Lab 5b - Applicazioni openMP

https://elly2023.smfi.unipr.it/mod/page/view.php?id=5067&forceview=1

OBIETTIVO
Parallelizzare heat con openMP
- Heat: Analisi dello speedup ed eventuali ottimizzazioni.

Parallelizzare factorize con openMP
- Factorize scaling

Attività svolte

Parallelizzazione Heat con openMP

Dopo aver creato la directory "openMP/heat" ho copiato il file sorgente seriale heat.c, rinominandolo in omp_heat.c, e ho fatto le seguenti modifiche:
- Aggiungere l'intestazione "#include <omp.h>"
- Aggiunto la direttiva "#pragma omp for" nella funzione Jacobi

```
 99  /*************************************/
100  /* JACOBI ITERATION FUNCTION - CPU */
101  /*************************************/
102  void Jacobi_Iterator_CPU(float * __restrict T, float * __restrict T_new, const
     int NX, const int NY)
103  {
104  int i,j;
105
106          // --- Only update "interior" (not boundary) node points
107          #pragma omp parallel for private (j,i)
108          for(j=1; j<NY-1; j++)
109              for(i=1; i<NX-1; i++) {
110                  float T_E = T[(i+1) + NX*j];
111                  float T_W = T[(i-1) + NX*j];
112                  float T_N = T[i + NX*(j+1)];
113                  float T_S = T[i + NX*(j-1)];
114                  T_new[NX*j + i] = 0.25*(T_E + T_W + T_N + T_S);
115              }
116  }
```

- Determinare i tempi di calcolo T_start, T_stop e T_total (T_stop-T_start) e Tj (tempo jacobi).

```
57  T_start=omp_get_wtime();
58
59  for(iter=0; iter<MAX_ITER; iter=iter+1)
60  {
61    h_T_temp=h_T_new;  // SWAP
62    h_T_new=h_T_old;
63    h_T_old=h_T_temp;
64
65    Init_cross(h_T_old, NX, NY);
66    Init_center(h_T_old,  NX, NY);
67    Init_left(h_T_old,    NX, NY);
68    Init_top(h_T_old,     NX, NY);
69    copy_rows(h_T_old, NX, NY);
70    copy_cols(h_T_old, NX, NY);
71
72    Tj_start=omp_get_wtime();
73    Jacobi_Iterator_CPU(h_T_old, h_T_new, NX, NY);
74    Tj_stop=omp_get_wtime();
75
76    Tj=Tj+Tj_stop-Tj_start;
77
78    if (debug) { system("clear"); print_colormap(h_T_old); sleep(1);  }
79  }
80
81  T_stop=omp_get_wtime();
82
83  T_total=T_stop-T_start;
```

Dopodiché ho compilato ed eseguito il codice tramite i seguenti comandi:

- gcc -O2 omp_heat.c -fopenmp -o omp_heat
- OMP_NUM_THREADS=8 ./omp_heat > omp_heatmap.out

Output:

```
[martina.genovese@ui01 heat]$ OMP_NUM_THREADS=8 ./omp_heat > omp_heatmap.out
OMP, 8, 512, 512, 1000, 0.745, 0.256
```

Analisi dello speedup ed eventuali ottimizzazioni

Ho realizzato lo script slurm omp_heat_scaling.slurm e lo script python omp_heat_scaling.py per il plot dello strong scaling per superfici di dimensione crescente (2048x2048 e 4096x4096).

[risultato di omp_heat_scaling.dat dopo l'esecuzione dello script]

```
[martina.genovese@ui01 heat]$ more omp_heat_scaling.dat
OMP, 1, 2048, 2048, 1000, 18.324, 6.932
OMP, 2, 2048, 2048, 1000, 19.034, 7.594
OMP, 4, 2048, 2048, 1000, 17.761, 5.329
OMP, 8, 2048, 2048, 1000, 15.626, 3.446
OMP, 16, 2048, 2048, 1000, 14.562, 2.354
OMP, 32, 2048, 2048, 1000, 17.989, 5.573
OMP, 1, 4096, 4096, 1000, 287.318, 32.606
OMP, 2, 4096, 4096, 1000, 265.675, 27.346
```

[Output esecuzione omp_heat_scaling.py]

```
[martina.genovese@ui01 heat]$ python omp_heat_scaling.py
     nt     r     c  iter       t     tj
OMP   1  2048  2048  1000  18.324  6.932
OMP   2  2048  2048  1000  19.034  7.594
OMP   4  2048  2048  1000  17.761  5.329
OMP   8  2048  2048  1000  15.626  3.446
OMP  16  2048  2048  1000  14.562  2.354
OMP  32  2048  2048  1000  17.989  5.573
```

[plot di scaling con i tempi e lo speedup]

Parallelizzazione factorize con openMP

Dopo aver creato la directory "openMP/factorize" ho copiato il file sorgente seriale factorize.c, rinominandolo in omp_factorize.c, e ho fatto le seguenti modifiche:

- Aggiunto l'intestazione "#include <omp.h>"
- Aggiunto la direttiva "#pragma omp parallel for" nella regione parallelizzabile, quindi il ciclo for presente nel main
- Aggiunto l'uso della direttiva critical nelle stampe all'interno della regione parallela, quindi per ogni printf(…) in block_factorize ho aggiunto "#pragma omp critical"

Dopodichè ho realizzato ed eseguito lo script omp_factorize.slurm per compilare ed eseguire il programma.
N:B. nello script ho inserito il modulo precedentemente generato con openssl tramite i comandi:

- openssl genrsa -out rsa_key.pem 68
- openssl rsa -in rsa_key.pem -modulus -noout

```
[martina.genovese@ui01 factorize]$ openssl genrsa -out rsa_key.pem 68
Generating RSA private key, 68 bit long modulus
..++++++++++++++++++++++++++
.++++++++++++++++++++++++++
e is 65537 (0x10001)
[martina.genovese@ui01 factorize]$ openssl rsa -in rsa_key.pem -modulus -noout
Modulus=94555659DD6529C5B
```

Factorize scaling

Factorize scaling

La partizione cpu_guest ha un time-limit di 24 ore. Determinare la dimensione massima in bit di un blocco (MB) che un singolo core può elaborare in 24 ore.
Determinare il numero massimo di blocchi di dimensione MB da analizzare per fattorizzare un modulo di 128 bit.
Progettare ed eventualmente realizzare una tecnica di check-pointing per salvare lo stato di avanzamento del programma omp_factorize, per poterlo riprendere in un run successivo.
Scrivere una breve relazione in factorize_scaling.txt

Codice

…