Progetto Algoritmo di Crook

Struttura del codice

Nel blocco del main del programma, inizializzo la struttura di mpi e le variabili per ottenere il numero totale di thread e l'indice del thread, rispettivamente "com_size" e "rank". Per problemi, ho scelto di usare thread solo multipli di tre, per evitare di ottenere dei lavori di diversa grandezza. A questo punto, grazie al calcolo dei trapezi, mi calcolo quante celle un thread deve elaborare. Successivamente, in base al rank del thread, mi calcolo da dove deve iniziare il thread a lavorare le celle.

Il thread master (rank = 0) crea/ottiene il sudoku da risolvere, la struttura è una lista contigua di interi, le posizioni vuote sono indicate da uno 0.

Tutti gli altri thread si fermano su una barriera, poiché all'istruzione dopo, i thread dovrebbero chiamare il Bcast, se il thread master non ha pronto il sudoku, questo genera una eccezione.

Superata la barriera, tutti i thread eseguono il Bcast, ottenendo tutta la lista del sudoku. Ogni thread inizia ad elaborare la sua parte di sudoku, itera dalla cella specificata dal suo lavoro per il numero di celle che ogni thread deve elaborare.

Nell'elaborare la cella, controlla se questa è uguale a zero, allora inizia a cercare tutti i possibili valori (1-9) che possono andar bene. Bisogna controllare se nella riga, nella colonna e nel quadrato se c'è o meno il valore che si sta verificando. Se non c'è in nessuno di questi 3 spazi, allora inserisci il numero in una struttura chiamata "note".

Finito il lavoro del thread, questo controlla se nella sua struttura (locale al thread) ci sono dei possibili valori composti da 1 solo numero segnato, questi valori vengono scritti nel sudoku e vengono cancellati dalle note.

A questo punto bisogna ricomporre il sudoku e iniziare il cuore dell'algoritmo di Crook.

Creo una sotto stringa del sudoku modificato in locale di ogni thread, la sottostringa contiene tutti e le celle che sono state lette dal thread. Usando il Gather ricompongo il sudoku nel thread master.

Limitazioni riscontrate

Dovendo suddividere il lavoro tra thread, usando gli indici calcolati con il metodo dei trapezi, risultava complicato usare una struttura liste di liste.

Risultati delle prestazioni

Avendo implementato solo parte del programma, sono mostrati i risultati parziali.

Multi Thread

Il programma con ben 2 miliardi di cicli riesce a trovare i possibili valori nelle varie celle e a risolvere i valori ovvi (i singleton nelle note).

Il print del sudoku è dopo aver risolto i singleton nelle note.

```
0 0 0 1 8 9 3 6 0
0 7 0 2 3 4 0 1 8
0 0 0 0 5 7 2 0 9
1 6 4 3 7 0 9 8 5
8 0 3 5 4 1 6 7 2
2 5 7 0 9 0 1 3 4
5 0 2 7 6 0 0 0 0
9 0 1 4 0 8 0 5 0
0 8 6 0 1 5 0 0 0

 Performance counter stats for 'mpiexec --allow-run-as-root -n 3 crook_threaded_run':

         1.421,58 msec task-clock                #    0,933 CPUs utilized
            1.691      context-switches          #    0,001 M/sec
               32      cpu-migrations            #    0,023 K/sec
            8.897      page-faults               #    0,006 M/sec
    2.963.296.520      cycles                    #    2,085 GHz                      (49,12%)
                0      stalled-cycles-frontend                                     (52,35%)
                0      stalled-cycles-backend    #    0,00% backend cycles idle      (54,15%)
                0      instructions              #    0,00  insn per cycle
                         (50,88%)
                0      branches                  #    0,000 K/sec                    (47,65%)
                0      branch-misses             #    0,00% of all branches          (45,85%)

      1,523996661 seconds time elapsed

      0,212869000 seconds user
      0,973112000 seconds sys
```

Multi Thread (Ottimizzato)

La ottimizzazione -O3 riduce il numero di context switches, essendo più veloce di 1 decimo di secondo.

```
0 0 0 1 8 9 3 6 0
0 7 0 2 3 4 0 1 8
0 0 0 0 5 7 2 0 9
1 6 4 3 7 0 9 8 5
8 0 3 5 4 1 6 7 2
2 5 7 0 9 0 1 3 4
5 0 2 7 6 0 0 0 0
9 0 1 4 0 8 0 5 0
0 8 6 0 1 5 0 0 0

 Performance counter stats for 'mpiexec --allow-run-as-root -n 3 crook_threaded_optimized_run'

         1.396,73 msec task-clock                #    0,914 CPUs utilized
            1.654      context-switches          #    0,001 M/sec
               35      cpu-migrations            #    0,025 K/sec
            8.865      page-faults               #    0,006 M/sec
    2.944.885.954      cycles                    #    2,108 GHz                      (53,10%)
                0      stalled-cycles-frontend                                     (50,25%)
                0      stalled-cycles-backend    #    0,00% backend cycles idle      (46,16%)
                0      instructions              #    0,00  insn per cycle
                         (46,90%)
                0      branches                  #    0,000 K/sec                    (49,75%)
                0      branch-misses             #    0,00% of all branches          (53,84%)

      1,528366228 seconds time elapsed

      0,220059000 seconds user
      0,961463000 seconds sys
```