Pristup i korištenje resursa Supek i Vrančić

R810

OBRAZOVNI PROGRAMI srca

priručnik

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu su verziju priručnika izradili:

Autori: Maja Arnaut, Martin Belavić, Marko Hrženjak, Emir Imamagić, Ivan Mitrović, Kristijan Mrkalj, Jurica Špoljar, Daniel Vrcić, Katarina Zailac

Recenzent: Nenad Mijić

Urednica: Zrinka Popić

Lektorica: Ana Đorđević

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

ISBN (PDF) 978-953-382-027-9

Verzija priručnika: R810-20241002

Sadržaj

1. Uvod u napredno računanje

Po završetku ovoga poglavlja polaznik će moći:

- *objasniti što je paralelizacija*
- *razlikovati HPC i HTC resurse*
- *iskazati red veličine trenutne procesorske snage računala.*

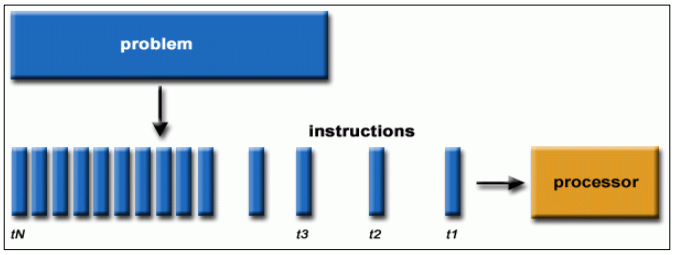
Trajanje poglavlja:

20 min

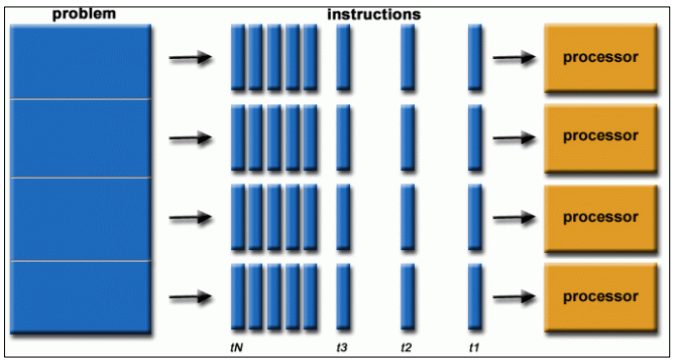
1.1. Paralelizacija

Program je niz naredbi koje procesor izvršava u svrhu rješavanja nekog problema. Ako taj problem možemo razdijeliti na više manjih neovisnih dijelova, otvara se mogućnost istodobnog i neovisnog izvršavanja dijelova na različitim, ali povezanim procesorskim jezgrama, odnosno kažemo da se takav program može izvršavati paralelno.

Jedan od bitnijih koncepata u računarstvu visokih performansi (engl. *High Performance Computing* −HPC*)* je paralelizacija. Aplikacije koje nisu paralelizirane koriste jedan procesor i provode naredbe jednu za drugom te zato izračun traje dulje. Paralelizacijom se problem razdvaja na više manjih od kojih se svaki istovremeno provodi na zasebnoj procesorskoj jezgri, a prednost je višestruko ubrzanje računa. Mana je puno složeniji razvoj aplikacija jer to zahtijeva korigiranje serijskog koda programa (Slika 1) u paralelni kod (Slika 2).



Slika 1: Serijsko izvođenje programa



Slika 2: Paralelno izvođenje programa

Ahmdalov zakon je ograničenje, a on objašnjava kako će ubrzanje bilo kojeg procesa uvijek biti ograničeno uskim grlom procesa, tj. onim

dijelom koji se ne može poboljšati (u našem slučaju paralelizirati). Jednostavnije rečeno: program će biti brz onoliko koliko njegov najsporiji dio dopušta.

1.2. HPC, HTC i Cloud

HPC podrazumijeva napredno korištenje računalnih resursa koji omogućavaju brže izvršavanje računa nego što bi bio slučaj, na primjer, na osobnim računalima.

Superračunala su vrhunac izvedbe HPC-a. Za razliku od tradicionalnih računala, superračunala mogu sadržavati tisuće procesora. U slučaju računalnih klastera računalni se resursi (CPU, GPU, memorija) grupiraju u takozvane čvorove, tj. odvojene poslužitelje/računala, a visokopropusna lokalna mreža omogućava da djeluju kao skup umreženih računala ili jedan resurs.

Superračunala su smještena u podatkovnim centrima (Slika 3), a koriste se u svim područjima gdje je potrebna velika računalna snaga, bilo za računski zahtjevna istraživanja, obradu podataka, umjetnu inteligenciju ili nešto treće.

Računarstvo s visokom propusnošću (engl. *High-throughput computing*, HTC) fokusira se na izvršavanje velikog broja neovisnih zadataka kroz duži vremenski period. Cilj HTC-a je da se postigne što veća propusnost poslova, odnosno da se obradi što veći broj poslova u određenom razdoblju (puno dužem nego na skali HPC-a). Najčešće se koristi za aplikacije s manjim zahtjevima za performansama. Za razliku od HPC resursa, HTC ne zahtijeva visokoučinkovitu lokalnu mrežu. Postoje aplikacije kod kojih potpuna paralelizacija svih zadataka nije moguća ili gdje paralelizacija na većoj skali nije efikasna. Važno je napomenuti kako se na HTC resursu mogu gotovo jednako efikasno, kao i na HPC resursu, izvršavati paralelni zadaci koji ne zahtijevaju korištenje više od jednog računalnog čvora.

Računarstvo u oblaku (engl. *cloud computing*) je fleksibilan resurs koji korisnicima omogućuje konfiguraciju operacijskog sustava, upravljanja resursima i nadzora prilagođavajući se specifičnim potrebama korisnika

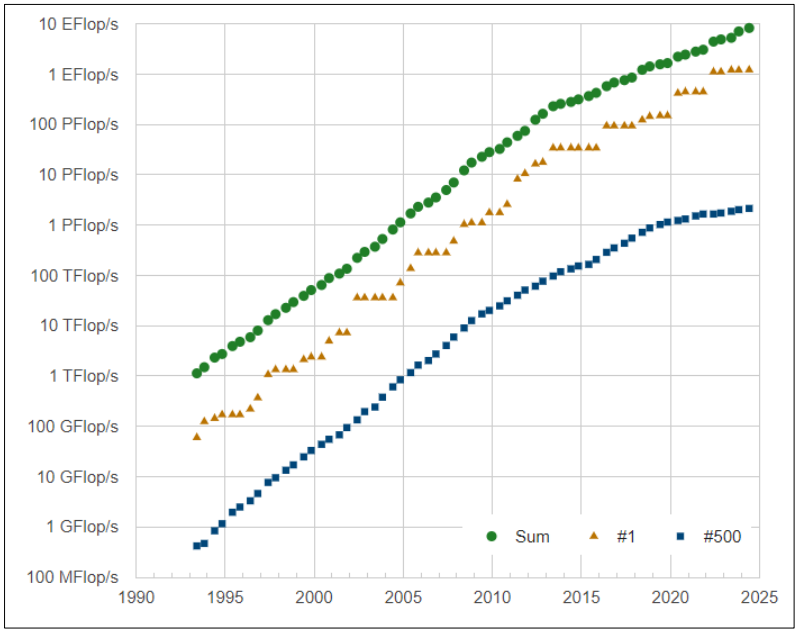
Slika 3: Podatkovni centar

1.3. Procesorska snaga računala

Mjera za brzinu računala je FLOPS (engl. *floating-point operations per second*), što označava broj aritmetičkih operacija s pomičnim zarezom izvršenih u jednoj sekundi.

Lista TOP500 je lista najmoćnijih računala (Slika 4). Za listu TOP500 koristi se *High Performance Linpack* (HPL) kao referentni *benchmark* program za mjerenje performansi superračunala. HPL (https://www.netlib.org/benchmark/hpl/) je softverski paket koji rješava linearni sustav u dvostrukoj preciznosti na računalima s raspodijeljenom memorijom. Današnja najmoćnija računala imaju računalnu moć koja se mjeri u stotinama PFLOPS-a (petaskalarana računala), a sljedeći korak su egzaskalarna računala (EFLOPS).

Prosječna brzina računanja kod ljudi reda je veličine mFLOPS-a, dok je obični kalkulator u prosjeku na 10 FLOPS-a. Supek je petaskalarno računalo, a ima 1,25 PFLOPS-a.



Slika 4: Lista TOP500 (podaci za lipanj 2024. godine)

2. Resursi Supek i Vrančić

Po završetku ovoga poglavlja polaznik će moći:

- *navesti arhitekturu i karakteristike okoline superračunala Supek*
- *navesti arhitekturu i karakteristike okoline HTC-a Vrančić.*

Trajanje poglavlja:

40 min

2.1. Superračunalo Supek

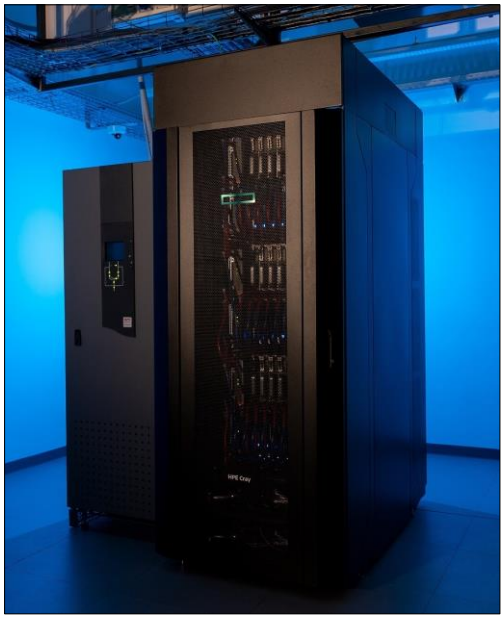
Jedna od ključnih značajki Supeka (Slika 5) je interkonekcijski sustav koji koristi komercijalnu mrežu *Cray Slingshot*. Mreža *Cray Slingshot* omogućuje brzu razmjenu podataka između čvorova, što je ključno za vrlo brzo izvođenje računalnih operacija.

Osim toga, Supek ima i visokoučinkovit sustav hlađenja koji omogućuje održavanje optimalne temperature unutar računalnog sustava te pouzdanost sustava i u najzahtjevnijim uvjetima rada. Ova vrsta hlađenja omogućuje efikasno odvođenje topline vodom putem sustava cijevi i izmjenjivača topline.

Performanse Supeka su 1250 TFLOPS-a R_{max} i 1830 TFLOPS R_{peak}.

R_{max} je vrijednost vršnih performansi koje su ostvarene pomoću *benchmarka* koji rješavaju realni problem na optimalni način. R_{max} u tom smislu predstavlja maksimalnu izmjerenu vrijednost performansi sustava koja u sebi uključuje korištenje sustava kao cjeline, što podrazumijeva i visokopropusnu mrežu.

R_{peak} odnosi se na teoretski maksimalni broj operacija koji sustav može dostići ako koristi punu računalnu snagu svih procesora istovremeno.



Slika 5: Supek

2.1.1. Arhitektura (Supek)

Na Supeku postoje 52 CPU čvora, koji ukupno sadrže 6656 jezgri što pružaju R_{peak} 260 TFLOPS-a (Tablica 1).

Tablica 1: Arhitektura superračunala Supek

CPU čvorovi	GPU čvorovi	Čvorovi s velikim memorijskim kapacitetom	Pristupni čvorovi CPU čvor
• 52 čvora • 6 656 CPU jezgri • 260 TFLOPS-a	• 20 čvorova • 80 GPU-a • 1280 CPU jezgri • 1560 TFLOPS-a	• 2 čvora • 256 CPU jezgri • 10 FLOPS-a	• 2 x AMD Epyc 7763 2,45 GHz • 128 CPU jezgri • 256 GB RAM-a • 1,6 TB NVMe SSD-a
Čvorovi	**Čvorovi**	**Čvorovi**	**GPU čvor**
• 2 x AMD Epyc 7763 2,45 GHz • 128 CPU jezgri • 256 GB RAM	• 1 x AMD Epyc 7763 2,45 GHz • 4 x NVIDIA A100 • 64 CPU jezgre • 512 GB RAM-a	• 2 x AMD Epyc 7763 2,45 GHz • 128 CPU jezgri • 4 TB RAM-a • 1,7 TB NVMe SSD-a	• 1 x AMD Epyc 7763 2,45 GHz • 1 x NVIDIA A100 • 64 CPU jezgri • 128 GB RAM-a • 3,4 TB NVMe SSD-a

Svaki **CPU poslužitelj** sastoji se od 2 procesora AMD EPYC 7763 sa 64 jezgre (128 jezgri po poslužitelju odnosno 6656 na svim CPU poslužiteljima) te ukupno 16 DDR4 memorijska modula od 16 GB radne memorije (256 GB ukupno – 241 GB dostupno).

Na Supeku se nalazi ukupno **20 GPU čvorova**. Svaki čvor sadrži 1 AMD procesor EPYC 7763 sa 64 jezgre i 4 grafička procesora NVIDIA A100, svaki po 40 GB ugrađene memorije HBM2e. GPU poslužitelji opremljeni su s ukupno 8 DDR4 memorijska modula od 64 GB radne memorije (512 GB ukupno, odnosno 493 GB dostupno).

Supek sadrži i **2 CPU čvora s više RAM memorije**. Svaki od njih opremljen je s 2 procesora AMD EPYC 7763. Sadrži sveukupno 32 DDR4 memorijskih modula kapaciteta 128 GB radne memorije, što osigurava svakom čvoru ukupno 4 TB radne memorije. Poslužitelj je opremljen lokalnim NVMe SSD-om kapaciteta 1,92 TB. Ostali čvorovi nemaju „vlastiti“ disk, već koriste zajedničko brzo spremište (*scratch*).

Na Supeku su dostupna dva pristupna poslužitelja/čvora.

CPU pristupni poslužitelj ne sadrži grafički procesor, ali sadrži 2 procesora AMD EPYC 7763 sa 64 jezgre i ukupno 16 DDR4 memorijska modula od 16GB radne memorije (256 GB ukupno − 241 GB slobodno).

GPU pristupni poslužitelj opremljen je jednim grafičkim procesorom NVIDIA A100 u i jednim procesorom AMD EPYC 7763 sa 64 procesorske jezgre. Poslužitelj je opremljen s 8 DDR4 memorijskih modula kapaciteta 16 GB radne memorije (128 GB ukupno).

Na Supeku je uključen *Simultaneous Multithreading* (SMT) koji omogućuje izvođenje 2 dretvi (*threada*) na jednog fizičkoj CPU jezgi. U slučaju kada je višedretvena aplikacija ograničena latencijom može imati

benefit prilikom izvođenja. Korisnicima je ostavljena odluka hoće li koristiti SMT ili ne s obzirom na aplikacije koje pokreću.

2.1.2. Osnovne karakteristike okoline

Operacijski sustav na Supeku je **Red Hat Enterprise Linux 8**, a sustav za upravljanje poslovima je **PBS Pro**.

Datotečni sustav na Supeku je **Lustre**. To je paralelni datotečni sustav kao i BeeGFS, namijenjen za HPC okuženja, ali nije optimiziran za rad s većim brojem manjih datoteka. Iz tog razloga na Supeku koristimo **Apptainer** za kontejnerizaciju Python virtualnih okruženja, tj. *pip* i *conda* virtualnih okruženja.

Modulima upravljamo **Lua modulefilesom**.

Na Supeku je na raspolaganju sistemski Cray MPICH i Open MPI v5.0.. Novi OpenMPI 5, za razliku od starijih verzija, može upotrebljavati Slingshot mrežu u punom potencijalu.

2.1.3. Spremišta

Spremište na Supeku je organizirano u 2 segmenta (Tablica 2).

Tablica 2: Organizacija spremišta na Supeku

	Kapacitet	Dostupnost	Putanja	Namjena
Supek	580 TB	Svi čvorovi	`/lustre/home/<username>`	**Korisnički** podaci potrebni za izvođenje poslova
			`/lustre/group/<groupname>`	**Grupni** podaci potrebni za izvođenje poslova
			`/lustre/scratch/<jobid>`	Privremeni **korisnički** podaci prilikom izvođenja poslova
Štampar	2 PB	Pristupni čvorovi	`/storage/home/<username>`	**Korisnički** podaci koji se trenutačno ne koriste u poslovima
			`/storage/group/<groupname>`	**Grupni** podaci koji se trenutačno ne koriste u poslovima

ClusterStor E1000, veličine **580 TB**, služi za općenitu primjenu te se na njega spremaju korisnički i grupni podaci potrebni za izvođenje poslova, a u Tablica 2 prikazane su te putanje pojedinih mapa. Svi čvorovi imaju pristup njima, pa se mogu koristiti u skriptama za podnošenje poslova.

Štampar je zasebno, izdvojeno spremište, na kojem je za potrebe Supeka osigurano **2 PB**, tj. 2000 TB, što služi isključivo za pohranu velikih podataka, tj. za arhiviranje. Ovo spremište vide samo pristupni čvorovi na Supeku i Padobranu (opisanom u poglavlju 2.2. Računarstvo u oblaku: Vrančić), pa se ne može koristiti u skriptama za podnošenje poslova. Na obama resursima može mu se pristupiti na

/storage/home/username lokaciji ili za grupe na /storage/group/groupname.

Kada datoteke više nisu potrebne u radu, potrebno ih je prebaciti na Štampar kako se ne bi brzo „zatrpavalo“ radno spremište.

2.1.4. Standardni (produkcijski) redovi poslova

Prilikom podnošenja poslova potrebno je odabrati jedan od dostupnih redova poslova. Redovi se odabiru s obzirom na tražene resurse i namjenu poslova. Po namjeni poslove možemo podijeliti na standardne, tj. produkcijske (Tablica 3) i testne (Tablica 4).

Tablica 3: Produkcijski redovi poslova (Supek)

	Opis	Dostupno resursa	Zadane vrijednosti	Ograničenje	TMPDIR
cpu	Standardni red za CPU čvorove	**Ukupno:** • 6656 CPU jezgri **Po čvoru:** • 128 CPU jezgri, 241 GiB RAM-a	• 1 *chunk* • 2 CPU jezgre • 1800 MiB RAM-a • 48 h izvođenja	Maks. 7 dana	`/lustre/scratch/<jobid>`
gpu	Standardni red za GPU čvorove	**Ukupno:** • 80 GPU-a, 1280 CPU jezgri **Po čvoru:** • 4 GPU-a, 64 CPU jezgri, 493 GiB RAM-a	• 1 *chunk* • 1 CPU jezgra • 1 GPU • 120 GiB RAM-a • 24 h izvođenja	Maks. 7 dana	`/lustre/scratch/<jobid>`
bigmem	Standardni red za čvorove s velikim memorijskim kapacitetom	**Ukupno:** • 256 CPU jezgri **Po čvoru:** • 128 CPU jezgri, 4019 GiB RAM-a	• 1 *chunk* • 1 CPU jezgra • 30 GiB RAM-a • 24 h izvođenja	Maks. 7 dana	`/scratch/<jobid>` (1,7 TB)
cpu-single	Red za serijske CPU poslove	**Ukupno:** • 640 CPU jezgri	• 1 *chunk* • 1 CPU jezgra • 1800 MiB RAM-a • 48 h izvođenja	Maks. 7 dana	`/lustre/scratch/<jobid>`

Tablica 4: Testni redovi (Supek)

	Opis	Dostupno resursa	Zadane vrijednosti	Ograničenje	TMPDIR
cpu-test	Red za testiranje na CPU čvorovima	• 6656 CPU jezgri	• 1 *chunk* • 2 CPU jezgre • 1800 MiB RAM-a • 1 sat izvođenja	• Maks. 1 sat • Maks. 256 jezgri • Maks. 480 GiB RAM-a	`/lustre/scratch/<jobid>`
gpu-test	Red za testiranje na GPU čvorovima	• 256 CPU jezgri • 16 GPU-a	• 1 *chunk* • 1 GPU • 1 CPU jezgra • 120 GiB RAM-a • 1 sat izvođenja	• Maks. 1 sat • Maks. 4 GPU-a • Maks. 493 GiB RAM-a	`/lustre/scratch/<jobid>`
login-cpu	Red za testiranje na CPU *login* čvoru	• 100 CPU jezgri	• 1 *chunk* • 1 CPU jezgre • 1 sat izvođenja	• Maks. 1 sat	`/scratch/<jobid>` (1,6 TB)
login-gpu	Red za testiranje na GPU *login* čvoru	• 1 GPU-a • 40 CPU jezgri	• 1 *chunk* • 1 GPU • 1 CPU jezgra • 1 sat izvođenja	• Maks. 1 sat	`/scratch/<jobid>` (3,4 TB)

CPU red odabire se ako nije potrebna značajna količina RAM-a, tj. radne memorije, ili grafička kartica, a primarni je cilj korištenje CPU-a. Svaki čvor ima 128 jezgri (256 logičkih, koje se mogu vidjeti htop-u) i 241 GiB RAM-a (ne 256, u vidu 2*128 jer dio koristi operacijski sustav). Poslovi mogu koristiti i privremene direktorije u **/lustre/scratch/** prilikom izvršavanja.

GPU red odabire se ako je primarni cilj korištenje grafičkih kartica. Svaki čvor ima 64 CPU jezgri, 4 GPU-a i 128 GB RAM-a. Ukupna memorija također neće biti kako je „očekivano“ direktnim množenjem 128*4=512, nego 493 GiB zbog operacijskog sustava.

Bigmem je red koji se odabire ako poslovi zahtijevaju velike količine radne memorije. Svaki čvor sadrži 128 jezgri i 4 TB radne memorije. Ovi čvorovi koriste „pravi“ **/scratch** za privremeni radni direktorij, odnosno posjeduju vlastite „diskove“, za razliku od ostalih. Treba obratiti pozornost na to kako na cpu i gpu redovima *scratch* ima „neograničenu“ veličinu, dok je na bigmem redu ograničen stvarnim kapacitetom diskova.

Za potrebe testiranja dostupni su redovi login-cpu i login-gpu, gdje su prisutni realni lokalni diskovi.

Napomena

***Za one koji su koristili Isabellu.**

Na Isabelli je *scratch* bio poseban disk, dok na CPU redu nije. Neće dati dodatnu brzinu, ali će se izbrisati po završetku poslova, pa se ne mora voditi računa o velikom broju datoteka koje neke aplikacije generiraju. Datoteke koje su potrebne mogu se kopirati u radni direktorij.

2.2. Računarstvo u oblaku − Vrančić

Vrančić je zasnovan na platformi za računarstvo u oblaku OpenStack koja je značajno drugačija od Supeka. Korisnici na Vrančiću mogu kreirati vlastito virtualno računalo odnosno zauzeti dio Vrančićeva čvora, tj. onoliko resursa koliko je potrebno te na njega postaviti operacijski sustav po želji.

Dio Vrančića obuhvaća računalni klaster Padobran, koji je po načinu rada gotovo identičan Supeku, no namijenjen je za aplikacije koje ne skaliraju dobro, npr. ako se ne mogu širiti na više računalnih čvorova ili loše skaliraju iznad određenog broja CPU jezgara.

Padobranove resurse koristi bioinformatička platforma Galaxy koja omogućava interaktivno podnošenje poslova i rad u grafičkom sučelju.

Također, u sklopu Vrančića dostupna je i platforma Jupyter, popularna u sklopu programskih jezika kao što su Julia, Python i R, koja omogućava rad u tzv. *notebooksima*.

Slično kao Supek, Vrančić ima CPU i GPU čvorove te čvorove s velikim memorijskim kapacitetom (Tablica 5). Arhitektura je slična, procesori su također AMD-ovi iste generacije, no imaju nešto niži takt.

86 je CPU čvorova koji u ovom slučaju imaju i dvostruko više RAM-a, 4 GPU čvora te 2 čvora s velikim memorijskim kapacitetom.

Značajan dio resursa, odnosno 50 CPU čvorova, koristi računalni klaster Padobran.

Tablica 5: Arhitektura Vrančića

CPU čvorovi	GPU čvorovi	Čvorovi s velikim memorijskim kapacitetom
• 86 čvora • 11 008 CPU jezgri • 384 TFLOPS-a	• 4 čvora • 16 GPU-a • 256 CPU jezgri • 310 TFLOPS-a	• 2 čvora • 256 CPU jezgri • 9 FLOPS-a
Čvorovi	**Čvorovi**	**Čvorovi**
• 2 x AMD Epyc 7713 2,0 GHz • 128 CPU jezgri • 512 GB RAM-a	• 1 x AMD Epyc 7713 2,0 GHz • 4 x NVIDIA A100 • 64 CPU jezgre • 512 GB RAM-a	• 2 x AMD Epyc 7713 2,0 GHz • 128 CPU jezgri • 2 TB RAM-a

2.2.1. Računalni klaster Padobran

Padobran je računalni klaster koji koristi dio Vrančićevih resursa, odnosno 50 CPU poslužitelja, i uz to ima vlastiti mali pristupni poslužitelj sa 16 jezgri. Strogo se preporučuje da se na pristupnom čvoru Vrančića ne kompiliraju aplikacije niti izvode probni poslovi.

Za razliku od Supeka, Padobran nema Slingshot interkonekciju između čvorova, ima nešto slabije procesore, nema GPU čvorove, ali ima više RAM-a te omogućava izvođenje poslova do 30 dana, dok je to na Supeku samo 7 dana.

Spremište mu je ukupnog kapaciteta 464 TB.

2.2.2. Osnovne karakteristike okoline

Operacijski sustav na Padobranu je **Rocky Linux 8**, sustav binarno kompatibilan RHEL-u 8.

Sustav za upravljanje poslovima je OpenPBS, potpuno analogan PBS-u Pro sa Supeka.

Datotečni sustav na Padobranu je BeeGFS.

Modulima se upravlja Lua modulefilesom.

Što se tiče MPI-a, na Padobranu se nalazi Open MPI i Intelov MPI, no budući da Padobran nema brzu vezu, oni služe samo za širenje unutar čvora.

2.2.3. Spremišta

U slučaju Padobrana (Tablica 6) postoji i brzo spremište na */beegfs-fast/scratch* koji koriste aplikacije što generiraju i učitavaju jako puno podataka s diska (I/O intenzivne aplikacije).

Standardno spremište nalazi se u */beegfs/home* ili */beegfs/group*.

Štamparu možete analogno pristupiti i s Padobrana.

Tablica 6: Organizacija spremišta na Padobranu

	Kapacitet	Dostupnost	Putanja	Namjena
Padobran	384 TB	Svi čvorovi	`/beegfs/home/<username>`	**Korisnički** podaci potrebni za izvođenje poslova
			`/beegfs/group/<groupname>`	**Grupni** podaci potrebni za izvođenje poslova
	80 TB	Svi čvorovi	`/beegfs-fast/scratch/<jobid>`	Privremeni **korisnički** podaci prilikom izvođenja poslova
Štampar	2 PB	Pristupni čvorovi	`/storage/home/<username>`	**Korisnički** podaci koji se trenutačno ne koriste u poslovima
			`/storage/group/<groupname>`	**Grupni** podaci koji se trenutačno ne koriste u poslovima

2.2.4. Standardni (produkcijski) redovi poslova

Redovi poslova na Padobranu razlikuju se od onih na Supeku. Na Padobranu je moguće koristiti samo CPU-ove raspodijeljene u 2 reda.

Razlika u redovima je maksimalno vrijeme izvođenja posla, 7 i 30 dana (Tablica 7).

Lokacija privremenog direktorija ili TMPDIR varijable također je drugačija, nalazi se na /beegfs-fast/scratch/$PBS_JOBID (a ne na/tmp/$PBS_JOBID).

Tablica 7: Redovi poslova na Padobranu

	Opis	Dostupno resursa	Zadane vrijednosti	Ograničenje	TMPDIR
cpu	Red za poslove do 7 dana	**Ukupno:** • 5760 CPU jezgri **Po čvoru:** • 128 CPU jezgri, 475 GiB RAM-a	• 1 *chunk* • 1 CPU jezgra • 7 dana izvođenja	Maks. 7 dana	`/beegfs-fast/scratch/<jobid>`
cpu_30	Red za poslove do 30 dana	**Ukupno:** • 1408 CPU jezgri **Po čvoru:** • 128 CPU jezgri, 475 GiB RAM-a	• 1 *chunk* • 1 CPU jezgra • 30 dana izvođenja	Maks. 30 dana	`/beegfs-fast/scratch/<jobid>`

3. Pohrana i upravljanje podacima − Puh

Po završetku ovoga poglavlja polaznik će moći:

- *navesti što je usluga Puh*
- *koristiti Puh na resursima usluga za napredno računanje.*

Trajanje poglavlja:

20 min

Puh je usluga Srca koja služi za pohranu podataka u oblaku, poput servisa Google Disk, OneDrive, DropBox, iCloud i slično. Jednom kad se korisnik registrira na uslugu (https://www.srce.unizg.hr/puh/zahtjev) pristupa joj elektroničkim identitetom iz sustava AAI@EduHr i može raditi u grafičkom sučelju na *webu*.

Usluga Puh omogućava 200 GB prostora s mogućnošću proširenja.

Vlastiti Puh „disk“ može se montirati ili *mountati* na Supek ili Padobran i koristiti za preuzimanje i učitavanje datoteka.

Na stranici vlastitog profila u Puhu, u sigurnosnim postavkama, generira se lozinka specifična za Supek, a zatim se na Supeku upisuje naredba **`puh-mount`** i vlastiti elektronički identitet iz sustava AAI@EduHr. U *promptu* je potrebno upisati lozinku koja je generirana u *web*-sučelju usluge Puh i nakon toga će vlastiti disk biti montiran na Supek ili Padobran.

Kada se disk želi odmontirati, potrebno je upisati naredbu **`puh-umount`**.

4. Prijava i pristup

Po završetku ovoga poglavlja polaznik će moći:

- *navesti što je usluga Napredno računanje*
- *podnijeti zahtjev za korištenje usluge Napredno računanje*
- *navesti tko i pod kojim uvjetima ostvaruje pristup usluzi Napredno računanje.*

Trajanje poglavlja:

20 min

Na *web*-stranicama usluge Napredno računanje može se pronaći detaljan pravilnik korištenja usluge Napredno računanje koji se odnosi i na Supek i Vrančić (https://wiki.srce.hr/display/NR/Pristup#Pristup-Pravila).

Zahtjev za korištenje resursa podnosi se preko *web*-aplikacije dostupne na https://computing.srce.hr/. U aplikaciju se korisnik može prijaviti koristeći vlastiti elektronički identitet iz sustava AAI@EduHr, a inozemni korisnici mogu pristupiti putem eduGain federacije identiteta.

Pravo pristupa HPC usluzi moguće je dobiti ako postoji potreba za izračune u sklopu projekata financiranih iz javnih izvora ili u slučaju izrade završnih, diplomskih specijalističkih i doktorskih radova na javnim visokim učilištima i javnim znanstvenim institutima u Republici Hrvatskoj.

Ako se radi o projektima, postoje dvije kategorije projekata:

- projekti upisani u sustav CroRIS, odnosno istraživački
- projekti koji nisu upisani u CroRis, odnosno institucijski projekti.

Ako je korisniku potreban pristup resursima u svrhu istraživačkih projekata, i projekt i voditelj (voditelj na ustanovi u slučaju međunarodnih projekata) i svi suradnici za koje se traži pristup računalnim resursima moraju biti upisani u CroRIS. Ako projekt nije verificiran u sustavu CroRIS ili nisu svi suradnici prijavljeni na projektu, korisnik se mora obratiti CroRIS admistratoru vlastite ustanove za pomoć ili više informacija.

U slučaju institucijskih projekata zahtjev podnosi osoba upisana u Upisnik znanstvenika, ali pritom ta osoba **nije** prijavljena ni na jednom aktivnom projektu u sustavu CroRIS. Taj projekt ima maksimalno trajanje od 1 godine, ali voditelj projekta može tražiti produljenje ako su uvjeti i dalje zadovoljeni.

Pristup za potrebe izvođenja praktične nastave dozvoljen je i na Supeku i na Vrančiću, a rezervacije resursa moguće su samo na Vrančiću.

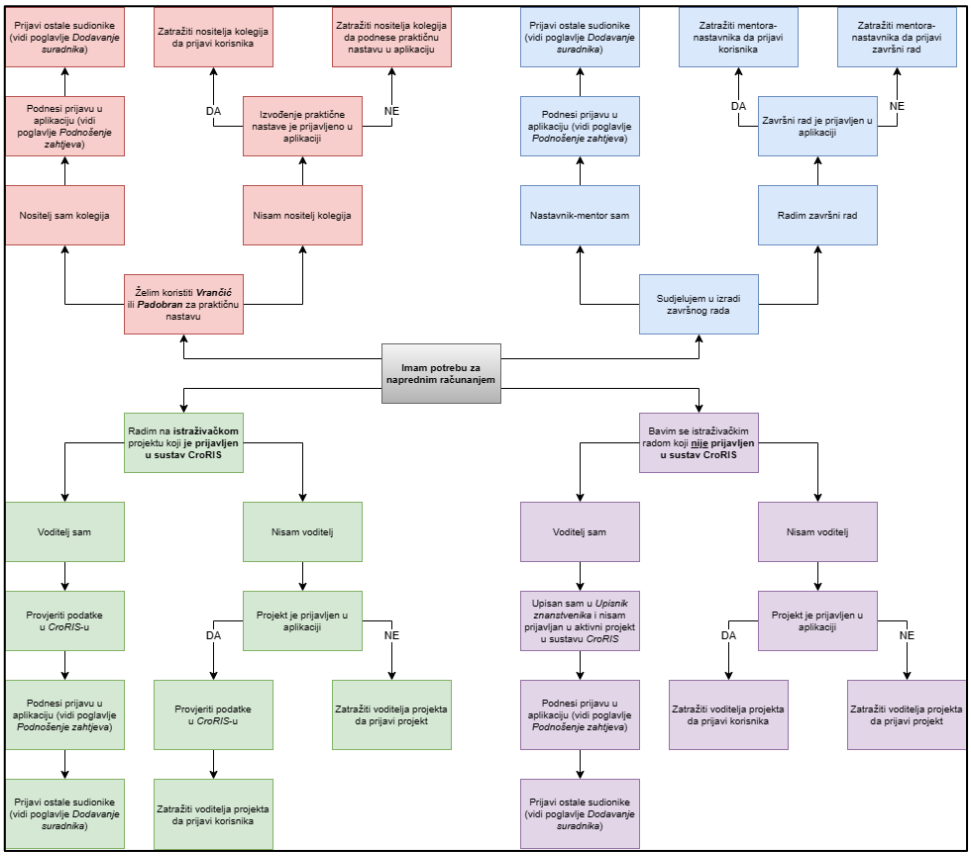
Zahtjeve za korištenje Supeka podnosi voditelj, bio to voditelj projekata ili nastavnik, tj. mentor u slučaju završnih, diplomskih ili specijalističkih i doktorskih radova. Odobreni zahtjev voditelja postaje njegov projekt na usluzi Napredno računanje na koji onda poziva i dodaje suradnike

Napomena

***Za inozemne korisnike.**

Inozemni korisnici mogu biti samo suradnici na projektu, ali ne i voditelji projekta.

Na Slici 6 prikazan je dijagram procesa prijave.



Slika 6: Proces prijave za korištenje usluga

5. Spajanje na Supek i Padobran

Trajanje poglavlja:

40 min

Po završetku ovoga poglavlja polaznik će moći:

- *stvoriti SSH ključeve u terminalu*
- *stvoriti SSH ključeve korištenjem aplikacije PuTTY Key Generator*
- *navesti adrese pristupnih čvorova na Supeku*
- *navesti adresu pristupnog čvora na Padobranu*
- *spojiti se korištenjem terminala*
- *spojiti se korištenjem aplikacije PuTTY.*

Za rad s naredbama *ssh* i *scp* u Windowsovu terminalu potrebno je instalirati značajku OpenSSH Client. Windows Subsystem for Linux (WSL) potpuno je funkcionalan Linux podsustav u Windowsu u kojem postoji izbor najpopularnijih distribucija Linuxa u kojima je rad identičan kao na Linuxu, a Windows i Linux međusobno su vidljivi.

Nakon što se potvrdi poziv na projekt, potrebno je izraditi i učitati (*uploadati*) SSH ključ. Na Supek i Padobran ne pristupa se pomoću lozinke, već isključivo SSH ključevima i to iz bilo koje mreže.

SSH ključevi sastoje se od dva dijela − privatnog i javnog ključa. Privatni ključ ne dijeli se ni s kime i on se nalazi na računalu s kojega se korisnik spaja na Supek. Javni će ključ korisnik učitati (*uploadati*) koristeći istu *web*-aplikaciju na https://computing.srce.hr/.

5.1. Kreiranje SSH ključeva u terminalu

U operacijskom sustavu Linux i macOS-u SSH ključevi se izrađuju korištenjem aplikacije ssh-keygen za kreiranje para ključeva, a aplikacija se poziva jednostavno upisom naredbe **`ssh-keygen`**.

Nakon upisa naredbe slijedi pitanje u koju se datoteku želi spremiti ključ. Ako se ostavi prazno i pritisne tipka [Enter], ključevi će biti spremljeni u zadane (*defaultne*) datoteke na lokaciji *$home/.ssh/* pod nazivima *id_rsa* i *id_rsa.pub*. Ako se ostavi zadani (*defaultni*) naziv datoteke i putanje, prilikom spajanja neće biti potrebno eksplicitno navoditi putanju datoteke u kojoj se nalazi privatni ključ.

Ključevi se moraju zaštititi lozinkom (tzv. *passphraseom*).

Nakon što se izrade ključevi, potrebno je u *web*-aplikaciju na https://computing.srce.hr/ učitati javni dio ključa ili sadržaj datoteke koja završava ekstenzijom.pub. Ako se koristi zadana (*defaultna*) lokacija, sadržaj se može vidjeti npr. naredbom **`cat $home/.ssh/id_rsa.pub`**.

Javni ključ učitava se na *web*-aplikaciji odabirom opcije *Javni ključevi* → ***Dodaj javni ključ***. Ključ se može nazvati po želji, a u polje **Javni ključ** kopira se sadržaj datoteke te zatim pritisne **Dodaj**.

5.2. Kreiranje SSH ključeva korištenjem PuTTY Key Generatora

U operacijskom sustavu MS Windows za izradu ključeva može se koristiti aplikacija PuTTY Key Generator. Jednostavno se pokrene aplikacija, ostavi već označena opcija vrste ključa **RSA** i pritisne **Generate**. Miš se pomiče unutar prozora dok se ključevi ne generiraju. Ključ se može, ali i ne mora, zaštititi upisivanjem lozinke.

Privatni i javni ključ spremaju se u dvije odvojene datoteke pritiskom na **Save public key** i **Save private key**.

Potrebno je i dalje ostaviti otvoren prozor Putty Key Generatora. Na vrhu prozora nalazi se **Public key for pasting into OpenSSH authorized_keys file**. Cijeli sadržaj potrebno je kopirati, a zatim, kao i u slučaju Linux OS-a, javni ključ učitati na *web*-aplikaciji.

5.3. Pristupni čvorovi i spajanje

Nakon što se učita ključ, potrebno je pričekati e-poštu s korisničkim imenom i potvrdu postavljanja javnog ključa prije samog spajanja. Iako je ključ učitan, nije valjan dok se ne primi obavijest.

Adrese pristupnih čvorova:

- Supek

  login-cpu.hpc.srce.hr

  login-gpu.hpc.srce.hr

- Padobran

  padobran.srce.hr

5.3.1. Spajanje korištenjem terminala

Ako se koristi terminal, može se spojiti naredbom:

`ssh korisničkoime@login-cpu.hpc.srce.hr` za spajanje na cpu pristupni čvor ili **`ssh korisničkoime@login-gpu.hpc.srce.hr`** za spajanje na gpu pristupni čvor. Uglavnom nije bitno na koji se spaja, radi se jednako i ima se pristup istim resursima. Bitno je jedino kada se kompiliraju aplikacije na pristupnom čvoru da se vidi GPU-ove.

Ako korisnik dobije grešku **Permission denied**, najvjerojatnije se privatni ključ ne nalazi na zadanoj (*defaultnoj*) lokaciji, pa je potrebno definirati točnu lokaciju opcijom **-i,** nakon koje slijedi putanja do ključa. U

ovom primjeru ključ je spremljen u datoteci **id_rsa**, na lokaciji **$home/.ssh/id_rsa** :

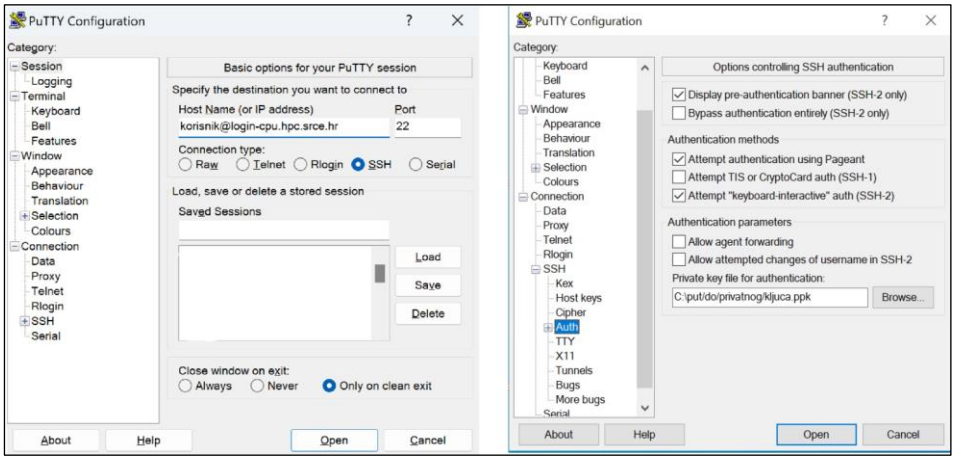
```
$ ssh -i ~/.ssh/id_rsa korisnik@login-cpu.hpc.srce.hr
```

5.3.2. Spajanje korištenjem aplikacije PuTTY

U slučaju da se koristi MS Windows može se koristiti alat PuTTY (https://www.putty.org/) uz upisivanje putanje privatnog ključa.

U postavkama pod **Session** potrebno je upisati pod **Host Name** korisničkoime@login-cpu-hpc.srce.hr ili login-gpu-hpc.srce.hr (Slika 7).

Također, potrebno je upisati putanju do SSH privatnog ključa u postavkama **Connection-SSH-Auth** polje **Private key file for authentication**.



Slika 7: Alat PuTTY

6. Kopiranje datoteka

Po završetku ovoga poglavlja polaznik će moći:

- *upravljati datotekama pomoću terminala na lokalnom računalu*
- *upravljati datotekama pomoću* File Manager *aplikacije*

Trajanje poglavlja:

30 min

6.1. Kopiranje datoteka – Terminal

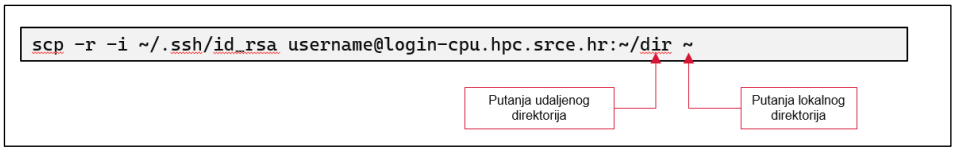
Kopiranje datoteka vrši se uz pomoć protokola SCP (engl. *Secure copy protocol*) koji je već dostupan na svim *Unix-like* operacijskim sustavima (Linux i macOS). Za prijenos podataka između osobnog računala i udaljenog poslužitelja koristi se naredba **scp**.

Za prijenos datoteka isključivo se koristi terminal na lokalnom računalu.

U primjeru na Slici 8 prikazan je prijenos podataka s lokalnog računala na udaljeni poslužitelj, u ovom slučaju Supek. Cijela naredba sastoji se od scp-a, pridodaje se flag -r koji označava rekurzivno kopiranje ili kopiranje cijelog direktorija, flagu -i daje se putanja do privatnog ključa koji se koristi za spajanje na udaljeni poslužitelj. Nakon toga slijedi putanja do samog direktorija/datoteke koja se želi prebaciti s lokalnog računala i za kraj vlastito korisničko ime s punim imenom pristupnog čvora nakon čega je potrebno upisati dvotočku i putanju do željene lokacije na udaljenom poslužitelju za spremanje direktorija/datoteke. U ovom slučaju to je **home**.

Slika 8: Prijenos podataka s lokalnog računala na udaljeni poslužitelj

Prijenos podataka s udaljenog poslužitelja na lokalno računalo je isto, samo se putanje zamijene (Slika 9), tj. prvo se unosi putanja udaljenog poslužitelja nakon čega slijedi putanja lokalnog računala, u ovom slučaju **~** ili **home**.



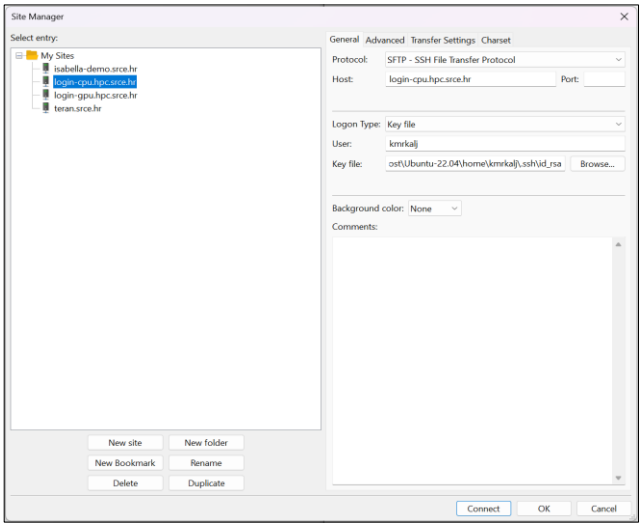
Slika 9: Prijenos podataka s udaljenog poslužitelja na lokalno računalo

6.2. Kopiranje datoteka − GUI (FileZilla)

Datoteke se mogu kopirati i s nekom *file manager* aplikacijom poput **Filezille**. Ista se može koristiti neovisno o operacijskom sustavu.

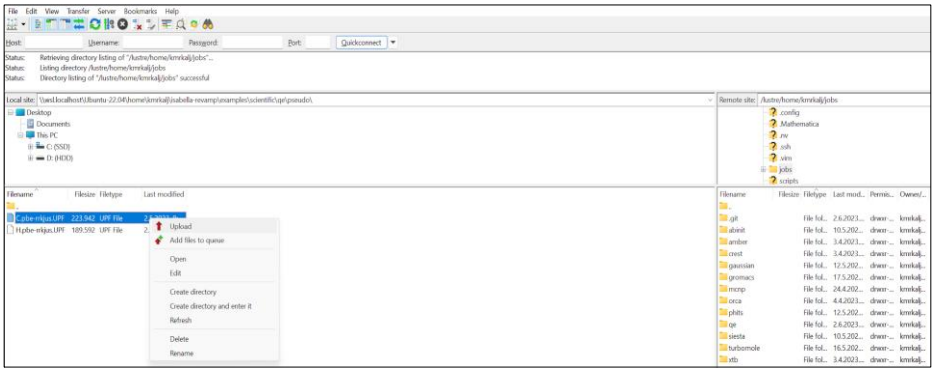
Prvo je potrebno dodati adrese *login* čvorova služeći se istim podacima kao što je objašnjeno u prethodnim poglavljima. Pritiskom na *File* → **Site Manager** potrebno je u prozoru postaviti SFTP − SSH File Transfer Protocol, a za *Host* → login-cpu.hpc.srce.hr ili login-gpu.hpc.srce.hr (Slika 10).

Logon Type bit će *Key file*, a za *username* potrebno je postaviti vlastiti dodijeljeni *username* na Supeku. Za *Key file* potrebno je postaviti putanju do privatnog SSH ključa na vlastitom računalu.



Slika 10: Prozor Site Manager aplikacije FileZilla

Kada se doda udaljeni poslužitelj i spoji se, prijenos datoteka poprilično je intuitivan (Slika 11). Na lijevoj strani nalaze se datoteke vlastitog lokalnog računala, a s desne strane nalaze se datoteke udaljenog poslužitelja ili Supeka. Datoteke se mogu prenositi opcijama *Download/Upload* ili jednostavno *Drag & Drop*.



Slika 11: Prijenos datoteka u aplikaciji FileZilla

7. Okolina Supeka i Padobrana

Trajanje poglavlja:

15 min

Po završetku ovoga poglavlja polaznik će moći:

- *pretraživati module i ključne riječi modula*
- *očistiti okolinu korištenjem naredbe* module purge
- *pretražiti module prema namjeni.*

HPC okolina inicijalno je prazna, pa se za jednostavno upravljanje softverskim okruženjem i resursima koriste *module* skripte. *Module* skripte su skripte koje definiraju okruženje za određeni softver, uključujući varijable okruženja kao što su putanje do izvršnih datoteka, knjižnica itd. Prednosti korištenja modula uključuju jednostavnost, koja omogućuje lako mijenjanje softverskog okruženja bez potrebe za ručnim podešavanjem varijabli okruženja, fleksibilnost, koja omogućava brz prelazak između različitih verzija softvera, i dosljednost jer svi korisnici imaju pristup istim postavkama smanjujući mogućnost grešaka zbog pogrešnih konfiguracija.

7.1. Modulefiles

Supek koristi Lua modules koji služi za postavljanje okoline za aplikacije, što omogućava instaliranje više različitih verzija određene aplikacije. Module je prije korištenja potrebno aktivirati, tj. *loadati*, u skriptama za podnošenje poslova.

Popis dostupnih modula može se pogledati upisivanjem naredbe **module avail**. Isto tako može se dobiti uvid u popis različitih verzija naredbom npr. **module avail gcc** za izlistavanje svih dostupnih verzija gcc prevodioca. Sama aktivacija vrši se s **module load**, a čišćenje cijele okoline izvodi se s **module purge**. Unutar poslova to nije potrebno jer se samim završetkom posla čisti okolina.

Tablica 8: Najčešće korištene naredbe

Naredba	Opis
module avail	Popis svih dostupnih modula
module spider <pojam>	Pretraga modula
module load <ime modula>	Učitavanje modula
module purge	Čišćenje (resetiranje) okoline

Postoji i općenitija naredba za pretraživanje ključnih riječi modula kada korisnik nije siguran što traži − **module spider**.

Naredba **module avail** izlistat će dostupne module koji se mogu učitati u vlastitu okolinu, dok će **module spider** prikazati detaljnije informacije odnosno hijerarhiju, ovisnosti i druge informacije o modulu.

Programska okruženja uglavnom su bitna za one korisnike koji razvijaju aplikacije ili kompiliraju. Različiti paketi prevoditelja (programska okruženja, engl. *programming environments*) dostupni su korištenjem **PrgEnv modula**. Ako se koriste gotove aplikacije, vjerojatno neće biti potrebni.

Moduli su većinom grupirani prema namjeni, tako da ako se upiše **module avail scientific**, izlistat će se samo znanstvene aplikacije, a **module avail libs** za nesistemske knjižnice koje su naknadno dopremljene i **utils** za razne alate.

Učitavanjem modula znanstvenih aplikacija, osim putanji i varijabli, automatski se postavljaju i ovisnosti tih modula, tako da nije potrebno učitavati zasebno sve module.

8. Sustav za upravljanje poslovima

Trajanje poglavlja:

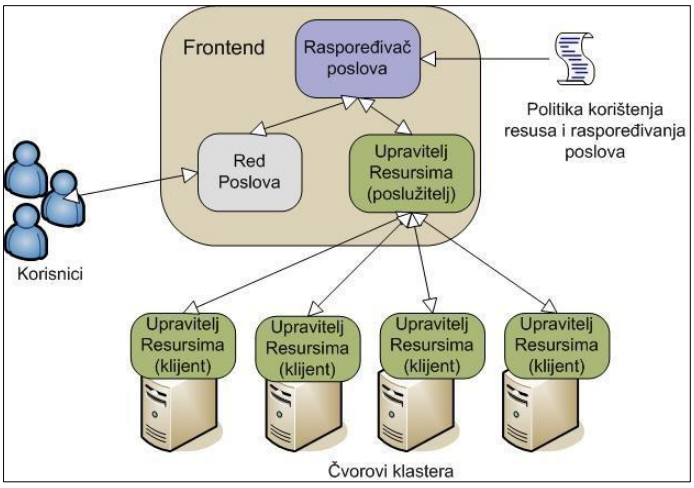
45 min

Po završetku ovoga poglavlja polaznik će moći:

- *prepoznati PBS varijable okoline*
- *pregledati zauzeće superračunala*
- *nadzirati i upravljati poslovima*
- *opisivati poslove korištenjem različitih PBS opcija*
- *razumjeti koncept* chunka.

Supek i Padobran koriste sustav za upravljanje poslovima PBS (engl. *Portable Batch System*).

Računalni poslovi podnose se sustavu za upravljanje poslovima. Posao se definira unutar skripte koja se podnosi sa zahtjevom za računalnim resursima i s maksimalnim vremenom izvođenja. Nakon ispravnog podnošenja skripte sustav stavlja podneseni posao u red za izvođenje, koji se rangira silazno po kriteriju prioriteta. Sustav kontinuirano nadgleda dostupne resurse te, u trenutku kada ima dostupne slobodne resurse koje posao zahtijeva, sustav rezervira te resurse i pokreće prvi posao u redu izvođenja na njima.



Slika 12: Sustav za raspoređivanje poslova

Postoji više vrsta poslova. *Batch poslovi* ne zahtijevaju naknadni *input* korisnika, već se nakon pokretanja sami izvode. Drugi tip su *interaktivni* koji traže *input* korisnika za daljnje izvođenje. Na Supeku je moguće podnijeti i interaktivne poslove, tj. raditi u interaktivnom modu.

Po broju jezgri koje koriste u svojem izračunu poslovi se dijele na serijske i paralelne. Serijski poslovi koriste jednu jezgru i naredbe se izvršavaju jedna za drugom, a na paralelnom se naredbe, tj. procesi, izvršavaju jedna paralelno s drugom, što znači da se poslovi i brže izvršavaju. Idealno bi se na klasterima izvršavali samo paralelni poslovi,

no dozvoljavaju se i serijski tipovi poslova na za to predviđenim redovima.

Sustav za upravljanje poslovima primarno raspoređuje poslove unutar klastera, tj. obavlja raspodjelu poslova na slobodne jezgre i grafičke kartice.

Kad u skripti za podnošenje korisnik zahtijeva resurse za podnošenje posla (npr. broj jezgri kao najbitniji), sustav neće korisnika trenutačno prebaciti „negdje“ na radni čvor, već analizira koliko je slobodnih jezgri na klasteru i razmišlja gdje smjesti korisnika a da nitko nikome ne smeta.

Kako bi se osiguralo što pravednije korištenje klastera za sve korisnike, u određivanju prioriteta poslova koristi se *fair-share* politika − svi projekti na klasteru su ravnopravni.

Bitan faktor prioriteta u kontekstu korištenja klastera je umnožak broja traženih procesorskih jezgara i vremena koje te jezgre provedu zauzete, tj. *CPU walltime*. Ovakva politika ne uzima u obzir iskorištenost zatraženih jezgri, već gleda jesu li one dostupne drugim korisnicima. Korisnik je zadužen osigurati da njegov posao koristi dodijeljene jezgre u što većem postotku.

Opcija **strict ordering** osigurava izvođenje prvo poslova višeg prioriteta i tako onemogućuje „provlačenje“ poslova manjeg prioriteta jer bi suprotno rezultiralo time da veliki poslovi nikad ne mogu doći na red.

Međutim, kod pokretanja poslova prvo će se popunjavati poslovi koji traže testni red, a zatim poslovi koji traže produkcijske redove, nevezano na njegov izračunati prioritet. Restrikcije uvedene na testnu okolinu onemogućavaju zloupotrebu ove opcije.

Supek:

P = 10 * (ncpus/6656) + 1000 * (1 - FS_p) + (Et/86400)

Padobran:

P = 10 * (ncpus/128) + 1000 * (1 - FS_p) + (Et/86400)

Gore u okviru navedene su jednadžbe za izračun prioriteta na Supeku i Padobranu.

Na vrijednost izračunatog prioriteta utječu broj traženih jezgara (**ncpus**) − veći broj, veći prioritet, *fairshare* udio (**FS_p**), tj. udio *cpu walltimea* projekta ukupnom *CPU walltimeu* klastera, te vrijeme provedeno u redu čekanja (**Et**) − što duže se čeka raste prioritet.

Prvi udio u formuli je onaj koji povećava prioritet velikim poslovima kako bi ih se dodatno istaknulo u odnosu na male poslove. Drugi faktor povećava prioritet poslovima čiji je *fairshare* udio malen, odnosno koji nisu puno koristili klaster. Zadnji faktor povećava prioritet poslovima koji su proveli dugo vremena u redu za čekanje.

8.1. Varijable okoline PBS-a

PBS varijable definira PBS sustav automatski na temelju opcija definiranih u PBS skripti. Neke od uobičajenih PBS varijabli prikazane su u Tablici 9.

Tablica 9: PBS varijable okoline

Varijabla	Opis
PBS_JOBID	Brojčani identifikator posla, generiran kao *output* naredbe **qsub**
PBS_JOBNAME	Ime posla
PBS_NODEFILE	Popis radnih čvorova (važan za pokretanje MPI procesa)
PBS_O_WORKDIR	Direktorij u kojem je podnesen posao, odnosno u kojem je pozvana naredba **qsub**
NCPUS	Broj zatraženih CPU jezgara (odgovara vrijednosti iz opcije **ncpus** iz zaglavlja PBS skripte)
OMP_NUM_THREADS	OpenMP varijabla koju PBS izvozi u okolinu, a koja je jednaka vrijednosti opcije **ncpus** iz zaglavlja PBS skripte.
TMPDIR	Putanja do privremenog direktorija (korisna za aplikacije koje generiraju velik broj nepotrebnih privremenih datoteka)

8.2. Pregled zauzeća klastera

Uz pomoć naredbe **pbsnodes** mogu se vidjeti statusi pojedinih čvorova. Na slici 13 vidi se primjer ispisa naredbom **pbsnodes -aSj**, koja detaljno ispisuje imena čvorova, njihovo stanje, broj poslova koji su gore, broj poslova koji se vrte trenutačno na njima, slobodna i ukupna memorija, slobodan i ukupan broj jezgara. Predzadnji stupac prikazuje broj dostupnih i svih GPU-ova, a zadnji stupac prikazuje koji se poslovi izvode na tom stroju.

```
                                mem      ncpus  nmics  ngpus
vnode          state      njobs run susp    f/t      f/t    f/t    f/t  jobs
---------- ---------- ------ --- ---- ------ -------- ------ ------ ------ --------
x8000c0s0b0n0   free        6   6    0   9gb/241gb  34/128   0/0    0/0 126812,128130,126812,128130,125901,128130
x8000c0s0b0n1   free        2   2    0  25gb/241gb  60/128   0/0    0/0 128320,127230
x8000c0s0b1n0   free        2   2    0  34gb/241gb  69/128   0/0    0/0 125901,128320
x8000c0s0b1n1   free        5   5    0   7gb/241gb  60/128   0/0    0/0 128130,128270,128088,128270,125901
x8000c0s1b0n0   free        4   4    0  69gb/241gb  64/128   0/0    0/0 128160,128270,128167,128230
x8000c0s1b0n1   free        7   7    0  13gb/241gb  35/128   0/0    0/0 126672,128130,126672,128130,125901,128130,126672
x8000c0s1b1n0   free        4   4    0   8gb/241gb  48/128   0/0    0/0 128147,128270,128147,128270
```

Slika 13: Primjer ispisa naredbe pbsnodes -aSj

Tablica 10: Opis parametara ispisa naredbe pbsnodes -aSj

Najvažnije informacije o poslovima	
vnode	Radni čvor
state	Stanje radnog čvora (slobodan, zauzet, ugašen...)
njobs	Ukupan broj poslova
run	Broj poslova koji se izvode
susp	Broj poslova koji su suspendirani
mem f/t	Slobodna / ukupna radna memorija
ncpus f/t	Broj slobodnih / ukupnih CPU jezgara
nmics f/t	Broj slobodnih / ukupnih MIC-a (nije primjenjivo)
ngpus f/t	Broj slobodnih / ukupnih GPU-ova
jobs	ID-ovi poslova koji se izvode na tom radnom čvoru

8.3. Pregled poslova

Naredba koja se koristi za praćenje poslova je **qstat**. Sama naredba bez argumenata ispisuje sve poslove svih korisnika koji se izvode na Supeku.

```
Job id              Name             User             Time Use S Queue
----------------------  ---------------- ----------------  -------- - -----
120513[].x3000c0s25b0* run-14D-66-15.s* user_1                 0 B cpu-single
124590.x3000c0s25b0n0  singlegpu_run.p* user_2                 0 Q gpu
125639.x3000c0s25b0n0  zh8243_7         user_3          2178:51* R cpu
125729.x3000c0s25b0n0  bp_8000-8500     user_3           612:36:* R cpu
125808.x3000c0s25b0n0  be_pea          user_4          1243:25* R gpu
```

Slika 14 Ispis naredbe qstat

Prvi stupac sadrži **id_posla** koji je PBS dodijelio tom poslu i ime čvora s kojeg je pokrenut posao, drugi stupac sadrži samo ime posla koje definira korisnik, treći stupac prikazuje ime korisnika koji je pokrenuo posao, a četvrti stupac sadrži vrijeme izvođenja posla, tj. CPU vrijeme. Peti stupac prikazuje stanje posla − slovo podrazumijeva stanje. Najčešće su to **R** (*running*) za poslove koji se izvode i **Q** (*queued*) za posao koji je u redu čekanja. Zadnji stupac sadrži red u kojem se posao izvodi.

Ako se naredba **qstat** proširi dodatnim opcijama **-f**, dobiva se detaljniji prikaz, *flagom* **-x** definiraju se već gotovi poslovi, a *flag* **-w** daje pregledniji format.

U ispisu naredbe **qstat -fwx <job id>** mogu se vidjeti najvažnije informacije o poslu (Tablica 11), resursi, određene putanje za posao, na kojima se čvorovima posao izvodio/izvodi, *walltime* itd.

Tablica 11: Najvažnije informacije o poslovima u ispisu naredbe qstat -fxw

Najvažnije informacije o poslovima	
Job_Name	Ime posla
resources_used.mem	Memorijski *peak* posla
resources_used.ncpus	Količina traženih procesorskih jezgara
resources_used.walltime	Trajanje posla
job_state	Stanje posla (u čekanju, u izvođenju, završen...)
Queue	Red u kojem se posao izvodio
exec_vnode	Radni čvor na kojem se posao izvodio

Postoji mnogo drugih opcija naredbe **qstat**, a jedna od korisnijih je **-u** ako korisnika zanimaju samo poslovi nekog drugog korisnika (**qstat -u $USER**).

8.4. Opisivanje poslova

Svi poslovi koji se podnose opisuju se unutar takozvane **PBS skripte**. U suštini to je *shell* skripta − u zaglavlju se navodi željeni interpreter, a zadano (po *defaultu*) je to *bash*. Ostatak skripte sastoji se od naredbi koje se izvršavaju slijedno, jedna za drugom.

Poslovi se opisuju oz pomoć opcija, a one najčešće korištene prikazane su u Tablici 12.

Tablica 12: Opcije opisivanja poslova

Opcija	Primjer argumenta	Značenje
-N	homo_volans	Postavljanje imena posla „homo_volans”
-q	gpu	Specificiranje reda posla „gpu”
-l	select=2:ngpus=1:ncpus=4:mem=8gb	Traženje 2 *chunka*, svaki po 1 GPU, 4 CPU jezgre i 8 GiB memorije
-l	place=pack	Smještanje svih *chunkova* na isti čvor
-o	izlaz.log	Definiranje imena/putanje u koje se sprema standardni izlaz
-e	greska.log	Definiranje imena/putanje u koje se sprema standardna greška
-M	faust.vrancic@padobran.hr	Postavljanje primatelja e-pošte
-m	be	Uvjet slanja e-pošte na gore navedenu adresu (npr. **b**eggining, **e**nd)
-j	oe	Povezivanje standardnog izlaza i greške u istu datoteku izlaza (ostavlja grešku praznom)

8.5. Kontrola korištenih resursa

Napomena

Ako definirate poslove koristeći **ncpus bez opcije select**, poželjno je definirati i količinu memorije jer će u suprotnom dostupna radna memorija iznositi 1800 MiB.

Supek upotrebljava tehnologiju cgroups, tj. *kernel* značajku koja uvjetno rečeno „ograničava“ korištenje aplikacije na način da aplikacija vidi isključivo resurse koje je korisnik zatražio. Korisnik ne može koristiti resurse koje nije zatražio. Osim za kontrolu korištenja procesora, tehnologija cgroups postavljena je tako da kontrolira i potrošnju memorije. To znači da su poslovi koje korisnik pokreće ograničeni na traženu količinu memorije. Ako posao pokuša iskoristiti više memorije nego što je zatraženo u opisu posla, sustav će prekinuti taj posao i u izlaznu *error* datoteku zapisati *error* poruku: *Cgroup mem limit exceeded: oom-kill:....* Kod svakog posla ova poruka bit će malo drugačija jer sadrži podatke kao što su UID (jedinstvena brojčana oznaka korisnika), PID (brojčana oznaka procesa koji je prekinut), JOB_ID (ID posla koji dodjeljuje PBS).

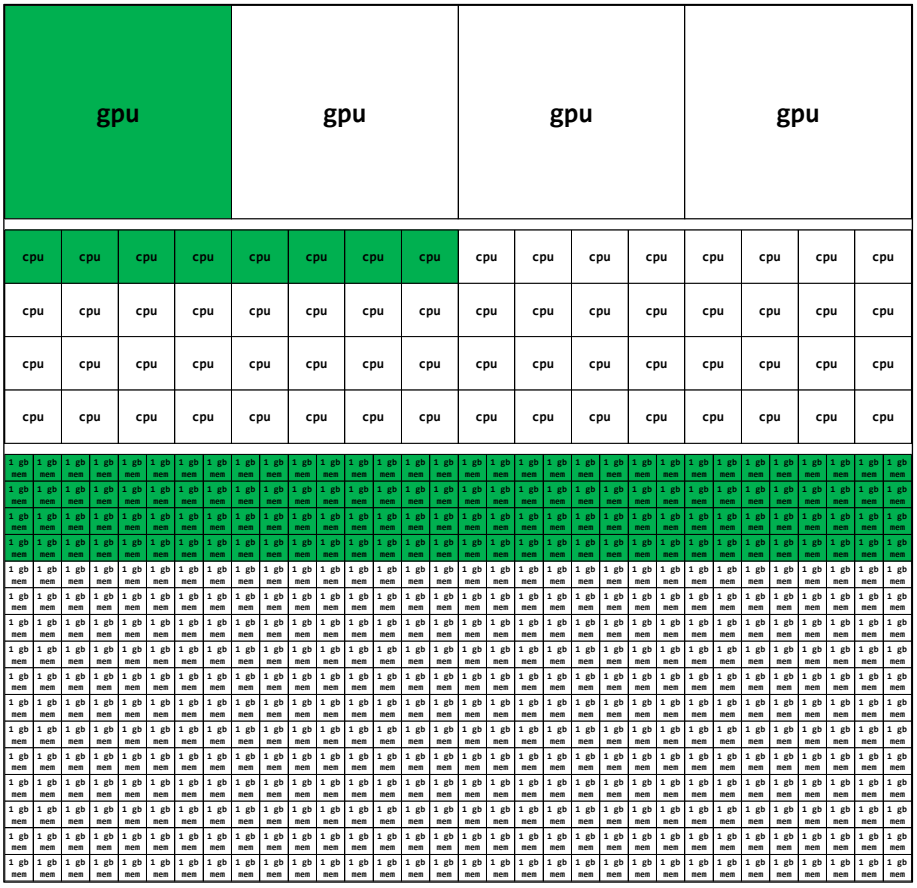
8.6. Chunk

Chunk (dio čvora) je PBS-ov koncept koji predstavlja set resursa koji se dodjeljuju nekom poslu. Unutar jednog takvog *chunka* (koji mora biti unutar granica jednog čvora), definira se broj MPI procesa, fizičkih jezgri, grafičkih procesora i radne memorije. Broj takvih *chunkova* definira se opcijom ***select***.

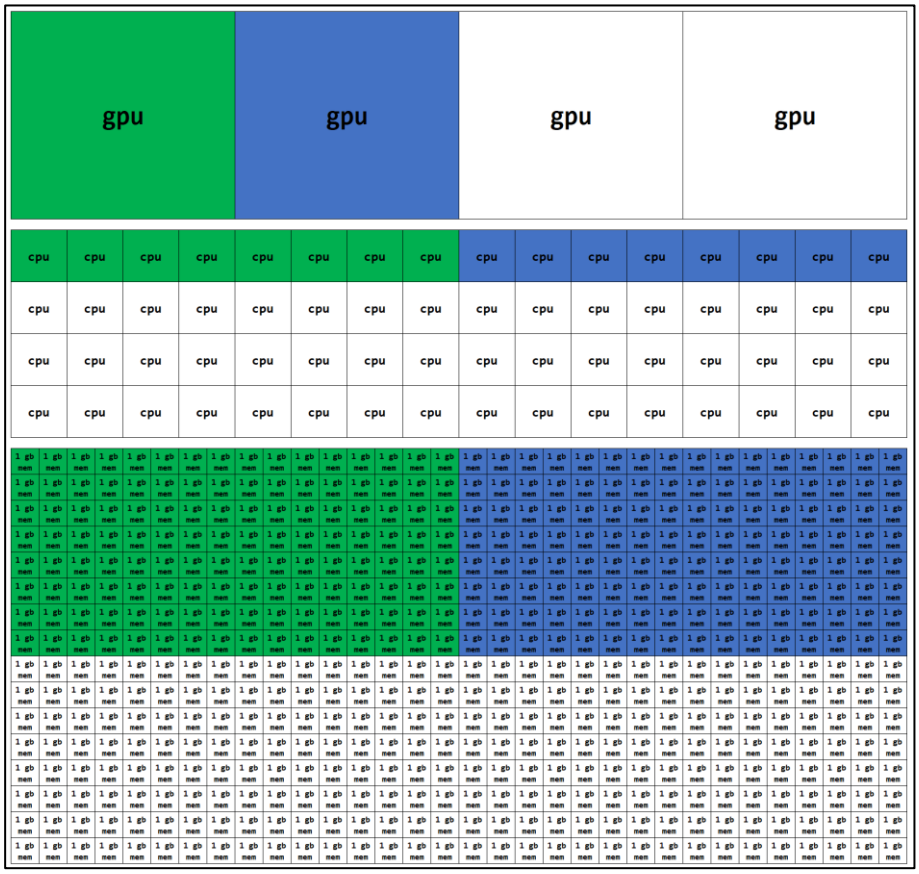
Broj MPI procesa i fizičkih jezgri je zadano (*defaultno*) 1!

Korisnik opcijom ***place*** može odabrati gdje želi smjestiti *chunkove*. Ako odabere ***pack***, svi će *chunkovi* biti uzeti s istog čvora. Ako odabere ***scatter***, svaki *chunk* bit će uzet s drugog čvora. Ako odabere ***free***, prepušta se PBS-u gdje će smjestiti *chunkove*, ovisno o raspoloživosti.

U nastavku je prikazano kako se resursi raspodjeljuju kada korisnik traži 1 *chunk* u odnosu na to što se događa kada traži 2 *chunka*.



Slika 15: Raspodjela resursa za #PBS -l ***select=1****:ngpus=1:ncpus=8:mem=128gb*



Slika 16: Raspodjela resursa za #PBS -l ***select=2****:ngpus=1:ncpus=8:mem=128gb #PBS -l place=pack*

8.7. Podnošenje i prekidanje poslova – qsub i qdel

Podnošenje poslova na Supeku provodi se kroz komandolinijski alat **qsub** koji dolazi s PBS-om. Sintaksa za izvođenje može se provoditi na dva načina: napisati skriptu i željene parametre opisati u skripti ili direktno u nastavku opcije qsub.

Opcije izvođenja iste su za opisivanje posla u skripti ili izvan nje.

Output uspješno podnesenog posla vraća ID posla koji se koristi dalje za nadzor izvođenja.

Ako korisnik pošalje posao u red čekanja, a zaboravio je nešto napisati ili je krivo napisao, može potpuno zaustaviti posao ili ga maknuti iz reda čekanja pomoću naredbe **qdel**.

Za poslove koji se zaglave u redu čekanja mora se koristiti prisilno zaustavljanje opcijom **force:**

```
qdel -W force -x <ID_posla>
```

9. MPI paralelizacija

Po završetku ovoga poglavlja polaznik će moći:

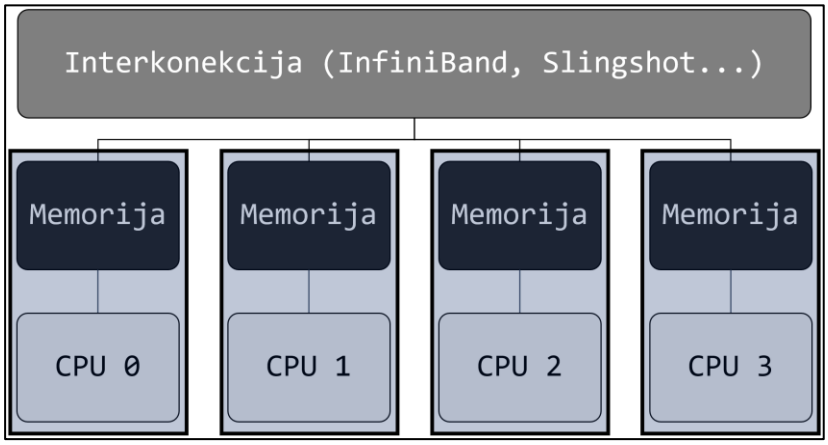
- *razlikovati MPI i OpenMP paralelizaciju*
- *razumjeti sadržaj hostfilea.*

Trajanje poglavlja:

45 min

U računarstvu visokih performansi dvije su osnovne tehnologije za implementaciju paralelizacije kod programskih jezika kao što su C/C++ i Fortran. To su **MPI** i **OpenMP**.

MPI (*Message Passing Interface*) je standard koji definira razmjenjivanje poruka između paralelnih procesa.



Slika 17: Shematski prikaz MPI paralelizacije

Aplikacije koje koriste MPI paralelizaciju mogu širiti svoje procese izvan jednog računalnog čvora. Za takve aplikacije kaže se da rade s raspodijeljenom (distribuiranom) memorijom budući da svaki proces može dobiti svoj dio fizičke (RAM) memorije.

MPI program pokreće procese koji paralelno izvršavaju program na zasebnim podacima, svaki radi u vlastitom memorijskom prostoru izolirano od ostalih, a međusobno komuniciraju prosljeđivanjem poruka (engl. *message passing*).

U kontekstu računalnih klastera prednost MPI-ja je osim što omogućuje pristup većoj količini radne memorije također omogućuje pristup većem broju procesorskih jezgri, dakle i većoj računalnoj snazi.

Ako je složenost određenog programa tolika da podaci kojima barata ne stanu u radnu memoriju jednog čvora, MPI je dobar izbor jer otvara mogućnost pokretanja MPI-procesa na više radnih čvorova. Tako se maksimalno iskorištava dostupna memorija.

„**mpiexec**“ (ili „**mpirun**“ na drugim sustavima) su naredbe, odnosno alati za pokretanje MPI programa (tzv. kopija) na različitim procesorskim jezgrama.

9.1. MPI nodefile (hostfile, machinefile)

Koncept *chunka*, objašnjen u poglavlju 8.6 postaje bitan kod pokretanja MPI aplikacija.

PBS izvozi posebnu datoteku PBS_NODEFILE (generalno poznatiji kao MPI *machinefile* ili *hostfile*) unutar koje u svaku liniju zapisuje čvor na koji će smjestiti MPI proces. Pokretačka naredba „**mpiexec**“ će za svaku takvu liniju pokrenuti po jedan MPI proces, na tom čvoru.

Sadržaj te datoteke definiran je kombinacijom opcija **select**, **mpiprocs** i **place**.

```
#PBS -l select=4:mpiprocs=2
#PBS -l place=scatter
$PBS_NODEFILE
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s1b0n0.hsn.hpc.srce.hr
x8000c0s1b0n0.hsn.hpc.srce.hr
x8000c0s1b0n1.hsn.hpc.srce.hr
x8000c0s1b0n1.hsn.hpc.srce.hr
```

```
#PBS -l select=4:mpiprocs=2
#PBS -l place=pack
$PBS_NODEFILE
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
x8000c0s0b1n0.hsn.hpc.srce.hr
```

```
#PBS -l select=8[:mpiprocs=1]
#PBS -l place=pack
$PBS_NODEFILE
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
x8000c0s0b0n1.hsn.hpc.srce.hr
```

Slika 18: Različiti načini traženja 8 procesa

Ako se pogleda primjer lijevo, korisnik je tražio 4 *chunka*, od kojih svaki ima po 2 MPI procesa i svaki *chunk* dolazi s drugog čvora. Iz tog razloga vide se po dva ista zapisa za jedan čvor. Sve skupa sadrži 8 zapisa te će **`mpiexec`** pokrenuti 8 MPI procesa.

U srednjem primjeru korisnik je tražio također 4 *chunka* i 2 MPI procesa za svaki, međutim ovaj put svi dolaze s istog čvora, pa ćemo imati 8 istih zapisa.

Na desnom primjeru vidimo 8 *chunkova* koji svaki ima po 1 MPI proces. Također, svi dolaze s istog čvora, pa je sadržaj isti kao i prethodni – 8 zapisa i 8 MPI procesa.

9.2. Podnošenje MPI poslova

```
#PBS -l select=4:mpiprocs=2:ncpus=2:mem=5000mb
#PBS -l place=scatter
```

U ovom primjeru korisnik je tražio 4 *chunka* s različitih čvorova (**place=scatter**).

Svaki će imati 2 MPI procesa, 2 fizičke jezgre i 5000 MiB memorije.

Kod MPI *only* aplikacija poželjno je da broj MPI procesa odgovora broju fizičkih jezgara tako da se svaka jezgra ima svoj MPI proces.

9.3. MPI implementacije − Cray MPICH

Postoji više implementacija MPI-ja od kojih su najpopularniji **Open MPI**, **MVAPICH** i **MPICH**.

Na računalnom klasteru Supek preferira se korištenje Crayeve sistemske implementacije MPICH-a i omogućava puno iskorištavanje postojeće superbrze Crayeve Slingshot veze.

Ona je također implementirana u većini dopremljenih MPI aplikacija − a to se može provjeriti pomoću naredbe **ldd** uz aplikaciju, tj. *binary* kao argument. Tada će biti vidljivo da je aplikacija povezana na **libmpi.so** dijeljenu knjižnicu, u **mpich** direktorijima.

```
$ ldd aplikacija
…
libmpi.so.40 => /apps/utils/openmpi/5.0.1/gnu/11.2.1/lib/libmpi.so.40 (0x0000147569e29000)
…
```

Postoje neke aplikacije koje se ne mogu kompilirati s Crayevim MPICH-em, primjerice ORCA ili BAGEL, tako da one koriste Open MPI.

9.4. MPI implementacije − Open MPI v5.0

Mana starijih verzija Open MPI-ja je što ne mogu u potpunosti iskoristiti Crayevu Slingshot mrežu, što znači da je komunikacija među procesima s različitih čvorova sporija.

Međutim, novija Open MPI verzija 5 nema to ograničenje. Dopremljena je na Supek jer je sistemski Crayev MPICH imao problema prilikom korištenja na više različitih čvorova. Naime, samo su se 3 posla na nekom čvoru mogla proširiti izvan tog čvora. Kada bi došao 4. i pokušao isto – prekinuo bi se.

Ako korisnik sam kompilira neku aplikaciju Open MPI-jem, korištenjem naredbe **ldd** na glavni *binary* može vidjeti je li povezana s dobrom MPI knjižnicom.

9.5. OpenMP paralelizacija

OpenMP (engl. *Open Multi-Processing*) je tehnologija paralelizacije koja implementira koncept dijeljene memorije, oblik memorije kakav danas nalazimo i na osobnim računalima te pametnim telefonima. Dijeljena memorija znači da se aplikacija izvršava na jednom izoliranom sustavu prilikom čega koristi više jezgri koristeći isti memorijski prostor, odnosno dijeli radnu memoriju.

Za aplikacije koje koristi isključivo OpenMP kažemo da se „ne znaju“ širiti izvan radnog čvora.

OpenMP koristi paralelizaciju na razini dretvi (engl. *threads*). Inicijalni program u određenim segmentima (koje je definirao programer) razdvaja se na više paralelnih niti, pri čemu svaka koristi (zajedničku) radnu

memoriju s računala na kojem je i pokrenuta. Svaka takva paralelna nit ima pristup svim podacima u radnoj memoriji (Slika 19).

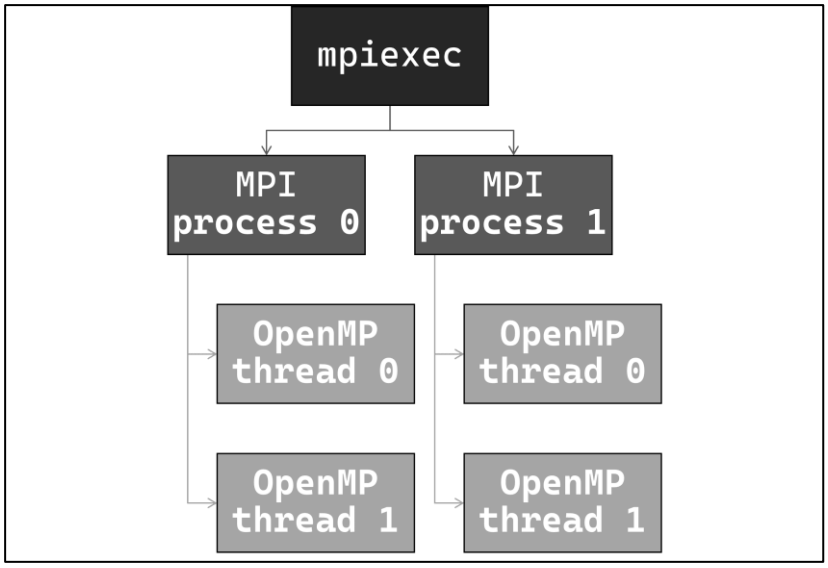
Slika 19: Shematski prikaz OpenMP paralelizacije

Neovisno o PBS sustavu, broj procesorskih jezgri na koje će se takva aplikacija smjestiti definiran je OpenMP varijablom (dakle, ne PBS varijablom, već OpenMP varijablom) OMP_NUM_THREADS.

Kako bi korisniku olakšao podnošenje poslova, PBS sam definira OMP_NUM_THREADS vrijednost na temelju vrijednosti definirane u PBS skripti, odnosno na temelju vrijednosti opcije ncpus.

9.6. Hibridna (MPI + Open MP) paralelizacija

Aplikacije koje koriste hibridnu paralelizaciju koriste oba prethodno navedena tipa paralelizacije odjednom. Funkcionira tako da se svaki MPI proces stvara određen broj OpenMP *threadova*. Svaki taj OpenMP *thread* trebali bismo smjestiti na vlastiti fizičku jezgru.



Slika 20: Shematski prikaz / stablo hibridne paralelizacije

Aplikacije koje paraleliziramo ovom tehnikom često postavljamo na način da svaki MPI proces sjedne na svoj čvor, a onda se OpenMP paralelizacija izvodi dalje unutar tog svog čvora.

10. Korištenje grafičkih procesora

Po završetku ovoga poglavlja polaznik će moći:

- *razumjeti kako prilagoditi PBS skriptu u svrhu dobivanja pristupa grafičkim procesorima.*

Trajanje poglavlja:

10 min

Ako korisnička aplikacija podržava izvođenje na grafičkim procesorima, nužno je koristiti red poslova „gpu“ kako bi korisnik dobio pristup računalnim čvorovima koji posjeduju grafičke procesore. Osim toga, potrebno je definirati broj grafičkih procesora opcijom **ngpus**.

Definiranjem opcije ngpus u korisničku će se okolinu izvesti varijabla **CUDA_VISIBLE_DEVICES** koja će aplikaciji učiniti grafičke procesore vidljivima te joj dopremiti njihove identifikacije.

Naravno, korisnik može dodatno tražiti i CPU procesore uz GPU procesore.

11. Vježba − Primjeri PBS skripti i korištenja aplikacija

Po završetku ovoga poglavlja polaznik će moći:

- *samostalno napisati PBS skriptu za pokretanje poslova*
- *prilagoditi postojeće PBS skripte prema svojim potrebama.*

Trajanje poglavlja:

120 min

PBS skripta sastoji se od 3 dijela koji moraju biti navedeni isključivo ovim slijedom:

1. Interpreter (shebang)

Shebang je prva linija u skripti koja specificira koji interpreter treba koristiti za izvršavanje skripte. Na Unix/Linux sustavima, shebang linija započinje znakovima #! nakon čega slijedi puna putanja do interpretera. Na primjer, #!/bin/bash specificira da će skripta koristiti Bash shell kao interpreter. Ova linija je ključna jer operacijskom sustavu govori kako izvršiti skriptu. Ako shebang nije pravilno definiran, sustav možda neće znati kako interpretirati naredbe u skripti, što može uzrokovati pogreške.

2. PBS direktive

PBS direktive su linije unutar skripte koje se koriste za konfiguriranje i kontrolu ponašanja posla unutar sustava za raspoređivanje poslova kao što je PBS. Direktive obično počinju s #PBS i specificiraju različite opcije kao što su naziv posla, resursi potrebni za posao (CPU jezgre, memorija, trajanje), red u koji posao treba biti poslan i druge karakteristike posla.

3. Naredbe unutar posla

Naredbe unutar posla su stvarne komande koje se izvršavaju kada je posao pokrenut. Ove naredbe nalaze se nakon PBS direktiva i uključuju sve što se treba izvesti u okviru posla − pokretanje aplikacija, izvođenje skripti, premještanje datoteka itd. Ove naredbe pripremaju radno okruženje, učitavaju potrebne module i izvršavaju program koji korisnik želi pokrenuti. Izvršenje naredbi može uključivati bilo što od osnovnih shell komandi do složenih skripti i aplikacija.

Svaki od ovih dijelova ima ključnu ulogu u pravilnom funkcioniranju PBS skripte osiguravajući da posao bude ispravno konfiguriran, raspoređen i izvršen na računalnom sustavu.

11.1. Općeniti primjeri

11.1.1. Primjer red.sh

Nakon što se (pomoću naredbe cd) korisnik pozicionira u mapu (engl. *folder*) u kojoj se nalazi primjer (**~/tutorial/1_general/red.sh),** može ga ispisati korištenjem naredbe **cat** kako bi vidio sadržaj skripte.

```
1  #!/bin/bash
2
3  #PBS -q cpu-radionica
4  #PBS -l select=1:ncpus=1
5
6  echo "Ovo je primjer s redom posla."
```

Prva linija na primjeru iznad je *interpreter* koji će se koristiti za prijevod PBS skripte.

Treća linija karakteristična je za PBS i njome se definira red u kojem želite da se posao izvodi, a to je red poslova **cpu-radionica**.

Šesta je linija konkretna „naredba“ koja će se izvršiti na radnim čvorovima, a u ovom slučaju to je samo ispis rečenice.

Skripta se podnosi s naredbom **qsub**, a kao izlaz dobit će se **.o** i **.e datoteke**. To su standardni izlazi, odnosno ono što bi se inače ispisalo na zaslon da se naredbu pokrenula klasično u terminalu. Točnije, u .o datoteci nalazi se standardni izlaz, a u .e datoteci standardna greška. Budući da se u ovom slučaju ne predviđa greška, .e datoteka trebala bi biti prazna.

Budući da u priručniku još nije obrađeno definiranje imena posla, posao će se zvati kao skripta, a po njemu će biti nazvani i *outputi*.

11.1.2. Primjer ime.sh

U sljedećoj skripti definirano je ime posla kao **moje_ime**.

```
1  #!/bin/bash
2
3  #PBS -q cpu-radionica
4  #PBS -l select=1:ncpus=1
5  #PBS -N moje_ime
6
7  echo "Ovo je primjer s imenom posla."
```

Ime posla će se prikazivati u ispisu naredbe **qstat** i bit će dodijeljeno imenima standardnog izlaza i pogreške.

Nakon što se skripta podnese, može se vidjeti da su se sada .o i .e datoteke nazvale po imenu posla, odnosno moje_ime. Ne očekuje se greška, pa je .e datoteka prazna.

11.1.3. Primjer izlazi.sh

Izlazi se mogu i strogo definirati, ne moraju se zvati po imenu posla (odnosno imenu skripte). Mogu se zvati proizvoljno, a ta imena definiraju se pomoću parametara -o (za standardni izlaz) i -e (za standardnu grešku).

```
1   #!/bin/bash
2
3   #PBS -q cpu-radionica
4   #PBS -l select=1:ncpus=1
5   #PBS -o izlaz.log
6   #PBS -e greska.log
7
8   cd $PBS_O_WORKDIR
9
10  cat tekst.txt
11  cat drugi-tekst.txt
```

U ovom slučaju to su **izlaz.log** i **greska.log**.

Skripta ispisuje sadržaje dviju datoteka. Jedna od njih postoji (tekst.txt), a druga ne postoji (drugi-tekst.txt).

Kada se piše PBS skripta, PBS ju čita na način kao da se korisnik nalazi u svom *home* direktoriju. To znači da se sve relativne putanje koje se nalaze u korisničkoj skripti čitaju od *home* direktorija kao polazišnog.

Međutim, direktorij u kojem je pokrenuta naredba **qsub** je „radni direktorij“. Na tom su mjestu najčešće korisnički *input fileovi* i skripta, pa je preporuka i pisati skriptu iz perspektive kao da se korisnik nalazi u njemu. Iz tog razloga u ovom primjeru uvrštena je naredba **cd** u $PBS_O_WORKDIR, a onda će se druge dvije naredbe odnositi na datoteke koje se nalaze u radnom direktoriju. U suprotnom sustav bi pretpostavio da se nalaze u *home* direktoriju.

Kada se skripta pokrene naredbom **qsub**, dobit će se dva izlaza − u **izlaz.log** datoteci nalazi se sadržaj datoteke **tekst.txt**. U **greska.log** datoteci nalaze se sve greške, a u ovom slučaju to je upozorenje da datoteka **drugi-tekst.txt** ne postoji.

11.1.4. Primjer spoj.sh

Ako se ne želi odvajati izlaz i pogreška, može ih se spojiti opcijom **join**, tj. **-j**, tako da se ispisuju isto kako bi se ispisivale na zaslonu, tj. u terminalu. To znači da postoji samo jedan *output*, definiran opcijom **-o**.

```
1   #!/bin/bash
2
3   #PBS -q cpu-radionica
4   #PBS -l select=1:ncpus=1
5   #PBS -j oe
6   #PBS -o spoj_izlaza.log
7
8   cd $PBS_O_WORKDIR
9
10  cat tekst.txt
11  cat drugi_tekst.txt
```

Izlazi nisu spojeni na kraju izvođenja, već se greške i izlazi ispisuju redoslijedom kako se pojavljuju. Skripta je po svemu ostalom ista kao i prije − oba izlaza će se nalaziti u datoteci **spoj_izlaza.log**, prvo *output* prve naredbe, a zatim *output* druge (koji je zapravo samo greška).

11.1.5. Primjer niz.sh

Polja poslova su koncept koji omogućava podešavanju jedne PBS skripte za više poslova. Korisna su kada, primjerice, korisnik ima poslove koji se razlikuju samo u *input* datoteci.

```
1   #!/bin/bash
2
3   #PBS -q cpu-radionica
4   #PBS -l select=1:ncpus=1
5   #PBS -J 1-3:1
6
7   cd $PBS_O_WORKDIR
8
9   cat array_dir/file_${PBS_ARRAY_INDEX}.txt
```

Opcijom **-J** definira se rad s nizom, a argument opcije je niz cijelih brojeva, u ovom slučaju 1 do 3, za korak od 1. To znači da se nalazi niz od 3 posla, a svaki će biti definiram takozvanim „indeksom niza“, odnosno varijablom **$PBS_ARRAY_INDEX**.

U praktičnom smislu to znači da će se ova skripta „podnijeti“ 3 puta, pri čemu će varijabla PBS_ARRAY_INDEX svakim „podnošenjem“ dobiti novu vrijednost.

U direktoriju **array-dir** imamo 3 datoteke, odnosno 3 *input filea* koji se razlikuju u nazivu - **file-1**, **file-2**, **file-3**. Svaki ima malo drugačiji sadržaj (ispisuju samo vlastito ime).

Njihovo nazivlje može se iskoristiti jer brojevi *fileova* odgovaraju brojevima u nizu pa se *input fileovi* mogu ispisati kao **file-${PBS_ARRAY_INDEX}**.

To znači da će svako „podnošenje“ ispisati sadržaj drugi datoteke, od 1 do 3.

Nakon što se skripta podnese i posao završi, svaki će *output* biti obilježen i drugim brojem, odnosno indeksom niza.

11.1.6. Primjer tmpdir.sh

Za spremanje privremenih rezultata koji se generiraju tijekom izvođenja posla ne preporučuje se koristiti *home* direktorij − smanjuje se učinkovitost aplikacije i ne iskorištava brzo spremište (BeeGFS-fast) rezervirano za pohranu privremenih datoteka. PBS za svaki pojedini posao kreira privremeni direktorij na adresi pohranjenoj u varijabli **TMPDIR (/tmp/<job id>).**

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=1:ncpus=1

cd $TMPDIR

pwd &>file.txt
cp file.txt $PBS_O_WORKDIR
```

U ovom primjeru se nakon pozicioniranja u TMPDIR u **file.txt** zapisuje trenutna putanja korištenjem naredbe **pwd** nakon čega je potrebno datoteku **file.txt** kopirati u direktorij iz kojeg je skripta pokrenuta.

11.2. Primjeri CPU aplikacija

11.2.1. Primjeri MPI aplikacija

Primjer korištenja aplikacije Quantum Espresso (QE)

U idućim primjerima MPI aplikacija korištenjem **mpiexec** pokreće 8 MPI procesa s istog čvora (select=8, place=pack ili select=1:mpiprocs=8).

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=8:ncpus=1:mem=500mb
#PBS -l place=pack

cd $PBS_O_WORKDIR
module load scientific/qe/7.2-gnu
mpiexec pw.x -i input.inp
```

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=1:mpiprocs=8:ncpus=8:mem=4000mb

cd $PBS_O_WORKDIR
module load scientific/qe/7.2-gnu
mpiexec pw.x -i input.inp
```

U *inputu* su definirane postavke računa, a u radnom direktoriju još se nalaze i pseudopotencijali svih atoma – Ca, C i O.

Primjer korištenja aplikacije ORCA

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=16:ncpus=1:mem=1gb
#PBS -l place=pack
#PBS -j oe

cd $PBS_O_WORKDIR
module load scientific/orca/5.0.4
runorca.mpi --input input.inp
```

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=1:mpiprocs=16:ncpus=16:mem=16gb
#PBS -l place=pack
#PBS -j oe

cd $PBS_O_WORKDIR
module load scientific/orca/5.0.4
runorca.mpi --input input.inp
```

U ovom primjeru MPI aplikacije **ORCA** njezin **mpiexec** (tj. *mpirun*) je inkorporiran unutar **runorca.mpi** *wrappera*, koji osim toga prepravlja *input* tako da mu definira broj jezgri tj. MPI procesa i seli se u TMPDIR jer aplikacija stvara velik broj privremenih datoteka, tj. radi veliki *input/output*.

Ovdje se pokreće 16 MPI procesa s istog čvora (select=16, place=pack ili select=1:mpiprocs=16). 16 fizičkih jezgri je implicitno definirano (16 × 1 ili 1 x 16 ncpus) tako da svaka jezgra ima svoj MPI proces

11.2.2. Primjeri OpenMP aplikacija

Primjer korištenja aplikacije Q-Chem

Q-Chem je OpenMP aplikacija, a kod nje je broj OpenMP *threadova* definiran kao argument opcije **-nt** (*number of threads*). Koristi se 8 fizičkih jezgri i 8 OpenMP *threadova* – svaka jezgra ima svoj *thread*.

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=1:ncpus=8:mem=1500mb
#PBS -N $USER

cd $PBS_O_WORKDIR
module load scientific/q-chem/latest
qchem -nt $OMP_NUM_THREADS input.inp
```

U **input.inp** definirane su postavke računa, slično kao i u primjeru s Gaussianom te je račun sam po sebi sličan.

Primjer korištenja aplikacije Gaussian

Gaussian je još jedan primjer aplikacije koja se ne zna širiti izvan jednog čvora, odnosno koristi OpenMP (točnije, Gaussian koristi drugu tehnologiju sličnu OpenMP-u, ali koncept je sličan).

Budući da nemamo MPI, ne koristimo MPI pokretače.

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=1:ncpus=8:mem=1gb
#PBS -o izlaz.log
#PBS -e greska.log
#PBS -N g16

cd $PBS_O_WORKDIR
module load scientific/gaussian/16-C01
dog.mem input.com
```

U *input* datoteci definirane su postavke računa – funkcionalnosti, bazni skup te koordinate molekule kojoj će se računati energija.

11.2.3. Primjer korištenja hibridnih aplikacija

Primjer korištenja aplikacije Gromacs

GROMACS je primjer aplikacije koja se paralelizira hibridno. U ovom slučaju tražena su 4 *chunka*. Svaki *chunk* ima po jedan MPI proces, po 4 fizičke jezgre, a i OMP_NUM_THREADS implicitno je postavljen na 4.

Iz MPI procesa stvorit će se 4 OMP_NUM_THREADS *threadova* pri čemu će svaki imati po jednu svoju fizičku jezgru. To je definirano pozivom **mpiexec -d ${OMP_NUM_THREADS} --cpu-bind depth**.

```
#!/bin/bash

#PBS -q cpu-radionica
#PBS -l select=4:ncpus=4:mem=1gb
#PBS -l place=pack

cd $PBS_O_WORKDIR
module load scientific/gromacs/2022.5-gnu
mpiexec -d $OMP_NUM_THREADS --cpu-bind depth gmx mdrun -pin on -v -deffnm nvt
```

Ova aplikacija radi s molekulskom dinamikom, odnosno provodi NVT ekvilibraciju proteina u vodi. Prethodne aplikacije (ORCA, Gaussian, xTB, Q-Chem) radile su s malim (organskim) molekulama i kvantnom mehanikom, dok Gromacs i njemu slični (AMBER) rade s velikim molekulama (biomakromolekulama) i klasičnom mehanikom.

11.3. Primjeri GPU aplikacija

TeraChem je primjer aplikacije koja će se izvršavati na grafičkom procesoru. Zatražen je jedan *chunk* s 1 GPU-om, a i jedan CPU je definiran implicitno.

```
#!/bin/bash

#PBS -q gpu-radionica
#PBS -l select=1:ngpus=1:ncpus=1:mem=6gb

cd $PBS_O_WORKDIR
module load scientific/terachem/1.96
terachem start.sp
```

Aplikacija će preuzeti CUDA_VISIBLE_DEVICES iz pozadine te će tako znati na kojem će se GPU-u izvršavati.

Postavke računa definirane su u datoteci **start.sp**, a koordinate molekule u izdvojenom **coords.xyz**. Aplikacija radi sličan račun kao i neke aplikacije ranije (ORCA, Gaussian…), ali ovaj put na GPU.

11.4. Primjeri prema tipu paralelizacije – Monte Carlo procjena broja π

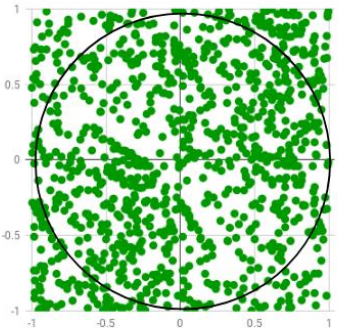
Monte Carlo metode zasnivaju se na velikom broju nasumično generiranih vrijednosti kako bismo statistički procijenili vrijednost koja se računa. Jedan od osnovnih primjera primjene algoritma Monte Carlo je procjena broja π.

Ideja je generirati nasumične točke (x, y) u 2D ravnini u kojoj je kvadrat stranice $2r$ centriran u ishodište (0,0). Zamislimo kružnicu s istim radijusom r, upisanu u kvadrat. Za $r = 1$ imamo

$$\frac{A_c}{A_r} = \frac{r^2\,\pi}{4r^2} = \frac{\pi}{4}$$

Odnosno, za ogroman broj točaka u prvom kvadratnu aproksimiramo

$$\pi \approx 4 \cdot \frac{broj\ točaka\ unutar\ kružnice}{ukupan\ broj\ nasumičnih\ točaka}$$



Svaki primjer ispisuje na kojem čvoru i procesorskoj jezgri se izvršava!

Slijedni primjer

```
double monte_carlo_pi(size_t iterations) {
    size_t i;
    double x, y;
    int inside_circle = 0;

    for (i = 0; i < iterations; i++) {
        x = random_point();
        y = random_point();
        if (x * x + y * y <= 1.0) {
            inside_circle++;
        }
    }
    return 4.0 * inside_circle / iterations;
}
```

```
$./serial -n 1000000000
Running on node x3000c0s27b0n0, CPU: 99
Using sample size: 1000000000
Estimate of pi: 3.141647
Time taken: 24.306372 seconds
```

Paralelni primjer – OpenMP

Kod paralelizacije na razini dretvi i razini podataka petlju for možemo jednostavno paralelizirati tako da svaka dretva izvrijednjuje dio iteracija.

```
double monte_carlo_pi(size_t iterations) {
    size_t i;
    size_t inside_circle = 0;

    #pragma omp parallel
    {
        double x,y;
        unsigned int seed = time(NULL) ^ omp_get_thread_num();
        size_t local_inside_circle = 0;

        #pragma omp for
        for (i = 0; i < iterations; i++) {
            x = random_point();
            y = random_point();
            if (x * x + y * y <= 1.0) {
                local_inside_circle++;
            }
        }
        #pragma omp atomic
        inside_circle += local_inside_circle;
    }
    return 4.0 * inside_circle / iterations;
}
```

```
#PBS -l select=1:ncpus=10
./paralel -n 10000000000

Thread 2 running node x3000c0s23b0n0, logical CPU: 197, physical CPU: 5
Thread 6 running node x3000c0s23b0n0, logical CPU: 199, physical CPU: 7
Thread 8 running node x3000c0s23b0n0, logical CPU: 195, physical CPU: 3
Thread 4 running node x3000c0s23b0n0, logical CPU: 196, physical CPU: 4
Thread 5 running node x3000c0s23b0n0, logical CPU: 73, physical CPU: 9
Thread 9 running node x3000c0s23b0n0, logical CPU: 72, physical CPU: 8
Thread 3 running node x3000c0s23b0n0, logical CPU: 200, physical CPU: 8
Thread 7 running node x3000c0s23b0n0, logical CPU: 72, physical CPU: 8
Thread 1 running node x3000c0s23b0n0, logical CPU: 201, physical CPU: 9
Thread 0 running node x3000c0s23b0n0, logical CPU: 64, physical CPU: 0
Using sample size: 10000000000
Estimate of pi: 3.141590
Time taken: 13.057063 seconds
```

Sljedeći ispis pokazuje kako pomoću OMP varijabli okruženja možemo utjecati na dretve. Zadali smo da svaka dretva bude na jednoj fizičkoj jezgri i da se ne može seliti na druge jezgre. U tom slučaju vidimo da je svaka dretva i zaista na svojoj jezgri te vidimo malo poboljšanje u vremenu izvršavanja.

```
#PBS -l select=1:ncpus=10
export OMP_PLACES=CORES
export OMP_PROC_BIND=TRUE
./paralel -n 10000000000

Thread 0 running node x3000c0s23b0n0, logical CPU: 64, physical CPU: 0
Thread 8 running node x3000c0s23b0n0, logical CPU: 200, physical CPU: 8
Thread 2 running node x3000c0s23b0n0, logical CPU: 194, physical CPU: 2
Thread 4 running node x3000c0s23b0n0, logical CPU: 196, physical CPU: 4
Thread 1 running node x3000c0s23b0n0, logical CPU: 193, physical CPU: 1
Thread 7 running node x3000c0s23b0n0, logical CPU: 199, physical CPU: 7
Thread 3 running node x3000c0s23b0n0, logical CPU: 195, physical CPU: 3
Thread 9 running node x3000c0s23b0n0, logical CPU: 201, physical CPU: 9
Thread 5 running node x3000c0s23b0n0, logical CPU: 197, physical CPU: 5
Thread 6 running node x3000c0s23b0n0, logical CPU: 198, physical CPU: 6
Using sample size: 10000000000
Estimate of pi: 3.141586
Time taken: 12.418772 seconds
```

MPI

Kod MPI verzije svaki proces računa dio iteracija te na kraju lokalnu vrijednost broja pi šalje na proces sa rank=0.

```
// Initialize MPI
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Each process calculates its share of the iterations
size_t local_iterations = iterations / size;
pi_estimate = monte_carlo_pi(local_iterations, rand());

// Reduce all local pi estimates to get the global estimate
MPI_Reduce(&pi_estimate, &global_pi_estimate, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
```

Donji primjer pokazuje pokretanje programa s 4 MPI procesa na jednom čvoru te je program konfiguriran da svaki proces dobije jednu procesorsku jezgru.

```
$mpirun -n 4 --display-map --map-by L1CACHE --rank-by SLOT ./mpi -n 10000000
========================   JOB MAP   ========================
Data for JOB prterun-x3000c0s27b0n0-1787538@1 offset 0 Total slots allocated 64
   Mapping policy: BYL1CACHE:NOOVERSUBSCRIBE  Ranking policy: SLOT Binding policy:
L1CACHE:IF-SUPPORTED
   Cpu set: N/A  PPR: N/A  Cpus-per-rank: N/A  Cpu Type: CORE
Data for node: x3000c0s27b0n0   Num slots: 64   Max slots: 0    Num procs: 4
      Process jobid: prterun-x3000c0s27b0n0-1787538@1 App: 0 Process rank: 0 Bound:
package[0][core:0]
      Process jobid: prterun-x3000c0s27b0n0-1787538@1 App: 0 Process rank: 1 Bound:
package[0][core:1]
      Process jobid: prterun-x3000c0s27b0n0-1787538@1 App: 0 Process rank: 2 Bound:
package[0][core:2]
      Process jobid: prterun-x3000c0s27b0n0-1787538@1 App: 0 Process rank: 3 Bound:
package[0][core:3]
=============================================================

Rank 0 running node x3000c0s27b0n0, CPU: 64
Rank 1 running node x3000c0s27b0n0, CPU: 1
Rank 3 running node x3000c0s27b0n0, CPU: 67
Rank 2 running node x3000c0s27b0n0, CPU: 2

Using sample size: 10000000
Estimate of pi: 3.141839
```

Hibridna – MPI + OpenMP

Kod hibridne paralelizacije svaki MPI proces stvara dodatne dretve.
Kada se pokretanje programa ne konfigurira ispravno, procesi imaju pristup samo jednoj fizičkoj jezgri 0, svaki na svojem čvoru.

```
mpirun -n 2 --display-map   ./hybrid -n 1000000000

========================   JOB MAP   ========================
Data for JOB prterun-x8000c2s6b0n0-154512@1 offset 0 Total slots allocated 2
   Mapping policy: BYCORE:NOOVERSUBSCRIBE  Ranking policy: FILL Binding policy: CORE:IF-
SUPPORTED
   Cpu set: N/A  PPR: N/A  Cpus-per-rank: N/A  Cpu Type: CORE

Data for node: x8000c2s6b0n0    Num slots: 1    Max slots: 0    Num procs: 1
      Process jobid: prterun-x8000c2s6b0n0-154512@1 App: 0 Process rank: 0 Bound:
package[0][core:0]

Data for node: x8000c2s6b0n1    Num slots: 1    Max slots: 0    Num procs: 1
      Process jobid: prterun-x8000c2s6b0n0-154512@1 App: 0 Process rank: 1 Bound:
package[0][core:0]

=============================================================
Rank 0, thread 3 running node x8000c2s6b0n0, CPU: 64
Rank 0, thread 2 running node x8000c2s6b0n0, CPU: 64
Rank 1, thread 1 running node x8000c2s6b0n1, CPU: 0
Rank 1, thread 0 running node x8000c2s6b0n1, CPU: 0
Rank 1, thread 3 running node x8000c2s6b0n1, CPU: 64
Rank 0, thread 1 running node x8000c2s6b0n0, CPU: 0
Rank 1, thread 2 running node x8000c2s6b0n1, CPU: 64
Rank 0, thread 0 running node x8000c2s6b0n0, CPU: 0
Using sample size: 1000000000
Estimate of pi: 3.141621
Time taken: 3.231557 seconds
```

Kada se pokretanje programa konfigurira ispravno, slot:PE=4 svaki proces može koristiti 4 procesorske jezgre na svojem čvoru.

```
mpirun -n 2 --display-map --map-by slot:PE=4  ./hybrid -n 1000000000

========================   JOB MAP   ========================
Data for JOB prterun-x8000c2s6b0n0-154553@1 offset 0 Total slots allocated 2
   Mapping policy: BYSLOT:NOOVERSUBSCRIBE  Ranking policy: SLOT Binding policy: CORE:IF-SUPPORTED
   Cpu set: N/A  PPR: N/A  Cpus-per-rank: 4  Cpu Type: CORE

Data for node: x8000c2s6b0n0    Num slots: 1    Max slots: 0    Num procs: 1
      Process jobid: prterun-x8000c2s6b0n0-154553@1 App: 0 Process rank: 0 Bound: package[0][core:0-3]

Data for node: x8000c2s6b0n1    Num slots: 1    Max slots: 0    Num procs: 1
      Process jobid: prterun-x8000c2s6b0n0-154553@1 App: 0 Process rank: 1 Bound: package[0][core:0-3]

=============================================================
Rank 0, thread 0 running node x8000c2s6b0n0, CPU: 66
Rank 0, thread 1 running node x8000c2s6b0n0, CPU: 65
Rank 0, thread 2 running node x8000c2s6b0n0, CPU: 67
Rank 0, thread 3 running node x8000c2s6b0n0, CPU: 0
Rank 1, thread 0 running node x8000c2s6b0n1, CPU: 2
Rank 1, thread 1 running node x8000c2s6b0n1, CPU: 0
Rank 1, thread 2 running node x8000c2s6b0n1, CPU: 1
Rank 1, thread 3 running node x8000c2s6b0n1, CPU: 67
Using sample size: 1000000000
Estimate of pi: 3.141583
Time taken: 1.384174 seconds
```

Hibridna – MPI + OpenMP (cray-mpich)

Ovi primjeri pokazuju kako konfigurirati hibridne programe koji koriste cray-mpich.

```
#PBS -l select=2:ncpus=10
mpiexec  ./hybrid -n 10000000000

Rank 0, thread 1 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 0 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 2 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 3 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 4 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 5 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 6 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 7 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 8 running node x3000c0s21b0n0, CPU: 64
Rank 0, thread 9 running node x3000c0s21b0n0, CPU: 64
Using sample size: 1410065408
Estimate of pi: 3.141595
Time taken: 55.011679 seconds
Rank 1, thread 1 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 0 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 2 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 3 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 4 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 5 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 6 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 7 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 8 running node x3000c0s23b0n0, CPU: 64
Rank 1, thread 9 running node x3000c0s23b0n0, CPU: 64
```

```
mpiexec --cpu-bind depth -d 10 ./hybrid -n 10000000000

Rank 0, thread 9 running node x3000c0s21b0n0, CPU: 41
Rank 0, thread 4 running node x3000c0s21b0n0, CPU: 36
Rank 0, thread 6 running node x3000c0s21b0n0, CPU: 38
Rank 0, thread 2 running node x3000c0s21b0n0, CPU: 34
Rank 0, thread 0 running node x3000c0s21b0n0, CPU: 32
Rank 0, thread 8 running node x3000c0s21b0n0, CPU: 40
Rank 0, thread 1 running node x3000c0s21b0n0, CPU: 33
Rank 0, thread 7 running node x3000c0s21b0n0, CPU: 39
Rank 0, thread 5 running node x3000c0s21b0n0, CPU: 37
Rank 0, thread 3 running node x3000c0s21b0n0, CPU: 35
Using sample size: 1410065408
Estimate of pi: 3.141582
Time taken: 5.993236 seconds
Rank 1, thread 6 running node x3000c0s23b0n0, CPU: 38
Rank 1, thread 7 running node x3000c0s23b0n0, CPU: 39
Rank 1, thread 4 running node x3000c0s23b0n0, CPU: 36
Rank 1, thread 5 running node x3000c0s23b0n0, CPU: 37
Rank 1, thread 2 running node x3000c0s23b0n0, CPU: 34
Rank 1, thread 1 running node x3000c0s23b0n0, CPU: 33
Rank 1, thread 3 running node x3000c0s23b0n0, CPU: 35
Rank 1, thread 9 running node x3000c0s23b0n0, CPU: 41
Rank 1, thread 8 running node x3000c0s23b0n0, CPU: 40
Rank 1, thread 0 running node x3000c0s23b0n0, CPU: 32
```