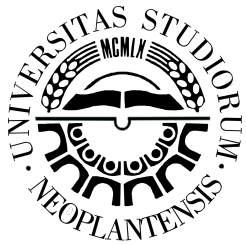
OpenACC

Рачунарски системи високих перформанси

Горана Гојић Вељко Петровић

Факултет техничких наука
Универзитет у Новом Саду

Рачунарске вежбе, Зимски семестар 2018/2019.



Шта је OpenACC

API за пребацивање извршавања делова кода на акцелератор.
Обухвата:

- директиве (`pragma acc <direktiva>`)
- промењљиве окружења (енг. *runtime environment variables*)
- библиотечке рутине (енг. *library routines*)

Подршка за **C**, **C++** и **Fortran**.

Концептуално веома сличан OpenMP.

Циљна архитектура

Акцелератори.

У лабораторији ћемо као акцелератор користити *Nvidia Quadro* графичке картице.

OpenACC компајлери

Commercial Compilers

CRAY THE SUPERCOMPUTER COMPANY

Contact Cray Inc for more information.

PGI Community EDITION

Annual license. Free download.

国家超级计算无锡中心 National Supercomputing Center in Wuxi

Contact National Supercomputing Center in Wuxi for more information.

Open Source Compilers

GCC 7
Includes initial support for OpenACC 2.5

Academic Compilers

RIKEN 筑波大学 University of Tsukuba

Omni compiler project, RIKEN/University of Tsukuba

OAK RIDGE National Laboratory

OpenARC, Oak Ridge National Laboratory

UNIVERSITY OF HOUSTON
Stony Brook University

OpenUH, University of Houston, Stony Brook University

Lawrence Livermore National Laboratory
UNIVERSITY OF DELAWARE

ROSEACC, LLNL/University of Delaware

На вежбама ћемо користити GNU GCC компајлер.

[1] Извор OpenACC званична страна

Компајлирање OpenACC програма

Позиционирати се у директоријум у којем се налази датотека са OpenACC кодом и у терминалу унети:

```
gcc -o izvrsna_dat izvorna_dat.c -fopenacc
```

За покретање програма, позиционирати се у директоријум у којем је извршна датотека и унети `./izvrsna_dat`.

Уколико програм позива неку од OpenACC функција (назив почиње са `acc_`), у изворну датотеку је потребно додати и

```
#include <openacc.h>
```

.

Терминологија

Домаћин (енг. *Host*) - централна процесна јединица са својом хијерархијом меморије.

Акселератор (енг. *Accelerator* или *Device*) - акцелераторски уређај, нпр. графичка картица.

Паралелни регион - Део кода обележен за извршавање на акцелераторском уређају са придруженим структурама података. Обухвата регионе кода обележене `parallel` и `kernels` директивама (другачије ће се звати и рачунски региони).

OpenACC модел извршавања

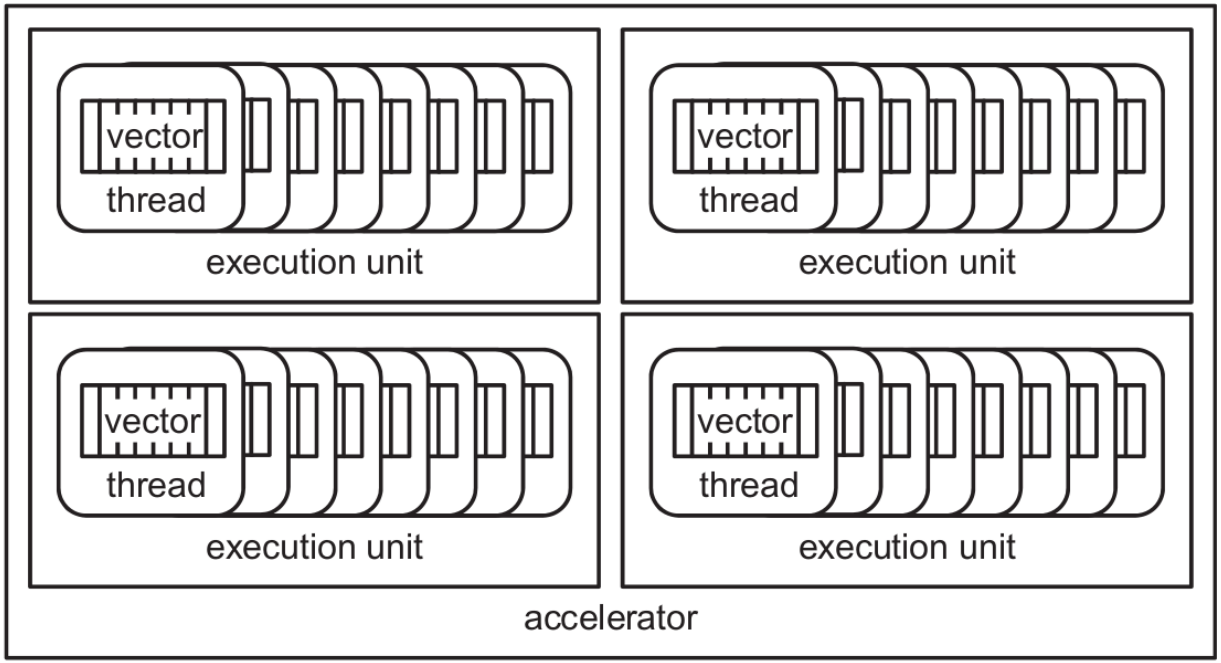


Figure: Концептуална архитектура акселератора

OpenACC подржава три нивоа паралелизма: *gang*, *worker*, *vector*

OpenACC модел извршавања

Општи формат OpenACC директиве:

```
#pragma acc <directive> [clause-list] new-line
structured block
```

Директиве за обележавање паралелног кода:

- `kernels`
- `parallel`

kernels директива

Означава део кода који може бити преведен за извршавање на акцелератору прављењем једног или више кернела. Компајлер одлучује **шта** ће и **како** ће паралелизовати.

```
#pragma acc kernels [clause-list] new-line
structured block
```

Неке од клаузула (параметри нису наведени):

- `async`
- `wait`
- `copy`
- `copyin`
- `copyout`
- `...`

Пример 1: kernel.c

```
int main() {

  /* ... */

  #pragma acc kernels
  {
    for(i = 0; i < MATRIX_SIZE; i++)
      for(j = 0; j < MATRIX_SIZE; j++)
        randomMatrix[i * MATRIX_SIZE + j] =
        randomMatrix[i * MATRIX_SIZE + j] * 2;
  }

  return 0;
}
```

Пример 2: ptraliasing.c

```
void assign(int *a, int *b, int size) {
    #pragma acc kernels
    {
        for (int i = 0; i < size - 1; i++)
            a[i] = b[i + 1];
    }
}
```

Питање: Када се користи `kernels` директива, компајлер проналази делове кода који су безбедни за паралелизацију, односно, у којима нема зависности међу подацима. Шта мислите, да ли би OpenACC компајлер превео ово у код за паралелно извршавање?

parallel директива

Означава део кода који ће бити преведен за извршавање на акцелератору. Компајлер одређује **како** ће сегмент кода бити паралелизован. Подразумевано извршавање креће у *gang-redundant* режиму.

```
#pragma acc parallel [clause-list] new-line
structured block
```

Неке од клаузула (параметри нису наведени):

- `async`
- `wait`
- `num_gangs`
- `num_workers`
- `vector_length`
- `reduction`
- `copy`
- `copyin`
- `copyout`
- `...`

Пример 3: parallel.c

```c
#include <openacc.h>

int main() {
  float *values = (float *) malloc(sizeof(float) * size);

  #pragma acc parallel
  for (int i = 0; i < 1024; i++)
    values[i] = 1.f;
  free(values);

  return 0;
}
```

Питање: Колико пута ће сваком пољу `value` низа бити додељена вредност?

`parallel` клаузуле

- **async** - Нит која је наишла на `parallel` или `kernels` директиву (локална нит) може да настави извршавање кода који следи иза паралелног региона без чекања да уређај заврши свој посао.
- **wait** - Када наиђе на ову клаузулу, локална нит чека.
- **num_gangs(int-exp)** - Број *gang*-ова који извршавају паралелни регион.
- **num_workers(int-exp)** - Број радника унутар *gang*-а.
- **vector_length(int-exp)** - Дужина вектора која се користи за SIMD операције.
- **reduction(op:var-list)** - Прави локалну копију променљиве и иницијализује је. На крају паралелног региона се локалне копије редукују.
- **copy, copyin, copyout**

loop директива

Даје компајлеру додатне информације о томе како да паралелизује петљу на коју се односи директива. Може бити унутар `parallel` или `kernels` директива.

```
#pragma acc loop [clause-list] new-line
        for loop
```

Комбиноване конструкције:

```
#pragma acc parallel loop [clause-list] new-line
        for loop
```

```
#pragma acc kernels loop [clause-list] new-line
        for loop
```

loop клаузуле

Клаузуле за оптимизацију извршавања:

- **gang** - Партиционише итерације између *gang*-ова. Преводи извршавање из *gang-redundant* у *gang-partitioned* режим. У случају угњеждених петљи, спољна мора бити *gang* петља.
- **worker** - Патриционише итерације између *worker*-а. Преводи извршавање из worker-single у worker-partitioned режим. У случају угњеждених петљи, било која унутрашња петља, осим оне најугњежденије, може бити *worker* петља.
- **vector** - Векторизација петље (*SIMD* или *SIMT*). У случају угњеждених петљи, најугњежденија је *vector* петља.
- **seq** - Петља ће се извршити секвенцијално без обзира на потенцијалне оптимизације које је компајлер пронашао. У сучају угњеждених петљи, може бити на било ком нивоу.

Оптимизацијом петљи се губи на портабилности програма.

loop клаузуле

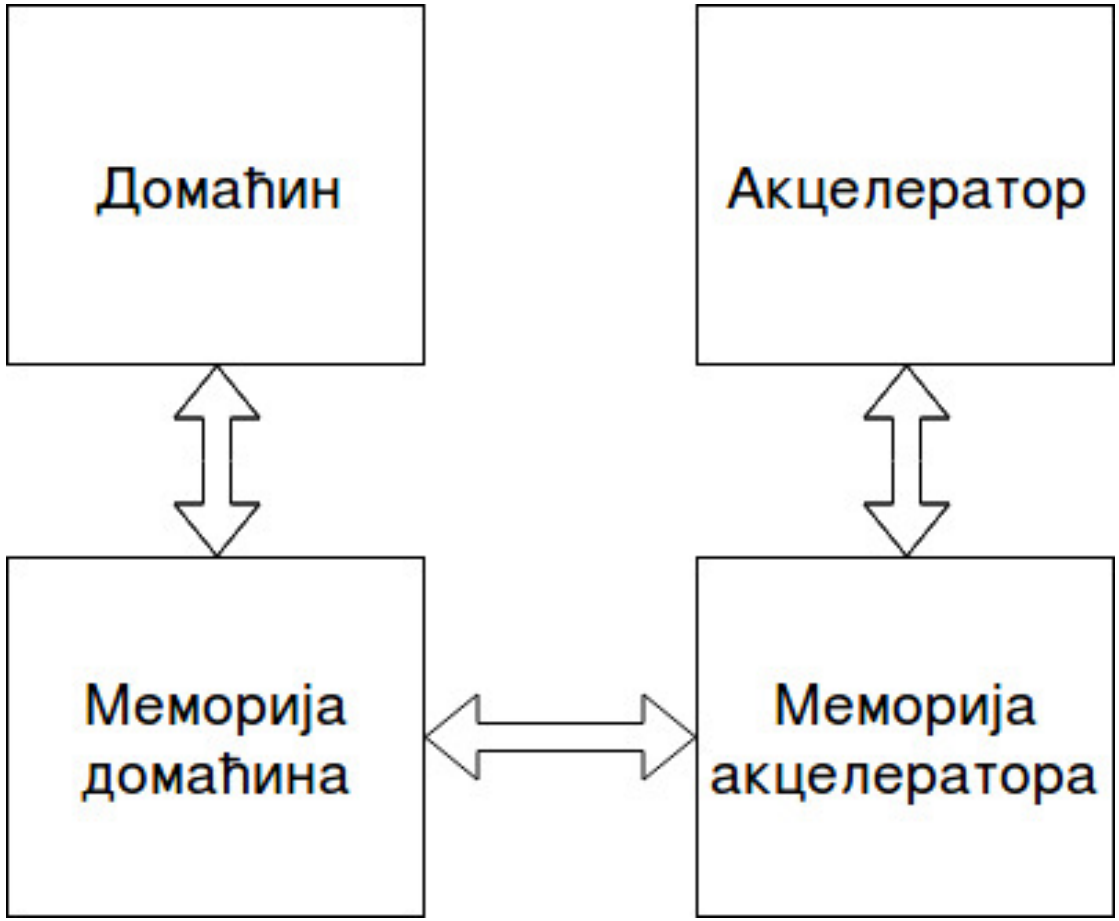
Остале клаузуле:

- **private**
- **reduction**
- **independent** - Сигнализира компајлеру да нема зависности података између итерација петље.
- ...

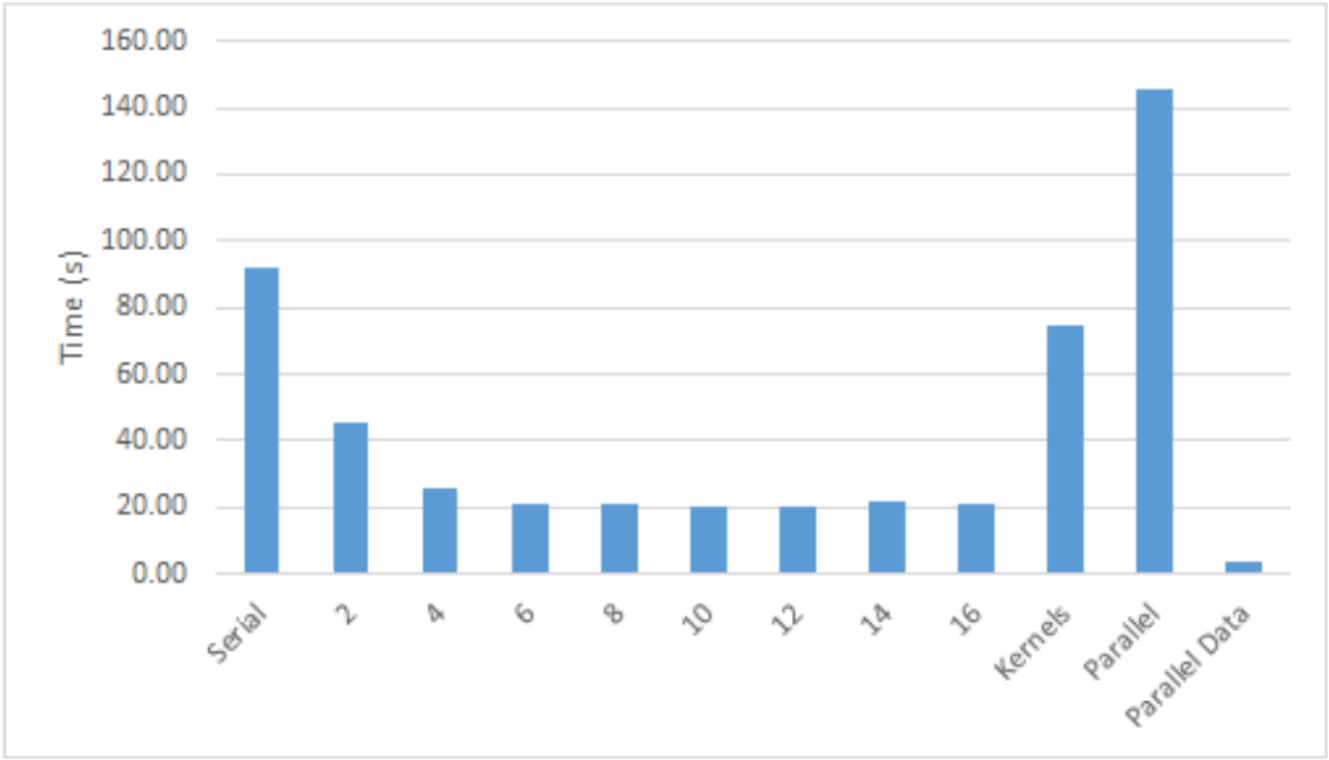
Пример 4: parallelloop.c

```
int main() {

  /* ... */

  #pragma acc parallel loop gang
  for (int i=0; i<N; i++)
      #pragma acc loop vector
      for (int j=0; j<M; j++)
        A[i * N + j] = 1.f;

  return 0;
}
```

Модел података



Утицај модела података на перформансе



Оптимизација кернела без оптимизације преноса података обично не води побољшању перформанси извршавања програма!

[1] Извор: OpenACC Programming and Best Practices Guide

Управљање подацима

Пренос података између домаћина и уређаја је могуће контролисати на више начина:

- Без било каквог специфицирања понашања, компајлер ће сам одредити када и које податке треба пренети са домаћина на уређај или обрнуто.
- Модификовањем понашања `parallel` и `kernels` одговарајућим клаузулама за рад са подацима (нпр. `copy`, `copyin`, `copyout`...). Модификације важе за паралелни регион над којим су примењене клаузуле за модфикацију.
- Коришћењем `data` директиве.
- Коришћењем `data enter` и `data exit` директива (углавном за објектно програмирање).

data директива

Дефинише блок кода у којем се клаузулама контролише пренос података на релацији домаћин уређај.

```
#pragma acc data [clause-list] new-line
structured block
```

- **copy**
- **copyin**
- **copyout**
- **create**
- **present**

Пример 5: data.c

```
int main() {
  /* ... */
  #pragma acc data
  {
    #pragma acc parallel loop
    for (int i=0; i<N; i++) {
      y[i] = 0.0f;
      x[i] = (float)(i+1);
    }
    #pragma acc parallel loop
    for (int i=0; i<N; i++) {
      y[i] = 2.0f * x[i] + y[i];
    }
  }
  /* ... */
}
```

Клаузуле

- **copy(var-list)** - Алоцира простор за променљиве на уређају, копира вредности са домаћина на уређај и са уређаја назад на домаћина када се заврши блок података (или `parallel` и `kernels` блок ако клаузула стоји уз њих)
- **copyin(var-list)** - Алоцира простор за променљиве на уређају и копира вредности са домаћина на уређај. По завршетку блока не преноси вредности назад на домаћина.
- **copyout(var-list)** - Алоцира простор за променљиве на уређају без иницијализације подацима са домаћина. По завршетку блока, подаци се преносе са уређаја на домаћина.
- **create(var-list)** - Алоцира простор за променљиве на уређају без иницијализације. Вредности променљивих се на крају блока не пребацују на домаћина.
- **present(var-list)** - Означава да су променљиве присутне у меморији уређаја.

OpenACC променљиве окружења

GNU GCC компајлер:

- `ACC_DEVICE_TYPE` (стандард)
- `ACC_DEVICE_NUM` (стандард)
- `ACC_PROF` (стандард)
- `GOPMP_OPENACC_DIM` (gcc)
- `GOMP_DEBUG` (gcc)

Задатак 1: Рачунање броја π

Модификовати дати секвенцијални програм за рачунање вредности броја π коришћењем OpenACC директива. Користити NVIDIA графичку картицу као акцелератор (GNU GCC тренутно не подржава Radeon картице). Мерити извршавање секвенцијалног и имплементираног убрзаног програма.

Напомена: Задатак свакако имплементирати и у случају да на рачунару немате доступну NVIDIA графичку картицу.

Задатак 2: Рачунање Јакобијана

Модификовати дати секвенцијални програм за рачунање Јакобијана. Основну верзију програма у C++ програмском језику скинути са Гитхаб налога OpenACCUserGroup.Користити NVIDIA графичку картицу као акцелератор (GNU GCC тренутно не подржава Radeon картице). Мерити извршавање секвенцијалног и имплементираног убрзаног програма.

Напомена: Задатак свакако имплементирати и у случају да на рачунару немате доступну NVIDIA графичку картицу.

Савети за инкрементално портовање секвенцијалног у OpenACC код

1. Идентификовати паралелизам петљи. Паралелизовати део по део секвенцијалног кода при том контролишући коректност извршавања програма. Наставити са овим типом оптимизације без обзира на то да ли се време извршавања повећава.
2. Оптимизовати пренос података. Преклопити пренос података са рачунањем, уклонити непотребна копирања, ажурирања променљивих, итд...
3. Паралелизоване петље оптимизовати за циљну архитектуру коришћењем клаузула.

[1] Извор: OpenACC Programming and Best Practices Guide

Литература

Текстуални материјали:

- OpenACC Programming and Best Practices Guide
- OpenACC Specification 2.5
- GNU GCC OpenACC Wiki
- David B. Kirk, Wen-mei W. Hwu, *Programming Massively Parallel Processors*, A Hands on Approach, 2nd edition, 2012

Видео туторијали:

- Introduction to Parallel Programming with OpenACC
- Advanced OpenACC