Multiprocesorski sistemi

MPI
Analiza performansi

Marko Mišić, Aleksandar Majdak

13S114MUPS, 13E114MUPS,

MS1MPS, RI5MS, IR4MPS, SI4MPS

2016/2017.

Zasnovano na:
Blaise Barney, Lawrence Livermore National Laboratory
Message Passing Interface
https://computing.llnl.gov/tutorials/mpi/

Analiza performansi programa

- Postoje različiti alati za analizu performansi na sekvencijalnim i paralelnim platformama
  - Jednostavni alati za rad iz komandne linije
  - C i Fortran rutine za merenje vremena
  - Profajleri programskog koda
  - Generatori putanje izvršavanja (engl. *execution trace*)
  - Grafički analizatori izvršavanja
- Neki od ovih alata rade u realnom vremenu, dok neki analiziraju trag programa nakon izvršavanja
- Većina ozbiljnijih alata zahteva duže učenje i savladavanje

Faktori koji utiču na performanse (1)

- Pisanje skalabilnih, paralelnih i distribuiranih aplikacija koje optimalno koriste resurse je zahtevan posao
  - Resursi su najčešće slabo iskorišćeni ili se koriste neefikasno
- Faktori koji utiču na performanse su složeni, međusobno isprepletani i često skriveni u odnosu na programera
  - Faktori se mogu podeliti na aplikativne, hardverske i softverske
- Alati za analizu performansi su neophodni za optimizaciju performansi
  - Oni pomažu programeru da razume šta program zapravo radi i kako performanse mogu da se poboljšaju

Faktori koji utiču na performanse (2)

- Aplikativni faktori
  - Izabrani algoritam
  - Veličina
  - Obrasci pristupa memoriji i komunikacije
  - Korišćenje I/O operacija
  - Granularnost zadataka i balansiranje opterećenja
  - Amdalov zakon
- Hardverski faktori
  - Procesorska arhitektura
  - Memorijska hijerarhija
  - Interkonekciona mreža
- Softverski faktori
  - Operativni sistem
  - Korišćeni prevodilac
  - Protokoli za komunikaciju
  - Biblioteke koje se koriste

Faktori koji utiču na MPI performanse (1)

- Zbog složenosti i međusobnog odnosa faktora koji utiču na MPI performanse njihova generalizacija je veoma teška
- Platforma i arhitektura
  - Procesorska arhitektura
    - Brzina i broj procesora
  - Memorijski podsistem
    - Konfiguracija memorije i keš hijerarhija
  - Mrežni uredjaji
    - Tip, protok i kašnjenje
  - Karakteristike operativnog sistema

Faktori koji utiču na MPI performanse (2)

- Mrežni faktori
  - Mrežni hardver
    - Ethernet, FDDI, switch-evi, ruteri
  - Protokoli
    - TCP/IP, UDP/IP
  - Konfiguracija, način rutiranja
  - Mrežno zasićenje i takmičenje za resurse
- MPI implementacija
  - Baferisanje poruka
  - Message passing protokoli – *eager vs. rendezvous*
  - Sinhronizacija slanja i prijema – *polling vs. interrupt*
  - Implementacija rutina za slanje i prijem poruka

Faktori koji utiču na MPI performanse (3)

- Izrada aplikacije
  - Efikasnost algoritama i skalabilnost
  - Odnos komunikacije i proračuna
  - Balans opterećenja
  - Šema korišćenja memorije
  - Ulaz/Izlaz
  - Veličina poruka
  - Tip korišćenih MPI komunikacionih rutina
    - Blokirajuća vs. neblokirajuća komunikacija
    - *Point-to-point*
    - Kolektivna komunikacija

Performanse i strategije (1)

- Podešavanje performansi (engl. *tuning*) se u suštini svodi na smanjivanje tzv. *wall clock* vremena izvršavanja
  - *Wall clock time* – vreme koje prođe od početka do kraja izvršavanja programa
- Ovo je iterativan proces i zahteva pronalaženje vrućih mesta (engl. *hot spots*) i eliminaciju uskih grla (engl. *bottleneck*) u njima
  - *Hot spot* – deo koda koji koristi neproporcionalno mnogo procesorskog vremena
  - *Bottleneck* – deo koda koji neefikasno koristi procesorsko vreme i izaziva nepotrebna kašnjenja

Performanse i strategije (2)

- Podešavanje performansi obično zahteva *profiling*
  - Korišćenje softverskih alata za merenje programskih *run-time* karakteristika i iskorišćenja resursa
- Tipična strategija podešavanja performansi:
  - Odrediti delove koda kod kojih se najviše mogu povećati performanse
    - Odabrati one koji konzumiraju najviše vremena
  - Razmotriti optimizaciju algoritama odabranih delova koda
  - Za izračunavanja koja zavise od podataka, pažljivo izabrati ulazne skupove za testiranje
    - I u smislu vrednosti i u smislu veličine
    - Održavati konzistentnost ulaznih podataka prilikom finog podešavnja
  - Koristiti dostupne optimizacije prevodioca i pretprocesora
  - Odlučiti kada stati sa optimizacijom

Rutine za merenje vremena (1)

- Operativni sistemi pružaju mogućnost za merenje vremena izvršavanja programa i dobijanje drugih informacija
  - Koliko je proces konzumirao procesorsko vreme, korišćenje resursa...
- Jedna od ovih komandi je *time* koja prikazuje
  - *Real time* – prikazuje ukupno *wall clock* vreme
  - *User time* – ukupno vreme koje je potrošeno na CPU
  - *System time* – ukupno režijsko vreme provedeno u sistemskim pozivima
- Sistemsko i korisničko vreme mogu biti drugačije definisani na različitim arhitekturama
- Komanda *time* ima i druge mogućnosti za prikaz informacija o pozvanom programu

Rutine za merenje vremena (2)

- Na većini UNIX sistema u okviru standardne C biblioteke je dostupna rutina *gettimeofday*
  - Vraća vreme u sekundama i mikrosekundama u odnosu na ponoć, 1.1.1970.
  - Koristi se na početku i na kraju dela koda koji nam je od interesa
  - Proteklo vreme je razlika ova dva vremena
  - Rezolucija je hardverski zavisna, ali bi trebalo da je izražena u mikrosekundama
  - Koristi tip (strukturu) *timeval* da smesti podatke o dohvaćenom vremenu
    - Polje *sec* se odnosi na sekunde
    - Polje *usec* se odnosi na mikrosekunde
  - Drugi parametar se koristi za podešavanje vremenske zone
    - Obično se postavlja na *NULL*
- Mogu postojati i različite rutine, specifične za sam sistem
  - Obično se odnose na hardverske tajmere najveće preciznosti

Rutine za merenje vremena (MPI)

- MPI poseduje ugrađene rutine za merenje vremena
- *MPI_Wtime* vraća proteklo *wall clock* vreme na pozvanom procesoru
  - Poziva se na početku i na kraju koda čije nam je merenje od interesa
  - Proteklo vreme je razlika ova dva vremena
- *MPI_Wtick()* vraća rezoluciju *MPI_Wtime* rutine u sekundama na datom procesoru
  - Rezolucija je reda veličine mikrosekunde

Baferisanje poruka (1)

- Baferski prostor može obezbediti sistem ili programer
  - Sistemski bafer je uglavnom obezbeđen implementacijom i programer ne može uticati na njega
  - Korisnički bafer obezbeđuje programer i on treba da vodi računa o njemu
- Realizacija sistemskog bafera zavisi od implementacije
  - Bafer na strani slanja poruke
  - Bafer na strani primanja poruke
  - Bez bafera
  - Bafer pod određenim uslovima

Baferisanje poruka (2)

- Prednosti
  - Osnovna prednost baferisanja je omogućavanje asinhrone komunikacije
- Nedostaci
  - Baferski prostor je ograničen resurs i njegovo prepunjanje ili ogromno korišćenje može dovesti do grešaka
  - Nije uvek očigledno vidljivo
  - Izvršavanje programa može varirati u zavisnosti od načina korišćenja bafera
- Korektan MPI program ne zavisi od baferisanja

MPI Message Passing protokoli

- Određuju način i politiku koju MPI implementacija koristi da bi izvršila dostavu poruke
  - Nisu definisani standardom i zavise od implementacije
- *Eager* – asinhroni protokol koji omogućava kompletiranje rutine slanja bez znanja o odgovarajućem prijemu
- *Rendezvous* – sinhroni protokol za slanje poruke koji zahteva potvrdu prijema od odgovarajuće rutine
- MPI implementacija može koristiti kombinaciju protokola za istu rutinu
- Ovi protokoli često rade zajedno sa protokolima za baferisanje

Eager message passing protokol (1)

- Proces pošiljalac pretpostavlja
  da proces primalac može skladištiti poruku
- Odgovornost je procesa primaoca
  da baferuje poruku nakon pristizanja
- Pretpostavka je bazirana
  na implementacionim garancijama
  o postojanju bafera određene veličine
- Uglavnom se koristi za manje poruke
  - Reda veličine kilobajta
  - Veličina poruke se može ograničiti za primenu ovog protokola u odnosu na broj aktivnih procesa

Eager message passing protokol (2)

- Prednosti
  - Umanjuje sinhrona kašnjenja
    - Nije potrebna potvrda da je primalac spreman
  - Pojednostavljuje program
- Nedostaci
  - Najvažniji nedostatak je neskalabilnost
    - Ako se povećava broj pošiljalaca može se mnogo povećati bafer
  - Može prouzrokovati iscrpljivanje memorije i neočekivani završetak programa pri prekoračenju bafera
  - Nepotrebno alociranje bafera za mali broj poruka
  - Veći režijski troškovi na strani primaoca za primanje poruka
    - Često kopiranje poruka u bafer

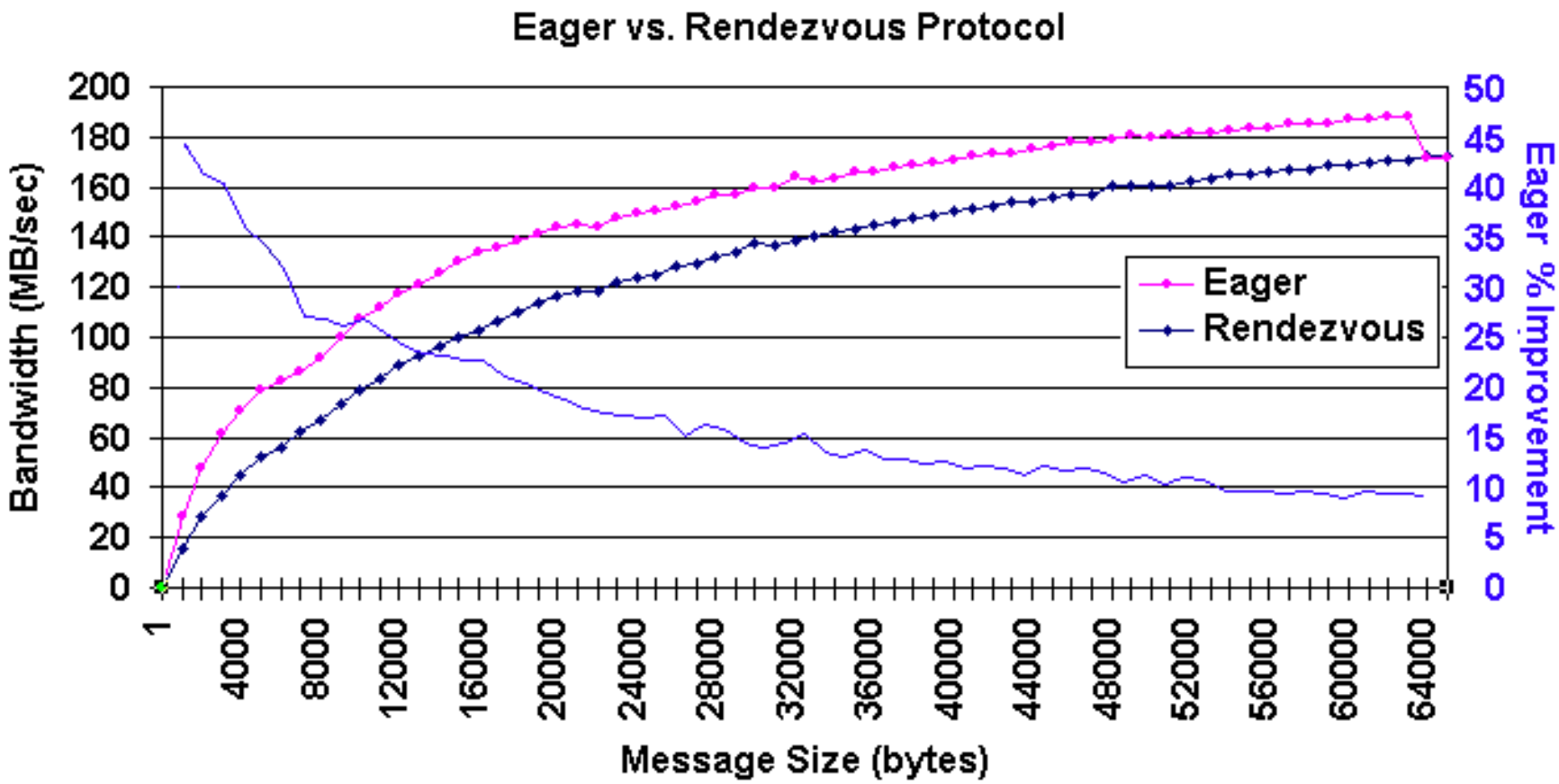
Rendezvous message passing protokol (1)

- Protokol se koristi kada se ne može pretpostaviti veličina bafera primaoca ili kada je adekvatniji od eager protokola
- Zahteva "handshaking" između pošiljaoca i primaoca
  - Pošiljalac šalje omotač poruke primaocu
  - Omotač je primi i sačuva
  - Primalac javlja pošiljaocu kada je odgovarajući prostor dostupan
  - Pošiljalac prima odgovor i šalje poruku
  - Primalac prihvata poruku

Rendezvous message passing protokol (2)

- Prednosti
  - Skalabilan za razliku od *eager* protokola
  - Otporan na iscrpljivanje memorije
  - Otporan na nemogućnost primanja poruke
  - Zahteva baferovanje jedino malih zaglavlja poruke
  - Eliminiše kopiranje poruka iz korisničkog prostora u korisnički prostor
- Nedostaci
  - Sadrži kašnjenje usled sinhronizacije
  - Povećava složenost aplikacije

Eager vs Rendezvous poređenje



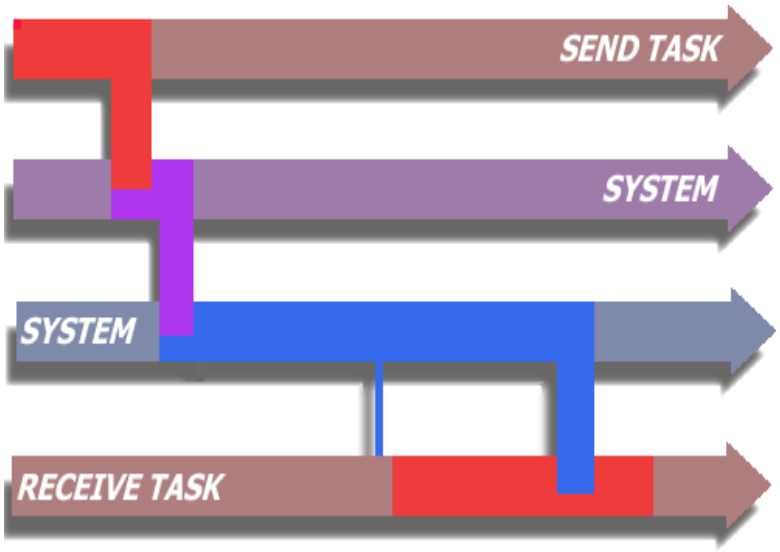
Sinhronizacija pošiljaoca i primaoca (1)

- MPI standard ne definiše način realizacije
  - Kako primalac zna da pošiljalac zahteva slanje?
  - Koliko često primalac treba da proverava da li je pristigla poruka?
  - Kako se koristi procesor za slanje?
- MPI implementacija ovoga se zasniva na dva načina: *polling* ili *interrupt*
  - *Polling* – korisnički MPI proces će biti prekidan od strane sistema radi provere pristigle poruke
  - *Interrupt* – korisnički MPI proces će biti prekinut kada poruka pristigne

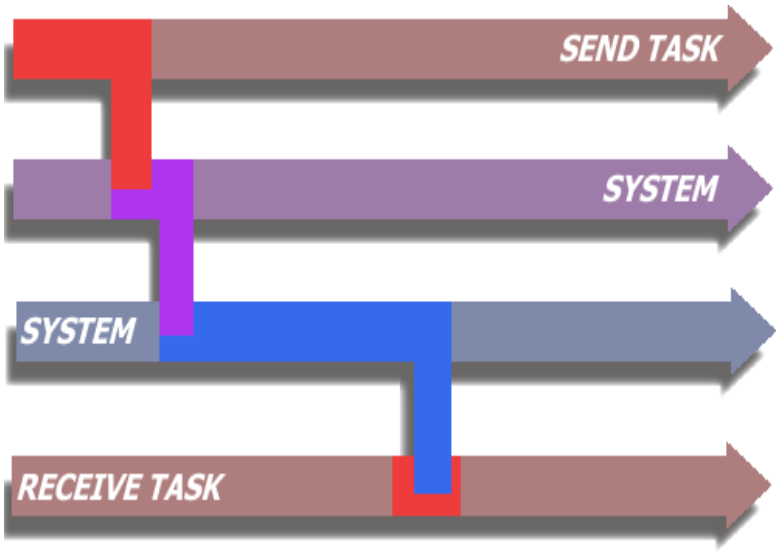
Sinhronizacija pošiljaoca i primaoca (2)

- Prednosti *polling-a*
  - Kada je veliki broj pošiljalaca
  - Primanje se obavlja u nepredvidivim trenucima
  - Interrupt tada konzumira značajno procesorsko vreme
- Prednosti *interrupt-a*
  - Kada se koriste neblokirajuće slanje i prijem
  - Kod nesinhronizovanog para primanja i slanja
    - Izdaju se u različitim vremenskim trenucima
  - Kod preklapanja obrade i komunikacije
    - Ne izdaje se odmah naredba za čekanje nakon izdavanja naredbi za slanje ili prijem

Sinhronizacija pošiljaoca i primaoca (3)



Polling

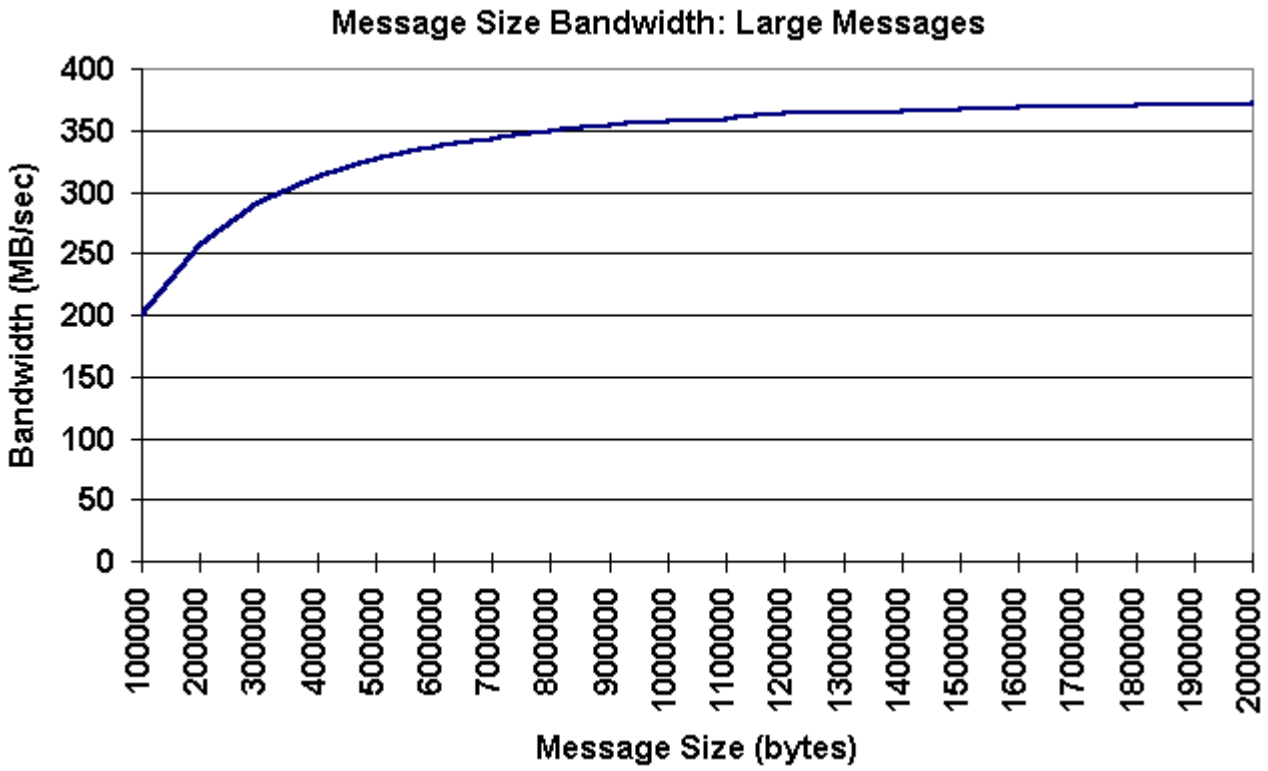


Interrupt

Veličina poruka (1)

- Veličina poruka može značajno uticati na performanse
- U većini slučajeva povećavanje veličine poruka donosi poboljšanje performansi
- Za aplikacije koje intenzivno komuniciraju sa porukama, povećavanje veličine poruka je vredno truda
- Zbog početne eksponencijalne zavisnosti protoka od veličine poruka,
  malo povećanje poruke donosi bolji protok,
  odnosno bržu komunikaciju

Veličina poruka (2)



Izvedeni tipovi podataka

- *Point-to-point* rutine zahtevaju da se poruke sastoje od kontinualnih podataka istog tipa
- Izvedeni tipovi obezbeđuju veću fleksibilnost i funkcionalnost
  - Omogućavaju programeru da kreira kontinualne tipove sastavljene od osnovnih MPI tipova
  - Konstruišu poruke koje sadrže različite tipove
- Poboljšavanje performansi kroz smanjivanje slanja velikog broja malih poruka
- Način implementacije izvedenih tipova nije propisan standardom

Takmičenje za mrežu

- Takmičenje za mrežu se pojavljuje
  kada je količina podataka koju treba poslati
  između MPI procesa veća od protoka mreže
- Ovo zasićenje mreže
  umanjuje komunikacione performanse
- Programer ništa ne može učiniti povodom ovoga

Reference

- Livermore Computing specific information:
  - https://computing.llnl.gov/tutorials/performance_tools/
  - https://computing.llnl.gov/tutorials/mpi_performance/