OpenACC — део 2

Рачунарски системи високих перформанси

Петар Трифуновић Вељко Петровић

Факултет техничких наука
Универзитет у Новом Саду

Рачунарске вежбе, Зимски семестар 2022/2023.

OpenACC променљиве окружења

- GNU GCC компајлер:
  - `ACC_DEVICE_TYPE` (стандард) — на који тип уређаја се повезати; може се прегазити функцијом *acc_set_device_type*
  - `ACC_DEVICE_NUM` (стандард) — број уређаја на који се треба повезати; може се прегазити функциојм *acc_set_device_num*
  - `ACC_PROFLIB` (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

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

[1] Извор: OpenACC Programming and Best Practices Guide

OpenACC и CUDA

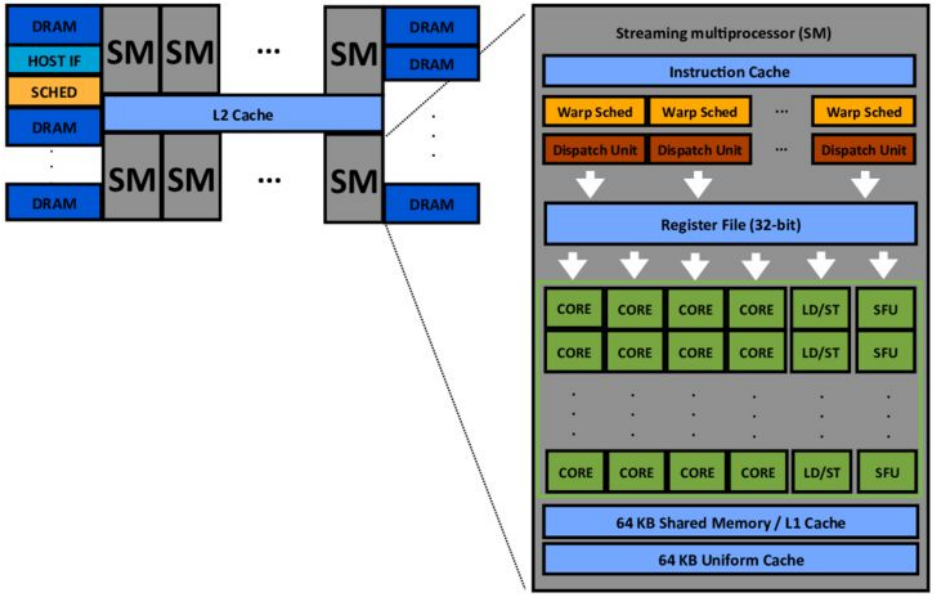
CUDA понављање

- **CUDA** (енг. *Compute Unified Device Architecture*) — платрофма за паралелно програмирање на NVIDIA графичким картицама.
- Мапирање OpenACC на CUDA термине:

OpenACC	**CUDA**
Домаћин (енг. *Host*)	Домаћин (енг. *Host*)
Акцелератор (енг. *Accelerator*)	Уређај (енг. *Device*)
Паралелни или рачунски регион	Један или више CUDA кернела

- У OpenACC код је на вишем нивоу апстракције у односу на CUDA код и може се превести за извршавање на GPU.

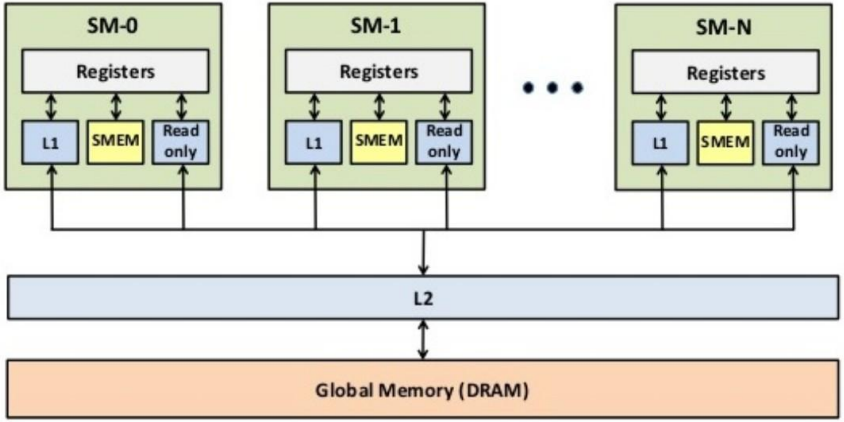
Архитектура CUDA графичке картице



- Streaming Multiprocessors (SM)
- Куда језгра

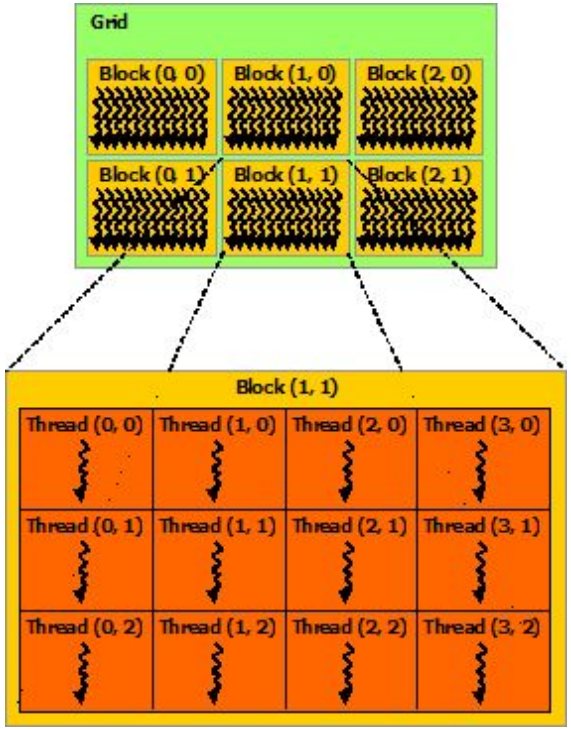
- Велики број хардверских нити чини графичке картице погодним за проблеме за израженим SIMD паралелизмом.

Меморијска хијерархија CUDA графичке картице



- **Глобална меморија**, **L2 кеш** — могу да јој приступе сви SM. **Дељена меморија**, **L1 кеш** — могу јој приступити сва језгра унутар једног SM.
- **Регистри** — приступ на нивоу језгра.
- Константна меморија, меморија текстура

CUDA модел извршавања



- Нит
- Блок нити (енг. *thread block*) — садржи једну или више нити
- Основа (енг. *warp*) — садржи једну или више нити из истог блока, блок је подељен на једну или више основа
- Мрежа нити (енг. *grid*) — садржи један или више блокова нити

OpenACC CUDA мапирања

CUDA логички	**OpenACC логички**	**CUDA физички ниво**
Мрежа нити	више *gang* -ова	сви SM на картици
Блок нити	*gang*	SM
warp	*worker*	SM
CUDA нит	*vector*	CUDA језгро

- Мапирање појмова CUDA модела извршавања (прва колона) на OpenACC модел извршавања (друга колона) и на физичке компоненте GPU која подржава CUDA технологију.

Технике за убрзавање CUDA програма

- Препоруке за убрзање извршавања CUDA програма:
  - Уколико је могуће, CUDA нити треба да приступају узастопним локацијама глобалне меморије (енг. *coalesced access*).
  - Избећи дивергенцију извршавања нити унутар једне основе.
  - Податке који се често користе пребацити у бржу меморију у односу на глобалну (`cache` директива).
  - Оптимизовати пренос података у/из глобалне меморије графичке картице (`data` директива).
  - ...
- Препоручено читање: *Programming Massively Parallel Processors: A Hands-on Approach*, David B. Kirk and Wen-mei W. Hwu, поглавље 6 *Performance Considerations*

Пример 6: matrixop.c

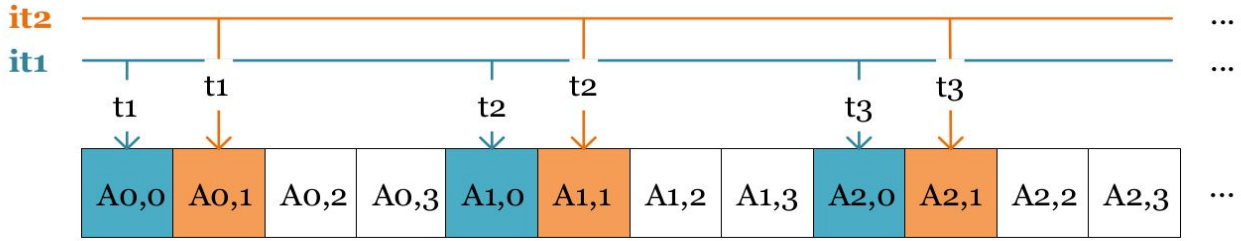
```
int main(int argc, char *argv[]) {
    int *randomMatrix =
        (int *) calloc(MSIZE * MSIZE * sizeof(int));

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

Пример 7: matrixop-coalesced.c

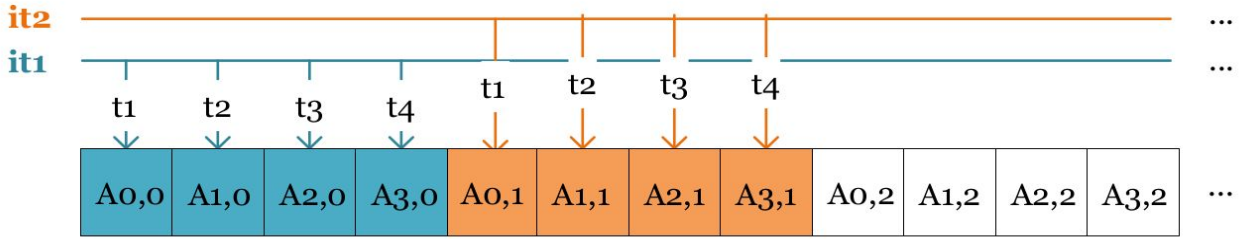
```
int main(int argc, char *argv[]) {
    int *randomMatrix =
        (int *) calloc(MSIZE * MSIZE * sizeof(int));

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

Приступ елементима матрице

Приступ локацијама које нису узастопне

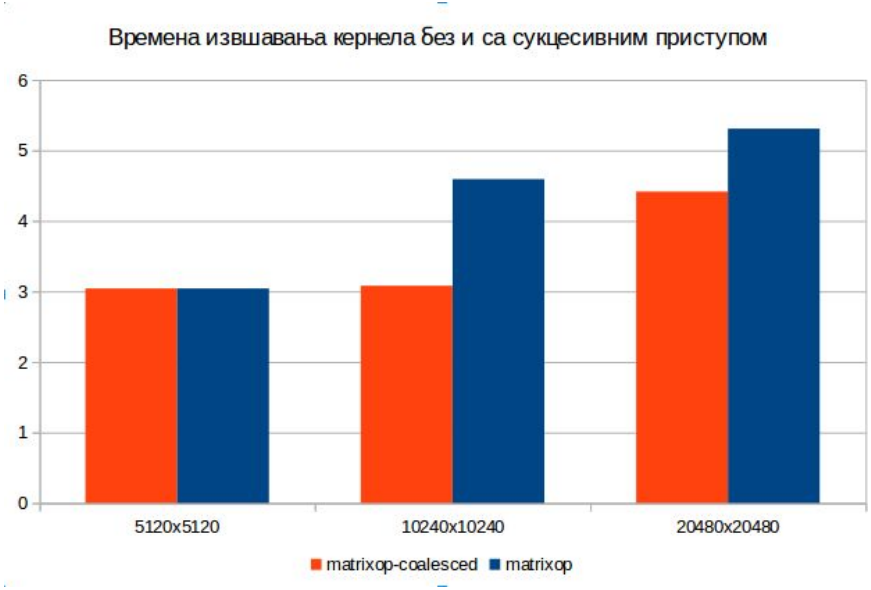


Приступ узастопним локацијама



Претпоставка: сваки елемент матрице обрађује једна CUDA нит.

Анализа времена извршавања



Време извршавања је приказано на логаритамској скали. Нпр. оригинална времена за матрицу 20480x20480 су 26ms и 230ms.

[1] Резултати добијени извршавањем на рачунару са Nvidia GTX1060 картицом и AMD Ryzen 1700 процесором.

Дивергенција при извршавању нити

- Уколико различите нити унутар исте основе имају различите токове извршавања, извршавање група нити са различитим токовима унутар основе се секвенцијализује.
- До дивергенције у извршавању могу довести наредбе за контролу тока извршавања `if-else`, `switch`, `do`, `for`, `while`.
- Видети: CUDA C Best Practices Guide - Branching and Divergence

Неки радови на тему OpenACC, CUDA и OpenCL

- *A Comprehensive Performance Comparison of CUDA and OpenCL*
- *CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application*
- *An in-depth evaluation of GCC's OpenACC implementation on Cray systems*
- *OpenACC cache Directive: Opportunities and Optimizations*

Литература

- Текстуални материјали:
  - OpenACC Programming and Best Practices Guide
  - OpenACC Specification 2.7
  - 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
  - OpenACC Overview Course 2015