CI1164 – Introdução à Computação Científica

Prof. Guilherme Derenievicz
Prof. Armando Delgado

Exercícios de Revisão para Prova 02

GABARITO

Questão 1

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

(a) 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. $d = (A \times B) \times s$
   ii. $d = A \times (B \times s)$

A forma mais eficiente de computar d é a versão **(ii)**. Isto porque inicialmente é efetuada uma multiplicação de matriz por vetor, com ordem $O(N^2)$, resultando em um vetor, e então segue nova multiplicação de matriz por vetor, totalizando $O(N^2)+O(N^2)$. Na outra alternativa há uma multiplicação de matriz por matriz, com custo $O(N^3)$, seguida de uma multiplicação de matriz por vetor, de ordem $O(N^2)$, totalizando $O(N^3)+O(N^2)$.

Além de realizar menos operações, a opção **ii)** faz todos os acessos aos elementos das matrizes em linha, aproveitando melhor a linha de cache, enquanto que na opção **i)** o acesso à matriz B é feito em coluna.

(b) 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


}
```

```
// Opção A
void updCell(double *s, double *A, double *B, double *d, long SIZE)
{
```

```
  double *aux;
  // aloca estrutura aux, seja ela qual for
  ...
  for (long i=0; i<SIZE; ++i)
    for (long j=0; j<SIZE; ++j) {
      aux[i*SIZE+j] = 0.0;
      for (long k=0; k<SIZE; ++k)
        aux[i*SIZE+j] += A[i*SIZE+k] * B[k*SIZE+j];
    }
  for (long i=0; i<SIZE; ++i) {
    d[i] = 0.0;
    for (long j=0; j<SIZE; ++j)
      d[i] += aux[i*SIZE+j] * s[j];
  }
}
```

```
// Opção B
void updCell(double *s, double *A, double *B, double *d, long SIZE)
{
  double *aux;
  // aloca estrutura aux, seja ela qual for
  ...
  for (long i=0; i<SIZE; ++i) {
    aux[i] = 0.0;
    for (long j=0; j<SIZE; ++j)
      aux[i] += B[i*SIZE+j] * s[j];
  }

  for (long i=0; i<SIZE; ++i) {
    d[i] = 0.0;
    for (long j=0; j<SIZE; ++j)
      d[i] += A[i*SIZE+j] * aux[j];
  }
}
```

Questão 2

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

$$\int_a^b\int f(x,y)\,dx\,dy, \quad onde \quad f(x,y) = 10^5 x^2 + y^2 - (x^2+y^2)^2 + 10^{-5}(x^2+y^2)^4$$

```
double calc_integral_mc(int n, double a, double b){
   double sum, x, y;
   for(int i=0; i < n; i++) {
      x = a + (double) rand() * ( (double) 1.0 / (RAND_MAX * (b – a)) );
      y = 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 as decisões de implementação que aumentam a eficiência do seu código, **justificando-as** (i.e. você deve explicar por que sua alternativa aumenta o desempenho). A eficiência do código é o principal critério de avaliação. A corretude é atributo indispensável.

```
// OPÇÃO A (simples)

double calc_integral_mc(int n, double a, double b){
   double sum=0, x, y, xsq_ysq;
   double intervalo = (1.0 / RAND_MAX) * (b – a);
   for(int i=0; i < n; i++) {
      x = a + (double) rand() * intervalo;
      y = a + (double) rand() * intervalo;
      x = x * x;
      y = y * y;
      xsq_ysq = (x + y) * (x + y); // (x² + y²)²
      sum += 1e5 * x + y – xsq_ysq + 1e-5 * xsq_ysq * xsq_ysq;
   }
   return (b - a)*(b – a) * sum / n;
}
```

```
// OPÇÃO B (considerando pipeline e SIMD)

#define UNRL ...

double calc_integral_mc(int n, double a, double b){
   double sum[UNRL]={0,0,...,0}, x[UNRL], y[UNRL], xsq_ysq[UNRL];

   double intervalo = (1.0 / RAND_MAX) * (b – a);

   int i, unroll = n – n % UNRL;

   for(i=0; i < unroll; i+=UNRL) {
      x[0] = a + (double) rand() * intervalo;
      y[0] = a + (double) rand() * intervalo;
      ...
      x[UNRL-1] = a + (double) rand() * intervalo;
      y[UNRL-1] = a + (double) rand() * intervalo;

      x[0] *= x[0]; ... x[UNRL-1] *= x[UNRL-1];
      y[0] *= y[0]; ... y[UNRL-1] *= y[UNRL-1];

      xsq_ysq[0] = (x[0] + y[0]) * (x[0] + y[0]); // (x² + y²)²
      ...
      xsq_ysq[UNRL-1] = (x[UNRL-1] + y[UNRL-1]) * (x[UNRL-1] + y[UNRL-1]);

      sum[0] += 1e5 * x[0] + y[0] – xsq_ysq[0] +
                1e-5 * xsq_ysq[0] * xsq_ysq[0];
      ...
      sum[UNRL-1] += 1e5 * x[UNRL-1] + y[UNRL-1] – xsq_ysq[UNRL-1] +
                     1e-5 * xsq_ysq[UNRL-1] * xsq_ysq[UNRL-1];
   }

   // Resíduo
    double sum = 0.0, xi, yi, xsq_ysqi;
    for(  ; i < n; i++) {
      xi = a + (double) rand() * intervalo;
```

```
      yi = a + (double) rand() * intervalo;
      xi *= xi;   yi *= yi;

      xsq_ysqi = (xi + yi) * (xi + yi); // (x² + y²)²
      sum += 1e5 * xi + yi – xsq_ysqi + 1e-5 * xsq_ysqi * xsq_ysqi;
   }

   return (b - a)*(b – a) * ( sum + sum[0] + ... + sum[UNRL-1] ) / n;
}
```

Justificativa:

- Cálculo do intervalo do número aleatório fora do laço, sendo feito apenas uma vez.
- Eliminação de chamadas a função 'pow()', que aumentam tempo de execução.
- Desenrolar o laço para aumentar a quantidade de operações aritméticas disponíveis melhorando o preenchimento do pipeline
- Permite SIMD

Questão 3

Sejam um conjunto de pontos $(x_i, y_i) \in \mathbb{R}^2$, $x_i < x_{i+1}$, $1 \leqslant i \leqslant N$ e $f(x_i) = y_i$ representando uma função a ser utilizada por determinada aplicação. Você foi contratada(o) para implementar um programa que retorne/calcule o valor de $f(z)$, $z \in \mathbb{R}$ para $x_2 < z < x_{N-1}$. 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 x_i mais próximos de z. Considerando que os pontos não são uniformemente espaçados, responda:

a) Quantos pontos são necessários para calcular um valor interpolado $f(z)$?

5 pontos

b) Qual dos métodos de interpolação pode ser utilizado: Newton ou Newton-Gregory? **Justifique!**

Como o intervalo entre os pontos não é uniforme, não podemos utilizar Newton-Gregory, que depende de intervalos iguais entre todos os pontos.

c) 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?

Em um conjunto muito grande de pontos, a interpolação polinomial por Newton ou Lagrange apresenta distorções nos extremos do intervalo (fenômeno de Runge). A solução seria produzir interpolar por partes em pequenos intervalos usando Splines.

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

O principal custo computacional será de acesso à memória pois é necessário localizar o valor mais próximo de z no vetor de pontos (busca binária) e então fornecer o valor de $f(z)$ correspondente (caso exista) ou calculá-lo utilizando os 5 pontos vizinhos.

Questão 4

a) Por que o código abaixo é ineficiente? **Justifique!**

Os desvios condicionais impedem que o pipeline do processador seja devidamente preenchido por não ser possível saber qual a próxima instrução a ser executada. Desta forma o processador precisa esperar a conclusão da comparação para iniciar a chamada de função apropriada. Um “branch misprediction” custa muito caro. O custo do ‘if’ em si é desprezível.

b) 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);
   }
}
```

```
if( x == A ) {
   for(i=0; i<n; i++)  {
      FuncaoA(i);
   }
}
else if( x == B ) {
   for(i=0; i<n; i++)  {
       FuncaoB(i);
   }
else {
   for(i=0; i<n; i++)  {
       FuncaoC(i);
   }
}
```

Questão 5

Qual das versões de código abaixo é mais rápida e por que? **Justifique**!

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 é mais rápida porque faz o acesso da memória em linha, ou seja, os valores são acessados na ordem em que estão na memória física, aproveitando ao máximo a banda de memória e os dados da cache, enquanto que na versão A o acesso é intercalado (strided), causando uma grande quantidade de “cache misses”.	

Questão 6

Sejam um conjunto de pontos $(x_i, y_i) \in \mathbb{R}^2$, $x_i < x_{i+1}$, $1 \leq i \leq N$ e $f(x_i) = y_i$ uma função utilizada por determinada aplicação. Você foi contratada(o) para implementar um programa que retorne/calcule o valor de $f(z), z \in \mathbb{R}$ para $x_2 \leq z \leq x_{N-1}$. 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 x_i mais próximos de z. Responda:

a) Quantos pontos são necessários para calcular um valor interpolado $f(z)$?

4 pontos

b) Qual dos método de interpolação pode ser utilizado: Newton ou Newton-Gregory? **Justifique!**

Como o intervalo entre os pontos não é definido, não podemos utilizar Newton-Gregory, que depende de intervalos iguais entre todos os pontos.

c) Caso você utilizasse o polinômio interpolador de Lagrange ao invés de Newton, qual das estruturas de dados abaixo seria mais eficiente para armazenar o conjunto de pontos? **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;`

A estrutura B seria mais indicada pois no cálculo do polinômio interpolador de Lagrange as coordenadas x são utilizadas no laço mais interno enquanto que as coordenadas y apenas no laço externo. Utilizando a estrutura B teríamos um acesso contínuo de memória no laço interno, aumentando o aproveitamento da linha de cache e diminuindo o fluxo de dados.	

Questão 7

Seja uma função $f: \mathbb{R}^2 \rightarrow \mathbb{R}$. Escreva um programa em linguagem C que calcule a integral $\int_a^b \int f(x, y)\, dx\, dy$ utilizando o Método de Monte Carlo e a função `f(x,y)` declarada na primeira linha do código. O número de pontos n a ser inicialmente amostrado é dado. O programa deve executar diversas iterações, dobrando o número de pontos amostrados a cada iteração, até que a diferença entre a integral calculada entre duas iterações consecutivas seja menor do que ϵ (epsilon).

Integral por Monte Carlo: $$\int_a^b \int f(x, y)\, dx\, dy \approx \frac{(b-a)^2}{n} \left(\sum_{i=0}^{n-1} f(x_i, y_i) \right)$$

```
double f (double x, double y);
...
double integral (double a, double b, double epsilon, uint n)
{
   double soma, x, y, it=0.0, ita;
   double intervalo = (1.0 / RAND_MAX ) * (b – a);

   do {
      soma=0.0;
      ita = it;
      for(int i=0; i < n; i++) {
         x = a + (double) rand() * intervalo;
         y = a + (double) rand() * intervalo;
         soma += f(x,y);
      }

      it = (b - a)*(b – a) * soma / n;

      erro = fabs(it – ita);

      n *= 2;

   } while (erro > epsilon);

   return it;
}
```

Questão 8

Otimize o código abaixo o máximo possível. Destaque as decisões de implementação que aumentam a eficiência do seu código, **justificando-as** (i.e. você deve explicar por que sua alternativa aumenta o desempenho). A eficiência do código é o principal critério de avaliação. A corretude é atributo *sine qua non.*

```
int n;
double A, B, C, D, E;
double F[n*n], double G[n*n], double R[n*n];

for(int h = 0; h < pow(n, 2.0); h++) {
    R[h] = G[h] – A * F[h];
    if(h-1 >= 0)
      R[h] -= B * F[h - 1];
    if(h+1 < pow(n, 2.0) - 1)
      R[h] -= C * F[h + 1];
    if(h - n >= 0)
      R[h] -= D * F[h - n];
    if(h + n < pow(n, 2.0) – 1)
      R[h] -= E * F[h + n];
}
```

```
// * Eliminar computação repetida de pow(n,2)
// * Retirar if's de repetições para melhorar pipeline
// * Possibilidade de 'unroll' em cada repetição

// SOLUÇÃO 1
int n, h, lim;
```

```
double A, B, C, D, E;
double F[n*n], double G[n*n], double R[n*n];

lim = n*n;

for(h = 0; h < lim; h++) {
    R[h] = G[h] – A * F[h];
}
for(h = 1; h < lim; h++) {
    R[h] -= B * F[h – 1];
}
for(h = n; h < lim; h++) {
    R[h] -= D * F[h – n] ;
}
lim -= 2;
for(h = 0; h < lim; h++) {
    R[h] -= C * F[h + 1] ;
}
lim -= n + 1;
for(h = 0; h < lim; h++) {
    R[h] -= E * F[h + 1] ;
}


// SOLUÇÃO 2
int n, h, lim;
double A, B, C, D, E;
double F[n*n], double G[n*n], double R[n*n];

lim = n*n;

R[0] = G[0] – A * F[0] - C * F[1] - E * F[n];

for(h = 1; h < n; h++) {
    R[h] = G[h] – A * F[h] - B * F[h – 1] - C * F[h + 1] - E * F[h + n];
}
for(; h < lim – n - 1; h++) {
    R[h] = G[h] – A * F[h] - B * F[h – 1] - C * F[h + 1] - D * F[h – n] - E * F[h + n];
}
for(; h < lim - 2; h++) {
    R[h] = G[h] – A * F[h] - B * F[h – 1] - C * F[h + 1] - D * F[h – n] ;
}
R[lim-2] = G[lim-2] – A * F[lim-2] - B * F[lim – 3] - D * F[lim – n - 3];
R[lim-1] = G[lim-1] – A * F[lim-1] - B * F[lim – 2] - D * F[lim – n - 1];
```

Questão 9

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

```
double phi[iMAX+1][jMAX+1][2];
int t0=0, t1=1, aux;
...
for (long it=1; it<ITER; ++it) {
  for (long j=1; j < jMAX; ++j)
    for (long i=1; i < iMAX; ++i) {
       phi[i][j][t1] = 0.25 * (phi[i-1][j][t0] + phi[i+1][j][t0] +
                          phi[i][j-1][t0] + phi[i][j+1][t0]);
    }
   aux = t0; t0 = t1; t1 = aux; /* troca os vetores */
}
```

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

(1) Ordem dos dados faz com que TODO acesso seja *"strided"*.

(2) Pouco aproveitamento de valores de linhas subsequentes que já foram carregados para a cache

(3) O método acessa elementos vizinhos em duas dimensões. Assim, é necessário manter ao menos 3 linhas da matriz em cache a todo momento. O acesso sequencial conforme implementado faz com que os valores do início de uma linha sejam descartados antes que possam ser reaproveitados na linha seguinte.

(b) Reescreva o código de forma a torná-lo o mais eficiente possível

```
// PASSO INICIAL

double phi[2][iMAX+1][jMAX+1]; /* MAIS importante!! */
int t0=0, t1=1;
...
for (long it=1; it<ITER; ++it) {

  for (long i=1; i < iMAX; ++i)
    for (long j=1; j < jMAX; ++j) {
       phi[t1][i][j] = 0.25 * (phi[t0][i-1][j] + phi[t0][i+1][j] +
                          phi[t0][i][j-1] + phi[t0][i][j+1]);
    }
  aux = t0; t0 = t1; t1 = aux; /* troca os vetores */
}

-----------------------------------------------------------------------------------------------------------------
-

// PASSO ADICIONAL

double phi[2][iMAX+1][jMAX+1]; /* MAIS importante!! */
int t0=0, t1=1;
...
for (long it=1; it<ITER; ++it) {
```

```
    /* fazer blocking reaproveita linhas subsequentes */
    /* unroll aumenta a quantidade de operações no pipeline */
    for (long bi=0; bi < iMAX/BiSIZE + 1; ++bi) {
      long ibegin = bi*BiSIZE; iend = min(ibegin + BiSIZE, iMAX);
      for (long bj=0; bj < jMAX/BjSIZE + 1; ++bj) {
         long jbegin = bj*BjSIZE; jend = min(jbegin+BjSIZE, jMAX);
         for (long i=ibegin; i < iend; ++i) {
            for (long j=jbegin; j < jend; ++j) {
               phi[t1][i][j] = 1.0/4.0 * (phi[t0][i-1][j] +
                        phi[t0][i+1][j] + phi[t0][i][j-1] +
                        phi[t0][i][j+1]);
      }
   }

   aux = t0; t0 = t1; t1 = aux; /* troca os vetores */
}
```

(c) **Explique** por que sua versão melhora cada um dos problemas apresentados

(1) Mudança na ordem para evitar acesso *"strided"* que carrega um valor desnecessário a cada acesso

(2) *Blocking* para aproveitar dados das linhas anterior e posterior que ainda estão na cache

(3) O acesso em blocos permite que os valores em cache sejam reaproveitados antes de serem descartados, diminuindo a quantidade de *cache miss* e especialmente o tráfego de memória, já que os valores são carregados apenas uma vez.

Questão 10

Sejam um conjunto de pontos $P=(x_i, y_i)\in\mathbb{R}^2$, $x_i<x_{i+1}$, $1\le i\le N$ e $f(x_i)=y_i$ uma função utilizada por determinada aplicação. Você foi contratada(o) para implementar um programa que retorne/calcule o valor de $f(z), z\in\mathbb{R}$ para $x_2\le z\le x_{N-1}$. Caso o ponto $(z, f(z))$ não esteja definido no conjunto P, você deve interpolar a função através de um polinômio de grau três utilizando os pontos $x_i, x_{i+1}<z<x_{i+2}, x_{i+3}$ mais próximos de z.

Lagrange: $p_n(x)=\sum_{i=0}^{n} L_i(x) f(x_i)$ e $L_i(x)=\prod_{j=0,\, j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}$

Newton: $p_n(x)=d_0+d_1(x-x_0)+d_2(x-x_0)(x-x_1)+\cdots+d_n(x-x_0)\cdots(x-x_{n-1})$, onde $d_k, k=0, 1, \ldots, n$ são as diferenças divididas de ordem k.

a) Qual o método mais eficiente para o seu programa: Newton ou Lagrange? **Justifique!**

Depende da quantidade de vezes que o programa será utilizado para uma mesma função f. O Método de Newton precisa construir a tabela de diferenças divididas e armazená-la para ter uma eficiência maior do que o Método de Lagrange.

No caso de um polinômio de grau 3, Lagrange terá 3 termos em cada produtório, perfazendo $(2\,\text{sum}+1\,\text{div})*3+2\,\text{mult}=11$ FLOP por produtório. E quatro termos no somatório,

totalizando $(11+1\,\text{mult})*4+3\,\text{sum}=51$ FLOP para cada valor de z, Newton efetuará 15 FLOP para cada valor de z, desde que a tabela de diferenças divididas esteja calculada. Entretanto, o cálculo desta consome 30 FLOP se for utilizada apenas uma vez para cada conjunto de 4 pontos interpolados.

b) Considerando os polinômios interpoladores de Lagrange e Newton, qual das estruturas de dados abaixo seria mais eficiente para armazenar o conjunto de pontos em cada caso? **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: A Estrutura B que é um "Struct of Arrays" é mais eficiente porque os valores de x são utilizados no laço mais interno (produtório) enquanto que os valores de y são usados apenas no laço externo (somatório). Além disso, os valores de x utilizados são sempre vizinhos e esta estrutura favorece a localidade espacial.

Newton: A Estrutura B também é mais eficiente para o cálculo do polinômio de Newton, porque só utiliza valores de x consecutivos, desde que a tabela de diferenças divididas já esteja calculada. O cálculo das diferenças divididas, por outro lado, utiliza tanto valores de x quanto de y, e o desempenho de ambas estruturas deve ser similar.

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

O maior custo será no uso de CPU pois o acesso à memória ocorre somente a 4 pontos consecutivos. Entretanto, a busca pelo ponto intermediário z pode implicar em um acesso considerável de memória.

Questão 11

a) Reescreva o código abaixo de forma a melhorar seu desempenho

```
...
double a[n][n],x[n],y[n],b[n],z[n];
...
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    a[j][i] = x[i] + y[j] * cos((i%8)*M_PI/7.0);
for (i=0; i<n; i++)
  b[i]=1.0/x[i]+z[i];
```

```
double a[n][n],x[n],y[n],b[n],z[n];
```

```
double tabCos[8];
...
for (i=0; i<8; i++)
  tCos[i] = cos(i*M_PI/7.0);

for (j=0; j<n; ++j){
  for (i=0; i<n-(n%4); i+=4){
    int icos=i%8;
    a[j][i]  =x[i]+y[j]*tCos[icos];
    a[j][i+1]=x[i+1]+y[j]*tCos[icos+1];
    a[j][i+2]=x[i+2]+y[j]*tCos[icos+2];
    a[j][i+3]=x[i+3]+y[j]*tCos[icos+3];
  }
  for (; i<n; i++)
    a[j][i]=x[i]+y[j]*tCos[i%8];
}
for (i=0; i<n-(n%4); i+=4){
  b[i]  =1.0/x[i]+z[i];
  b[i+1]=1.0/x[i+1]+z[i+1];
  b[i+2]=1.0/x[i+2]+z[i+2];
  b[i+3]=1.0/x[i+3]+z[i+3];
}
for (; i<n; i++)
  b[i]=1.0/x[i]+z[i];
```

Por que sua versão do código é mais eficiente? **Justifique!**

Acesso à matriz a é feito em colunas e pode ser invertido, permitindo acesso a endereços contínuos em memória.

O coseno é um cálculo custoso e pode ser feito antecipadamente uma vez que há apenas um pequeno intervalo de valores para os quais será calculado. Usando uma "lookup table" pode-se evitar o cálculo no interior do laço.

Unroll em i permite o uso de SIMD.

Unroll em j diminui o acesso ao vetor x.

Os laços podem ser fundidos (loop fusion) desde que haja um laço separado para o primeiro índice j, de forma a diminuir a quantidade de tráfego de memória pois o vetor x será carregado apenas uma vez.

b) 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

```
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. }
```

Código B

```
1. #pragma unroll (8)
2. for (i=0; i<n; i++)
3.    a[i] = b[i]+c[i]*d[i];
4.
5. #pragma unroll (8)
6. for (i=0; i<n; i++)
7.    e[i] = f[i]-g[i]*h[i]+p[i];
```

	8. 9. #pragma unroll (8) 10. for (i=0; i<n; i++) 11. q[i] = r[i]+s[i];

Justifique!

O Código A utiliza muitos registradores, mais do que há disponível no processador, de forma que ocorre o “register spill”, e os valores dos registradores precisam ser devolvidos à cache e novamente carregados na próxima iteração.

Como há muitos vetores a serem carregados simultaneamente, eles disputam o mesmo espaço de cache.

O corpo do laço fica muito grande no código A, causando faltas na cache de instrução.

Como não há reaproveitamento de memória, o melhor é tratar os laços separadamente

Questão 12

Seja uma função $f:\mathbb{R}^2 \rightarrow \mathbb{R}$. Escreva um programa em linguagem C que calcule a integral $\int_a^b\int f(x,y)\,dx\,dy$ utilizando o Método dos Retângulos. O número de pontos n inicial é dado, e o espaçamento entre os pontos h é igual em ambas dimensões e dado por $h=b-a/n \Rightarrow \{x_i, y_i\}=a+h*i$. 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.

Método dos Retângulos: $\int_b^a f(x)dx \approx \int_b^a p_0(x)dx = h\left(\sum_{i=0}^{n-1} f(x_i)\right)$

```
double f (double x, double y);
...
double integral (double a, double b, double epsilon, uint n)
{
  double intAnterior, intAtual = DBL_MAX;
  double h;
  int xi, yi;
  do {
    intAnterior = intAtual;
    intAtual = 0.0;
    h = (b-a)/(double)n;
    for (xi=0; xi < n; xi++)
      for (yi=0; yi < n; yi++)
        intAtual += f(a+h*(double)xi, a+h*(double)yi);
    intAtual *= h*h;
    n = n * 2;
  } while (fabs(intAnt – intAtual) < epsilon);
  return intAtual;
}
```

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

O método dos retângulos tem seu custo aumentado exponencialmente com relação às dimensões da função e por isso não é apropriado para funções de alta dimensionalidade. Para estes casos pode-se utilizar métodos baseados em Monte Carlo.

Questão 13

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`
	`}`

Na Versão A a estrutura possui 4 números inteiros, enquanto que na versão B apenas 2. Apesar de apenas 2 inteiros serem utilizados, na versão A os 4 são carregados pois residem em uma mesma linha de cache. Assim, o volume de dados que deve trafegar entre a memória e o processador é o dobro em A quando comparado a B.	

Questão 14

Considere o código abaixo:

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

a) Reimplemente este código aplicando apropriadamente a técnica de “loop unroll” com tamanho quatro.

```
1.  for(i=0; i<N%4 ; ++i)
2.     for(j=0; j<N; ++j)
3.        c[i] = c[i] + A[i][j] * b[j];
4.
5.  for(i; i<N ; i+=4) {
6.     for(j=0; j<N; ++j) {
7.        c[i]   = c[i]   + A[i][j]   * b[j];
8.        c[i+1] = c[i+1] + A[i+1][j] * b[j];
9.        c[i+2] = c[i+2] + A[i+2][j] * b[j];
10.       c[i+3] = c[i+3] + A[i+3][j] * b[j];
11.    }
12.  }
```

b) O código com o laço desenrolado é mais eficiente que o código original em uma arquitetura x64? Justifique sua resposta.

Sim, pois na versão desenrolada o acesso ao vetor **b** é reduzido em quatro vezes, diminuindo o tráfego de memória;

Questão 15

Responda às seguintes questões:

a) 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?

O polinômio interpolador com grau muito alto terá muitas raízes e consequentemente oscilará muito entre os pontos dados (fenômeno de Runge). Para calcular um valor interpolado geralmente escolhe-se uma pequena quantidade de pontos no entorno do ponto de interesse, obtendo assim um polinômio de menor grau.

b) Explique que tipo de problemas com registradores podem ser causados por um “loop unroll”.

Register Spill ocorre quando não há registradores suficientes para armazenar as variáveis locais e elas precisam ser armazenadas na memória (cache). O *loop unroll* pode causar *register spill* pois aumenta a quantidade de variáveis locais na proporção do desenrolar do laço.

c) Porque o acesso em coluna é ineficiente para matrizes bidimensionais em linguagem C?

Porque na linguagem C as matrizes são organizadas em *row major order*, i.e., as linhas de uma matriz são contínuas na memória. O acesso em coluna faz com que as linhas de cache não sejam completamente aproveitadas, pois elas contém os elementos subsequentes de uma mesma linha da matriz.

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

Porque a dimensão do problema afeta o custo computacional do método dos trapézios de maneira exponencial. Assim, altas dimensões não são computacionalmente viáveis.

Questão 16

Considerando a seguinte implementação do Método de Lagrange para interpolação, responda as questões abaixo.

```
// Método de Lagrange para interpolação da função f(x) tabelada em n pontos.
// x: valor no qual f(x) deve ser aproximada
// n: quantidade de pontos
// tab: vetor de tamanho 2n contendo pares (xi, f(xi)): [x0, fx0, x1, fx1, ...]
double lagrange(double x, double *tab, int n) {
	double Px, Li, xi, xj, fi;

	Px = 0.0;
	for (i = 0; i < n; ++i)
	{
		Li = 1.0;
		xi = tab[2*i];
		for (int j = 0; j < n, ++j)
		{
			xj = tab[2*j];
			Li *= (x-xj)/(xi-xj);
		}
		fi = tab[2*i+1];
		Px += Li*fi;
	}
	return Px;
}
```

a) Identifique o que está incorreto no código acima e mostre como resolver, otimizando a sua solução.

j deve ser diferente de **i**. Ao invés de colocar o IF é melhor quebrar o FOR em 2.

b) Por que a estrutura tab não é boa para esse programa? Proponha uma alternativa melhor e justifique.

os valores $\mathbf{f_i}$ são usados apenas fora do laço interno. Melhor 2 vetores separados, maximizando a quantidade de $\mathbf{x_i}$ em cache.

c) Considerando o programa resultante dos itens (a) e (b) há a possibilidade do uso de AVX? Se sim, indique em qual trecho, justificando sua resposta. Se não, indique quais modificações podem ser feitas a fim de aproveitar os registradores AVX.

É preciso contornar a dependência de dados:

```
...
// aqui já considerando os vetores separadas tab_x[n] e tab_fx[n]
xi = tab_x[i]
double L_tmp[4] = {1.0, 1.0, 1.0, 1.0};
for (int j = 0; j < i-i%4, j+=4)
{
        for (int k = 0; k < 4; ++k)
            L_tmp[k] *= (x-tab_x[j+k])/(xi-tab_x[j+k]);
}
Li = L_tmp[0]*L_tmp[1]*L_tmp[2]*L_tmp[3];
for (int j = i-i%4, j < i; ++j)
        Li *= (x-tab_x[j])/(xi-tab_x[j]);
...

// repetir o mesmo para o for j=i+1..n
```

d) Sabendo que essa função nunca será usada com x pertencente à tabela de pontos (isto é, x != xi para todo i=0..n-1) modifique o programa resultante dos itens (a) e (b) a fim de diminuir a quantidade de operações de ponto flutuante executadas e responda: ao comparar as duas versões, há diferença no tempo de execução? e na taxa de MFLOP/s? Justifique sua resposta.

Ver slide 75. O tempo de execução será menor, pois menos trabalho está sendo feito, porém, a taxa de MFLOP/s não deve mudar, já que na versão original a cada iteração do FOR interno são usados **x** e $\mathbf{x_i}$ (que estão em CACHE) e $\mathbf{x_j}$ (buscado da memória) e aqui continuam sendo usado o $\mathbf{x_j}$. Assim, nenhuma otimização foi feita com relação à memória.

```
...
Num = 1.0;
for (int i = 0; i < n; ++i)
// aqui já considerando os vetores separados tab_x[n] e tab_fx[n]
Num *= (x-tab_x[i]);

Px = 0.0;
for (int i = 0; i < n; ++i)
{
        Den = 1.0;
        xi = tab_x[i];
        for (int j = 0; j < i, ++j)
                Den *= (xi-tab_x[j]);
```

```
        for (int j = i+1; j < n, ++j)
            Den *= (xi-tab_x[j]);

        Li = Num/Den;
        Px += Li*tab_fx[i];
    }
    ...
```

e) É possível otimizar o acesso à memória do programa resultante dos itens (a) e (b) com a técnica de Loop Unroll & Jam? Se sim, mostre como fica o código otimizado. Se não, justifique sua resposta.

```
…
// aqui já considerando os vetores separados tab_x[n] e tab_fx[n]
xi = tab_x[i]
double x_tmp[4], L_tmp[4] = {1.0, 1.0, 1.0, 1.0};
for (int j = 0; j < i-i%4, j+=4)
{
    x_tmp[0] = tab_x[j];
    x_tmp[1] = tab_x[j+1];
    x_tmp[2] = tab_x[j+2];
    x_tmp[3] = tab[j+3];
    L_tmp[0] *= (x-x_tmp[0])/(xi-x_tmp[0]);
    L_tmp[1] *= (x-x_tmp[1])/(xi-x_tmp[1]);
    L_tmp[2] *= (x-x_tmp[2])/(xi-x_tmp[2]);
    L_tmp[3] *= (x-x_tmp[3])/(xi-x_tmp[3]);
}
Li = L_tmp[0]*L_tmp[1]*L_tmp[2]*L_tmp[3];
for (int j = i-i%4, j < i; ++j)
{
    xj = tab_x[j];
    Li *= (x-xj)/(xi-xj);
}

...

// repetir o mesmo para o for j=i+1..n
```