Curso: LEF
U.C.: Programação Paralela

Modelo Dez24
Duração: 2h

Nota: Este teste tem 2 partes que devem respondidas em 2 cadernos distintos para permitir que cada um dos docentes da UC possa corrigir em paralelo cada uma das 2 partes.

Parte I (40% - 80 pontos)

1. [30]Considere o seguinte código para execução num processador com uma arquitetura básica elementar.

```
void combine4(float *data, float *dest, int length)
{
  float t = 0;
  for (int i = 0; i < length; i++) {
    t = t + data[i];
  }
  *dest = t;
}
```

a) [10]Apresente um grafo com as dependências de dados para 5 iterações do ciclo `for` (considere apenas as operações aritméticas básicas e os acessos à memória).

b) [10]Calcule quantos ciclos de relógio serão necessários para executar essas 5 iterações do ciclo `for`, numa arquitetura ideal (recursos ilimitados), onde que cada operação demora um ciclo de relógio. **Explique** sucintamente que otimizações permitiriam melhorar o desempenho da execução do código.

c) [10]Os compiladores atuais são bastante eficientes para conseguirem analisar ciclos `for` e vetorizar o código que lá se encontra. Se este código for compilado com indicação para vetorizar, **indique**, justificadamente, se o compilador iria conseguir fazê-lo.

2. [20]**Caracterize** e compare o modelo de memória distribuída com os que usam GPUs, na perspetiva **(i)** da arquitetura do sistema de computação e **(ii)** do respetivo modelo de programação.

3. [10]Os servidores homogéneos *dual-socket* com *chips manycore* da Intel podem tirar partido da memória que é fisicamente partilhada por todos os *cores* do mesmo servidor. **Explique**, justificando, a diferença entre usar um modelo de programação baseado em muitas *threads* num mesmo processo, ou em muitos processos com poucas *threads* cada.

4. [10]Um processador x86-64 para servidores, desenvolvido pela AMD, o EPYC baseado na microarquitetura Zen 3 (também conhecido por "Milan), pode trabalhar com uma frequência de relógio de 3 GHz e ter até 64 *cores*, e cada *chip* tem 8 canais para aceder à memória DDR4-3200 (cada canal tem a capacidade de transferir 25 600 MB/sec).

Se todos os *cores* precisarem de aceder no mesmo ciclo de *relógio* a valores *double-precision FP* que ainda estão em memória (um valor por *core*, e assumindo que as *caches* com dados ainda estão vazias), **mostre**, apresentando os cálculos que efetuar, se a largura de banda agregada deste *chip* é ou não suficiente para satisfazer, com a mesma latência, esta solicitação de todos os *cores*.

5. [10]Duas vezes por ano é divulgada a lista TOP500 com a distribuição ordenada dos 500 sistemas de computação mais potentes a nível mundial. **Mostre** como é construída essa lista.

Parte II (60% -120 pontos)

Considere o programa seguinte, com várias funções, que efetua uma transformação de uma imagem, representada por uma matriz de valores inteiros compreendidos entre 0 e 255, com a dimensão indicada no código e que será executada num servidor com um processador Xeon Ivy Bridge (igual ao utilizado durante as aulas práticas, Intel® Xeon® E5-2695 v2).

```
#define N 10000

int imgIn[N][N];  // 256-gray input image
int imgOut[N][N]; // 256-gray output image

void init(int *h) {
    for(int i=0; i<256; i++)
        h[i]=0;
}

void computeHistogram(int imgIn[N][N], int *h) {
    int j, i;
    for(j=0; j<N; j++)
        for(i=0; i<N; i++)
            h[ imgIn[i][j] ]++;
}

void acumulateHistogram(int *h) {
    int i;
    for(i=1; i<256; i++)
        h[i] += h[i-1];
}

void transformImage(int imgIn[N][N], int *h, int imgOut[N][N]) {
    int j, i;
    for(j=0; j<N; j++)
        for(i=0; i<N; i++)
            imgOut[i][j] = h[ imgIn[i][j] ];
}

int main() {
    int hist[256];

    // ... /* read imgIn ... */

    init(hist);

    computeHistogram(imgIn,hist);

    acumulateHistogram(hist);

    transformImage(imgIn,hist,imgOut);

    // ... /* save imgOut ... */
}
```

1. [30]Pretende-se analisar e otimizar a versão sequencial deste programa.

a) [10]**Indique** a complexidade de cada uma das fases de execução do algoritmo e quais a(s) fases que apresentam maior benefício em ser otimizada(s).

b) [10]**Calcule** o número de *cache misses* no nível 1 originado pela execução da função `computeHistogram` (utilize os parâmetros do processador Xeon Ivy Bridge: 32 KiB de L1 e linhas com 64 bytes).

c) [10]**Apresente** duas otimizações possíveis e **discuta** o seu impacto no tempo de execução com base na equação de desempenho do processador.

2. [30]**Desenvolva** uma versão paralela com **OpenMP**.

a) [10]**Discuta, apresentando** argumentos, se a função `acumulateHistogram` beneficia da execução paralela.

b) [20]**Desenvolva** a versão paralela das funções `computeHistogram` e `transformImage`, e **comente** o propósito de cada uma das diretivas que usar no contexto destas funções.

3. [30]Pretende-se desenvolver um *kernel* em CUDA para implementar a função `transformImage`. Considere o *kernel* CUDA apresentado em baixo e ignore a chamada do *kernel*, as alocações e as cópias de memória entre *host* e *device*.

```
__global__
void transformImage (int *imgIn, int *hist, int *imgOut) {
      int thread_id = threadId.x + blockId.x * blockDim.x;

      ... // transformar imgIn
```

a) [20]**Complete** o *kernel* CUDA apresentado. Quantas *threads* deverão ser lançadas para executar este *kernel*? Justifique.

b) [10]**Desenvolva** uma versão **otimizada** do *kernel* CUDA implementado em **a)** tirando partido de *shared memory*. Quantas *threads* deverão ser lançadas por bloco de *threads*? Justifique.

4. [30]Pretende-se desenvolver uma versão paralela do código para memória distribuída com **passagem de mensagens (i.e., MPI)** e em que o processo 0 contém a imagem original e, no final, deverá conter a imagem processada.

a) [15]**Identifique**, justificando, qual o padrão que se adequa melhor à resolução do problema com MPI, a partir dos padrões paralelos *pipeline* e *farm*. **Apresente** um diagrama com as trocas de mensagens necessárias entre os vários processos, para o padrão escolhido (**ilustre** com 3 processos).

b) [15]**Implemente** a versão proposta anteriormente.