Стандарт директивного программирования OpenACC

Лекция 8

Спецификация v 1.0

Полная Спецификация OpenACC 1.0 доступна на сайте:

http://www.openacc-standard.org

Также доступна памятка по OpenACC

Пробная версия компилятора:

http://www.nvidia.com/object/openacc-gpu-directives.html

Онлайн курсы и вебинары:

http://www.nvidia.com/object/webinar.html

Пример SAXPY на C: OpenMP

Простота

Открытый стандарт

Высокая производительность

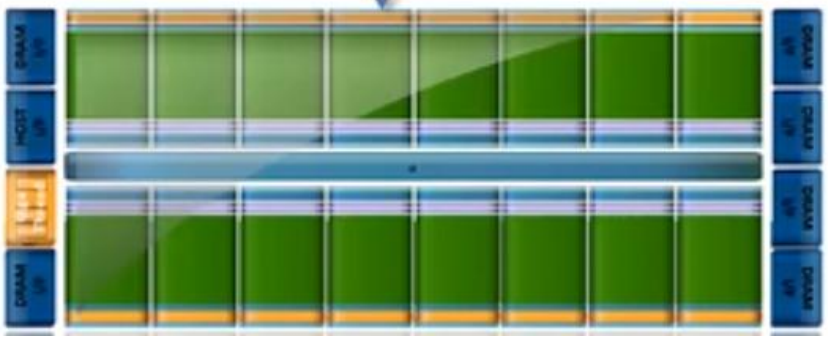
```
void saxpy(int n, float a, float *x,
           float *restrict y){
#pragma omp parallel
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Пример SAXPY на C: OpenACC

Простота

Открытый стандарт

Высокая производительность



```
void saxpy(int n, float a, float *x,
           float *restrict y){
#pragma acc parallel
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Модель исполнения OpenACC

CPU

- выполняет большую часть программы
- выделяет память на ускорителе
- инициирует копирование данных из памяти хоста в память ускорителя
- отправляет код ядра на ускоритель
- устанавливает ядра в очередь для исполнения на ускорителе
- ожидает выполнения ядра
- инициирует копирование данных из памяти ускорителя в память хоста
- освобождает память ускорителя

Ускоритель

- исполняет ядра
- одновременно может передавать данные между хостом и ускорителем

Модель исполнения OpenACC

Модель исполнения OpenACC имеет три уровня: **gang, worker, vector**

На архитектуру эта модель отображается, как набор обрабатываемых элементов (PEs)

Каждый PE содержит много рабочих и каждый рабочий может выполнять векторные инструкции

Для GPU в большинстве случаев отображение происходит так: gang=block, worker=warp, vector=threads-in-warp

Зависит от компилятора

Синтаксис директив

Fortran

!$acc *directive [атрибут [, атрибут] …]*

структурированный блок

!$acc end *directive*

C

#pragma acc *directive [атрибут [, атрибут] …]*

структурированный блок

Компиляция (с помощью компилятора PGI)

pgcc –acc -ta=nvidia,time –Minfo=accel <filename>

pgfortran -acc -ta=nvidia,time –Minfo=accel <filename>

Конструкция Parallel

Fortran

```
!$acc parallel [атрибут [, атрибут]…]
        структурированный блок
!$acc end parallel
```

C

```
#pragma acc parallel [атрибут [, атрибут]…]
        структурированный блок
```

Атрибуты конструкции Parallel

Основные атрибуты

- if (condition)
- async [(exp)]
- num_gangs (exp)
- num_workers (exp)
- vector_length(exp)
- reduction(operator:list)

атрибуты данных

- copy*(list)
- create(list)
- present(list)
- present_or_copy*(list)
- present_or_create(list)
- deviceptr(list)
- private(list)
- firstprivate(list)

*<blank>|in|out

Конструкция Kernels

Fortran

```
!$acc acc kernels [атрибут [, атрибут]…]
        структурированный блок
!$acc end kernels
```

C

```
#pragma acc kernels [атрибут [, атрибут]…]
        структурированный блок
```

Конструкция Kernels

```
!$acc kernels
  do i = 1,n
  do j = 1,n                    ядро 1
     a(i,j) = 0.0
  enddo
  enddo
  do k = 1,n
     b(k) = 1.0                 ядро 2
  enddo
!$acc end kernels
```

Конструкция Loop

Fortran

```
!$acc loop [атрибут [, атрибут]…]
        do loop
```

C

```
#pragma acc loop [атрибут [, атрибут]…]
        for loop
```

Атрибуты

- collapse(n)
- gang[(exp)]
- worker[(exp)]
- vector[(exp)]
- seq
- independent
- private(list)
- reduction(op:list)

Атрибуты для области Data

Fortran

Синтаксис: array (начало : конец [, н:к] …)

Примеры: a(:, :), a(1:100, 2:n)

C

Синтаксис: array[начало : длина]

Примеры: a[2:n] // это значит a[2], a[3], …, a[2+n-1]

Использование региона Data

```
int a [1000];
int b [1000];

#pragma acc parallel
for (int i=0; i<1000; i++)
{
    a[i] = i - 100 + 23;
}

#pragma acc parallel
for (int j=0; j<1000; j++)
{
    b[j] = a[j] - j - 10 + 213;
}
```

Директивы ***parallel*** размещены отдельно и потребуют генерации ядер с отдельным копированием данных

Использование региона Data

```
int a [1000];
int b [1000];
#pragma acc data copyout (a[0:1000],b[0:1000])
{
#pragma acc parallel
for (int i=0; i<1000; i++)
{
    a[i]=i-100+23;
}

#pragma acc parallel
for (int j=0; j<1000; j++)
{
    b[j]=a[j]-j-10+213;
}
}
```

Директивы ***parallel*** размещены в рамках объемлющего региона ***data*** и потребуют генерации отдельных ядер, однако данные копируются один раз

Остальные директивы

host_data

Делает адрес данных на ускорителе доступным для хоста

cache

Кэширует данные через программно-управляемый кэш. (CUDA разделяемая память)

update

Обновляет существующие данные после их изменения

wait

Ожидает выполнения асинхронных операций на ускорителе

declare

Указывает, что необходимо выделить память на ускорителе для использования в рамках региона data