APD - Lab1

Tuesday, October 02, 2007

12:20 PM

Algoritmi paraleli si distribuiti, curs an 5

Corina Stratan: corina@cs.pub.ro

programare (impartire problema in subprobleme)
- Paralela: multiprocesor
- Distribuita: intre mai multe calculatoare, de regula
- Concurenta: folosesc aceleasi resurse dar nu fac aceeasi sarcina

Firele de executie se creeaza in cadrul unui proces si se creeaza mai repede decat procesele.

Comunicarea se poare face:
- In fire de executie separate, cu resurse partajate
- Prin mesaje, intre procese, eventual chiar intre masini separate

Limbaje pentru prelucrare paralela: SR, Paralaxis, **biblioteci pentru programe C**
Ex:
- OpenMP: creare de fire de executie; OpenMP e portabil (pthreads din C e specific Linux)
- MPI: message passing interface: pe sisteme distribuite, intre care se schimba mesaje
- Java: suporta nativ fire de executie

Open MP
Compilator: Omni Open MP
Comanda: ompcc (similar gcc): ex "ompcc -o program program.c"

De inclus header: #include <omp.h>
De setat o variabila de mediu: "export OMPC_NUM_PROCS = 5"

Numarul de fire de executie nu trebuie sa fie fix, depinzand de masina pe care ruleaza. Open MP incearca sa aloce automat firele de executie.

Program OpenMP: fork-join
Pretate la cicluri mari.
Fork la creare de fire, join la terminarea calculelor.

Directive pragma: pentru compilator, optiuni specifice date de programator; compilatorul ce nu recunoaste pragma-urile le ignora
Directivele actioneaza asupra unei regiuni de cod, intre {}.
Pentru o linie de cod sau o singura instructiune compusa nu e nevoie de {}.

Clauze: optiuni la directive

#pragma omp <directiva> [clauze]

Directiva de creare a unei regiuni paralele:

```
#pragma omp parallel
{
    //cod
}
```

Va incepe sa execute in paralel cu toate threadurile disponibile zona de cod.

#pragma omp parallel num_threads(5)
Sau:
omp_set_num_threads(5);
Apelata inaintea crearii regiunii paralele. La directiva parallel se creeaza automat 5 threaduri.

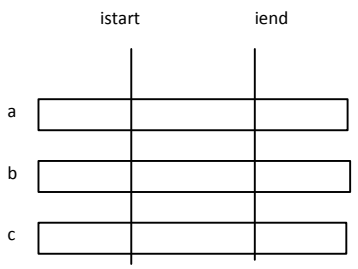
Forteaza 5 threaduri.
De fapt va aloca atatea threaduri cate procesoare are, deci poate sa aloca mai putine (5 e un maxim).

omp_set_dynamic(0);
1: dinamic, 0: static
Setarea din variabila de mediu il face sa considere automat ca are atatea procesoare cate s-au specificat, nu mai e nevoie de dynamic

int omp_get_num_threads();
Cate threaduri sunt in mod curent
Rulat in regiune paralela, arata cate threaduri ruleaza acolo; in afara regiunii paralele (ar trebui sa arate) numarul definit la set_num_threads()

int omp_get_thread_num();
Ce id are thradul curent

N elemente
Pot fi impartite in mod egal intre threaduri



Program OpenMP de adunare a doi vectori:
Primul thread aduna primele x elemente, al doilea urmatoarele y, etc… vectorul A+B=C

```
#pragma omp parallel
{
    Int id, I, Nthreads, istart, iend; //indici intre care ruleaza fiecare thread
    Id = omp_get_thread_num();
    Nthreads = omp_get_num_threads();
    Istart = id * N / Nthreads;
    Iend = (id+1) * N / Nthreads;
    For(i=istart, i<iend;i++)
        c[i]=a[i]+b[i];
```

}

Alta directiva, for, generaaza threadurile automat:

```
#pragma omp parallel
#pragma omp for
For(i=0, i<N, i++)
        c[i]=a[i]+b[i];
```

Race condition: rezultat nedeterminist

```
#pragma omp critical
{
```

Regiune critica: secventa de cod care nu poate fi executata la un moment dat decat de un singur thread

- Diferenta fata de cod in afara "parallel": executata de mai multe threaduri, dar nu simultan
- Codul din afara "parallel" este executat o singura data doar de threadul master.

```
{

#pragma omp barrier
{

}
```

Bariera: punct In program la care se asteapta toate threadurile sa ajunga inainte de a se trece mai departe

- La sfarsitul regiunii paralele este o bariera implicita; nu se iese din directiva parallel sau for pana cand n-au ajuns toate threadurile acolo

Obs:

```
#pragma omp parallel for
```

Iterativ:

```
#include<stdio.h>
#include<stdlib.h>
#include <omp.h>
static long num_steps = 10000;
double step;
void main()
{
        int I;
        double x,pi,sum=0.0;
        step = 1.0/(double)num_steps;
        for(i=1;i<num_steps;i++)
        {
                x=(i-0.5)*step;
                sum=sum+4.0/(1.0+x*x);
        }
        pi=step*sum;
}
```

Paralelizare:

(in shell:) export OMPC_NUM_PROCS=4
(fara export-ul de mai sus, programul da eroare la rulare spunand ca nu poate aloca 4 threaduri, conform lui opm_set_num_threads(), ci doar unul)

```
#include<stdio.h>
#include<stdlib.h>
#include <omp.h>
static long num_steps = 10000;
double step;
int main()
{
        int i;
        double x,pi,sum=0.0;
        step = 1.0/(double)num_steps;
        omp_set_num_threads(4)
#pragma omp parallel for reduction (+:sum)
{
        for(i=1;i<num_steps;i++)
        {
                x=(i-0.5)*step;
                sum=sum+4.0/(1.0+x*x);
        }
}
        pi=step*sum;
}
```

pentru paralelizare, s-ar putea pune sum= intr-o regiune critica

altfel,

un vector de sume si fiecare thread calculeaza suma lui

sau,

```
#pragma parallel for reduction(+:sum)
```

ce operatie se face la reducere (adunare) si cu ce variabila (sum)

Reducere = operatia de reunire a threadurilor