Politecnico di Milano
Algoritmi e Calcolo Parallelo
Prof. Pier Luca Lanzi Ing. Daniele Loiacono
Prima prova in itinere
10 Febbraio 2011

COGNOME E NOME

MATRICOLA

Spazio riservato ai docenti

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili. È possibile usare il libro di testo e un manuale di C++ privo di annotazioni. **Non è permesso usare supporti contenente esercizi svolti (eserciziari, slide, ecc.). Non è possibile consegnare esercizi scritti in matita. Non consegnare altri fogli. Un esercizio lasciato in bianco corrisponde a 0 punti. Gli studenti che hanno svolto il progetto dovranno risolvere soltanto gli esercizi 2, 4 e 5.**

Esercizio 1 (5 pt). Date le coordinate **x** e **y** di un punto **P** nel piano cartesiano ed **N** rettangoli, calcolare il numero di rettangoli che contengono il punto **P**. Ogni rettangolo è definito dalle coordinate di due vertici opposti **Q (x1,y1)** e **R(x2,y2)** (tali che x1<x2 e y1<y2), come nel seguente esempio:

Implementare un kernel CUDA che possa essere utilizzato per risolvere tale problema. Definire opportunamente i parametri in ingresso del kernel implementato e descrivere le eventuali strutture dati utilizzate per rappresentare nella memoria della scheda grafica i dati del probleama.

Esercizio 2 (3 pt). Dato il seguente frammento di programma in C++ (dove la funzione **load** legge da tastiera un vettore di N elementi):

```
int a[N],b[N],c[N];
#pragma omp parallel
{
  a = load(N);
  for (int i=1; i<N; i++)
      a[i] = a[i] + a[i-1];
  b = load(N);
  for (int i=1; i<N; i++)
      b[i] = b[i] + b[i-1];
  for (int i=0; i<N; i++)
      c[i] = a[i]*b[i];
}
```

Aggiungere le direttive OpenMP necessarie per ottenere una corretta implementazione parallela del frammento.

Esercizio 3 (4 pt). Scrivere un programma in MPI per realizzare in parallelo lo smoothing di un array A, di cui viene fornita l'implementazione seriale (dove la funzione **load** legge da tastiera un vettore di N elementi):

```
float *a,*b;
int N;
cin >> N;
a = load(N);
b = new float[N];
for (int i=0; i<N; i++) {
  float s = a[i];
  int j=1;
  if (i>0)   { s +=  a[i-1]; j++;}
  if (i<N-1) { s +=  a[i+1]; j++;}
  b[i] = s / j;
}
```

Esercizio 4 (3 pt). Si implementino due thread (usando lo standard POSIX) che hanno il seguente comportamento:

A. thread **deposita**: riceve come parametro di ingresso una variabile **x** di tipo **float** ed incrementa di **x** il valore della variabile condivisa **capitale (**sempre di tipo **float)**.

B. thread **preleva**: riceve come parametro di ingresso una variabile **x** di tipo **float** e decrementa di **x** il valore della variabile condivisa **capitale (**sempre di tipo **float)** se questa ha valore maggiore o uguale ad **x**; in caso contrario, si sospende in attesa che il valore della variabile **capitale** aumenti.

Nota: Non occorre implementare il programma principale (main) ma è sufficiente implementare le due funzioni principali dei thread e dichiarare le eventuali variabili globali utilizzate.

Esercizio 5 (3 pt). Applicare l'algoritmo di Huffman per costruire l'albero di codifica/decodifica ottimo per il seguente messaggio: "AFFALAFFELAF". Codificare quindi il messaggio utilizzando l'albero di codifica ottenuto.