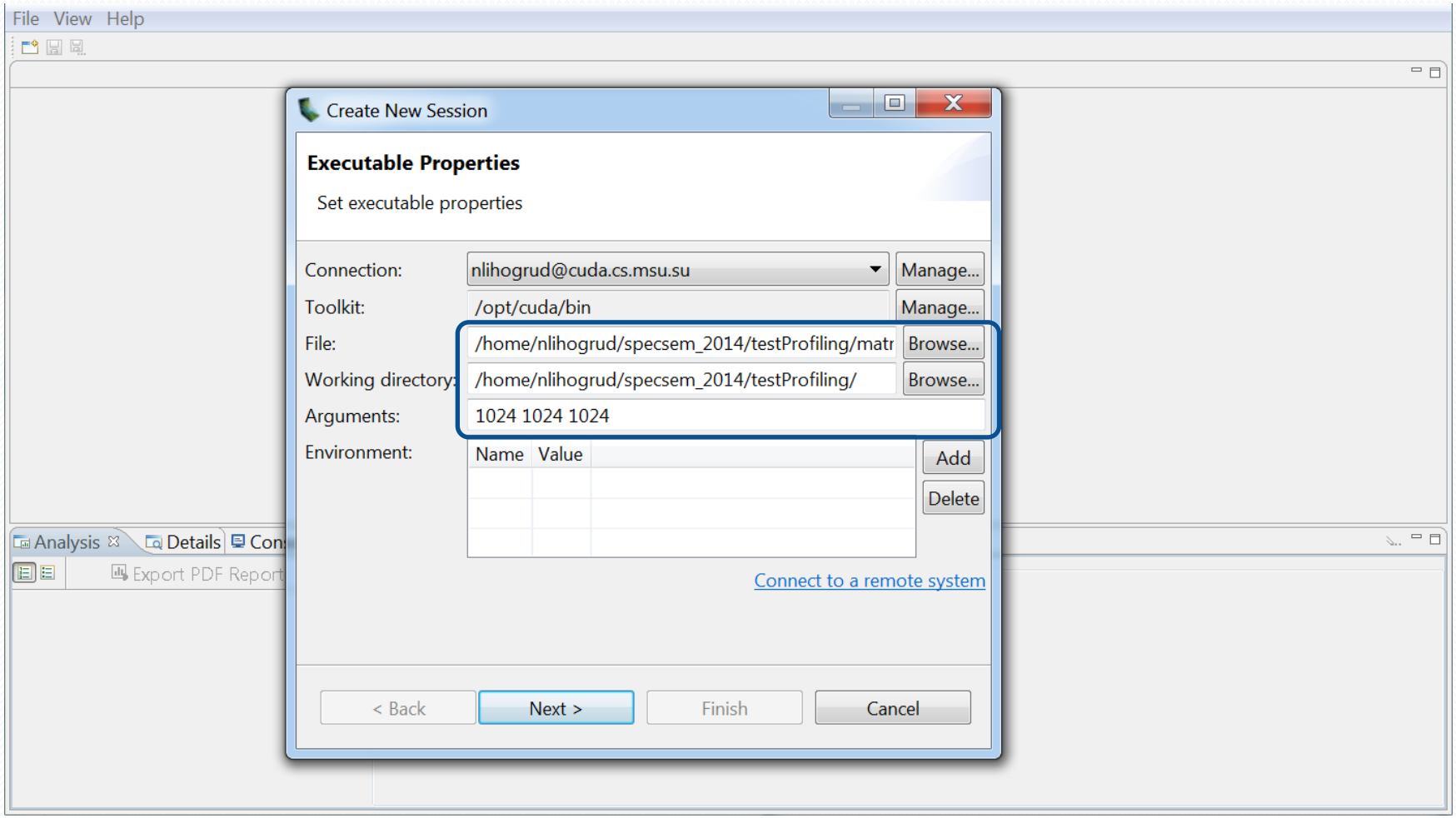
Удаленное профилирование в nvvp

- Указание пути до тулкита на удаленном сервере



Удаленное профилирование в nvvp

- Исполняемый файл, параметры запуска



Доступ без пароля

- Если нет возможности установить пароль
  - Положить приватный ключ в
    - `$HOME\.ssh\id_rsa` на Unix
    - `%UserProfile%\.ssh\id_rsa` на Windows
  - При подключении к серверу из **nvvp** оставить поле пароля пустым
- **nvvp** не позволяет указать приватный ключ, поэтому кладем с именем **id_rsa**

Отладка с cuda-gdb

Компиляция и запуск

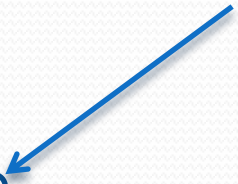
- Компилируем с флагами –g –G:

```
$nvcc –arch=sm_20 -c -g -G kernel.cu -o kernel.o
```

  - -g – для отладки хост-кода, -G для device-кода
  - Не проводит оптимизации, добавляет в объектные файлы отладочные символы

- Запуск:

```
$cuda-gdb ./matmul
NVIDIA (R) CUDA Debugger
6.5 release
...
(cuda-gdb) r 1024 1024 1024
```



Запустить программу с заданными параметрами

Новые команды в cuda-gdb

- `cuda-gdb` содержит расширенный набор команд
- Новые команды разделены на три группы:

`(cuda-gdb) cuda <команда>`

- Переключение между нитями, ядрами, варпами и т.д., просмотр текущего местоположения (focus)

`(cuda-gdb) info cuda <команда>`

- Состояние системы: устройства, работающие ядра, нити и т.д.

`(cuda-gdb) set cuda <команда>`

- Различные настройки

- Для всех команд работает автодополнение

Справка по командам

- Справка по отдельной команде выводится командой `help`:

```
(cuda-gdb) help cuda <команда>
(cuda-gdb) help info cuda <команда>
(cuda-gdb) help set cuda <команда>
```

- Если команда не указана, то выведется информация о всей группе

```
(cuda-gdb) help set cuda
(cuda-gdb) help cuda
(cuda-gdb) help set cuda
```

Стандартные команды gdb

- cuda-gdb поддерживает все стандартные команды gdb применительно к CUDA-коду

```
(cuda-gdb) b[reak] kernel.cu:15
(cuda-gdb) b[reak] simpleKernel
(cuda-gdb) cond[ition] 3 threadIdx.x == 1 && i < 5
(cuda-gdb) list
(cuda-gdb) n[ext]
(cuda-gdb) c[ontinue]
...
```

Остановка в начале ядра

- Вручную поставить breakpoint на ядро:

```
(cuda-gdb) break <имя ядра>
```

- Автоматически поставить метки в начала всех ядер:

```
(cuda-gdb) set cuda break_on_launch application
(cuda-gdb) set cuda break_on_launch all
```

- Справка:

```
(cuda-gdb) help set cuda break_on_launch
```

Фокус

- Фокус – полное определение остановленной нити
  - Ядро, грид, индексы блока (`blockIdx`), индексы нити (`threadIdx`), устройство, мультипроцессор, варп, номер нити в варпе (`lane`)

- Автоматически выводится при остановке:

```
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,21,0),
device 1, sm 1, warp 21, lane 0]

Breakpoint 1, simple<<<(64,64,1),(32,32,1)>>> (A=..., B=..., C=...) at
kernel.cu:15
15    int blockY = blockIdx.y * blockDim.y;
```

Фокус

- Вывести текущий фокус:
  - Частично:

```
(cuda-gdb) cuda block thread
block (0,0,0), thread (0,21,0)
```

  - Полностью:

```
(cuda-gdb) cuda kernel grid block thread device sm warp lane
kernel 0, grid 1, block (0,0,0), thread (0,21,0), device 1, sm 1, warp
21, lane 0
```

Переключение фокуса

- Переключение фокуса:

```
(cuda-gdb) cuda <параметр> <значение>, <параметр> <значение>, ...
```

- Фокус полностью задается
  - Физическими параметрами: устройство, мультипроцессор, варп, нить в варпе (lane)

```
(cuda-gdb) cuda device 1 sm 3 warp 0 lane 3
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (3,0,0),
device 1, sm 3, warp 0, lane 3]
12   __global__ void simple(cudaPitchedPtr A, cudaPitchedPtr B) {
```

Переключение фокуса

- Переключение фокуса:

  `(cuda-gdb) cuda <параметр> <значение>, <параметр> <значение>, ...`

- Фокус полностью задается
  - Виртуальными параметрами: ядро, грид, блок, нить

```
(cuda-gdb) cuda kernel 0 grid 1 block (2,0) thread (23,6,0)
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (23,6,0),
device 1, sm 3, warp 6, lane 23]
12   __global__ void simple(cudaPitchedPtr A, cudaPitchedPtr B) {
```

Переключение фокуса

- Если указаны не все параметры, то остальным будут неявно присвоены значения из текущего фокуса,

```
(cuda-gdb) cuda thread
thread (0,3,0)
(cuda-gdb) cuda thread 2
[Switching focus to CUDA kernel 0, grid 1, block (5,0,0), thread (2,3,0),
device 1, sm 2, warp 3, lane 2]
14        int blockX = blockIdx.x * blockDim.x;
```

Поиск ошибок использования API

- Если `cudaGetLastError()` вернул что-то отличное от `cudaSuccess`, то
  1. Включаем опцию остановки при ошибке API
  2. Выводим стек вызовов в точке остановки

```
$cuda-gdb ./matmul
(cuda-gdb) set cuda api_failures stop
(cuda-gdb) r
Starting program:
. . .
Cuda API error detected: cudaLaunch returned (0x9)
(cuda-gdb) bt
. . .
#8  0x00000000004019cd in launchKernels (A=..., B=..., C=...) at kernel.cu:181
#9  0x000000000040135f in main (argc=1, argv=0x7fffffffdab8) at main.cpp:44
```

Строчка, содержащая ошибку использования API

Поиск ошибок работы с памятью

- Если `cudaGetLastError()` вернул ошибку «`an illegal memory access was encountered`», то отладчик автоматически остановится на строчке внутри ядра, в которой произошла ошибка

```
(cuda-gdb) r
...
CUDA Exception: Warp Out-of-range Address
Program received signal CUDA_EXCEPTION_5, Warp Out-of-range Address.
[Switching focus to CUDA kernel 1, grid 3, block (4,0,0), thread (0,0,0),...]
0x00000000007e6b40 in sharedMem<<<(64,64,1),(32,32,1)>>> (...) at kernel.cu:59
59                  res += partA[threadIdx.y + 1000][k] * partB[k][threadIdx.x];
```

Выводы

Выводы

- Используйте `nvprof/nvvp` вместо событий для замера времени выполнения ядер/копирований
- Используйте профилировщик для выявления проблемных мест программы и выбора направления оптимизации через анализ timeline/метрик
- Поставьте локально тулкит для полноценной удаленной профилировки
- Если произошла ошибка в ядре или вызове функции API – сразу запускайте `cuda-gdb`, включите `set cuda api_failures stop`
- В тулкит также входит утилита `cuda-memcheck` – аналог обычного `memcheck`

Документация

- По профилировщикам
  http://docs.nvidia.com/cuda/profiler-users-guide
- По отладчику
  http://docs.nvidia.com/cuda/cuda-gdb
- Cuda-memcheck
  http://docs.nvidia.com/cuda/cuda-memcheck/

The end