MAC0438 – Programação concorrente – 1s2014
EP2
Data de entrega: 22/05/2014

Prof. Daniel Macêdo Batista

1 Introdução

Da wikipedia: "*Na matemática, o número de Euler, denominado em homenagem ao matemático suíço Leonhard Euler, é a base dos logaritmos naturais. As variantes do nome do número incluem: número de Napier, constante de Néper, número neperiano, constante matemática, número exponencial etc. A primeira referência à constante foi publicada em 1618 na tabela de um apêndice de um trabalho sobre logaritmos de John Napier. No entanto, este não contém a constante propriamente dita, mas apenas uma simples lista de logaritmos naturais calculados a partir desta. A primeira indicação da constante foi descoberta por Jakob Bernoulli...*" (`http://pt.wikipedia.org/wiki/N%C3%BAmero_de_Euler`)

2 Problema

O número de Euler (e) aparece em diversos tópicos da matemática e diversas aproximações existem para calculá-lo. Uma dessas aproximações é derivada da série de Taylor e define e como:

$$\sum_{n=0}^{\infty} \frac{1}{n!} \tag{1}$$

Somatórios infinitos como este acima costumam ser implementados de forma paralela com o objetivo de encontrar a melhor aproximação para uma dada constante. Quanto maior a quantidade de termos do somatório, mais preciso o valor, desde claro que o hardware e a linguagem de programação utilizada tenham a precisão necessária para representar o número sendo calculado. Encontrar valores mais precisos de somatórios como este acima costuma ser considerado como um bom benchmark para avaliar novas arquiteturas de super computadores. Quanto mais preciso e mais rápido o cálculo das aproximações, melhor tende a ser visto o computador. Claro que de nada adianta um hardware poderoso se a implementação do código não for tão boa quanto e não souber tirar proveito do hardware.

Em implementações paralelas de somatórios (na verdade de séries infinitas de um modo geral) cada termo pode ser calculado de forma independente por uma thread/processo. De tempos em tempos, os termos são avaliados a fim de se verificar se o critério de parada foi atendido. O critério de parada pode ser por exemplo a diferença entre dois valores consecutivos ou o mínimo valor de um termo. O critério de parada pode ser verificado através de uma barreira de sincronização. No caso de somatórios, essa barreira pode por exemplo ter também o papel de somar todos os termos encontrados naquela iteração em uma variável global.

Sua tarefa neste EP será implementar um algoritmo paralelo que calcule uma aproximação para o valor de e. Seu programa deve ser implementado de modo a parar em duas condições (o usuário deve informar na linha de comando qual critério de parada ele quer): quando a diferença entre o valor de e em duas iterações consecutivas for menor do que um valor `f` passado como entrada para o programa ou quando alguma thread calcular um termo menor do que um valor `m` também passado como entrada para o programa.

3 Requisitos e entregas

3.1 Fórmula para cálculo de e

Você pode implementar qualquer fórmula que calcule uma aproximação de e, desde que essa fórmula seja definida como uma série infinita. Sinta-se livre para escolher a fórmula que você quiser. Converse com amigos dos outros cursos do IME, pesquise no google, na wikipedia, etc... Escolher a fórmula é a primeira etapa do EP.

3.2 Threads e barreira de sincronização

O seu EP terá dois modos de operação. No primeiro ele deverá criar uma quantidade de threads igual à quantidade de núcleos do computador (n) e essas threads deverão se sincronizar sempre que todas elas terminarem de encontrar n novos termos da série infinita. Ao fim de cada iteração uma variável global contendo o valor de e até o momento deve ser atualizada. O programa deve parar (i) quando a diferença entre dois valores de e calculados por iterações consecutivas for menor do que o valor `f` passado como parâmetro para o seu programa; ou (ii) quando alguma thread calcular um termo menor do que um valor `m` passado como parâmetro para o seu programa.

O segundo modo de operação difere do primeiro porque o usuário precisa passar a quantidade de threads que ele deseja criar. Ou seja, nesse modo o seu programa não precisa descobrir o valor de n já que o usuário vai informar quantas threads ele quer.

A barreira de sincronização utilizada no seu programa pode vir pronta de uma biblioteca ou de forma nativa da linguagem de programação que você utilizar. Você deve informar no LEIAME do seu programa qual barreira você utilizou.

IMPORTANTE: Seu programa não pode utilizar informações pré-calculadas que facilitem o cálculo do valor de e. Por exemplo, se a fórmula que você utilizar para calcular o valor de e precisa computar fatoriais para cada termo, você deve encontrar os fatoriais em tempo de execução. Utilizar uma tabela com fatoriais pré-calculados ou qualquer outra tabela com números pré-calculados que facilitem o cálculo dos termos implicará em nota ZERO no EP.

IMPORTANTE: Você não pode utilizar nenhuma biblioteca que já faça o cálculo de e sozinha. O cálculo deve ser feito por você implementando alguma fórmula descrita como uma série infinita. Programas que utilizem funções já prontas para calcular e implicarão em nota ZERO no EP.

3.3 Linguagem

Seu programa pode ser escrito em qualquer linguagem de programação que tenha compilador ou interpretador de código aberto disponível para o GNU/Linux.

3.4 Entrada e saída

O seu programa receberá como entrada obrigatoriamente na linha de comando e **nesta ordem**:

- primeiro argumento: um valor inteiro maior ou igual a zero. Quando for igual a zero significa que seu programa deve criar uma quantidade de threads igual à quantidade de núcleos do computador. Quando for diferente de zero significa que seu programa deve criar uma quantidade de threads igual a esse valor;
- segundo argumento: um caracter `f` ou `m`. Se for `f` significa que o EP deve parar quando a diferença entre dois valores de e calculados por iterações consecutivas for menor do que o valor do terceiro argumento. Se for `m` significa que o EP deve parar quando alguma thread calcular um termo menor do que o valor do terceiro argumento;
- terceiro argumento: um valor real que terá significado a depender do segundo argumento passado pelo usuário;
- quarto argumento (opcional): um caracter `d` ou um caracter `s` (ou nenhum desses dois parâmetros será passado ou apenas um será passado) Quando nenhum parâmetro opcional for passado na linha de comando o seu programa deverá imprimir apenas ao término da execução:
  - Número de iterações, que vai ser um contador que armazena quantas vezes as threads se encontraram na barreira;
  - Valor encontrado de e.

  Quando a opção `d` for passada na linha de comando o seu programa deverá imprimir:
  - A cada iteração, ordem com que as threads chegaram na barreira. Cada thread precisa de um identificador;
  - A cada iteração, valor parcial de e;
  - Ao término da execução, o número de iterações, que vai ser um contador que armazena quantas vezes as threads se encontraram na barreira;
  - Ao término da execução, o valor encontrado de e.

  Quando a opção `s` for passada na linha de comando o seu programa deverá calcular a fórmula de forma sequencial **sem a utilização de threads ou processos paralelos**. O programa deverá imprimir:
  - Valores parciais de e calculados **a cada novo termo encontrado**;
  - Ao término da execução, o valor encontrado de e.
  - Ao término da execução, a quantidade de termos encontrados.

O seu programa precisa ser entregue no paca até as 08:00 do dia 22/05. A cada hora de atraso a nota máxima da entrega será descontada de 1,0.

3.5 Relatório

Junto com o seu programa você deve entregar um relatório em .pdf apresentando uma análise de desempenho que você realizou no seu programa. O formato do relatório é livre mas deve conter obrigatoriamente:

- Configuração da(s) máquina(s) onde você executou o seu programa;
- Entradas (valores de `f` e `m`) que foram passadas para o seu programa;
- Quantidade de repetições que foram executadas para cada entrada;
- Gráficos comparando o desempenho, em termos do tempo de execução, do seu programa paralelo com diversas quantidades de threads e da versão sequencial para diversas entradas. Esses gráficos precisarão conter médias aritméticas e alguma informação de dispersão como desvio padrão, intervalo de confiança, etc...;
- Explicação de como o tempo de execução foi medido.
- Explicações sobre os resultados obtidos: foi o esperado? Não foi o esperado? etc...

O relatório deve ser entregue junto com o programa no arquivo .tar.gz. Ou seja, este .tar.gz deve conter os seguintes itens:

- fonte, Makefile (ou similar), arquivo LEIAME;
- relatório em .pdf.

O nome do arquivo .tar.gz deve ser `ep2-membros_da_equipe.tar.gz` e quando desempacotado ele deve produzir um diretório contendo os itens. O nome do diretório deve ser `ep2-membros_da_equipe`. Por exemplo: `ep2-joao-maria`.

Obs.: O EP pode ser feito individualmente ou em dupla.

4 Bônus para o programa mais eficiente

Os EPs que receberem nota 10,0 passarão por uma avaliação extra para verificar qual é o mais eficiente. Os códigos serão executados em diversos cenários definidos pelo professor e aquele que rodar, em média, mais rápido devolvendo os valores mais precisos de e será considerado o mais eficiente. Os autores deste EP ganharão 1,0 extra na média final.