Tutorijal: TBB, III deo

- Raspoređivač zadataka -

Sadržaj

- Uvod
- Programiranje zasnovano na zadacima
- Kada ga ne koristiti?
- Primeri

Uvod

- Raspoređivač zadataka je osnova šablona petlji
- Šabloni petlji (npr. `parallel_for`) skrivaju složenost raspoređivača
- Kada problem (algoritam) ne odgovara šablonu petlji visokog nivoa, može se direktno koristiti raspoređivač zadataka
- Mogu se praviti novi šabloni visokog nivoa

Sadržaj

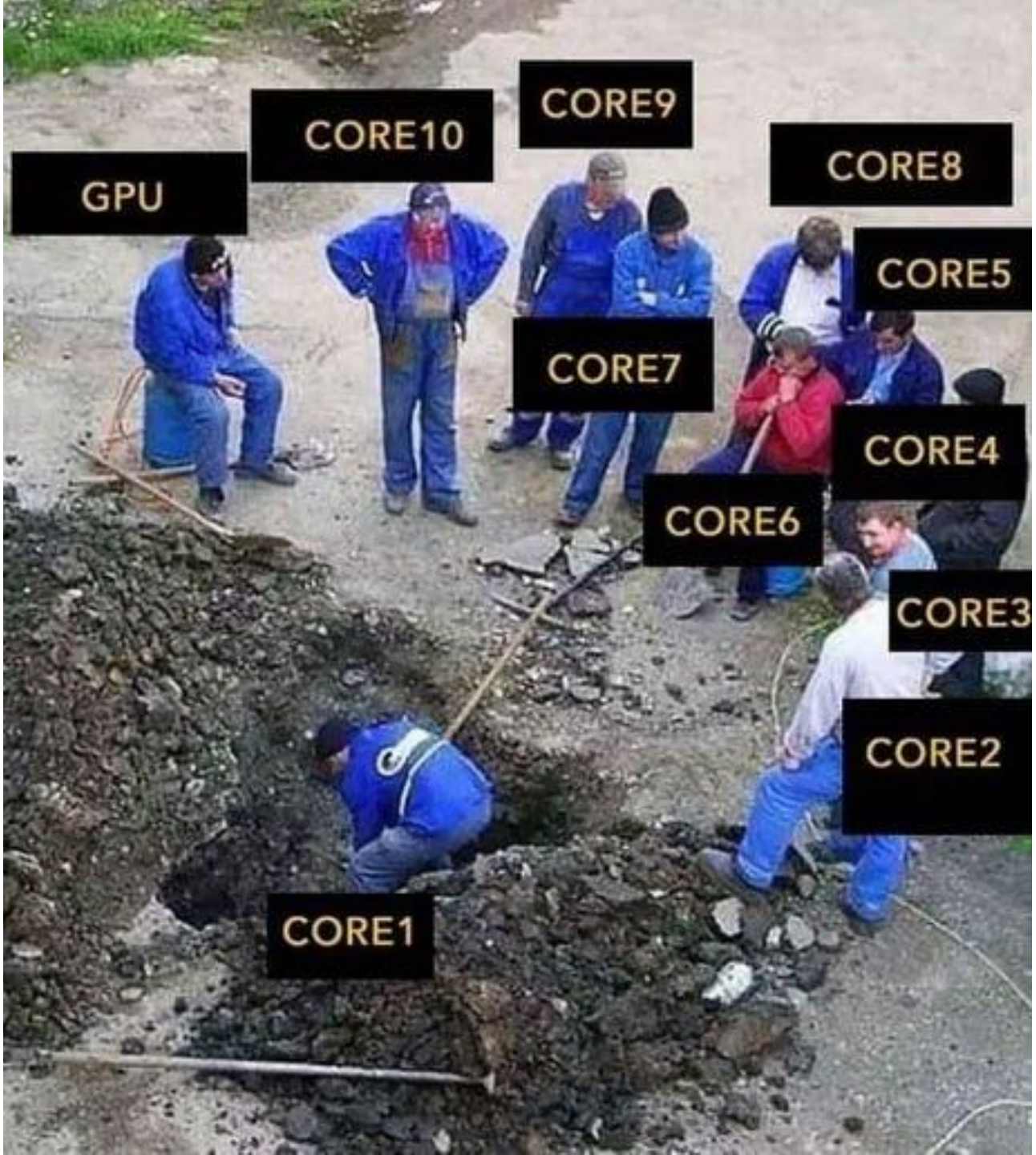
- Uvod
- Programiranje zasnovano na zadacima
- Kada ga ne koristiti?
- Primeri

Programiranje zasnovano na zadacima (1/7)

- Bolje je program formulisati pomoću logičkih zadataka, a ne niti, iz nekoliko razloga:
  - Uvođenje paralelizma korišćenjem raspoloživih resursa
  - Veća brzina pokretanja i uništavanja zadatka
  - Efikasnija procena
  - Bolje uravnoteženje opterećenja
  - Razmišljanje na visokom nivou

Programiranje zasnovano na zadacima (2/7)

- Niti napravljene sa dostupnim paketom bi bile logičke niti koje se preslikavaju na fizičke niti (tj. jezgra) procesora.
- Za račune koji ne zavise od spoljnih događaja, najveća efikasnost se postiže kada se tačno jedna logička nit izvršava unutar jedne fizičke.
- Mogući problemi:
  - Logičkih niti ima manje od fizičkih,
  - Logičkih niti ima više od fizičkih.


GPU
CORE10
CORE9
CORE8
CORE5
CORE7
CORE4
CORE6
CORE3
CORE2
CORE1

Programiranje zasnovano na zadacima (3/7)

- Drugi problem je veći jer dovodi do izvršavanja logičkih niti u vremenskim isečcima (*time slice*), što zahteva dodatno vreme izvršenja.
- Raspoređivač pokušava da zaobiđe ovaj problem stvarajući jednu logičku nit po fizičkoj niti, preslikavajući zadatke na logičke niti i uzimajući u obzir smetnje sa drugim nitima.

Programiranje zasnovano na zadacima (4/7)

- Važna prednost zadataka u odnosu na niti je da su mnogo bolji u pogledu vremena pokretanja i uništavanja (na Linux OS-u do 18 puta brže, na Windows-u i do 100 puta):
  - Niti, za razliku od TBB zadataka, imaju svoju lokalnu kopiju mnogih resursa (stanje registara, magacinska memorija, itd), pa čak i svoj identifikator procesa (*process id*),
  - TBB zadatak je najčešće samo mala rutina.

Programiranje zasnovano na zadacima (5/7)

- TBB zadaci su takođe efikasni jer je raspoređivanje nepravedno (*unfair*).
- Najčešće se vremenski odsečci dodeljuju redom u krug (pravedno), jer je to najbezbednija strategija koja se može preduzeti bez spoznaje višeg nivoa organizacije programa.
- Kako TBB ima informacije o višem nivou programa, može i da žrtvuje pravednost radi efikasnosti.

Programiranje zasnovano na zadacima (6/7)

- Raspoređivač radi uravnoteženja opterećenja (load balansing):
  - Pored korišćenja odgovarajućeg broja niti, potrebno je ravnomerno rasporediti zadatke na raspoložive niti.
  - Potrebno je da program bude izdeljen na dovoljno male zadatke (ali ne previše male) da bi raspoređivač korektno dodelio zadatak niti u cilju uravnotoženja opterećenja.

Savet: Projektovati program tako da stvara znatno više zadataka nego što ima niti, i da prepusti raspoređivaču preslikavanje zadataka na niti.

Programiranje zasnovano na zadacima (7/7)

- Osnovna prednost zadataka u odnosu na niti je mogućnost razmišljanja na višem nivou.
- Sa nitima se mora razmišljati na niskom nivou fizičkih niti da bi se postigla visoka efikasnost (jedna logička nit po fizičkoj).
- Takođe, mora se raditi sa dosta grubom podelom niti.
- Sa zadacima je moguće koncentrisati se na logičke zavisnosti zadataka, a raspoređivanje prepustiti samom raspoređivaču.

Sadržaj

- Uvod
- Programiranje zasnovano na zadacima
- Kada ga ne koristiti?
- Primeri

Kada ne treba koristiti programiranje zasnovano na zadacima (1/2)

- Raspoređivač zadataka je namenjen za algoritme visokih performansi koji se sastoje od isključivo neblokirajućih zadataka.
- Može biti koristan i kada se zadatak ponekad (retko) blokira.
- Ukoliko dolazi do čestog blokiranja niti (čekanje na ulaz/izlaz, semafor i sl.), dolazi do gubitka performansi (dok je nit blokirana, ne radi ni na jednom zadatku).

Kada ne treba koristiti programiranje zasnovano na zadacima (2/2)

- Ukoliko niti čekaju, program se neće dobro izvršavati bez obzira na broj niti koje postoje.
- Ako postoje blokirajući zadaci, najbolje je koristiti niti.
- Raspoređivač zadataka u potpunosti podržava kombinovanje niti sa TBB zadacima.

TBB zadaci - zaključak

- Raspoređivač zadataka radi najefikasnije kada imamo šablon „grananje-pridruživanje“ (*eng*. fork-join)
- Treba da postoji dovoljno mnogo mrešćenja kako bi se stvorio dovoljno veliki broj zadataka kako bi sve niti uvek bile uposlene i mogle da „ukradu“ deo posla od ostalih niti kada završe sa svojim

Sadržaj

- Uvod
- Programiranje zasnovano na zadacima
- Kada ga ne koristiti?
- Primeri

Primer 1 – Fibonačijevi brojevi (1/7)

- Primer računanja *n*-tog Fibonačijevog broja.
- Koristi neefikasan način računanja, ali demonstrira osnove biblioteke sa zadacima, koristeći jednostavan obrazac rekurzivnih zadataka.
- TBB obrazac rekurzivnih zadataka omogućava stvaranje većeg broja zadataka, što je uslov za skalabilnost ubrzanja u programiranju zasnovanom na zadacima.

Primer 1 – Fibonačijevi brojevi (2/7)

- Sekvencijalni kod:

```
long SerialFib( long n ) {
    if( n<2 )
        return n;
    else
        return SerialFib(n-1)+SerialFib(n-2);
}
```

- Kod najvišeg nivoa za paralelnu verziju zasnovanu na zadacima:

```
long ParallelFib( long n ) {
    long sum;
    task_group g;
    ...
    g.run([&] {x = ParallelFib(n - 1); });
    g.run([&] {y = ParallelFib(n - 2); });
    g.wait();
    return sum;
}
```

Grupa zadataka

Poziv lambda izraza koji pristupa parametrima preko reference

Pravljenje i pokretanje zadatka

Sinhronizacija

Lambda izrazi (1/2)

- Deo C++ jezika od verzije C++11
- Definište anonimni funkcijski objekat na mestu pozivanja
- Tipično se koriste da enkapsuliraju par linija koda koje se prosleđuju algoritmima ili asinhronim metodama (slučaj koji ćemo mi koristiti na ovom predmetu)

Lambda izrazi (2/2)

- Delovi lamba izraza:

```
  1       2        3              4              5
[=/&]   ()  mutable throw()  -> int  {
      6   telo izraza
}
```

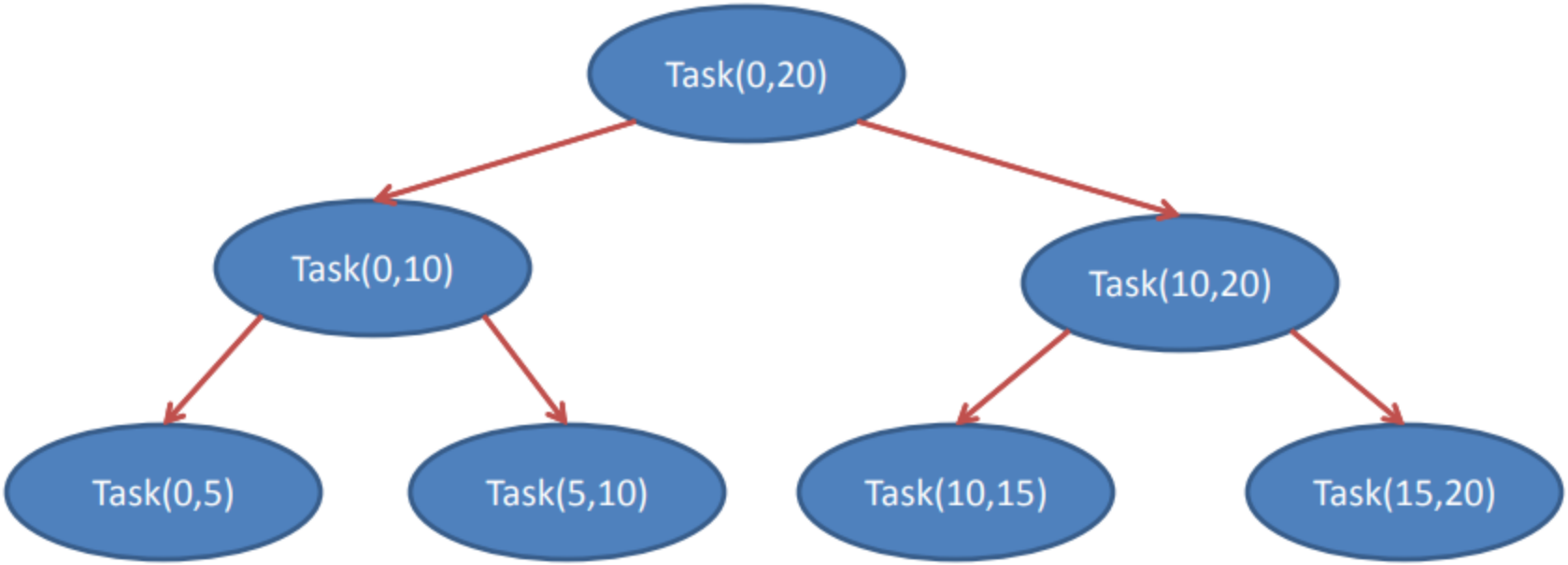
- U našim primerima ćemo mi koristiti samo delove 1 (`[&]` kao indikaciju da se promenljivima pristupa po referenci) i 6 (telo lambda izraza)

Primer 1 – Fibonačijevi brojevi (3/7)

- Prvo se kreira grupa zadataka (`task_group g`)
- Kada dođemo do mesta gde treba izazvati paralelizam (u ovom slučaju kada dolazi do rekurzivnog poziva), poziva se metoda `run`
- Ovoj metodi se prosleđuje lambda izraz koji obavlja račun od interesa
- `g.run([&]{x = ParallelFib(n - 1);});`
- Pošto se ove grane mogu izvršavati u paraleli, poziva se onoliko `run`-ova koliko računa može ići u paraleli nakon čega sledi poziv
- `g.wait();`

Primer 1 – Fibonačijevi brojevi (4/7)

- Dolazi do stvaranja strukture fork-join koja odgovara šablonu podeli i zavladaj slično kao što smo imali kod reduktora



Primer 1 – Fibonačijevi brojevi (5/7)

- Ovakav zadatak je opisan sa malo većim kodom u odnosu na `SerialFib`, jer izražava paralelizam zasnovan na zadacima, bez pomoći ikakvih dodataka na standardni C++.
- Metoda `ParallelFib` prvo proverava da li je vrednost broja za koji trenutno računamo manji od odsečne vrednosti (koja je u ovom slučaju eksperimentalno određena da bude 16) jer je u tom slučaju serijski algoritam brži

Primer 1 – Fibonačijevi brojevi (6/7)

- Prelazak na sekvencijalni algoritam u slučaju da je problem mali je karakteristika šablona paralelizma *podeli-i-zavladaj*
- U `else` grani, prave se dva zadatka potomka, koji računaju (n-1)-vi i (n-2)-gi Fibonačijev broj
- Proces se ponavlja i stvara se veliki broj zadataka
- Raspoređivač može odmah da pokrene zadatke čim se kreiraju jer su nezavisni jedan od drugog

Primer 1 – Fibonačijevi brojevi (7/7)

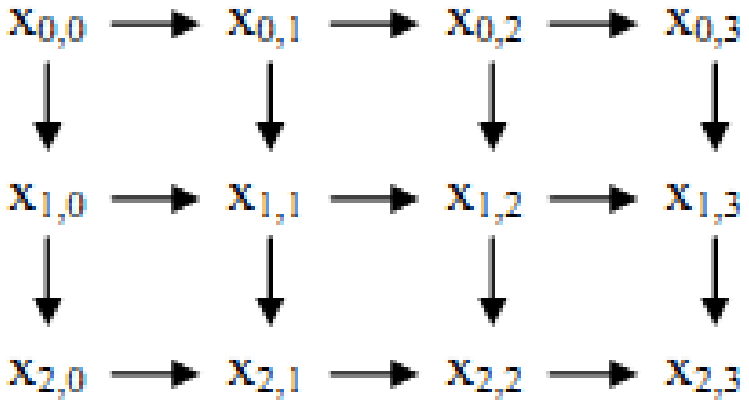
- Na prvi pogled se čini da je paralelizam ograničen jer se stvaraju samo dva zadatka potomka, ali trik je u rekurzivnom paralelizmu:
  - Svaki od dva zadatka potomka stvara nova dva zadatka potomka i tako dok nije `n<Cutoff`.
  - Prednost raspoređivača zadataka je što tako nastali potencijalni paralelizam pretvara u stvarni paralelizam na vrlo efikasan način – birajući zadatke koje pokreće tako da fizičke niti održava zauzetim uz relativno malu zamenu konteksta niti.

Primer 2 – Quick Sort

- Algoritam za „brzo“ sortiranje
- Za element se kaže da je sortiran ako su svi elementi levo od njega manji, a svi desno veći (bez obzira na to da li su elementi levo/desno sami po sebi sortirani)
- Takođe koristi podeli i zavladaj šablon kao i rekurzivna implementacija računanja N-tog elementa Fibonačijevog niza

Primer 3 – WaveFront (1/3)

- Simulacija fronttalasa
- Prvo se izvršavaju elementi koji su gore i levo, pa potom elementi ispod i desno



- Za sinhronizaciju se koriste atomske promenljive nad kojima su izvesne operacije „nedeljive“ iz perspektive drugih niti

Primer 3 – WaveFront (2/3)

- Svaki zadatak uzima svoju vrednost i sabira je sa vrednošću zadatka koji je iznad i levo od njega
- Zadatak se pokreće tek kada je vrednost njegove atomske promenljive 0
- Ubačena je i simulacija obrade kako bi se ilustrovalo kako se postiže paralelizam

Primer 3 – WaveFront (3/3)

- Prvo se napravi matrica koja se sastoji od atomskih `int` promenljivih, i matrica iste veličine koja ima `double` vrednosti
- Potom se dodele vrednosti za atomske promenljive (koliko zadataka moraju da sačekaju pre nego što krenu sa izvršavanjem) i vrednost koju sabiraju
- Kada paralelni zadatak odradi svoj posao, dekrementira atomske promenljive svog desnog i donjeg suseda i u slučaju da je vrednost atomskog `int`-a 0, pokreće zadatak

Dodatni materijali

- Samostalno analizirati priložene primere
- Knjiga Paralelno programiranje
- TBB dokumentacija: vodiči za programiranje i priručnici