Paralelização e Análise de Escalabilidade

Flávio Henrique Lopes Barbosa
Jaysa Keylla Siqueira Barbosa
José Augusto Agripino de Oliveira

Decomposição LU Lower Upper

Um dos motivos para introduzir a decomposição LU é que ela fornece uma maneira eficiente de calcular a matriz inversa, a qual tem muitas aplicações na engenharia; ela também fornece um meio de avaliar o condicionamento do sistema.

GCC

O GCC é um compilador com recursos avançados e com as flags de otimização, o programador pode indicar ao compilador para que procure por laços e otimize-os .Tais recursos foram aplicados ao código de decomposição LU para matrizes.

Resultados

-Matriz de ordem 4000

-Matriz de ordem 6000

-Matriz de ordem 8000

-Matriz de ordem 12000

-Matriz de ordem 16000

Comandos utilizados

1. $ gcc -pg -Wall omp_decomposicaoLU.c -o omp_decomposicaoLU.o -fopenmp
2. $ ./decomposicaoLU (parâmetro nº Threads)
3. $ gprof omp_decomposicaoLU gmon.out > omp_orden_n_T_x.txt

Trecho do código Paralelizado

```
20  void preencheMatriz(float *A, int thread_count) {
21  #   pragma omp parallel num_threads(thread_count) default(none) \
22          shared(A)
23      for(int i=0; i<n; i++) {
24          for(int j=0; j<n; j++) {
25              A[(i*n)+j] = rand()%100000;
26
27          }
28      }
29  }
```

```
32  void calculaMatrizes(int i, int j, float *aux, float *M, int thread_count) {
33      M[(i*n)+j] = aux[(i*n)+j]/aux[(j*n)+i];
34
35      //multiplica a linha do multiplicar por -(multiplicador)
36      //Movimenta a coluna
37  #   pragma omp for
38      for(int c=j; c<n; c++){
39          aux[(i*n)+c] = aux[(i*n)+c]+aux[(j*n)+c]*(-1*(M[(i*n)+j]));
40      }
41  }
```

```
43  void gauss(float *A, float *M, float *aux, int thread_count){
44  #   pragma omp parallel num_threads(thread_count)
45      {
46      #   pragma omp for
47          for(int i=0; i<n; i++) {
48              for(int j=0; j<n; j++) {
49                  M[(i*n)+j] = 0;
50                  aux[(i*n)+j] = A[(i*n)+j];
51              }
52          }
53          //Calculando os multiplicadores da Gauss
54          //Movimenta a coluna
55          for(int j=0; j<(n-1); j++){
56          //Movimenta a linha
57              for(int i=j+1; i<n; i++){
58                  ( aux[(i*n)+j] != 0 ) ? calculaMatrizes(i, j, aux, M, thread_count) : NULL;
59          }//Todos os multiplicadores da coluna j estão definidos
60          }
61      #   pragma omp for
62          for(int i=0; i<n; i++) M[(i*n)+i]=1;
63      }
```

Análise de Escalabilidade

Entrada / Threads	1	4	8	16
4000	77.11s	75.79s	75.13s	119.18s
6000	278.57s	280.31s	274.93s	401.34s
8000	677.59s	576.62s	627.14s	1045.73s
12000	2061.47s	2093.02s	2144.85s	3093.00s
16000	4955.13s	4943.75s	4937.96s	5715.28s

Análise de SpeedUp (tempo serial / tempo paralelo)

Entrada / Threads	1	4	8	16
4000	1	1.02	1.03	0.65
6000	1	0.99	1.01	0.70
8000	1	1.18	1.08	0.65
12000	1	0.98	0.96	0.67
16000	1	1.01	1.01	0.87

Análise de Eficiência (SpeedUp / nº de Threads)

Entrada / Threads	1	4	8	16
4000	1	0.26	0.13	0.04
6000	1	0.25	0.01	0.04
8000	1	0.26	0.14	0.04
12000	1	0.25	0.12	0.04
16000	1	0.25	0.13	0.05

Conclusão

Os resultados obtidos com a palalelização foram melhores que os obtidos de forma sequencial, no geral.

Com 16 threads apresentou uma piora considerável pelo computador utilizado possuir apenas 8 cores.

O tempo foi influenciado pelos fatores do computador, como o tempo de uso e a memória física ser um HD.

O gargalo ainda continua sendo a função calculaMatrizes. O seu tempo de execução melhorou um pouco com o uso das threads.