Parallel Numerical Integration

STUDENT: COSMIN-RĂZVAN VANCEA (343C1)

Descrierea problemei

Programul aproximează o integrală definită folosind algoritmul “Composite Simpson’s Rule”

Ideea din spatele algoritmului este de a împărți intervalul inițial *[a, b]* în *n* subintervale egale

Pe fiecare subinterval se va aplica algoritmul Simpson’s Rule.

La final se însumează rezultatele parțiale, obținând aproximarea integralei definite.

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right]$$

$$h = (b-a)/n$$

Implementări paralele

OpenMP

Se paralelizează calculul celor două sume prezentate în formula anterioară.

Fiecare fir de execuție calculează câte o subsumă din cele două sume (pare&impare)

Cum nu există dependințe între date, nu există nevoia de sincronizare între thread-uri.

pthreads

Se paralelizează calculul celor două sume prezentate în formula anterioară.

Fiecare fir de execuție calculează câte o subsumă din cele două sume (pare&impare)

Cum nu există dependințe între date, nu există nevoia de sincronizare între thread-uri.

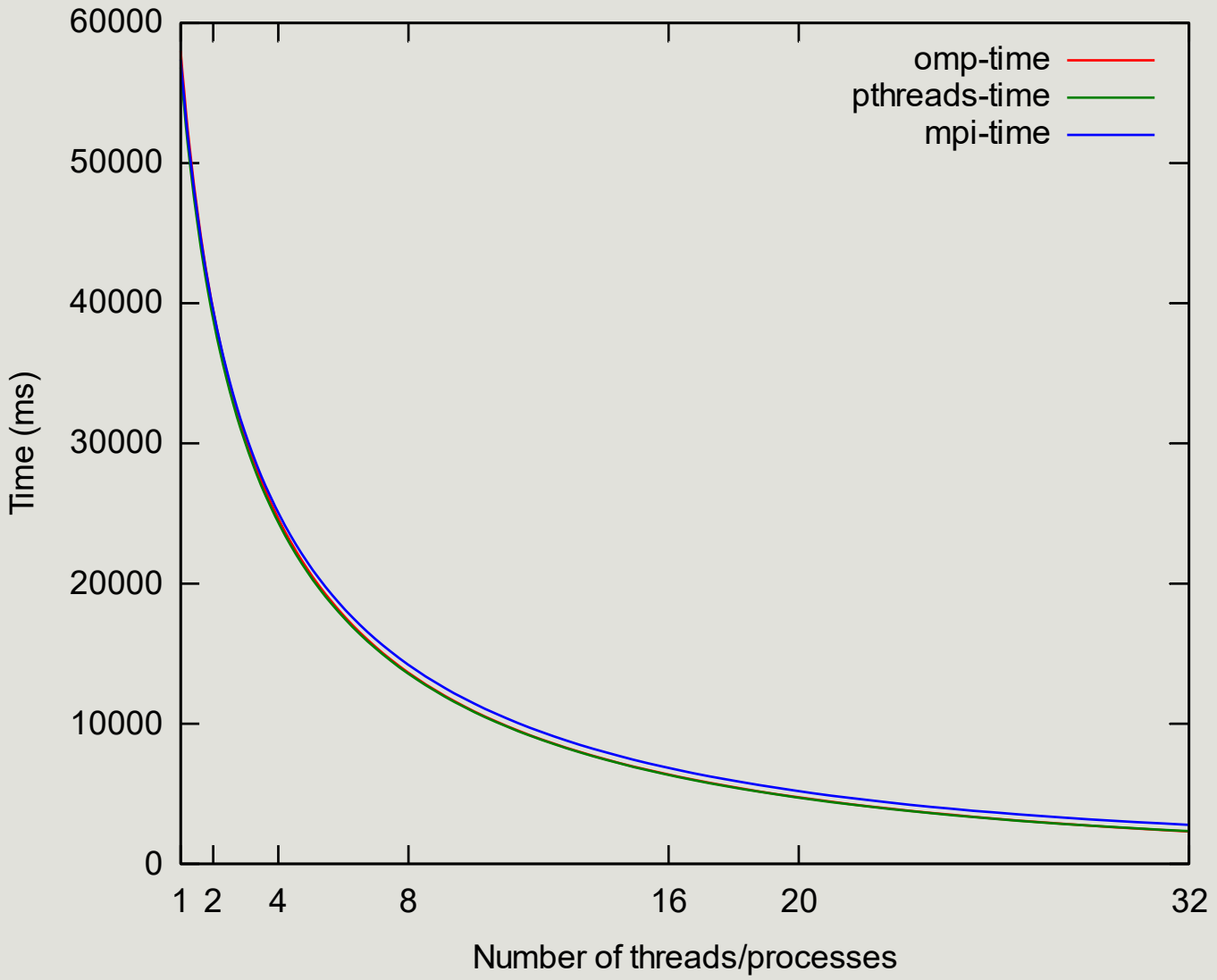
MPI

Se paralelizează calculul celor două sume prezentate în formula anterioară.

Fiecare proces calculează câte o subsumă din cele două sume (pare&impare)

Cum nu există dependințe între date, nu există nevoia de sincronizare între procese.

Colectarea subsumelor se face folosind operația specială *MPI_Reduce* ce permite însumarea rezultatelor parțiale în timp ce acestea sunt primite de la workeri.

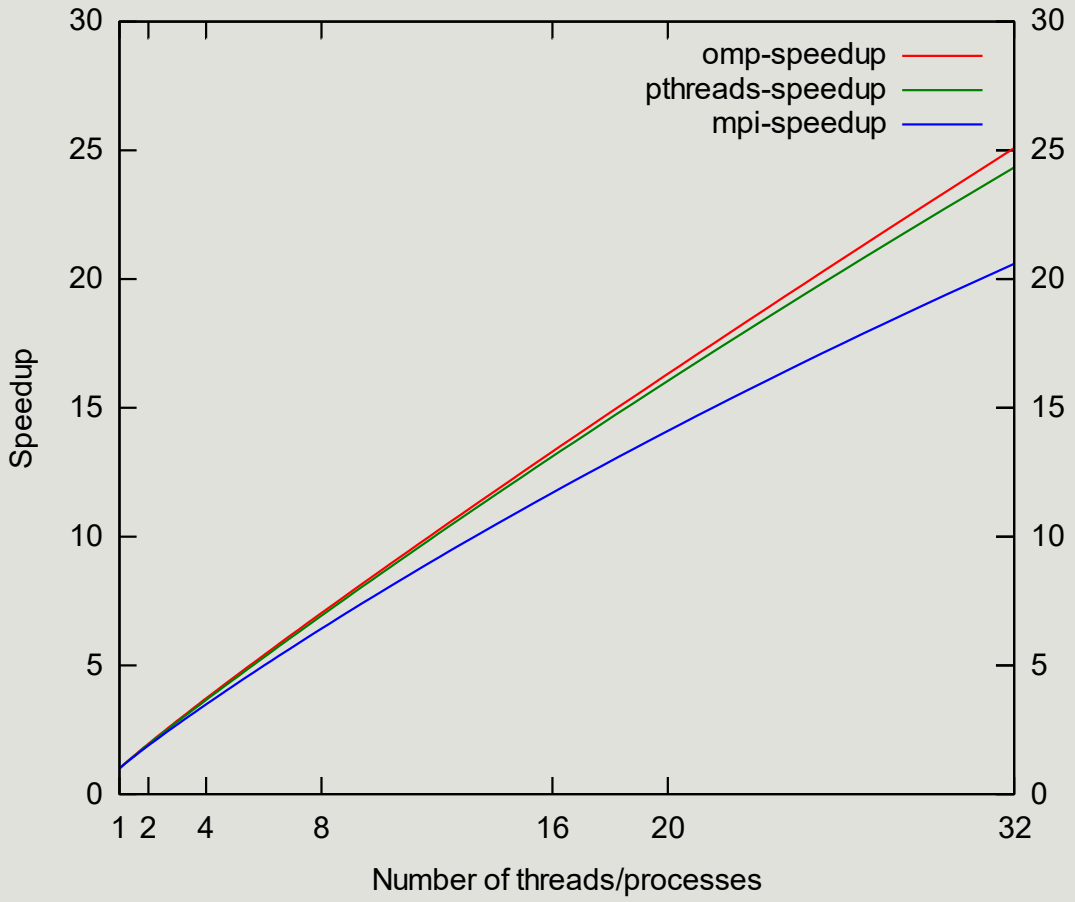


Scalabilitate

Graficul prezintă evoluția timpului de rulare o dată cu creșterea numărului de unități de procesare.

Se observă că implementările OpenMP și pthreads se comportă identic.

Implementarea MPI este în general mai lentă, indiferent de numărul de procese. (se datorează overhead-ului generat de crearea și comunicarea interprocess)



Speedup

Graficul prezintă speedup-ul implementărilor.

Cum problema aleasă este embarrassingly parallel, era de așteptat ca graficul de speedup să aibă o caracteristică liniară.

Se observă, din nou, că soluția MPI distribuită pe mai multe procese se comportă mai slab în comparație cu soluțiile ce împart același spațiu de memorie.

Implementare hibridă: OpenMP + MPI

Threads \ Processes	1	2	4	8
1	56971	29220	15407	8377
2	29239	16532	8283	4492
4	15799	8510	4420	2483
8	8611	4469	2669	2065

În slide-urile precedente am observat că implementările OpenMP și pthreads au comportament aproape identic. Din acest motiv am ales tratez doar implementarea hibridă ce folosește OpenMP + MPI.

Implementarea hibridă lansează *P* procese MPI, iar fiecare proces va porni *T* fire de execuție. În total sunt *P***T* unități de procesare în paralel.

Matricea de mai sus conține timpii de rulare(ms) pentru diferite combinații ale parametrilor *P* și *T*.

Comparând elementele (x, y) cu (y, x) nu se observă diferențe majore. În alte cuvinte, nu contează dacă sunt pornite **X** procese și **Y** thread-uri sau **Y** procese și **X** thread-uri, atât timp cât în final numărul total de unități de procesare este același. (**X*****Y** = **Y*****X** = **XY**)

Concluzii

Implementările OpenMP și pthreads se comportă aproape identic din punct de vedere al scalabilității.

Implementarea MPI este mai lentă decât implementările OMP și pthreads.

Diferența vine din faptul că MPI folosește procese ca unități de execuție. Evident, astfel se consumă mai mult timp pentru a le crea, configura, trimite/primi date și a le opri.

În comparație, firele de execuție din cadrul unui proces nu necesită toți pașii enumerați mai sus.

În schimb, implementarea MPI este utilă în cazul în care se dorește ca paralelizarea să folosească mai multe procesoare aflate în mașini diferite în cadrul unui datacenter.