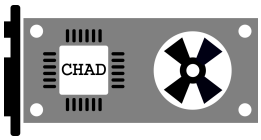


Computação Heterogénea de Alto Desempenho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

LAB 3

– OpenACC –

1. SAXPY (Single-Precision A.X plus Y) é uma operação muito comum em computação e é definida da seguinte forma: ***y=a.x+y***, onde ***a*** é um valor escalar, ***x*** e ***y*** são vetores com N elementos cada. Implemente, usando directivas OpenACC, uma rotina que execute esta operação. Compare a performance com a equivalente versão sequencial.

2. a) Escreva um programa em OpenACC para calcular a aproximação do logaritmo natural (de base *e*) usando os primeiros 10 000 000 de termos da expansão de Maclaurin:

$$ln(1 + x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \ldots \text{ , } para \; -1 < x \leq 1.$$

b) Desenhe a curva de speedup em função do número de iterações.

3. Escreva o código OpenACC de um programa que calcule o valor médio dos elementos que ocupam a parte triangular inferior (i.e., todos os elementos em e por baixo da diagonal) de uma matriz quadrada de grandes dimensões. É possível otimizar o programa de modo a que:

a) A eficiência das transferências de dados seja melhorada (e.g., evitando transferir dados que não são usados nos cálculos efetuados no device)?

b) O trabalho realizado em cada iteração seja equilibrado pelas diferentes threads da GPU? Implemente as otimizações possíveis. Como afetam o desempenho? Teste várias dimensões da matriz.

4. Para fazer debugging de um programa OpenACC, as partes irrelevantes do código foram removidas, restando o seguinte:

```
#include <stdio.h>
const int N=100, M=200;
int main() {
int m[N][M];
for(int i=0; i<N; i++)
   for(int j=0; j<M; j++)
      m[i][j]=1;
#pragma acc kernels
      for(int i=0; i<N; i++)
         for(int j=M-i; j<M; j++)
      m[i][j]=i+j+1;
// verify result
int errcnt=0;
for(int i=0; i<N; i++)
   for(int j=0; j<M; j++) {
      int expect=(j>=M-i)? i+j+1: 1;
      if(m[i][j]!=expect) errcnt++;
   }
printf("Encountered %d errors\n", errcnt);
return errcnt != 0;
}
```

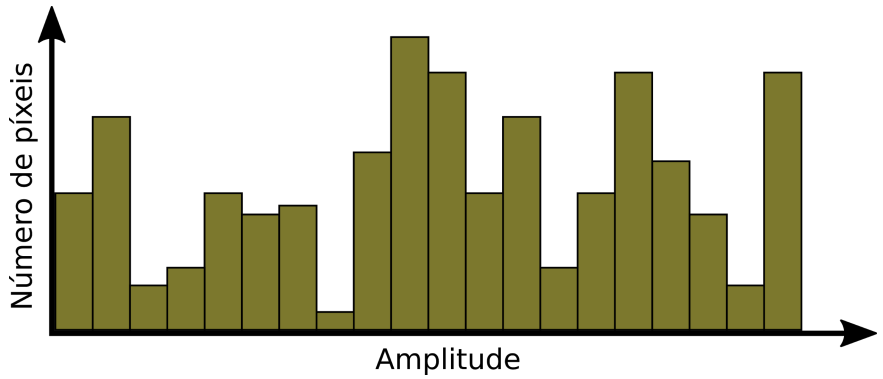
O código falha (produz um valor diferente de zero para *error count*) se compilado com alguns compiladores OpenACC. Qual poderá ser a causa? Como pode o erro ser prevenido?

5. A equação de Laplace ($\nabla^2 T = 0$) pode ser usada para descrever a variação de temperatura numa placa metálica. Na prática, esta equação calcula o valor de temperatura de um dado ponto como sendo a média aritmética dos valores de temperatura da vizinhança:

$$T(i,j) = [T_{old}(i+1, j+1) + T_{old}(i-1, j-1) + T_{old}(i-1, j+1) + T_{old}(i+1, j-1)]/4.$$

Neste exercício, a placa metálica é representada por uma grelha 2D e o estado inicial pode ser aleatoriamente gerado (valores entre 0 e 100 graus). Implemente um programa paralelo em OpenACC que seja capaz de iterar sobre a placa metálica, sendo a condição de paragem $T(i,j) - T_{old}(i,j) < 0.05$.

6. O histograma de uma imagem mostra a distribuição da intensidade dos píxeis nessa imagem. Na figura seguinte está ilustrado um histograma, com a amplitude no eixo horizontal e o número de píxeis, em função da amplitude, no eixo vertical.



Implemente um programa em OpenACC que obtenha o histograma da distribuição das intensidades de uma imagem com N píxeis. Implemente também a versão sequencial que permita comparar *throughput performance* com a correspondente versão paralela em OpenACC. Para representar a imagem, gere uma matriz de números aleatórios (intensidade dos píxeis entre 0 e 255).

7. Considere o seguinte programa que executa o método de Jacobi. Compile o código e corra-o usando `./jacobi 10 10 200`:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <stdlib.h>

int main(int argc, char** argv){
    int n, m, iter_max;
    if(argc > 1){
        n = atoi(argv[1]);
    } else {
        n = 4096;
    }

    if(argc > 2){
        m = atoi(argv[2]);
    } else {
        m = 4096;
    }
```

```
    if(argc > 3){
        iter_max = atoi(argv[3]);
    } else {
        iter_max = 1000;
    }

    const double tol = 1.0e-6;
    double error = 1.0;
    double err;

    double *restrict A    = (double*)malloc(sizeof(double)*n*m);
    double *restrict Anew = (double*)malloc(sizeof(double)*n*m);

    for(int i = 0; i < m; i++){
        A[i] = 1.0;
        Anew[i] = 1.0;
    }

    printf("Jacobi relaxation Calculation: %d x %d mesh\n", n, m);

    double st = omp_get_wtime();
    int iter = 0;

    #pragma acc data copy( A[:m*n]) copyin(Anew[:m*n] )
    {
        while ( error > tol && iter < iter_max ){
            err = 0.0;
            #pragma acc parallel loop
            for( int j = 1; j < n-1; j++){
                #pragma acc loop
                for( int i = 1; i < m-1; i++ ){
                    Anew[(j*m)+i] = 0.25 * ( A[(j*m)+i+1] +
A[(j*m)+i-1]+ A[((j-1)*m)+i] + A[((j+1)*m)+i]);
                    err = fmax( error, fabs(Anew[(j*m)+i] -
A[(j*m)+i]));
                }
            }

            #pragma acc parallel loop
            for( int j = 1; j < n-1; j++){
                #pragma acc loop
                for( int i = 1; i < m-1; i++ ){
                    A[(j*m)+i] = Anew[(j*m)+i];
                }
```

```
            }

            if(iter % 100 == 0){
                printf("%5d, %0.6f\n", iter, err);
                for( int j = 0; j < m; j++ ){
                    for( int i = 0; i < n; i++ ){
                        printf("%0.2f ", A[i+j*m]);
                    }
                    printf("\n");
                }
            }
            iter++;
        }
    }

    double runtime = omp_get_wtime() - st;
    printf(" total: %f s\n", runtime);
    free(A);
    free(Anew);
    return 0;
}
```

---

a) Comente o *output* gerado pelo programa.
b) Considere as diretivas `update self` e `update device`. Quel é a diferença entre estas diretivas?
c) Utilize uma dessas diretivas para corrigir o *output* do programa.