Título
MAC0438 - Programação Concorrente

André Meneghelli Vale - 4898948
andredalton@gmail.com

Marcello Souza de Oliveira - 6432692
mcellor210@gmail.com

1 Cálculo por série infinita de π

1.1 Fórmula:

A fórmula escolhida foi a *Fórmula de Bellard*[1]. Esta escolha se deve ao fato de poder ser calculado um enésimo dígito de pi na base 2 sem a necessidade de se calcular qualquer um dos termos anteriores, o que torna o programa possível de ser totalmente paralelizável. Esta fórmula foi desenvolvida a partir da fórmula *Bailey–Borwein–Plouffe*[2] (`BBP`) que por sua vez foi inspirada na série infinita para o cálculo de uma aproximação da inversa da arcotangente.

$$\pi = \frac{1}{2^6}\sum_{n=0}^{\infty}\frac{(-1)^n}{2^{10n}}\left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9}\right) \tag{1}$$

$$\pi = \sum_{i=0}^{\infty}\left[\frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right)\right]. \tag{2}$$

1.2 Vetor:

Para o cálculo foi definida uma estrutura de `N long int` chamada de `SuperLong`, cujo vetor principal estará no elemento `data`:

```
typedef struct superLong{
       unsigned long int *data;
       unsigned long int n;
} SuperLong;
```

1.3 Thread:

O objetivo é poder armazenar os dados calculados por cada uma das threads em uma estrutura de dados que não faz arredondamentos. As threads irão incrementar um buffer em variável `long double`. Desta maneira não é necessário realizar uma conversão de base depois de terminado o cálculo, o que irá economizar uma quantidade considerável de tempo.

Cada vez que um limite de precisão que o vetor esta trabalhando é alcançado as threads ficam paradas em uma barreira enquanto a primeira thread faz a sincronização dos resultados e atualiza algumas variáveis calculadas durante a execução do programa. Essas variáveis serão explicadas na próxima sessão.

1.4 Reaproveitando os termos da série:

Podemos observar que o algorítimo escolhido repete muitas vezes algumas multiplicações e o cálculo da potência inicial. Desta maneira decidimos utilizar 5 variáveis globais que irão manter parte dos valores calculados a cada sincronização:

- `N`: Número do ultimo termo a ser calculado no momento da sincronização;
- `NO`: Número do ultimo termo a ser calculado no momento da sincronização anterior. Inicializada com *0*;

- `m4`: Inicializada com *0* e incrementada a cada sincronização com *(N-N0)*4*. Desta forma os denominadores terão o formato *m4+4*(n-N)+k*, onde `n` é o termo que a thread esta calculando e `k` é a constante presente em cada um dos denominadores;

- `m10`: Inicializada com *0* e incrementada a cada sincronização com *(N-N0)*10*. Desta forma os denominadores terão o formato *m10+10*(n-N)+k*, seguindo a mesma representação do ítem anterior;

- `p2`: Inicializada com 2^6 e multuplicada a cada sincronização por $2^{(N-N0)*10}$. Então no final do cálculo o termo, até agora contendo a soma das parcelas, será dividido por $(p2 * 2^{10(n-N)})$;

Outro cálculo que é desnecessário é o de $(-1)^n$, esta parte será substituida por um `if` para definir a parcela negativa em caso de `n` ímpar e positiva caso contrário.

Referências

[1] `http://en.wikipedia.org/wiki/Pi`

[2] `http://en.wikipedia.org/wiki/Bellard%27s_formula`

[3] `http://en.wikipedia.org/wiki/Bailey-Borwein-Plouffe_formula`