Algoritmos Matriciais em Processamento de Alto Desempenho

Nicolas Maillard
nmaillard@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul

Escola Regional de Alto Desempenho, 2005

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Processamento de alto desempenho

- “Processamento” = cálculo,
  - A noção de cálculo tem a ver com o número de instruções executadas, *i.e.* com o tempo de execução.
- Cálculo se faz com dados.
  - A noção de dados implica no espaço na memória.
- “Desempenho” diz respeito a eficiência. É preciso de PAD para aplicações pesadas, em geral paralelas.
  - Freqüentemente se deve também ser eficiente em nível de rede.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Os níveis de atuação para a obtenção de Processamento de Alto Desempenho

O programador deve considerar pelo menos os pontos abaixo:

- otimizaçõ do hardware (processador/rede); **de Rose/Pilla — ERAD'05**
- adaptação do sistema operacional; **Rômulo de Oliveira/Carissimi — ERAD'02**
- uso de middlewares específicos (*e.g.* compiladores apropriados, bibliotecas para a programação paralela...); **Gerson Cavalheiro — ERAD'04**
- programação otimizada (*e.g.* uso de tipos de dados vetoriais);
- algoritmos com eficiência comprovada.
- programação otimizada (*e.g.* uso de tipos de dados vetoriais);
- algoritmos com eficiência comprovada.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Objetivos do Mini-curso

Vão ser apresentados:

- alguns algoritmos eficientes para o cálculo científico, mais especificamente o cálculo matricial;
- seu refinamento desde a versão mais simples até a versão mais complexa, com melhor desempenho;
- exemplos de implementações em bibliotecas padrão da área.
- exemplos de aplicações em quais se usam essas bibliotecas.

$\Rightarrow$ justificar a importância de conhecer este tipo de algoritmos para a obtenção de PAD.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Plano da apresentação

Plano da apresentação

Plano da seção

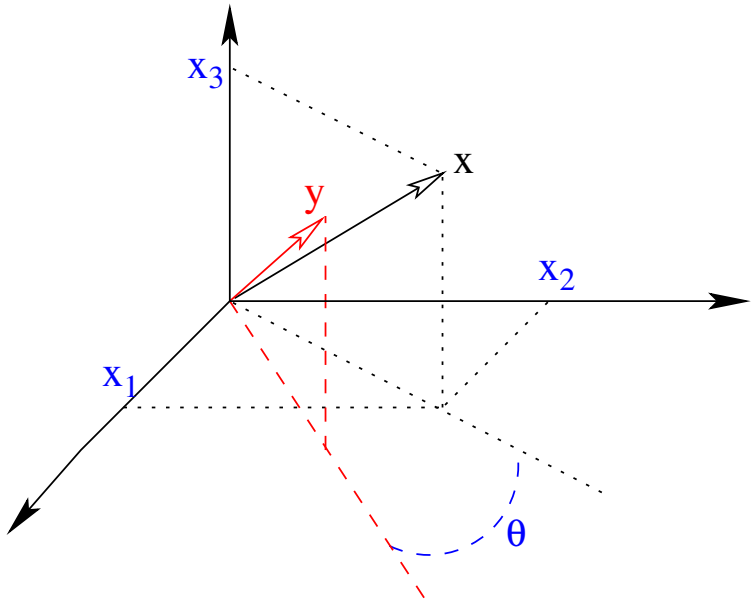
Vetor e matriz

- A é uma matriz $N \times N$;
  x é um vetor de tamanho N.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & \dots & a_{2N} \\ & \vdots & & & \\ & & a_{ij} & & \\ a_{N1} & \dots & a_{Nj} & \dots & a_{NN} \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

  - Vetores = pontos em um espaço a N dimensões; eles podem ser **somados** entre si, **multiplicados** por uma constante **escalar**, etc...
  - Uma matriz representa a ação de um operador linear sobre um vetor (rotação, translação). O vetor resultante é o produto Ax.

Matriz e geometria



Exemplo em $N = 3$ dimensões. O operador é uma rotação de ângulo θ, o eixo é (Ox_3).

Obtém-se

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1\cos\theta + x_2\sin\theta x_2 \\ -x_1\sin\theta + x_2\cos\theta \\ x_3 \end{pmatrix}.$$

Produto de matrizes

- Composição de operadores *Longleftrightarrow* protudo entre as matrizes;
- o produto de matrizes é fundamental pois aparece em quase todos os outros cálculos matriciais;
- a fórmula que dá o produto C da matriz A pela matriz B (ambas de tamanho $N \times N$ é

Fórmula do produto matricial

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}, \;\; i, j = 1 \ldots N.$$

Importância do produto matricial

Nessa parte vai ser estudada a implementação eficiente do produto matricial. Seu uso é fundamental:

- ele aparece em muitos outros algoritmos matriciais, uma vez que ele é a modelagem matemática da aplicação de uma função linear;
- sua implementação eficiente permite a obtenção de ótimo desempenho em um processador;
- ótimo exemplo pedagógico, simples, do ganho que pode se obter.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

O algoritmo `ijk`

Produto Matricial, algoritmo `ijk` trivial.

1: Entradas: 2 matrizes A e B de tamanho $N \times N$.
2: Saída: 1 matriz C de tamanho $N \times N$.
3:
4: $c_{ij} \leftarrow 0$
5: **for** $i = 1, 2, \ldots, N$ **do**
6: **for** $j = 1, 2, \ldots, N$ **do**
7: $c_{ij} \leftarrow 0$
8: **for** $k = 1, 2, \ldots, N$ **do**
9: $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$
10: **end for**
11: **end for**
12: **end for**

Plano da seção

Dois tipos de memória

Para este estudo, dois níveis de memória serão distinguidos:

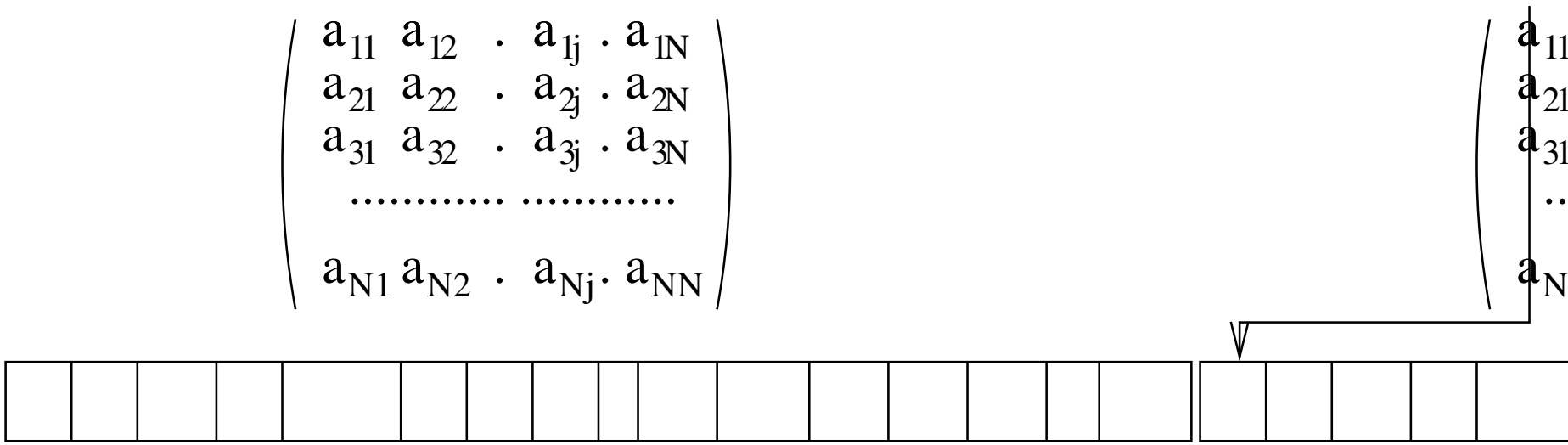
- a memória lenta, de capacidade grande;
- a memória rápida, de acesso muito mais rápido, mas que pode armazenar num dado momento apenas $\mathcal{C}$ elementos de matriz;
- um acesso a um dado d na memória lenta, que não está armazenado na memória rápida (miss), provoca a atualização da mesma com:
  - o elemento d,
  - os δ elementos próximos de d na memória lenta

  (mecanismo de paginação).

Essas hipóteses modelam uma hierarquia basica de memória (Cache), bem como os acessos numa memória distribuída através da rede.

Implementação das matrizes

Supõe-se que as matrizes são armazenadas na memória por coluna (*column-major*). É a norma Fortran, contrária à norma C.



(Segundo a sintaxe de C, o endereço de a_{ij} é $\&(a_{00}) + j * N + i$.)

Acessos na memória do algoritmo `ijk`

1: **for** $i = 1, 2, \ldots, N$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $c_{ij} \leftarrow 0$
4: **for** $k = 1, 2, \ldots, N$ **do**
5: $c_{ij} += a_{ik} b_{kj}$
6: **end for**
7: **end for**
8: **end for**

- Seja i, j fixados. Cada iteração de k provoca 1 *miss* de b_{kj} + 1 *miss* de c_{ij};
- (Não se levam em conta aqui as primeiras iterações);
- Quando j é incrementado, k passa de N a 1, e recomeça.
- Afinal, são $2N^3 + \mathcal{O}(N^2)$ *misses*.
- Obs.: pode-se poupar o miss de c_{ij} com um *buffer*.

Plano da seção

Ordens dos laços ijk

- Na verdade, pode-se executar os 3 laços em qualquer ordem.
- Basta cuidar na hora de inicializar c_{ij}. Pode ser feito uma vez por todas no início.
- A ordem `ijk` é a mais natural, devido à fórmula matemática...
- ...mas não é a mais eficiente!
- Os índices de linhas devem variar mais rapidamente: i e k.
- Logo, as ordens `ikj` e `kij` são naturalmente menos eficientes.

Três variações

Sobram então as ordens `jik`, `jki` e `kji`

Ordem jik

for $j = 1, \ldots, N$ **do**
  for $i = 1, \ldots, N$ **do**
    for $k = 1, \ldots, N$ **do**
      $c_{ij} += a_{ik} b_{kj}$
    end for
  end for
end for

Ordem jki

for $j = 1, \ldots, N$ **do**
  for $k = 1, \ldots, N$ **do**
    for $i = 1, \ldots, N$ **do**
      $c_{ij} += a_{ik} b_{kj}$
    end for
  end for
end for

Ordem kji

for $k = 1, \ldots, N$ **do**
  for $j = 1, \ldots, N$ **do**
    for $i = 1, \ldots, N$ **do**
      $c_{ij} += a_{ik} b_{kj}$
    end for
  end for
end for

Os *misses* em cada versão

- **A ordem jik** é parecida à ordem ijk.
  - Poupa-se apenas 1 *miss* a cada iteração de i;
  - isso leva a $N^3(1+\frac{3}{\delta})+\mathcal{O}(N^2)$ *misses*.
- **A ordem jki** é bem melhor. Após 1 iteração de i, durante as $\delta-1$ seguintes se encontram todos os coeficientes necessários.
  - Assim, tem 2 *misses* a cada δ iterações em i, ou seja $\frac{2N}{\delta}$ para um dado valor de j,k.
  - No total se obtém $\frac{2N^3}{\delta}+\mathcal{O}(N^2)$ *misses*.
- **A ordem kji** é quase igual à ordem jki, mas provoca alguns *misses* a mais: uma iteração de j provoca automaticamente um *miss* no coeficiente b_{kj}.

Em resumo...

Algoritmo	Número de *Misses*
ijk	$N^3(2+\frac{3}{\delta})$
jik	$N^3(1+\frac{3}{\delta})-N^2(1-\frac{1}{\delta})$
jki	$\frac{2N^3}{\delta}-N^2(1-\frac{1}{\delta})$

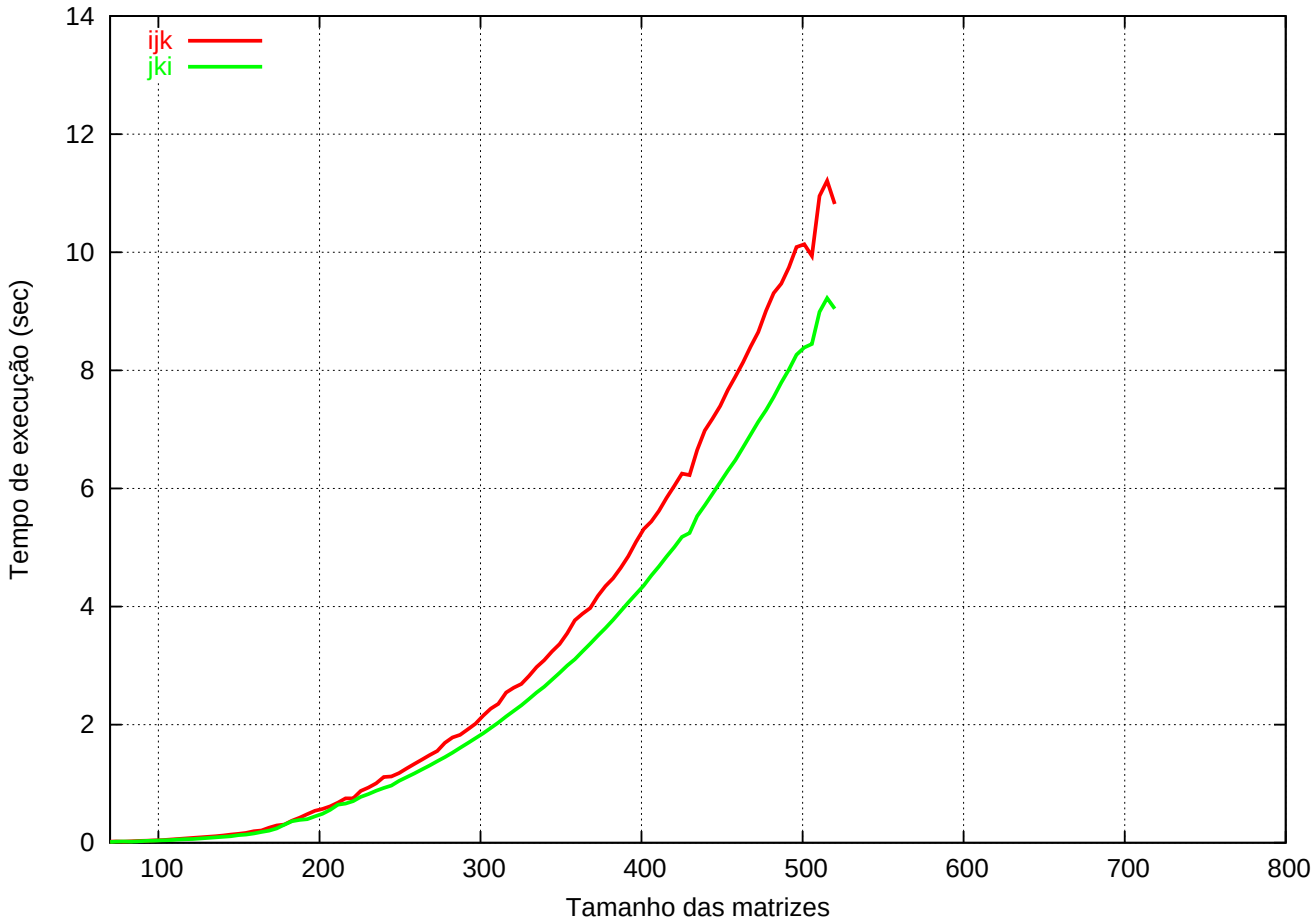
Entre a versão `ijk` e a versão melhor `jki`, a razão de *misses* é igual a

$$\frac{N^3(2+\frac{3}{\delta})}{\frac{2N^3}{\delta}-N^2(1-\frac{1}{\delta})} \underset{N\to\infty}{=} \delta.$$

Ele está enrolando com um modelo fora da real e fórmulas teóricas...

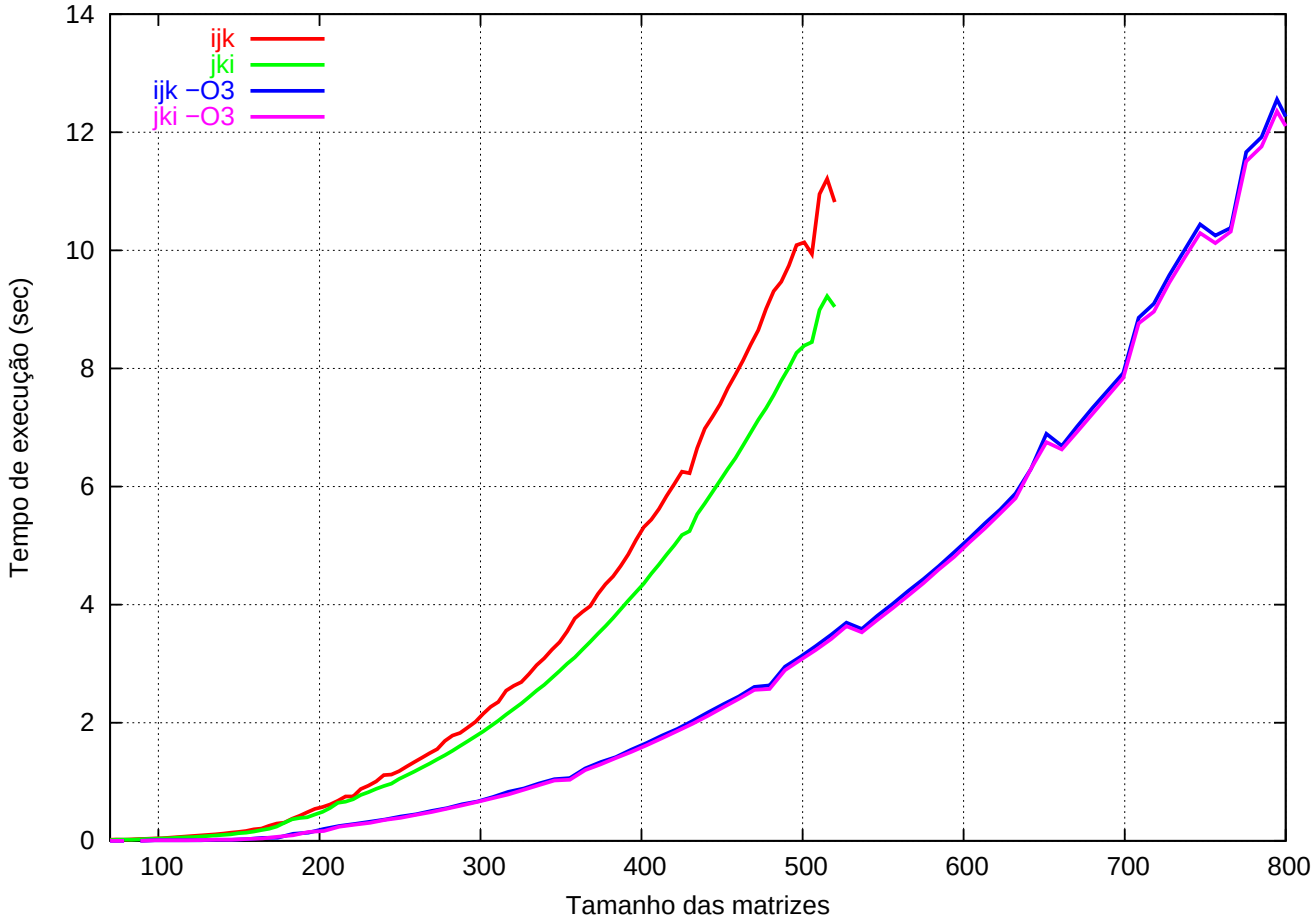
Algumas medições: ijk *vs.* jki (1)

Pentium III 733 Mhz, 800 Mhz, Cache 256Kb



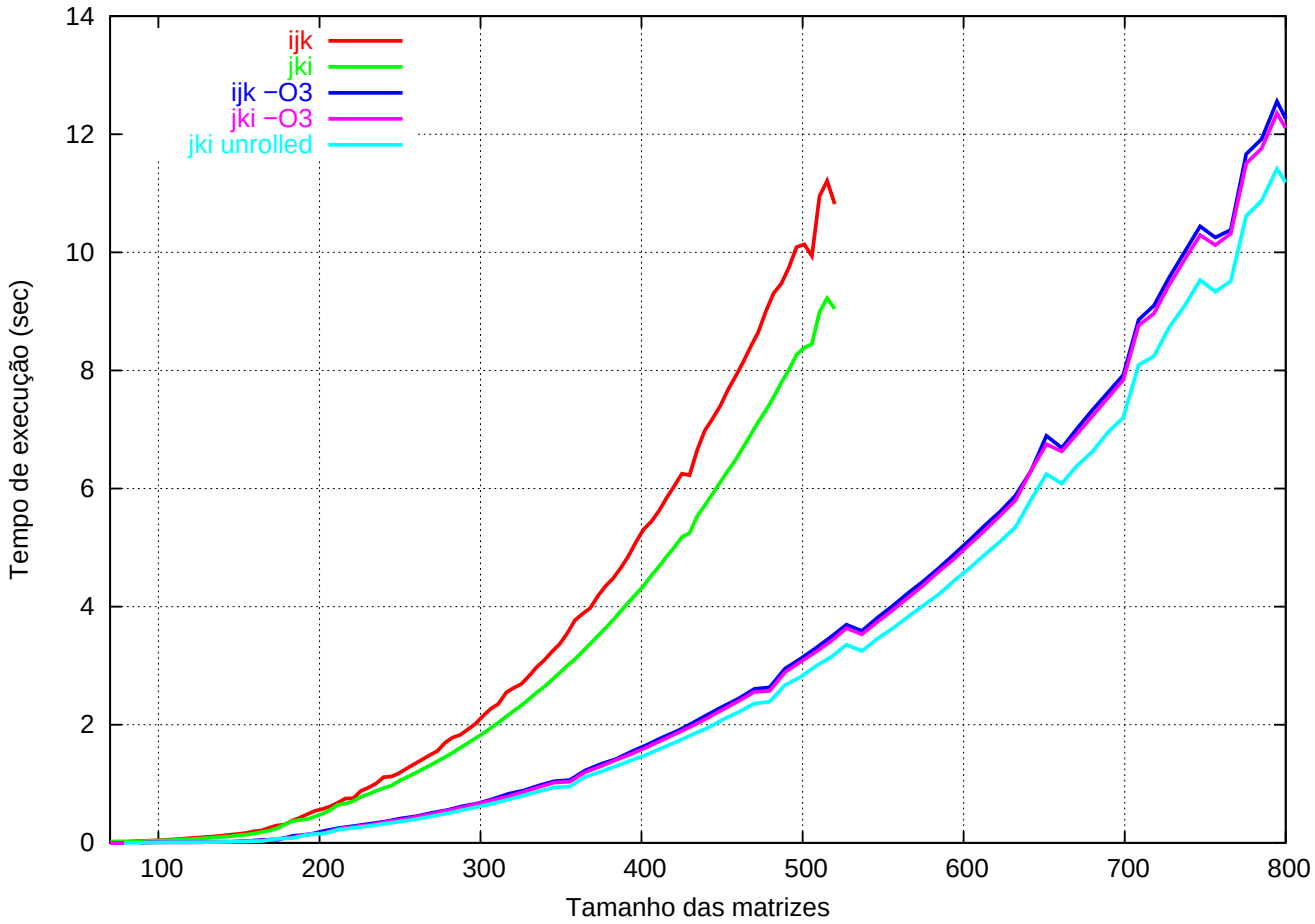
Bendito compilador!

gcc -O3 -funroll-all-loops



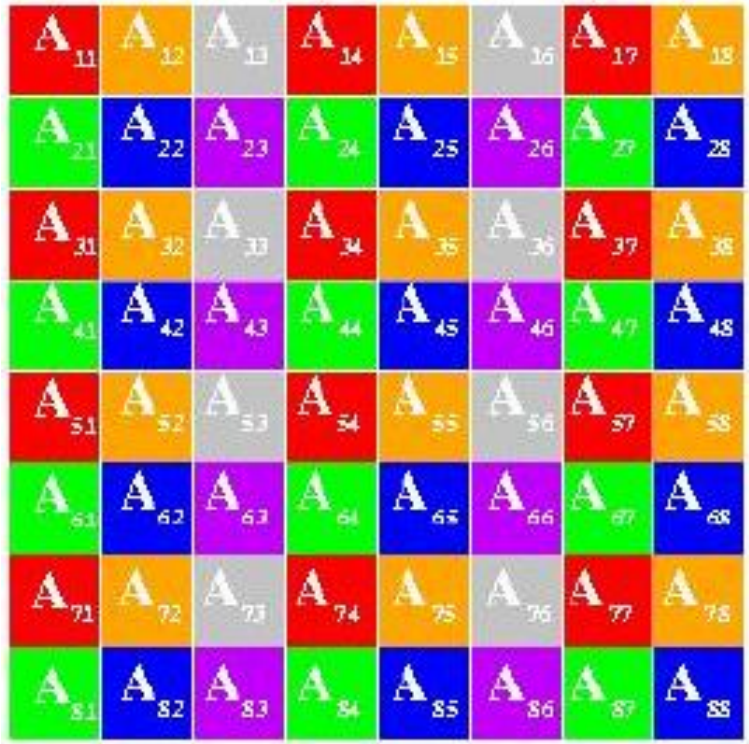
Além do compilador: desenrolar dos laços

O laço interno está desenrolado 8 vezes



Plano da seção

Decomposição da matriz em blocos



- Considera-se a matriz como uma “matriz de matrizes” (ou blocos);
- o tamanho *b* do bloco é tal que caibam na memória lenta “alguns” (*e.g.* 3) blocos.

Motivação da formulação em blocos

- Acessos na memória por linha/coluna = $\mathcal{O}(N^3)$ acessos à memória lenta;
- Usando blocos de tamanho b:
  - o cálculo do bloco $C_{ij}^{(k)} = A_{ik} \times B_{kj} \Rightarrow 2b^2$ acessos à memória distante;
  - precisa-se de N/b tais blocos para calcular a soma C_{ij} (que pode ser acumulada) na memória rápida;
  - assim, precisa-se de $\frac{N}{b} \times 2b^2 = 2Nb$ acessos à memória lenta para ter um dos $(N/b)^2$ blocos da matriz C.
  - Em soma, serão $2Nb \times \frac{N^2}{b^2} = \frac{2N^3}{b}$ acessos a serem realizados por essa versão em blocos.
- Em relação à versão sem blocos, ganha-se um fator b em número de acessos à memória.

Implementação de um algoritmo em blocos

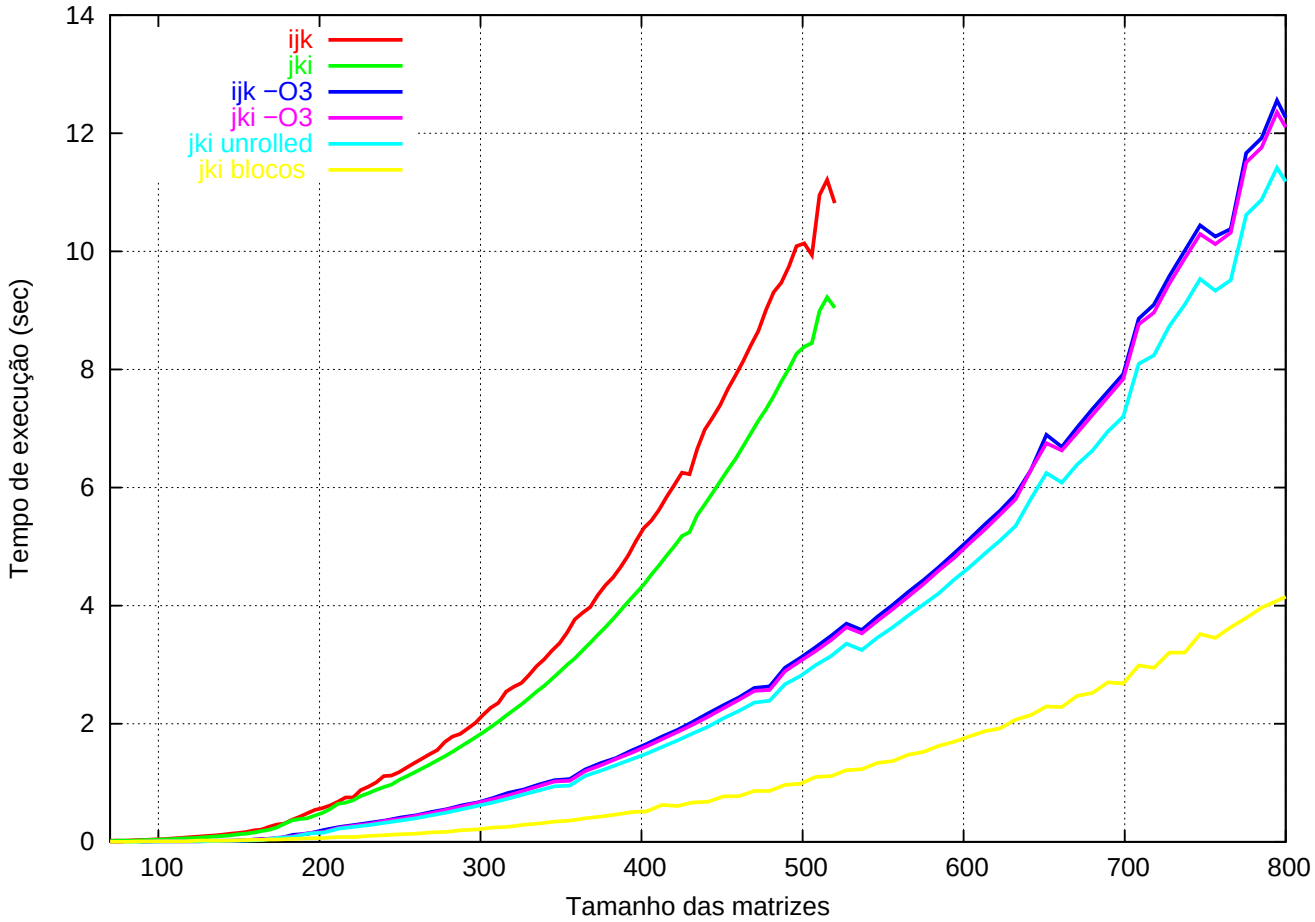
O código em C

```
#define BLOCK 64
inline void jki_blocos(int n, double* C, double* A, double* B) {
   int i, j, k, I, J, K;
   double Bkj;
   for (J=0 ; J<n ; J+=BLOCK) {
     for (K=0 ; K<n ; K+=BLOCK) {
       for (I=0 ; I<n ; I+=BLOCK) {
         for (j=J ; j<J+BLOCK-1 && j<n ; j++) {
           for (k=K; k<K+BLOCK-1 && k<n ; k++) {
             Bkj = B[j*n+k] ;
             for (i=I ; i<I+BLOCK-1 && i<n ; i++)
               C[j*n+i] += A[k*n+i]*Bkj ;

   }
         }
       }
     }
   }
}
```

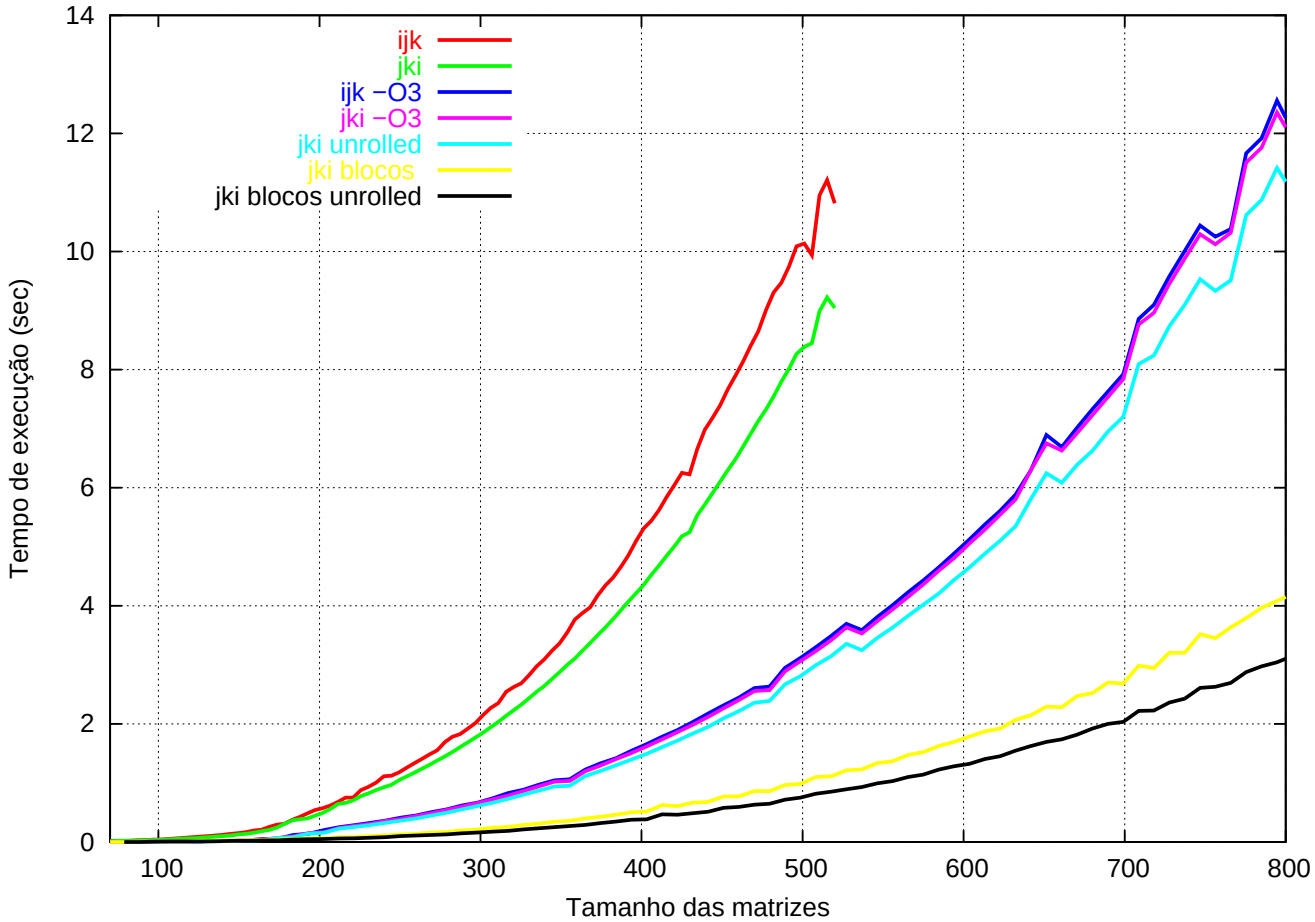
Desempenho da versão em blocos

Tamanho do bloco: 64 *doubles*



Versão em blocos com desenrolar dos laços

8 iterações são desenroladas



Conclusão: produto matricial

- Foram apresentados vários níveis de otimização:
  - Ordem dos laços (vide compilador!);
  - Desenrolo dos laços;
  - formulação em blocos.
- A versão em blocos apareceu teoricamente e experimentalmente muito melhor.
  - Teoricamente: ganha-se um fator δ.
  - Experimentalmente: ganha-se um fator 4.

Plano da seção

Algebra linear e aproximação de primeira ordem

- As equações usadas em física para modelar a realidade se baseam, em geral, em funções não lineares:
  - uso de potenciais não lineares (*e.g.* potencial de Coulomb).
- No entanto, os fenômenos não lineares são difíceis a analisar (caóticos), inclusive numericamente.
- A simplificação mais óbvia é a linearização das equações:
  - a fórmula de Taylor mostra que, quando o operador é relativamente regular, a simplificação é uma boa aproximação;
  - obtem-se desta forma uma equação tratável.
- A discretização dessa simplificação leva a matrizes e vetores.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

De onde vêm os sistemas?

1. as leis da física em geral são do tipo $F(x) = B$ (*e.g.* $B = 0$).
2. O espaço onde atuam os operadores é troncado (dimensão finita N).
3. O vetor x se define através de suas componentes na base $x_1, \ldots x_N$.
4. Os operadores se descrevem através de matrizes $N \times N$.

Sistema de equações lineares

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,N}x_N & = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,N}x_N & = b_2 \\ \vdots & \vdots \\ a_{N,1}x_1 + a_{N,2}x_2 + \ldots + a_{N,N}x_N & = b_N. \end{cases}$$

Usando uma notação matricial, escreve-se $Ax = b$.

Plano da seção

Uma etapa da eliminação de Gauss

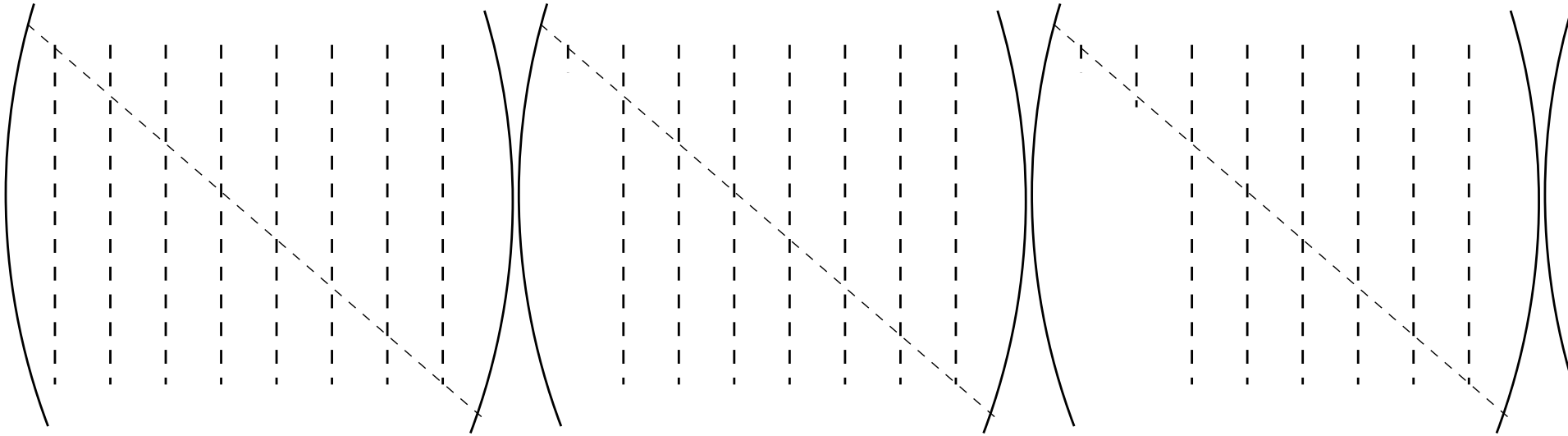
$$\times \frac{1}{a_{1,1}} \quad L_2 \leftarrow L_2 - \frac{a_{21}}{a_{11}} \times L_1$$

$$\begin{cases} a_{1,1}x_1 & +a_{1,2}x_2 & +\ldots & +a_{1,N}x_N & = b_1 \\ a_{2,1}x_1 & +a_{2,2}x_2 & +\ldots & +a_{2,N}x_N & = b_2 \\ & \vdots & & = \vdots & \\ a_{N,1}x_1 & +a_{N,2}x_2 & +\ldots & +a_{N,N}x_N & = b_N. \end{cases}$$

$$\begin{cases} x_1 & +\frac{a_{1,2}}{a_{11}}x_2 & +\ldots & +\frac{a_{1,N}}{a_{11}}x_N & = \frac{b_1}{a_{11}} \\ \frac{a_{2,1}}{a_{1,1}}x_1 & +\frac{a_{2,2}}{a_{1,1}}x_2 & +\ldots & +\frac{a_{2,N}}{a_{1,1}}x_N & = \frac{b_2}{a_{1,1}} \\ & \vdots & & = \vdots & \\ \frac{a_{N,1}}{a_{1,1}}x_1 & +\frac{a_{N,2}}{a_{1,1}}x_2 & +\ldots & +\frac{a_{N,N}}{a_{1,1}}x_N & = \frac{b_N}{a_{1,1}}. \end{cases}$$

$$\begin{cases} x_1 & +\frac{a_{1,2}}{a_{11}}x_2 & +\ldots & +\frac{a_{1,N}}{a_{11}}x_N & = \frac{b_1}{a_{11}} \\ & (\frac{a_{2,2}}{a_{1,1}} - \frac{a_{21}}{a_{11}})x_2 & +\ldots & +(\frac{a_{2,N}}{a_{1,1}} - \frac{a_{21}}{a_{11}})x_N & = (\frac{b_2}{a_{1,1}} - \frac{a_{21}}{a_{11}})b_1 \\ & \vdots & & & = \vdots \\ a_{N,1} & a_{N,2} & & a_{N,N} & b_{\cdot\cdot} \end{cases}$$

Etapas seguintes



- Pode-se repetir este procedimento para zerar os elementos das outras colunas da nova matriz.
- Obtém-se desta forma, após $N-1$ etapas, um sistema na forma:

$$\begin{cases} a_{1,1}^{(N-1)} x_1 + a_{1,2}^{(N-1)} x_2 + \ldots + a_{1,N}^{(N-1)} x_N &= b_1^{(N-1)} \\ a_{2,2}^{(N-1)} x_2 + \ldots + a_{2,N}^{(N-1)} x_N &= b_2^{(N-1)} \\ \vdots & \vdots \\ a_{N,N}^{(N-1)} x_N &= b_N^{(N-1)}. \end{cases}$$

Pivoteamento de Gauss e fatoraçã LU

- Este algoritmo se chama pivoteamento de Gauss. O pivô é o elemento da diagonal que serve para dividir as linhas.
- A matriz triangular é chamada U. Sua diagonal inclui os pivôs. As operações de eliminação se reduzem numa matriz L, também triangular.
- Se um pivô vale zero, é preciso permutar as linhas (ou as colunas) da matriz até achar um pivô não zero. A permutação pode se efetuar através do produto por uma matriz P.
- Afinal,

Fatoração LU

o pivoteamento de Gauss leva à fatoração da matriz A sob a forma $A = P \times L \times U$.

Complexidade do pivoteamento de Gauss

- Complexidade: para cada uma das $j = 2, \ldots, N-1$ colunas zerada, deve-se atualizar $(N-j)^2$ coeficientes através de produtos e divisões. O número exato de operações é

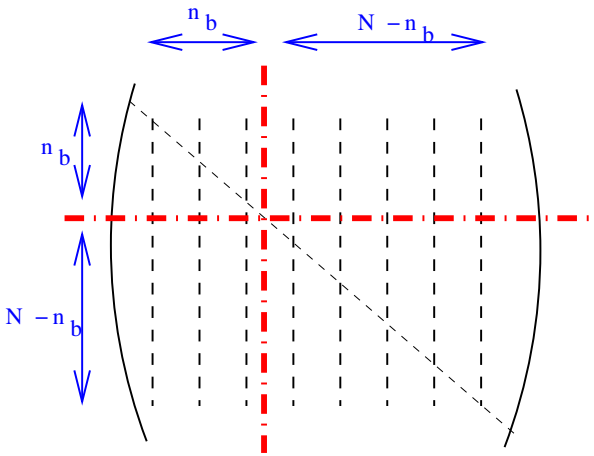
$$\frac{2N^3}{3} + \frac{N^2}{2} + \mathcal{O}(N)$$

- Observação: é também a ordem de magnitude da complexidade do produto matricial!

Plano da seção

Porque re-formular a fatoração *LU* em blocos?

- conforme foi apresentado, uma implementação baseada em blocos é mais eficiente;
- via de régra, é fácil alterar algoritmos tais como o pivoteamento de Gauss para usar blocos;



- Seja n_b um número que divide n. Nota-se

$$PA = \begin{pmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & 0 & I \end{pmatrix} \times \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & A'_{22} & A'_{23} \\ & A'_{32} & A'_{33} \end{pmatrix}$$

- $L_{\cdot\cdot}$, $U_{\cdot\cdot}$ e $A'_{\cdot\cdot}$ são blocos das matrizes

Fatoração em blocos

- Após mais uma etapa da fatoração, deve-se obter:

$$\begin{pmatrix} I & \\ & P_2 \end{pmatrix} PA = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & I \end{pmatrix} \times \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & A'_{33} \end{pmatrix}.$$

- As sub-matrizes $\begin{pmatrix} L_{22} & \\ L_{32} & I \end{pmatrix}$ e $\begin{pmatrix} L_{22} & U_{23} \\ & A'_{33} \end{pmatrix}$ correspondem à fatoração LU do bloco $\begin{pmatrix} A'_{22} & A'_{23} \\ A'_{32} & A'_{33} \end{pmatrix}$

Etapas da fatoração em blocos

1. Fatoração LU do bloco $P_2 \begin{pmatrix} A'_{22} \\ A'_{32} \end{pmatrix} = \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22}$.
2. Permutação das linhas (multiplicação por P_2);
3. Cálculo do bloco pivô $U_{23} \leftarrow L_{22}^{-1} A'_{23}$;
4. Atualização do bloco A'_{33}:

$$A'_{33} \leftarrow A'_{33} - L_{32} U_{23}$$

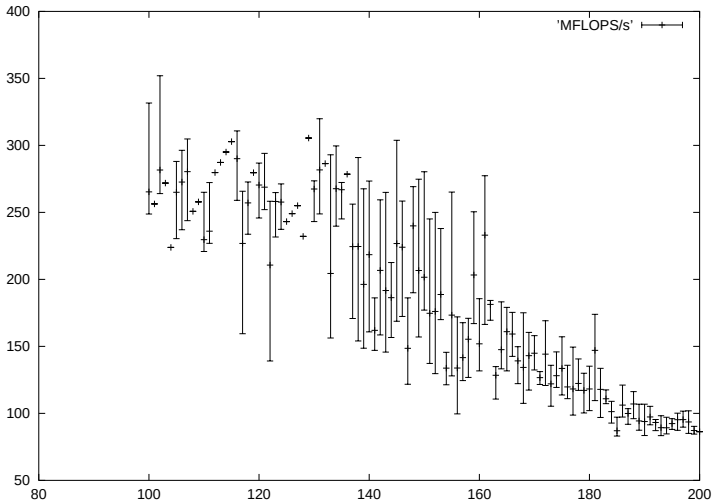
Plano da seção

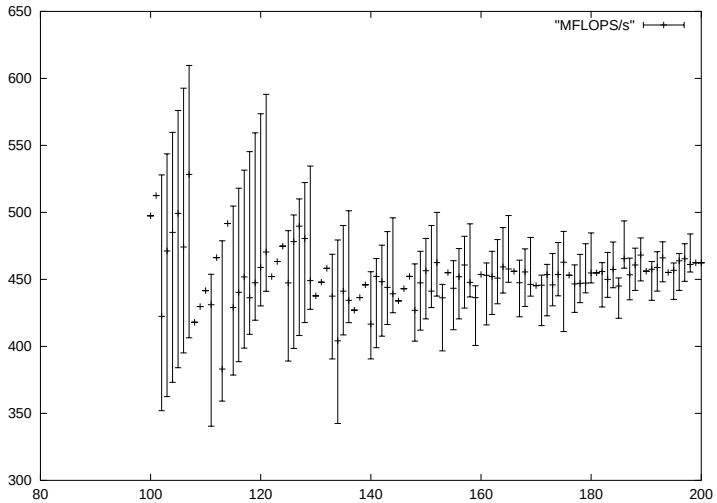
Basic Linear Algebra Subroutine

- 3 níveis de BLAS
  - BLAS-1: todas as operações tipo “vetor-vetor” (produto escalar, soma de vetores, multiplicação de um vetor por um escalar, etc.);
  - BLAS-2: operações que implicam uma matriz e um vetor (*e.g.* produto matriz-vetor). Complexidade $\mathcal{O}(N^2)$;
  - BLAS-3: operações entre matrizes (*e.g.* produto de duas matrizes). $\mathcal{O}(N^3)$.
- Existe várias implementações:
  - dos fabricantes (*cf.* compiladores) com otimizações em nível de linguagem de máquina.
  - código livre, em Fortran-77 (netlib).
- Impacto no desempenho!

Desempenho das BLAS

- Experimentos com um Pentium III 733 MHz.
- Desempenho máximo esperado: 733 MFlops/s (1 op. vírgula flutuante / ciclo de relógio).





pgcc

-O2 -tp p6 -Minfo -Munroll=n:16 -mp

-Mvect=assoc,cachesize:262144 -Ml3f

LAPACK

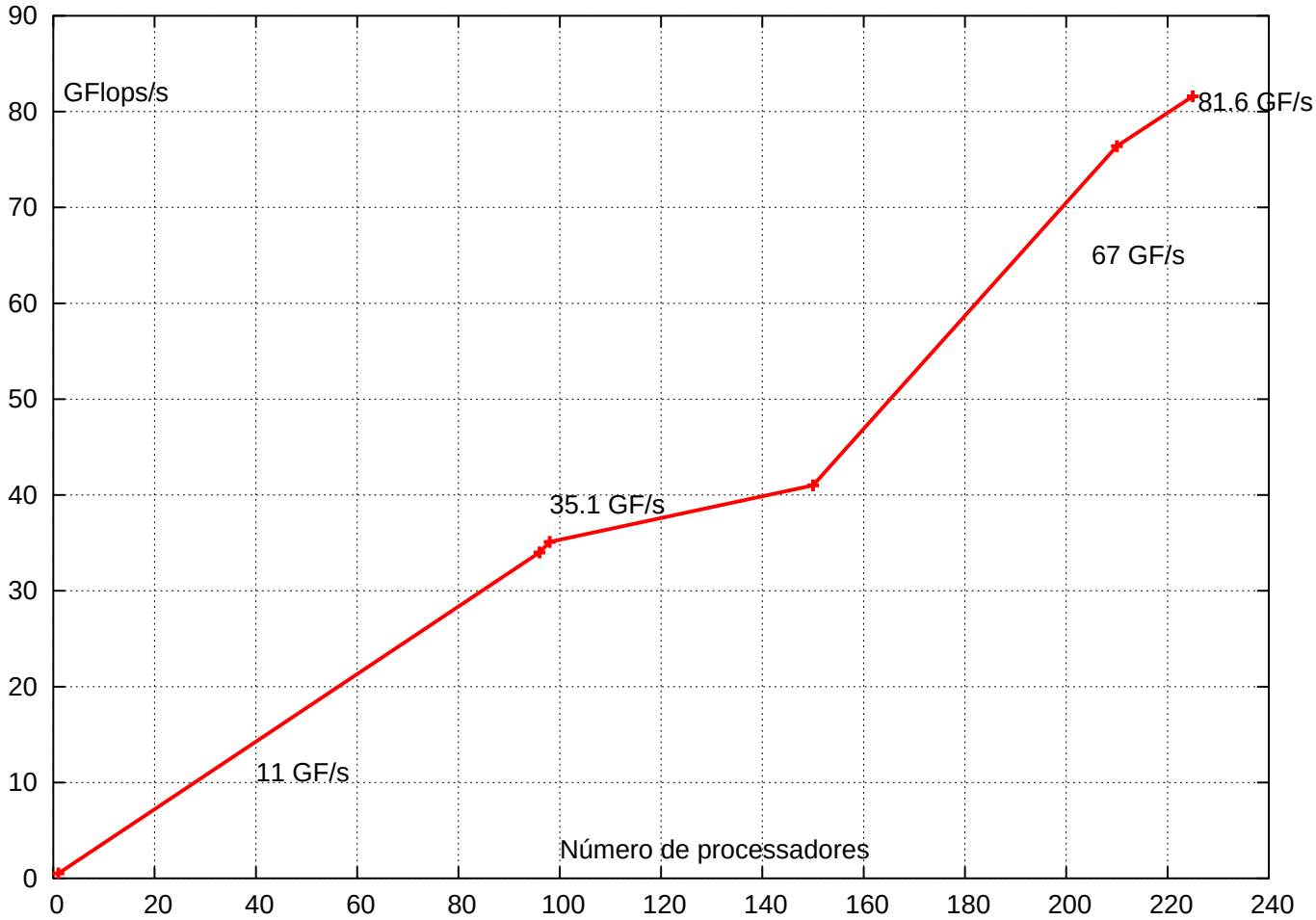
- A biblioteca Linpack foi desenvolvida nos anos 70 para resolver sistemas de equações lineares, para vários tipos de matrizes.
- Foi integrada com a biblioteca Eispack, passando a se chamar LAPACK (Linear Algebra Package) no fim dos anos 80.
  - Inclui mais algoritmos, tais como o cálculo de auto-vetores.
- LAPACK foi projetado em cima das BLAS para garantir a eficiência, com algoritmos em blocos.
  - as operações em nível de bloco implementadas nas BLAS.
- Para máquinas com memória distribuída, existe *ScaLAPACK*.
  - em cima de uma versão paralela das BLAS (P-BLAS),
  - com uma camada específica de comunicações dedicadas ao cálculo matricial (BLACS).

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

O *benchmark* Linpack

- surgiu a partir da biblioteca Linpack;
- Linpack-100 em 1979, com classificação de até 23 processadores;
- Linpack-1000 durante os anos 80;
- Highly-Parallel Linpack (HPL): usado como referência na comparação dos supercomputadores, através do TOP500, desde 1993.
- Testa a escalabilidade de uma máquina paralela e seu desempenho bruto.
  - O único critério imposto pelo *benchmark* é o algoritmo de fatoração LU com complexidade $\frac{2N^3}{3} + 2N^2$.
  - Pode-se alterar todos os outros parâmetros do teste: tamanho da matriz, algoritmos de comunicações globais, topologia de rede, etc.

Escalabilidade do linpack num agregado



O agregado é do INRIA, formado por 225 processadores PIII, rede

Plano da seção

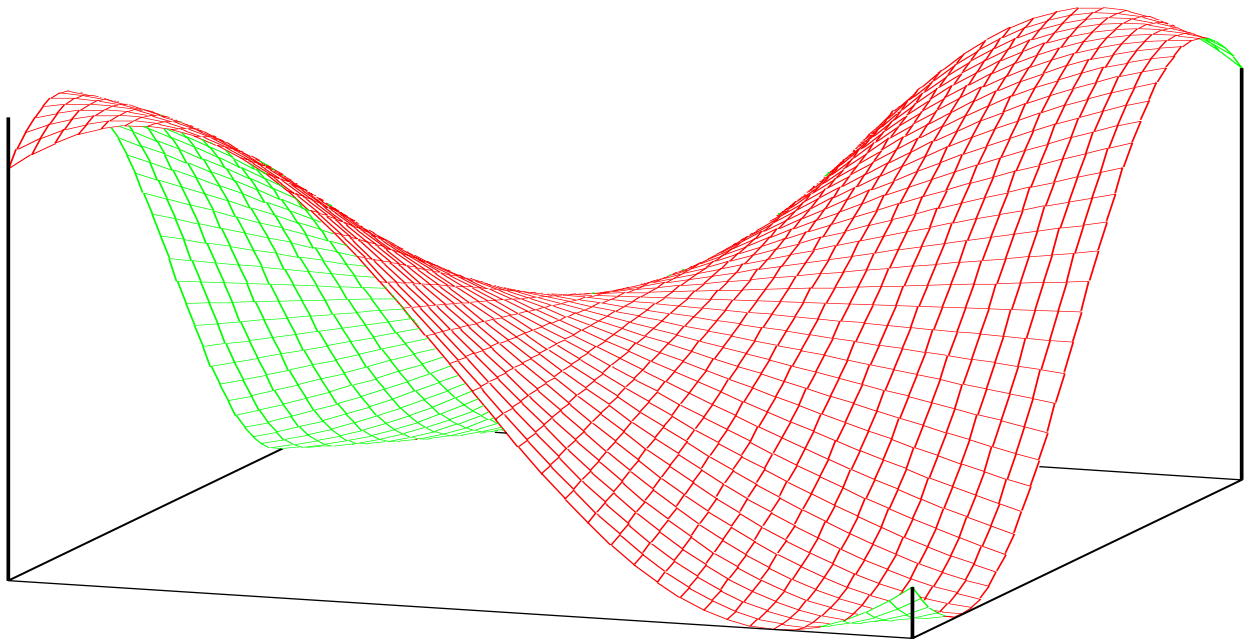
Métodos diretos e iterativos

- Métodos diretos: implicam na fatoração da matriz. O algoritmo se aplica diretamente à estrutura da matriz.
  - Vantagens: a fatoração pode servir a resolver mais de um sistema; a complexidade é determinista.
  - Inconvenientes: precisa-se de armazenar todos os coeficientes da matriz; tais algoritmos “enchem” a matriz durante a fatoração.
- Métodos iterativos: usam a matriz apenas como operador que age com vetores, para construir uma sequência de vetores cujo limite é a solução do sistema.
  - Vantagens: nem precisa da matriz; pode-se usar tipos de armazenamento específicos para matrizes esparsas.
  - Inconvenientes: convergência dependente do vetor inicial; garantia unicamente para dados tipos de matrizes.

O algoritmo do Gradiente Conjugado

- O Gradiente Conjugado (GC) é um dos algoritmos diretos mais antigos usados para resolver um sistema $Ax = b$: ele existe há 50 anos.
- o GC cria iterativamente uma sequência de vetores $x^{(i)}$ cujo limite é x quando converge.
- É um método de "gradiente", no sentido físico que, a cada iteração, calcula um vetor que dá a direção que minimiza a energia do operador $J(y) = \frac{1}{2} y^T A y - b^T y + c$.
- A norma do vetor também é calculada de forma tal que permita "descer" nesta direção até o ponto de energia mínima ao longo dela.

Ilustração do GC



Expressão matemática do GC

- Quer-se minimizar $\mathcal{E}(x) = \frac{1}{2} y^T A y - b^t y + c$.
- A direção de maior descida é dada por $-\mathcal{E}'(x) = b - Ax$. Nota-se $r = b - Ax$.
- Seguindo esta direção, chega-se até o ponto $x_1 = x + \alpha r$.
- Qual valor α é tal que se "ultrapassa" o ponto crítico?
- É α tal que $\frac{\partial}{\partial \alpha} \mathcal{E}(x_1) = 0$.
  - *i.e.* $\mathcal{E}'^T(x_1) \frac{\partial}{\partial \alpha}(x_1) = 0$,
  - *i.e.* $(b - Ax_1)^T r = 0$,
  - *i.e.* $(b - A(x + \alpha r))^T r = 0$,
  - *i.e.* $\alpha = \frac{r^T r}{r^T A r}$.
- No caso do GC, as direções de descida são A-ortogonais!
  - altera um pouco a definição da direção...

Algoritmo do Gradiente Conjugado

O algoritmo

1: $i \leftarrow 0$
2: $d^{(i)} \leftarrow b - Ax^{(i)}$
3: $r^{(i)} \leftarrow b - Ax^{(i)}$
4: **while** $r^{(i)T} r^{(i)} > \epsilon$ **do**
5: $\quad \alpha \leftarrow \frac{r^{(i)T} r^{(i)}}{d^{(i)T} A d^{(i)}}$
6: $\quad x^{(i+1)} \leftarrow x^{(i)} + \alpha d^{(i)}$
7: $\quad r^{(i+1)} \leftarrow r^{(i)} - \alpha A d^{(i)}$
8: $\quad \beta \leftarrow \frac{r^{(i+1)T} r^{(i+1)}}{r^{(i)T} r^{(i)}}$
9: $\quad d^{(i+1)} \leftarrow r^{(i+1)} + \beta d^{(i)}$
10: $\quad i \leftarrow i + 1$
11: **end while**

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Interesse do Gradiente Conjugado

O GC é especialmente bem projetado para o PAD. De fato, ele precisa efetuar a cada iteração:

- 3 cópias/somas de vetores. É uma operação tipo BLAS-1;
- 2 produtos escalares ($r^{(i)T} r^{(i)}$ e $d^{(i)T} A d^{(i)}$). São operações tipo BLAS-1;
- 1 produto matriz-vetor ($Ad^{(i)}$), que é uma operação BLAS-2.
- Basta armazenar, de uma iteração para a seguinte, apenas os valores dos vetores $x^{(i)}$, $r^{(i)}$ e $d^{(i)}$.

Além de serem implementadas eficientemente pelas BLAS, essas operações são fáceis de paralelizar!

Conclusão: resolução de sistemas de equações

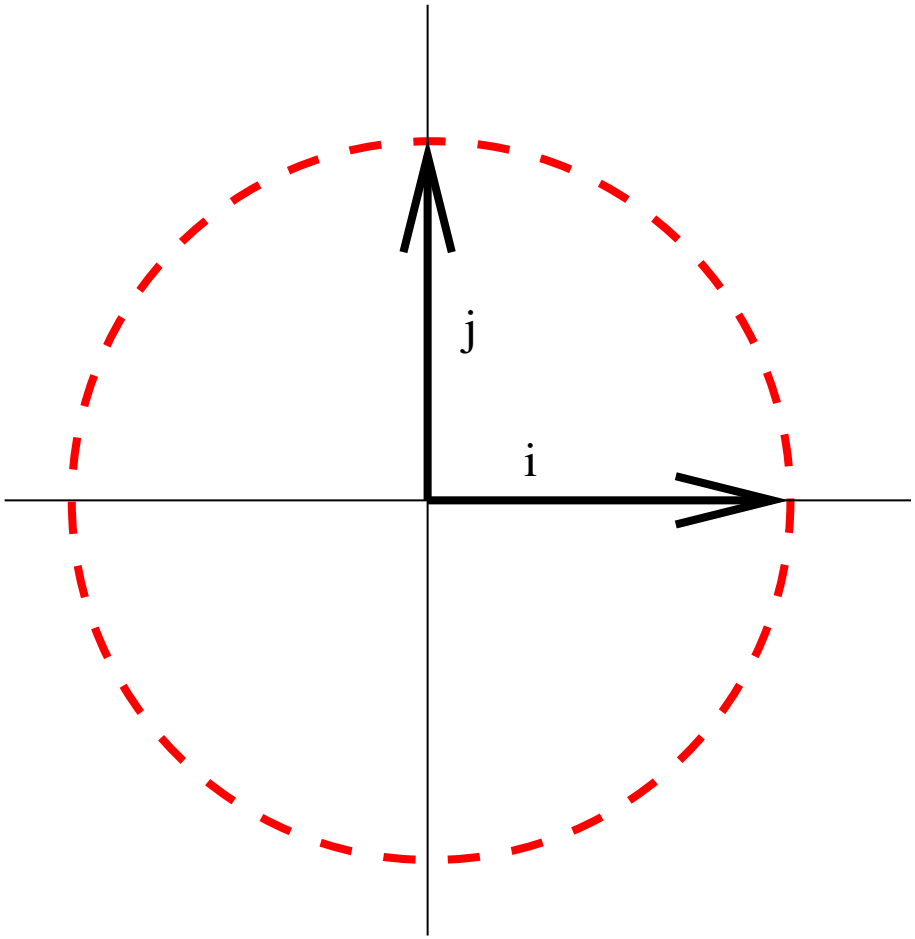
- Trata-se de uma operação fundamental para todo o cálculo numérico!
- Existe várias implementações padrões em bibliotecas: LAPACK para computadores seqüenciais, scaLapack para máquinas paralelas, o *benchmark* Linpack, ...
- as implementações de alto-desempenho usam versões em blocos.

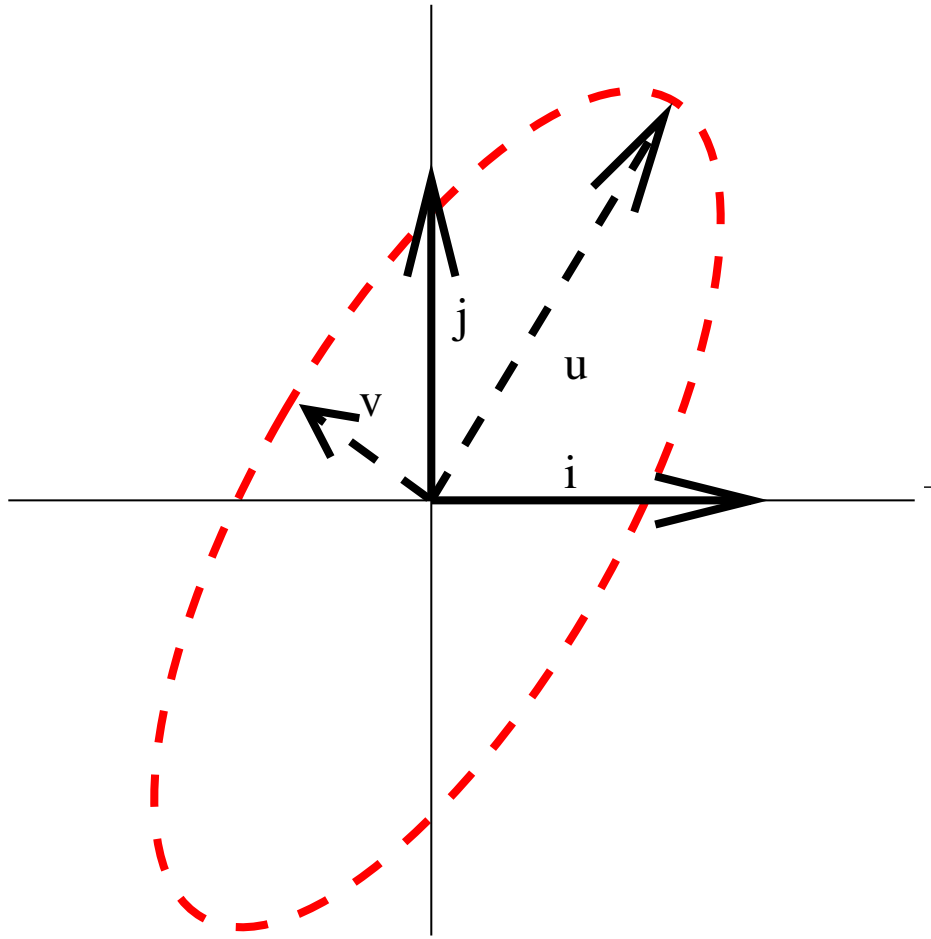
Plano da seção

Definição

- Um outro problema de alta relevância é o cálculo de auto-valores e vetores de uma matriz A.
- *auto-valores* e *auto-vetores* são escalares $\lambda_i, i = 1 \ldots, n$ e vetores $v_i, i = 1, \ldots, n$ tais que $Av_i = \lambda_i v_i$.
- Fisicamente:
  - os auto-vetores representam as direções no espaço vetorial nas quais A mais age;
  - o auto-valor associado a um auto-vetor representa a intensidade da ação de A nessa direção.

Representação geométrica em 2 dimensões





Motivação

Duas noções “concretas” com auto-valores:

1. auto-valor = valor mínimo.
   - $\lambda_i = (v_i, Av_i)$ quando v_i tem norma 1.
   - λ_i é o mínimo possível de (x, Ax) quando A é simétrica.

   Vide problemas de minimização (de energia).
2. *auto-valor* = estado estacionário = estabilidade.
   - vide a seqüência $u_n = a.u_{n-1}$;
   - bastante usado com cadéias de Markov.

   Vide o teorema do ponto fixo.

Plano da seção

Algoritmo da potência

idéia simples:

- Efetuar a fatoração LU.
- iterar a ação do operador sobre um vetor inicial aleatório u_0;
- Cada iteração vai "torcer" o espaço na direção do auto-vetor, até chegar a um espaço reduzido a uma reta = a direção do auto-vetor v_i procurado;
- Para obter λ_i, basta calcular (v_i, Av_i);
- isso significa calcular $A^k u_0$ à iteração k;
- este algoritmo se chama "algoritmo da potência".
- serve para calcular o auto-valor de maior módulo.

Idéia do algoritmo de Lanczos

Espaço de Krylov

É o espaço vetorial definido pelas iterações de uma matriz A a partir de um vetor u_0: $\mathcal{K}_{u_0}(A) = \mathcal{V}(u_0, Au_0, A^2u_0, \ldots, A^mu_0)$.

- Busca de uma solução num espaço menor e representativo da solução;
- o espaço dos iterados parece bom (espaço de Krylov);
- basta calcular a cada iteração k a projeção da matriz A neste espaço vetorial e seus auto-valores.

Algoritmo de Lanczos

1: $\beta_1 \leftarrow 1, q_0 \leftarrow u_0$.
2: **Iterações**
3: **for** $j = 1, 2, \ldots, m$ **do**
4: $\quad r_j \leftarrow Aq_j - \beta_j q_{j-1}$
5: $\quad \alpha_j \leftarrow r_j^* q_j$
6: $\quad r_j \leftarrow r_j - \alpha_j q_j$
7: $\quad \beta_{j+1} \leftarrow \|r_j\|_2$
8: $\quad q_{j+1} \leftarrow r_j / \beta_{j+1}$
9: **end for**
10: Retornar $Q^T \times A \times Q$ {Q é a matriz formada pelos q_j.}

Algoritmo de Lanczos com Restart

- A aplicação do algoritmo de Lanczos fornece uma aproximação do auto-vetor v_j procurado.
- Pode-se repetir a construção iterativa do espaço $\mathcal{K}_{v_j}(A)$ a partir deste auto-vetor para obter uma melhor aproximação!
- Essa versão, chamada com restart permite limitar o tamanho m do espaço de Krylov, e, logo, a memória usada.
  - Por iteração, deve se armazenar m vetores q_j que formam a base de $\mathcal{K}_{v_j}(A)$!
- Trata-se de um algoritmo iterativo que usa apenas produtos escalares e produtos matriz-vetor.

A biblioteca ARPACK

- Este algoritmo foi implementado na biblioteca ARPACK (ARnoldi PACKage).
- Existe uma versão distribuída chamada P-ARPACK, baseada no MPI.
- Serve para resolver problemas com até dezenas de milhões de variáveis.
- Usa um mecanismo de *template*: o usuário deve apenas fornecer o produto matriz-vetor distribuído.
  - O cálculo da norma para o controle da convergência já vem paralelizado.
  - O resto das operações é duplicado em todos os processadores.

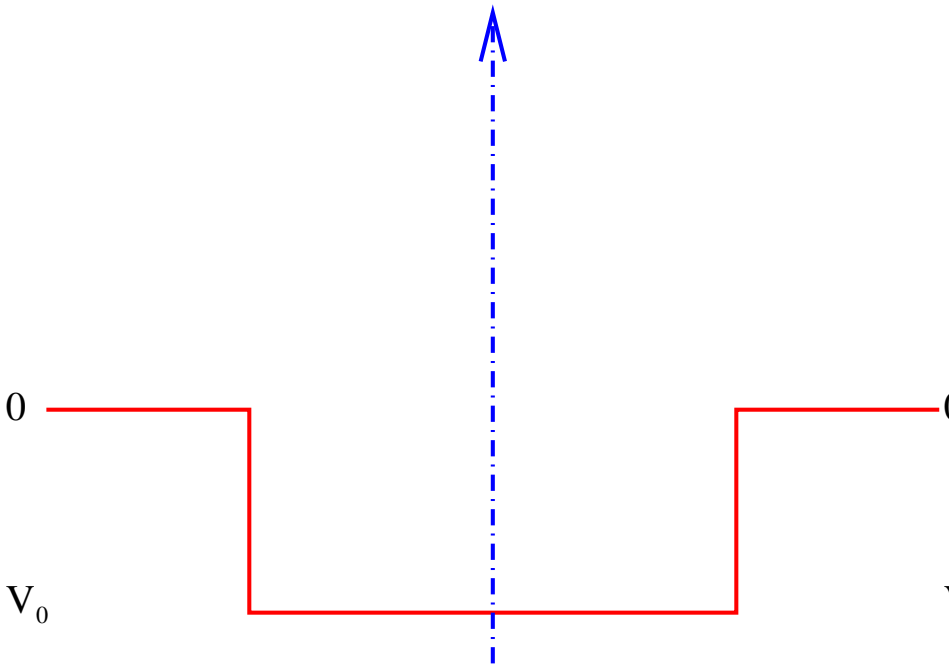
Plano da seção

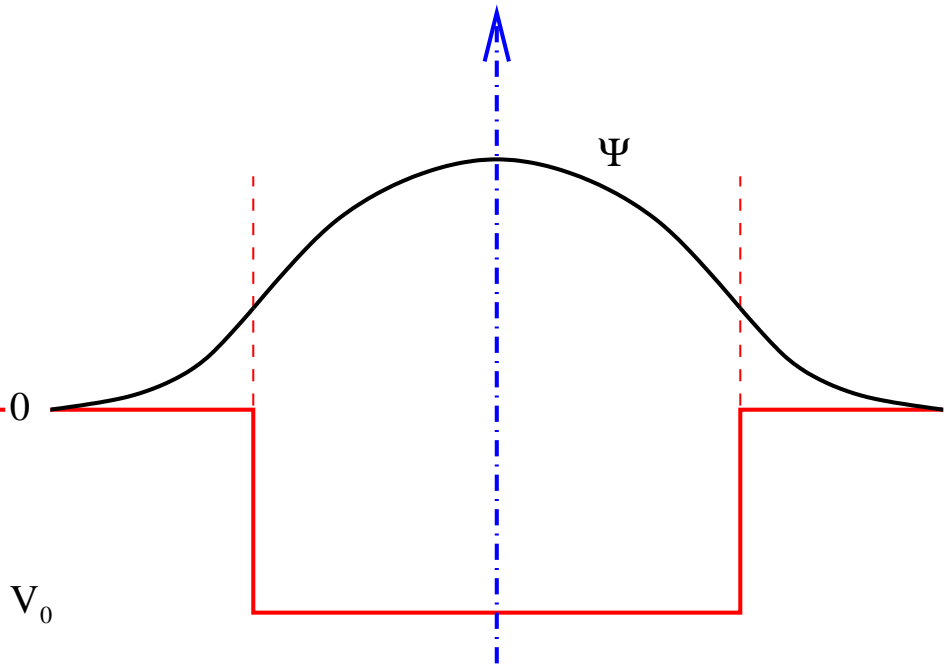
Mecânica quântica e equação de Schroedinger

- Modelagem do comportamento de conjuntos de partículas na escala atômica.
- Não determinista mas probabilístico
  - um eletro não é uma bolinha sólida...
- Cada sistema é descrito através de uma função de onda $\Psi(x, y, z, t)$ que dá a probabilidade de presença do sistema num ponto do espaço-tempo.
- As ações sobre o sistema (potenciais, energia...) se modelam através de operadores que transformam a função de onda. Por exemplo:

$$-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\Psi(x, y, z) + V(x, y, z)\Psi(x, y, z).$$

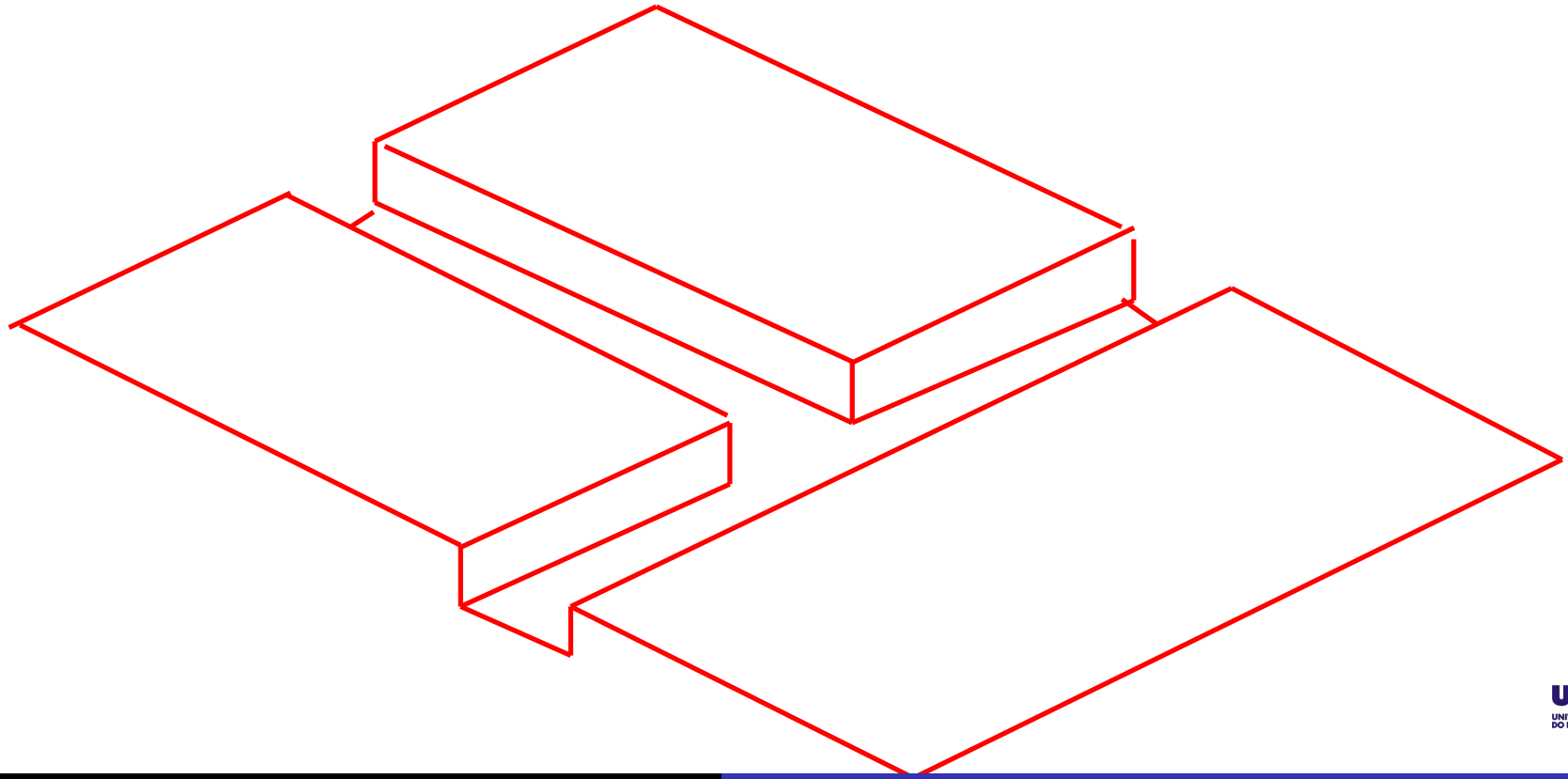
O problema do poço de potencial



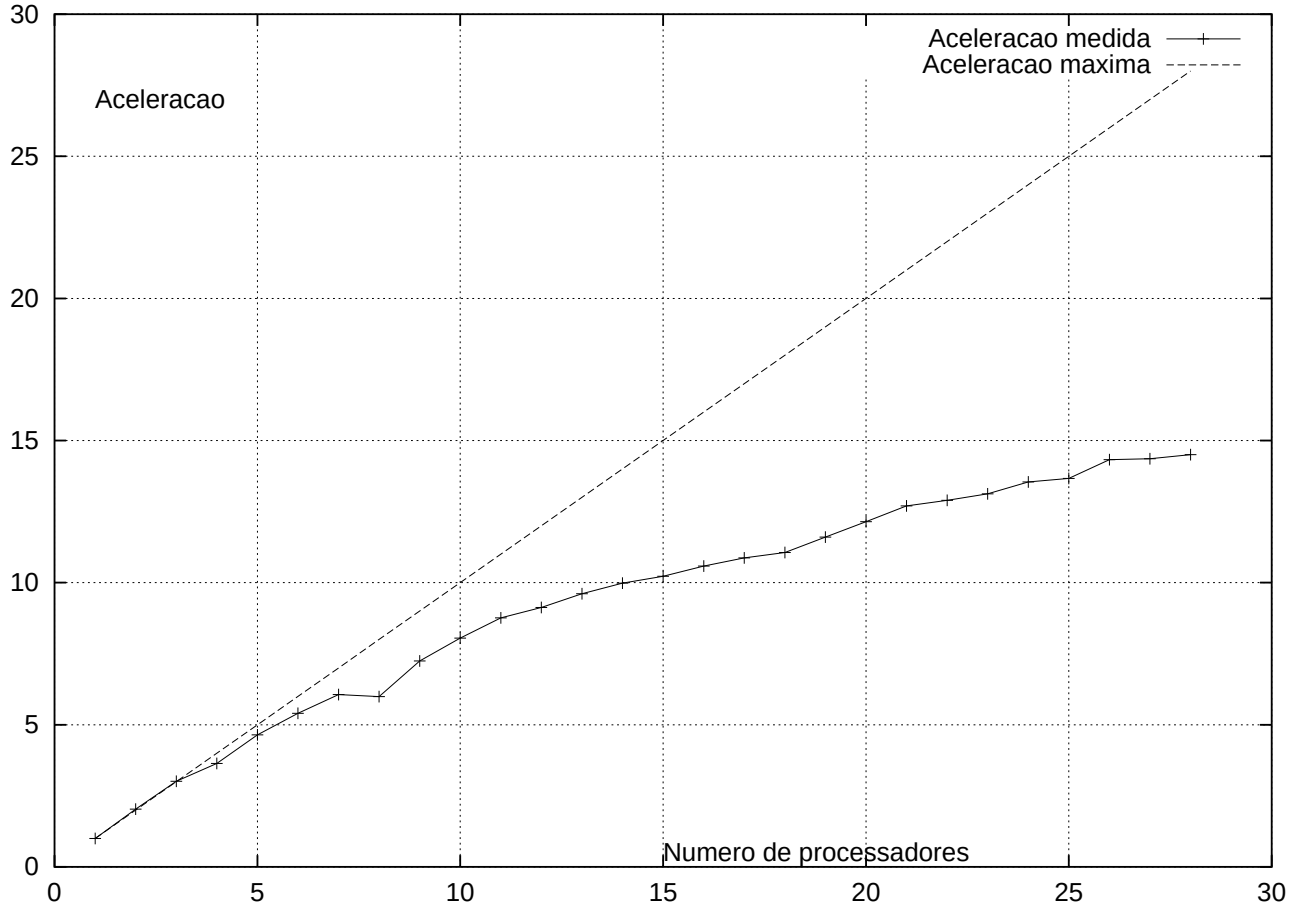


- Um exemplo 1D.
- Potencial $V(x)$ constante com 2 valores.

Exemplo: o potencial em T



Resolução com o P-ARPACK



As execuções foram feitas num Cray-T3D em 1999, e no cluster do INRIA em 2001.

Conclusão sobre o cálculo de auto-valores

- Auto-valores e auto-vetores são fundamentais para muitos modelos numéricos.
- Exemplos de aplicações: mecânica quântica, cadéias de Markov, ...
- Existe uma gama de algoritmos diretos e iterativos eficientes e altamente paralelos.
- Os mesmos se encontram em bibliotecas tais como a ARPACK.

Resumo

- Foram apresentados alguns algoritmos para o cálculo matricial:
  - o produto matricial;
  - a resolução de sistemas de equações lineares;
  - o cálculo de auto-valores e auto-vetores.
- Técnicas foram apresentadas para melhorar o desempenho: re-ordenação dos laços; formulação em blocos.
- Exemplos de aplicações foram apresentados.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Moral da história

- De nada adianta programar uma máquina distribuída sem buscar o melhor desempenho sequencial antes.
- Por isso é importante ter um bom entendimento do algoritmo.
- Não é a coisa mais importante em PAD — é apenas mais um elemento a dominar, ao lado da arquitetura, da programação, da avaliação de desempenho, etc. . .
- Trabalhar com aplicações é complicado, o diálogo com usuários também — a modelagem matemática pode servir como ponte.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Processamento de Alto Desempenho e paralelismo

- Falou-se relativamente pouco de paralelismo!
- A noção chave é a granularidade (*e.g.* tamanho dos blocos);
- isso vale quanto na programação seqüencial como na programação distribuída;
- Conclusão: um bom algoritmo seqüencial é um bom algoritmo paralelo, e reciprocamente.
- “Pensar paralelo faz bem para a programação”

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

E a pesquisa nisso todo?

- a pesquisa em nível dos algoritmos é fundamental. . . e complexa;
- ela usa ferramentas matemáticas complicadas, além de programação técnica;
- já tem muita coisa feita (vide as bibliotecas que foram apresentadas!).
- Temos já soluções genéricas (*e.g.* a formulação em blocos) quase otimais. O desafio é na disponibilização de ambientes de programação (paralela) que permitem o uso simples de tais algoritmos e a obtenção de alto-desempenho, independente da máquina:
  - como descrever o paralelismo do algoritmo?
  - como mapear esta descrição (o programa) em qualquer tipo de máquina? (Escalonamento)

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

E o Grid?

- Atualmente a moda é no Grid...
- Nas plataformas existentes, apenas programas trivialmente paralelos são executados;
- os algoritmos apresentados não são trivialmente paralelos (vide as dependências) mas iterativos. No entanto, eles são fundamentais (vide TOP500!).
- Desafio (2): como executar tais programas em Grids?

Para ir mais a fundo...

Referências bibliográficas:

BARRETT, R. et al.

Templates for the solution of linear systems: building blocks for iterative methods, 2nd edition.
Philadelphia, PA: SIAM, 1994.

Netlib repository.

http://www.netlib.org (dez. 2004).

PRESS WILLIAM H., et. al.

Numerical recipes in C: the art of scientific computing.
Cambridge University Press, 1992.

DONGARRA, J. et al. (Eds.).

Templates and numerical linear algebra.
Morgan Kaufmann, 2000. p.575–620.

That's all, folks!

Et voilà !...
Perguntas?
http://www.inf.ufrgs.br/∼nmaillard/ERAD2005/

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL