Universidade de Coimbra - Faculdade de Ciências e Tecnologia
Departamento de Matemática

COMPUTAÇÃO PARALELA - 2005/2006
4º ANO

PROBLEMA 2 – *MINIMIZAÇÃO DE UMA FUNÇÃO 4D*

*** RELATÓRIO ***

Sara Catarina Morgado Menoita – ma0308@mat.uc.pt
Sara Joana Fino dos Santos Rodrigues de Carvalho – tc01037@mat.uc.pt
Sara Margarida Gaspar da Silva – tc01038@mat.uc.pt

Coimbra, 10 de Fevereiro 2005

"As únicas pessoas que nunca fracassam são as que nunca tentam. "
(Ilka Chase)

"Só quem sabe que não sabe procura saber."
(Sócrates)

Índice

1. Problema Proposto

Computação Paralela 2005/2006

Problema 2 – Minimização de uma função 4D

Data Limite de entrega: 23 Janeiro 2006

A escolha da *gauge* em Cromodinâmica Quântica envolve a minimização de uma função real de um número muito grande de variáveis. Sendo uma teoria relativistica, o domínio das funções é $\Re^4$. No tratamento numérico da teoria cada direcção é aproximada por um número discreto de pontos $0,1,...,L-1$. Tipicamente, utilizam-se o mesmo número de pontos em cada uma das direcções da rede hipercúbica.

No caso da *gauge* de Landau, a função a minimizar é dada por

$$F[X]=\sum_x \sum_\mu^3 X\left(\vec{x}\right)U_\mu\left(\vec{x}\right)X\left(\vec{x}+\hat{e}_\mu\right),$$

onde $X\left(\vec{x}\right)$ e $U_\mu\left(\vec{x}\right)$ são números reais definidos independentemente em cada ponto da rede $\vec{x}$. Os vectores $\hat{e}_\mu, \mu = 0,...,3$ são os versores que definem as direcções das arestas de um cubo 4D. A formulação da teoria exige condições de fronteira periódicas em todas as direcções, ou seja $X\left(L\hat{e}_\mu\right)=X\left(0\hat{e}_\mu\right)$.

Os números $U_\mu\left(\vec{x}\right)$ são calculados através de uma cadeia de Markov. Para o nosso problema de minimização podemos considerar

$$U_\mu\left(\vec{x}\right)=(-1)^{n_x+n_y+n_t+\mu},$$

onde $\vec{x}=n_x\hat{e}_0+n_y\hat{e}_1+n_z\hat{e}_2+n_t\hat{e}_3$. De entre os métodos de minimização utilizados, o mais comum é o método da descida máxima.

1. Mostre que $F[X+\delta X]\le F[X]$ se

$$\delta X\left(\vec{x}\right)=-\alpha\sum_\mu\left[U_\mu\left(\vec{x}\right)X(\vec{x}+\hat{e}_\mu)+U_\mu\left(\vec{x}-\hat{e}_\mu\right)X\left(\vec{x}-\hat{e}_\mu\right)\right]$$

em que α é um número suficientemente pequeno.

2. Determine empiricamente o melhor valor de α para a minimização de $F[X]$. Escreva um código sequencial para minimizar $F[X]$ utilizando o método da descida máxima com a variação acima descrita e teste o seu desempenho em função de α para uma rede hipercúbica em que $L = 10$. Defina o melhor valor do parâmetro α como o valor para o qual a minimização satisfaz o critério

$$\left\| \frac{\partial F}{\partial X} \right\| = \frac{1}{4L^4} \sum_{\vec{x},\mu} \left[\frac{\partial F}{\partial X(\vec{x})} \right]^2 \leq \varepsilon$$

onde

$$\frac{\partial F}{\partial X(\vec{x})} = \sum_{\mu=0}^{4} \left[U_\mu(\vec{x}) X(\vec{x} + \hat{e}_\mu) + U_\mu(\vec{x} - \hat{e}_\mu) X(\vec{x} - \hat{e}_\mu) \right].$$

Para a definição de α considere $\varepsilon = 10^{-8}$.

3. Discuta estratégias de paralelização deste problema de minimização. Escreva um código usando MPI que resolva, em vários processadores, o problema acima descrito. Determine o *speedup* do código para redes $L = 8,10,12,14,16$.

2. Introdução

O objectivo deste trabalho é a implementação de um algoritmo paralelo eficiente que minimize a função dada no enunciado.

De entre os métodos de minimização utilizados, o mais comum é o método da descida máxima o qual usamos no nosso problema.

3. Método da descida máxima

Os algoritmos de optimização são baseados na solução iterativa:

$$X^{k+1} = X^k + \alpha_k p^k,$$

ou seja,

- Determinar a direcção de procura p;
- Executar a procura unidireccional e obter α_k;
- Determinar quando o processo converge.

Método da descida máxima

A direcção de procura é dada pelo gradiente $\nabla F[X]$. Usando $\nabla F[X]$ limita-se a direcção de procura, evitando a procura em todo espaço. O gradiente deve ser recalculado a cada nova direcção, conforme representado na Figura 1. A sua importância é permitir estabelecer o ponto inicial para métodos mais sofisticados.

Toma-se como direcção de busca aquela oposta ao gradiente:

$$p^k = -\nabla F\left[X^k\right]$$

onde

$$X^{k+1} = X^k + \alpha_k p^k.$$

A cada passo, determina-se α_k usando procura unidireccional. A procura unidireccional consiste na resolução do seguinte problema de minimização

$$\min_{\alpha \phi 0} f\left(x^k + \alpha p^k\right)$$

Nestes métodos, a convergência é boa no início, mas muito lenta quando se aproxima do mínimo.

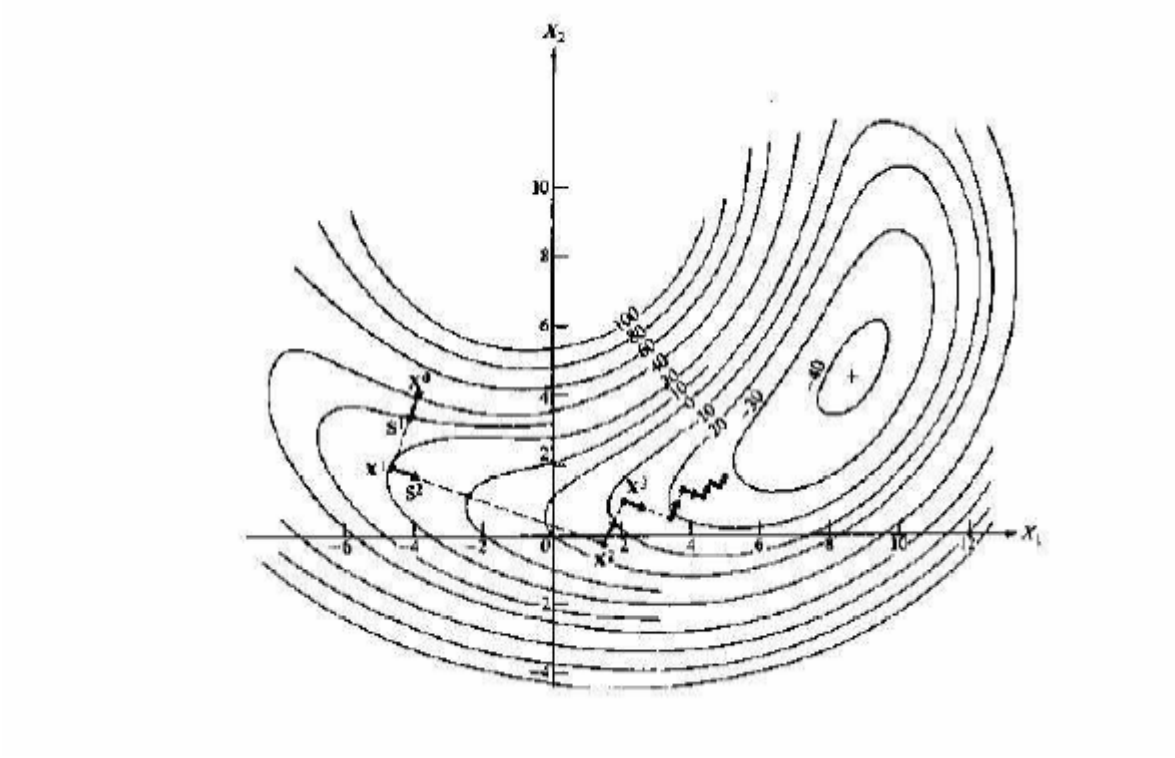

Figura 1: Representação do Método da Descida Máxima

3.1 Algoritmo

O algoritmo da descida máxima é muito simples. Considera-se um ponto inicial X_0 e usando procura unidireccional vamos calcular α_0 que é solução do problema seguinte:

$$\min_{\alpha>0} f\left(X^0 + \alpha p^0\right)$$

em que p_0 é dado por

$$p^0 = -\nabla F\left(X^0\right).$$

De seguida actualiza-se a solução, ou seja

$$X^1 = X^0 + \alpha_0 p^0 .$$

Se a solução anterior for a solução pretendida ou seja

$$\nabla F\left(X^1\right) = 0 ,$$

o algoritmo para. Também podemos considerar, como critério de paragem, a norma da diferença das duas soluções anteriores ou o número de iterações. Neste caso, o algoritmo para mais rápido dado que, por vezes o método da descida máxima é redundante.

Caso o algoritmo não pare, é efectuada uma nova iteração. Agora temos X^1 obtido da iteração anterior e calculamos o X^2. Primeiro é necessário calcular o α_1 por procura unidireccional, ou seja, resolver o problema de minimização seguinte

$$\min_{\alpha>0} f\left(X^1 + \alpha p^1\right) \text{ em que } p^1 = -\nabla F\left(X^1\right).$$

Em seguida actualizamos a solução, isto é,

$$X^2 = X^1 + \alpha_1 p^1 .$$

Verificamos se o algoritmo pára, caso isso não aconteça é efectuada uma nova iteração e assim sucessivamente até que o critério de paragem adoptado seja atingido. A solução obtida neste ponto é a solução óptima para o nosso problema.

4. Pergunta 1

PERGUNTA:

Mostre que $F[X+\delta X] \le F[X]$ se

$$\delta X(\vec{x}) = -\alpha \sum_{\mu} \left[U_{\mu}(\vec{x}) X(\vec{x}+\hat{e}_{\mu}) + U_{\mu}(\vec{x}-\hat{e}_{\mu}) X(\vec{x}-\hat{e}_{\mu}) \right]$$

em que α é um número suficientemente pequeno.

PROVA:

A prova será feita em $\Re^4$.

Consideremos $\mu = 0,1,2,3$.

Então temos a seguinte expressão para $F[X]$:

$$F[X] = \sum_{x} \sum_{i=0}^{3} X(\vec{x}) U_i(\vec{x}) X(\vec{x}+\hat{e}_i).$$

Fixemos $\vec{x} = k$. Temos então o seguinte desenvolvimento:

1º Passo: Desenvolvimento do segundo membro da desigualdade

$$F[X_k] = X_K U_0(k) X_{k+\hat{e}_0} + X_K U_1(k) X_{k+\hat{e}_1} + X_k U_2(k) X_{k+\hat{e}_2} + X_k U_3(k) X_{k+\hat{e}_3} + X_{k-\hat{e}_o} U_0(k-\hat{e}_0) X_k +$$
$$+ X_{k-\hat{e}_1} U_1(k-\hat{e}_1) X_k + X_{k-\hat{e}_2} U_2(k-\hat{e}_2) X_k + X_{k-\hat{e}_3} U_3(k-\hat{e}_3) X_k$$

$$= X_k \left[\sum_{i=0}^{3} \left[U_i(k) X_{k+\hat{e}_i} + X_{k-\hat{e}_i} U_i(k-\hat{e}_i) \right] \right]$$

2º Passo: Cálculo da derivada de $F[X]$

$$\frac{\partial F}{\partial X_k} = \frac{\partial}{\partial X_k} \sum_{k} \sum_{\mu=0}^{3} X_k U_{\mu}(k) X_{k+\hat{e}\mu}$$

$$= \frac{\partial}{\partial X_k}\left[\begin{array}{l}\left(\sum_{\mu=0}^{3} X_k U_\mu(k) X_{k+\hat{e}\mu}\right) + X_{k-\hat{e}_0} U_0(k-\hat{e}_0) X_k + X_{k-\hat{e}_1} U_1(k-\hat{e}_1) X_k + \\ + X_{k-\hat{e}_2} U_2(k-\hat{e}_2) X_k + X_{k-\hat{e}_3} U_3(k-\hat{e}_3) X_k\end{array}\right]$$

$$= \left(\sum_{\mu=0}^{3} U_\mu(k) X_{k+\hat{e}_\mu}\right) + X_{k-\hat{e}_0} U_0(k-\hat{e}_0) + X_{k-\hat{e}_1} U_1(k-\hat{e}_1) + X_{k-\hat{e}_2} U_2(k-\hat{e}_2) + X_{k-\hat{e}_3} U_3(k-\hat{e}_3)$$

$$= \sum_{\mu=0}^{3}\left[U_\mu(k) X_{k+\hat{e}_\mu} + U_\mu(k-\hat{e}_\mu) X_{k-\hat{e}_\mu}\right]$$

3º Passo: Desenvolvimento do primeiro membro da desigualdade

Seja $\mathrm{d} = \sum_{i=0}^{3}\left[U_i(k) X_{k+\hat{e}_i} + X_{k-\hat{e}_i} U_i(k-\hat{e}_i)\right]$

Fazendo uma expansão de Taylor com resto de grau 1 em torno de X vem

$$F(X+\delta X) = F(X) + \frac{\partial F}{\partial X}(X)\delta X + R_1(X, \delta X)$$

em que $R_1(X, \delta X)$ é o resto de primeira ordem e é tal que

$$\lim_{X \to \delta X} \frac{\left|R_1(X, \delta X)\right|}{\left\|X - \delta X\right\|} = 0 .$$

Assim, desprezando o resto vem que,

$$F(X+\delta X) = X_k d + d\delta X \qquad \text{(pelo 1º Passo)}$$

$$= X_k d + d(-\alpha d) \qquad \text{(pela definição de } \delta X \text{)}$$

$$= X_k d - \alpha d^2$$

4º Passo: Verificação da desigualdade

$F(X+\partial X) = X_k d - \alpha d^2 \leq X_k d = F(X)$ pois $\alpha \geq 0$ e $d^2 \geq 0$ logo $\alpha d^2 \geq 0$.

Deste modo verificámos a desigualdade $F(X+\delta X) \leq F(X)$.

5. Algoritmo Serial

5.1. Algoritmo

- Inicializar o MPI;

→ RANK=0 (visto que estamos a elaborar um programa serial)

- Pedir ao usuário do programa o valor de L e de niteracoes (número discreto de pontos em cada uma das direcções e número de iterações respectivamente);
- Preencher a matriz X em que cada elemento é o resultado da aplicação de uma função f;
- Inicializar alfa e alfafinal;
- Faz
  - Preencher a matriz X1, que resulta de um aumento da matriz X tendo em conta as condições de fronteira periódicas em todas as direcções;
  - Faz
    - Calcular FX (que representa $F[X]$);
    - Calcular dFX (que representa $\nabla F[X]$);
    - Calcular sigmaX1 (que representa $\delta X1$);
    - Calcular Xnovo ($Xnovo = X1 + \delta X1$);
    - Calcular FXnovo ou seja $F[Xnovo]$;
    - Calcular dFX em Xnovo:
    - Calcular a norma dFX;
    - Actualizar X1 ou seja $X1 = Xnovo$

    enquanto (norma> ε e conta<niteracoes);
  - alfa=alfa+0.01
- enquanto (alfa<= alfafinal e norma> ε);

→ RANK $\neq 0$

- Não faz nada;
- Finalizar o MPI

- Declaração da função $f(x,y,z,t)$;
- Declaração da função U (consideremos $U(x,y,z,t)=\cos\left((x+y+z+t)\frac{\pi}{10^4}\right)$);

Observação: No nosso projecto a declaração da função f foi feita logo no inicio do programa (declarámos a função constante igual a 1), de qualquer modo, achámos conveniente implementar uma função no final do programa para o caso de a querermos alterar. Todo o programa foi elaborado tendo em conta uma possível alteração da função f.

NOTA: O código do programa serial encontra-se em anexo com o nome QDCserialR4.c

5.2. Determinação do melhor valor de α

Após vários testes computacionais e muita persistência, admitindo L=10, concluímos que o melhor α que minimiza a função dada é 0,1. Para tal, alterámos várias vezes a função f, verificando que $f(x,y,z,t)=1$ nos traria melhores resultados e maior facilidade na procura do α óptimo.

6. Algoritmo Paralelo

6.1. Estratégias de paralelização

Devido ao facto de não conseguirmos visualizar o que acontece em $\Re^4$, vamos explicar a paralelização que fizemos em $\Re^3$, fazendo apenas no final uma breve alusão para $\Re^4$.

Para paralelizar o algoritmo serial consideramos uma rede cúbica com L×L×L pontos. Dividimos a rede de maneira a seccionar o cubo em fatias horizontais, formando assim vários paralelepípedos, numerando-os de 1 a nproc (número de processadores que vai trabalhar cada paralelepípedo individualmente, isto é, processador 1 trabalha o paralelepípedo 1, processador 2 trabalha o paralelepípedo 2 e assim sucessivamente) como se vê na imagem seguinte.

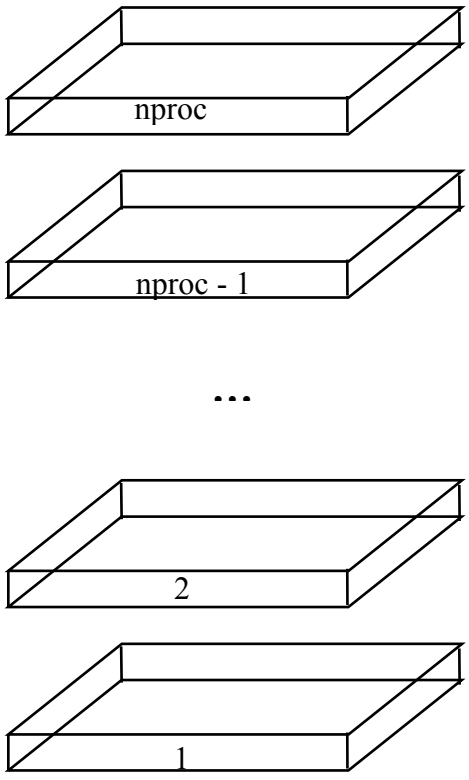

Figura 2

Para o cálculo da função num ponto interior de cada paralelepípedo é necessário conhecer os seus pontos vizinhos e nesse caso não temos qualquer problema. Surgem-nos problemas, quando

queremos calcular o valor da função em pontos particulares, pontos que pertencem à fronteira de cada paralelepípedo exceptuando os vértices (cujo valor da função nestes pontos é zero). Fixemos k.

- No caso dos pontos estarem na face superior do paralelepípedo k, excepto k = nproc, os pontos necessários para o cálculo da função são os pontos situados na face inferior do paralelepípedo k+1; sendo assim, temos de enviar os pontos da face inferior do paralelepípedo k+1 para o processo k. No caso de k = nproc necessitamos dos pontos da face inferior do paralelepípedo 1, pois temos condições fronteiras cíclicas, e portanto a face inferior do paralelepípedo 1 vai ser enviada para o processo nproc.

- No caso dos pontos estarem na face inferior do paralelepípedo k, excepto k = 1, os pontos necessários para o cálculo da função são os pontos situados na face superior do paralelepípedo k-1; sendo assim, temos de enviar os pontos da face superior do paralelepípedo k-1 para o processo k. No caso de k = 1 necessitamos dos pontos da face superior do paralelepípedo nproc, pois temos condições fronteiras cíclicas, e portanto a face superior do paralelepípedo nproc vai ser enviada para o processo 1.

Em $\Re^4$ passamos a ter quatro direcções $x, y, z, t.$ A rede passa a ser hipercúbica com L×L×L×L pontos. Dividimos a rede de maneira a seccionar o hipercubo em "fatias" segundo a direcção t, em que cada processador ficará responsável pelos cálculos necessários respeitantes a todos os pontos da sua "fatia" tendo em conta as condições de fronteira periódicas.

6.2. Algoritmo

- Inicializar o MPI:

→ RANK=0

- Pedir ao usuário do programa o valor de L e de niteracoes (número discreto de pontos em cada uma das direcções e número de iterações respectivamente);

→ TODOS OS RANK'S

- Todos os ranks vão fazer um broadcast de L;
- Todos os ranks vão fazer um broadcast de niteracoes;
- Calcular a medida da "altura", nt, do hipercubo que vai corresponder ao máximo atingido da variável t, para cada rank;
- Preencher a matriz X em que cada elemento é o resultado da aplicação de uma função constante 1;
- Inicializar alfa e alfafinal;
- Faz
    - Preencher a matriz X1, que resulta de um aumento da matriz X, sendo necessário efectuar a comunicação entre os processos, de modo a verificar as condições de fronteira periódicas em todas as direcções;
    - Faz
        - Calcular FX (que representa $F[X]$);
        - Calcular FXfinal;
        - Calcular dFX (que representa $\nabla F[X]$);
        - Calcular sigmaX1 (que representa $\delta X1$), sendo necessário efectuar a comunicação entre os processos, de modo a verificar as condições de fronteira periódicas em todas as direcções;
        - Calcular Xnovo ($Xnovo = X1 + \delta X1$);
        - Calcular FXnovo ou seja $F[Xnovo]$;
        - Calcular dFX em Xnovo:
        - Calcular a norma dFX;

- Actualizar X1 ou seja $X1 = Xnovo$

enquanto (norma> ε e conta<niteracoes);

- alfa=alfa+0.01

- enquanto (alfa<= alfafinal e norma> ε);

→ RANK = 0

- Escreve no ecrã os resultados

- Finalizar o MPI
- Declaração da função $f(x, y, z, t)$;
- Declaração da função U (consideremos $U(x, y, z, t) = \cos\left((x + y + z + t)\frac{\pi}{10^4} \right)$);
- Declaração do procedimento comunicacaodefatias (através de rotinas MPI).

Observação: No nosso projecto a declaração da função f foi feita logo no inicio do programa (declarámos a função constante igual a 1), de qualquer modo, achámos conveniente implementar uma função no final do programa para o caso de a querermos alterar. Todo o programa foi elaborado tendo em conta uma possível alteração da função f.

NOTA: O código do programa paralelo encontra-se em anexo com o nome QDCparaleloR4.c

7. Rotinas MPI utilizadas

Apresentamos de seguida as rotinas MPI utilizadas bem como as suas funções:

Inicialização do MPI:

- MPI_Init(&argc,&argv)
- MPI_Comm_size(MPI_COMM_WORLD,&nproc)
- MPI_Comm_rank(MPI_COMM_WORLD, &rank)

Sincronização de todos os processos:

- MPI_Barrier (MPI_Comm comm)

Terminar todos os processos:

- MPI_Abort(MPI_COMM_WORLD,erro)

Envio/recepção em simultâneo de/para vários processos:

- MPI_Bcast (void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Cálculo de FXfinal, FXnovofinal e normafinal:

- MPI_Allreduce (void* operand, void* result, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Comunicação entre "fatias":

- MPI_Send (void *sndbuf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm)
- MPI_Recv (void *recvbuf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_Status status)

Medição de tempos:

- MPI_Wtime(void)
- MPI_Wtick(void)

Finlização do MPI:

- MPI_Finalize()

8. Análise da Performance

Este capítulo é reservado à análise de dados que recolhemos através de testes efectuados nos *tatus* do laboratório de cálculo do Departamento de Matemática da Universidade de Coimbra.

Executámos várias vezes os programas serial e paralelo de onde obtivemos diversos resultados de tempos com grandes oscilações para os mesmos testes; no entanto achámos normal, pois cada vez que nos ligávamos em rede trabalhávamos com tatus diferentes e para além disso os tatus estavam a ser utilizados por mais alunos e pelo próprio sistema o que condicionava os tempos de execução dos programas. Após constatarmos estes factos optámos por realizar os testes de forma sequencial para manter as mesmas condições em todos os testes.

8.1. Speedup

O principal objectivo na utilização da computação paralela é a diminuição do tempo de execução dos processos envolvidos. Diferentes métricas podem ser utilizadas para verificar se a utilização do processamento paralelo é vantajosa e quantificar o desempenho alcançado. O *Speedup* é uma das métricas mais utilizadas para atingir esse objectivo.

$$S_P = \frac{T_1}{Tp}$$

onde S_P = speedup observado num computador com P processos;

T_1 = tempo de execução do programa serial;

T_P = tempo de execução do programa paralelo correspondente com P processos.

Apresentamos de seguida os gráficos do speedup correspondentes a redes L=8,10,12,14,16.

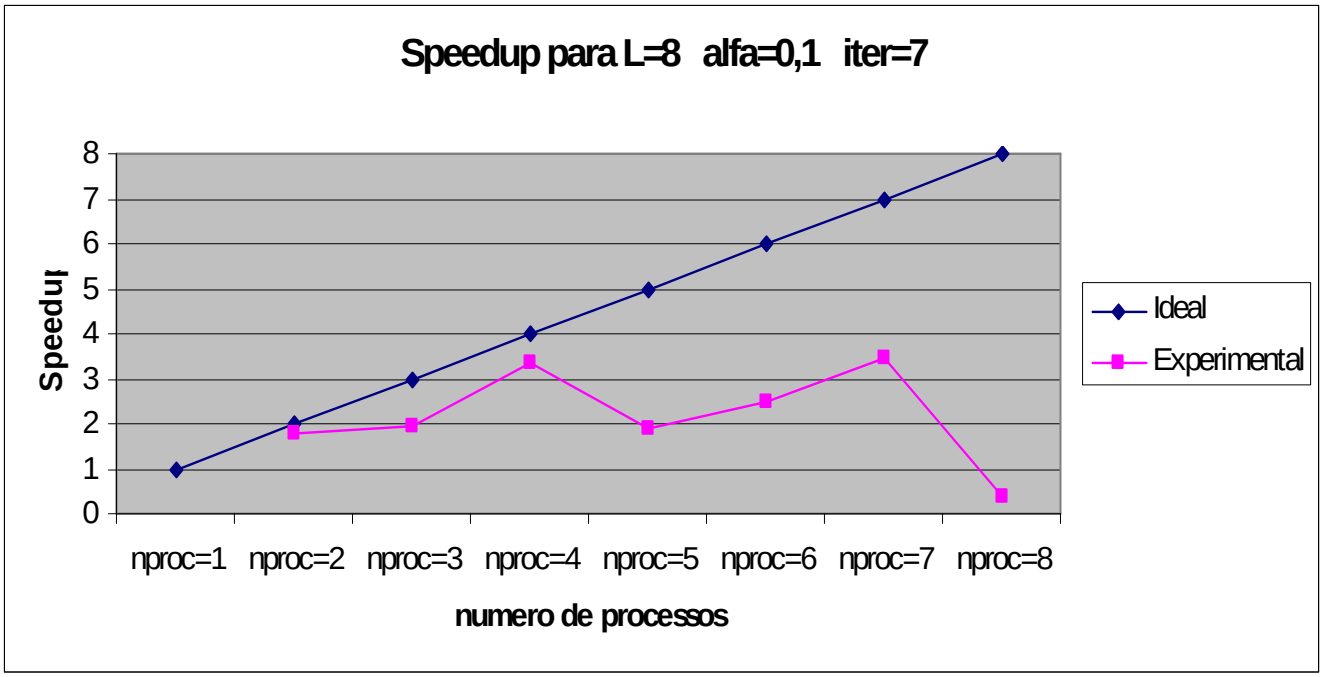

Figura 3

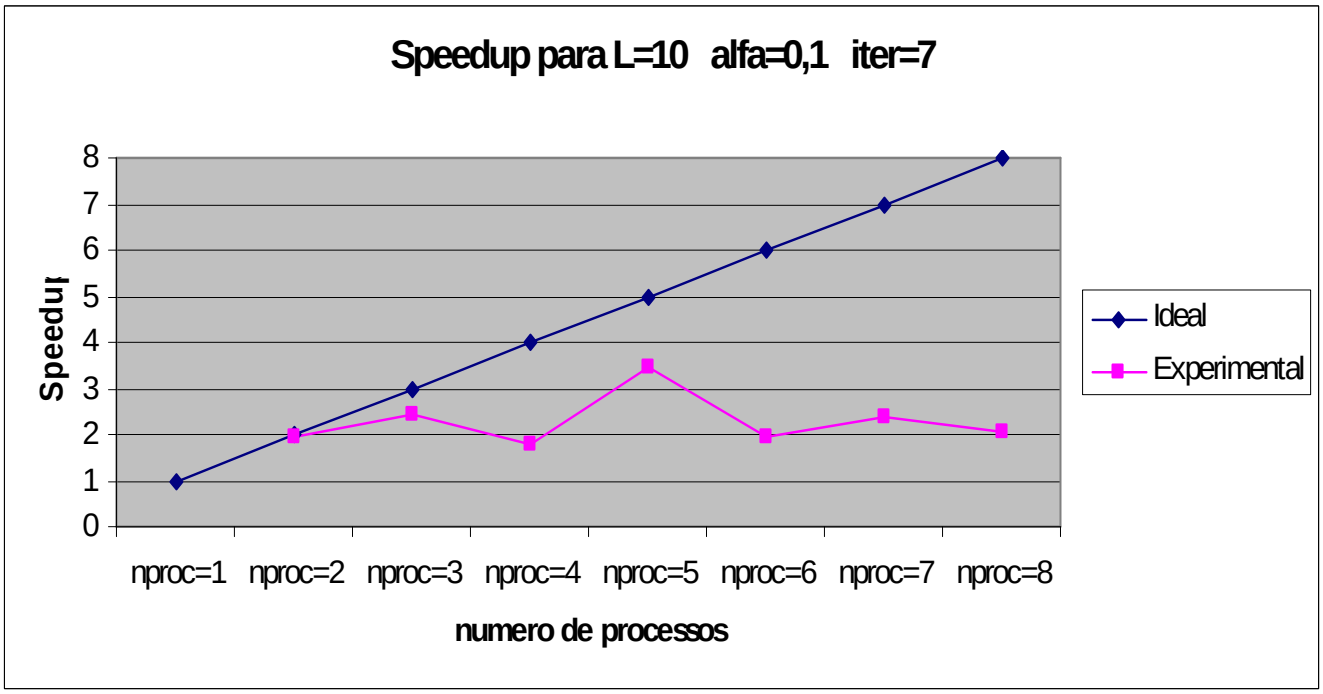

Figura 4

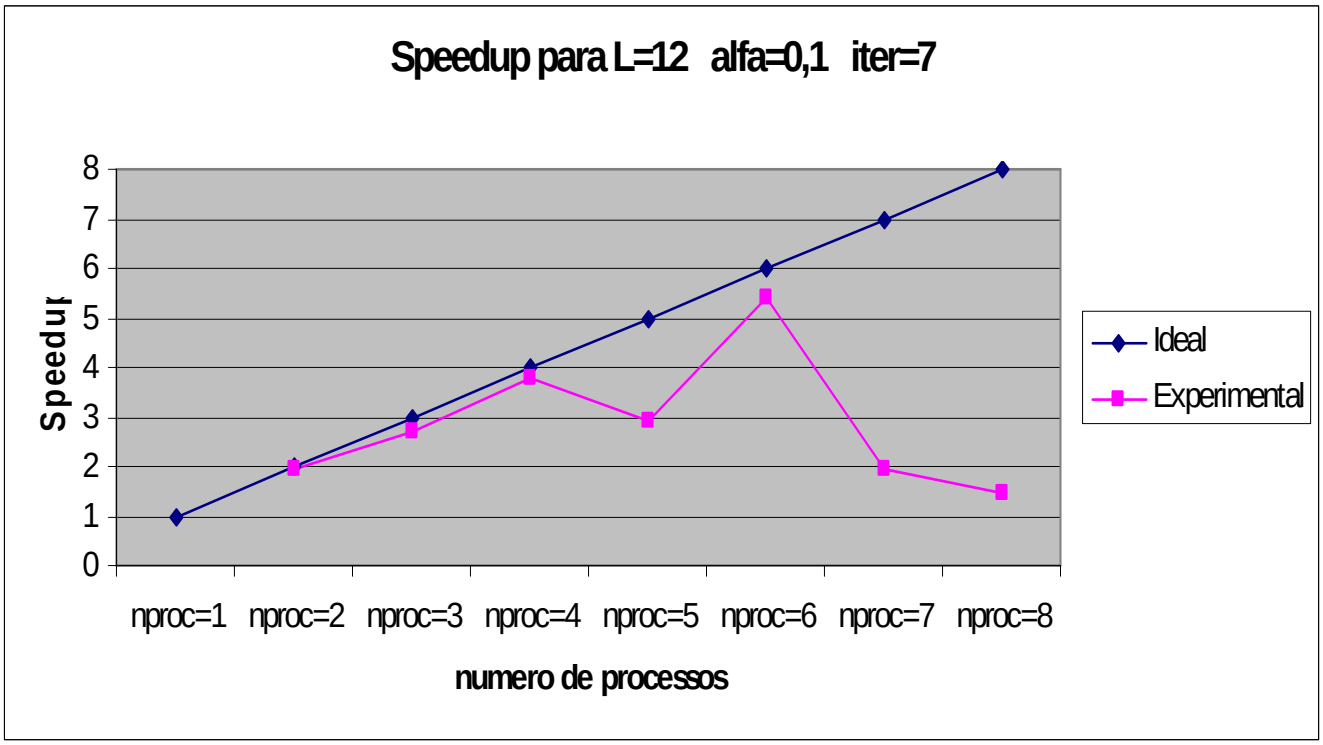

Figura 5

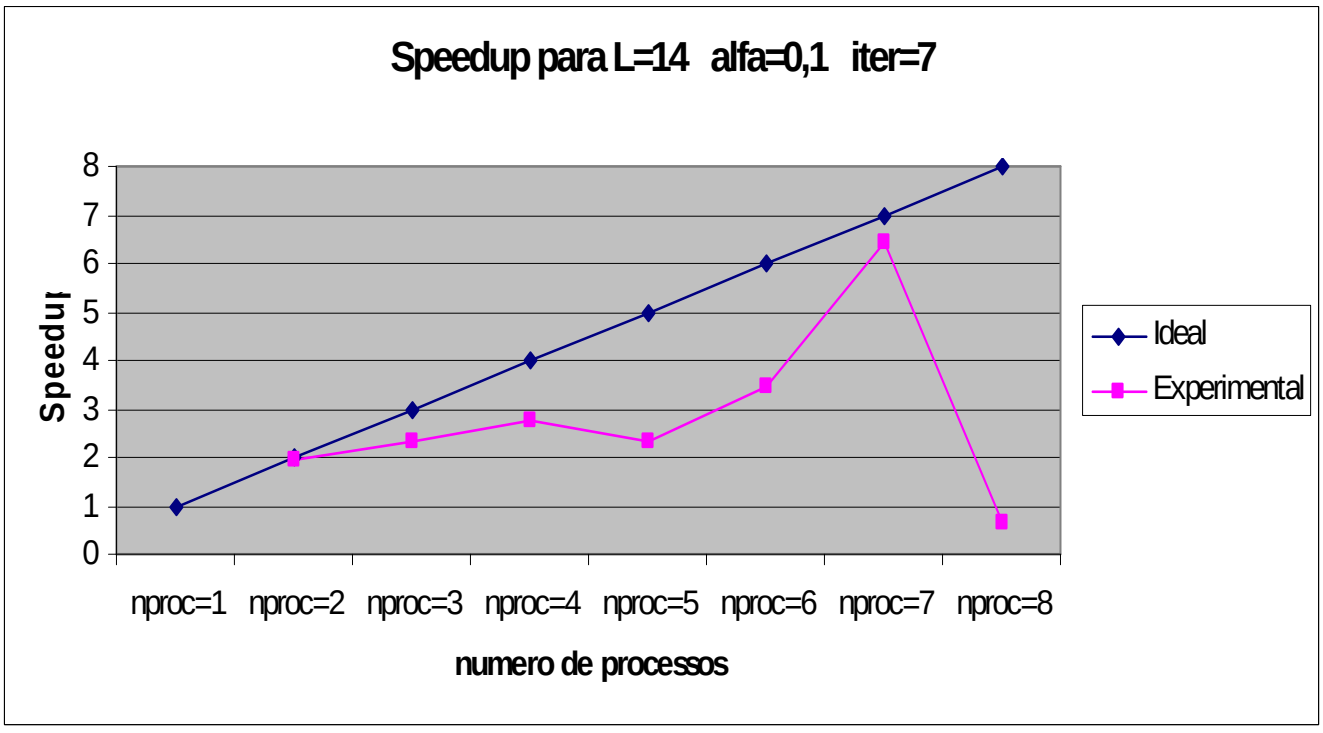

Figura 6

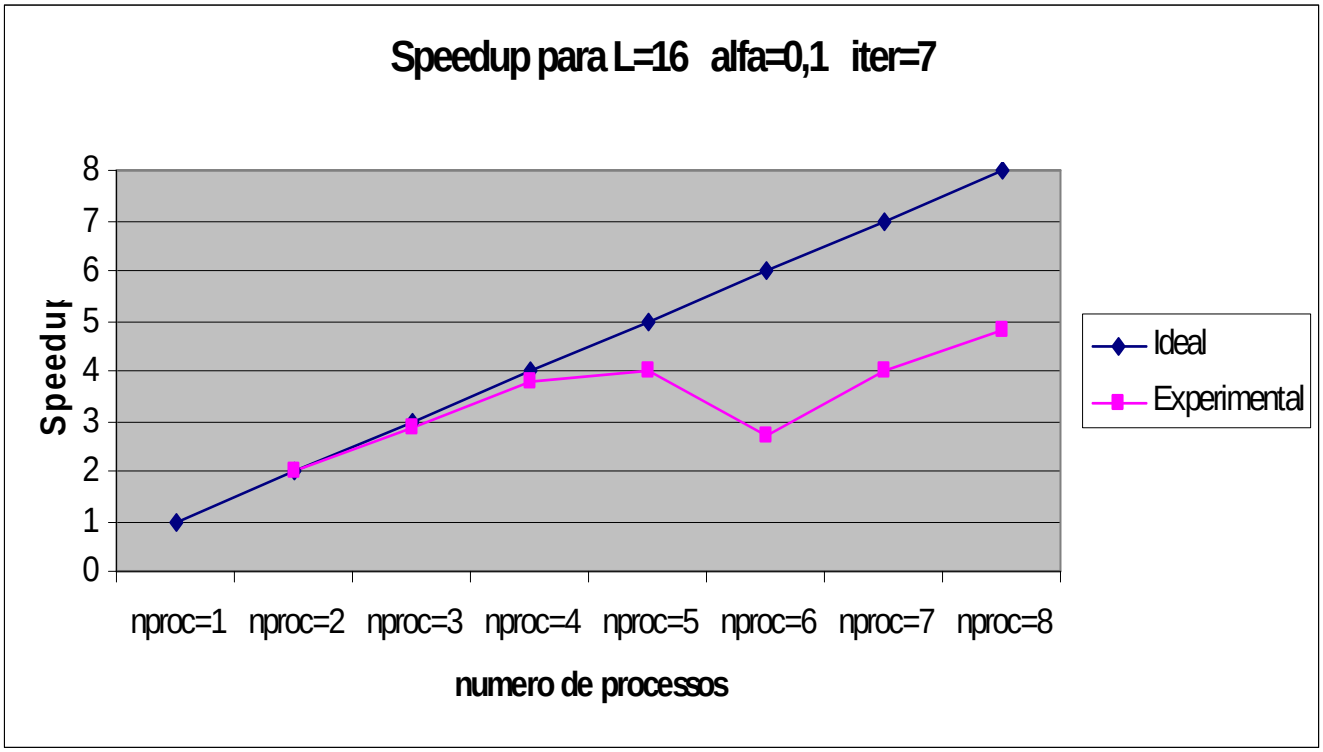
Speedup para L=16 alfa=0,1 iter=7
Speedup
0
1
2
3
4
5
6
7
8
nproc=1
nproc=2
nproc=3
nproc=4
nproc=5
nproc=6
nproc=7
nproc=8
numero de processos
Ideal
Experimental

Figura 7

De seguida apresentamos a sobreposição de todos os gráficos anteriores para uma melhor análise.

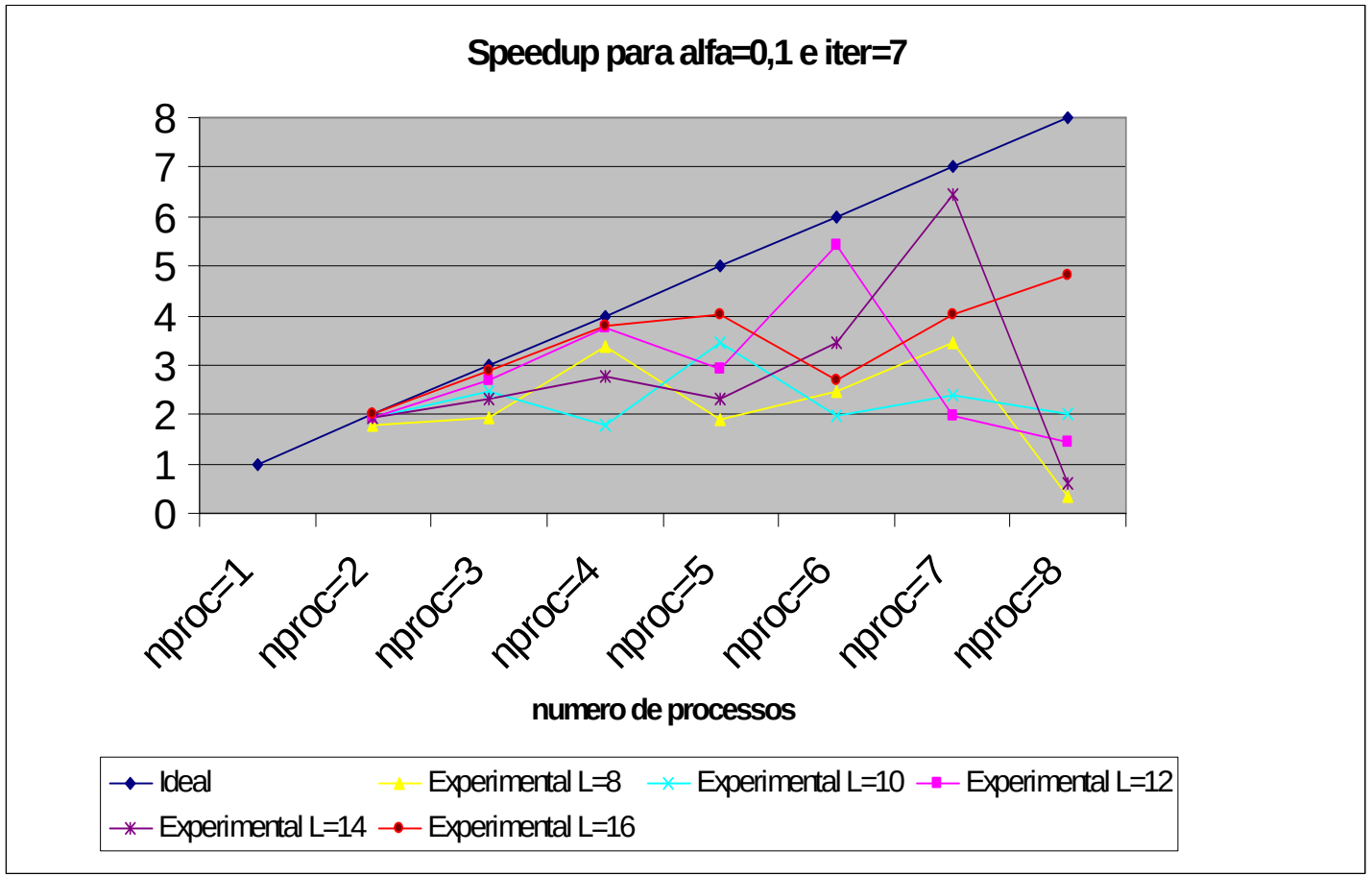

Figura 8

Admitimos para todos os casos que $\alpha = 0,1$ (óptimo no caso L=10) e iter=7 (visto que o programa serial satisfaz as condições de paragem ao fim de 7 iterações).

Devido ao facto do programa paralelo abortar quando o número de processadores for 1 (uma vez que a comunicação entre processos não é possível pois apenas existe 1 processo), os gráficos apenas apresentam resultados a partir de 2 processadores.

Como podemos observar, para todos os L testados, quando temos $1<nproc\leq 4$ o speedup está mais próximo de ser ideal.

Para L=16, e nproc ≤ 4 obtemos o speedup (quase) ideal.

Para L= 8 obtemos o pior speedup.

Há medida que nproc aumenta, a partir de 4, obtemos picos no gráfico de speedup continuando a ser o L=16 o que nos dá melhores resultados.

9. Limitações do programa

Devido ao facto de estarmos a trabalhar a quatro dimensões, considerámos MAX=16 pois para valores superiores, o programa não corre.

10. Conclusões

Concluímos que o uso do programa paralelo traz vantagens relativamente ao uso do programa serial, pois tem tempos de execução inferiores.

Concluímos por outro lado, que as normas obtidas no serial e no paralelo são diferentes (com um erro muito pequeno) e portanto devemos ter algum erro que não detectámos.

11. Bibliografia

- Páginas da web:
  http://www.di.ubi.pt/~crocker/paralel/docs/cursompi.pdf
  http://www-unix.mcs.anl.gov/mpi/
  http://paginas.fe.up.pt/~jbarbosa/ensino/PDP/2004-2005/acetatos/a2_MPI.pdf
  http://www.llnl.gov/computing/tutorials/mpi/

ANEXOS

PROJECTO REALIZADO POR:

(Sara Catarina Morgado Menoita)

(Sara Joana Fino dos Santos Rodrigues de Carvalho)

(Sara Margarida Gaspar da Silva)