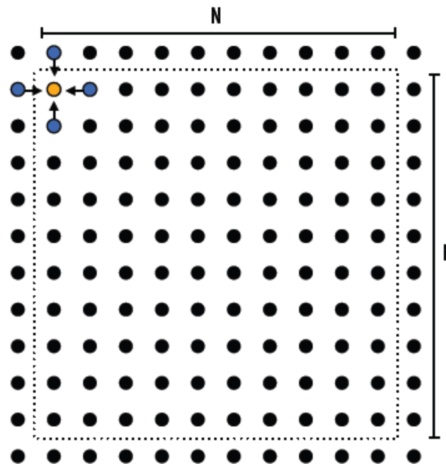
PDE SOLVER

Riccardo Fontanini

Consegna

Solve the following Gauss-Seidel iterative problem on a (N+2) x (N+2) grid



Change the algorithm to a more adequate for parallelism:

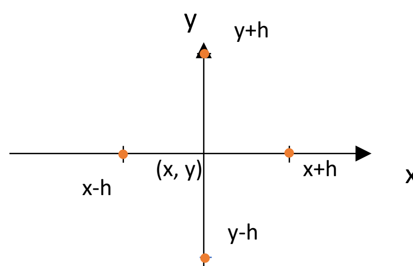
- realize that no matter the iteration scheme, the final solution is going to be the same
- change the cell update scheme

Metodo delle differenze finite

il metodo delle differenze finite è una strategia utilizzata per risolvere numericamente equazioni differenziali che, nelle sue varianti, si basa sull'approssimazione delle derivate con equazioni alle differenze finite.

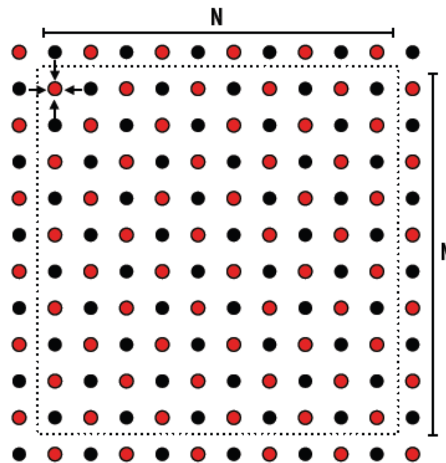
In pratica scegliendo una variazione costante h andremo a definire un campo discreto all'interno dello spazio del nostro problema, delimitato da una frontiera I in cui vale una equazione differenziale, per esempio l'equazione di Laplace. Tale problema quindi può essere discretizzato in base alla griglia così creata. Da letteratura si nota che partendo dall'intorno è possibile approssimare il valore della funzione nei punti di intersezione della griglia mediando rispetto ai suoi "vicini".

$$\phi(x,y) = \frac{1}{4}\left(\phi(x+h,y) + \phi(x-h,y) + \phi(x,y+h) + \phi(x,y-h)\right)$$



Coloring

Per ottenere una parallelizzazione efficace è stata sfruttata la tecnica del coloring cioè si è assegnato ad ogni punto un "colore", in modo che nell'intorno di un punto nero ci siano solamente punti rossi e vice versa.



Di fatto si è scelto di elaborare alternativamente i punti rossi e i punti neri, così da completare una evoluzione completa del sistema in due fasi. Ogni fase esegue la media sui vicini per ogni punto del colore di fase, terminata questa si passa alla fase successiva, alternando quindi i punti da elaborare.

Sistema per il test

I test sono stati eseguiti su una macchina server che sfrutta processori Intel Xeon CPU E5-2603 e una scheda grafica NVIDIA K40.

Codice e algoritmi

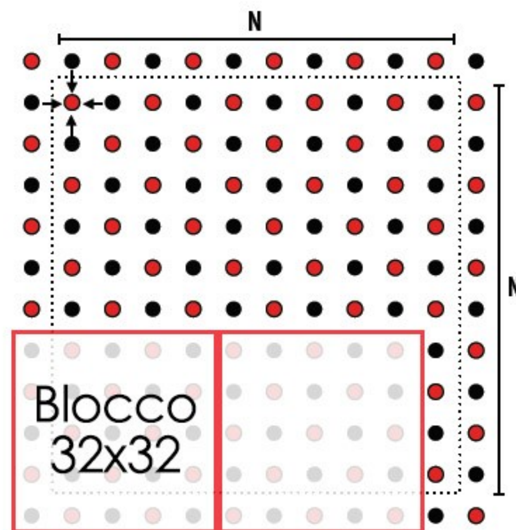
L'algoritmo base per l'elaborazione del sistema rispecchia quello espresso nella consegna:

```
const int N = aNumber;
const float TOLERANCE = aTolerance;
const int MAXITER = aMaxNumber;
float *A;

bool solve (float *A) {

  float diff, old, norm = N * N;
  bool done = false;
  maxit = 0;

  while (!done) {
    maxit++;
    diff = 0;
    for (int i = 1; i < N; i++) {
      for (int j = 1; j < N; j++) {
        old = A[i][j];
        A[i][j] = 0.2 * (A[i][j] +
                     +  A[i-1][j] + A[i+1][j] +
                     +  A[i][j-1] + A[i][j+1]);
        diff += fabs (A[i][j] - old);
      }
    }
    if (diff / norm) < TOLERANCE)
       done = true;
    if (done || maxit > MAXITER)
      return done;
  }
}
```

La maggiore sfida è stata l'adattamento per CUDA.

Mapping su CUDA

Si è scelto di mappare il problema in modo molto semplice: per sfruttare al massimo le potenzialità della GPU si è deciso di creare una griglia di blocchi da 32 x 32 thread (1024 è il massimo numero di thread per blocco). Ad ogni thread è stato assegnato un punto da elaborare. L'intero campo è stato diviso quindi in aree 32 x 32.



Al termine dell'elaborazione di ogni fase, cioè elaborazione dei punti rossi o neri, avviene una sincronizzazione di tutti i blocchi. In questo modo non è possibile che un blocco si trovi in uno stato di elaborazione più avanzato rispetto ad un altro. I thread che corrispondono ai punti di colore diverso rispetto alla fase in elaborazione in quel momento sono sospesi: entreranno in funzione con il cambio di fase.

Compilazione

Per la compilazione (in sistemi linux) è stato creato un makefile, una volta invocato il comando:

make pde

permette la compilazione di tutte le librerie utilizzate e la creazione dell'eseguibile nella cartella:

./build/linux

La compilazione avviene sfruttando lo standard C99.

È possibile specificare alcuni parametri aggiuntivi in fase di compilazione utilizzando la regola -D di gcc:

make pde D=SIMULATION

Esegue una simulazione "visuale" di come lavora il sistema.

make pde D=TOLLERANCE=xx

Imposta la tolleranza per fermare l'elaborazione

make pde D=N_x=100,N_y=100

Imposta la grandezza del campo

Make pde D=MAXITER=20

Impostare il numero massimo di iterazioni di fase

Consigliata:

make pde D=ELABCPU,N_x=10000,N_y=10000,SHOWRESULT,MAXITER=10000

Problematiche

Sincronizzazione

Durante lo sviluppo del programma, si è notato che poteva nascere una asincronia nell'evoluzione dei vari blocchi della griglia. Quindi si è deciso di strutturare il programma in modo che la variazione di fase (elaborazione punti neri - punti bianchi) sia scandita dalla CPU e non direttamente dalla GPU. Al cambio di fase è stata inserita una istruzione sincronizzante per due motivi:

- eliminare le asincronie
- possibilità di visualizzare l'elaborazione in tempo reale anche se non è completata l'elaborazione

Tale soluzione ha permesso di gestire facilmente la sincronizzazione dei blocchi, ma ha prodotto un calo prestazionale.

Analisi del sistema

Confronto CPU vs GPU

N_x	**N_y**	**Iterazioni massime**	**Tempo CPU**	**Tempo GPU**
100	100	100	3600	389
2000	2000	100	21759	23530
10000	10000	100	101120	95497
100	100	1000	50169	3567
2000	2000	1000	408516	413365
10000	10000	1000	1339962	786170
100	100	10000	349710	32932
2000	2000	10000	3994918	3775890
10000	10000	10000	14252461	6648360

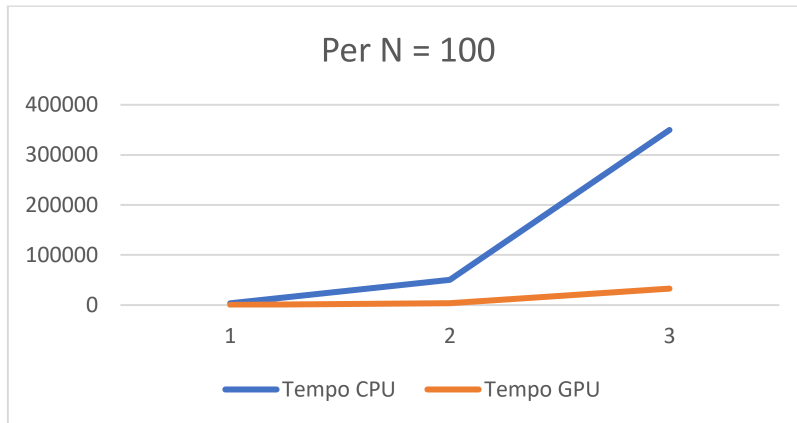
I tempi della tabella sono espressi in microsecondi.

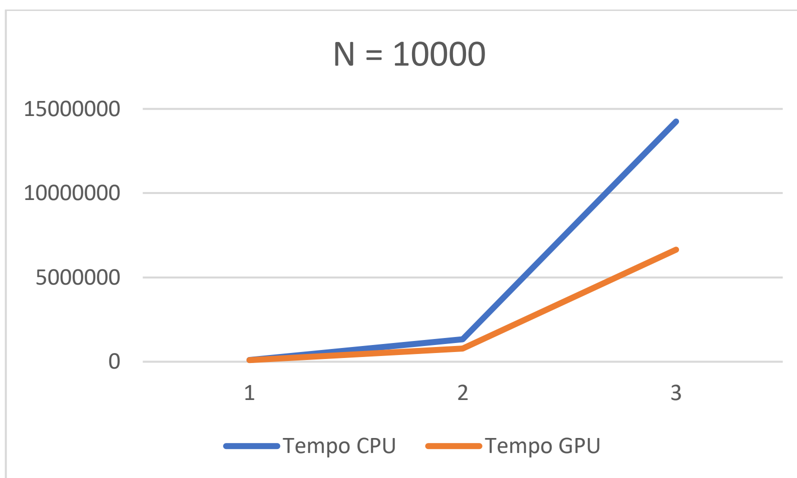
Nella tabella sono presenti la grandezza del campo (N_x e N_y), le iterazioni massime eseguite dall'algoritmo e vengono confrontati i tempi di elaborazione dell'algoritmo in CPU e GPU.

Per la valutazione delle prestazioni della GPU è stato utilizzato il tool NVPROF:

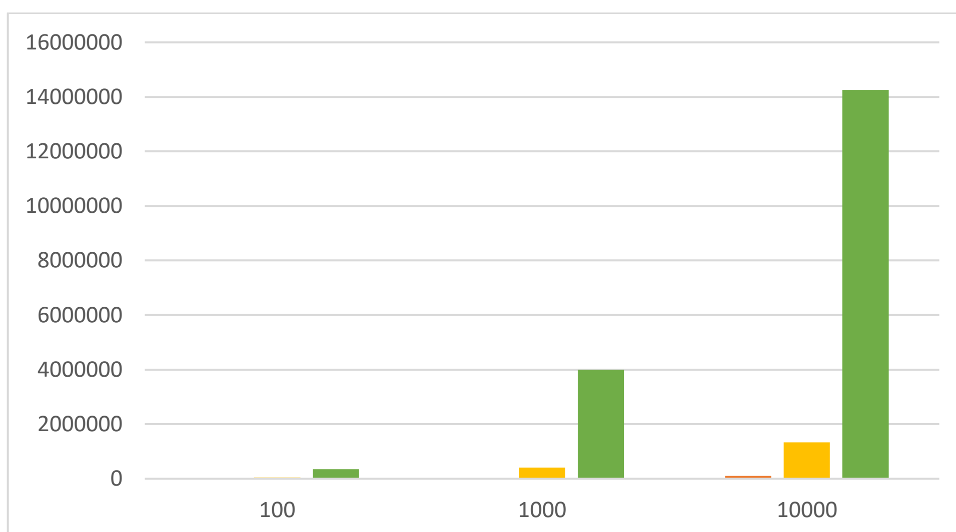
```
==31615== Profiling application: ./pde
==31615== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.98%  6.64836s     10000  664.84us  24.928us  35.773ms  solve(double*, int)
                    0.02%  1.1671ms         2  583.53us  12.193us  1.1549ms  set_default(double*, double)
      API calls:   94.03%  18.2838s     10004  1.8276ms  16.108us  37.898ms  cudaDeviceSynchronize
                    3.51%  681.86ms     10002  68.172us  23.820us  8.1106ms  cudaLaunch
                    2.38%  463.07ms         1  463.07ms  463.07ms  463.07ms  cudaMallocManaged
                    0.05%  9.2086ms     20004     460ns     230ns  896.56us  cudaSetupArgument
                    0.03%  5.3329ms     10002     533ns     380ns  5.7720us  cudaConfigureCall
                    0.01%  1.3590ms       188  7.2280us     277ns  360.51us  cuDeviceGetAttribute
                    0.00%  193.55us         2  96.773us  75.987us  117.56us  cuDeviceTotalMem
                    0.00%  111.00us         2  55.501us  53.096us  57.906us  cuDeviceGetName
                    0.00%  2.9200us         3     973ns     257ns  2.1000us  cuDeviceGetCount
                    0.00%  2.8890us         4     722ns     288ns  1.3310us  cuDeviceGet
```

Come si può vedere dalla tabella, le prestazioni CPU sono abbastanza costanti nell'andamento variando la dimensione del problema, mentre quelle GPU possiedono una grande variabilità, possiamo notarla meglio dal grafico.





Inoltre in termini assoluti possiamo notare come l'elaborazione CPU cresce esponenzialmente all'aumentare del numero di punti da elaborare e dal numero massimo di iterazioni da eseguire.



Simulazione

Nel programma, inserendo il parametro SIMULATION durante la compilazione, è possibile entrare in modalità simulazione. Tale modalità abilita una versione rallentata dell’algoritmo per permettere di apprezzare l’evoluzione del sistema in modo che l’occhio umano possa distinguerla. Tale modalità può essere impiegata per debugging. Di seguito sono esposti degli esempi di elaborazione simulata.

/home/fontanini/repo/pde_solver/build/linux

```
Ciclo: 6
        GRID

120 120 120 120   0   0   0   0   0   0
  0  57  75  71  36  27  23  20  11   0
  0  34  52  54  46  41  39  31  20   0
  0  27  43  49  48  48  44  39  22   0
  0  24  42  47  49  49  47  39  25   0
  0  25  39  47  49  49  46  41  23   0
  0  22  39  44  47  46  45  37  24   0
  0  20  31  39  39  41  37  34  19   0
  0  11  20  22  25  23  24  19  13   0
  0   0   0   0   0   0   0   0   0   0
```

/home/fontanini/repo/pde_solver/build/linux

```
Ciclo: 37
        GRID

120 120 120 120   0   0   0   0   0   0
  0  55  72  66  28  15   9   6   3   0
  0  30  44  44  31  22  15  10   5   0
  0  18  30  32  29  23  18  12   6   0
  0  13  21  26  25  23  18  13   7   0
  0   9  17  21  22  20  17  12   7   0
  0   7  12  16  17  17  14  11   5   0
  0   5   9  11  13  12  11   7   4   0
  0   2   4   6   6   6   5   4   2   0
  0   0   0   0   0   0   0   0   0   0
```

Sommario

Consegna 1
Metodo delle differenze finite 1
Coloring 2
Sistema per il test 2
Codice e algoritmi 2
Mapping su CUDA 3
Compilazione 3
Problematiche 4
Sincronizzazione 4
Analisi del sistema 4
Confronto CPU vs GPU 4
Simulazione 6