Cálculo de Pi com Soma de Elementos
(Semana 3 Computação Concorrente)

Aluno: Thierry Pierre Dutoit
DRE: 119040432

Ao aumentar o número de elementos como parâmetro do programa sequencial vemos um aumento na precisão do valor de pi. Podemos ver isso acontecendo na tabela a seguir:

Sequencial	Valor de PI Calculado	Erro
10^1 Elementos	3.041839618929403	0.099753034660390
10^3 Elementos	3.140592653839794	0.000999999749999
10^5 Elementos	3.141582653589720	0.000010000000073
10^7 Elementos	3.141592553589792	0.000000100000002
10^9 Elementos	3.141592652588050	0.000000001001743

Infelizmente o tempo de execução para $10^{(10)}$ elementos é muito alto, dificultando os testes. No mais, a tendência é que a precisão do valor de pi aumente com o aumento do número de elementos.

No programa concorrente tentou-se minimizar o erro numérico. Para isso, foi dado para cada thread um bloco contínuo de elementos, a fim de fazer operações com valores próximos entre si. Por serem valores próximos, diminuímos a chance de arredondamento da máquina nos valores somados.

Os valores encontrados estão mostrados a seguir:

10^1 Elem.	Tempo de Execução	Valor de PI Calculado	Erro
Sequencial	0.000009	3.041839618929403	0.099753034660390
1 thread	0.000245	3.041839618929403	0.099753034660390
2 threads	0.000281	3.041839618929403	0.099753034660390
4 threads	0.000251	3.041839618929403	0.099753034660390
8 threads	0.000334	3.041839618929403	0.099753034660390
10^3 Elem.	Tempo de Execução	Valor de PI Calculado	Erro
Sequencial	0.000048	3.140592653839794	0.000999999749999
1 thread	0.000227	3.140592653839794	0.000999999749999
2 threads	0.000280	3.140592653839795	0.000999999749999
4 threads	0.000229	3.140592653839791	0.000999999750002
8 threads	0.000427	3.140592653839794	0.000999999749999
10^5 Elem.	Tempo de Execução	Valor de PI Calculado	Erro
Sequencial	0.004343	3.141582653589720	0.000010000000073
1 thread	0.003175	3.141582653589720	0.000010000000073
2 threads	0.001735	3.141582653589780	0.000010000000013
4 threads	0.001723	3.141582653589787	0.000010000000006
8 threads	0.001684	3.141582653589778	0.000010000000015
10^7 Elem.	Tempo de Execução	Valor de PI Calculado	Erro
Sequencial	0.287849	3.141592553589792	0.000000100000002
1 thread	0.302483	3.141592553589792	0.000000100000002
2 threads	0.148659	3.141592553589780	0.000000100000014
4 threads	0.102366	3.141592553589743	0.000000100000050
8 threads	0.110536	3.141592553589832	0.000000099999961
10^9 Elem.	Tempo de Execução	Valor de PI Calculado	Erro
Sequencial	29.487745	3.141592652588050	0.000000001001743
1 thread	29.497056	3.141592652588050	0.000000001001743
2 threads	14.790061	3.141592652589258	0.000000001000535
4 threads	9.499196	3.141592652589210	0.000000001000583
8 threads	9.553023	3.141592652589324	0.000000001000469

No início da tabela, como o número de elementos é baixo, não vemos distinção nos valores de PI do programa sequencial e do programa concorrente.

Conforme N (número de elementos) aumenta, começamos a ver uma maior distinção entre os resultados, onde o programa concorrente sempre leva vantagem em precisão. Isso se dá pois o programa concorrente opera com valores mais próximos entre si, diminuindo o erro gerado.

Porém, como usamos uma precisão maior (double) em ambos os programas, a diferença de precisão entre eles é mínima, não tendo muita relevância.

Dessa forma, o uso de threads se justifica no ganho de tempo de execução. Com N pequeno, o programa sequencial leva vantagem devido à complexidade do programa concorrente e no overhead da criação de threads. Com N >= 10^5 elementos já vemos o programa concorrente tendo alto ganho de desempenho se comparado ao sequencial:

10^1 Elem.	Tempo de Execução	Ganho
Sequencial	0.000009	**1**
1 thread	0.000245	**0,036734694**
2 threads	0.000281	**0,03202847**
4 threads	0.000251	**0,035856574**
8 threads	0.000334	**0,026946108**
10^3 Elem.	Tempo de Execução	
Sequencial	0.000048	**1**
1 thread	0.000227	**0,211453744**
2 threads	0.000280	**0,171428571**
4 threads	0.000229	**0,209606987**
8 threads	0.000427	**0,112412178**

10^5 Elem.	Tempo de Execução	
Sequencial	0.004343	**1**
1 thread	0.003175	**1,367874016**
2 threads	0.001735	**2,503170029**
4 threads	0.001723	**2,520603598**
8 threads	0.001684	**2,578978622**
10^7 Elem.	Tempo de Execução	
Sequencial	0.287849	**1**
1 thread	0.302483	**0,951620422**
2 threads	0.148659	**1,93630389**
4 threads	0.102366	**2,811959049**
8 threads	0.110536	**2,604119925**
10^9 Elem.	Tempo de Execução	
Sequencial	29.487745	**1**
1 thread	29.497056	**0,999684341**
2 threads	14.790061	**1,993754116**
4 threads	9.499196	**3,104235874**
8 threads	9.553023	**3,086744897**

Observamos o real ganho do uso de threads com valores muito grandes de N, como $N = 10^9$. O ganho chega a ser no máximo a 3x pois, por mais que minha máquina tenha 4 núcleos, ela não poderá dedicar todos à execução do programa.

Concluímos então que os valores, seja do programa sequencial ou seja do programa concorrente, são muito parecidos. Isso acontece pois a conta feita é a mesma e a precisão usada (double) não abre muito espaço para erros de cancelamento.

Se fosse uma questão de precisão o uso do paralelismo não se justificaria. No entanto, para valores grandes de N, o ganho de desempenho no cálculo da soma justifica o trabalho da implementação de threads.