NVIDIA CUDA И OPENACC
ЛЕКЦИЯ 5

Перепёлкин Евгений

СОДЕРЖАНИЕ

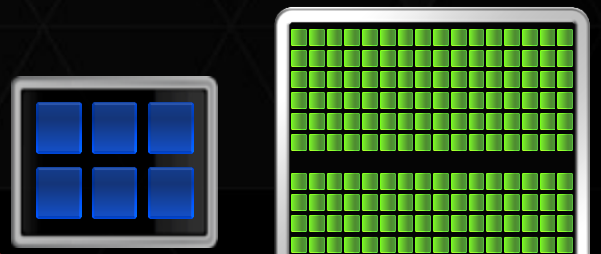
Лекция 5

- Введение в OpenACC
- Основные директивы OpenACC
  - parallel, kernels, loop, data
- Примеры – сравнения (OpenMP, OpenACC, CUDA)
  - Обработка двух массивов
  - Перемножение матриц
  - Задача N-тел

Введение в OpenACC

OPENACC

директивы для ускорителей




```
main() {
  …
  <serial code>
  …
  #pragma acc directive
  {
  <compute intensive code>
  }
  …
}
```

Выделение части кода, выполняемого на GPU

- CRAY, CAPS, PGI, NVIDIA
- Открытый стандарт
- Простота
- Возможность получить большую производительность
- Возможно использование языков C и Fortran

С ЧЕГО НАЧАТЬ ?

Полезные ссылки

- «Online-курсы» и «Webinar»
  - http://www.nvidia.com/object/tesla-gpu-computing-webinars-ru.html
- Пробная версия компилятора
  - http://www.nvidia.com/object/openacc-gpu-directives.html
- Набор директив OpenACC
  - http://openacc.org

Основные директивы OpenACC

ОСНОВНЫЕ ДИРЕКТИВЫ OPENACC

#pragma acc <директива> атрибуты

- **parallel**
  - основные атрибуты: `if, async, num_gangs, num_worker, vector_length, private, first_private, reduction`
  - атрибуты для данных: `copy, copyin, copyout, create, present, present_or_copy, present_or_create, deviceptr, private, firstprivate`
- **kernels**
  - основные атрибуты: `if, async`
  - атрибуты для данных: `copy, copyin, copyout, create, present, present_or_copy, present_or_create, deviceptr, private, firstprivate`

ОСНОВНЫЕ ДИРЕКТИВЫ OPENACC

#pragma acc <директива> атрибуты

- `loop`
  - атрибуты: `collapse, gang, worker, vector, seq, independent, private, reduction`
- `data`
  - атрибут: `array [начало:дина]`
- Комбинированные директивы
  - `parallel loop`
  - `kernels loop`

КОМПИЛЯЦИЯ

PGI компилятор языка «C»

```
pgcc -acc -ta=tesla:cuda6.0,time -Minfo=accel example.c
```

`-acc` – использование OpenACC директив

`-ta` – целевая архитектура (Tesla, CUDA 6.0),
time - время (выполнения ядер, копирования данных)

`-Minfo` – информация по оптимизации
(accel - по ускорению)

`pgcc -help|more` – информация по остальным опциям

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «parallel»

```
   #include <openacc.h>
   ...
30 #pragma acc parallel
31 {
32  for ( int i = 0; i < N; i++ ) a[i] = sinf ( i );
33  for ( int j = 0; j < N; j++ ) b[j] = cosf ( j );
34 }
```

```
30, Accelerator kernel generated
    32, #pragma acc loop vector(256) /* threadIdx.x */
    33, #pragma acc loop vector(256) /* threadIdx.x */
30, Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
    Generating Tesla code
32, Loop is parallelizable
33, Loop is parallelizable
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «kernels»

```
30 #pragma acc kernels
31 {
32  for ( int i = 0; i < N; i++ ) a[i] = sinf ( i );
33  for ( int j = 0; j < N; j++ ) b[j] = cosf ( j );
34 }
```

```
30, Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
    Generating Tesla code
32, Loop is parallelizable
    Accelerator kernel generated
    32, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
33, Loop is parallelizable
    Accelerator kernel generated
    33, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Две области «parallel»

```
    float a[N], b[N];
    ...
26  #pragma acc parallel
27  for ( int i = 0; i < N; i++ ) a[i] = sinf ( i );
28
29  #pragma acc parallel
30  for ( int j = 0; j < N; j++ ) b[j] = cosf ( a[j] );
```

Массив a[ ] используется в обеих областях

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Компиляция

```
26, Accelerator kernel generated
    27, #pragma acc loop vector(256) /* threadIdx.x */
26, Generating present_or_copyout(a[:100000])
    Generating Tesla code
27, Loop is parallelizable
29, Accelerator kernel generated
    30, #pragma acc loop vector(256) /* threadIdx.x */
29, Generating present_or_copyout(b[:100000])
    Generating present_or_copyin(a[:100000])
    Generating Tesla code
30, Loop is parallelizable
```

Дополнительное копирование массива a[ ]

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «data»

```
   float a[N],b[N];
   ...
24 #pragma acc data copyout (a[0:N], b[0:N])
25 {
26  #pragma acc parallel
27  for ( int i = 0; i < N; i++ ) a[i] = sinf ( i );
28
29  #pragma acc parallel
30  for ( int j = 0; j < N; j++ ) b[j] = cosf ( a[j] );
31 }
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Компиляция. Директива «data»

```
24, Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
26, Accelerator kernel generated
    27, #pragma acc loop vector(256) /* threadIdx.x */
26, Generating Tesla code
27, Loop is parallelizable
29, Accelerator kernel generated
    30, #pragma acc loop vector(256) /* threadIdx.x */
29, Generating Tesla code
30, Loop is parallelizable
```

Нет дополнительного копирования массива a[ ]

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «loop» [«independent»]

```
   float *a = (float*) malloc ( sizeof (float) * N * N );
   ...
30 #pragma acc kernels
31 {
32  #pragma acc loop independent
33  for ( int i = 0; i < N; i++ )
34  {
35   #pragma acc loop independent
36   for ( int j = 0; j < N; j++ ) a[j + i * N] = cosf ( i + j );
37  }
38 }
```

ПРОБЛЕМА при компиляции

```
36 Accelerator restriction: size of GPU copy of ‘a’ is unknown
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «loop» [«independent»]

```
   float *a = (float*) malloc ( sizeof (float) * N * N );
   ...
30 #pragma acc data copyout (a[0:N*N])
31 {
32  #pragma acc kernels
33  {
34   #pragma acc loop independent
35   for ( int i = 0; i < N; i++ )
36   {
37    #pragma acc loop independent
38    for ( int j = 0; j < N; j++ ) a[j + i * N] = cosf ( i + j );
39   } // for i
40  } // pragma kernels
41 } // pragma data
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Компиляция. Директива «loop» [«independent»]

```
30, Generating copyout(a[:N*N])
32, Generating Tesla code
35, Loop is parallelizable
38, Loop is parallelizable
    Accelerator kernel generated
    35, #pragma acc loop gang /* blockIdx.y */
    38, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Директива «loop» [«independent», «seq»]

```
   float *a = (float*) malloc ( sizeof (float) * N * N );
   ...
30 #pragma acc data copyout (a[0:N*N])
31 {
32  #pragma acc kernels
33  {
34   #pragma acc loop independent
35   for ( int i = 0; i < N; i++ )
36   {
37    #pragma acc loop seq
38    for ( int j = 0; j < N; j++ ) a[j + i * N] = cosf ( i + j );
39   } // for i
40  } // pragma kernels
41 } // pragma data
```

ШАБЛОНЫ РАБОТЫ С ДИРЕКТИВАМИ

Компиляция. Директива «loop» [«independent»,«seq»]

```
30, Generating copyout(a[:N*N])
32, Generating Tesla code
35, Loop is parallelizable
38, Loop is parallelizable
    Accelerator kernel generated
    35, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Примеры-сравнения (OpenMP, OpenACC, CUDA)

ОБРАБОТКА ДВУХ МАССИВОВ

Пример 3 из лекции 2

$$c_i = \sum_{j=0}^{99} sin(a_i b_i + j),$$

$$a_i = sin(i), b_i = cos(2i - 5),$$

$$i = 0, \ldots, N - 1$$

N = 512 * 50 000

ОБРАБОТКА ДВУХ МАССИВОВ

Реализация на OpenMP

```
#pragma omp parallel for shared (hA, hB, hC) private (i, j, sum)
for ( i = 0; i < N; i++ )
{sum = 0.f; ab = hA[i] * hB[i];
 for ( j = 0; j < 100; j++) sum += sinf (j + ab);
 hC[i] = sum;
}
```

```
CPU calculation time        : 9516  ms
CPU-OpenMP calculation time: 2402  ms
```

ОБРАБОТКА ДВУХ МАССИВОВ

Реализация на OpenACC

```
void function (float *restrict hC, float *hA, float *hB, int size)
{
 #pragma acc kernels loop present (hC, hA, hB)
 for ( i = 0; i < N; i++ )
 {sum = 0.f; ab = hA[i] * hB[i];
  for ( j = 0; j < 100; j++) sum += sinf (j + ab);
  hC[i] = sum;
 }
}
```

ОБРАБОТКА ДВУХ МАССИВОВ

Реализация на OpenACC

```
int main ()
{ ...
 #pragma acc data copyin (hA[0:size], hB[0:size]) copyout (hC[0:sise])
 {
  function (hC, hA, hB, size);
 }
 ...
```

```
GPU-OpenACC calculation time : 541 ms
GPU-CUDA calculation time    : 63  ms
```

ОБРАБОТКА ДВУХ МАССИВОВ

CPU Core2 Quad Q8300 2.5 ГГц (ICC, PGI) x64 GPU Tesla K40c CUDA 6.0

```
CPU calculation time*          : 9516  ms
CPU-OpenMP calculation time*   : 2402  ms
GPU-OpenACC calculation time** : 541 ms
GPU-CUDA calculation time***   : 63  ms
```

---

* компилятор ICC

** компилятор PGI

*** компилятор NVCC, вариант с 4-мя CUDA-потоками

$$C = AB,$$

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} b_{k,j},$$

$$a_{i,j} = 2j + i, \qquad b_{i,j} = j - i,$$

$$i, j = 0, \ldots, N - 1$$

N x N = 2048 x 2048

ПРИМЕР ПЕРЕМНОЖЕНИЯ МАТРИЦ (ЛЕКЦИЯ 5)

ПЕРЕМНОЖЕНИЕ МАТРИЦ

Реализация на OpenMP

```
#pragma omp parallel for shared (a, bT, c) private (n, m, k)
for ( n = 0; n < N; n++ )
{for ( m = 0; m < N; m++ )
 {sum = 0.f;
  for ( k = 0; k < N; k++) sum += a[k + n * N] * bT[k + m * N];
  c[m + n * N] = sum;
 }
}
```

```
CPU calculation time        : 4622  ms
CPU-OpenMP calculation time: 3712  ms
```

ПЕРЕМНОЖЕНИЕ МАТРИЦ

Реализация на OpenACC

```
void Matrix_Mul ( float *restrict c, float *a, float *bT, int N )
{
 #pragma acc parallel loop present (c, a, bT)
 for ( int n = 0; n < N; n++ )
 {for ( int m = 0; m < N; m++ )
  {float sum = 0.f;
   for ( int k = 0; k < N; k++) sum += a[k + n * N] * bT[k + m * N];
   c[m + n * N] = sum;
  }
 }
}
```

ПЕРЕМНОЖЕНИЕ МАТРИЦ

Реализация на OpenACC

```
int main ( )
{...
 #pragma acc data copyin (a[0:N], b[0:N]) copyout (c[0:N])
 {
  Matrix_Mul (c, a, bT, N);
 }
 ...
```

```
GPU-OpenACC calculation time : 655 ms
GPU-CUDA calculation time    : 26  ms
```

ПЕРЕМНОЖЕНИЕ МАТРИЦ

CPU Core2 Quad Q8300 2.5 ГГц (ICC, PGI) x64 GPU Tesla K40c CUDA 6.0

```
Precision                      : float
CPU calculation time*          : 4622  ms
CPU-OpenMP calculation time*   : 3712  ms
GPU-OpenACC calculation time** : 655 ms
GPU-CUDA calculation time***   : 26  ms
```

---

* компилятор ICC

** компилятор PGI

*** компилятор NVCC, вариант SMEM-5

$$\vec{a}_{n,i} = \frac{\vec{F}_{n,i}}{m} = Gm \sum_{k \neq n}^{N-1} \frac{\vec{r}_{k,i} - \vec{r}_{n,i}}{\left|\vec{r}_{k,i} - \vec{r}_{n,i}\right|^3},$$

$$\vec{v}_{n,i+1} = \vec{v}_{n,i} + \vec{a}_{n,i}\tau,$$

$$\vec{r}_{n,i+1} = \vec{r}_{n,i} + \vec{v}_{n,i}\tau + \vec{a}_{n,i}\frac{\tau^2}{2},$$

$$t_i = t_0 + i\tau,$$

$$\left|\vec{r}_{k,i} - \vec{r}_{n,i}\right| < 0.01\text{м}, \vec{F}_{n,i} = 0,$$

$$mG = 10\,\text{Нм}^2/\text{кг}, \tau = 0.001c$$

ЗАДАЧА N-ТЕЛ (ЛЕКЦИЯ 5)

ЗАДАЧА N-ТЕЛ

Реализация на OpenMP

```
#pragma omp parallel for shared (hX, hY, hVX, hVY, hAX, hAY)
                         private (i, j)
for ( id = 0; id < N; id++ )
{
 Acceleration_CPU (hX, hY, hAX, hAY, j, N, id);
 Position_CPU (hX, hY, hVX, hVY, hAX, hAY, tau, j, N, id);;
}
```

```
CPU calculation time        : 13821 ms
CPU-OpenMP calculation time: 3525  ms
```

ЗАДАЧА N-ТЕЛ

Реализация на OpenACC

```
void Acc_Pos ( float *restrict hX, float *restrict hY,
               float *hVX, float *hVY, float tau, int nt, int N )
{float ax, ay, xx, yy, rr, tau2 = tau * tau * 0.5f; int j, sh = (nt - 1) * N;
 #pragma acc kernels present (hX, hY, hVX, hVY)
 {
  #pragma acc loop independent vector(256) gang(N/256)
  for ( int id = 0; j < N; j++ )
  {ax = 0.f; ay = 0.f;
   #pragma acc loop seq
   for ( j = 0; j < N; j++ ) { ... } // вычисление ax, ax
   X[id + nt * Np] = X[id + sh] + VX[id] * tau + ax * tau2;
   Y[id + nt * Np] = Y[id + sh] + VY[id] * tau + ay * tau2;
   VX[id] += ax * tau; VY[id] += ay * tau;
  } // id
 } // kernels
}
```

ЗАДАЧА N-ТЕЛ

Реализация на OpenACC

```
int main ( )
{  ...
 #pragma acc data copyin (hVX[0:N], hVY[0:N]) copy (hX[0:N], hY[0:N])
 {
  for ( j = 1; j < NT; j++ )  Acc_Pos (hX, hY, hVX, hVY, tau, j, N );
 }
 ...
```

```
GPU-OpenACC calculation time : 566 ms
GPU-CUDA calculation time    : 215 ms
```

ЗАДАЧА N-ТЕЛ

CPU Core2 Quad Q8300 2.5 ГГц (ICC, PGI) x64 GPU Tesla K40c CUDA 6.0

```
Number of particles              : 20480
CPU calculation time*            : 13821 ms
CPU-OpenMP calculation time*     : 3525  ms
GPU-OpenACC calculation time**   : 566 ms
GPU-CUDA calculation time***     : 215 ms
```

---

* компилятор ICC

** компилятор PGI

*** компилятор NVCC, вариант с разделяемой памятью