OpenMP — припрема за тест

Рачунарски системи високих перформанси

Петар Трифуновић Вељко Петровић

Факултет техничких наука
Универзитет у Новом Саду

Рачунарске вежбе, Зимски семестар 2022/2023.

Пример 1 — паралелни радници

- У *pdf* фајлу дат је текст задатка.
- Сврха задатка је да пре свега тестира познавање *OpenMP* конструкција и клаузула.

Пример 1 — паралелни радници

- Разматрање упутстава за паралелизацију датих у тексту:
  1. Свака нит представља по једног радника:
     - Имплементација ове тачке је једноставна и очигледна.
     - Треба само креирати *parallel* регион и посао радника ће бити подељен по нитима.

Пример 1 — паралелни радници

- Разматрање упутстава за паралелизацију датих у тексту:
  2. Предузеће је овластило једног од радника да распореди материјал по кофама. Дакле, овај део сценарија треба да обави **само један од радника унутар паралелне секције кода**, користећи рачуницу идентичну оној из секвенцијалног алгоритма:
     - Подебљани део текста може се имплементирати *single* конструкцијом.
     - Може се искористити и *master* конструкција, али се након ње мора додати *barrier* конструкција, јер *master* нема имплицитну синхронизацију.

Пример 1 — паралелни радници

- Разматрање упутстава за паралелизацију датих у тексту:
  3. Сваки од радника самостално води евиденцију о послу који је урадио, без централизованог записника, користећи унапред припремљену променљиву *ukupanRadJednogRadnika*. Уместо ажурирања ставке у записнику, сваки радник ажурира ову променљиву тежином управо пренете кофе:
     - Наведена променљива мора бити приватна.
     - Њена декларација може остати ван паралелног региона, па у том случају мора да се обележи као *firstprivate*, да би у паралелном региону била иницијализована на нулу.
     - Друга опција је да се њена декларација и иницијализација пребаци у паралелни регион, што би је учинило приватном.

Пример 1 — паралелни радници

- Разматрање упутстава за паралелизацију датих у тексту:
  4. Предузеће је овластило сваког радника да, по завршетку свог посла, **самостално провери** да ли је он тренутно радник са највећом пренетом тежином и да, уколико јесте, ажурира поље *ukupanRad* унапред припремљене променљиве *ur*. Ову проверу сваки радник ради за себе, унутар паралелне секције кода:
     - *ur* је дељена променљива, па се њена провера и ажурирање мора сместити у *critical* конструкцију.
     - Да *ur* није типа структуре, већ неког од основних типова, могла би се уместо *critical* користити *reduction(max:ur)* конструкција.

Пример 1 — паралелни радници

- Додатни детаљи доступног решења:
  - Промењљиве *ur* и *tezineKofa* се могу, али не морају означити експлицитно као дељене (*shared*).
  - *num_threads(ZELJENI_BROJ_RADNIKA)* креира одговарајући број нити.
  - *schedule(static, brojKofaPoRadniku)* обезбеђује да све нити имају исти број итерација, осим последње, која ће можда имати мање. Свакој нити се редом додељује по *brojKofaPoRadniku* кофа, па уколико број кофа није дељив са бројем радника, за последњу нит ће остати мање посла.

Пример 1 — паралелни радници

- Поставка задатка уређена за самосталну имплементацију дата је у *zip*-у *zadaci*, директоријум *1. paralelni_radnici.*
- Решење је дато у *zip*-у *resenja*.

Пример 2 — креирање *for* петље

- Потребно је размотрити функцију *sequential_function* и *while* петљу дату у њој.
- Ову петљу треба претворити у форму коју је могуће паралелизовати помоћу *OpenMP*-а, након чега треба и имплементирати паралелизацију.

Пример 2 — креирање *for* петље

- Разматрање ствари које спречавају паралелизацију:
  1. Петља *while* облика се не може директно паралелизовати:
     - Треба наћи начин за пребацивање у *for* облик.
     - Видимо да број итерација зависи од вредности *x* променљиве, која се у свакој итерацији ажурира по истом принципу.
     - Могуће је израчунати унапред број итерација.

Пример 2 — креирање *for* петље

- Разматрање ствари које спречавају паралелизацију:
  1. Петља *while* облика се не може директно паралелизовати:
     - Размотримо законитост по којој се мења *x*:

```
x += step;
x = x * -1;
step = step * -1;
```

     - Наредна табела показује како се, по горенаведеној рачуници, мењају вредности *x* и *step*

	X (почетно 0)	Step (почетно 10)
Итерација 1	-10	-10
Итерација 2	20	10
Итерација 3	-30	-10
Итерација 4	40	10

Пример 2 — креирање *for* петље

- Разматрање ствари које спречавају паралелизацију:
  1. Петља *while* облика се не може директно паралелизовати:
     - Из табеле, а и из кода, види се да се *x* повећава за 10 по апсолутној вредности, а знак варира.
     - Границу *while* петље (променљива *limit*) зато треба за почетак поделити са 10 и додати 1. На пример, ако је *limit=15*, целобројним дељењем добијамо 1, повећавамо на 2 и тиме добијамо број итерација. Ово би био крај рачунице да *x* не мења знак.
     - Рецимо да је граница 25. Дељењем са 10 добијамо 2 и повећавамо на 3. Након 3 итерације, што се види из табеле, *x* ће бити -30, што није веће од 25. Због тога се број итерација додатно повећава на 4.
     - Коначно — ако се након дељења са 10 и повећавања за 1 добије непаран број итерација, треба га повећати за 1. Ако је број паран, не треба га увећавати. То је све што је потребно за рачунање тачног броја итерација.

Пример 2 — креирање *for* петље

- Разматрање ствари које спречавају паралелизацију:
  2. Претходно *x* зависи од наредног:
     - Да бисмо извршили паралелизацију, потребно је изгубити зависности између итерација, односно учинити да наредно *x* не зависи од претходног.
     - Разматрајући вредности ове променљиве, закључујемо да је апсолутну вреност могуће добити по формули *step * i,* где је *i* ознака тренутне итерације.
     - Знак такође зависи од итерације, па се апсолутна вредност множи са $(-1)^i$ (није неопходно овако одредити знак, може и једноставним *if*-ом које испитује итерацију).

Пример 2 — креирање *for* петље

- Разматрање ствари које спречавају паралелизацију:
  3. Све нити приступају истој вредности *max_y:*
     - Због тога део кода који испитује ову променљиву, и ажурира најбоље *x* и *y,* треба сместити у критичну секцију.
     - Да се ажурира само једна од ове две променљиве, могла би се искористити *reduction(max:x)* конструкција.

Пример 2 — креирање *for* петље

- Поставка задатка уређена за самосталну имплементацију дата је у *zip*-у *zadaci*, директоријум *2. kreiranje_for_petlje.*
- Решење је дато у *zip*-у *resenja*.

Пример 3 — паралелизација зависности

- У задатку постоји *for* петља која по одређеном алгоритму обилази и мења елементе низа (линеаризоване матрице).
- Потребно је установити да ли се петља може паралелизовати.

Пример 3 — паралелизација зависности

- У телу петље, види се да тренутни елемент (одређен индексима *i* и *j*) зависи од неког од претходних елемената низа (одређеног индексима *i-1* и *j*).
- Ово онемогућава паралелизацију спољашње петље, јер оно што би некој нити било на *i-1* индексу, односно индексу за читање, некој другој нити би било на *i* индексу, односно индексу за упис.
- Немамо гаранцију да ће нит прочитати одговарајући елемент пре него што нека друга нит у тај елемент упише нову вредност.

Пример 3 — паралелизација зависности

- Да ли је паралелизација могућа по унутрашњој петљи?
- Свака нит користила би исти индекс *j* и за читање и за упис и не би дошло до преклапања елемената за упис и читање између различитих нити.
- Због тога је могуће паралелизовати унутрашњу петљу.
- Проблем — ако бисмо ставили *#pragma omp parallel for* одмах изнад унутрашње петље, без икаквих других измена, у свакој итерацији по *i* креирао би се нови паралелни *fork-join* регион, што може да утиче негативно на перформансе.
- Као решење, могуће је *j* петљу извући испред *i* петље, јер то неће променити резултат, али ће онда један паралелни регион бити довољан.
- **Напомена:** обратити пажњу на то да *i* мора бити приватно када постане унутрашња петља. Зато или декларисати *i* у самој *for* наредби, или је декларисати ван *for*-а и означити експлицитно као приватну променљиву.

Пример 3 — паралелизација зависности

- Поставка задатка уређена за самосталну имплементацију дата је у *zip*-у *zadaci*, директоријум *3. paralelizacija_zavisnosti.*
- Решење је дато у *zip*-у *resenja*.

Сажетак — најважније ствари са припреме

- Први пример служи искључиво као провера познавања *OpenMP* конструкција и клаузула.
- Друга два примера су много важнија
  - Други пример:
    - Могућност претварања петље која се не може паралелизовати у петљу која може. Потребно је обратити пажњу на то чиме је почетна петља ограничена и на основу тога одредити број итерација.
    - Отклањање зависности између променљивих у различитим итерацијама. Проучити како је могуће вредност променљиве израчунати директно на основу броја текуће итерације.
  - Трећи пример:
    - Приметити која петља је независна и по којој се може извршити паралелизација.
    - Ако је могуће, извући ту петљу као спољашњу, како би се креирао минималан број паралелних региона.
  - Оба примера:
    - Установити који су критични делови кода и сместити их у *critical* секцију. Ако је могуће, **увек** искористити *reduction* уместо *critical*.

Решења

- Директоријум *vezba7* садржи запаковане задатке и решења.

Додатна вежба

- За додатну вежбу, погледати задатке са претходних термина вежби.
- Свакако, задаци које прати ова презентација су најважнији за припрему.