Arquitetura de Computadores 2 - Trabalho Final - Programação Paralela

*GIOVANNA ALBURQUERQUE, LUCIANO PEREIRA E RYAN MATHEUS*

1. CODIGO ORIGINAL

Abaixo o código original, contendo 3 loops, o primeiro usando loop para preencher 4 arrays com operação de multiplicação com double e os outros dois loops fazem o papel de realizar multiplicações entre valores dos arrays

1. #include <chrono>
2. #include <iostream>
3. // GRUPO : LUCIANO PEREIRA, GIOVANNA ALBURQUERQUE, RYAN MATHEUS
4. static long NUM = 10000;
5. double step;
6. using namespace std;
7. int main () {
8. double a[NUM], b[NUM], c[NUM], d[NUM]; // arrays
9. int i,j; // loop counters
10. auto start = std::chrono::*high\_resolution\_clock*::now();
12. for (i=0; i < NUM; i++) {
13. a[i] = i \* 1.0;
14. b[i] = i \* 2.0;
15. c[i] = i \* 3.0;
16. d[i] = i \* 4.0;
17. }
18. for (i=0; i < NUM-2; i++) {
19. for (j=0; j < NUM-2; j++) {
20. a[i] = b[i] + c[i] \* d[j];
21. }
22. }
23. for (i=0; i < NUM; i++) {
24. d[i] = a[i] \* a[i];
25. }
26. auto finish = std::chrono::*high\_resolution\_clock*::now();
27. auto duration = std::chrono::duration\_cast<std::chrono::*microseconds*>(finish - start).count();
28. float time = duration / 1000000.0;
29. // print time
30. cout << "Time: " << time << " seconds" << endl;
31. }

Tabela de tempo de execução com N = 10000

(Desconsiderado a primeira execução devido ao cache miss e o consequente tempo discrepante aos demais testes)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| Execução 1 | 0.272001 | 0.239 | 0.354988 |
| Execução 2 | 0.269 | 0.227033 | 0.355017 |
| Execução 3 | 0.280987 | 0.222999 | 0.359786 |
| Média | 0.274 | 0.229677 | 0.3566 |

1. OTIMIZAÇÃO DE MEMÓRIA

Para otimizar o código foi feito a substituição de variáveis de ponto flutuante para inteiros bem como o uso de apenas uma matriz ao invés de 4 arrays

#include <chrono>

#include <iostream>

// GRUPO : LUCIANO PEREIRA, GIOVANNA ALBURQUERQUE, RYAN MATHEUS

static long NUM = 10000;

double step;

using namespace std;

int main () {

    //double a[NUM], b[NUM], c[NUM], d[NUM];

    int effarray[NUM][4];

    int i,j, temp;

    auto start = std::chrono::*high\_resolution\_clock*::now();

    for (i=0; i < NUM; i++) {

        temp = i \* 1;

        effarray[i][0] = temp;

        effarray[i][1] = temp \* 2;

        effarray[i][2] = temp \* 3;

        effarray[i][3] = temp \* 4;

    }

    for (i=0; i < NUM; i++) {

        for (j=0; j < NUM; j++) {

            effarray[i][0] = effarray[i][1] + effarray[j][2];

        }

    }

    for (i=0; i < NUM; i++) {

        effarray[i][3] = effarray[i][0] \* effarray[i][0];

    }

    auto finish = std::chrono::*high\_resolution\_clock*::now();

    auto duration = std::chrono::duration\_cast<std::chrono::*microseconds*>(finish - start).count();

    float time = duration / 1000000.0;

    // print time

    cout << "Time: " << time << " segundos" << endl;

}

Tabela de tempo de execução com N = 10000

(Desconsiderado a primeira execução devido ao cache miss e o consequente tempo discrepante aos demais testes)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| Execução 1 | 0.225999 | 0.184994 | 0.333088 |
| Execução 2 | 0.225998 | 0.190997 | 0.327285 |
| Execução 3 | 0.226998 | 0.196997 | 0.32019 |
| Média | 0.22633 | 0.191 | 0.32685 |

1. OTIMIZAÇÃO NA COMPILAÇÃO

Sendo usado o código otimizado em memoria acima, foram compilados 3 arquivos usando a seguinte argumentação:

* g++ -O1 '.\projeto com otimização de memoria.cpp' -o projetoO1
* g++ -O2 '.\projeto com otimização de memoria.cpp' -o projetoO2
* g++ -O3 '.\projeto com otimização de memoria.cpp' -o projetoO3

A seguir será apresentado os valores de tempo para cada grau de otimização (O1, O2, O3)

Tabela de tempo de execução com N = 10000

|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| O1 |  | 0.056989 |  |
| O2 |  | 0.061028 |  |
| O3 |  | 0.060991 |  |

1. OTIMIZAÇÃO DE PARALELIZAÇÃO

Sendo usado o código otimizado em memoria acima, a seguir será apresentado os valores de tempo para cada número de threads usadas

Tabela de tempo de execução com N = 10000

|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| 1 Thread |  | 0.202995 |  |
| 2 Threads |  | 0.125032 |  |
| 3 Threads |  | 0.104511 |  |
| 4 Threads |  | 0.098022 |  |
| 5 Threads |  | 0.103873 |  |
| 6 Threads |  | 0.105993 |  |
| 7 Threads |  | 0.102997 |  |
| 8 Threads |  | 0.102996 |  |

Conclui-se a partir dos dados acima expostos que quando o numero de threads ultrapassa o número de CPUs de uma máquina a otimização se torna irrelevante

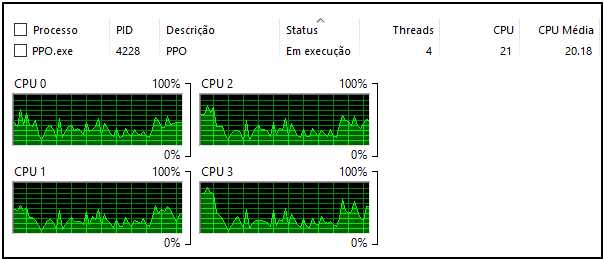
Tabela de tempo de execução com N = 50000

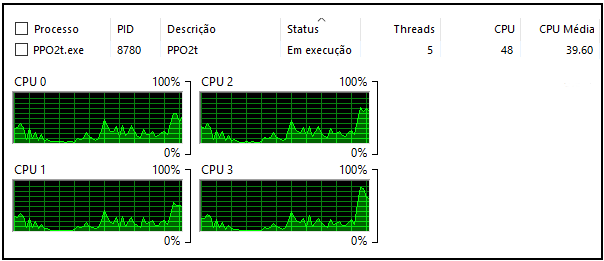
|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| 1 Thread |  | 4.91612 |  |
| 2 Threads |  | 3.03005 |  |
| 3 Threads |  | 2.37024 |  |
| 4 Threads |  | 2.2135 |  |

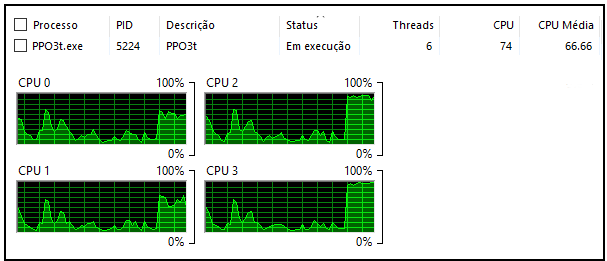
Tabela de tempo de execução com N = 10000 + Otimização O1

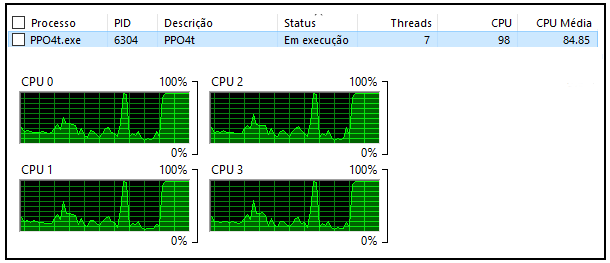
|  |  |  |  |
| --- | --- | --- | --- |
|  | Giovanna | Luciano | Ryan |
| 1 Thread |  | 0.064829 |  |
| 2 Threads |  | 0.041515 |  |
| 3 Threads |  | 0.035011 |  |
| 4 Threads |  | 0.027966 |  |

Uso nas CPUs (N=120000)

Uma Thread no Código

Duas Threads no Código

Três Threads no Código

Quatro Threads no Código

Conclusões

1. COMPARAÇÃO