*PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS*

*Instituto de Ciências Exatas e Informática*

*Departamento de Ciência da Computação - Curso de Ciência da Computação*

*Disciplina: Seminários II Segundo semestre de 2019*

*Professor: Saulo Augusto de Paula Pinto —* [***saulo@pucminas.br***](mailto:saulo@pucminas.br)

***Segundo Roteiro/Tarefa (25 pontos)***

***Speed-up, seções críticas e reduções***

**Atenção: poste no SGA um arquivo texto (editável) contendo seu código até 23:59h do dia 13/10/2019.**

Antes da tarefa, faça o roteiro do Prof. Luís Fabrício (Cabra) que está em <http://www.eitas.com.br/tutorial/12/31>. É importante fazer o roteiro, pois são apresentados e aplicados conceitos importantes que serão utilizados na tarefa.

Inspirado pelo roteiro que usa o cálculo de π, escreva um programa apenas que faça o que é descrito. Lembre-se que se trata de código experimental, então é permitido fazer coisas não recomendadas pelas boas práticas de programação, como usar efeito colateral em funções: imprimir mensagens na saída e/ou atualizar parâmetros passados por referência [com referências de C++ (‘&’) ou ponteiros (‘\*’) de C/C++].

1. Calcule o valor de π com precisão arbitrária informada pelo usuário, caso ele não queira a precisão *default* (padrão) que é 0.000000001. Utilize, para este cálculo, a série alternada infinita de Leibniz (“série de Leibniz”):



Como a precisão não é simplesmente o número de termos, mas é dada pela diferença absoluta entre dois valores sucessivos de π, ou seja, é o valor absoluto de um termo, você pode calcular o número de termos necessários para alcançar a precisão. Isso é necessário, pois o OpenMP precisa saber o número de iterações do *loop* a ser paralelizado.

Escreva uma função para encapsular seu código do cálculo de PI.

1. Escreva uma versão paralelizada da sua função de cálculo de π. Observe que para fazer isso você tem que tomar alguns cuidados e algumas decisões. Por exemplo, o cálculo envolve adições e subtrações. É possível fazer redução para as duas operações no mesmo *loop*? É melhor usar um *loop* para fazer adições e outro para as subtrações? É melhor usar apenas adições, transformando os termos em positivos/negativos a cada iteração? Você decide!
2. Calcule o *speedup* da versão paralelizada como foi feito no tutorial usado em laboratório. Para isso, você deve usar a função do OpenMP que marque o tempo inicial e final das versões sequencial e paralela: *omp\_get\_wtime()*, que retorna um valor em segundos. Como o sistema operacional é de tempo compartilhado, rodar o programa apenas uma vez pode não resultar em dados confiáveis. Assim, rode suas funções (sequencial e paralela) um certo número N de vezes, N > 10, colete os tempos para cada execução, e reporte o *speedup* médio para as N execuções.