Устройство CUDA-компилятора

Андрей Сафронов

a.safronov@parallel-compute.com

July 1, 2012

Стадии компиляции

.cu

cpp

cudafe → cpp

cpp → cudafe → cpp

nvopencc/cicc

ptxas

fatbin → device repo

fatbin → cpp

Поддерживаемые форматы файлов

.cu	Исходный фаил CUDA. Содержит host- и device-код
.cup	Исходный файл CUDA после препроцессинга
.c	Исходный файл C
.cc,.cxx,.cpp	Исходный файл C++
.gpu	Промежуточный файл GPU
.ptx	Промежуточный файл ассемблера
.o,.obj	Объектный файл
.a,.lib	Файл библиотеки
.res	Файл ресурсов
.so	Файл динамической библиотеки

Типы опций компилятора

- Бинарные опции
- Опции с единственным значением
- Опции-списки

Формы записи опций

- Краткая форма
- Полная полная

```
-o file
-o=file
-Idir1,dir2 -I=dir3 -I dir4,dir5
```

Выбор фазы компиляции

-c	C/C++ компиляция в объектный файл
-ptx	Генерация PTX-кода
-E	C/C++ препроцессинг

Опции nvcc

-v	Вывод информации о промежуточных шагах компиляции
-keep	Компиляция с сохранением промежуточных файлов
-clean	Удаление промежуточных файлов

```
$nvcc sum_kernel.cu -keep
$nvcc sum_kernel.cu -keep -clean
```

Опции nvcc

--output-file *file*	-o	Задает имя выходного файла.
--pre-include *file,...*	-include	Задает заголовочные файлы, которые будут включены в сборку.
--library *file,...*	-l	Задает файлы библиотек, которые будут использованы при компоновке.
--define-macro *macrodef,...*	-D	Задает макроопределения.
--undefine-macro *macrodef,...*	-U	Удаляет макроопределения.
--include-path *path,...*	-I	Задает путь для поиска заголовков.
--library-path *path,...*	-include	Задает путь для поиска библиотек.

Опции nvcc

--gpu-architecture *gpuarch*	-arch	Задает название архитектуры NVIDIA GPU для компиляции.
--gpu-code *gpuarch*,...	-code	Задает названия архитектур для генерации кода.
--use_fast_math	-use_fast_math	Заменяет математические функции быстрыми аналогами.

Опции для определенных фаз

--compiler-options *options*,...	-Xcompiler	Задает опции для компилятора/препроцессора.
--cudafe-options *options*,...	-Xcudafe	Задает опции для cudafe.
--opencc-options *options*,...	-Xopencc	Задает опции для opencc.
--ptxas-options *options*,...	-Xptxas	Задает опции для ptx-ассемблера.
--linker-options *options*,...	-Xlinker	Задает опции для компоновщика.

Процесс копиляции

```
[edu-acad12-prep01@graphit sum_kernel]$ nvcc -v -DBLOCK_SIZE=512 sum_kernel.cu
```

Информация о переменных окружения:

```
#$ _SPACE_=
#$ _CUDART_=cudart
#$ _HERE_=/usr/local/cuda/bin
#$ _THERE_=/usr/local/cuda/bin
#$ _TARGET_SIZE_=64
#$ TOP=/usr/local/cuda/bin/..
#$ LD_LIBRARY_PATH=/usr/local/cuda/bin/../lib:/usr/local/cuda/bin/../extools/lib:
#$ PATH=/usr/local/cuda/bin/../open64/bin:/usr/local/cuda/bin/../nvvm:/usr/local/cuda↩
    /bin:/opt/jre1.6.0_18/bin:/usr/lib64/qt-3.3/bin:/export/opt/pgi/linux86↩
    -64/12.3/bin:/opt/openmpi/1.4.2/bin:/usr/kerberos/bin:/usr/local/cuda/bin:/usr/↩
    local/cuda/open64/bin:/usr/local/bin:/bin:/usr/bin:/export/usr/local/bin:/↩
    export/usr/bin:/home/edu-acad12-prep01/bin
#$ INCLUDES="-I/usr/local/cuda/bin/../include" "-I/usr/local/cuda/bin/../include/↩
    cudart"
#$ LIBRARIES=  "-L/usr/local/cuda/bin/../lib64" -lcudart
#$ CUDAFE_FLAGS=
#$ OPENCC_FLAGS=
#$ PTXAS_FLAGS=
```

Процесс компиляции

Препроцессинг:

```
#$ gcc -D__CUDA_ARCH__=100 -E -x c++   -DCUDA_FLOAT_MATH_FUNCTIONS -↩
    DCUDA_NO_SM_11_ATOMIC_INTRINSICS -DCUDA_NO_SM_12_ATOMIC_INTRINSICS -↩
    DCUDA_NO_SM_13_DOUBLE_INTRINSICS  -D__CUDACC__  "-I/usr/local/cuda/bin/../↩
    include" "-I/usr/local/cuda/bin/../include/cudart"   -D"BLOCK_SIZE=512" -↩
    include "cuda_runtime.h" -m64 -o "/tmp/tmpxft_00007f67_00000000-4_sum_kernel.↩
    cpp1.ii" "sum_kernel.cu"
```

Процесс копиляции

Разделение device- и host-кода:

```
#$ cudafe --m64 --gnu_version=40102 -tused --no_remove_unneeded_entities --↩
    gen_c_file_name "/tmp/tmpxft_00007f67_00000000-1_sum_kernel.cudafe1.c" --↩
    stub_file_name "/tmp/tmpxft_00007f67_00000000-1_sum_kernel.cudafe1.stub.c" --↩
    gen_device_file_name "/tmp/tmpxft_00007f67_00000000-1_sum_kernel.cudafe1.gpu" ↩
    --include_file_name "tmpxft_00007f67_00000000-3_sum_kernel.fatbin.c" "/tmp/↩
    tmpxft_00007f67_00000000-4_sum_kernel.cpp1.ii"
```

Процесс компиляции

генерация промежуточного PTX-кода:

```
#$ nvopencc  -TARG:compute_10 -m64 -OPT:ftz=1 -CG:ftz=1 -CG:prec_div=0 -CG:prec_sqrt↩
    =0  "/tmp/tmpxft_00007f67_00000000-10_sum_kernel" "/tmp/↩
    tmpxft_00007f67_00000000-7_sum_kernel.cpp3.i"  -o "/tmp/↩
    tmpxft_00007f67_00000000-2_sum_kernel.ptx"
```

Процесс компиляции

Генерация асемблера из PTX-кода:

```
#$ ptxas  -arch=sm_10 -m64  "/tmp/tmpxft_000056ae_00000000-2_sum_kernel.ptx"  -o "/↩
    tmp/tmpxft_000056ae_00000000-11_sum_kernel.sm_10.cubin"
```

Процесс компиляции

Внедрение ассемблера в С код:

```
#$ fatbinary --create="/tmp/tmpxft_000056ae_00000000-3_sum_kernel.fatbin" --key="77↩
    f5b7eca9cc46be" --ident="sum_kernel.cu" -cuda "--image=profile=compute_10,file↩
    =/tmp/tmpxft_000056ae_00000000-2_sum_kernel.ptx" "--image=profile=sm_10,file=/↩
    tmp/tmpxft_000056ae_00000000-11_sum_kernel.sm_10.cubin" --embedded-fatbin="/tmp↩
    /tmpxft_000056ae_00000000-3_sum_kernel.fatbin.c"
```

Процесс компиляции

Компиляция host-кода со встроенным device-ассемблером в объектный файл:

```
#$ gcc -c -x c++ "-I/usr/local/cuda/bin/../include" "-I/usr/local/cuda/bin/../include↩
    /cudart"   -fpreprocessed -m64 -o "/tmp/tmpxft_000056ae_00000000-13_sum_kernel.↩
    o" "/tmp/tmpxft_000056ae_00000000-12_sum_kernel.ii"
```

PTX ассемблер

```
.version 1.4
.target sm_10, map_f64_to_f32

  .entry _Z10sum_kernelPfS_S_ (
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__a,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__b,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__c)
  {
  .reg .u16 %rh<4>;
  .reg .u32 %r<5>;
  .reg .u64 %rd<10>;
  .reg .f32 %f<5>;
  .loc  14  2  0
$LDWbegin__Z10sum_kernelPfS_S_:
  .loc  14  8  0
```

Версия PTX ассемблера,
целевая архитектура,
использование float вместо double

PTX ассемблер

```
.version 1.4
.target sm_10, map_f64_to_f32

  .entry _Z10sum_kernelPfS_S_ (
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__a,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__b,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__c)
  {
  .reg .u16 %rh<4>;
  .reg .u32 %r<5>;
  .reg .u64 %rd<10>;
  .reg .f32 %f<5>;
  .loc  14  2 0
$LDWbegin__Z10sum_kernelPfS_S_:
  .loc  14  8 0
```

Заголовок ядра
с тремя параметрами

PTX ассемблер

```
.version 1.4
.target sm_10, map_f64_to_f32

  .entry _Z10sum_kernelPfS_S_ (
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__a,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__b,
    .param .u64 __cudaparm__Z10sum_kernelPfS_S__c)
  {
  .reg .u16 %rh<4>;
  .reg .u32 %r<5>;
  .reg .u64 %rd<10>;
  .reg .f32 %f<5>;
  .loc  14  2  0
$LDWbegin__Z10sum_kernelPfS_S_:
  .loc  14  8  0
```

Выделение необходимых регистров

PTX ассемблер

```
  cvt.u32.u16   %r1, %tid.x;
  mov.u16   %rh1, %ctaid.x;
  mov.u16   %rh2, %ntid.x;
  mul.wide.u16  %r2, %rh1, %rh2;
  add.u32   %r3, %r1, %r2;
  cvt.s64.s32   %rd1, %r3;
  mul.wide.s32  %rd2, %r3, 4;
  ld.param.u64  %rd3, [__cudaparm__Z10sum_kernelPfS_S__a];
  add.u64   %rd4, %rd3, %rd2;
  ld.global.f32   %f1, [%rd4+0];
  ld.param.u64  %rd5, [__cudaparm__Z10sum_kernelPfS_S__b];
  add.u64   %rd6, %rd5, %rd2;
  ld.global.f32   %f2, [%rd6+0];
  add.f32   %f3, %f1, %f2;
  ld.param.u64  %rd7, [__cudaparm__Z10sum_kernelPfS_S__c];
  add.u64   %rd8, %rd7, %rd2;
  st.global.f32   [%rd8+0], %f3;
  .loc  14  9  0
  exit;
$LDWend__Z10sum_kernelPfS_S_:
  } // _Z10sum_kernelPfS_S_
```

Загрузка параметров сетки
из специальных регистров
в регистры общего назначения,
вычисление абсолютного индекса массива

PTX ассемблер

```
$LDWbegin__Z10sum_kernelPfS_S_:
  .loc   14  8  0
  cvt.u32.u16   %r1, %tid.x;
  mov.u16   %rh1, %ctaid.x;
  mov.u16   %rh2, %ntid.x;
  mul.wide.u16  %r2, %rh1, %rh2;
  add.u32   %r3, %r1, %r2;
  cvt.s64.s32   %rd1, %r3;
  mul.wide.s32  %rd2, %r3,  4;
  ld.param.u64  %rd3, [__cudaparm__Z10sum_kernelPfS_S__a]
  add.u64   %rd4, %rd3, %rd2;
  ld.global.f32   %f1, [%rd4+0];
  ld.param.u64  %rd5, [__cudaparm__Z10sum_kernelPfS_S__b]
  add.u64   %rd6, %rd5, %rd2;
  ld.global.f32   %f2, [%rd6+0];
  add.f32   %f3, %f1, %f2;
  ld.param.u64  %rd7, [__cudaparm__Z10sum_kernelPfS_S__c];
  add.u64   %rd8, %rd7, %rd2;
  st.global.f32   [%rd8+0], %f3;
  .loc   14  9  0
  exit;
```

Загрузка базовых адресов 3-х массивов, применение смещения, вычисление результата

Типы памяти в .ptx

.reg, .sreg	Регистры.
.local	Локальная память.
.global	Глобальная память.
.param	Параметры функций.
.shared	Распределенная память.
.tex	Текстурная память.
.const	Константная память.

Архитектура GPU

--gpu-architecture *gpuarch*	-arch	Задает название архитектуры NVIDIA GPU для компиляции.
--gpu-code *gpuarch*,...	-code	Задает названия архитектур для генерации кода.

Поддерживаемые архитектуры

sm_10	ISA_1 Базовый функционал
sm_11	+ атомарные операции на глобальной памяти
sm_12	+ атомарные операции на распределенной памяти + избирательные инструкции
sm_13	+ числа с плавающей точкой двойной точности
sm_2x	+ поддержка FERMI
sm_3x	+ поддержка KEPLER

Статическая компиляция

x.cu (device-код)

Стадия 1 (nvopencc/cicc)

Виртуальная *compute* архитектура

x.ptx

Стадия 2 (ptxas)

Реальная *sm* архитектура

x.cubin

CUDA device driver

Execute

Виртуальные архитектуры

compute_10	ISA_1 Базовый функционал
compute_11	+ атомарные операции на глобальной памяти
compute_12	+ атомарные операции на распределенной памяти + избирательные инструкции
compute_13	+ числа с плавающей точкой двойной точности
compute_2x	+ поддержка FERMI
compute_3x	+ поддержка KEPLER

JIT компиляция

x.cu (device-код)

Стадия 1
(nvopencc/cicc)

Виртуальная *compute* архитектура

x.ptx

CUDA device driver

Стадия 2 (ptxas)

Реальная *sm* архитектура

x.cubin

Execute

Примеры использования

Пример 1:

```
nvcc x.cu -arch=compute_10 -code=sm_10,sm_13
```

Пример 2:

```
nvcc x.cu -arch=compute_10 -code=compute_10,sm_10,sm_13
```

Пример 3:

```
nvcc x.cu -code=sm_13
nvcc x.cu -code=compute_10
```

является сокращением для команд:

```
nvcc x.cu -arch=compute_10 -code=sm_13
nvcc x.cu -arch=compute_10 -code=compute_10
```

Примеры использования

Пример 4:

```
nvcc x.cu -arch=sm_13
nvcc x.cu -arch=compute_10
```

является сокращением для команд:

```
nvcc x.cu −arch=compute_13 -code=sm_13,compute_13
nvcc x.cu −arch=compute_10 -code=compute_10
```

Пример 5:

```
nvcc x.cu
```

является сокращением для команды:

```
nvcc x.cu −arch=compute_10 -code=sm_10,compute_10
```

Статистика PTX

```
$nvcc -Xptxas -v sum_kernel.cu
```

```
ptxas info    : Compiling entry function '_Z10sum_kernelPfS_S_' for 'sm_10'
ptxas info    : Used 4 registers, 24+16 bytes smem
```

Оптимизация кода (глобальная память)

```
__global__ void kernel(float3 * a)
{
    a[threadIdx.x] = f3();
}
```

```
.loc 14 7 0
ld.param.u64 %rd1, [__cudaparm__Z3badP6float3_a];
cvt.u64.u16 %rd2, %tid.x;
mul.lo.u64 %rd3, %rd2, 12;
add.u64 %rd4, %rd1, %rd3;
mov.f32 %f1, %f2;
st.global.f32 [%rd4+0], %f1;
mov.f32 %f3, %f4;
st.global.f32 [%rd4+4], %f3;
mov.f32 %f5, %f6;
st.global.f32 [%rd4+8], %f5;
.loc 14 8 0
exit;
```

Оптимизация кода (глобальная память)

```
__global__ void good(float4 * a)
{
  a[threadIdx.x] = f4();
}
```

```
.loc   14   7  0
  ld.param.u64  %rd1, [__cudaparm__Z3goodP6float4_a];
  cvt.u64.u16   %rd2, %tid.x;
  mul.lo.u64  %rd3, %rd2, 16;
  add.u64   %rd4, %rd1, %rd3;
  mov.f32   %f1, %f2;
  mov.f32   %f3, %f4;
  mov.f32   %f5, %f6;
  mov.f32   %f7, %f8;
 st.global.v4.f32  [%rd4+0], {%f1,%f3,%f5,%f7};
  .loc   14   8  0
  exit;
```

Оптимизация кода (быстрая математика)

```
__global__ void kernel(float *data)
{
    float a = (float) threadIdx.x;
      data[threadIdx.x] = sinf(a);
}
```

```
ptxas info    : Compiling entry function '_Z6kernelPf' for 'sm_10'
ptxas info    : Used 11 registers, 28+0 bytes lmem, 8+16 bytes smem, 24 bytes cmem↩
     [0], 100 bytes cmem[1]
```

Оптимизация кода (быстрая математика)

```
__global__ void kernel(float *data)
{
    float a = (float) threadIdx.x;
      data[threadIdx.x] = __sinf(a);
}
```

```
ptxas info    : Compiling entry function '_Z6kernelPf' for 'sm_10'
ptxas info    : Used 3 registers, 8+16 bytes smem
```

Оптимизация кода (локальная память)

```
__global__ void kernel (float4 *pDst, int * pSrc)
{
  float4 r;
  int a = pSrc[threadIdx.x];
  ((float*)(&r))[a] = 0.0f;
  pDst[threadIdx.x] = r;
}
```

```
ld.param.u64  %rd9, [__cudaparm__Z6kernelP6float4Pi_pDst];
  mul.lo.u64  %rd10, %rd1, 16;
  add.u64   %rd11, %rd9, %rd10;
  ld.local.f32  %f2, [__cuda___cuda_local_var_14497_10_non_const_r_016+0];
  ld.local.f32  %f3, [__cuda___cuda_local_var_14497_10_non_const_r_016+4];
  ld.local.f32  %f4, [__cuda___cuda_local_var_14497_10_non_const_r_016+8];
  ld.local.f32  %f5, [__cuda___cuda_local_var_14497_10_non_const_r_016+12];
  st.global.v4.f32  [%rd11+0], {%f2,%f3,%f4,%f5};
  .loc  14  7 0
  exit;
```

Оптимизация кода (локальная память)

```
__global__ void kernel (float4 *pDst, int * pSrc)
{
  float4 r;
  int a = pSrc[threadIdx.x];
  r.x = (a == 0 ? 0.0f : r.x);
  r.y = (a == 1 ? 0.0f : r.y);
  r.z = (a == 2 ? 0.0f : r.z);
  r.w = (a == 3 ? 0.0f : r.w);

  pDst[threadIdx.x] = r;
}
```

В ассемблере отсутствуют обращения в локальную память.

```
.loc 14 4 0
  cvt.u64.u16   %rd1, %tid.x;
  ld.param.u64  %rd2, [__cudaparm__Z6kernelP6float4Pi_pSrc];
  mul.lo.u64  %rd3, %rd1, 4;
  add.u64   %rd4, %rd2, %rd3;
  ld.global.s32   %r1, [%rd4+0];
  .loc  14  10  0
  ld.param.u64  %rd5, [__cudaparm__Z6kernelP6float4Pi_pDst];
```

Оптимизация кода (локальная память)

```
add.u64    %rd7, %rd5, %rd6;
mov.f32    %f1, %f2;
mov.f32    %f3, 0f00000000;      // 0
mov.s32    %r2, 0;
setp.ne.s32    %p1, %r1, %r2;
selp.f32   %f4, %f1, %f3, %p1;
mov.f32    %f5, %f6;
mov.f32    %f7, 0f00000000;      // 0
mov.s32    %r3, 1;
setp.ne.s32    %p2, %r1, %r3;
selp.f32   %f8, %f5, %f7, %p2;
mov.f32    %f9, %f10;
mov.f32    %f11, 0f00000000;     // 0
mov.s32    %r4, 2;
setp.ne.s32    %p3, %r1, %r4;
selp.f32   %f12, %f9, %f11, %p3;
mov.f32    %f13, %f14;
mov.f32    %f15, 0f00000000;     // 0
mov.s32    %r5, 3;
setp.ne.s32    %p4, %r1, %r5;
selp.f32   %f16, %f13, %f15, %p4;
st.global.v4.f32   [%rd7+0], {%f4,%f8,%f12,%f16};
```

Оптимизация кода (auto память)

```
#define AUTO_SIZE 64
__global__ void kernel_with_auto(int * out) {
  int auto_array[AUTO_SIZE];
  for(int i = 0; i < AUTO_SIZE; i ++)
    auto_array[i] = i;
  *out = auto_array[AUTO_SIZE - 1];
}
```

Массив auto_array размещается в регистрах

```
  .entry _Z16kernel_with_autoPi (
    .param .u64 __cudaparm__Z16kernel_with_autoPi_out)
  {
  .reg .u32 %r<129>;
  .reg .u64 %rd<3>;
  .loc  14  3 0
$LDWbegin__Z16kernel_with_autoPi:
  .loc  14  7 0
  mov.s32   %r1, 0;
  mov.s32   %r2, %r1;
  mov.s32   %r3, 1;
  mov.s32   %r4, %r3;
```

Оптимизация кода (auto память)

```
mov.s32   %r6, %r5;
mov.s32   %r7,  3;
mov.s32   %r8, %r7;
mov.s32   %r9,  4;
mov.s32   %r10, %r9;
mov.s32   %r11,  5;
mov.s32   %r12, %r11;
mov.s32   %r13,  6;
mov.s32   %r14, %r13;
mov.s32   %r15,  7;
mov.s32   %r16, %r15;
mov.s32   %r17,  8;
mov.s32   %r18, %r17;
mov.s32   %r19,  9;
mov.s32   %r20, %r19;
mov.s32   %r21,  10;
mov.s32   %r22, %r21;
mov.s32   %r23,  11;
```

......

Задание на дом

Найти различия в работе с глобальными переменными в CUDA-C и C++.

Исходные коды программ:

```
/tmp/cuda-tasks/double/
```

Задание на дом