V МЕЖДУНАРОДНАЯ ЗАОЧНАЯ НАУЧНО-ПРАКТИЧЕСКАЯ КОНФЕРЕНЦИЯ

internauka.org
МЕЖДУНАРОДНЫЙ ЦЕНТР НАУКИ И ОБРАЗОВАНИЯ

Научная дискуссия: вопросы математики, физики, химии, биологии

Москва, 2013

2.3. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

OPENACC: ВЫЧИСЛЕНИЯ НА GPU С ПОМОЩЬЮ ПРОСТЫХ ДИРЕКТИВ

Муслимова Агима Зейнешовна

канд. пед. наук, доцент кафедры ИиМ КГУ им. А. Байтурсынова, г. Костанай

Нурмагамбетов Асхат Саятович

преподаватель кафедры ИиМ КГУ им. А. Байтурсынова, г. Костанай

В ноябре 2011 года был анонсирован стандарт OpenACC — совместное детище суперкомпьютерных гигантов CRAY, CAPS и PGI и лидера рынка графических процессоров NVIDIA. Сам стандарт призван значительно упростить работу программиста и создать высокоуровневую прослойку над уже известными CUDA и OpenCL.

Стоит отметить, что до недавнего времени стандарт не поддерживался в полной мере ни одним компилятором, но даже то, что уже есть, впечатляет своей простотой и результативностью. Теперь написание программы, выполняемой параллельно на тысячах ядер современных GPU не требует почти никаких усилий и практически полностью перекладывается на компилятор. Все что нужно сделать — расставить директивы по коду на манер OpenMP. Набор директив достаточно велик (полную спецификацию можно посмотреть по ссылке) и за один день его весь не освоить, но простейшую программу можно сделать за 5 минут, особенно если есть однопоточная реализация. Отсюда и вытекает основная идея — спрятать от разработчика почти все детали архитектуры, освободить его от тонкостей (а ведь лет шесть назад до появления CUDA использовать GPU могли только знатоки шейдеров) и оставить время на работу над научным или пользовательским проектом.

Как и его прародители (PGI accelerator и CAPS HMPP) OpenACC поддерживает языки C и Fortran [1]. Итак, все директивы в C-версии стандарта начинаются как обычно с #pragma, далее ставится спецификатор "acc" и одна из основных директив, дополненная одним,

или несколькими условиями. Чаще всего используются 3 директивы: parallel, kernels и data.

Как использовать:

Рассмотрим на простом примере как можно ускорить перемножение матриц:

```
#include <openacc.h>
#include <stdio.h>
#include <stdlib.h>
void main() {
 int n = 100;
 float a[n][n];
 float b[n][n];
 float c[n][n];
 float elements [n];
 for(int i = 0; i < n; i++)
  for (int j=0; j<n; j++){
   a[i][j] = i+j;
   b[i][j] = 100 + 2 * i;
  }
#pragma acc kernels loop independent
 for(int i = 0; i < n; i++)
   for (int j=0; j < n; j++){
      for (int k=0; k<n; k++)
            c[i][j]=+a[i][k]*b[k][j];
      }
 free(a); free(b); free(c);
} // main
```

Эта программа отличается от простой версии, выполняемой на одном ядре CPU только строкой 15, где мы видим директиву kernels, говорящую компилятору создать потоки, сгруппированные в несколько блоков, количество которых он выбирает на свое усмотрение. Кроме того, здесь же добавлена директива loop, после которой обязан начинаться цикл, loop служит для того, чтобы указать, как выполнять итерации цикла: independent — независимо, seq — последовательно.

Немного о директивах.

Рассмотрим краткое описание некоторых директив и условий к ним:

Директива parallel указывает на необходимость распараллеливания. Компилятор, проводя анализ кода, определяет необходимость исполнения различных его частей на GPU, или на хосте.

Директива kernels — аналог parallel, указывает на то, что для каждого нового цикла необходимо создать отдельную __device__ функцию [2].

Директива loop предшествует оператору цикла и используется для спецификации его свойств. Современные компиляторы не требуют её явного указания.

Несмотря на всю мощь компилятора, иногда нужно подсказывать, какие данные необходимо передать с хоста на устройство и обратно, а поскольку зачастую копирование выполняется дольше расчетов, нужно заранее продумать, где и как оптимизировать доступ к данным. Все условия передачи данных требуют входные данные, выглядящие следующим образом: a[start:length], где a — массив, или указатель на него, start — номер стартового элемента для копирования, а length –длина региона данных, копируемого на GPU, или с него; start и length указываются в элементах массива (для Fortran есть существенное отличие — вместо length указывается end — конечный элемент). Эти условия можно использовать только с директивами kernels, parallel и data region. Ниже представлены те из них, которые используются наиболее часто:

- copy — говорит компилятору скопировать данные на устройство перед выполнением ядра и назад после его завершения;
- copyin — указывает, что данные на GPU используются только для чтения, и нет необходимости копировать их обратно на хост;
- copyout — данные появятся только в результате выполнения ядра на GPU и никак не зависят от предыдущих значений по этому адресу, их нужно скопировать на хост после выполнения kernel;
- create — выделяет в памяти устройства место для данных, не требующих какого-либо копирования, например массив для хранения промежуточных результатов;
- present — подсказывает компилятору, что эти данные уже были переданы на устройство ранее. Вызывает ошибку, если данных на GPU нет.

Плюсы и минусы

Вот, как прост и неприхотлив в использовании OpenACC, как вы могли заметить, он очень сильно напоминает OpenMP (это точно неспроста) — он задумывался как ответвление. Значит, скоро можно будет легко распараллеливать свои задачи на огромных гетерогенных кластерах, почти не имея представления об их архитектуре. К плюсам можно отнести также высокую степень абстракции и кроссплатформенность — сразу после выхода новых архитектур

необязательно переписывать весь код, большую часть компилятор сделает за нас. К примеру, CAPS HMPP уже объявил о поддержке ускорителей не только NVIDIA, но и Intel MIC и даже AMD FirePro.

Плюсов и правда много, но не может же быть все так хорошо. Давайте обратимся к минусам: самое первое, что бросается в глаза — все компиляторы с поддержкой OpenACC стоят денег. Может для научных лабораторий лицензия и не такое уж дорогое удовольствие, но студенты вряд-ли соберутся потратиться на это. Второй минус — производительность: ни один компилятор не сможет оптимизировать код лучше, чем это можно сделать вручную, или с использованием библиотек от NVIDIA.

В заключение можно отметить, что OpenACC и правда дает возможность по-быстрому переписать свои проекты под использование GPU и практически не требует навыков их программирования. С его помощью уже ускорены десятки проектов в областях изучения и прогнозирования поведения атмосферы, газо- и гидродинамики и финансовых потоков. Пять лет назад началась революция массивно-параллельных вычислений и на сегодня OpenACC — лучший способ остаться на плаву, не потеряв позиции и не потратив сотни часов на изучение всех тонкостей CUDA или OpenCL.

Список литературы:

1. Антон Джораев. GPU NVidia: компьютерная графика и высокопроизводительные вычисления ObjectWatch, Inc. 2012 — [Электронный ресурс] — Режим доступа. — URL: http://old.kpfu.ru/inf/bin_files/dzhoraev-nvidia!28.pdf (дата обращения 25.04.2013).
2. Nguyen Minh Duc. Формула науки будущего. ObjectWatch, Inc. 2012 — [Электронный ресурс] — Режим доступа. — URL: http://supercomputers.ru/index.php?option=com_k2&view=item&id=403:%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0-%D0%.BD%D0%B0%D1%83%D0%BA%D0%B8-%D0%B1%D1%83%D0%B4%D1%83%D1%89%D0%B5%D0%B3%D0%BB (дата обращения 05.05.2013).