**Exercícios de Revisão**

Prof. Daniel Weingaertner

Prof. Armando L.N. Delgado

***Além dos exercícios abaixo, faça também os exercícios existentes nas notas de aula.***

# Questão 1

Considere uma função para calcular *d*=*A*×*B*×*s* , onde {*s,d*}∈ℝ*N*  são dois vetores de tamanho *N* e {*A,B*}∈{ℝ*N*×ℝ*N* } duas matrizes de tamanho *N*×*N*. Considere ainda que esta função é executada muitas vezes, e que todas estruturas cabem na cache do processador. Responda:

1. Supondo que a ordem das operações não seja relevante neste caso, qual a forma mais eficiente de computar o valor de *d*? Justifique sua resposta.

i. *d* = (*A*×*B*)×*s*

ii. *d* = *A*×(*B*×*s*)

1. Escreva o código que efetue o cálculo de *d*  de acordo com sua opção no item anterior.

**void** updCell(double \*s, double \*A, double \*B, double \*d, long SIZE)

{

double \*aux;

// aloca estrutura aux, seja ela qual for (não precisa alocar)

...

// inicia os cálculos

}

# Questão 2

Observe o código abaixo que calcula a seguinte integral pelo método de Monte Carlo:

*b*

∬*f* (*x , y*)*dx dy , onde f* (*x , y*) = 105*x*2+ *y*2−(*x*2+*y*2)2+10−5(*x*2+*y*2)4

*a*

|  |
| --- |
| **double** calc\_integral\_mc(**int** n, **double** a, **double** b){  **double** sum, x, y;  **for**(**int** i=0; i < n; i++) {   1. = a + (**double**) rand() \* ( (**double)** 1.0 / (RAND\_MAX \* (b – a)) ); 2. = a + (**double**) rand() \* ( (**double)** 1.0 / (RAND\_MAX \* (b – a)) );   sum += 1e5 \* pow(x, 2) + pow(y, 2) – pow( pow(x,2) + pow(y,2), 2 )  + 1e⁻5 \* pow(pow(x,2.0) + pow(y,2), 4) }  **return** (b - a)\*(b – a) \* sum / n; } |

Otimize este código o máximo possível. Destaque **cada** melhoria implementada que aumente a eficiência do seu código, **justificando-a!** A eficiência do código resultante é o principal critério de avaliação. A corretude é atributo indispensável*.*

# Questão 3

Sejam um conjunto de pontos (*xi, yi*)∈ℝ², *xi*<*xi*+1*,* 1⩽*i*⩽*N* e *f* (*xi*)=*yi* representando uma função a ser utilizada por determinada aplicação. Você foi contratada(o) para implementar um programa que retorne/calcule o valor de *f* (*z*)*, z*∈ℝ para *x*2<*z*<*xN*−1 . Caso o ponto

(*z ,f* (*z*)) não esteja definido no conjunto, você deve interpolar a função através de um polinômio de grau 4 (quatro) utilizando os pontos *xi* mais próximos de *z* . Considerando que os pontos não são uniformemente espaçados, responda:

1. Quantos pontos são necessários para calcular um valor interpolado *f* (*z*) ?

São necessários 5 pontos para calcular um polinômio de grau 4.

1. Qual dos métodos de interpolação deve ser utilizado: Newton ou Newton-Gregory? **Justifique!**

Newton, já que o método de Newton-Gregory serve para métodos de diferençaas ordinárias (pontos uniformemente espaçados).

1. Qual o problema em se utilizar um único polinômio interpolador definido a partir de todos os pontos, quando o número de pontos é muito grande?

O polinômio resultante pode acabar com maior grau e mais distânte que o polinômio correto (Fenômeno de Runge)

1. Considerando uma implementação eficiente do programa definido no enunciado, qual será o maior custo computacional: acesso à memória ou uso de CPU? Justifique.

CPU, já que serão realizadas várias operações com pontos flutuantes (???)

# Questão 4

1. Por que o código abaixo é ineficiente? **Justifique!**

Pois a cada iteração executada pelo loop, será calculado um desvio de condição não necessário.

1. Reescreva-o de forma a sanar o problema.

...

/\* n é muito grande \*/ for(i=0; i<n; i++) {

if( x == A ) {

FuncaoA(i);

}

else if( x == B ) {

FuncaoB(i);

}

else {

FuncaoC(i);

}

}

# Questão 5

Para cada par de códigos abaixo, indique qual das versões de código é mais rápida e por que?

(a)

|  |  |
| --- | --- |
| **Versão A** | **Versão B** |
| **int** p[SIZE];  **for** (**long** x=0; x<NUM\_COL; ++x) { **for** (**long** y=0; y<NUM\_LIN; ++y) {  p[x+y\*NUM\_COL]++  }  } | **int** p[SIZE];  **for** (**long** y=0; y<NUM\_LIN; ++y) { **for** (**long** x=0; x<NUM\_COL; ++x) {  p[x+y\*NUM\_COL]++  }  } |

A **versão B**, já que ao deixar x como loop interno, será utilizado os valores de p[ ] vizinhos, aproveitando a linha de *cache.*

(b)

|  |  |
| --- | --- |
| **Versão A** | **Versão B** |
| ... for (i=0; i<N; ++i) {  if (p==1)  norm += fabs(x[i]);  else if (p==2)  norm += x[i] x[i];∗  else  norm += pow(x[i],p); } | ... if (p==1)  for (i=0; i<N; ++i) norm += fabs(x[i]); else if (p==2)  for (i=0; i<N; ++i) norm += x[i] x[i];∗ else  for (i=0; i<N; ++i)  norm += pow(x[i],p); |

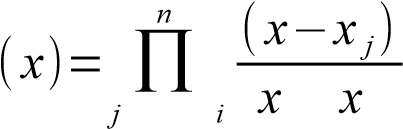
A **versão B**. Mesmo motivo da questão 4.

# Questão 6

Sejam um conjunto de pontos (*xi, yi*)∈ℝ², *xi*<*xi*+1*,* 1≤*i*≤*N* e *f* (*xi*)=*yi* uma função utilizada por determinada aplicação. Você foi contratada(o) para implementar um programa que retorne/calcule o valor de *f* (*z*)*,z*∈ℝ para *x*2≤*z*≤*xN*−1 . Caso o ponto (*z,f* (*z*)) não

esteja definido no conjunto, você deve interpolar a função através de um polinômio de grau 3 (três) utilizando os pontos *xi* mais próximos de *z* . Responda:

*n*

Lagrange: *pn*(*x*)=∑*i*=0 *Li*(*x*)*f* (*xi*) e *Li* =0, *j*≠ ( *i*− *j*)

Newton: *pn*(*x*)=*d*0+*d*1(*x*−*x*0)+*d* 2(*x*−*x*0)(*x*−*x*1)+⋯+*dn*(*x*−*x*0)⋯(*x*−*xn*−1) , onde *d k ,k*=0,1,…*,n* são as diferenças divididas de ordem *k* .

*d d d*

Newton-Gregory: *pn* *h* 2*h n!h* ,

onde *d k ,k*=0,1,…*,n* são as diferenças ordinárias de ordem *k* e *h* = *xi*+1−*xi ,i*=0,1,…*,n* .

1. Quantos pontos são necessários para calcular um valor interpolado *f* (*z*) ?

São necessários 4 pontos para calcular um polinômio de grau 3.

1. Qual dos métodos de interpolação pode ser utilizado: Newton, Laplace ou Newton-Gregory? **Justifique!**

Como o exercício não garante que os pontos sejam de diferenças ordinárias, não se usaria Newton-Gregory.

Como se pede para utilizar os pontos mais próximos e Newton utiliza somente os pontos anteriores ao a ser calculado, utilizaria Lagrange.

1. Qual das estruturas de dados abaixo seria mais eficiente para armazenar o conjunto de pontos caso você utilizasse o polinômio interpolador de Lagrange? E se você usasse Newton? **Justifique!**

|  |  |
| --- | --- |
| **Estrutura A** | **Estrutura B** |
| **struct** Ponto { **double** x,y;  } **struct** Ponto p[MAXPTOS]; | **struct** Pontos { **double** x[MAXPTOS];  **double** y[MAXPTOS];  } struct Pontos p; |

**Lagrange:** já que serão feitos vários cálculos utilizando somente os valores do vetor x, isso faz com que o acesso à memória seja mais efeciente na Estrutura B

**Newton:** já que serão feitos cálulos anterando entre os valores do vetor x e y, a estrutura A tem acesso à memória mais eficiente.

# Questão 7

1. Reescreva o código abaixo de forma a melhorar seu desempenho. Por que sua versão do código é mais eficiente? **Justifique!**

|  |
| --- |
| **double** a[n],x[n],y[n],b[n],z[n]; ...  **for** (i=0; i<n; i++){  a[i]=x[i]+y[i]\*sin((i%8)\*M\_PI);  } **for** (i=0; i<n; i++){ b[i]=1.0/x[i]+z[i];  } |

1. Considerando que a instrução “pragma unroll (8)” desenrola o laço 8 vezes, por que motivo o **Código A** tem um desempenho pior do que o **Código B**?

|  |  |
| --- | --- |
| **Código A** | **Código B** |
| 1. #pragma unroll (8) 2. for (i=0; i<n; i++) 3. { 4. a[i] = b[i]+c[i]\*d[i]; 5. e[i] = f[i]­g[i]\*h[i]+p[i]; 6. q[i] = r[i]+s[i]; 7. } | 1. #pragma unroll (8) 2. for (i=0; i<n; i++) 3. a[i] = b[i]+c[i]\*d[i]; 4. 3. #pragma unroll (8) 4. for (i=0; i<n; i++) 5. e[i] = f[i]­g[i]\*h[i]+p[i];8. 6. #pragma unroll (8) 7. for (i=0; i<n; i++) 8. q[i] = r[i]+s[i]; |

Pois o código A faz vários acessos a diferentes estruturas de dados. Ou seja, fazer os calculos de a[i], e[i] e q[i] em loops diferentes como no código B faz com que se aproveite a linha de *cache* para elementos vizinhos dos vetores (estes que são perdidos no código A).

# Questão 8

Analise o código apresentado abaixo.

1. **double** a[nd1][nd2], y[nd2], x[nd1]
2. ...
3. **for** (**long** i=0; i < nd2; ++i)
4. {
5. **double** t = 0.0;
6. **for (long** j=0; j < nd1; ++j)
7. t = t + a[j][i]\*x[j];
8. y[i] = t;
9. }

(a) Descreva dois **problemas** na implementação do código acima que o tornam ineficiente

A criação da variável t no loop de i, que poderia ser só zerado em vez de recriado a cada iteração, e o acesso à memória da matriz a, que calcula primeiro a coluna que a linha, não aproveitando a linha de *cache* e/ou a sequência de memória em que cada linha foi criada.

(b) Reescreva o código de forma a sanar/diminuir estas deficiências:

(c) Expliquepor que sua versão melhora cada um dos problemas apresentados no item (a)

Ta em a.

# Questão 9

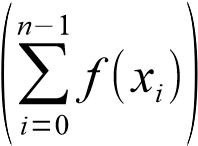
Seja uma função *f* :ℝ2→ℝ . Escreva um programa em linguagem C que calcule a integral

*b*

∬*f* (*x, y*)*dxdy* utilizando o Método dos Retângulos. O número de pontos *n* inicial (para

*a*

cada dimensão) é dado, e o espaçamento entre os pontos *h* é igual em ambas dimensões e dado por *h*=*b*−*a*/*n* ⇒ {*xi , yi*}=*a*+*hi* . O programa deve executar diversas iterações, reduzindo o valor do intervalo ao meio a cada iteração, até que o Erro Aproximado Absoluto da integral seja menor do que ϵ dado.

 *a a*

Método dos Retângulos: *b f* (*x*)*dx*≈∫*b p*0(*x*)*dx*=*h*

∫

**double** f (**double** x, **double** y); ... **double** integral (**double** a, **double** b, **double** epsilon, **uint** n)

{

}

(a) O Método dos Retângulos é apropriado para calcular a integral de funções de alta dimensionalidade? Justifique.

Não, já que a área de um retângulo pode acabar se diferenciando demais da área da função, dependendo do intervalo entre os pontos e o grau da integral.

# Questão 10

Observe o código para o método de Jacobi em duas dimensões apresentado abaixo.

|  |
| --- |
| **double** phi[iMAX][jMAX][2]; **int** t0=0, t1=1; ...  **for** (**long** it=1; it<ITER; ++it) {  **for** (**long** i=0; i < iMAX; ++i) {  **for (long** j=0; j < jMAX; ++j) {  phi[i][j][t1] = 1.0/4.0 \* (phi[i­1][j][t0] + phi[i+1][j][t0] +  phi[i][j­1][t0] + phi[i][j+1][t0]);  }  }  aux = t0; t0 = t1; t1 = aux; /\* troca os vetores \*/ } |

Reimplemente este código utilizando a técnica de “loop blocking” e **Justifique** por que o “loop blocking” torna este código mais eficiente!

# Questão 11

A versão A do código abaixo demora o dobro do tempo para executar do que a versão B. Por que isso ocorre?

|  |  |
| --- | --- |
| **Versão A** | **Versão B** |
| **struct** DATA { int a, b, c, d;  };  DATA p[N];  **for** (long i=0; i<N; ++i) {  p[i].a = p[i].b  } | **struct** DATA { int a, b;  };  DATA p[N];  **for** (long i=0; i<N; ++i) {  p[i].a = p[i].b  } |

Por causa da forma em que a struct DATA foi alocada em memória. Como na Versão B os valores de a e b são mais próximos, estes são melhor aproveitados pela linha de *cache* e acessso a memória.

# Questão 12

|  |
| --- |
| for (int i=0; i<N; ++i) for (int j=0; j<N; ++j)  c[i] = c[i] + A[i][j] \* b[j] |

Considere o código acima:

1. Reimplemente este código aplicando apropriadamente a técnica de “loop unroll” com tamanho quatro.
2. O código com o laço desenrolado é mais eficiente que o código original em umaarquitetura x64? Justifique sua resposta.

# Questão 13

Responda às seguintes questões:

1. Qual o problema de se utilizar muitos pontos para calcular o polinômio interpolador de uma função tabulada? Como proceder para calcular um valor interpolado a partir de um grande conjunto de pontos?

Fenômeno de Runge. Fazer com que em intervalos diferentes se tenha poliômios de menor grau (????)

1. Porque o acesso em coluna é ineficiente para matrizes bidimensionais em linguagem C?

Pois matrizes são alocadas linha a linha, fazendo com que o acesso em coluna faça “saltos” na memória para usar cada elemtento da matriz, enquanto o acesso em linha vá para os valores vizinhos.

1. Por que a integração numérica pelo método dos trapézios não é uma boa solução para problemas de alta dimensionalidade?

9.a

# Questão 14

Dado um conjunto de **n** pontos **(xi , yi )**, escreva um procedimento para calcular a tabela completa de diferenças divididas de Newton. Faça também um procedimento que recebe o valor do grau **p** de um polinômio interpolador entre dois pontos **(xk , yk )** e **(xj , yj ) , p < n**, e calcula os coeficientes do polinômio interpolador de Newton de grau **p**, usando a tabela de diferenças divididas gerada pelo primeiro procedimento. Finalmente faça um procedimento que calcule o valor do polinômio interpolador para um valor qualquer.