Računarski sistemi visokih performansi

Nikola Vukić, Petar Trifunović, Veljko Petrović

Računarske vežbe
Zimski semestar 2024/25.

Hibridno programiranje
MPI + OpenMP

Sadržaj

- Šta je hibridno programiranje?
- Arhitektura hibridnog programiranja.
- Hibridno programiranje *MPI + OpenMP*.
- Mapiranje procesa i niti na hardver.
- Zadaci

Šta je hibridno programiranje?

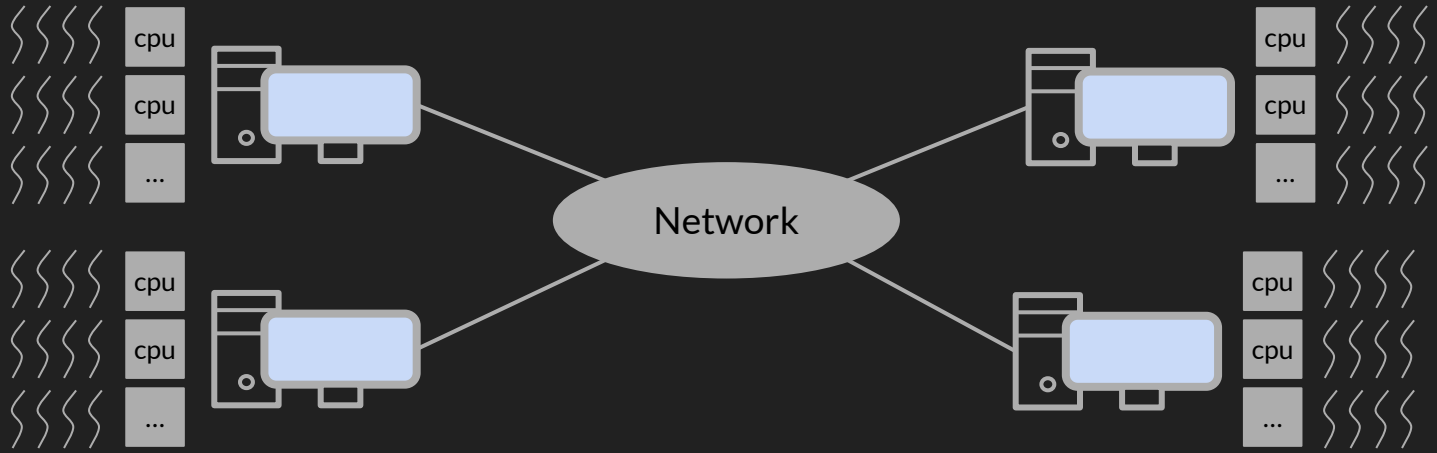
Šta je hibridno programiranje?

- U *HPC*-u, podrazumeva korišćenje više različitih modela paralelnog programiranja unutar jedne aplikacije.
- U našem slučaju, odnosi se na kombinovano korišćenje *OpenMP* i *MPI* programskih modela.

Arhitektura hibridnog programiranja

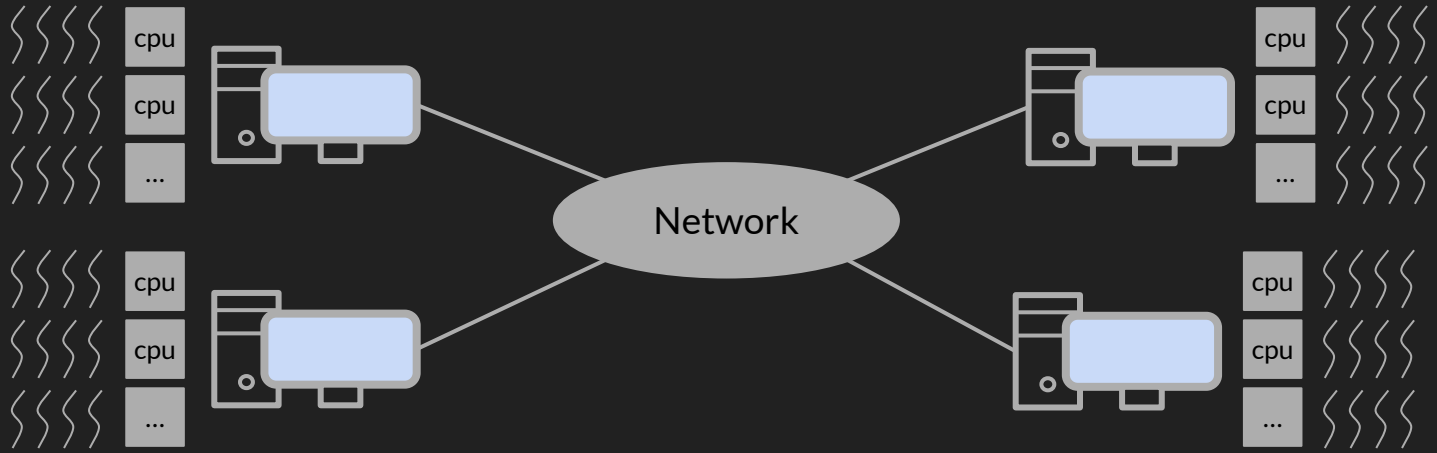
Arhitektura hibridnog programiranja

- Čvorovi koji komuniciraju preko mreže i imaju podršku za višenitno izvršavanje.



Arhitektura hibridnog programiranja

- Globalno se koristi distribuirana memorija, a lokalno deljena.



Arhitektura hibridnog programiranja

- *MPI* model se koristi za komunikaciju između udaljenih čvorova, ali se može koristiti i za komunikaciju između različitih procesa jednog čvora.
- *OpenMP* se koristi da na nivou niti paralelizuje poslove koje obavlja jedan proces.

Hibridno programiranje *MPI* + *OpenMP*

Različiti režimi hibridnog programiranja

- Grubo gledano, postoje dva važna režima hibridnog programiranja, kada se koriste *MPI* i *OpenMP*[1]:
  - **vektorski režim** – *MPI* pozivi su izvan paralelnih *OpenMP* regiona, odnosno nalaze se u delovima koda gde je aktivna samo *master* nit, i
  - **režim zadataka** (engl. *task mode*) – *MPI* pozivi se nalaze unutar *OpenMP* paralelnih regiona; od nivoa interoperabilnosti zavisi koje niti će moći da izvršavaju *MPI* pozive

1 – Ovo nije zvanična, opštevažeća podela, već podela data u knjizi Hager, G., i Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers* (prvo izdanje). CRC Press. https://doi.org/10.1201/EBK1439811924

Nivoi interoperabilnosti u *MPI*

- Nivo interoperabilnosti određuje koliko niti može da podrži jedan *MPI* proces, kao i to koje niti mogu izvršavati *MPI* pozive.
- Nivoi interoperabilnosti navedeni u *MPI* 4.1 standardu su sledeći:
  - *MPI_THREAD_SINGLE* – samo jedna nit u procesu
  - *MPI_THREAD_FUNNELED* – može postojati više niti u procesu, ali samo glavna nit može izvršavati *MPI* pozive
  - *MPI_THREAD_SERIALIZED* – može postojati više niti u procesu, sve mogu izvršavati *MPI* pozive, ali se oni ne izvršavaju konkurentno, već se serijalizuju
  - *MPI_THREAD_MULTIPLE* – može postojati više niti u procesu i sve mogu izvršavati *MPI* pozive bez ograničenja

Kompajliranje i izvršavanje hibridnog programa

- Kompajliranje:
  - $ mpicc <izvorna_datoteka> -fopenmp
- Pokretanje:
  - korišćenjem *mpiexec*, kao i u običnom *MPI* programu, uz eventualno navođenje broja procesa (*-np N* opcija), ili broja niti po procesu (podešavanjem *OMP_NUM_THREADS* promenljive okruženja)

Hello, world!

```
#include "mpi.h"
#include <omp.h>
#include <stdio.h>

int main(int argc, char* argv[]) {

   MPI_Init(&argc, &argv);

   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   #pragma omp parallel
   {
       printf("Hello world from thread %d of process %d!\n", omp_get_thread_num(), rank);
   }

   MPI_Finalize();
}
```

Inicijalizovanje hibridnog programa

- *MPI* nudi funkciju *MPI_Init_thread* za inicijalizaciju hibridnog programa:
  `int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)`
- Ovo je modifikovana verzija *MPI_Init* funkcije sa dva dodatna parametra, i to:
  - *required*:
    - parametar kojim se zahteva određen nivo interoperabilnosti
    - može uzeti neku od ranije opisanih vrednosti (*MPI_THREAD_SINGLE*, *MPI_THREAD_FUNNELED*, *MPI_THREAD_SERIALIZED*, ili *MPI_THREAD_MULTIPLE*)
  - *provided* – parametar koji će se postaviti na jednu od četiri navedene vrednosti i nagovestiti koji nivo interoperabilnosti je zapravo dostupan

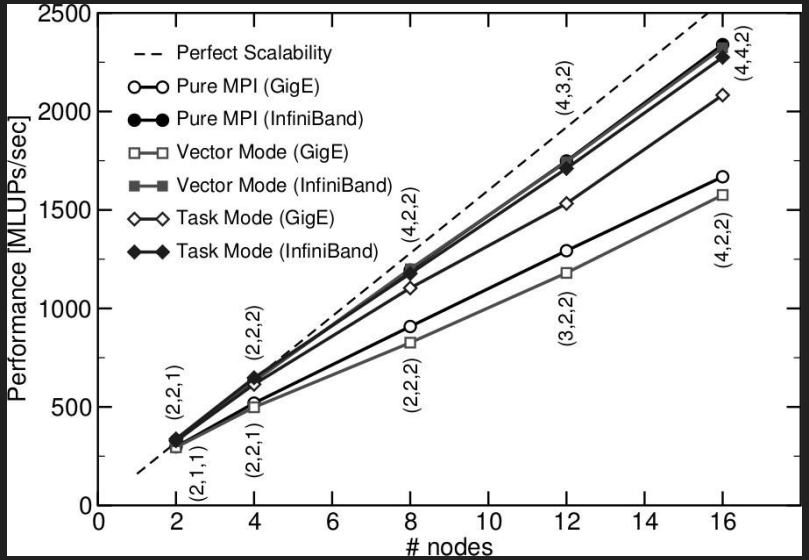
Inicijalizovanje hibridnog programa

- *OpenMPI* implementacija ne daje gotovo nikakve garancije za odnos zahtevane i omogućene interoperabilnosti, pa je zato neophodno proveriti izlazni parametar *provided* nakon inicijalizacije programa.
- I pored ovog parametra, realan broj niti u procesu u mnogome zavisi od verzije korišćene *MPI* biblioteke, načina njenog *build*-ovanja, okruženja na kome se program izvršava i slično.
- Moguće je dobiti i viši nivo od zahtevanog, ukoliko, na primer, biblioteka radi tako da na arhitekturama koje su višenitne svakako obezbedi *MPI_THREAD_MULTIPLE* interoperabilnost.
- U takvim okruženjima je najbolje eksplicitno postaviti željeni broj niti i procesa prilikom pokretanja programa.

Poređenje različitih režima hibridnog programiranja

- Primer poređenja preuzet je iz knjige *Introduction to High Performance Computing for Scientists and Engineers*.
- Kao primer uzet je Jakobi algoritam za rešavanje sistema linearnih jednačina.
- Izvršeno je poređenje sledećih implementacija:
  - čist *MPI*,
  - hibridni program sa vektorskim režimom, i
  - hibridni program sa režimom zadataka.
- Implementacije su testirane na klasterima sa:
  - *Gigabit Ethernet (GigE)* mrežom, i
  - *Infiniband* mrežom.

Poređenje različitih režima hibridnog programiranja



preuzeto iz knjige Hager, G., i Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers* (prvo izdanje). CRC Press. https://doi.org/10.1201/EBK1439811924

- Komunikacija je usko grlo u bilo kojoj implementaciji algoritma.
- Kada je komunikacija efikasna (kao što je slučaj sa *InfiniBand*-om), tada dodavanje *OpenMP*-a ne utiče previše na izvršavanje.
- Kada je komunikacija dovoljno znatno sporija (*GigE*), dodatna paralelizacija unutar čvorova menja performanse:
  - Promena performansi zavisi od prirode problema.
  - Vektorski režim ne ubrzava rad jer su komunikacija i *OpenMP* paralelizacija još uvek serijalizovani.
  - Režim zadataka u ovom slučaju ubrzava stvari, jer se komunikacija i *OpenMP* paralelizacija prepliću.

Poređenje različitih režima hibridnog programiranja

- Važan zaključak demonstriran na primeru sa sledećeg slajda, a koji je eksplicitno dat i u knjizi, glasi:
  - ***Consider going hybrid only if pure MPI scalability is not satisfactory.***

Hibridno programiranje – prednosti i mane

- **Prednosti:**
  - Niti lakše komuniciraju i sinhronizuju se (imaju zajednički memorijski prostor, mogu da koriste zajednički keš ukoliko je dostupan…).
  - Eksploatiše se dodatni nivo paralelizma u odnosu na čist *MPI*.
  - Omogućava preklapanje izvršenja i komunikacije.
  - …
- **Mane:**
  - Zahtevnije za pisanje – neretko ne postoji način da se inkrementalno čista *MPI* implementacija prebaci u hibridnu; više prostora za grešku…
  - Može se ispostaviti neefikasno, ako je deo vremena u kom se vrši neki proračun (koji bi se paralelizovao pomoću *OpenMP*-a) neuporediv sa vremenom koje se troši na komunikaciju.
  - …

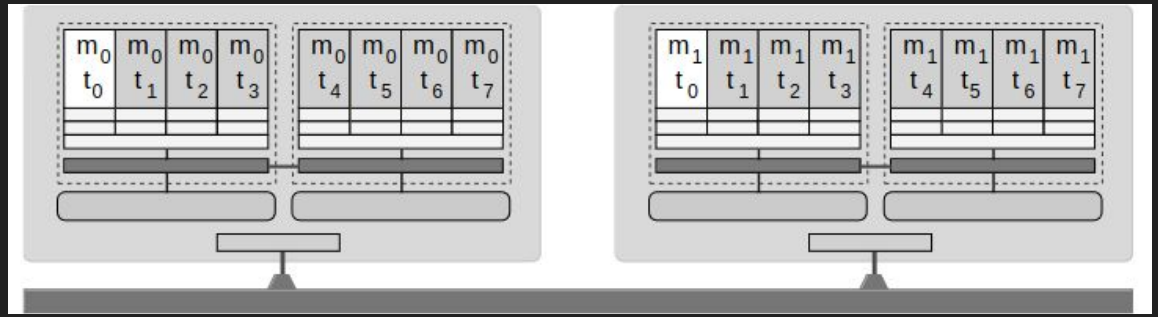
Mapiranje procesa i niti na hardver

Mapiranje procesa i niti na dostupan hardver

- Performanse hibridnog programa zavise od broja procesa, broja niti po procesu, kao i rasporeda procesa i niti po raspoloživim hardverskim resursima.
- U nastavku je dat pregled različitih mapiranja procesa i niti na klasteru od dva čvora sa po dva *socket*-a, od kojih svaki ima po četiri jezgra.
- Primeri i slike preuzeti su iz knjige Hager, G., i Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers* (prvo izdanje). CRC Press. https://doi.org/10.1201/EBK1439811924.

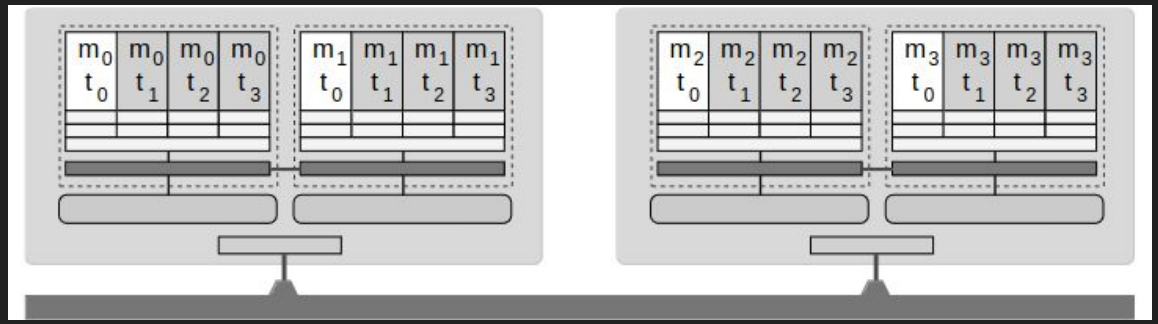
Mapiranje procesa i niti na dostupan hardver

- Jedan *MPI* proces po čvoru:
  - Procesi m_0 i m_1, svaki sa po osam niti (t_0,…,t_7), raspoređeni su svaki na svoj čvor.
  - Problemi u deljenju keša (nivo niti) i *NUMA* pristupu (nivo procesa).
  - Jedan proces na čvoru potencijalno neće moći da iskoristi pun potencijal brze mrežne konekcije.
  - Lako vezivanje (*pin*-ovanje, engl. *pinning*) niti i procesa za hardver.
  - Minimizacija broja *MPI* procesa.



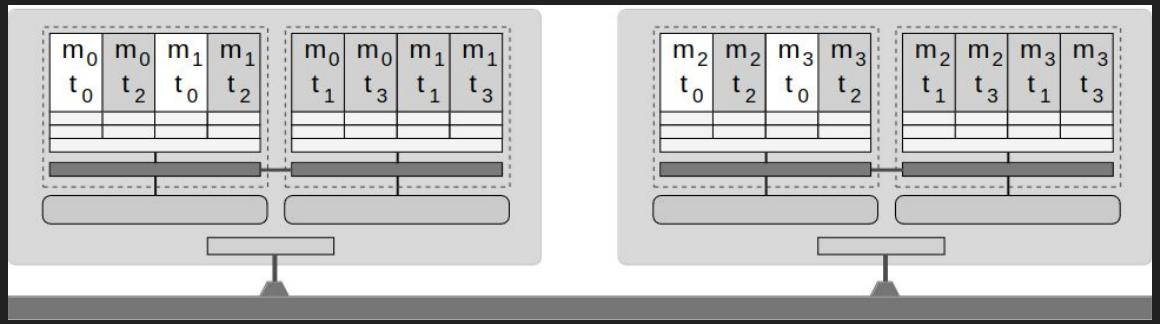
Mapiranje procesa i niti na dostupan hardver

- Jedan *MPI* proces po *socket*-u (procesoru):
  - Procesi m_0, m_1, m_2 i m_4, svaki sa po četiri niti (t_0,…,t_3), raspoređeni su svaki na svoj *socket*.
  - Komplikovanije vezivanje procesa i niti za hardver.
  - Veći broj *MPI* procesa.
  - Moguće konkuretno preklapanje komunikacije između *socket*-a i komunikacije između čvorova.
  - Lokalizovaniji keš (nivo niti) i *NUMA* pristup (nivo procesa).



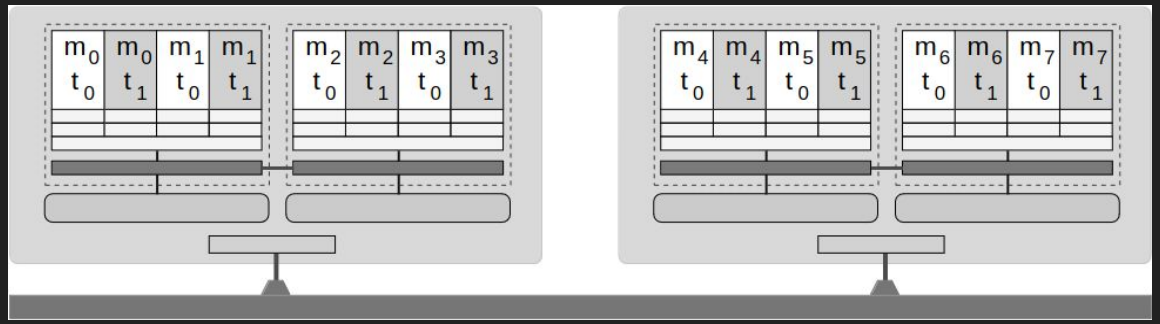
Mapiranje procesa i niti na dostupan hardver

- Svaki *MPI* proces podeljen na dva soketa, *round-robin* raspoređivanje niti:
  - Modifikacija prethodnog mapiranja, gde su procesi podeljeni na po dva *socket*-a, a njihove niti se po *socket*-ima raspoređuju po *round-robin* principu.
  - Dobra strana je potencijalna brza komunikacija putem keša između dva procesa koji dele *socket*.
  - Ostali aspekti ovakvog mapiranja su loši – komplikovanije vezivanje za hardver, slabija lokalizacija *NUMA* pristupa, teža sinhronizacija niti jednog procesa…



Mapiranje procesa i niti na dostupan hardver

- Više *MPI* procesa po soketu:
  - Procesi $m_0,...,m_7$, svaki sa po dve niti (t_0, t_1), raspoređeni po soketima, tako da dva procesa dele jedan soket.
  - Pruža mogućnost tri nivoa komunikacije između procesa – *intrasocket* (unutar istog *socket*-a), *intersocket* (između *socket*-a istog čvora), *internode* (između različitih procesa).
  - Strategija optimizacije može biti minimizovanje najskupljeg nivoa komunikacije, a forsiranje najjeftinijeg.
  - Lokalizovani *NUMA* pristupi i pristupi kešu.
  - Komplikovanije vezivanje za hardver.



Mapiranje procesa i niti na dostupan hardver

- Strategije su opisane sa pretpostavkom da arhitektura i implementacija alata omogućava vezivanje (*pin*-ovanje) za hardverske resurse.
- Mogućnosti vezivanja za hardverske resurse zavise od okruženja.
- Opcije *--bind-to* i *--map-by* komande *mpiexec* se mogu koristiti za upravljanje mapiranjem.

Literatura

- Glavni izvor za ovu temu vežbi je knjiga Hager, G., i Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers* (prvo izdanje). CRC Press. https://doi.org/10.1201/EBK1439811924

Zadaci

Zadatak 1 — Računanje vrednosti broja π

- Napisati hibridni *MPI + OpenMP* program koji računa vrednost broja π preko sledećeg integrala:

$$\int_0^1 \frac{4}{\left(1+x^2\right)} \mathrm{d}x$$

- Razmisliti o tome koji deo posla se može podeliti po procesima, a zatim da li unutar jednog procesa postoji prostor za *OpenMP* paralelizaciju.
- Meriti vreme izvršavanja korišćenjem *omp_get_wtime* funkcije *omp.h* biblioteke.
- Uporediti ovo rešenje sa sekvencijalnim i čistim *OpenMP* rešenjem sa prvog termina vežbi.
- Menjati broj niti i procesa i upoređivati performanse.
- Primer rešenja: datoteka *resenja/01_hyb_pi.c*.

Zadatak 2 — Pretraga niza celih brojeva

- Napisati hibridni *MPI + OpenMP* program koji traži zadati broj u nizu od najmanje milion celih brojeva iz opsega [-100, 100]. Element niza nasumično generisati u bilo kom procesu, pre početka obrade.
- Treba ispisati indeks prve pojave traženog elementa u nizu.
- Uporediti hibridni program sa sekvencijalnim, kao i sa čistim *OpenMP* i *MPI* implementacijama.
- Menjati broj niti i procesa i upoređivati performanse.

Zadatak 3 — Množenje matrica

- Napisati hibridni *MPI + OpenMP* program koji množi dve kvadratne matrice.
- Razmisliti kako bi se određeni delovi matrice mogli distribuirati različitim procesima.
- Iskoristiti skripte za rad sa *hdf5* formatom datoteka sa prethodnih vežbi.
- Uporediti hibridni program sa sekvencijalnim, kao i sa čistim *OpenMP* i *MPI* implementacijama.
- Menjati broj niti i procesa i upoređivati performanse.