INTRODUÇÃO A COMPUTAÇÃO DE ALTO DESEMPENHO
TRABALHO PRÁTICO 2
OpenMP

DATA DE ENTREGA: 04/07/2016.

INSTRUÇÕES:

1. Deve ser elaborado um relatório contendo os resultados e discussões dos exercícios abaixo. Quando for o caso, o código-fonte também deve ser enviado. O relatório deve ser salvo no formato PDF. O relatório tem peso predominante na nota final de trabalho (60%). Assim, produza um relatório de qualidade!!
2. A entrega será pelo email: camata@nacad.ufrj.br.
3. Serão aceitos trabalhos enviados até às 23:59 do dia da entrega.
4. **ATENCÃO:** No assunto da mensagem, use a seguinte regra:
    a. CAD-2016 - TRAB2 - NOME COMPLETO
5. Trabalhos “similares” podem ser penalizados.

Exercício 1: Perfilando códigos com OpenMP

O objetivo desse exercício é demonstrar o funcionamento de perfiladores de códigos paralelos: TAU + Opari. A seguir é detalhado o processo de instalação e utilização dessas duas ferramentas.

- Pré-requisitos: MPI, PDT
    - MPI: Verique se o mpi está instalado em sua máquina.
        - No linux basta digitar o comando mpicc, por exemplo. Se o comando for reconhecido, pode seguir para a proxima etapa. Caso contrário, instale a biblioteca openmpi-dev e openmpi-bin através do gerenciador de software do linux. (Em sistemas como baseado no debian, apt-get install openmpi-dev openmpi-bin).
        - Para sistemas OSX, pode ser instalado usando macport ou brew.
        - Há possibilidade de compilar o código fonte do MPI para ambos sistemas. Basta seguir a documentacão disponível em: https://www.open-mpi.org/

    - PDT: Responsável pela instrumentacão do código.

        - Faca o download dos arquivos em http://tau.uoregon.edu/pdt_lite.tar.gz
        - Descompacte-o e siga o processo padrão de compilação em ambientes unix/linux.
        - `./configure --prefix=$HOME/local/pdt`
        - `make`
        - `make install`

- Instalando o TAU:

    - Faça o download dos arquivos em http://tau.uoregon.edu/tau.tgz
    - Descompacte-o e configure a instalação usando os seguintes parâmetros
        - `./configure -prefix=$(HOME)/local/tau -pdt=$(HOME)/local/pdt -papi=$(HOME)/local/papi -openmp -opari -opari_region -opari_construct`
            - Note que no comando acima foi incluído o PAPI. Se o PAPI não estiver disponível, a flag -papi=$(HOME)/local/papi pode ser omitida.
        - `make install`
    - Iremos também gerar uma configuração com o MPI que será usada no próximo trabalho prático.
        - `./configure -prefix=$(HOME)/local/tau -pdt=$(HOME)/local/pdt -papi=$(HOME)/local/papi -openmp -opari -opari_region -opari_construct -mpi`

- Utilizando o TAU.

    - Se tudo funcionou corretamente, dentro da pasta $(HOME)/local/tau haverá os seguintes diretórios:
        - `etc examples include man tools x86_64`
    - Incluir no PATH o diretorio contendo os scripts do TAU:
        - Export PATH=$PATH:$(HOME)/x86_64/bin
    - Há necessidade também de definir a variável TAU_MAKEFILE. Observe que foi criado, dentro do diretório $(HOME)/x86_64/lib, um makefile referente as configurações usadas na instalação do TAU.
        - `export TAU_MAKEFILE=$(HOME)x86_64/lib/Makefile.tau-papi-pdt-openmp-opari`

- Testando TAU

Para testar a instalação do TAU usaremos novamente o benchmark STREAM. A compilação agora ser feito da seguinte forma:

```
tau_cc.sh -openmp -O -DSTREAM_ARRAY_SIZE=<N> stream.c -o stream
```

Rode o programa e apos a execução cada thread criará um arquivo profile.0.0.<ThreadID>.

A visualizacão dos dados de perfilagem pode ser realizada de duas formas. Uma através da visualização em formato texto chamando o comando pprof a partir do diretório que contem os arquivos de profiles. A outra forma é a visualizacão gráfica pode ser obtida pelo comando paraprof.

Explore a interface gráfica e gere um gráfico com uma visualizacão 3D. Qual laco consume mais tempo de processamento. Através da analise obtida pelo TAU, você diria que o carga de trabalho entre as threads está adequada?

Mais detalhes podem ser obtidos em:
https://www.cs.uoregon.edu/research/tau/docs.php

Exercício 2: Paralelizando a propagacão da onda.

Paralelize a versão otimizada do código de propagacão da onda usando OpenMP. Justifique as escolhas de implementacão com dados comparativos de perfilagem.

- Teste diversos tipos de escalonamento
- Teste loops com e sem a cláusula collapse
- Teste regiões sem sincronização implícita.

Dentre a melhor configuração obtida, faca o gráfico de speedup e eficiência para verificar a *escalabilidade forte* de sua implementacão.

Exercício 3: Cálculo do π usando Monte Carlo

Se um círculo de raio R é inscrito dentro de um quadrado com um comprimento de lado 2R , então a área do círculo será πR^2 e a área do quadrado vai ser $(2R)^2$. Assim, a razão entre a área do círculo para a área do quadrado será $\pi/4$. Isto significa que, se

você escolher N pontos aleatoriamente dentro do quadrado, aproximadamente N * π/4 desses pontos devem cair dentro do círculo.

Escreva um programa OpenMP que gere pontos aleatoriamente dentro de quadrado unitário. Em seguida, verifique se o ponto está no interior ou não do círculo. O programa mantém o controle de quantos pontos ele foram escolhidos até agora (N) e quantos desses pontos cairam dentro do círculo (M). O valor de π pode ser então aproximado por 4M/N.

Use uma subrotina adequada para obter uma sequencia de números aleatórios distintos entre as threads. Certifique-se de que a adição de mais threads em um cenário de *escalonamento fraco* realmente melhora a estatística.

Exercício 4: QuickSort

Paralelize o QuickSort usando OpenMP. Meça o desempenho paralelo e comente-o. QuickSort ordena uma sequencia de tamanho n (admitido potência de base 2) pelo algoritmo abaixo:

```
Se n = 1, retorna sem fazer nada;
Senão:
     Escolhe um elemento como pivô.
     Particiona os elementos em duas sequencias.
          Elementos menores ou iguais ao pivô
          Elementos maiores que o pivô.
     Aplica QuickSort a cada uma das duas sequencias;
```

O programa fonte deve considerar como único argumento de entrada, um inteiro *k* que define o tamanho da sequencia *(n = 2^k)*. Além disso, o programa deve gerar uma sequencia de inteiros aleatórios do tamanho desejado.

1. Execute a versão sequencial de QuickSort para k=20,21.
2. Execute o procedimento paralelizado utilizando 2, 4 e 8 threads para cada um dos valores de k definidos no item (1).
3. Calcule o speed-up para cada tamanho do problema e para os três números de threads com relação à versão sequencial.
4. Reporte os tempos e o speed-up por meio de tabelas ou gráficos.

DICA.: Utilize OpenMP parallel sections.