Адаптация программ для GPU

Романенко А.А.
arom@ccfit.nsu.ru

Новосибирский государственный университет

Особенности GPU

- GPU — процессор с SIMD архитектурой
- Большое количество потоков (~1000), выполняющих одни и те же действия над разными данными
- «Простые» операции
- Нет рекурсии

Переносимые задачи

- Возможность разбиения задачи/данных на большое количество подобластей, в которых вычисления идентичны по выполняемым операциям
- Работа с целыми данными или данными в одинарной точности. При работе с двойной точностью падение производительности более 4 раз.
- Примеры адаптированных задач на www.nvidia.com/object/cuda_home.html

Классы задач

- Классы задач, которые в общем случае невозможно распараллелить:
  - сжатие данных
  - IIR-фильтры
  - другие рекурсивные алгоритмы

Подход к адаптации

- Анализ программы на счет поиска участков, которые можно обработать в SIMD режиме.
- Выделение найденных участков в функции.
- Выбор способа распределения данных.
- Реализация для каждой из функций ядра для GPU.
- Замена функций из п.2 последовательностью копировать данные на GPU — запуск ядра — копировать данные на CPU — проверка результатов вычислений.
- Избавление от ненужных пересылок данных между GPU и CPU
- Оптимизация программы.

Пример 1. Поиск мотивов

- **Задача**: Есть массив РНК последовательностей. Необходимо найти для всех мотивов количество последовательностей РНК, в которых данный мотив встречается.
- Длина последовательностей – 50..64 символа. ATGC-код
- Длина мотива – 8 символов. 15-ти буквенный код.
- Способы распределения данных:
  - Один поток обрабатывает один мотив и одну последовательность РНК
  - Один поток обрабатывает один мотив и все последовательности РНК
  - Один поток обрабатывает группу мотивов и одну последовательность РНК

Пример 2. Цунами

Распространение волны цунами описывается системой уравнений:

$$H_t + (uH)_x + (vH)_y = 0,$$

$$u_t + uu_x + vu_y + gH_x = gD_x,$$

$$v_t + uv_x + vv_y + gH_y = gD_y,$$

где ***H(x, y, t) = (x, y, t) + D(x, y, t)***; – общий столб воды; ***D*** – профиль глубин, ***u(x, y, t), v(x, y, t)*** – компоненты скорости вдоль осей ***x*** и ***y***; ***g*** – ускорение свободного падения.

Основной цикл расчета

```
for(n=0; n<mmax; n++){ // for each time step
        for(j = 0; j < n1; j ++) {  // calculate along Y
            for(i=0; i<n2; i++) {
                qw[i] = u[j*n2 + i] - 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
                uw[i] = u[j*n2 + i] + 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
                vw[i] = v[j*n2 + i];
            } //for i
            swater(uw, qw, vw, &d[j*n2], u1, q1, v1, &n2, h2, &grnd, &t);
            for(i=0; i<n2; i++){
                q[j*n2 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
                u[j*n2 + i] = (u1[i] + q1[i]) * .5f;
                v[j*n2 + i] = v1[i];
            } //for i
        } //  for j
        transpose(q, n2, n1); transpose(u, n2, n1); transpose(v, n2, n1);
        for(j=0; j<n2; j++){  // calculate along X
            for(i=0; i<n1; i++){
                qw[i] = v[j*n1 + i] - 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
                uw[i] = v[j*n1 + i] + 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
                vw[i] = u[j*n1 + i];
            } //for i
            swater(uw, qw, vw, &dt[j*n1], u1, q1, v1, &n1, h1, &grnd, &t);
            for(i=0; i<n1; i++){
                q[j*n1 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
                v[j*n1 + i] = (u1[i] + q1[i]) * .5;
                u[j*n1 + i] = v1[i];
            } //for i
        } //  for j
        transpose(q, n1, n2);  transpose(u, n1, n2);  transpose(v, n1, n2);
        // getting data from "sensors" and save result
    }
```

Перенос на GPU

```
for(int i=0; i<cfg.steps; i++){
    // calculate along X
    Invariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    RInvariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);

    // calculate along Y
    transpose<<<dimGrid, dimBlock>>>(d_qwdata, d_q1data, x_size, y_size);
    transpose<<<dimGrid, dimBlock>>>(d_uwdata, d_u1data, x_size, y_size);
    transpose<<<dimGrid, dimBlock>>>(d_vwdata, d_v1data, x_size, y_size);

    Invariants_Y<<<dimGrid_, dimBlock>>>(... , y_size, x_size);
    SWater_<<<dimGrid_, dimBlock>>>(..., y_size, x_size);
    RInvariants_Y<<<dimGrid_, dimBlock>>>(..., y_size, x_size);

    transpose<<<dimGrid_, dimBlock>>>(d_qwdata, d_q1data, y_size, x_size);
    transpose<<<dimGrid_, dimBlock>>>(d_uwdata, d_u1data, y_size, x_size);
    transpose<<<dimGrid_, dimBlock>>>(d_vwdata, d_v1data, y_size, x_size);

    // getting data from "sensors" and save result
    …
}
```

Перенос на GPU

```
for(int i=0; i<cfg.steps; i++){
    // calculate along X
    Invariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    RInvariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);

     // calculate along Y

     Invariants_Y<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_r<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    RInvariants_Y<<<dimGrid, dimBlock>>>(..., x_size, y_size);

     // getting data from "sensors" and save result
    …

}
```

Перенос на GPU

```
Inv1<<<dimGrid, dimBlock>>>(... , x_size, y_size);

for(int i=0; i<cfg.steps; i++){
    // calculate along X

    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    Inv2<<<dimGrid, dimBlock>>>(... , x_size, y_size);

     // calculate along Y
    SWater_r<<<dimGrid_, dimBlock>>>(..., x_size, y_size);
    Inv3<<<dimGrid, dimBlock>>>(..., x_size, y_size);

     // getting data from "sensors" and save result
    … RInv<<<dimGrid, dimBlock>>>(... , x_size, y_size);

}
```

Результаты профилирования

Method	gld_ incoherent	gld_ coherent	gst_ incoherent	gst_ coherent
SWater_	4.92214e+06	105505	5.24688e+06	48144
Inv2	2.62344e+06	10935	5.24688e+06	43740
SWater_r	4.72e+06	91004	5.24688e+06	47772
RInv	2.62344e+06	10935	1.74896e+06	14580
Inv3	2.60116e+06	10935	5.20233e+06	43740

Шаги по оптимизации

- Переход к текстурам – исключение лишних обращений к глобальной памяти и отказ от транспонирования;
- Выравнивание начала строк массивов;

Method	**gld_ incoherent**	**gld_ coherent**	**gst_ incoherent**	**gst_ coherent**
SWater_	0	459794	0	767260
Inv2	0	174897	0	699588
SWater_r	0	488749	0	761316
RInv	0	174900	0	233200
Inv3	0	174897	0	699588

Результат адаптации

- Исходная версия программы – 3 сек\итерацию
- На Tesla C1060 – 0,02 сек\итерацию

- Сравнение не корректное поскольку
  - Исходная программа не оптимизировалась и исполнялась последовательно
  - Для программы на GPU был изменен алгоритм вычислений (все вычисления в инвариантах и пр.)

CUDA и Фортран

- NVidia работает с Portland Group (PGI) по разработке поддержки CUDA в фортране
- На Международной конференции по Суперкомпьютерам в Дрездене (июнь 2009) была представлена спецификация поддержки CUDA в фортране. Обещана поддержка CUDA в PGI Fortan compiler в уже ноябре 2009. Уже есть бета-версия.
- NOAA разработала свой компилятор с фортрана - www-ad.fsl.noaa.gov/ac/Accelerators.html

CUDA и Фортран

- FLAGON: Fortran-9X Library for GPU Numerics
  - Поддержка Intel Compiler на Linux и Windows
  - sourceforge.net/apps/trac/flagon
- Поддержка фотрана от NVidia
  - www.nvidia.com/object/cuda_programming_tools.html