

PROCESSAMENTO VETORIAL E  
VETORIZAÇÃO DE ALGORITMOS  
NA MÁQUINA CONVEX C210

por

Tiaraju Asmuz Divério

RP-160

Agosto/1991

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO  
Av. Osvaldo Aranha, 99  
90210 - Porto Alegre - RS - BRASIL  
Telefone: (0512) 281633  
Telex: (051) 2680 - CCUF BR  
FAX: (0512) 244164  
E-MAIL: PGCC@sbu.ufrgs.anrs.br

Correspondência: UFRGS-CPGCC  
Caixa Postal 1501  
90001 - Porto Alegre - RS - BRASIL



UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA

Resumo

Este trabalho descreve técnicas para reestruturação de programas sequenciais utilizadas para a vetorização e paralelização destes programas. São descritos conceitos básicos sobre processamento paralelo e aplicações às ciências e engenharia, especialmente na solução de sistemas de equações lineares. São analisados exemplos de programas de resolução de sistemas de equações lineares executados no Convex C210.

Palavras chave: Vetorização de algoritmos, Convex C210, Processamento vetorial

Abstract

This report describes some techniques for sequential programs reestructuration used to vectorization of this programs. Some basic concepts about parallel processing and its applications, specially for solution of linear systems are described. Examples of programs for solution of linear systems in the Convex C210 are described.

Key words: Vectorial Processing, Convex C210, Algorithms vectorization.

*Matemática computacional*  
SBV

*Análise numérica*  
*Processamento paralelo*  
*Processamento vetorial*

*ENPg 1.01.04.00-3*

UFRGS INSTITUTO DE INFORMÁTICA BIBLIOTECA		
N.º CHAMADA FL 2145		N.º REG: 36132
ORIGEM: D		DATA: 24, 2, 92
DATA: 16, 9, 91		PREÇO R\$10.000,00
FUNDO: EPGCC	FURN.: EPGCC	



## SUMÁRIO

LISTA DE FIGURAS .....	4
1 INTRODUÇÃO .....	5
2 PROCESSAMENTO PARALELO E VETORIZAÇÃO .....	6
2.1 Conceitos básicos e níveis de paralelismo .....	6
2.2 Máquinas paralelas .....	9
2.2.1 Máquina Pipeline .....	9
2.2.2 Processadores Matriciais .....	11
2.2.3 Multiprocessadores .....	12
2.3 Níveis de paralelismos .....	12
2.4 Técnicas de reestruturação .....	14
2.4.1 Vetorização .....	16
2.4.2 Vetorização parcial .....	17
2.4.3 Paralelização .....	18
2.4.4 Distribuição de loops .....	19
2.4.5 Mudanças internas de loops .....	20
2.5 Processamento vetorial .....	20
3 APLICAÇÕES DE PROCESSAMENTO PARALELO .....	24
3.1 Modelação preditiva e simulação .....	25
3.1.1 Previsão numérica atmosférica .....	25
3.1.2 Oceanografia e astrofísica .....	25
3.1.3 Uso socioeconômico e governamental .....	26
3.2 Engenharia de Projeto e automação .....	26
3.2.1 Análise de elementos finitos .....	26
3.2.2 Aerodinâmica computacional .....	27
3.2.3 Inteligência artificial e automação .....	27
3.2.4 Aplicações em sensoriamento remoto .....	28
3.3 Exploração de recursos energéticos .....	28
3.3.1 Exploração sísmica .....	28
3.3.2 Modelagem de reservatórios .....	29
3.3.3 Poder de fusão do plasma .....	29
3.3.4 Segurança de reatores nucleares .....	30
3.4 Pesquisa Básica, militar e médica .....	30
3.4.1 Tomografia computadorizada .....	30
3.4.2 Engenharia genética .....	31
3.4.3 Pesquisa de armamento e defesa .....	31
3.4.4 Problemas da Pesquisa básica .....	32
3.5 Engenharia Mecânica .....	32
4 CONSIDERAÇÕES SOBRE ALGORITMOS NUMERICOS PARALELOS .....	34
4.1 Solução de relações de recorrência .....	34
4.2 Solução de equações diferenciais .....	37
4.3 Solução de sistemas de equações lineares .....	40

5	SOLUÇÃO DE SISTEMAS EQUAÇÕES IMPLEMENTADOS NO CONVEX .....	43
5.1	Métodos Diretos .....	43
5.1.1	Método de Eliminação de Gauss com pivotamento..	43
5.1.2	Método de Gauss-jordan com Inversa .....	48
5.2	Métodos Compactos .....	52
5.2.1	Método de Banachievicz .....	52
5.2.2	Método de Cholesky .....	55
5.3	Métodos Iterativos .....	59
5.3.1	Método de Gauss-Jacobi .....	59
5.3.2	Método de Gauss-Seidel .....	62
6	RESULTADOS E CONCLUSÕES .....	66
ANEXOS	.....	72
1	Execução do teste dos métodos compactos .....	73
2	Execução do teste dos métodos diretos .....	75
3	Execução do teste dos métodos iterativos .....	81
4	Programa MDENS50.f (Fortran 77 Convex) .....	83
5	Listagem da otimização e vetorização .....	88
6	Programa MESPARGA.f (Fortran 77 Convex) .....	91
7	Listagem da otimização e vetorização .....	93
8	Programa MDENSA.f (Fortran 77 Convex) .....	95
BIBLIOGRAFIA	.....	98



## LISTA DE FIGURAS

Figura 2.1	Arquitetura de máquina SISD .....	7
Figura 2.2	Arquitetura de máquina SIMD .....	8
Figura 2.3	Arquitetura de máquina MISD .....	8
Figura 2.4	Arquitetura de máquina MIMD .....	9
Figura 2.5	Pipeline de adição de ponto-flutuante .....	10
Figura 2.6	Quadro comparativo das arquiteturas .....	12
Figura 2.7	Níveis X Tipo X Planejador do paralelismo .....	14
Figura 2.8	Operações vetoriais .....	21
Figura 3.1	Interação experimental, teórico e computacional.	24
Figura 4.1	Algoritmo da soma parcial por cascata e execução	35
Figura 4.2	Algoritmo paralelo Redução cíclica .....	37
Figura 4.3	Problema do estado cte de distribuição de temp.	38
Figura 6.1	Quadro comparativo de Método de Banachiewicz ...	67
Figura 6.2	Quadro comparativo de Método de Cholesky .....	67
Figura 6.3	Quadro comparativo de Método de Elim de Gauss ..	68
Figura 6.4	Quadro comparativo de Método de Gauss-Jordan ...	69
Figura 6.5	Quadro comparativo de Método de Gauss-Jacobi ...	70
Figura 6.6	Quadro comparativo de Método de Gauss-Seidel ...	70

## 1 INTRODUÇÃO

Este documento visa relatar o estágio do Prof. Tiaraju Asmuz Diverio na Universidade Federal de Santa Catarina (em Florianópolis), no Departamento de Ciências Estatísticas e da Computação (DCEC) e no Núcleo de Processamento de Dados (NPD).

O estágio foi financiado pelo CNPq dentro do programa de Formação de Recursos Humanos em Áreas Estratégicas (RHAÉ), cujo projeto foi intitulado: **Aperfeiçoamento em processamento vetorial, uso da máquina Convex C210 e intercâmbio Cultural**, (processo Nro 46.0139/91-6 RHAÉ INFO) e que se realizou nos meses de abril e maio de 1991.

Além deste documento foi produzido a monografia intitulada "Máquina Convex C210 primeiros passos: utilização e programação", a qual inicia com uma caracterização da UFSC e dos recursos computacionais disponíveis e tem uma descrição do sistema Convex C210, do sistema operacional ConvexOS/Unix. É apresentado um roteiro para se ter acesso ao computador (login), são enumerados alguns dos comandos do sistema operacional para auxiliar usuários iniciantes dar os primeiros passos com sistema operacional. É descrito a filosofia dos compiladores disponíveis no Convex e suas potencialidades de vetorização. É feita uma descrição do editor de texto do Unix, denominado Vi, que possibilita a criação e edição programas e textos. É dado, ainda, um roteiro para edição, compilação, acertos de erros e execução de programas em Fortran. E por fim, é descrito a biblioteca de rotinas vetoriais e a forma de utilização de suas rotinas.

Este relatório inicia com uma revisão sobre máquina paralelas; sobre conceitos básicos de paralelização e vetorização; são caracterizados os níveis de paralelismo. É descrito a metodologia adotada pelos compiladores disponíveis no Convex para tentar garantir um melhor desempenho na vetorização e paralelização de loops e, por fim, são apresentados alguns requerimentos do processamento vetorial, mais especificamente dentro do ambiente do supercomputador Convex.

No terceiro capítulo são apresentadas algumas aplicações do processamento paralelo em diferentes áreas e aplicações práticas que tem sido desenvolvidas pelo Departamento de Engenharia Mecânica da UFSC.

Por fim, são apresentados alguns testes desenvolvidos em Fortran no Convex, de onde se procurou tirar algumas conclusões da potencialidade e aplicabilidade do processamento vetorial e do supercomputador Convex na matemática computacional.



## 2 - PROCESSAMENTO PARALELO E VETORIZAÇÃO

### 2.1 Conceitos básicos de paralelismo

A busca de computadores com desempenho superiores aos existentes em cada momento e o aumento do poder computacional, tem sido atingido pela exploração de aspectos básicos, como tecnologia, algoritmos e arquitetura. O aperfeiçoamento tecnológico refere-se à pesquisa para confecção de componentes eletrônicos mais velozes. Entretanto, existem limitações que se devem a fatores como velocidade máxima de propagação de sinais e chaveamento de transistores.

Quanto ao aspecto do desenvolvimento de novos algoritmos, tem sido um processo lento, devido especialmente ao fato que muitos dos algoritmos já estão próximo aos seus limites teóricos de eficiência.

O aspecto da evolução das arquiteturas dos computadores, em particular com o uso de processamento paralelo, permite superar os limites de desempenho impostos pelos fatores tecnológicos de seus componentes.

O emprego de algoritmos e arquiteturas paralelas tem se constituído em uma alternativa promissora para superar as barreiras de desempenho impostas pelo estágio da tecnologia de componentes eletrônicos.

Uma arquitetura paralela geralmente é um sistema de propósito específico, otimizado para realizar de forma eficiente uma certa gama de funções dentro de uma aplicação.

**Processamento paralelo** é uma forma eficiente de processar informações na qual enfatiza a exploração de eventos concorrente na computação do processo. Concorrência implica paralelismo, simultaneidade e pipeline.

No Paralelismo os eventos paralelos ocorrem em múltiplos recursos no mesmo instante de tempo. É o paralelismo de recursos assíncronos.

Na Simultaneidade os eventos são simultâneos, no mesmo instante de tempo. É o paralelismo de recursos síncronos.

No Pipeline os eventos pipelines ocorrem em instantes sobrepostos. É o paralelismo temporal.

Em função do tipo de paralelismo empregado, os computadores paralelos são divididos em três grupos: máquinas pipelines, processadores matriciais e multiprocessadores.

As máquinas Pipeline superpõem a execução de instruções explorando um paralelismo denominado temporal, onde os eventos ocorrem em instantes de tempos sobrepostos.



Processadores matriciais empregam a multiplicidade de unidades processadoras executando instruções sincronamente utilizando o paralelismo dito espacial síncrono, onde os eventos são simultâneos, pois ocorrem em múltiplos recursos no mesmo instante de tempo.

Por último, temos o paralelismo espacial assíncrono, que é empregado em sistemas multiprocessadores através da divisão de recursos comuns (memória, periféricos, etc...) por um conjunto de processadores, onde os eventos ocorrem em múltiplos recursos no mesmo instante de tempo.

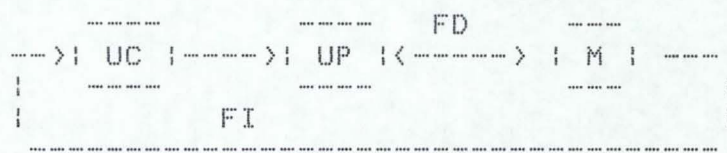
As estruturas que exploram o paralelismo são vinculadas à máquina em que a aplicação será executada. Essa característica implica em que o programador deve ter o conhecimento da arquitetura da máquina empregada e levar isso em consideração quando da escrita do algoritmo ou programa. Isso se constitui em uma dificuldade adicional na elaboração de programas para máquinas paralelas.

Estes eventos concorrentes são atribuídos em sistemas de computação em vários níveis de processamento. Processamento paralelo demanda execução de vários programas num computador. Isto é um contraste com processamento sequencial. Isto é um dos significados do custo-efetivo para prover a performance do sistema através de atividades concorrentes em um computador.

M. Flynn ([FLY72]) propôs a classificação que leva em conta a forma pela qual é executada uma instrução em um conjunto de dados, ou seja:

- SISD - Single Instruction stream Single Data stream
- SIMD - Single Instruction stream Multiple Data stream
- MISD - Multiple Instruction stream Single Data stream
- MIMD - Multiple Instruction stream Multiple Data stream

A categoria SISD tem fluxo de instrução e de dados únicos. São máquinas baseadas nos princípios de Von Neumann. As instruções são executadas sequencialmente, mas podem ser sobrepostas nos seus estágios de execução. Exemplos são arquiteturas pipelines. Este tipo de arquitetura que proporcionou o surgimento de máquinas de alto desempenho chamadas de supercomputadores.



- UC Unidade de controle
- M Memória
- FI Fluxo de Instruções (único)
- UP Unidade de Processamento
- FD Fluxo de Dados (único)

Fig. 2.1 Arquitetura de máquinas SISD



A categoria **SIMD** tem um único fluxo de instruções com vários fluxos de dados. Nesse tipo de arquitetura vários elementos de processamento (EP) são supervisionados por uma única unidade de controle que encaminhará a mesma instrução para execução nos elementos sobre seus dados. Nestas máquinas, portanto, uma única operação é executada simultaneamente por diversos elementos de processadores sobre dados distintos. A memória pode ser dividida em módulos vinculados a cada elemento de processamento ou então ser de acesso global. Exemplos são arquiteturas matriciais.

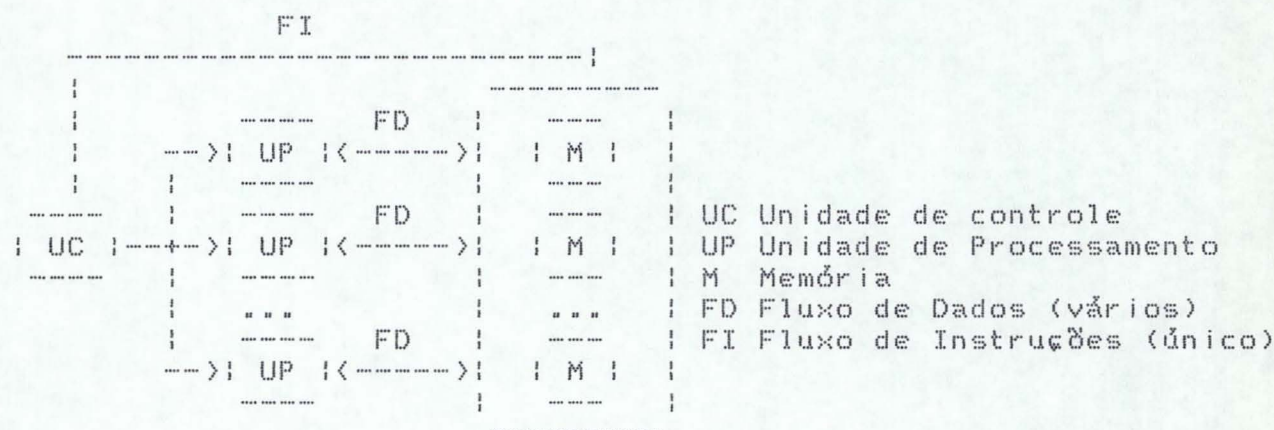


Fig. 2.2 Arquitetura de máquinas SIMD

A categoria **MISD** caracteriza-se por ter múltiplos fluxos de instruções e um único fluxo de dados. Este tipo de arquitetura não existe na prática, mas corresponderia a uma máquina que possuísse vários elementos de processamento recebendo instruções distintas de várias unidades de controle, que processariam o mesmo fluxo de dados.

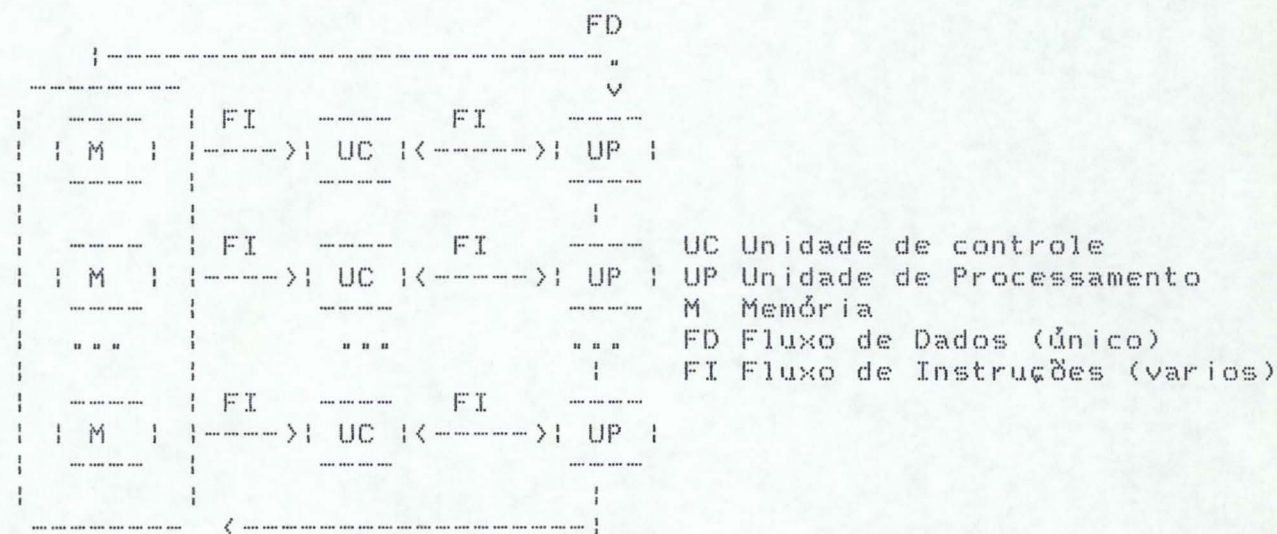
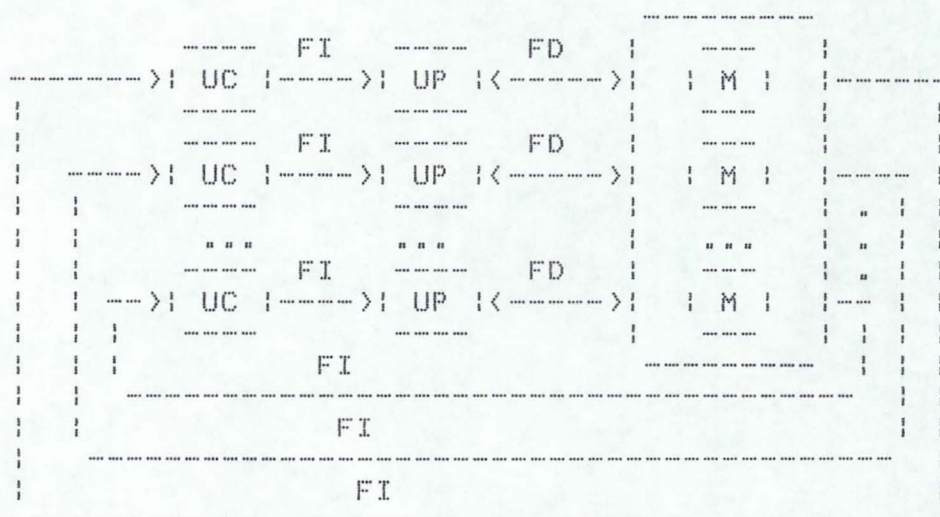


Fig. 2.3 Arquitetura de máquina MISD

A categoria MIMD caracteriza-se por ter múltiplos fluxos de instruções com múltiplos fluxos de dados. Nessa arquitetura cada unidade de controle possui sua unidade de processamento executando instruções sobre um conjunto de dados de forma independente. São as máquinas capazes de executar simultaneamente diversas instruções, sobre dados distintos. A maioria dos sistemas multiprocessadores são deste tipo. A interação entre os processadores é obtida através da memória global ou sistema de memórias gerenciado por um único sistema operacional.



UC Unidade de controle  
 UP Unidade de Processamento (Elemento de Processamento)  
 M Memória  
 FD Fluxo de Dados (varios)  
 FI Fluxo de Instruções (varios)

Fig. 2.4 Arquitetura de máquinas MIMD

## 2.2 Máquinas paralelas

Computadores paralelos ou máquina paralelas são todos os sistemas que enfatizam o processamento paralelo. A estrutura básica de computadores paralelos é introduzida a seguir. Divide-se computadores paralelos em três configurações de arquitetura: Máquinas Pipeline, processadores matriciais e Multiprocessadores.

### 2.2.1 Máquinas Pipeline

É um exemplo de paralelismo temporal, onde uma tarefa é subdividida numa sequência de subtarefas de modo que seja executada mais fácil e mais rapidamente, pois cada uma é executada por um estágio de hardware específico, que trabalha concorrentemente com os outros estágios do pipeline, criando desta forma, um paralelismo temporal na execução das subtarefas. Um exemplo clássico é o de uma fábrica de automóveis, onde estes



são feitos em série ou por uma linha de produção, onde cada grupo especializado realiza uma subtarefa, de forma que um carro demore o mesmo tempo de fabricação, mas assim que o primeiro for produzido, a cada período  $t$  de tempo, teremos outro carro produzido. Exemplos de máquinas: CRAY 1 e 2, CYBER 200, NETS, FUJITSU (VP 200 e VP 400).

A principal abordagem para ganhar velocidade em computadores vetoriais é Pipelining. O princípio do pipeline tem sido aplicado para acessar memórias e decodificar instruções desde os anos 60. Unidades aritméticas pipeline foram primeiramente incluídas em máquinas como o sistema 360 da IBM. Entretanto, estas máquinas são máquinas escalares porque suas instruções aritméticas são executadas com apenas um par de operandos. O primeiro computador comercial vetorial foi o Texas Instruments Inc, ASC (em 1972), onde as instruções de hardware aceitam vetores como operandos.

Um pipeline típico para soma em ponto-flutuante é composto por quatro segmentos: compara expoentes, alinha frações, soma de frações e normaliza, como é ilustrado na figura 2.5.

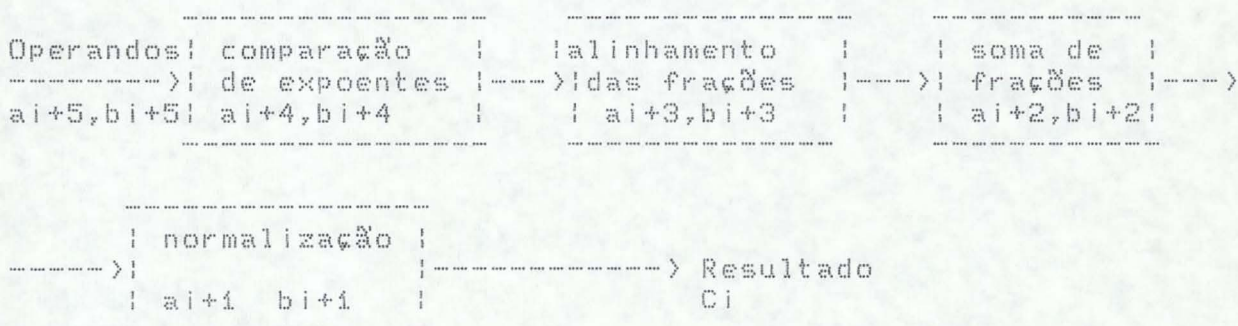


Fig. 2.5 Pipeline de adição de ponto-flutuante

Quando uma instrução vetorial é editada, é inicializado a corrente de operandos do pipeline, cada segmento do pipeline aceita um par de operandos, realiza sua função particular e passa o resultado para o próximo segmento e recebe o novo par de operando do fluxo. A cada instante durante a execução, vários pares de operandos são processados concorrentemente no pipeline. O efeito de cadeia proporciona um atraso inicial para completar o primeiro resultado, o qual é chamado de tempo de startup, mas cada resultado subsequente segue rapidamente uma vez que só falta um segmento.

Outra abordagem para ganhar velocidade é a técnica chamada de **encadimento** (chaining), na qual múltiplos pipeline são unidos juntos e operam como um único pipeline longo. Chaining é oferecido no IBM 3090 Vector Facility na forma de instruções compostas, instruções multiplicação-adição e multiplicação-acúmulo. Além de capacitar o pleno uso de pipelines, instruções vetoriais também reduzem substancialmente a necessidade de executar ramos de instruções.



A velocidade de computadores escalares é usualmente medida pelo número de instruções executadas por unidade de tempo, assim como "milhões de instruções por segundo" (MIPS). Para processadores vetoriais, os quais são principalmente utilizados para computações científicas, isto é universalmente aceito para medir a velocidade pelo o número de operações aritméticas calculadas por unidade de tempo, assim como o uso de milhão de operações em ponto-flutuante por segundo (MEGAFLOPS ou MFLOPS).

### 2.2.2 Processadores Matriciais

Também denominados de array processors ou SIMD, são exemplos de paralelismos espacial síncrono.

Um processador matricial é um conjunto paralelo síncrono com múltiplas unidades lógicas e aritméticas (ULA), denominados de elementos processadores (PE), que podem operar em paralelo de forma síncrona realizando uma mesma função em um mesmo instante. Uma unidade de controle distribui para vários elementos de processamento, a mesma instrução que será executada em paralelo por esses EPs sobre seus dados no mesmo instante de tempo. Um exemplo, é a operação com matrizes, por exemplo multiplicar a matriz por dois. Isto implica em nove multiplicações (matriz quadrada de ordem três), mas se são feitos pelos elementos de processamento, levaremos apenas um tempo. Exemplos de máquinas são: Illiac IV, MPP, Pepe, Staran.

Os processadores matriciais utilizam o endereço do dado para acesso, existe uma variação de processadores, os quais acessam os dados via conteúdo; esses exploram a capacidade de memória associativas e, por isso, são denominados **matriciais associativos**.

Os processadores matriciais são compostos por uma unidade de controle, um conjunto de elementos processadores, uma memória associada a cada elemento processador e uma rede de interconexão. A unidade de controle é responsável pelo gerenciamento da operação dos elementos processadores decodificando as instruções a serem executadas e gerenciando os sinais de controle.

Alguns exemplos de emprego de máquinas SIMD são: algoritmo de mapeamento do nível de cinza como o gradiente de Sobel (Processamento de imagens); Processamento simbólico, como algoritmos de sorting e manipulação de grafos e solução de sistemas de equações [CAP85], operações com matrizes, resolução de equações diferenciais [SCH84].

Cada um dos elementos de processamento podem operar com operandos do tipo palavras ou com um único bit, durante cada ciclo da memória, esta característica define dois tipos de sistemas SIMD; o da organização de palavras (word-organized) e o de organização de bits (bit-organized). As operações aritméticas em um sistema de organização por palavras é análogo ao de



máquinas seriais; já os organizados por bit, tem influenciado grandemente o projeto de algoritmos paralelos.

O termo processador de array tem certa ambiguidade, pois pode referir a máquinas SIMD ou máquinas MIMD. O processador matricial geralmente incorpora um grande número de elementos processadores idênticos conectados numa particular topologia.

### 2.2.3 Multiprocessadores

Também conhecidos como arquiteturas MIMD são exemplos de paralelismo espacial assíncrono. É o multiprocessamento propriamente dito, onde diversos processadores trabalham em paralelo, processando suas tarefas concorrentemente de forma assíncrona para num intervalo de tempo concluírem a tarefa. Um multiprocessadores que se comunicam e cooperam para resolverem uma dada tarefa.

Classif de Flynn	Concorrência	Máquinas	Paralelismo	Exemplos de Maq.
SISD	Pipeline	Pipeline Sequenciais	Temporal -- x --	Cray 1 2 Ciber200 Nets Fujitsu
SIMD	Simultaneidade	Matriciais Arrays	Espacial Sincrono	Illiac IV MPP Pepe Staran
MISD	-- x --	-- x --	-- x --	Não Há
MIMD	Paralelismo	Multiprocessadores	Espacial Assíncrono	Cray-X-MP CM* HELP

Fig. 2.6 Quadro comparativo das arquiteturas

### 2.3 Níveis de paralelismos

O nível mais elevado de processamento paralelo é conduzido entre vários jobs ou programas através de multiprogramação, tempo compartilhado (time sharing) e multiprocessamento. Este nível requer o desenvolvimento de algoritmos processáveis paralelos. A implementação de algoritmos paralelos depende de uma eficiente alocação dos limitados recursos de hardware e software para múltiplos programas sendo usados para resolver problemas de grandes computações. O multiprocessamento não diminui o tempo de processamento de um programa, mas permite que vários sejam executados ao mesmo tempo.



O nível seguinte de processamento paralelo é dirigido entre rotinas ou tarefas (segmentos do programa) de um mesmo programa. Isto envolve a decomposição do problema em múltiplas tarefas, o que pode diminuir o tempo de processamento de um programa.

O terceiro nível é para explicar a concorrência entre múltiplas instruções. A análise de dependência dos dados é necessária para revelar paralelismos entre instruções. Vetorização pode ser desejável entre operações escalares com loops (DO).

Finalmente, o último nível é interno das instruções. Deve-se ter operações rápidas e concorrentes dentro de cada instrução.

Portanto para HWANG e BRIGGS ([HWAB84]), estes são os quatro níveis programáticos de paralelização, ou seja:

- Nível de jobs e programas;
- Nível de rotinas e tarefas de um programa;
- Nível de Instruções;
- Nível de intrainstrução (operações).

Nos níveis mais baixos são mais desenvolvidos via hardware, enquanto que os níveis superiores são mais desenvolvidos via software. Isto pode implicar no custo-efetivo do sistema.

Devido as várias características é necessário para o processamento paralelo uma combinação de vários campos de estudo. Ele requer conhecimento e experiência em vários aspectos de algoritmos, linguagens de programação, software, hardware, avaliação de performance e alternativas de computação.

Em estudos realizados em algoritmos numéricos, identificou-se alguns níveis de paralelismo envolvidos. O primeiro nível, denominado de nível de instrução, trata da carga da instrução e de sua execução no nível mais baixo, envolvendo a aritmética implementada (operação e operandos), o que corresponde ao nível mais baixo de Hwang e Briggs.

O segundo nível, o nível de execução trata do paralelismo no momento da compilação para sua posterior execução, o que corresponde ao nível de instruções, onde a vetorização de operações escalares e de loops, são muitas vezes obtidas.

Os dois níveis seguintes referem-se ao nível de rotinas e tarefas de Hwang e Briggs. No terceiro nível, o de rotinas, desenvolve-se versões baseadas na versão sequencial para os processadores disponíveis na máquina. Isto não implica necessariamente em aumento da velocidade de processamento, mas pode acelerar o processo de convergência das rotinas numéricas.



Por fim, tem-se o nível de paralelismo propriamente dito, onde se desenvolve programas baseados na arquitetura da máquina e na dependência das tarefas, explorando seu paralelismo. O programa em paralelo pode não ter nada a ver com a versão sequencial, pois deve explorar a natureza paralela do problema e combiná-la com os recursos que a máquina proporciona.

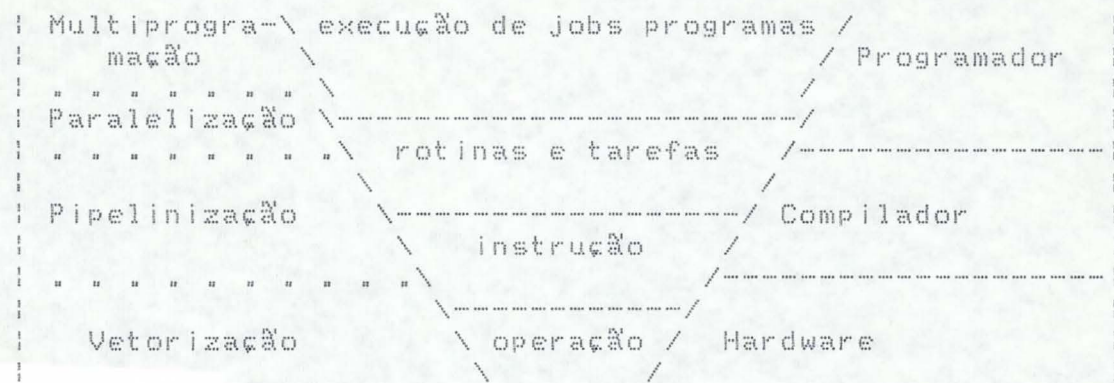


Fig. 2.7 Níveis x Tipo x Planejador do paralelismo

Observa-se que a coluna de níveis é um pouco abstrata, uma vez que os ditos níveis não são tão independentes e podem ocorrer uns nos outros.

## 2.4 Técnicas de reestruturação

As técnicas para reestruturar programas sequenciais para torná-los paralelos podem ser de três tipos: técnicas para geração de comandos paralelos de alto nível, técnica para eliminação de dependências e de comandos desnecessários e técnicas para adequar o programa às características da máquina.

A técnica de geração de comandos paralelos é utilizada para gerar comandos de alto nível existentes nas extensões de linguagens convencionais para computadores vetoriais ou multiprocessadores. Dentre este grupo de técnica se destacam:

i) Vetorização de comandos repetitivos - comandos repetitivos aninhados podem ser decompostos em diversos comandos repetitivos de menor escopo, e aqueles resultantes que não possuam ciclos de dependências em seu escopo podem ser vetorizados;

ii) Vetorização por redução - Muitos computadores vetoriais possuem instruções de máquina que implementam operadores matemáticos de redução, como o somatório e o produtório;

iii) Distribuição de iterações por processadores - iterações de um comando repetitivo podem ser distribuídas pelos processadores em um multiprocessador. Um exemplo é o comando doall, o qual quando utilizado expressa as instâncias distintas da sequência de comandos que podem ser executadas simultaneamente;



iv) Paralelização parcial de comandos repetitivos - Quando existem dependências, as expressões de um comando repetitivo não podem ser executadas simultaneamente. Há casos em que é possível uma simultaneidade parcial, ou seja, uma iteração pode começar a ser executada antes do término da iteração anterior, bastando isolar as dependências existentes. Usa-se primitivas de sincronização para garantir a dependência.

v) Método da frente da onda - é usado para obter paralelismo em aninhamentos de comandos de repetição quando seu escopo contiver ciclos de dependência.

vi) Reordenação de comandos de atribuição - possibilita a vetorização de comandos repetitivos de escopo não unitário e minimiza os efeitos negativos da sincronização de programas concorrentes.

Durante a elaboração de um programa sequencial é comum atribuir múltiplas interpretações a uma mesma posição de memória ou mesmo introduzir cálculos desnecessários, degradando o potencial de paralelismo do programa pelo estabelecimento de dependências desnecessárias. Muitos destes casos podem ser detectados e resolvidos pelo compilador, através das técnicas:

i) Uso de novos identificadores - o uso de um mesmo identificador com propósitos diferentes em diversos pontos do programa pode introduzir antidependências e dependências de saída, para eliminá-las, basta atribuir um identificador para cada propósito da variável;

ii) Expansão de variáveis escalares - a atribuição de valores a variáveis escalares dentro de comandos repetitivos é também causa de antidependências e dependências de saída, além de impedir a vetorização ou exigir excessiva sincronização. Para eliminar tais dependências substitui-se as variáveis escalares por indexados;

iii) Fissão de comandos - alguns comandos repetitivos contêm ciclos de dependências que podem ser eliminados através de atribuições temporárias;

iv) Substituição de variáveis por expressões - em caso de dependência direta entre dois comandos de atribuição, pode-se substituir a variável causadora da dependência pela expressão que a atualiza, eliminando desta forma a dependência;

v) Eliminação de variáveis de indução - variáveis de indução são funções dos índices de comandos repetitivos, com valores formando uma progressão aritmética. Quando elas aparecem como subscritos de variáveis indexadas, induzem a falsas dependências que podem ser eliminadas;

vi) Eliminação de recorrência de índices - alguns índices de variáveis indexadas são obtidos por recorrência internas a comandos repetitivos que geram falsas dependências.



As Técnicas de adaptação do programa à máquina são devidas as arquiteturas sempre imporem restrições à execução paralela de programas, seja pelo número de processadores, pela taxa de transferência de memória ou do sistema de entrada e saída, ou seja pelo modo de interconexão dos seus vários componentes. Cada compilador deve esconder estas restrições do usuário e produzir código que utilize as características favoráveis das máquinas.

As técnicas de reordenação planejadas pelo otimizador dos compiladores do Convex, incluem vetorização, vetorização parcial, paralelização, distribuição de loops e mudanças internas de loops.

#### 2.4.1 Vetorização

A vetorização substitui um loop que usa instruções escalares por um loop que usa instruções vetoriais, por exemplo,

```
DO 10 I = 1, 512
10 A(I) = B(I) + C(I)
```

torna-se, em Fortran na notação 8x (8x16 = 128),

```
DO 10 I = 1, 512, 128
10 A(I: I+127) = B(I: I+127) + C(I: I+127)
```

onde, elementos são carregados em registradores escalares, operandos são armazenados um por vez e, agora, uma única instrução vetorial carrega 128 elementos de B em um registrador, outra carrega 128 elementos de C e, outra ainda, soma os dois vetores. Uma quarta instrução vetorial armazena o resultado no vetor A. A capacidade do loop aqui é chamada **strip-mine loop** (passo mínimo do loop) é introduzida para acomodar o limite de 128 elementos do tamanho do registrador vetorial.

Vetorização é possível quando um loop não tem recorrências. Uma recorrência está presente quando uma operação de uma iteração depende de uma operação correspondente a mesma instrução mas executada em uma iteração anterior, por exemplo;

```
DO 10 I = 3,N
10 A(I) = A(I-2) * A(I-1)
```

aqui a armazenagem em A(I) na segunda iteração depende da carga de A(I-1), a qual por sua vez, depende do valor armazenado em A(I) da primeira iteração.

O compilador Fortran tem uma poderosa capacidade de vetorização, a qual se utiliza tanto de algoritmos gerais como do modelo de abordagem casada. O único modelo casado utilizado pelo compilador atual é para o reconhecimento de recorrências que podem na prática ser vetorizadas mas não por métodos diretos. O vetorizador pode manusear arbitrariamente trechos com loops, reordenar comandos e expansões escalares, como por exemplo:

```

        DO 10 I = 1, N
        IF ( A(I).LT.B(I)) GO TO 20
20     Z(I) = T * H(I)
30     T = X(I)
10     CONTINUE

```

é vetorizável, na prática, com o If sendo reescrito por:

```

        TEMP(I) = T
        DO I = 1, N
        TEMP(I+1) = X(I)
        T = X(I)
        IF ( A(I).LT.B(I)) GO TO 20
        Z(I) = TEMP(I)*G(I)
        GOTO 10
20     Z(I) = TEMP(I)*H(I)
10     CONTINUE

```

#### 2.4.2 Vetorização parcial

Muitos loops contêm recorrências, mas as recorrências se constituem de uma pequena parte, uma percentagem do corpo do loop. A **vetorização parcial** divide o loop em dois ou mais loops para separar a recorrência do código vetorizável. Então, vetoriza todos os loops que não contêm recorrências. O compilador 5.0 isola com sucesso códigos vetorizáveis, mesmo que tenha que separar comandos que só são executados dependendo de certas condições, como por exemplo:

```

        DO 10 I =1, 100
        IF (A(I).LT.B(I)) GOTO 10
        X(I) = X(I-1) * SQRT(B(I))
10     CONTINUE

```

é efetivamente reescrito como:

```

        DO 10 I =1, 100
        IF (A(I).LT.B(I)) GOTO 10
        TEMP(I) = SQRT(B(I))
10     CONTINUE

        DO 20 I =1, 100
        IF (A(I).LT.B(I)) GOTO 20
        X(I) = X(I-1) * TEMP(I)
20     CONTINUE

```

permitindo assim, que o compilador vetorize a computação da raiz quadrada.



### 2.4.3 Paralelização

**Paralelização**, no contexto, refere-se a geração de código que capacite iterações de um loop serem executadas simultaneamente e sem sincronização de múltiplos processadores. Um loop é paralelizado se não existir dependência entre iterações, isto é, se o resultado de uma iteração não influencia sobre resultados das próximas iterações. Tais loops tem sido referenciados na literatura como DOALL.

A paralelização é mais frequentemente aplicada para loops de loop ou para **strip-mine loop** gerado pela vetorização. Por exemplo;

```
DO 10 J =1, M
DO 10 I =1, N
A(I,J) = B(I,J) + C(I,J)
10 CONTINUE
```

aqui, o I-loop é vetorizado, mas o J-loop é paralelizado. Isto permite que N processadores sejam engajados simultaneamente na execução vetorial das operações. O computador pode ainda efetuar em paralelo strip-mining sobre o loop, para garantir que haja sincronização mínima no esquema das iterações.

Paralelismo strip-mining separa o loop paralelo em dois loops, mas paraleliza somente o loop de fora. Cada processador então executa uma porção contígua de iterações do loop paralelo original.

Quando não existe loop de fora do loop vetorial, o loop strip-mine é geralmente paralelizado, como por exemplo;

```
DO 10 I =1, N
10 A(I) = B(I)+C(I)
```

torna-se,

```
DO 10 I =1, N, 128
DO 10 II = I, Min(N,I+128)
10 A(II)= B(II)+C(II)
```

esses dois ninhos do loops podem ser agora vetorizados e paralelizados. Se N for relativamente pequeno, mas maior que 64, o compilador pode gerar um strip-mine de tamanho menor que 128 para que seja paralelizado. Se N é variável, o compilador montará dinamicamente o vetor strip-mining para selecionar o melhor tamanho de stripe-mine em tempo real.

O compilador pode também manipular mais argumentos escalares e reduções de loops paralelos. Como por exemplo, o seguinte loop será vetorizado e paralelizado.



```

DO 10 I =1, N
IF (A(I).GT.0) GOTO 10
S = S + B(I) * C(I)
X = B(I)
10 CONTINUE

```

Os escalares são distribuídos para os processadores, somas parciais são acumuladas por reduções e números iterativos se valem de outros argumentos escalares. Depois de extinção do loop, a soma parcial de S é adicionada e a tarefa privada copia X para iteração sucessiva copiando em X.

O compilador pode também extrair concorrência dos loops que contém recorrências ou que tenha dependência entre iterações mas não recorrências, como nos assim chamados loops DOACROSS e FORALL. O código paralelizado é gerado para loops paralelos, mas é suplementado com código para sincronizar cada iteração prévia tão necessária para satisfazer dependências.

Entretanto, loops nos quais estas técnicas providenciam speedups significativos são relativamente raros. Também, muitos loops com recorrência são parcialmente vetorizados pelo compilador. Os loops seriais restantes da vetorização parcial são plenamente recorrente e geralmente não se beneficiam de aplicar multiprocessadores.

#### 2.4.4 Distribuição de loops

Distribuição de loops criam uma sequência de ninhos de loops simples de um ninho, por exemplo;

```

DO 10 I =1, N
A(I) = 0
DO 10 J = 1, N
B(I,J) = A(I)
10 CONTINUE

```

torna-se

```

DO 10 I =1, N
10 A(I)= 0
DO 20 I =1, N
DO 20 J =1, N
B(I,J) = A(I)
20 CONTINUE

```

A distribuição de loops é realizada para criar mais loops interiores mais propícios a vetorização, para criar um maior potencial no paralelizar loops pelo isolar recorrências de loops externos e para criar ninhos simples onde iteração de loops podem ser usadas efetivamente. Vetorização parcial é uma real aplicação de distribuição de loops a um loop interior seguido de vetorização de resultados de loops vetorizáveis.



#### 2.4.5 Mudanças internas de loops

Mudança interna de loops recorda os loops de ninhos simples. O compilador pode realizar mudanças interiores por algumas razões. Isto pode ser realizado para diminuir o passo no qual vetores são acessados num loop ou, em alguns casos, para converter um vetor em um escalar.

Mudanças internas de loops podem também ser usadas para incrementar a quantidade de código executável entre pontos de sincronização pelo mover o loop para ser paralelizado. Alocação do vetor registrador global é feito possível pela mudança no loop, relativa a quais vetores são constantes dentro do strip-mine loop da vetorização.

Finalmente, mudanças internas do loop pode admitir um loop que pode ser altamente vetorizável, para ser deslocado internamente em loop interior que não pode ser vetorizado.

#### 2.5 Processamento vetorial

Processamento vetorial ou vetorização tem diferentes significados para diferentes tipos de pessoas. Para pessoas que desenvolvem linguagens, por exemplo, vetorização é utilizada para caracterizar linguagens do tipo array, como a extensão do fortran vectran. Para pessoas que desenvolvem compiladores, significa análise de dependências de comandos do código fonte (do-loop) e conversões de operações sequenciais para operações vetoriais equivalentes. Para usuários, vetorização significa a introdução de instruções vetoriais de hardware em seus programas, para que a alta velocidade destas instruções possam ser utilizadas de forma a melhorar o desempenho de seus programas.

Adotando o ponto de vista do usuário, a seguir é mostrado as três formas de se utilizar a vetorização e uma caracterização mais formal das instruções vetoriais.

A primeira forma de se obter a vetorização é simplesmente recompilar os códigos escalares em Fortran, utilizando o compilador que vetoriza automaticamente os códigos, como o compilador fortran disponível no Convex.

Outra forma de vetorizar é reestruturar o código fonte para auxiliar o compilador a reconhecer mais oportunidades para gerar códigos vetoriais, obtendo assim mais benefícios, um maior grau de vetorização das rotinas.

Por fim, recodificar completamente a aplicação pela escolha ou pelo imaginar um novo algoritmo para colher o maior benefício das características vetoriais da máquina.

Entre as características do processamento vetorial, está o operando vetorial, que contém um conjunto ordenado de  $N$  elementos, onde  $N$  é chamado do tamanho do vetor. Cada elemento em



um vetor é uma quantidade escalar, a qual pode ser um número em ponto-flutuante, um inteiro, um valor lógico ou um caracter (byte). Instrução vetorial pode ser classificada em quatro tipos de primitivas:

f1:  $V \rightarrow V$                       f2:  $V \rightarrow S$   
 f3:  $V \times V \rightarrow V$                 f4:  $V \times S \rightarrow V$

onde  $V$  e  $S$  denotam um operando vetorial e um operando escalar, respectivamente. As funções  $f1$  e  $f2$  são operações unárias, enquanto que  $f3$  e  $f4$  são binárias. Na figura 2.8 mostra algumas operações vetoriais (que podem ser achadas em processadores vetoriais modernos) segundo o tipo de operação.

O produto interno de dois vetores, definido como a soma dos produtos das componentes dos vetores, é gerada pela aplicação de uma operação  $f3$  e, então uma operação  $f2$  na sequência.

TIPO	MNEMONICO	DESCRIÇÃO	FÓRMULA
f1	VSQR	raiz quadrada vetorial	$B(i)=A(i)^{(1/2)}$
	VSIN	seno vetorial	$B(i)=\sin(A(i))$
	VCOM	complemento vetorial	$A(i)=\text{comp}(A(i))$
f2	VSUM	soma vetorial	$S=A(1)+\dots+A(n)$
	VMAX	máximo vetorial	$S= \max \{A(i)\}$
f3	VADD	soma vetorial	$C(i)=A(i)+B(i)$
	VMPY	multiplicação vetorial	$C(i)=A(i)*B(i)$
	VAND	And vetorial	$C(i)=A(i)\&B(i)$
	VLAR	Maior vetorial	$C(i)=\max(A(i),B(i))$
	VTGE	teste maior vetorial	$C(i)=0$ se $A(i)<B(i)$ $C(i)=1$ se $A(i)>B(i)$
f4	SADD	soma escalar vetorial	$B(i)=S+A(i)$
	SDIV	divisão escalar vetorial	$B(i)=A(i)/S$

Fig. 2.8 Operações vetoriais

O processamento vetorial elimina a maioria dos testes e controle de operações, principalmente as associadas a loops. Ele minimiza o tempo dispendido na espera da carga dos operandos e de armazenagem do resultado, pelo referenciar grupos de locação de memória e pelo uso de registradores de múltiplos elementos.

Outra vantagem do processamento vetorial é que ele reduz o número de instruções em linguagem de máquina que precisam ser carregados, decodificados e executados. Um exemplo ilustrativo são os comandos em linguagem de máquina necessários para implementar o seguinte loop;

```
DO 10 I = 1,100
  A(I) = B(I) * C(I)
10 CONTINUE
```



para processar usando o processamento escalar seria necessário, as instruções:

```
LOAD B(1)          LOAD B(2)          LOAD B(100)
LOAD C(1)          LOAD C(2)          LOAD C(100)
  MULT      ==>    MULT      ==> [3 a 99] ==>    MULT      ==> FIM
STORE A(1)        STORE A(2)          STORE A(100)
TESTA I          TESTA I              TESTA I
INCREMENTA       INCREMENTA
```

utilizando processamento vetorial seria necessário apenas o seguinte conjunto de instruções de máquina:

```
LOAD B(1:100)
LOAD C(1:100)
  MULT
STORE A(1:100)
```

A unidade de processamento vetorial tem um total de 8 vetores registradores com 128 elementos cada e de 64 bits por elemento. Existem três unidades funcionais controladoras que controlam as operações vetoriais de LOAD/STORE, Multiplicação Divisão e Soma/Operações lógicas.

Neste item, pretende-se, ainda, identificar quem pode ser vetorizado e que não pode ser. Loops que contém referências a arrays ou que se utilizam de variáveis indutivas como subscrito, são fortes candidatas a serem vetorizados. Um loop de variável de indução é um loop onde há uma variável que é incrementada ou decrementada por um valor constante a cada iteração do loop, também chamada de variável de controle do loop.

Loops aninhados, geralmente só pode ser vetorizado o loop mais interno. Loops com condições também podem ser vetorizados.

O compilador pode algumas vezes distribuir ou fazer mudanças internas nos loops aninhados de maneira a providenciar que seja vetorizado. Alguns argumentos em valores escalares podem ser vetorizados utilizando operações de redução vetorial.

O compilador providencia no código dos loops com uma quantidade de iteração maior que 128, que seja feito um strip mine, ou seja, providência um loop em assembler no qual são processados um grupo das primeiras 128 iterações, então são processados o restante, mas sempre em módulos de 128 iterações.

O tamanho do vetor registrador indica o número de elementos (0 a 128) sobre o qual o próximo vetor de instrução irá operar. O registrador do passo vetorial indica o pulo (gap), em bytes, entre sucessivos elementos vetoriais na memória.

A vetorização de um loop pode ser inibida pela existência nos loops de chamadas de subrotinas ou funções,



comandos de leitura e impressão, múltiplas entradas ou saídas, arrays ou variáveis equivalentes e recorrência.

A recorrência existe quando um novo valor de um dos elementos do vetor depende do uso de outro resultado do mesmo vetor de operação. Resultados de um vetor de operações nunca estão disponíveis para o uso do mesmo vetor operação.

Outra situação que inibe a vetorização é quando existe um valor subscripto que não pode ser determinado em tempo de compilação, por ser um valor, por exemplo lido do terminal. Isto ocorre por medida de precaução, evitando uma possível recorrência.

Um exemplo de recorrência é dado se tivermos os vetores A e B, como definidos abaixo e sobre eles operarmos o seguinte loop:

```
DO 10 I= 2, 100
  A(I) = A(I-1) +B(I)
10 CONTINUE
```

sendo  $A := \langle 1, 2, 3, \dots, 99, 100 \rangle$ , e  
 $B := \langle 100, 100, 100, \dots, 100 \rangle$

Como cada atribuição depende do resultado da operação de atribuição anterior, a avaliação do processamento escalar produzira como resultado o vetor,

$A = \langle 1, 101, 201, 301, 401, \dots, 901 \rangle$ .

Se o loop fosse vetorizado, teríamos a operação vetorial:

$A(2:100) = A(1:99) + B(2:100)$

cuja a avaliação vetorial produziria o vetor resultante A,

$A = \langle 1, 102, 103, 104, 105, \dots, 199 \rangle$ .

Em uma máquina escalar, o algoritmo mais rápido é o que requer a menor quantidade de operações aritméticas. Isto não é necessariamente verdade em máquinas vetoriais. Um algoritmo com grande número de operações mas que quando vetorizado não tem custo extra de operações aritméticas, pode ser mais rápido. Um exemplo disto é a solução de sistemas tridiagonais, onde o algoritmo vetorial requer cerca do dobro de operações do algoritmo escalar, mas é muito mais rápido que o algoritmo escalar executado no modo escalar para vetores longos suficientemente. O desempenho de um código vetorial também depende de uma verificação do tamanho do vetor e do tempo de startup do vetor de instruções.



### 3 APLICAÇÕES DE PROCESSAMENTO PARALELO

Computadores rápidos e eficientes estão em alta demanda nas áreas científicas e militares, nas engenharias, no estudo e exploração de fontes de energia, na medicina, na inteligência artificial e na pesquisa básica. Computações de larga escala são realizadas nessas áreas de aplicações. Computadores de processamento paralelo são necessários para reunir esta demanda. Sem utilizar supercomputadores, muitos dos desafios para o avanço da civilização humana seriam difíceis de serem vencidos.

Para projetar o custo-efetivo de supercomputadores ou para melhor utilizar um sistema existente de processamento paralelo é necessário identificar as necessidades computacionais de importantes aplicações.

Resolução de problemas científicos de larga escala envolvem a interação de três atividades: da teoria, da experimentação e da computação; como é mostrado na figura 4.1. Cientistas teóricos desenvolvem modelos matemáticos que resolvem numericamente computações da engenharia; o resultado numérico pode então sugerir novas teorias. A ciência experimental providencia dados para ciência computacional e, por fim, podem modelar processos que são difíceis de se projetar em laboratórios. Uso de simulações computacionais tem vantagens, como:

- Simulações computacionais são muito mais baratas e rápidas que experimentos físicos;
- Computadores podem resolver um intervalo muito maior que equipamentos específicos de laboratórios;
- Abordagem computacional é somente limitada pela velocidade computacional e pela capacidade de memória, enquanto que experimentos físicos tem muitas restrições práticas.

Cientistas teóricos e experimentais são usuários de programas grandes desenvolvidos por cientistas computacionais. Os códigos podem ainda gerar resultados com uso mínimo de esforço. Cientistas computacionais aplicam tecnologias avançadas em modelagem numérica, engenharia de hardware e desenvolvimento de software.



Fig. 3.1 Interação entre experimentais, teóricos e computacionais



### 3.1 Modelagem preditiva e simulação

Modelação multidimensional da atmosfera, do ambiente terrestre, do espaço externo e do mundo econômico tem se tornado a maior concentração de cientistas. Modelação preditiva é feita através de extensas simulações computacionais de experimentos, nas quais podem envolver computações de larga-escala para garantir a exatidão desejada e a minimização do tempo necessário para solução. Tais modelações numéricas requerem um estado da arte computacional de velocidade aproximada de 1000 milhões de megaflops.

#### 3.1.1 Previsão numérica atmosférica

Recursos atmosféricos e climáticos nunca satisfarão suas necessidades em computadores rápidos. Modelagem atmosférica é necessária para diminuir o intervalo de predição e para estimar intervalos longos de predição azarada. O analista atmosférico necessita resolver equações de modelos gerais de circulação em computadores. O estado atmosférico é representado pela superfície precionada, pelo campo virado, pela temperatura e pela variação do vapor d'água. Estes estados variáveis são governados pelas equações da dinâmica de fluídos da Navier-Stoke num sistema de coordenadas esféricas.

A computação é levada a cabo por grade tri-dimensional que particiona a atmosfera verticalmente em  $k$  níveis, horizontalmente em  $M$  intervalos da longitude e  $N$  intervalos de latitude. A quarta dimensão é adicionada com o número  $P$  de passos usados na simulação. Utilizando uma grade de 270 milhas de um lado, uma previsão de 24h precisa-se realizar cerca de 100 bilhões operações de dados.

Esta previsão pode ser feita num computador de 100 megaflops em cerca de 100 minutos. A grade de 270 milhas dá uma previsão entre Nova York e Washington, mas não em Philadelphia. Aumentando a previsão pelo manuseio o tamanho da grade nas quatro dimensões, precisa-se fazer a computação 16 vezes mais rápida. A máquina de 100 megaflops, como a Cray-1 pode levar 24 horas para completar a previsão de 24h. Em outras palavras, requer um computador 16 vezes mais poderoso (1.6 gigaflops) para terminar a previsão em 100 minutos.

#### 3.1.2 Oceanografia e astrofísica

Como oceanos podem armazenar e transferir calor e intercambiar com a atmosfera, um bom entendimento de oceanos pode auxiliar nas áreas de: análise climática preditiva, controle de peixes, exploração dos recursos do oceano, mares e mudanças da costa.



Estudos oceanográficos usam um tamanho de grade de escala pequena e um tempo variável em grande escala, em todos os usos nos estudos atmosféricos. Para fazer uma simulação completa do oceano Pacífico com uma resolução adequada (1 grau) para 50 anos, necessitaria 1000 horas em um computador Cyber-205.

A formação da terra no sistema solar pode ser simulada em um computador de alta velocidade. O intervalo dinâmico de estudos astrofísicos podem ser bilhões de anos por milissegundos. Problemas interessantes incluem a dinâmica das galáxias tridimensional, integração de n-corpos caem em tal estudo, envolvendo dez na cinco partículas movendo-se consistentemente sobre forças Newtonianas. O processador matricial Illiac IV tem sido usado neste estudo.

### 3.1.3 Uso socioeconômicos e governamentais

Grandes computadores estão em grande demanda em áreas econômicas, engenharia social, censos governamentais, controle de crime e a modelagem do mundo econômico para o ano 2000. Nobel W W Leontief ( em 1980) propôs um modelo de controle do mundo econômico com a realização de larga-escala de operações matriciais num computador científico CDC. Ainda se suporta simulações do mundo econômico sugerindo como um sistema de relações econômicas internacionais que caracteriza um desarmamento parcial pode anunciar a diferença entre o rico e o pobre.

Nos Estados Unidos, o FBI usa grandes computadores para controle do crime; IRS usa um grande número de mainframes rápidos para controlar o recebimento de taxas e auditorias. Não existe dúvida sobre o uso de supercomputadores para o censo nacional e para levantamentos da opinião pública sobre assuntos gerais. Foi estimado que cerca de 57% dos supercomputadores manufaturados nos Estados Unidos foram utilizados pelo governo americano no passado.

## 3.2 Engenharia de projeto e automação

Supercomputadores rápidos tem estado em alta demanda para resolver muitos problemas de projeto de engenharia, tais como análise de elementos finitos necessários em projetos estruturais e experimentos de túnel de vento para estudos de aerodinâmica. Desenvolvimento industrial também necessita de computadores para possibilitar automação, inteligência artificial e sensoriamento remoto de recursos terrestres.

### 3.2.1 Análise de elementos finitos

O projeto de represas, pontes, navios, jatos supersônicos, altos edifícios e veículos espaciais requerem a resolução de grandes sistemas de equações algébricas ou de



equações diferenciais parciais. Abordagens convencionais utilizadas no desenvolvimento de pacotes de softwares (escritos em códigos sequenciais) requerem tempos de retorno intoleráveis. Muitos pesquisadores e engenheiros tem atentado para construção de computadores mais eficientes para realizar análise de elementos finitos ou para buscar solução finita e diferenciável, o que pode implicar numa mudança fundamental no desenvolvimento de ferramentas e em uma alta produtividade no futuro.

Engenheiros computacionais tem desenvolvido códigos de elementos finitos para análise dinâmica de estruturas. Elementos finitos de alta ordem são usados para descrever o comportamento do espaço. Os comportamentos do tempo podem ser aproximados pelo uso de uma diferença central no esquema explícito. Vetorização de rotinas podem ser usadas para gerar a rigidez e a massa dos elementos das matrizes para decompor a matriz global e para multiplicar a matriz rígida global por um vetor. O CDC Star 100 e o Cyber-205 tem sido usados para implementar estas computações para análise estrutural

### 3.2.2 Aerodinâmica computacional

Computador de larga-escala tem dado contribuições significativas na produção de novas capacidades tecnológicas e econômicas na pressão enfrentada em aeronaves, no lançamento de naves espaciais e nos estudos de turbulência. O centro de pesquisa Ames da NASA, está utilizando o Illiac IV para simulações tridimensionais de teste de tunel de vento em velocidade gigaflop. As principais limitações em túnel de vento são o tamanho do modelo, velocidade do vento, densidade, temperatura, distorções aeroelásticas, atmosfera e corrente uniforme. Cada túnel de vento é limitado por uma escala de efeito atribuídas as limitações. As simulações numéricas são limitadas pela velocidade do processador e pela capacidade de memória, mas não é limitada por nenhuma das limitações físicas.

Dois supercomputadores gigaflops, conhecidos como NASF (Numerical Aerodynamic Simulation Facilities) tem sido propostos pela Burroughs e pela Control Data. Existem máquinas especializadas "Navier-Stokes", capazes de realizar simulações completas do projeto de aeronaves, tanto para o governo americano como para companhias comerciais.

### 3.2.3 Inteligência artificial e automação

Interfaces inteligentes de I/O são necessárias para os novos supercomputadores, que usam comunicação direta com seres humanos através de imagens, falas e linguagens naturais. As funções inteligentes que necessitam de processamento paralelo são: processamento de imagem, reconhecimento de padrão, visão computacional, reconhecimento de voz, máquina de inferência, CAD/CAM/CAI/OA, robótica, sistemas especialistas e engenharia do conhecimento.



Arquiteturas especiais de computadores tem sido desenvolvidas ou projetadas para algumas dessas funções. Recentemente, o Japão lançou um projeto nacional para desenvolver o computador de quinta geração para ser utilizado nos anos 90. A visão japonesa sobre a nova geração de computadores inclui subsistemas inteligentes de I/O, capazes de realizar a maioria das funções citadas. O poder computacional projetado no sistema que está sendo desenvolvido é de 100 mega a 1 giga de inferências lógicas por segundo (LIPS). O tempo para executar uma inferência lógica é igual a execução de 100 a 1000 instruções de máquina. Portanto, a máquina seria capaz de executar 10 000 a 1 mega milhões de instruções por segundo (MIPS). Com tal poder computacional é esperado processar bases de conhecimento e servir como um sistema especialista de multipropósito para aplicações futuras.

#### 3.2.4 Aplicações em sensoriamento remoto

Análise computacional remotamente sensoriada (via satélite, por exemplo) de dados de recursos terrestres tem muitas aplicações potenciais, como na agricultura, geologia e procura de água. Quantidades explosiva de informações de pontos (píxels) são necessárias para processar nesta área. Por exemplo, uma simple faixa da imagem do LANDSAT contém 30 milhões de bytes. Ele necessita fazer 13 destas faixas de imagem para cobrir o estado de Alabama.

A NASA vem usando o processador MPP para processar imagens de satélites com recursos terrestres. O MPP tem um pico de computação perto de 6 bilhões de operações inteiras por segundo.

### 3.3 Exploração de recursos energéticos

Energia afeta o progresso de toda economia na base global. Computadores podem desempenhar um importante papel no descobrimento de óleo e de gás e o gerenciamento de suas descobertas, no desenvolvimento do trabalho de energia da fusão do plasma e no garantir a segurança do reator nuclear. Usando computadores na área energética, os resultados são um menor custo de produção e uma maior segurança nas medidas.

#### 3.3.1 Exploração sísmica

Muitas companhias de óleo estão investindo no uso da união de processadores matriciais ou supercomputadores vetoriais para processamento de dados sísmicos, os quais foram responsáveis por cerca de dez por cento do óleo encontrado na costa.

Exploração sísmica reúne conjuntos de ondas sonoras produzidas por explosivos ou por vibrações geradas por passadas



pressas hidráulicas sobre uma base, segundo um modelo de vibração controlado computacionalmente. Para captar o eco produzido, espalha-se uns milhares de fones em uma malha. Esses ecos são utilizados para desenhar o cruzamento bidimensional de seções que mostram as camadas geométricas subterrâneas. Técnicas de reconstrução são usadas para identificar tipos de camadas que podem conter óleo. Tais explorações sísmicas podem evitar que sejam cavados muitos poços secos.

Um campo típico para guardar a resposta da terra de um impulso sonoro, tem 3000 diferentes valores de tempo, cada um com cerca de 48 diferentes localizações. Isto produz cerca de 2 a 5 milhões de pontos-flutuantes por Km sobre uma linha da superfície. Em 1979, somente 10 elevado na 15 bits de dados sísmicos eram processados. Uma companhia geofísica em Houston tinha cerca de 2 milhões de rolos magnéticos com dados sísmicos, dos quais 300 000 rolos aguardavam para serem processados. A necessidade de computadores para processamento sísmicos vem crescendo rapidamente.

### 3.3.2 Modelagem de reservatórios

Supercomputadores tem sido utilizados para realizar modelos tridimensionais de campos de óleo. O problema de reservatório é resolvido pelo método das diferenças finitas numa representação tridimensional do campo. Exemplos centrais geológicos são examinados para servir de base para projetar futuros reservatórios de performance ótima.

Atualmente cerca de 1000 flops são necessários para processar cada ponto do modelo tridimensional de um campo de óleo. Isto significa que um computador superpoderoso precisa ser projetado para proporcionar um desempenho preciso nas avaliações em um período de tempo razoável, em um campo de tamanho grande.

Devido a importância do campo de óleo da Baía Prudhoe, nos Estados Unidos, a companhia Petróleo Sohio construiu um simulador numérico para todo o campo num computador vetorial (Cyber-203). O campo é de cerca de 168 metros de espessura e foi dividido em 12 camadas com uma área de não menos 160 acres. Um modelo de 16 421 blocos de atividades subterrâneas podem produzir 1000 barris de óleo. As equações de diferença finita para o modelo do reservatório é resolvido iterativamente no Cyber-203. A simulação de um ano requer 33 minutos de processamento. O sucesso desta grande modelagem é atribuída a alta velocidade e a grande memória principal disponível no supercomputador.

### 3.3.3 Poder de fusão do plasma

Pesquisas de fusão nuclear tem empurrado para o uso de computadores cem vezes mais poderosos que qualquer outro para modelar plasmas dinâmicos na proposta de geração de poder de fusão de Tokamak. Programas de pesquisa de fusão magnética tem



sido ajudada por supercomputadores vetoriais no Laboratório Nacional Lawrence Livermore e no Laboratório de Plasma Físico de Princeton. O potencial da fusão magnética proporciona uma fonte energética alternativa que tem se tornado próxima, graças aos resultados de esforços cooperativos de programas experimentais com programas de simulação computacional.

Fusão nuclear sintética requer o aquecimento do plasma a uma temperatura de cem milhões de graus. Isto é um esforço muito caro. A alta temperatura do plasma consiste da carga positiva dos ions e da carga negativa dos eletrons, que precisa ser magneticamente separadas. O Centro Nacional de Fusão Magnética de Energia utiliza dois computadores Cray-1 e um CDC-7600 para conseguir controlar experimentos com o plasma. Supercomputadores tem se tornado uma ferramenta indispensável na exploração de energia na fusão magnética.

### 3.3.4 Segurança de reatores nucleares

Projetos de reatores nucleares e controle de segurança podem ambos serem conseguidos por estudos de simulação computacional. Esses estudos pretendem providenciar uma análise on-line das condições do reator, controle automático para operações normais e anormais, simulação de treinamento de operação, assessoramento rápido em caso de um potencial acidente.

As regras importantes nas operações anteriores tem sido feito em tempo real. Por segurança pela análise da luz do reator, o código TRAC tem sido desenvolvido para simular o desequilíbrio, o fluxo de água não homogêneo de alta temperatura. Outro código, o SIMMER II tem sido desenvolvido para analisar o centro de fusão na criação do reator. Somente supercomputadores podem realizar estes cálculos em tempo real.

## 3.4 Pesquisa básica, militar e médica

Na área médica, computadores rápidos são necessários na assistência computadorizada de tomografia, projeto de coração artificial, diagnóstico de vida, estimativa de dano no cérebro e estudos de engenharia genética.

Defesa militar necessita de supercomputadores para projetos de armamentos, simulação de efeitos e outras estratégias eletrônicas de guerra. Quase todas áreas de pesquisa básica necessitam de computadores rápidos para desenvolver seus estudos.

### 3.4.1 Tomografia computadorizada

O corpo humano pode ser modelado pelo exame de tomografia computadorizada (CAT - Computer Assistent Tomography). A Clínica Mayo em Rochester, Minnesota, tem desenvolvido pesquisas no exame de tomografia computadorizada quanto a



tridimensionalidade, ação parada, ação cruzada vendo o coração humano. No Instituto Courant de Ciências Matemáticas, cientistas pesquisam um processador matricial para a sequência de tempo, modelagem tridimensional do fluxo de sangue no coração, com o objetivo de desenvolver corações artificiais.

Abordagens similares podem ser aplicadas para revelar segredos de outros organismos humanos em tempo real.

O cruzamento por seções de imagens no exame CAT, usam de 6 a 10 minutos para serem gerados em computadores convencionais. Utilizando um processador matricial dedicado, o tempo de processamento pode ser reduzido para 5 a 20 segundos. A reconstrução da imagem da anatomia humana nos atuais exames CAT é em duas dimensões. Ela é gerada muito lentamente para conseguir congelar o movimento de órgãos como o coração ou pulmões. O super exame CAT da Clínica Mayo é esperado ter uma velocidade de 2000 a 3000 megaflops, produzindo imagens tridimensionais da batida do coração, em poucos segundos, com uma figura de 60 a 240 seções adjacentes cruzadas empilhadas uma sobre as outras. Devido ao pequeno processamento e a explosão do tempo, imagens tridimensionais e parada do movimento da batida do coração foram possíveis pela primeira vez. A Injeção de tinta pode ser usada para traçar o fluxo do sangue.

#### 3.4.2 Engenharia genética

Sistemas biológicos podem ser simulados em supercomputadores. Engenharia genética tem avançado rapidamente nos últimos anos. Existe uma grande necessidade de larga-escala de computações para estudar biologia molecular, para sintetizar moléculas orgânicas complexas, tais como proteínas. Cristalografia também pode ser ajudada com processamento computacional.

Uma máquina altamente Pipeline chamada Cytocomputer, tem sido desenvolvida pelo Instituto de Pesquisa Ambiental de Michigan para processamento de imagem biomédica. Ele pode ser usado para pesquisas de mutações genéticas. Técnicas sofisticadas biomédicas e computacionais estão sendo aplicadas para derivar estimativas precisas da ordem de mutações de espécies. Para um dado casal, pode-se prever as características de um filho usando no Cytocomputer algumas técnicas paralelas de casamento mapeado.

#### 3.4.3 Pesquisa de armamento e defesa

A pouco tempo, agências de pesquisa militares americanas usavam a maioria dos supercomputadores existentes. De fato, o primeiro Cray-1 instalado foi no Laboratório Científico Los Alamos em 1976. Em 1981, mais quatro Cray-1 foram comprados pelo Los Alamos.

Entre as aplicações militares relacionadas a defesa que se utilizam de supercomputadores, destacam-se: projeto de



armamento nuclear (Cray-1); simulação de efeitos de armamentos atômicos pela solução de problemas hidrodinâmicos e de radiação (Cyber-205); levantamento de informação, tais como processamento de sinal de radar em um processador associativo para o programa de míssil antibalístico Scud (ABM, PEPE); processamento de dados cartográficos para geração automática de mapas (Staran), vigilância submarina para defesa contra submarinos (Multiprocessador S-1).

#### 3.4.4 Problemas de pesquisa básicos

Muitas outras áreas de aplicação estão relacionadas por pesquisa científica básica. Entre as outras áreas que necessitam de supercomputadores destacam-se: a química computacional para resolver problemas da mecânica quântica, mecânica estatística, química fina e crescimento de cristal; análise física computacional de rastro de partículas geradas em quartos de falasca, estudos da dinâmica de fluídos, exame da teoria do campo quântico e investigações da dinâmica molecular; na engenharia eletrônica para resolver larga escala de equações de circuitos usados no algoritmo de Newton multinível e no projeto de conexões VLSI em chips semicondutores.

#### 3.5 Aplicações na Engenharia Mecânica

No departamento de Engenharia Mecânica da UFSC, existem três grupos principais que se utilizam do supercomputador Convex C210. O SINMEC (grupo de Simulação Numérica da Mecânica de Fluídos); o NRVA (grupo de Refrigeração, Ventilação e Ar condicionado) e o GRANTE (Grupo de Análise de Tensões).

No GRANTE são estudados métodos matemáticos em mecânica dos sólidos, mais especificamente elementos finitos, elementos de contorno e o método da função de Galerkin modificado. Os elementos finitos para efeito de resolução, são discretizados resultando em grandes sistemas de equações. Elementos finitos geram matrizes densas, enquanto que elementos de contorno geram sistemas esparsos. A resolução numérica destes sistemas pode ser feita pelo método dos gradientes conjugados.

A ordem dos sistemas de equações resolvidos no GRANTE chegou a 20 mil equações, o que levou cerca de cinco horas de processamento do Convex para sua resolução. Problemas resultantes em autovalores e autovetores resolvidos no GRANTE, chegaram a ordem de doze mil equações.

Outro problema estudado no GRANTE são os elementos finitos adaptativos, onde cada elemento é interpolado por polinômio de grau um a oito, dependendo da exatidão gerada pela interpolação do elemento. A resolução é feita com o auxílio do software solver, adaptado da França por um aluno de doutorado da Engenharia Mecânica. Este tipo de solução só é possível de se obter com o uso de supercomputadores.



O Grupo SINMEC, estuda a mecânica de fluidos. A modelagem do problema recai na solução de sistemas matriciais, onde as matrizes são tridiagonais, um tipo específico de matriz esparsa.



## 4 CONSIDERAÇÕES SOBRE ALGORITMOS NUMÉRICOS PARALELOS

Neste capítulo serão descritos algoritmos paralelos para a solução de relações de recorrência, equações diferenciais parciais e sistemas de equações lineares. Uma relação de recorrência é uma equação que expressa o valor da função em um ponto em termos de valores de outros pontos. Estudos mostram um número de instâncias na qual equações de recorrência ocorrem em análise numérica. Estas instâncias incluem a solução de equações lineares através da eliminação de Gauss; solução de equações diferenciais ordinárias no tempo e métodos que levam para a solução de equações diferenciais no esparso.

Equações diferenciais parciais (PDE) aparecem frequentemente na engenharia, na física, na química e em outras ciências físicas. O caso de estudo considerado aqui, é o simples, o problema intuitivo de achar o estado constante de distribuição de temperatura em uma placa fina de metal retangular com condições fixas de limites.

O outro problema considerado são calculos matriciais, como a solução de sistemas de equações lineares através do algoritmo de eliminação de Gauss.

### 4.1 Resolução de relações de recorrência

Uma relação de recorrência geral de primeira ordem tem a forma:

$$x_j = a_j \cdot x_{j-1} + d_j, \text{ para } j=1, \dots, n$$

onde os valores  $x_0, a_1, a_2, \dots, a_n$  e  $d_1, \dots, d_n$  são dados. Pode ser assumido, sem perda de generalidade, que  $x_0 = a_1 = 0$ , portanto se os valores não são nulos,  $d_1$  pode ser redefinido como igual a soma  $d_1 + a_1 x_0$ . Assumiu-se isto para facilitar a análise e subsequente simplificação no algoritmo para entendimento.

Dada uma sequência de valores  $d_1, d_2, \dots, d_n$  a soma parcial  $x_i$  é definida pelo somatório dos  $d_j$ , com  $j=1$  a  $i$ . Achados todas as somas parciais  $x_1, \dots, x_n$  da sequência de valores é na realidade um caso especial de recorrência linear de primeira ordem, o caso onde todos os  $a_j$  são iguais a um.

Um algoritmo para calcular as somas parciais da sequência de valores é dado na figura 4.1. O algoritmo é chamado de método da soma parcial por cascata. Assumindo que  $n=2^k$ , os valores  $d_1, \dots, d_n$  são armazenados nos elementos processadores  $P_0, P_1, \dots, P_{n-1}$ . Em outras palavras, a variável  $d(i)$  no elemento processador  $P_0$  contém  $d_1$  e assim por diante. Quando o algoritmo termina,  $x(i)$  irá conter a soma parcial  $x_{i+1}$ . A complexidade do algoritmo é  $O(\log n)$  com  $n$  elementos processadores no modelo.



```

BEGIN
  FOR ALL Pi, where 0 ≤ i ≤ n-1
    DO x(i)=d(i);
  FOR i=0 to log n-1
    DO FOR ALL Pj, where 2i + 1 ≤ j ≤ n
      DO BEGIN
        t(j)= x(j-2i)
        x(j)= x(j)+t(j)
      ENDFOR;
    END
  END

```

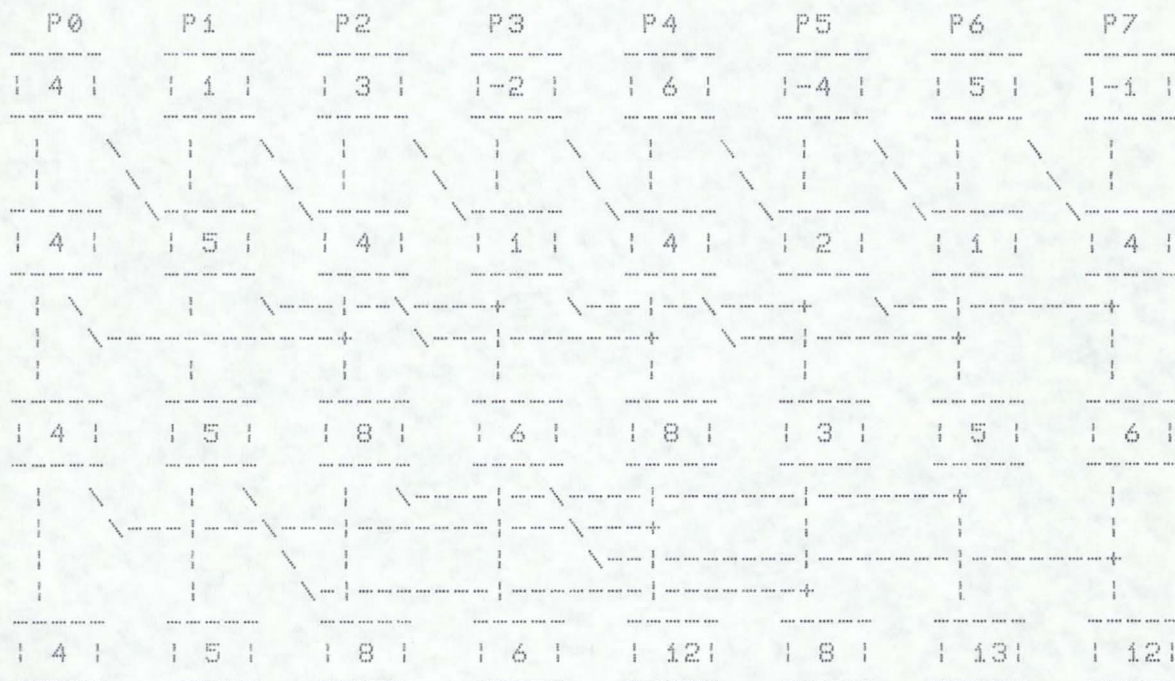


Fig. 4.1 Algoritmo da Soma Parcial por cascata e execução (modelo com 8 processadores)

Um modelo bastante similar pode ser desenvolvido para resolver uma relação de recorrência linear de primeira ordem geral. O algoritmo é chamado de **resolução cíclica** e ele inicia pela redução de relações entre termos adjacentes na sequência com relações entre cada outro termo na sequência. No segundo passo, relações entre dois termos na sequência são reduzidos para relações entre quatro termos na sequência. Depois de um número logarítmico de passos de reduções, os termos são tomados no seu valor final, pois eles são usados para valores constantes e não para outros termos.

Considerando dois sucessivos termos na relação de recorrência, temos:







```

BEGIN
  FOR i=0 to log n-1
    DO FOR ALL Pj, where  $2^i + 1 \leq j \leq n-1$ 
      DO BEGIN
        IF i >= 1
          THEN BEGIN
            t(j) = a(j-2^i)
            a(j) = a(j)+t(j)
          ENDIF
        t(j) = d(j-2^i)
        d(j) = a(j)*t(j) + d(j)
      ENDFOR
    FOR ALL Pj DO x(j) = d(j);
  END.

```

Fig. 4.2 Algoritmo Paralelo Redução cíclica

## 4.2 Solução de equações diferenciais parciais

Métodos para solução de equações diferenciais parciais (PDE) podem ser classificados em métodos diretos e métodos iterativos. Métodos diretos resolvem PDEs analiticamente; métodos iterativos iniciam com valores estimados em certa localização específica e, então, convergem para estimativas mais exatas. Algoritmos iterativos para resolver PDEs podem consumir um tempo de processamento bastante grande, por isto é interessante olhar para computações paralelas. Quando a utilidade de um método iterativo é verificada, o intervalo de convergência dos valores para se ter a solução é de grande importância.

J. Rice apresenta uma taxonomia para métodos paralelos, baseada no uso de três rotinas: particionamento, discretização e iteração.

**Particionamento** - As variáveis no problema são divididas em grupos. O critério de agrupamento pode ser geométrico, ou uma propriedade da matriz, ou uma propriedade física. Diferentes particionamentos podem ser usados em diferentes vezes e partições podem ser mais particionadas. Particionamento geralmente conduz ao uso de paralelismo.

**Discretização** - O problema contínuo PDE é substituído por um problema finito com um conjunto de números reais como variáveis. As duas técnicas mais comuns são diferenças finitas e funções bases (elementos finitos, polinômios e expansões em séries). Métodos de diferenças finitas e funções bases com suporte local são muito bons para métodos paralelos na fase de discretização, por serem esses cálculos altamente independentes uns dos outros. Discretizações são também divididas em métodos de alta ordem e de baixa ordem. Métodos de alta ordem são vantajosos para paralelização pois mais de um trabalho pode ser feito na fase de discretização dos cálculos.



**Iteração** - Iterações é uma técnica de propósito geral que tem dificuldades (não linearidade, problemas muito grandes, dependência de tempo). Iteração é inerentemente sequencial e portanto, não é muito propícia para exploração de paralelismo. Ainda que iteração seja essencial para resolver muitos PDEs, é bom só introduzi-la quando o número de iterações for pequeno e o trabalho em cada iteração bem grande e que se possa dividi-lo em componentes paralelos.

Considere o estado constante de temperatura bi-dimensional no problema de distribuição na figura 5.3. A placa fina retangular é cercada por três lados por corrente de condensação (temperatura a 100 graus centígrados). O quarto lado é tocado por uma barra de gelo (temperatura a zero grau centígrado). Um cobertor isolante cobre de cima a placa. O problema é achar o estado constante de distribuição da temperatura em 100 pontos espaçados formando uma malha na placa de 10x10 pontos.

Este problema é um exemplo de equação diferencial parcial linear de segunda ordem. Quando o estado constante de distribuição da temperatura é encontrado, o conjunto de equações diferenciais relatam os valores das variáveis nos pontos vizinhos na malha:

$$\theta = \frac{\theta_{x-1,y} + \theta_{x,y-1} + \theta_{x+1,y} + \theta_{x,y+1}}{4}$$

aqui, as variáveis subscritas x e y referem-se as coordenadas dos pontos na malha. Este problema é bastante simples e poderia ser resolvido analiticamente, entretanto, PDEs mais complicados necessitam ser resolvidos iterativamente. Por esta razão será explorada a solução iterativa. A rotina iterativa inicia pelo assumir um valor inicial estimado para cada variável  $\theta_{x,y}$ . A equação diferencial é então, usada para calcular valores sucessivos. Se os valores das variáveis convergirem para a solução, a rotina terá sucesso.

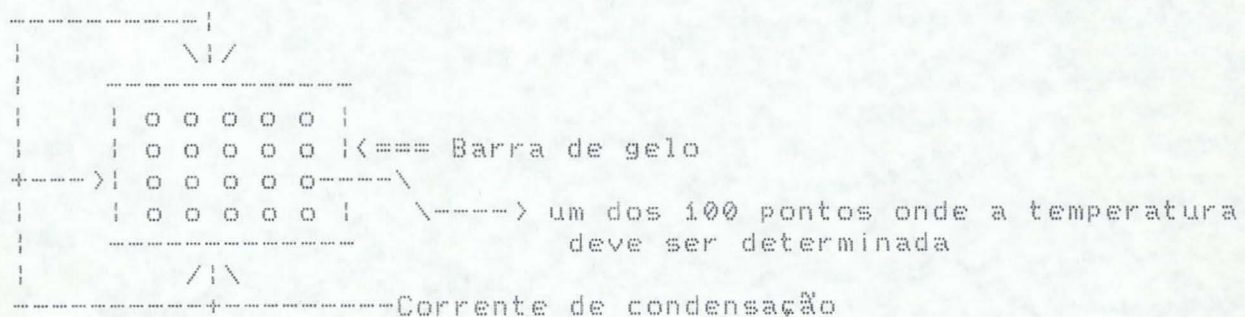


Fig. 4.3 Problema do estado cte de distribuição de temperatura



Uma rotina simples iterativa para o cálculo dos valores das variáveis  $\phi$  pode ser desenvolvida, onde todos os valores dos pontos da malha são calculados simultaneamente pela fórmula:

$$\phi' = (\phi_{x-1,y} + \phi_{x,y-1} + \phi_{x+1,y} + \phi_{x,y+1})/4$$

os valores  $\phi$  do lado direito da equação são os valores antigos, os valores  $\phi'$  representam os novos valores.

O método proposto por Jacobi (em 1845) é bastante propício a paralelização, pois todas as variáveis que necessitam ser acessadas por um processador estão disponíveis em um processador adjacente.

O mais popular método iterativo para a solução de PDEs em computadores sequenciais é o sobrerelaxação sucessivas (SOR) por pontos. O SOR difere do método de Jacobi em dois importantes aspectos:

- i) Um peso médio de  $\phi'_{x,y}$  e  $\phi^{old}_{x,y}$  é tomado para determinar um novo valor da variável  $\phi$ ;
- ii) Novos valores substituem os novos assim que calculados.

Num algoritmo sequencial pontos da malha são processados um por vez, coluna por coluna. A substituição dos valores antigos pelos novos é feita assim que os novos são calculados, aumentando a velocidade de convergência. Entretanto, este método sofre pela desvantagem que somente uma variável é calculada por vez.

Felizmente, uma variante paralela deste algoritmo retém esta velocidade de convergência. O algoritmo paralelo é referenciado como ordenação impar-par com aceleração de Chebyshev. Imagine os elementos processadores como se formando uma tábua de xadrez. Cada iteração tem duas fases, na primeira, todos os processadores impar recebem novos valores para suas respectivas variáveis. Na segunda fase, todos os processadores pares geram novos valores para suas variáveis. O algoritmo paralelo também muda o peso entre  $\phi'_{x,y}$  e  $\phi^{old}_{x,y}$  em cada meia iteração.

Observa-se que estes métodos iterativos de solução de PDEs são convenientes para máquinas do tipo processador matricial. Em 1982, Deminet implementou alguns algoritmos para solução de PDEs em máquinas multiprocessadoras. Ele resolveu a equação de Laplace com as condições de contorno de Dirichlet pelo método das diferenças finitas. A equação:

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0$$



é resolvida por uma malha de 150x150 pontos, usando técnicas iterativas similares as já descritas. Em cada iteração um novo valor por elemento é transmitido a média dos valores aos seus vizinhos.

Cada processo executa em sua própria CPU, achando os valores para uma subseção contínua da malha. Para fazer o cálculo dos índices de uma forma mais simples, a cada processo é atribuído um número completo de colunas e iterativamente calculados os valores das variáveis nessas colunas até os valores se estabilizarem.

A primeira providência feita no algoritmo por Deminet foi um ótimo trabalho de seleção de tarefa. O algoritmo original tanto atribuía subseções por processadores randômicos ou baseados na atribuição de números de módulos computacionais. O algoritmo proposto admite subseções que podem necessitar mais iterações, todas no meio da malha, para processadores rápidos, todas no mesmo conjunto de dados.

A segunda modificação é para distribuir dados entre o grupo contendo processadores ativos, com o objetivo de providenciar localização de referência.

#### 4.3 Solução de sistemas de equações

Cálculos matriciais estão entre as pedras angulares de computações científicas. Problemas de álgebra linear, envolvem sistemas de equações lineares, problemas lineares dos mínimos quadráticos e, problemas algébricos de autovalores são fundamentais para o cálculo da solução de equações diferenciais, problemas de otimização e de análise de várias estruturas discretas. O uso efetivo de computadores de arquiteturas paralelas em computações científicas é, entretanto, criticamente dependente da exploração do paralelismo em cálculos matriciais.

Algoritmos matriciais tem sido a vanguarda de desenvolvimento de algoritmos em multiprocessadores, não somente porque eles vão construindo blocos nos quais muitos outros cálculos científicos são baseados, mas também porque eles servem como protótipos reais que apresentam muitos dos desafios fundamentais de computações paralelas na forma pura. Por isso, o desenvolvimento de algoritmos paralelos para cálculo matricial tem recebido forte ênfase dos pesquisadores em processamento paralelo, tanto como uma ferramenta, quanto um paradigma para computação científica em geral em arquiteturas paralelas.

Os problemas computacionais mais comuns em álgebra linear são a solução de equações lineares e o problema algébrico de autovalores para vários tipos de matrizes. O problema das características afetam qualquer implementação computacional, incluindo matrizes quadradas ou retangulares; simétricas ou não; densas ou esparsas; explicitamente representada pelas suas entradas ou implicitamente representada pelas suas ações em



vetores. Tais propriedades determinam qual algoritmo ou família de algoritmo é apropriada para resolver o problema em qual ambiente computacional, sequencial ou paralelo, podendo sofrer maior impacto no ambiente paralelo. Por exemplo, a necessidade de linhas ou colunas intercambiarem-se para estabilidade numérica pode ser uma complicação mais séria ao ambiente paralelo do que ao sequencial. Outra questão importante é o tamanho da matriz, o que determina a quantidade de recursos computacionais que serão necessários, assim como que classes de algoritmos e estrutura de dados deve ser mais apropriada.

Os algoritmos paralelos para solução de sistemas de equações lineares podem ser por métodos diretos ou iterativos.

Fatoração de matrizes densas é um caso interessante de estudo para implementação paralela por causa que existem duas características que tendem a inibir a eficiência paralela. Elas são: restrições de precedência sequenciais, devido a sucessivas linhas, colunas e submatrizes da matriz fatorada, necessitam ser calculadas em ordem consecutiva e; comunicação global ser requerida porque cada sucessiva linha, coluna ou submatriz da matriz fatorada depender de todas precedentes linhas, colunas ou submatrizes.

Muitos desenvolvedores de algoritmos tem mostrado na prática que os cálculos podem ser suficientemente gerenciados e a comunicação suficientemente rápida de forma a atender a elevada utilização de processadores e eficiente paralelização.

As principais distinções entre algoritmos paralelos para fatoração de matrizes densas são a concorrência e a comunicação. O grau de concorrência atendida e o custo de comunicação são determinados pela escolha de granularidade. Uma implementação de granularidade fina, com subtarefas de complexidade de ordem um ( $O(1)$ ), como em algoritmos sistólicos e "topo de onda" potencialmente atendem o máximo de concorrência possível, mas este potencial não é alcançado, a menos que o gerenciamento de comunicação seja muito pequeno, com custo de comunicação correspondente ao custo de uma operação aritmética. Uma implementação com granularidade média, com subtarefas de complexidade de ordem  $n$  ( $O(n)$ ), é semelhante para providenciar um ótimo equilíbrio entre concorrência e custo de comunicação na maioria das arquiteturas multiprocessadoras de propósito geral.

Fatoração de matrizes reduz o problema de solução de um sistema de equações geral em um problema de solução de um sistema triangular de equações. Um sistema de equações lineares  $AX=B$  pode ser resolvido, por fatorização, em três passos:

- i) Decomposição da matriz dos coeficientes  $A$  no produto de uma matriz triangular inferior  $L$  e outra triangular superior  $U$ ;
- ii) Resolvendo o sistema  $LY=B$ ;
- iii) Resolvendo o sistema  $UX=Y$ .



Destes três passos, a rotina que triangulariza (i) tem alta complexidade computacional. O Algoritmo completo é dado no capítulo 6, sob o nome de método de Banachiewicz. Outro método existente lá é o de Cholesky.

Problemas esparsos tem tanto vantagens quanto desvantagens em relação ao caso denso. Sua principal vantagem é que a potencialidade esparsa leva a grande concorrência e menos comunicação devido a independência de dados de algumas partes do problema em relação a outras.

Métodos iterativos para solução de sistemas de equações lineares também se apresentam como um caso interessante de estudo para implementação em paralelo. Eles diferem dos métodos diretos, nos quais um conjunto fixo de cálculos precisam ser realizados em ordem para chegar a solução correta, a natureza autocorritiva dos métodos iterativos permite que se altere os cálculos para herdar paralelismos. O speedup de iterações individuais pelo acrescentar da concorrência não se beneficia, entretanto, facilita o cálculo da próxima iteração.

Métodos iterativos são frequentemente recomendados para implementações em paralelo por limitarem mais os requisitos de comunicação, em relação a comunicação global necessária em métodos diretos por fatoração.

Algoritmos para cálculo de autovalores de matrizes são menos desenvolvidos para máquinas paralelas do que dos para solução de sistemas de equações lineares. Uma das áreas que tem apresentado resultados satisfatórios para algoritmos paralelos para problemas de autovalores é no projeto de array sistólicos. Arrays sistólicos tem sido desenvolvidos para calcular autovalores e valores singulares de vários tipos de matrizes. Outra área de estudo que vem se desenvolvendo são os algoritmos paralelos baseados na divisão-e-conquista para cálculo de autovalores.



## 5 SOLUÇÃO DE SISTEMAS DE EQUAÇÕES IMPLEMENTADAS NO CONVEX

Neste capítulo são descritos alguns métodos implementados em fortran no Convex C210. Os métodos implementados referem-se a solução de sistemas de equações lineares.

A introdução dos métodos é feita através de uma breve descrição, seguida do algoritmo, do código fonte em fortran 77, da execução de um exemplo sem otimização e, da execução usando a opção de otimização e vetorização.

Os métodos foram implementados segundo o algoritmo desenvolvido para o sistema de software do Laboratório de Ensino e Pesquisa em Matemática Aplicada e Computacional (LEPMAC), algoritmos sequenciais. Nesses algoritmos não se utilizou as rotinas otimizadas da biblioteca VecLib, com o que se poderia obter melhores resultados. Este estudo de otimização ficará para um trabalho futuro.

Os exemplos tomados, inicialmente, foram exemplos acadêmicos, de pequeno porte, que visavam verificar a correção do algoritmo e a potencialidade da vetorização automática das rotinas e, na medida do possível, as vantagens produzidas por isto. A seguir foram desenvolvidos testes com matrizes de maior porte, onde se tentou quantificar a diferença de tempo de processamento entre as versões sequenciais e otimizadas.

Os exemplos são limitados, a implementação dos algoritmos não é e, nem pretende ser, a forma mais eficiente de se programar, mesmo porque isto exigiria um grande conhecimento não só do fortran 77, das transformações de reordenação (vistas no capítulo 3), como também da biblioteca VecLib, como ilustra o manual "Convex Training Advance Optimization", onde a execução pode chegar a vinte vezes menos que a execução normal.

Neste item são abordados métodos diretos (MEG com pivotamento simples e Gauss-Jordan com a matriz inversa); métodos compactos (método de Banachievicz e o método de Cholesky) e métodos iterativos (de Gauss-jacobi e Seidel).

### 5.1 Métodos Diretos

#### 5.1.1 Método de Eliminação de Gauss com pivotamento

Os métodos de Eliminação de Gauss, são métodos diretos de resolução de sistemas, que buscam calcular a solução exata do sistema através da realização de um número finito de operações aritméticas nos reais.

Esta versão tem duas etapas: a triangularização e a retrossubstituição. Na triangularização, as operações para transformar a matriz A em uma matriz triangular são da forma:



$$LEZ = \frac{-EZ}{ED} * LD + LEZ$$

onde:

EZ - Elemento a ser zerado;  
 ED - Elemento da diagonal principal, da mesma coluna que EZ;  
 LEZ - Linha que contém o elemento a ser zerado EZ;  
 LD - Linha que contém o elemento da diagonal ED.

A técnica do pivotamento é uma tentativa de minimizar os erros de arredondamento nas operações durante a triangularização da matriz, e conseqüentemente, minimizar a propagação deste erro para o resultado do sistema.

O pivotamento simples consiste em nomear um PIVÔ, um elemento escolhido, que ocupará a posição na diagonal principal e servirá de base nas operações elementares para zerar os demais elementos de sua coluna. Este elemento sempre é escolhido, considerando o elemento da diagonal principal para baixo e que seja o MAIOR em módulo, ou seja:

$$PIVÔ(j) = \max_{i=j \text{ até } N} ( | a_{ij} | )$$

A seguir é apresentado o algoritmo do método de eliminação de Gauss com pivotamento simples. Ele está escrito em uma linguagem lógica, próxima ao português.

```

inicio
Dim A(N,N), B(N), X(N), AB(N,N+1);
leia "Entre com a ordem N do sistema";N
para i=1 ate N
  para j=1 ate N faça leia A(i,j);
para i=1 ate N
  faça leia B(i);

%%%% Concatenação de A com B
para i=1 ate N
  faça inicio
    AB(i,N+1)=B(i)
    para j=1 ate N
      faça AB(i,j)=A(i,j);
  fim;

%%%% Triangularização
para i=1 ate N-1
  para j=i+1 ate N
    faça inicio
      PIVÔ(i)
      pivo= -AB(j,i)/AB(i,i)
      para k=1 ate N+1
        faça AB(j,k)=pivo*AB(i,k)+AB(j,k);
    fim;

```



```

%%%%%% Retrossubstituição
para i=N ate i incr -1
  faça inicio
    X(i)=AB(i,N+1)
    para j=i+1 ate N  faça X(i)=X(i)-AB(i,j)*X(j);
    X(i)= X(i)/AB(i,i);
  fim;

%%%%%% Impressão dos resultados
para i=1 ate N
  faça inicio
    para j=1 ate N  faça escreva A(i,j);
    escreva X(i),B(i)
  fim;

PIVO ( k:inteiro):
inicio
  lmaior=k;
  maior=abs(A(k,k));
  para m=k+1 ate N
    faça inicio
      Se maior <= abs(A(m,k))
        Então inicio
          maior=abs(A(m,k))
          lmaior=m;
        fim
      Se lmaior < k
        Então para U=1 ate N+1
          faça inicio
            aux= A(k,U);
            A(k,U)=A(lmaior,U);
            A(lmaior,U)= aux;
          fim;
        fim;
    fim;
  fim;
retorna
end.

```

Este algoritmo foi implementado em fortran77 e executado no Convex com o sistema de equações abaixo, produzindo a solução correta. A listagem do código-fonte é dada a seguir.

```

Program main
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
n=4
call init
call concab
call tngab
call retro
call out
end

```

d



```

Subroutine init
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
data a/3,9,-6,3,2,4,-3,4,1,-5,-4,3,-2,-4,12,9/
data b/-7,-11,36,22/
end

```

d

```

Subroutine Concab
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
do i=1,n
  ab(i,n+1)=b(i)
  do j=1,n
    ab(i,j)=a(i,j)
  enddo
enddo
end

```

d

```

Subroutine Tngab
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
do i=1,n
  do j=i+1,n
    call piv(j)
    pivo=-ab(j,i)/ab(i,i)
    do k=1,n+1
      ab(j,k)=pivo*ab(i,k)+ab(j,k)
    enddo
  enddo
enddo
end

```

d

```

Subroutine retro
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
do i=n,1,-1
  x(i)=ab(i,n+1)
  do j=i+1,n
    x(i)=x(i)-ab(i,j)*x(j)
  enddo
  x(i)=x(i)/ab(i,i)
enddo
end

```

d

```

Subroutine out
real a(4,4),b(4),x(4),ab(4,5)
integer n
common /data/a,b,ab,x,n
write(*,*) "Matriz A"
do i=1,4
  write(*,*) (a(i,j),j=1,4)

```



```

enddo
write(*,*)
write(*,*) "vetores B, X"
do i=1,4
  write(*,*) b(i),x(i)
enddo
end

```

d

```

Subroutine Piv(k)
real a(4,4),b(4),x(4),ab(4,5)
integer n,lmaior
common /data/a,b,ab,x,n
lmaior=k
maior=abs(ab(k,k))
do m=k+1,n
  if (maior .le. abs(ab(m,k))) then
    maior=abs(a(m,k))
    lmaior=m
  endif
  if (lmaior .ne. k) then
    do u=1,n+1
      aux=ab(k,u)
      ab(k,u)=ab(lmaior,u)
      ab(lmaior,u)=aux
    enddo
  endif
enddo
end

```

Este programa compilado sem opção de vetorização produziu a solução:

Matriz A

3.000000	2.000000	1.000000	-2.000000
9.000000	4.000000	-5.000000	-4.000000
-6.000000	-3.000000	-4.000000	12.000000
3.000000	4.000000	3.000000	9.000000

Vetores B e X

-7.000000	1.000000
-11.000000	-2.000000
36.000000	0.
22.000000	3.000000



### 5.1.2 Método de Gauss-Jordan com inversa

O Método de Gauss-Jordan, pode ser utilizado para o cálculo matrizes inversas e, então resolver o sistema usando-a. Parte-se da concatenação de A com a matriz identidade I, transformando a matriz A em identidade e, onde estava a identidade produz a matriz inversa.

Uma vez em posse da matriz inversa  $A^{-1}$  de A, pela álgebra se tem:

$$\text{Se } AX = B, \text{ então } A^{-1}(AX) = A^{-1}B$$

usando as propriedades algébricas das matrizes tem-se:

$$(A^{-1}A)X = I X = X$$

e, portanto:

$$X = A^{-1} * B$$

ou seja, o vetor solução X pode ser calculado, simplesmente multiplicando a inversa  $A^{-1}$  pelo vetor dos termos independentes B.

```
Program main
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
call init
call concainv
call tngainv
call montinv
call prodinvb
call out2
end

Subroutine init
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
data n/4/
data a/3,9,-6,3,2,4,-3,4,1,-5,-4,3,-2,-4,12,9/
data b/-7,-11,36,22/
end

d

Subroutine concainv
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
do i=1,n
  do j=1,n
    ainv(i,j)=a(i,j)
    ainv(i,n+1)=0.0
```



```

        enddo
        ainv(i,n+i)=1.0
    enddo
end
d
Subroutine tngainv
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
do i=1,n
    do j=i+1,n
        pivo=-ainv(j,i)/ainv(i,i)
        do k=1,n+n
            ainv(j,k)=pivo*ainv(i,k)+ainv(j,k)
        enddo
    enddo
enddo
do i=n,1,-1
    do k=i-1,1,-1
        pivo=-ainv(k,i)/ainv(i,i)
        do j=1,n+n
            ainv(k,j)=ainv(i,j)*pivo+ainv(k,j)
        enddo
    enddo
enddo
do i=1,n
    e1=ainv(i,i)
    do j=1,n+n
        ainv(i,j)=ainv(i,j)/e1
    enddo
enddo
end

```

```

d
Subroutine montinv
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
do i=1,n
    do j=1,n
        inv(i,j)=ainv(i,n+j)
    enddo
enddo
end

```

```

d
Subroutine prodinvb
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
do i=1,n
    x(i)=0.0
    do j=1,n
        x(i)=inv(i,j)*b(j)+x(i)
    enddo
enddo
end

```



```

Subroutine out2
real a(4,4),b(4),x(4),inv(4,4),ainv(4,8)
integer n
common /data/ a,b,x,inv,ainv,n
write(*,*) "Matriz A"
do i=1,n
  write(*,*) (a(i,j),j=1,n)
enddo
write(*,*)
write(*,*) "Matriz inversa"
do i=1,n
  write(*,*) (inv(i,j),j=1,n)
enddo
write(*,*)
write(*,*) "Vetor B", (b(i),i=1,n)
write(*,*)
write(*,*) "Vetor Solucao", (x(i),i=1,n)
end

```

A solução produzida pelo programa é dada a seguir.

Matriz A

3.000000	2.000000	1.000000	-2.000000
9.000000	4.000000	-5.000000	-4.000000
-6.000000	-3.000000	-4.000000	12.000000
3.000000	4.000000	3.000000	9.000000

Matriz inversa

-4.798611	0.5763888	-1.263889	0.8749999
7.458333	-0.7916666	1.916667	-1.250000
-1.770833	0.1041667	-0.5416666	0.3750000
-1.125000	0.1250000	-0.2500000	0.2500000

Vetor B -7.000000 -11.00000 36.00000 22.00000

Vetor solucao 0.9999962 -2.000006 1.9073486E-06 3.000000



## 5.2 Métodos Compactos

Os métodos compactos são equivalentes ao Método de Eliminação de Gauss. Neles a matriz A é decomposta em duas matrizes triangulares L e U, onde L é triangular inferior e U triangular superior. Isto reduz o problema  $AX = B$  a dois sistemas simples  $LY = B$  e  $UX = Y$ .

Os métodos compactos empregam, em geral, o mesmo número de operações que os métodos de Gauss, ou seja

$1/6 n(n-1)(2n+5)$       adições e subtrações

$1/6 n(n-1)(2n+5)$       multiplicações

$1/2 n(n+1)$               divisões

Entre os métodos compactos temos os métodos de Banachiewicz e de Cholesky.

### 5.2.1 Método de Banachiewicz

O método de Banachiewicz, também é conhecido como o método de Decomposição LU, pois a matriz A é decomposta nas matrizes triangulares L e U, tais que  $L*U = A$ , onde:

$$L = \begin{bmatrix} L_{11} & 0 & 0 & \dots & 0 \\ L_{21} & L_{22} & 0 & \dots & 0 \\ L_{31} & L_{32} & L_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ L_{n1} & L_{n2} & L_{n3} & \dots & L_{nn} \end{bmatrix} \quad U = \begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & u_{2n} \\ 0 & 0 & 1 & \dots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

A resolução do sistema  $AX = B$ , se resume à resolução dos sistemas:

$$LY = B \quad \text{e} \quad UX = Y,$$

que podem ser calculados facilmente por retrossubstituição.

Para se determinar os valores das componentes de L e U, utilizaremos as fórmulas:

Para  $i=1$  até N

Para  $j=i$  até N

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} * U_{kj} ;$$

Para  $j=1$  até  $i-1$

$$L_{ij} = (A_{ij} - \sum_{k=1}^{i-1} L_{ik} * U_{kj}) / U_{jj}$$



Como L e U são triangulares, ambas podem ser armazenadas na mesma matriz LU, ficam as fórmulas para decomposição reduzidas a:

Para i=1 até N  
 Para j=i até N

$$LUIj = Aij - \sum_{k=1}^{i-1} LUIk * LUKj ;$$

Para j=1 até i-1

$$LUIj = (Aij - \sum_{k=1}^{i-1} LUIk * LUKj) / LUJj$$

Algoritmo do método de Banachiewicz

Início

Dim A(N,N), B(N),X(N),LU(N,N),Y(N)

%%% Decomposição de A e cálculo de LU

Para i=1 até N

Faça início

Para j=i até N

Faça início

sum = 0.0

Para k=1 até i-1

Faça sum = sum + LU(i,k)\*LU(k,j);

LU(i,j) = A(i,j)-sum

fim

Para J=1 até i-1

Faça início

sum = 0.0

Para k=1 até i-1

faça sum = sum +LU(i,k)\*LU(k,j);

LU(i,j) = (A(i,j)-sum)/LU(j,j)

fim

%%% Resolução do sistema LY =B

Para i=1 até N

faça início

sum = 0.0

Para j=1 ate i-1

Faça sum = sum + LU(i,j)\*Y(j);

Y(i) = B(i)-sum

fim

%%% Resolução do sistema UX = Y

Para i=N até 1 incr -1

faça início

sum = 0.0

Para j=N até i+1 incr -1

faça sum = sum + LU(j,i) \* X(j);

X(i) = ( Y(i) -sum) / LU(i,i)

fim

%%% Impressão dos resultados

Para I=1 até N

faça escreva X(i);

fim.

A implementação no computador Convex em fortran é dada a seguir, juntamente com o resultado produzido.

```
Program main
real a(4,4),b(4),x(4),lu(4,4),y(4)
integer n
d inicializacao de dados
data n/4/
data a/3,9,-6,3,2,4,-3,4,1,-5,-4,3,-2,-4,12,9/
data b/-7,-11,36,22/
d montagem da LU
do i=1,n
do j=1,n
lu(i,j)=a(i,j)
enddo
enddo
d Decomposicao LU
do i=1,n
do j=1,i
sum=0.0
do k=1,j-1
sum=sum+lu(i,k)*lu(k,j)
enddo
lu(i,j)=lu(i,j)-sum
enddo
do j=i+1,n
sum=0.0
do k=1,i-1
sum=sum+lu(i,k)*lu(k,j)
enddo
lu(i,j)=(lu(i,j)-sum)/lu(i,i)
enddo
enddo
d Resolucao do sistema LY=B
do i=1,n
sum=0.0
do j=1,i-1
sum=sum+lu(i,j)*y(j)
enddo
y(i)=(b(i)-sum)/lu(i,i)
enddo
d Resolucao do sistema UX=Y
do i=n,1,-1
sum=0.0
do j=i+1,n
sum=sum+lu(i,j)*x(j)
enddo
x(i)=y(i)-sum
enddo
d Impressao de resultados
write(*,*) "Matriz A"
do i=1,n
write(*,*) (a(i,j),j=1,n)
enddo
write(*,*)
```



```

write(*,*) "Matriz LU"
do i=1,n
  write(*,*) (lu(i,j),j=1,n)
enddo
write(*,*)
write(*,*) "Vetor B", (b(i),i=1,n)
write(*,*) "Vetor Y", (y(i),i=1,n)
write(*,*) "Vetor X", (x(i),i=1,n)
end

```

Matriz A

3.000000	2.000000	1.000000	-2.000000
9.000000	4.000000	-5.000000	-4.000000
-6.000000	-3.000000	-4.000000	12.000000
3.000000	4.000000	3.000000	9.000000

Matriz LU

3.000000	0.66666666	0.33333333	-0.66666666
9.000000	-2.000000	4.000001	-1.000000
-6.000000	0.9999998	-6.000000	-1.500000
3.000000	2.000000	-6.000002	3.999997

Vetor b	-7.000000	-11.00000	36.00000	22.00000
Vetor y	-2.333333	-5.000001	-4.500000	3.000001
Vetor x	1.000003	-2.000004	9.5367432E-07	3.000001

### 5.2.2 Método de Cholesky

O método de Cholesky é um método que não requer muita memória para sua implementação e é econômico em termos de tempo de computação.

A base da utilização do método é que a matriz dos coeficientes A concatenada com o vetor B, que formam a matriz estendida AB, pode ser reduzida a um sistema onde a matriz seja triangular superior U da forma:

$$U = \begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} & u_{1n+1} \\ 0 & 1 & u_{23} & \dots & u_{2n} & u_{2n+1} \\ 0 & 0 & 1 & \dots & u_{3n} & u_{3n+1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & u_{nn+1} \end{bmatrix}$$

Se esta forma poder ser achada, então a solução do sistema pode ser facilmente calculada por retrossubstituição, a partir da última equação.

Uma forma de calcular a matriz triangular superior U é utilizar o MEG. Outra forma, é calcular a matriz transformação L que, quando multiplicada pela matriz U, a transforma na matriz original A. Pode-se mostrar que a matriz L é triangular inferior e na forma:

$$L = \begin{bmatrix} L_{11} & 0 & 0 & \dots & 0 \\ L_{21} & L_{22} & 0 & \dots & 0 \\ L_{31} & L_{32} & L_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ L_{n1} & L_{n2} & L_{n3} & \dots & L_{nn} \end{bmatrix}$$

isto resulta que  $L*U = AB$ , ou seja (considerando um sistema de ordem 4):

$$\begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix} * \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{45} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & : & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & : & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & : & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & : & a_{45} \end{bmatrix} = AB.$$

Como a matriz U é triangular superior e sua diagonal principal é composta por um e a matriz L também é triangular, mas inferior, ambas podem ser armazenadas numa mesma matriz LU de ordem (N, N+1), inicializada pela matriz A concatenada ao vetor B. Efetua-se então,

a) Para primeira linha, considerando  $j = 2, \dots, N+1$

$$LU_{1j} = LU_{1j}/LU_{11}$$



b) Para a m-ésima coluna e linha, onde  $m = 2, \dots, N$ .

$$LU_{im} = LU_{im} - \sum_{k=1}^{m-1} LU_{ik} * LU_{km}, \text{ para } i = m, \dots, n$$

$$LU_{mj} = \frac{LU_{mj} - \sum_{k=1}^{m-1} LU_{mk} * LU_{kj}}{LU_{mm}}, \text{ para } j = m+1, \dots, n+1$$

Uma vez terminado, inicia-se a retrossubstituição para a determinação do vetor solução através das operações:

$$X_n = LU_{n, n+1}$$

$$X_i = LU_{i, n+1} - \sum_{k=i+1}^n LU_{ik} * X_k, \text{ onde } i = n-1, \dots, 1.$$

Algoritmo do método de Cholesky

Início

leia "Entre com a ordem do sistema";N

Dim A(N,N),B(N),X(N),LU(N,N+1)

Para i=1 ate N

Para j=1 ate N

Faça Leia A(i,j);

Para i=1 ate N

Faça leia B(i);

%%% Montagem de LU

Para i=1 ate N

Faça inicio

Para j=i ate N

Faça LU(i,j)= A(i,j);

LU(i,N+1) = B(i);

fim

%%% Cálculo da i linha de U

Para j=2 ate N+1

faça LU(i,j)= LU(i,j)/LU(i,i);

Para m=2 ate N

faça inicio

para i=M até N

faça inicio

sum=0

Para k=i ate m-1

faça sum = sum +LU(i,k)\*lu(k,m);

LU(i,m) = LU(i,m) -sum;

fim

```

%%%% Cálculo da linha m
  Para j= m+1 ate N+1
    faça inicio
      sum = 0;
      para k=i ate M-1
        faça sum = sum + LU(m,k)*LU(k,j);
      LU(m,j)= (LU(m,j)-sum)/LU(m,m);
    fim
  fim
ESCREVA AS MATRIZES A, B, L, e UX=B'
%%%% Retrossubstituição
X(N) = LU(N,N+1);
Para M=i ate N-1
  faça inicio
    I= N-m
    sum =0;
    para j= I+1 ate N
      faça sum =sum + LU(i,j)*X(j);
    X(I) = LU(I,N+1)-sum;
  fim
ESCREVA X
fim.

```

A seguir é apresentado o programa fortran que implementa o método no Convex e sua execução.

```

d metodo de cholesky
  Program main
  real a(4,4),b(4),x(4),lu(4,5)
  integer n
d inicializacao de dados
  data n/4/
  data a/3,9,-6,3,2,4,-3,4,1,-5,-4,3,-2,-4,12,9/
  data b/-7,-11,36,22/
d montagem da LU
  do i=1,n
    do j=1,n
      lu(i,j)=a(i,j)
    enddo
    lu(i,n+1)=b(i)
  enddo
d calculo da 1 linha de lu
  do j=2,n+1
    lu(1,j)=lu(1,j)/lu(1,1)
  enddo
d calculo das demais linhas e colunas
  do m=2,n
    do i=m,n
      sum1=0.0
      do k=i,m-1
        sum1=sum1+lu(i,k)*lu(k,m)
      enddo
      lu(i,m)=lu(i,m)-sum1
    enddo
  enddo

```



```

        do j=m+1,n+1
            sum2=0.0
            do k=1,m-1
                sum2=sum2+lu(m,k)*lu(k,j)
            enddo
            lu(m,j)=(lu(m,j)-sum2)/lu(m,m)
        enddo
    enddo
d  retrosubstituicao
    x(n)=lu(n,n+1)
    do m=1,n-1
        i=n-m
        sum=0.0
        do j=i+1,n
            sum=sum+lu(i,j)*x(j)
        enddo
        x(i)=lu(i,n+1)-sum
    enddo
d Impressao de resultados
write(*,*) "Matriz A"
do i=1,n
    write(*,*) (a(i,j),j=1,n)
enddo
write(*,*)
write(*,*) "Matriz LU"
do i=1,n
    write(*,*) (lu(i,j),j=1,n+1)
enddo
write(*,*)
write(*,*) "Vetor B", (b(i),i=1,n)
write(*,*) "Vetor X", (x(i),i=1,n)
end

```

Matriz A

3.000000	2.000000	1.000000	-2.000000
9.000000	4.000000	-5.000000	-4.000000
-6.000000	-3.000000	-4.000000	12.000000
3.000000	4.000000	3.000000	9.000000

Matriz LU

3.000000	0.6666666	0.3333333	-0.6666666	-2.333333
9.000000	-2.000000	4.000001	-1.000000	-5.000001
-6.000000	0.9999998	-6.000000	-1.500000	-4.500000
3.000000	2.000000	-6.000002	3.999997	3.000001

Vetor b	-7.000000	-11.00000	36.00000	22.00000
Vetor x	1.000003	-2.000004	9.5367432E-07	3.000001

### 5.1.3 Métodos Iterativos

Um método é dito iterativo quando a solução  $X$  é obtida como o limite de uma seqüência de aproximações sucessivas  $X_1, X_2, X_3, \dots, X_m$ .

Os métodos diretos são normalmente mais eficientes, mas no caso de matrizes esparsas se mostram menos eficientes. Enquanto que os métodos iterativos, se mostram eficientes, pois não requerem tanta memória para armazenar a matriz (caso sejam utilizadas técnicas de armazenamento de matrizes esparsas).

O cálculo da exatidão em métodos iterativos, é feito pelo cálculo do número de Algarismos Significativos Corretos, componente a componente, podendo ser dado individualmente, ou sendo dado o menor de todos.

Convenciona-se uma notação para descrever as componentes do vetor solução. a cada iteração, utiliza-se a notação  $x_1, x_2, \dots, x_n$  para descrever as componentes; utiliza-se um índice superior, entre parênteses para denotar a iteração da aproximação.

EXEMPLOS:  $\langle 0 \rangle$   
 $x_1$  - componente  $x_1$ , valor inicial;  
 $\langle 2 \rangle$   
 $x_1$  - componente  $x_1$ , na segunda iteração;  
 $\langle 2 \rangle$   
 $x_2$  - componente  $x_2$ , na segunda iteração;  
 $\langle k \rangle$   
 $x_i$  - componente  $x_i$ , na  $k$ -ésima iteração.

Para os métodos iterativos foram testados, inicialmente, pelo sistema de equações de ordem 6, o qual é apresentado a seguir.

$$\begin{aligned} & \{ 10x + 2y - 2s = 5 \\ & \{ x + 10y + z = 10 \\ & \{ 2x + 20z + t = 10 \\ & \{ 3y + 30r + 3s = 0 \\ & \{ 2t - 2r + 20s = 5 \\ & \{ z + 10t - r = 0 \end{aligned}$$

### 5.3.1 Método de Gauss-Jacobi

O método de Jacobi, utiliza uma "função vetorial de iteração"  $G(X)$ , para o cálculo das iterações. Os valores iniciais são utilizados para calcular toda a primeira iteração e, estes, para os cálculos da segunda iteração. Portanto, os valores da iterações são calculados pelos valores da iteração anterior, ou seja:

$$X^{(k+1)} = G(X^{(k)}), \text{ onde } G(X) \text{ é da forma:}$$



$$G(X^{(k+1)}) = \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \dots - a_{1n}x_n^{(k)})/a_{11} \\ (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \dots - a_{2n}x_n^{(k)})/a_{22} \\ \vdots \\ (b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - \dots - a_{nn}x_{n-1}^{(k)})/a_{nn} \end{pmatrix}$$

Este método também é conhecido como método da substituição simultânea, pois todos os valores são atualizados ou substituídos ao mesmo tempo, no final do cálculo de cada iteração.

Uma desvantagem deste método, é que necessitamos duplicar o vetor solução, um para o valor recente calculado, outro para o valor da iteração anterior. Isto não é só um uso ineficiente do espaço, como também ocasiona uma convergência mais lenta.

#### Algoritmo de Jacobi

```

Início
  Dim A(N,N), B(N), X(N), Xnovo(N)
  ascmin=0
  ascreq= B <valor arbitrário>
  iteração = 0
  limit = 15 <valor arbitrário>
LEITURA DA MATRIZ A E DO VETOR B;
  Para i=1 ate N   faça X(i)=1.0;

  Enquanto ascmin < ascreq e iteração <= limit
  Faça início
    Para i=1 ate N
      faça início
        sum=0
        Para j=1 ate N
          Faça Se i < j
            Então sum= sum+ a(i,j)*X(j);
        Xnovo(i) = (b(i) -sum)/a(i,i)
        asc=- (0.3+Log(0.5*10^(-8)+Abs((Xnovo(i)-x(i))/Xnovo(i))))
        Se i=1
          Então ascmin=asc
          Senão Se ascmin>asc
            Então ascmin= asc;
      fim
    Para j=1 ate N
      Faça escreva X(j);
      iteração = iteração +1;
      escreva ascmin, iteração
    Para j=1 ate N
      faça X(j)=Xnovo(j)
  fim enquanto;

```

```

Se iteração = limit e ascmin < ascreq
Então inicio
  Escreva "Não convergiu em";iteração;"iterações"
  Escreva "Ultima iteração "
  Para i=1 ate N
    Faça escreva X(i);
  escreva ascmin
fim
Senão inicio
  escreva "Solução final";
  escreva iteração
  Para i=1 ate N Faça escreva X(i);
  Escreva ascmin
fim
fim.

```

Programa em fortran no Convex, para o método iterativo de Gauss-Jacobi, com o exemplo de ordem seis.

```

d metodo de gauss-jacobi
Program main
real a(6,6),b(6),x(6),asc(6),xnovo(6)
integer it,limit,n
real e1,e2,e3,ascreq,mi
data n/6/
data ascmin/0.0/
data it/0/
data ascreq/8.0/
data limit/50/
data a/10,2,0,-2,0,0,1,10,1,0,0,0,2,0,20,0,0,1,
*0,3,0,30,3,0,0,0,0,-2,20,2,0,0,1,-1,0,10/
data b/5,10,10,0,5,0/
data x/6*1.0/
mi=0.000000005
do i=1,n
  write(*,10)(a(i,j),j=1,n),b(i)
enddo
10 format(6f8.4,f12.4)
do while ((ascmin.lt.ascreq).and.(it.le.limit))
  write(*,20) it,(x(i),i=1,n),ascmin
  do i=1,n
    sum=0.0
    do j=1,n
      if (i.ne.j) sum=sum+a(i,j)*x(j)
    enddo
    xnovo(i)=(b(i)-sum)/a(i,i)
    e2=(xnovo(i)-x(i))/xnovo(i)
    e3=abs(e2)+mi
    asc(i)=-(0.3+log10(e3))
    if (asc(i) .le. 0.0) asc(i)=0.0
    if (i .eq. 1) then
      ascmin=asc(i)
    else if (ascmin .ge. asc(i)) then
      ascmin=ac(i)
    endif
  enddo
enddo

```



```

        enddo
        it=it+1
20    format(i2,6f12.8,f6.2)
        do j=1,n
            x(j)=xnovo(i)
        enddo
    enddo
write(*,20) it,(x(i),i=1,n),ascmin
end

```

Execução:

Matriz A e vetor B

```

10.0000  1.0000  2.0000  0.0000  0.0000  0.0000      5.0000
 2.0000 10.0000  0.0000  3.0000  0.0000  0.0000     10.0000
 0.0000  1.0000 20.0000  0.0000  0.0000  1.0000     10.0000
-2.0000  0.0000  0.0000 30.0000 -2.0000 -1.0000      0.0000
 0.0000  0.0000  0.0000  3.0000 20.0000  0.0000      5.0000
 0.0000  0.0000  1.0000  0.0000  2.0000 10.0000      0.0000

```

Solução:

```

0 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 0.0
1 0.20000000 0.50000000 0.40000000 0.16666666 0.10000000 -0.30000000 0.0
2 0.37000000 0.91000000 0.49000000 0.01000000 0.22500000 -0.06000000 0.0
3 0.31099999 0.92299999 0.45749999 0.03766666 0.24849999 -0.09400000 0.0
4 0.31620000 0.92650000 0.45854999 0.03416666 0.24435000 -0.09545000 0.6
5 0.31564000 0.92651000 0.45844750 0.03418833 0.24487500 -0.09472500 1.8
6 0.31565950 0.92661550 0.45841070 0.03421010 0.24487170 -0.09481970 2.7
7 0.31565630 0.92660500 0.45841020 0.03420800 0.24486840 -0.09481540 3.9
8 0.31565740 0.92660620 0.45841050 0.03420780 0.24486870 -0.09481470 4.7
9 0.31565720 0.92660661 0.45841040 0.03420790 0.24486880 -0.09481480 5.1
10 0.31565730 0.92660661 0.45841040 0.03420790 0.24486880 -0.09481480 6.3
11 0.31565720 0.92660661 0.45841040 0.03420790 0.24486880 -0.09481480 6.7
12 0.31565720 0.92660661 0.45841040 0.03420790 0.24486880 -0.09481480 6.3
12 0.31565720 0.92660661 0.45841040 0.03420790 0.24486880 -0.09481480 8.0

```

### 5.3.2 Método de Gauss-Seidel

O método de Gauss-Seidel também é conhecido como método das sucessivas substituições, devido ao fato que a "função vetorial de iteração"  $G(X)$ , utiliza sempre o valor mais recente, ou seja, a medida que um valor é calculado, ele é utilizado para o cálculo dos demais valores. Este procedimento, em geral, acelera a convergência em relação ao método de Jacobi, além de não necessitar a duplicação de espaço para armazenar o vetor solução.

A função vetorial de iteração  $G(X)$  é da forma:

```
( <k+1>      <k>      <k>      <k>      )
( x1      =(b1 -a12x2  -a13x3  - ... -a1nxn  )/a11  )
( )
( <k+1>      <k+1>      <k>      <k>      )
( x2      =(b2 -a21x1  -a23x3  - ... -a2nxn  )/a22  )
( . . .      . . .      . . .      . . .      )
( )
( <k+1>      <k+1>      <k+1>      <k+1>      )
( xn      =(bn -an1x1  -an2x2  - ... -ann-1xn-1 )/ann  )
```

Algoritmo do método de Gauss-Seidel -Sistemas Esparsos

Início

```
Dim A(N,N), B(N), X(N), Xnovo
ascmin=0
ascreq= 8 <valor arbitrário>
iteração = 0
limit = 15 <valor arbitrário>
```

LEITURA DA MATRIZ A E DO VETOR B;

Para i=1 ate N Faça X(i)=1.0;

Enquanto ascmin < ascreq e iteração <= limit

Faça inicio

Para i=1 ate N

faça inicio

sum=0

Para j=1 ate N

Faça Se i < j Então sum= sum+ a(i,j)\*X(j);

Xnovo = (b(i) -sum)/a(i,i)

asc=-(<math>0.3 + \text{Log}(0.5 \* 10^{(-8)} + \text{Abs}((Xnovo-x(i))/Xnovo))</math>)

X(i) = Xnovo

Se i=1

Então ascmin=asc

Senão Se ascmin>asc Então ascmin= asc;

fim

Para j=1 ate N Faça escreva X(j);

iteração = iteração +1;

escreva ascmin, iteração

fim enquanto;

Se iteração = limit e ascmin < ascreq

Então inicio

Escreva "Não convergiu em";iteração;"iterações"

Escreva "Ultima iteração "

Para i=1 ate N Faça escreva X(i);

escreva ascmin

fim

Senão inicio

escreva "Solução final";

Para i=1 ate N Faça escreva X(i);

Escreva ascmin

fim

fim.



Programa em fortran no Convex, para o método iterativo de Gauss-Seidel com o exemplo de ordem seis.

```
d metodo de gauss-seidel
  Program main
  real a(6,6),b(6),x(6),asc(6)
  integer it,limit,n
  real ascreq,mi,xnovo,ascmin
  data n/6/
  data ascmin/0.0/
  data it/0/
  data ascreq/8.0/
  data limit/50/
  data a/10,2,0,-2,0,0,1,10,1,0,0,0,2,0,20,0,0,1,
*0,3,0,30,3,0,0,0,-2,20,2,0,0,1,-1,0,10/
  data b/5,10,10,0,5,0/
  data x/6*1.0/
  mi=0.000000005
  do i=1,n
    write(*,10)(a(i,j),j=1,n),b(i)
  enddo
10 format(6f8.4,f12.4)
  do while ((ascmin.lt.ascreq).and.(it.le.limit))
    write(*,20) it,(x(i),i=1,n),ascmin
    do i=1,n
      sum=0.0
      do j=1,n
        if (i.ne.j) sum=sum+a(i,j)*x(j)
      enddo
      xnovo=(b(i)-sum)/a(i,i)
      e2=(xnovo-x(i))/xnovo
      e3=abs(e2)+mi
      asc(i)=-(0.3+log10(e3))
      if (asc(i) .le. 0.0) asc(i)=0.0
      x(i)=xnovo
      if (i .eq. 1) then
        ascmin=asc(i)
      else if (ascmin .ge. asc(i)) then
        ascmin=asc(i)
      endif
    enddo
    it=it+1
20 format(i2,6f12.8,f6.2)
  enddo
  write(*,20) it,(x(i),i=1,n),ascmin
end
```

Execução:

10.0000	1.0000	2.0000	0.0000	0.0000	0.0000	5.0000
2.0000	10.0000	0.0000	3.0000	0.0000	0.0000	10.0000
0.0000	1.0000	20.0000	0.0000	0.0000	1.0000	10.0000
-2.0000	0.0000	0.0000	30.0000	-2.0000	-1.0000	0.0000
0.0000	0.0000	0.0000	3.0000	20.0000	0.0000	5.0000
0.0000	0.0000	1.0000	0.0000	2.0000	10.0000	0.0000

Soluções:

0	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	0.0
1	0.20000000	0.65999999	0.41700000	0.11333333	0.23299999	-0.08830000	0.0
2	0.35060000	0.89588000	0.45962100	0.03596333	0.24460540	-0.09488320	0.0
3	0.31848770	0.92551340	0.45846840	0.03437670	0.24484340	-0.09481550	0.7
4	0.31575490	0.92653590	0.45841390	0.03421270	0.24486800	-0.09481500	1.7
5	0.31566360	0.92660350	0.45841060	0.03420820	0.24486870	-0.09481480	3.2
6	0.31565750	0.92660590	0.45841040	0.03420790	0.24486880	-0.09481480	4.4
7	0.31565730	0.92660510	0.45841040	0.03420790	0.24486880	-0.09481480	5.8
8	0.31565720	0.92660610	0.45841040	0.03420790	0.24486880	-0.09481480	6.3
9	0.31565720	0.92660610	0.45841040	0.03420790	0.24486880	-0.09481480	9.0



## 6 RESULTADOS E CONCLUSÕES

Juntamente com esse trabalho desenvolveu-se um estudo sobre a utilização do supercomputador CONVEX C210, disponível na UFSC. Como resultado deste estudo, produziu-se um roteiro básico para novos usuários da máquina Convex. Esse roteiro abrange a descrição da máquina, o sistema operacional incluindo seus principais comandos, formas de acesso, o editor de texto vi, o compilador Fortran e a biblioteca VecLib. Esse roteiro foi publicado separadamente e encontra-se disponível sob título: "Máquina Convex C210 primeiros passos: utilização e programação".

Desenvolveu-se, ainda, um estudo sobre processamento vetorial e paralelo, o qual, também pode servir como uma introdução ao processamento vetorial e paralelo em disciplinas que tratem de algoritmos vetoriais e paralelos.

Foram, efetivamente testados alguns métodos de resolução de sistemas de equações lineares, como os métodos compactos de Banachievsk e Choleski e os métodos iterativos de Gauss-Jacobi e Gauss-Seidel. Estes métodos haviam sido implementados com sucesso no sistema de software do laboratório LEPMAC (Laboratório de Ensino e Pesquisa em Matemática Aplicada e Computacional), desenvolvido no instituto de informática e no PGCC da UFRGS, em ambientes de microcomputadores do tipo PCxt. Apesar destes métodos não serem específicos para sistemas de grande porte, eles foram utilizados para evidenciar duas características, dois "paradigmas": tempo de processamento e vetorização.

Os testes foram desenvolvidos para sistemas de equações onde a matriz dos coeficientes é densa e para matrizes esparsas. Para sistemas com matrizes densas utilizou-se os métodos compactos e para sistemas com matrizes esparsas usou-se métodos iterativos.

No teste de matrizes densas, utilizou-se matrizes de Hilbert, que apesar de malcondicionadas tem solução. Foram testadas matrizes de ordem  $N=100, 200, 300, 400, 500, 600, 700, 800, 900$  e  $1000$ . Para cada sistema foi tomado o tempo de processamento escalar e com a otimização vetorial, para se fazer a taxa de otimização de cada caso, como consta nas figuras 6.1 e 6.2 a seguir.

Métodos Compactos - tempo médio de processamento (segundos)			
ORDEM DA MATRIZ	BANACHIEVICZ sem otimização	BANACHIEVICZ com vetorização	TAXA DE REDUÇÃO
100	0,615	0,110	5,591 %
200	4,758	0,724	6,572 %
300	15,880	2,246	7,070 %
400	37,434	5,111	7,324 %
500	72,877	9,732	7,488 %
600	126,743	16,531	7,667 %
700	199,999	26,077	7,670 %
800	297,618	38,774	7,676 %
900	429,554	54,818	7,836 %
1000	592,702	73,573	8,056 %

Fig.6.1 Quadro comparativo do método de Banachievicz

Métodos Compactos - tempo médio de processamento (segundos)			
ORDEM DA MATRIZ	CHOLESKY sem otimização	CHOLESKY com vetorização	TAXA DE REDUÇÃO
100	0,629	0,109	5,771 %
200	4,884	0,718	6,802 %
300	16,293	2,243	7,264 %
400	38,393	5,102	7,525 %
500	75,092	9,734	7,714 %
600	129,458	16,540	7,827 %
700	204,722	26,319	7,778 %
800	309,354	38,789	7,975 %
900	443,016	54,740	8,093 %
1000	611,329	73,709	8,294 %

Fig.6.2 Quadro comparativo do método de Cholesky

Os métodos de eliminação de Gauss e Gauss-Jordan, com matrizes de Hilbert de ordem elevada ( $N > 50$ ) não puderam ser testados devido a perda de exatidão o que provocou divisão por zero na triangualização e retrosubstituição, portanto foram testadas apenas sistemas de ordem 3 a 50. (figuras 6.3 e 6.4)



METODO DE ELIMINAÇÃO DE GAUSS							
ORD	ESCALAR	VETORIAL	TAXA	ORD	ESCALAR	VETORIAL	TAXA
03	0.000122	0.000095	1.284%	27	0.345529	0.143166	2.413%
04	0.000225	0.000141	1.596%	28	0.400492	0.165755	2.416%
05	0.000515	0.000263	1.958%	29	0.461156	0.190650	2.419%
06	0.000935	0.000480	1.948%	30	0.528531	0.218518	2.419%
07	0.001668	0.000810	2.059%	31	0.603081	0.249083	2.421%
08	0.002774	0.001307	2.122%	32	0.685301	0.283006	2.422%
09	0.004395	0.001994	2.204%	33	0.776107	0.320114	2.424%
10	0.006605	0.002957	2.234%	34	0.875187	0.361032	2.424%
11	0.009567	0.004242	2.255%	35	0.981391	0.405189	2.429%
12	0.013437	0.005901	2.277%	36	1.104445	0.454343	2.431%
13	0.018467	0.007978	2.315%	37	1.233961	0.507167	2.433%
14	0.024848	0.010689	2.325%	38	1.374623	0.564849	2.434%
15	0.032711	0.013950	2.345%	39	1.528219	0.626942	2.438%
16	0.042231	0.018014	2.344%	40	1.692204	0.695089	2.435%
17	0.053820	0.022843	2.356%	41	1.872751	0.768176	2.438%
18	0.067615	0.028651	2.360%	42	2.063411	0.847309	2.435%
19	0.083943	0.035457	2.367%	43	2.270007	0.931347	2.437%
20	0.103052	0.043426	2.373%	44	2.494243	1.022780	2.439%
21	0.125383	0.052593	2.384%	45	2.732690	1.120046	2.440%
22	0.151137	0.063326	2.387%	46	2.985720	1.224499	2.438%
23	0.181134	0.075451	2.401%	47	3.259621	1.335215	2.441%
24	0.214926	0.089504	2.401%	48	3.547190	1.454015	2.440%
25	0.253584	0.105132	2.412%	49	3.858004	1.579703	2.442%
26	0.296982	0.123126	2.412%	50	4.190893	1.714208	2.445%

Fig. 6.3 Quadro comparativo do método de Elim de Gauss



METODO DE GAUSS JORDAN COM INVERSA							
ORD	ESCALAR	VETORIAL	TAXA	ORD	ESCALAR	VETORIAL	TAXA
03	0.000165	0.000142	1.162%	27	0.078481	0.037097	2.116%
04	0.000331	0.000236	1.403%	28	0.087684	0.041513	2.112%
05	0.000597	0.000374	1.516%	29	0.097522	0.046175	2.112%
06	0.000960	0.000558	1.720%	30	0.108011	0.051279	2.106%
07	0.001471	0.000779	1.888%	31	0.119263	0.056561	2.109%
08	0.002147	0.001123	1.912%	32	0.131239	0.062268	2.108%
09	0.002959	0.001496	1.978%	33	0.144005	0.068587	2.100%
10	0.004019	0.001989	2.021%	34	0.157424	0.074891	2.102%
11	0.005237	0.002528	2.072%	35	0.171733	0.081844	2.098%
12	0.006728	0.003208	2.097%	36	0.186871	0.088850	2.103%
13	0.008516	0.003991	2.134%	37	0.202787	0.096611	2.099%
14	0.010564	0.004919	2.148%	38	0.219688	0.104635	2.100%
15	0.012974	0.006016	2.157%	39	0.237491	0.113238	2.097%
16	0.015759	0.007299	2.159%	40	0.256413	0.122117	2.100%
17	0.018903	0.008783	2.152%	41	0.276020	0.131512	2.099%
18	0.022422	0.010440	2.148%	42	0.296419	0.141433	2.096%
19	0.026455	0.012280	2.154%	43	0.318187	0.151764	2.097%
20	0.030999	0.014415	2.150%	44	0.341892	0.162490	2.104%
21	0.036027	0.016804	2.144%	45	0.364491	0.173941	2.095%
22	0.041788	0.019545	2.138%	46	0.389242	0.185701	2.096%
23	0.047924	0.022514	2.129%	47	0.415160	0.198004	2.097%
24	0.054703	0.025746	2.125%	48	0.442450	0.210972	2.097%
25	0.061974	0.029240	2.119%	49	0.470475	0.224548	2.095%
26	0.069896	0.033005	2.118%	50	0.500419	0.238337	2.100%

Fig. 6.4 Quadro comparativo do método de Gauss-Jordan



No teste das matrizes esparsas, foi adotado matrizes do tipo banda, com a característica de serem diagonalmente dominante para garantir a convergência. Exigiu-se uma exatidão de oito algarismos significativos corretos e um limite de iteração de 50 iterações como critérios de parada. Foram testados sistemas de ordem N=500, 1000, 1500, 2000, 2500 e 3000. Os resultados em termos de tempo de processamento foram tabelados nas figuras 6.5 e 6.6.

Métodos Iterativos - tempo médio de processamento (segundos)			
ORDEM DA MATRIZ	JACOBI sem otimização	JACOBI com vetorização	TAXA DE REDUÇÃO
500	23,877	3.870	6,170 %
1000	122,515	39.766	3,081 %
1500	314,064	124.296	2,527 %
2000	565,676	228,451	2,476 %
2500	890,790	365.889	2,435 %
3000	1295,718	533,942	2,427 %

Fig.6.5 - Quadro comparativo do método de Gauss Jacobi

Métodos Iterativos - tempo médio de processamento (segundos)			
ORDEM DA MATRIZ	SEIDEL sem otimização	SEIDEL com vetorização	TAXA DE REDUÇÃO
500	3,728	0.591	6,308 %
1000	21,583	7.083	3,047 %
1500	67,664	26.985	2,507 %
2000	132,796	54.814	2,426 %
2500	888,863	368.842	2,410 %
3000	1292,752	536,173	2,411 %

Fig. 6.6 - Quadro comparativo do método de Gauss Seidel

Os programas foram executados três vezes cada um no modo escalar e no modo vetorial, para tomada do tempo de execução médio, uma vez que a quantidade de programas rodando ao mesmo tempo no Convex, pode alterar ou mascarar o resultado real. O tempo foi tomado através de uma função da VECLIB que calcula o tempo de uso da CPU em segundos.

No primeiro momento, não foi feita uma análise das dependências com o objetivo de aumentar o grau de vetorização dos loops. Apenas foi utilizada a vetorização automática dos algoritmos básicos que já haviam sido testados em outros sistemas computacionais. Tentou-se, apenas, monitorar a otimização para se verificar que rotinas ou que loops foram efetivamente vetorizados, produzindo ganho de tempo de execução. A listagem da vetorização encontra-se no anexo.

Com os contatos com os pesquisadores da Engenharia Mecânica; com a verificação no manual Convex Training Advance Optimization e com os testes feitos, verificou-se que para uma boa utilização do sistema Convex é necessário um grande conhecimento da máquina, da linguagem fortran e das rotinas otimizadas da Veclib, pois dependendo de como se desenvolve o programa, esse poderá gastar até 20 vezes mais do que o tempo de um programa otimizado.

Verificou-se que o teste para sistemas com matrizes esparsas, onde a ordem máxima era 3000 elementos, o programa ocupou 106 Megabytes de memória e sua execução não pode ser feita no modo interativo, tendo que ser submetido as filas de processamento. Este programa acabou sobrecarregando o sistema, comprometendo o tempo de resposta do sistema.

Esta máquina tem um grande potencial de utilização para computações de larga-escala, especialmente se houver mais de uma CPU, como é o caso da USP, em São Paulo (C220 - 2 CPUs).

Como sugestão a novos trabalhos, pretende-se rodar os testes desenvolvidos no Convex C210 (vetorizados) no Convex C220 (com paralelização) e fazer a análise dos resultados.

Outros testes que se pretende desenvolver é o desenvolvimento de uma biblioteca vetorial para a Aritmética Intervalar com vistas a suportar versões intervalares do método de resolução de equações algébricas de Newton-Raphson, as quais tem sido testadas e implementadas em PASCAL-SC em micros de 16 bits.



A N E X O S

ANEXO 1: Execução do teste dos métodos compactos

U F S C - Florianópolis - 1991

Teste do tempo de processamento Escalar

Solucao de sistemas de equacoes - matrizes densas

ORDEM	HILBERT	BANACHIEVIZC	CHOLESKI
100	0.016141	0.613744	0.628237
200	0.061752	4.749834	4.872020
300	0.138275	15.856930	16.263510
400	0.243598	37.396183	38.354591
500	0.379068	72.793785	74.686371
600	0.545560	125.508575	128.812469
700	0.744814	198.992752	204.321243
800	0.972044	296.713287	304.693970
900	1.234237	422.171112	433.651184
1000	1.534230	581.132385	597.391052

tempo total de processamento = 3565.480

ORDEM	HILBERT	BANACHIEVIZC	CHOLESKI
100	0.015952	0.616840	0.629677
200	0.061430	4.767834	4.896128
300	0.138255	15.903665	16.323185
400	0.244879	37.471027	38.431637
500	0.380670	72.961143	75.498619
600	0.584124	127.978531	130.103455
700	0.754377	201.006165	205.122742
800	0.976855	298.522308	314.014862
900	1.344438	436.936005	452.380951
1000	1.670615	604.272400	625.266785

tempo total de processamento = 3669.287



Teste do tempo de processamento Vetorial

Solucao de sistemas de equacoes - matrizes densas

ORDEM	HILBERT	BANACHIEVIZC	CHOLESKI
100	0.003322	0.110462	0.109678
200	0.012143	0.716773	0.713765
300	0.026820	2.229119	2.223425
400	0.047190	5.054604	5.048315
500	0.073023	9.610176	9.599292
600	0.104622	16.304766	16.296192
700	0.144644	25.544910	25.554331
800	0.186315	37.765900	37.783127
900	0.239728	53.368237	53.420242
1000	0.294461	72.825127	72.894081

tempo total de processamento = 448.3121

ORDEM	HILBERT	BANACHIEVIZC	CHOLESKI
100	0.003323	0.111843	0.110384
200	0.012449	0.734441	0.726085
300	0.027053	2.279458	2.273228
400	0.051398	5.186587	5.186444
500	0.077129	9.934032	9.964168
600	0.111195	16.859535	16.910810
700	0.149801	26.561981	26.680984
800	0.204950	39.360710	39.536098
900	0.273450	56.047131	56.098900
1000	0.326926	74.397568	74.057831

tempo total de processamento = 464.2681

ORDEM	HILBERT	BANACHIEVIZC	CHOLESKI
100	0.003333	0.110488	0.109483
200	0.012174	0.723176	0.715033
300	0.027395	2.230424	2.232015
400	0.047191	5.094457	5.071763
500	0.072905	9.652547	9.640153
600	0.105063	16.430748	16.415802
700	0.146250	26.127661	26.723244
800	0.215057	39.196899	39.048637
900	0.253507	55.038219	54.701027
1000	0.294000	73.496033	74.175972

tempo total de processamento = 458.1206



ANEXO 2: Execução do teste dos métodos Diretos

U F S C - Florianopolis - 1991

Teste do tempo de processamento Escalar

Solucao de sistemas de equacoes - matrizes densas

ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVICZ	CHOLESKY
3	0.000091	0.000122	0.000165	0.000080	0.000070
4	0.000062	0.000225	0.000331	0.000138	0.000124
5	0.000086	0.000515	0.000597	0.000217	0.000204
6	0.000113	0.000935	0.000960	0.000325	0.000310
7	0.000143	0.001668	0.001471	0.000478	0.000451
8	0.000177	0.002774	0.002147	0.000644	0.000626
9	0.000214	0.004395	0.002959	0.000857	0.000864
10	0.000257	0.006605	0.004019	0.001120	0.001103
11	0.000301	0.009567	0.005237	0.001437	0.001416
12	0.000350	0.013437	0.006728	0.001781	0.001770
13	0.000402	0.018467	0.008516	0.002210	0.002195
14	0.000458	0.024848	0.010564	0.002726	0.002688
15	0.000519	0.032711	0.012974	0.003268	0.003277
16	0.000587	0.042231	0.015759	0.003895	0.003872
17	0.000648	0.053820	0.018903	0.004599	0.004589
18	0.000719	0.067615	0.022422	0.005411	0.005387
19	0.000793	0.083943	0.026455	0.006284	0.006268
20	0.000872	0.103052	0.030999	0.007270	0.007210
21	0.000951	0.125383	0.036027	0.008345	0.008263
22	0.001036	0.151137	0.041788	0.009492	0.009403
23	0.001124	0.181134	0.047924	0.010751	0.010650
24	0.001217	0.214926	0.054703	0.012147	0.012013
25	0.001312	0.253584	0.061974	0.013636	0.013472
26	0.001411	0.296982	0.069896	0.015255	0.015047
27	0.001514	0.345529	0.078481	0.017008	0.016760
28	0.001620	0.400492	0.087684	0.018861	0.018606
29	0.001730	0.487676	0.097522	0.020890	0.020582
30	0.001844	0.528531	0.108011	0.023000	0.022786
31	0.001964	0.603081	0.119263	0.025298	0.024989
32	0.002106	0.685301	0.131239	0.027737	0.027499
33	0.002215	0.776107	0.144005	0.030320	0.030143
34	0.002342	0.875187	0.157424	0.033100	0.032991
35	0.002467	1.032079	0.171733	0.035997	0.035958
36	0.002625	1.104445	0.186871	0.039085	0.039165
37	0.002740	1.233961	0.202787	0.042314	0.042619
38	0.002883	1.374623	0.219688	0.045756	0.046089
39	0.003049	1.528219	0.237491	0.049363	0.049846
40	0.003187	1.692204	0.256413	0.053183	0.054158
41	0.003341	1.872751	0.276020	0.057167	0.057855
42	0.003511	2.063411	0.296419	0.061388	0.062139
43	0.003672	2.270007	0.318187	0.065900	0.066661
44	0.003822	2.494243	0.341892	0.070482	0.071502
45	0.004004	2.732690	0.364491	0.075155	0.076278
46	0.004162	2.985720	0.389242	0.080117	0.081392
47	0.004336	3.259621	0.415160	0.085575	0.086769
48	0.004516	3.547190	0.442450	0.090800	0.092417
49	0.004707	3.858004	0.470475	0.096707	0.098162
50	0.004919	4.190893	0.500419	0.102810	0.104380

tempo total de processamento = 52.97307



ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVIZC	CHOLESKI
3	0.000092	0.000124	0.000165	0.000081	0.000070
4	0.000066	0.000226	0.000331	0.000137	0.000124
5	0.000086	0.000482	0.000591	0.000217	0.000203
6	0.000113	0.000935	0.000961	0.000325	0.000323
7	0.000143	0.001668	0.001471	0.000467	0.000452
8	0.000177	0.002775	0.002164	0.000649	0.000627
9	0.000216	0.004375	0.002973	0.000859	0.000845
10	0.000257	0.006600	0.004001	0.001115	0.001104
11	0.000301	0.009565	0.005256	0.001426	0.001411
12	0.000349	0.013451	0.006706	0.001775	0.001788
13	0.000401	0.018462	0.008466	0.002218	0.002199
14	0.000457	0.024863	0.010564	0.002706	0.002684
15	0.000520	0.032774	0.012994	0.003294	0.003249
16	0.000582	0.042254	0.015766	0.003893	0.003895
17	0.000648	0.053770	0.018905	0.004640	0.004580
18	0.000719	0.067608	0.022464	0.005391	0.005404
19	0.000793	0.083864	0.026500	0.006313	0.006249
20	0.000872	0.103012	0.030997	0.007268	0.007217
21	0.000951	0.125372	0.036038	0.008322	0.008255
22	0.001036	0.151152	0.041720	0.009494	0.009406
23	0.001124	0.181181	0.047963	0.010755	0.010655
24	0.001217	0.214970	0.054838	0.012139	0.012010
25	0.001324	0.253619	0.062018	0.013622	0.013481
26	0.001411	0.297048	0.070024	0.015241	0.015067
27	0.001514	0.345629	0.078539	0.016982	0.016764
28	0.001619	0.400704	0.087729	0.018874	0.018595
29	0.001743	0.461263	0.097547	0.020867	0.020683
30	0.001863	0.528609	0.108062	0.023003	0.022729
31	0.001970	0.603406	0.119273	0.025413	0.025009
32	0.002082	0.685673	0.131371	0.027755	0.027473
33	0.002206	0.776256	0.144025	0.030325	0.030215
34	0.002354	0.875594	0.157534	0.033144	0.032961
35	0.002466	0.984190	0.171814	0.036052	0.035967
36	0.002613	1.103901	0.187000	0.039132	0.039179
37	0.002739	1.233673	0.202854	0.042313	0.042560
38	0.002892	1.374776	0.219795	0.045761	0.046079
39	0.003028	1.527763	0.237507	0.049388	0.049862
40	0.003188	1.692938	0.256284	0.053184	0.053748
41	0.003333	1.871692	0.275919	0.057184	0.057854
42	0.003500	2.064076	0.296396	0.061380	0.062178
43	0.003671	2.271082	0.318071	0.065703	0.066695
44	0.003846	2.493879	0.340697	0.070307	0.071496
45	0.003991	2.732204	0.364339	0.075135	0.076246
46	0.004152	2.986188	0.389295	0.080065	0.081417
47	0.004336	3.257284	0.414991	0.085312	0.086766
48	0.004527	3.546644	0.442058	0.090893	0.092338
49	0.004708	3.857587	0.470229	0.096572	0.098073
50	0.004896	4.185188	0.499415	0.102659	0.104058

tempo total de processamento = 52.88781



ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVIZC	CHOLESKI
3	0.000090	0.000121	0.000164	0.000080	0.000070
4	0.000063	0.000225	0.000330	0.000138	0.000124
5	0.000086	0.000482	0.000590	0.000217	0.000204
6	0.000112	0.000935	0.000959	0.000325	0.000310
7	0.000144	0.001668	0.001470	0.000484	0.000454
8	0.000177	0.002774	0.002128	0.000640	0.000626
9	0.000216	0.004414	0.002963	0.000859	0.000846
10	0.000258	0.006605	0.003984	0.001121	0.001104
11	0.000302	0.009568	0.005218	0.001421	0.001451
12	0.000352	0.013438	0.006707	0.001783	0.001770
13	0.000403	0.018467	0.008466	0.002204	0.002195
14	0.000459	0.024826	0.010561	0.002735	0.002690
15	0.000521	0.032697	0.012973	0.003270	0.003240
16	0.000584	0.042250	0.015700	0.003940	0.003883
17	0.000651	0.053803	0.018913	0.004605	0.004600
18	0.000720	0.067586	0.022418	0.005399	0.005356
19	0.000795	0.083999	0.026444	0.006257	0.006231
20	0.000874	0.102994	0.031022	0.007256	0.007170
21	0.000954	0.125325	0.036045	0.008343	0.008261
22	0.001045	0.151169	0.041703	0.009478	0.009397
23	0.001133	0.180731	0.047993	0.010722	0.010605
24	0.001218	0.214677	0.054771	0.012106	0.011981
25	0.001314	0.253265	0.062019	0.013607	0.013424
26	0.001413	0.296799	0.069928	0.015208	0.015017
27	0.001515	0.345673	0.078638	0.016963	0.016695
28	0.001655	0.400297	0.087782	0.018845	0.018555
29	0.001733	0.461156	0.097577	0.020830	0.020527
30	0.001846	0.528696	0.108093	0.022968	0.022741
31	0.001972	0.603535	0.119330	0.025327	0.024998
32	0.002084	0.685830	0.131300	0.027730	0.027529
33	0.002229	0.776558	0.144055	0.030335	0.030108
34	0.002348	0.875670	0.157553	0.033097	0.032960
35	0.002476	0.984391	0.171800	0.036039	0.036037
36	0.002612	1.103479	0.186897	0.039098	0.039145
37	0.002740	1.233683	0.203166	0.042450	0.042540
38	0.002898	1.374801	0.219626	0.045866	0.046059
39	0.003042	1.527840	0.237595	0.049398	0.049788
40	0.003191	1.692985	0.256236	0.053211	0.053785
41	0.003344	1.889616	0.275864	0.057188	0.057861
42	0.003513	2.064245	0.296491	0.061465	0.062147
43	0.003650	2.271280	0.318277	0.065788	0.066707
44	0.003824	2.519880	0.340825	0.070381	0.071414
45	0.003989	2.758455	0.364622	0.075125	0.076284
46	0.004152	3.000517	0.389269	0.080209	0.081503
47	0.004336	3.263186	0.415211	0.085364	0.086759
48	0.004528	3.546703	0.442196	0.090859	0.092386
49	0.004709	3.867560	0.470218	0.096612	0.098111
50	0.004883	4.185253	0.499575	0.102634	0.104150

tempo total de processamento = 52.98877



U F S C - Florianopolis - 1991  
**Teste do tempo de processamento Vetorial**  
 Solucao de sistemas de equacoes - matrizes densas

ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVIZC	CHOLESKI
3	0.000099	0.000092	0.000142	0.000099	0.000071
4	0.000034	0.000134	0.000236	0.000125	0.000117
5	0.000039	0.000255	0.000374	0.000191	0.000179
6	0.000043	0.000463	0.000558	0.000279	0.000264
7	0.000048	0.000782	0.000799	0.000386	0.000366
8	0.000054	0.001272	0.001123	0.000520	0.000496
9	0.000059	0.001968	0.001496	0.000681	0.000653
10	0.000065	0.002912	0.001989	0.000870	0.000837
11	0.000072	0.004160	0.002528	0.001101	0.001049
12	0.000078	0.005817	0.003208	0.001344	0.001299
13	0.000086	0.007912	0.003991	0.001636	0.001597
14	0.000091	0.010560	0.004919	0.001966	0.001933
15	0.000107	0.013777	0.006016	0.002338	0.002273
16	0.000107	0.017795	0.007299	0.002752	0.002703
17	0.000136	0.022537	0.008783	0.003217	0.003158
18	0.000138	0.028319	0.010440	0.003723	0.003647
19	0.000151	0.034977	0.012280	0.004289	0.004187
20	0.000159	0.042985	0.014415	0.004905	0.004783
21	0.000169	0.052109	0.016804	0.005577	0.005455
22	0.000177	0.062775	0.019545	0.006296	0.006156
23	0.000191	0.074900	0.022514	0.007077	0.006941
24	0.000199	0.088975	0.025746	0.007910	0.007768
25	0.000213	0.104569	0.029240	0.008823	0.008687
26	0.000221	0.122477	0.033005	0.009805	0.009644
27	0.000233	0.142396	0.037097	0.010859	0.010678
28	0.000243	0.164837	0.041513	0.011970	0.011777
29	0.000256	0.189564	0.046175	0.013164	0.012962
30	0.000267	0.217465	0.051279	0.014424	0.014221
31	0.000281	0.248129	0.056561	0.015780	0.015558
32	0.000292	0.282129	0.062268	0.017222	0.016964
33	0.000335	0.319462	0.068587	0.018708	0.018473
34	0.000346	0.360594	0.074891	0.020327	0.020066
35	0.000367	0.404492	0.081844	0.022079	0.021774
36	0.000387	0.453510	0.088850	0.023836	0.023528
37	0.000399	0.506204	0.096611	0.025644	0.025380
38	0.000418	0.564730	0.104635	0.027645	0.027321
39	0.000433	0.627211	0.113238	0.029748	0.029369
40	0.000444	0.694652	0.122117	0.031881	0.031519
41	0.000478	0.767796	0.131512	0.034156	0.033786
42	0.000477	0.847289	0.141433	0.036579	0.036197
43	0.000509	0.931697	0.151764	0.039003	0.038658
44	0.000522	1.023007	0.162490	0.041609	0.041185
45	0.000538	1.120506	0.173941	0.044311	0.043896
46	0.000552	1.225008	0.185701	0.047135	0.046696
47	0.000578	1.335807	0.198004	0.050067	0.049648
48	0.000604	1.455612	0.210972	0.053209	0.052689
49	0.000667	1.581352	0.224548	0.056315	0.055860
50	0.000671	1.716445	0.238337	0.059630	0.059145

tempo de total processamento = 22.6460



ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVIZC	CHOLESKI
3	0.000098	0.000095	0.000168	0.000078	0.000072
4	0.000038	0.000141	0.000237	0.000126	0.000116
5	0.000039	0.000263	0.000382	0.000190	0.000177
6	0.000043	0.000480	0.000572	0.000277	0.000261
7	0.000049	0.000810	0.000820	0.000384	0.000378
8	0.000054	0.001307	0.001142	0.000517	0.000492
9	0.000059	0.001994	0.001536	0.000680	0.000663
10	0.000065	0.002957	0.002018	0.000870	0.000833
11	0.000072	0.004242	0.002600	0.001092	0.001050
12	0.000080	0.005901	0.003272	0.001344	0.001309
13	0.000086	0.007978	0.004128	0.001640	0.001585
14	0.000092	0.010689	0.005027	0.001969	0.001907
15	0.000113	0.013950	0.006123	0.002340	0.002273
16	0.000108	0.018014	0.007405	0.002757	0.002676
17	0.000132	0.022843	0.008880	0.003215	0.003135
18	0.000138	0.028651	0.010526	0.003738	0.003633
19	0.000149	0.035457	0.012374	0.004309	0.004186
20	0.000160	0.043426	0.014420	0.004925	0.004787
21	0.000169	0.052593	0.016810	0.005573	0.005444
22	0.000176	0.063326	0.019558	0.006298	0.006157
23	0.000191	0.075451	0.022534	0.007077	0.006954
24	0.000201	0.089504	0.025789	0.007929	0.007788
25	0.000221	0.105132	0.029312	0.008845	0.008681
26	0.000220	0.123126	0.033068	0.009811	0.009662
27	0.000241	0.143166	0.037207	0.010857	0.010675
28	0.000244	0.165755	0.041580	0.012003	0.011785
29	0.000257	0.190650	0.046360	0.013186	0.012972
30	0.000267	0.218518	0.051430	0.014473	0.014237
31	0.000282	0.249083	0.056855	0.015799	0.015567
32	0.000291	0.283006	0.062557	0.017216	0.016983
33	0.000343	0.320114	0.068757	0.018733	0.018505
34	0.000354	0.361032	0.075224	0.020335	0.020074
35	0.000369	0.405189	0.082064	0.022048	0.021735
36	0.000380	0.454343	0.089274	0.023824	0.023485
37	0.000398	0.507167	0.096907	0.025685	0.025371
38	0.000418	0.564849	0.105029	0.027641	0.027313
39	0.000432	0.626942	0.113448	0.029747	0.029371
40	0.000446	0.695089	0.122350	0.031874	0.031532
41	0.000475	0.768176	0.131781	0.034159	0.033799
42	0.000478	0.847309	0.141507	0.036571	0.036150
43	0.000503	0.931347	0.151774	0.039007	0.038627
44	0.000522	1.022780	0.162425	0.041661	0.041182
45	0.000539	1.120046	0.173872	0.044318	0.043884
46	0.000563	1.224499	0.185546	0.047129	0.046716
47	0.000588	1.335215	0.197857	0.050065	0.049676
48	0.000603	1.454015	0.210704	0.053166	0.052659
49	0.000657	1.579703	0.224230	0.056335	0.055854
50	0.000681	1.714208	0.238488	0.059635	0.059117

tempo total de processamento = 22.65905



ORDEM	HILBERT	MEG-PS	GAUSS-JORDAN	BANACHIEVIZC	CHOLESKI
3	0.000102	0.000096	0.000167	0.000078	0.000070
4	0.000034	0.000141	0.000236	0.000126	0.000116
5	0.000039	0.000263	0.000381	0.000189	0.000177
6	0.000043	0.000495	0.000573	0.000278	0.000261
7	0.000049	0.000810	0.000819	0.000385	0.000364
8	0.000054	0.001307	0.001141	0.000517	0.000491
9	0.000059	0.001994	0.001535	0.000681	0.000651
10	0.000066	0.002955	0.002051	0.000871	0.000834
11	0.000073	0.004210	0.002611	0.001091	0.001050
12	0.000078	0.005898	0.003274	0.001345	0.001296
13	0.000086	0.007990	0.004099	0.001637	0.001583
14	0.000092	0.010691	0.005049	0.001967	0.001907
15	0.000104	0.013980	0.006114	0.002338	0.002285
16	0.000108	0.018012	0.007365	0.002766	0.002677
17	0.000130	0.022830	0.008895	0.003238	0.003142
18	0.000139	0.028651	0.010520	0.003726	0.003646
19	0.000149	0.035388	0.012377	0.004284	0.004183
20	0.000166	0.043421	0.014452	0.004895	0.004794
21	0.000168	0.052611	0.016843	0.005560	0.005456
22	0.000178	0.063375	0.019586	0.006282	0.006171
23	0.000191	0.075450	0.022536	0.007079	0.006927
24	0.000211	0.089476	0.025779	0.007926	0.007777
25	0.000213	0.105164	0.029296	0.008838	0.008700
26	0.000228	0.123109	0.033109	0.009811	0.009640
27	0.000233	0.143137	0.037234	0.010885	0.010684
28	0.000253	0.165891	0.041570	0.011972	0.011775
29	0.000269	0.190681	0.046307	0.013185	0.012966
30	0.000267	0.218548	0.051416	0.014447	0.014218
31	0.000282	0.249130	0.056853	0.015809	0.015555
32	0.000292	0.283012	0.062591	0.017219	0.016963
33	0.000345	0.320080	0.068767	0.018745	0.018491
34	0.000345	0.361120	0.075204	0.020345	0.020054
35	0.000367	0.405222	0.082001	0.022042	0.021740
36	0.000379	0.454378	0.089288	0.023802	0.023487
37	0.000398	0.507064	0.096937	0.025674	0.025385
38	0.000418	0.564924	0.104980	0.027646	0.027370
39	0.000442	0.626939	0.113414	0.029719	0.029387
40	0.000454	0.694868	0.122311	0.031924	0.031527
41	0.000479	0.767918	0.131735	0.034171	0.033806
42	0.000478	0.847268	0.141518	0.036542	0.036149
43	0.000502	0.931402	0.151768	0.039027	0.038617
44	0.000521	1.022769	0.162460	0.041626	0.041171
45	0.000539	1.119912	0.173813	0.044303	0.043897
46	0.000563	1.224418	0.185519	0.047119	0.046709
47	0.000589	1.335201	0.197880	0.050106	0.049614
48	0.000594	1.454193	0.210736	0.053126	0.052646
49	0.000666	1.579744	0.224291	0.056332	0.055828
50	0.000671	1.714251	0.238471	0.059639	0.059118

tempo total de processamento = 22.65860



ANEXO 3: Execução do teste dos métodos Iterativos

U F S C - Florianopolis - 1991  
Teste do tempo de processamento Escalar  
Solucao de sistemas de equacoes -Metodos Iterativos

ORDEM	ESPARSA	SEIDEL	JACOBI
500	0.016855	3.735023	23.951582
1000	0.027478	21.661345	122.957573
1500	0.036432	67.743668	314.190704
2000	0.044714	132.836227	565.897705
2500	0.049837	889.039917	890.781921
3000	0.050556	1292.750122	1295.722290

tempo total de processamento 5621.498

ORDEM	ESPARSA	SEIDEL	JACOBI
500	0.016804	3.720770	23.803711
1000	0.027507	21.505850	122.072853
1500	0.036372	67.584862	313.938812
2000	0.044537	132.756058	565.454834
2500	0.049786	888.687195	890.799805
3000	0.050507	1292.753296	1295.714722

tempo total de processamento 5619.022



U F S C - Florianopolis - 1991  
**Teste do tempo de processamento Vetorial**  
 Solucao de sistemas de equacoes -Metodos Iterativos

ORDEM	ESPARSA	SEIDEL	JACOBI
500	0.013413	0.587391	3.779836
1000	0.020477	6.822572	38.731705
1500	0.027364	26.473307	122.817886
2000	0.033409	53.022766	225.493118
2500	0.036470	359.368286	359.519501
3000	0.035454	531.409607	531.631165

tempo total de processamento 2259.828

ORDEM	ESPARSA	SEIDEL	JACOBI
500	0.013428	0.595892	3.961614
1000	0.025246	7.343166	40.800880
1500	0.031690	27.497118	125.775002
2000	0.034051	55.814640	231.409622
2500	0.045041	378.316437	372.258026
3000	0.039978	540.936523	536.252686

tempo total de processamento 2321.158

ANEXO 4: Programa MDEN50.F (FORTRAN 77 - CONVEX)

Teste de matrizes densas de ordem 3 a 50: Mden50.f

```

Program main
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
real*4 cputime,t0,tf0,t1,tf1,t2,tf2,t3,tf3,t4,tf4,t5,tf5
common /data/ a,b,x,y,ai,n
t5=cputime(0.0)
write(*,*) "      U F S C      - Florianopolis - 1991"
write(*,*)
write(*,*)
write(*,*) "Teste do tempo de processamento"
write(*,*)
write(*,*) " Solucao de sistemas de equacoes - matrizes densas"
write(*,*)
write(*,*)"ORDEM  HILBERT  MEG-PS  GAUSS-JORDAN  BANACHIEVICZ
*  CHOLESKI"
write(*,*)
do n=3,50,1
  t0=cputime(0.0)
  call hilbert
  tf0=cputime(t0)
  t1=cputime(0.0)
  call megps
  tf1=cputime(t1)
  t2=cputime(0.0)
  call jordan
  tf2=cputime(t2)
  t3=cputime(0.0)
  call banac
  tf3=cputime(t3)
  t4=cputime(0.0)
  call choleski
  tf4=cputime(t4)
  write(*,10) n,tf0,tf1,tf2,tf3,tf4
10  format(i5,3x,5(f12.6,3x))
enddo
t5=cputime(t5)
write(*,*) "tempo total de processamento =",t5
end

Subroutine Hilbert
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
common /data/a,b,x,y,ai,n
do i=1,n
  do j=1,n
    a(i,j)=1.0/(i+j-1.0)
  enddo
  b(i)=i**3-i**2
enddo
end

```



```

Subroutine jordan
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
common /data/ a,b,x,y,ai,n
do i=1,n
  do j=1,n
    ai(i,j)=a(i,j)
    ai(i,n+j)=0.0
  enddo
  ai(i,n+i)=1.0
enddo
do i=1,n
  do j=1+i,n
    pivo=-ai(j,i)/ai(i,i)
    do k=1,n+n
      ai(j,k)=pivo*ai(i,k)+ai(j,k)
    enddo
  enddo
enddo
do i=n,1,-1
  do k=i-1,1,-1
    pivo=-ai(k,i)/ai(i,i)
    do j=1,n+n
      ai(k,j)=ai(i,j)*pivo+ai(k,j)
    enddo
  enddo
enddo
do i=1,n
  e1=ai(i,i)
  do j=1,n+n
    ai(i,j)=ai(i,j)/e1
  enddo
enddo
do i=1,n
  x(i)=0.0
  do j=1,n
    x(i)=ai(i,n+j)*b(j)+x(i)
  enddo
enddo
end

```

```

Subroutine megps
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
common /data/a,b,x,y,ai,n

do i=1,n
  ai(i,n+1)=b(i)
  do j=1,n
    ai(i,j)=a(i,j)
  enddo
enddo

```

```

do i=1,n
  do j=i+1,n
    call piv(j)
    pivo=-ai(j,i)/ai(i,i)
    do k=1,n+1
      ai(j,k)=pivo*ai(i,k)+ai(j,k)
    enddo
  enddo
enddo
do i=n,1,-1
  x(i)=ai(i,n+1)
  do j=1+i,n
    x(i)=x(i)-ai(i,j)*x(j)
  enddo
  x(i)=x(i)/ai(i,i)
enddo
end

```

```

Subroutine Piv(k)
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n,lmaior
common /data/a,b,x,y,ai,n

```

```

lmaior=k
maior=abs(ai(k,k))
do m=k+1,n
  if (maior .le. abs(ai(m,k))) then
    maior=abs(ai(m,k))
    lmaior=m
  endif
  if (lmaior .ne. k) then
    do u=1,n+1
      aux=ai(k,u)
      ai(k,u)=ai(lmaior,u)
      ai(lmaior,u)=aux
    enddo
  endif
enddo
end

```

```

Subroutine Banac
real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
common /data/ a,b,x,y,ai,n

```

```

do i=1,n
  do j=1,n
    ai(i,j)=a(i,j)
  enddo
enddo

```



```

do i=1,n
  do j=1,i
    sum=0.0
    do k=1,j-1
      sum=sum+ai(i,k)*ai(k,j)
    enddo
    ai(i,j)=ai(i,j)-sum
  enddo
  do j=i+1,n
    sum=0.0
    do k=1,i-1
      sum=sum+ai(i,k)*ai(k,j)
    enddo
    ai(i,j)=(ai(i,j)-sum)/ai(i,i)
  enddo
enddo
do i=1,n
  sum=0.0
  do j=1,i-1
    sum=sum+ai(i,j)*y(j)
  enddo
  y(i)=(b(i)-sum)/ai(i,i)
enddo
do i=n,1,-1
  sum=0.0
  do j=i+1,n
    sum=sum+ai(i,j)*x(j)
  enddo
  x(i)=y(i)-sum
enddo
end

```

Subroutine Choleski

```

real*8 a(100,100),b(100),x(100),y(100),ai(100,200)
integer n
common /data/a,b,x,y,ai,n

do i=1,n
  do j=1,n
    ai(i,j)=a(i,j)
  enddo
  ai(i,n+1)=b(i)
enddo
do j=2,n+1
  ai(1,j)=ai(1,j)/ai(1,1)
enddo
do m=2,n
  do i=m,n
    sum1=0.0
    do k=1,m-1
      sum1=sum1+ai(i,k)*ai(k,m)
    enddo
    ai(i,m)=ai(i,m)-sum1
  enddo
enddo

```

```

do j=m+1,n+1
  sum2=0.0
  do k=1,m-1
    sum2=sum2+ai(m,k)*ai(k,j)
  enddo
  ai(m,j)=(ai(m,j)-sum2)/ai(m,m)
enddo
enddo
x(n)=ai(n,n+1)
do m=1,n-1
  i=n-m
  sum=0.0
  do j=i+1,n
    sum=sum+ai(i,j)*x(j)
  enddo
  x(i)=ai(i,n+1)-sum
enddo
end

```



ANEXO 5: Listagem da otimização e vetorização

Optimization by Loop for Routine MAIN

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
-----------	------------	---------------------------	-------------------------------------	------------

17	N	Scalar		
----	---	--------	--	--

Line Num.	Iter. Var.	Analysis		
-----------	------------	----------	--	--

17	N	No induction variable		
----	---	-----------------------	--	--

Optimization by Loop for Routine HILBERT

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
-----------	------------	---------------------------	-------------------------------------	------------

43	I	Scalar		
44	J	FULL VECTOR		

Line Num.	Iter. Var.	Analysis		
-----------	------------	----------	--	--

43	I	Unable to distribute		
----	---	----------------------	--	--

Optimization by Loop for Routine JORDAN

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
-----------	------------	---------------------------	-------------------------------------	------------

56	I	Dist		
56-1	I	FULL VECTOR	Inter	
56-2	I	FULL VECTOR		
57-1	J	Scalar		
63	I	Scalar		
64	J	Dist		
64-1	J	FULL VECTOR		
64-2	J	Inter		
66	K	Scalar		
71	I	Scalar		
72	K	Dist		
72-1	K	FULL VECTOR		
72-2	K	Inter		
74	J	Scalar		
79	I	Dist		
79-1	I	FULL VECTOR		
79-2	I	FULL VECTOR	Inter	
81-2	J	Scalar		
85	I	Dist		
85-1	I	FULL VECTOR		
85-2	I	FULL VECTOR	Inter	
87-2	J	Scalar		

Line Num.	Iter. Var.	Analysis
56-1	I	Interchanged to innermost
63	I	Inner loop has varying trip count
64-2	J	Insufficient vector code
71	I	Inner loop has varying trip count
72-2	K	Insufficient vector code
79-2	I	Interchanged to innermost
85-2	I	Interchanged to innermost

Optimization by Loop for Routine MEGPS

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
96	I	Dist		
96-1	I	FULL VECTOR		
96-2	I	FULL VECTOR Inter		
98-2	J	Scalar		
102	I	Scalar		
103	J	Scalar		
106	K	FULL VECTOR		
111	I	Scalar		
113	J	FULL VECTOR	Reduction	

Line Num.	Iter. Var.	Analysis
96-2	I	Interchanged to innermost
102	I	Inner loop has varying trip count
103	J	Inner loop has varying trip count
111	I	Inner loop has varying trip count

Optimization by Loop for Routine PIV

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
125	M	Scalar		
131	U	Scalar		

Line Num.	Iter. Var.	Analysis
125	M	Inner loop has varying trip count
131	U	Insufficient vector code



Optimization by Loop for Routine BANAC

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
143	I	FULL VECTOR Inter		
144	J	Scalar		
148	I	Scalar		
149	J	Scalar		
151	K	FULL VECTOR	Reduction	
156	J	Scalar		
158	K	FULL VECTOR	Reduction	
164	I	Scalar		
166	J	FULL VECTOR	Reduction	
171	I	Scalar		
173	J	FULL VECTOR	Reduction	

Line Num.	Iter. Var.	Analysis
143	I	Interchanged to innermost
148	I	Inner loop has varying trip count
149	J	Inner loop has varying trip count
156	J	Unable to distribute
164	I	Inner loop has varying trip count
171	I	Inner loop has varying trip count

Optimization by Loop for Routine CHOLESKI

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
183	I	Dist		
183-1	I	FULL VECTOR Inter		
183-2	I	FULL VECTOR		
184-1	J	Scalar		
189	J	FULL VECTOR		
192	M	Scalar		
193	I	Scalar		
195	K	FULL VECTOR	Reduction	
200	J	Scalar		
202	K	FULL VECTOR	Reduction	
209	M	Scalar		
212	J	FULL VECTOR	Reduction	

Line Num.	Iter. Var.	Analysis
183-1	I	Interchanged to innermost
192	M	Inner loop has varying trip count
193	I	Unable to distribute
200	J	Unable to distribute
209	M	Inner loop has varying trip count

ANEXO 6: Programa MESPARGA.F (FORTRAN 77 - CONVEX)

```

Program main
real a(3000,3000),b(3000),x(3000),asc(3000),xnovo(3000)
integer n,it,limit
real ascreq,xnew,mi
real*4 cputime,t0,tf0,t1,tf1,t2,tf2,t3,tf3
common /data/ a,b,x,n,mi,ascreq,limit

t3=cputime(0.0)
mi=0.0000000005
ascreq=8.0
data x/3000*1.0/
limit=50
write(*,*) "      U F S C   - Florianopolis - 1991"
write(*,*)
write(*,*)
write(*,*) "      Teste do tempo de processamento"
write(*,*)
write(*,*) "Solucao de sistemas de equacoes -Metodos Iterativos"
write(*,*)
write(*,*)"ORDEM          ESPARSA          SEIDEL          JACOBI"
write(*,*)

do n=500,3000,500
  t0=cputime(0.0)
  call esparsa
  tf0=cputime(t0)
  t1=cputime(0.0)
  call seidel
  tf1=cputime(t1)
  t2=cputime(0.0)
  do i=1,n
    x(i)=1.0
  enddo
  call jacobi
  tf2=cputime(t2)
  write(*,10) n,tf0,tf1,tf2
10  format(i5,3x,3(f12.6,3x))
enddo
tf3=cputime(t3)
write(*,*) "tempo total de processamento",tf3
end

.

Subroutine esparsa
real a(3000,3000),b(3000),x(3000)
integer n,limit
real mi,ascreq
common /data/ a,b,x,n,mi,ascreq,limit
data a/90000000*0.0/

a(1,1)=50
a(1,2)=-6
a(1,5)=7

```



```

a(1,9)=2
a(2,2)=70
a(2,3)=-1
a(2,4)=7
a(2,9)=2
a(n-1,n-1)=60
a(n-1,n)=3
a(n-1,104)=-4
a(n-1,356)=12
a(n,n)=100
a(n,123)=-5
a(n,456)=67
a(n,2)=1
a(n,n-1)=4
do i=3,n-2
  a(i,i-2)=1
  a(i,i-1)=1
  a(i,i)=25
  a(i,n-i)=-3
enddo
do i=1,n
  b(i)=(i+2)*35
enddo
end

```

Subroutine seidel

```

real a(3000,3000),b(3000),x(3000)
real asc(3000)
real xnew,ascmin,ascreq,mi
integer it,n,limit
common /data/a,b,x,n,mi,ascreq,limit

```

```

ascmin=0.0
it=0
do while ((ascmin .lt. ascreq).and.(it .le. limit))
  do i=1,n
    sum=0.0
    do j=1,n
      if (i .ne. j) sum=sum+a(i,j)*x(j)
    enddo
    xnew=(b(i)-sum)/a(i,i)
    e2=(xnew-x(i))/xnew
    e3=abs(e2)+mi
    asc(i)=-(.3+log10(e3))
    if (asc(i) .le. 0) asc(i)=0.0
    x(i)=xnew
    if (i .eq. 1) then
      ascmin=asc(i)
    else if (ascmin .gt. asc(i)) then
      ascmin=asc(i)
    endif
  enddo
  it=it+1
enddo
end

```

```

Subroutine jacobi
real a(3000,3000),b(3000),x(3000)
real asc(3000),xnovo(3000)
real ascmin,ascreq,mi,e2,e3
integer n,limit,it
common /data/ a,b,x,n,mi,ascreq,limit

ascmin=0.0
it=0.0
do while ((ascmin .lt. ascreq).and.(it.le.limit))
  do i=1,n
    sum=0.0
    do j=1,n
      if (i .ne. j) sum=sum+a(i,j)*x(j)
    enddo
    xnovo(i)=(b(i)-sum)/a(i,i)
    e2=(xnovo(i)-x(i))/xnovo(i)
    e3=abs(e2)+mi
    asc(i)=-(.3+log10(e3))
    if (asc(i) .le. 0) asc(i)=0.0
    if (i.eq.1) then
      ascmin=asc(i)
      else if (ascmin.ge.asc(i)) then
        ascmin=asc(i)
    endif
  enddo
  it=it+1
  do j=1,n
    x(j)=xnovo(j)
  enddo
enddo
end

```



ANEXO 7: Listagem de otimização e vetorização

Optimization by Loop for Routine MAIN

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
21	N	Scalar		
29	I	FULL VECTOR		

Line Num.	Iter. Var.	Analysis		
21	N	No induction variable		

Optimization by Loop for Routine ESPARSA

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
63	I	FULL VECTOR		
69	I	FULL VECTOR		

Optimization by Loop for Routine SEIDEL

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
81	*NONE*	Scalar		
82	I	Scalar		
84	J	FULL VECTOR	Reduction	

Line Num.	Iter. Var.	Analysis		
81	*NONE*	No induction variable		
82	I	Unable to distribute		

Optimization by Loop for Routine JACOBI

Line Num.	Iter. Var.	Reordering Transformation	Optimizing / Special Transformation	Exec. Mode
110	*NONE*	Scalar		
111	I	Scalar		
113	J	FULL VECTOR	Reduction	
128	J	FULL VECTOR		

Line Num.	Iter. Var.	Analysis		
110	*NONE*	No induction variable		
111	I	Unable to distribute		

ANEXO 8: Programa MDENSA.F (FORTRAN 77 - CONVEX)

Teste de matrizes densas: Mdenso.f

```

Program main
real a(1000,1000),b(1000),x(1000),y(1000),ai(1000,1001)
integer n
real*4 cputime,t0,tf0,t3,tf3,t4,tf4,t5,tf5
common /data/ a,b,x,y,ai,n

t5=cputime(0.0)
write(*,*) "          U F S C   -   Florianopolis   -   1991"
write(*,*)
write(*,*)
write(*,*) "Teste do tempo de processamento"
write(*,*)
write(*,*) " Solucao de sistemas de equacoes - matrizes densas"
write(*,*)
write(*,*)"ORDEM          HILBERT          BANACHIEVICZ          CHOLESKI"
write(*,*)

do n=100,1000,100
  t0=cputime(0.0)
  call hilbert
  tf0=cputime(t0)
  t3=cputime(0.0)
  call banac
  tf3=cputime(t3)
  t4=cputime(0.0)
  call choleski
  tf4=cputime(t4)
  write(*,10) n,tf0,tf1,tf2,tf3,tf4
10  format(i5,3x,5(f12.6,3x))
enddo
tf5=cputime(t5)
write(*,*) "tempo total de processamento =",tf5
end

Subroutine Hilbert
real a(1000,1000),b(1000),x(1000),y(1000),ai(1000,1001)
integer n
common /data/a,b,x,y,ai,n

do i=1,n
  do j=1,n
    a(i,j)=1.0/(i+j-1.0)
  enddo
  b(i)=i**3-i**2
enddo
end

```



```

Subroutine Banac
real a(1000,1000),b(1000),x(1000),y(1000),ai(1000,1001)
integer n
common /data/ a,b,x,y,ai,n

do i=1,n
  do j=1,n
    ai(i,j)=a(i,j)
  enddo
enddo
do i=1,n
  do j=1,i
    sum=0.0
    do k=1,j-1
      sum=sum+ai(i,k)*ai(k,j)
    enddo
    ai(i,j)=ai(i,j)-sum
  enddo
  do j=i+1,n
    sum=0.0
    do k=1,i-1
      sum=sum+ai(i,k)*ai(k,j)
    enddo
    ai(i,j)=(ai(i,j)-sum)/ai(i,i)
  enddo
enddo
do i=1,n
  sum=0.0
  do j=1,i-1
    sum=sum+ai(i,j)*y(j)
  enddo
  y(i)=(b(i)-sum)/ai(i,i)
enddo
do i=n,1,-1
  sum=0.0
  do j=i+1,n
    sum=sum+ai(i,j)*x(j)
  enddo
  x(i)=y(i)-sum
enddo
end

```

```

Subroutine Choleski
real a(1000,1000),b(1000),x(1000),y(1000),ai(1000,1001)
integer n
common /data/a,b,x,y,ai,n

do i=1,n
  do j=1,n
    ai(i,j)=a(i,j)
  enddo
  ai(i,n+1)=b(i)
enddo

```

```

do j=2,n+1
  ai(1,j)=ai(1,j)/ai(1,1)
enddo
do m=2,n
  do i=m,n
    sum1=0.0
    do k=1,m-1
      sum1=sum1+ai(i,k)*ai(k,m)
    enddo
    ai(i,m)=ai(i,m)-sum1
  enddo
  do j=m+1,n+1
    sum2=0.0
    do k=1,m-1
      sum2=sum2+ai(m,k)*ai(k,j)
    enddo
    ai(m,j)=(ai(m,j)-sum2)/ai(m,m)
  enddo
enddo
x(n)=ai(n,n+1)
do m=1,n-1
  i=n-m
  sum=0.0
  do j=i+1,n
    sum=sum+ai(i,j)*x(j)
  enddo
  x(i)=ai(i,n+1)-sum
enddo
end

```



## BIBLIOGRAFIA

### 1 Processamento vetorial e paralelo:

- ALLEN, J.R. Dependence analysis for subscripted variables and its applications to program transformations. Houston: Rice University, 1983.(Ph.D dissertation).
- BAER, J.L. **Computer Systems Architecture**. Maryland: Potomak, 1980.
- DEMINET, J. Experience with multiprocessor algorithms. **IEEE Transactions on Computer**. New York: C-31, N.4, p.278-288, Apr, 1982.
- DU BRULLE, A.A; SCARBOROUGH, R.G; KOLSRY, H.G. **How to write good vectorizable fortran**. Palo Alto, IBM Palo alto Scientific Center, 1985. (G320-3478).
- ENSLOW, P.H. **Multiprocessors and Parallel Processing**. New York: Wiley-Interscience, 1974.
- HAYES, J.P. **Computer architecture and organization**. New York: McGraw Hill, 1978.
- HELLER, D. A survey of parallel algorithms in numerical linear algebra. **SIAM Review**, V.20, p.740-777, 1978.
- HOCKNEY, R.W.; JESSHOPE, C.R. **Parallel computer: architecture, programming and algorithms**. Bristol: Adam Hilger, 1981.
- HWANG, K; BRIGGS, F.A. **Computer architecture and parallel processing**. New York: McGraw Hill, 1984.
- JAMIESON, L; GANNON, D; DOUGLASS, R. **The Characteristics of Parallel Algorithms**. Cambridge: Mit Press, 1985.
- KUCK, D.J. **The structure of computer and computations**. New York: Wiley, 1978. V.1.
- MORKAZEL, F.C.; PANETTA, J. Representação automática de programas sequenciais para processamento paralelo. In: SIMPOSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES E PROCESSAMENTO PARALELO, 3, 1990, nov.7-9, Rio de Janeiro, **Anais**. Rio de Janeiro: SBC/PUC-RJ, 1990. 340p. p.5.B.1.1-7.
- ORTEGA, J.M; VOIGT, R.G; Solution of partial differential equations on vector and parallel computers. **SIAM Review**, V.27, p.149-240, 1985.
- QUINN, J.M. **Designing efficient algorithms for parallel computers**. New York: McGraw Hill, 1988.

- RODRIQUE,G.;GIROUX,E.D.;PRATT,M. Perspective on large-scale scientific computations. Computer, New York: V.13, N.10, p.65-80, oct, 1980.
- RUMP,S.M Rigorous sensivity analysis for systems of linear and nonlinear equations. Mathematics of Computation, V.54, N.190, p.721-736, april,1990.
- SAMEH,A.H; Numerical Parallel algorithms -a survey. In: High Speed computer and algorithm organization. New York: Academic Press, 1977. p.207-228.
- SHEDLER, G.S. Parallel numerical methods for the solution of equations. Communications ACM, V.10, p.286-291, 1967.
- STONE,H.S.et al. Introduction to computer architecture. Chicago: Science Research Associations, 1980.
- SUGARMAN,R. Superpower computer. IEEE Spectrum, New York: V.17, N.4, p.28-34, Apr, 1980.
- WANG,H.H Introduction to vectorizing techniques on the IBM 3090 vector facility. Palo Alto, IBM Palo Alto Scientific Center, 1986. (G320-3489).



## 2 Convex, sistema operacional e linguagens

CONVEX UNIX PRIMER. 1989. (Doc Nro.710.000221-202).

CONVEX FORTRAN Language Reference Manual

CONVEX FORTRAN User's Guide

CONVEX FORTRAN Overview Course

CONVEX C User's Guide

CONVEX C Language Reference Manual

CONVEX Ada User's Guide

CONVEX Ada Quick Reference

CONVEX TRAINNING ADVANCE OPTIMIZATION

HEHL, Maximilian E Linguagem de programação estruturada  
**fortran77.** São Paulo: McGraw Hill, 1986.

JONES, Tom. Engineering Design of the Convex C2. Convex  
Corporation.

KERMIT guia do usuário. New York: Columbia University, 1984.  
(Tradução da TELEBRAS, Brasília, 1986).

MASSON, B An introduction to parallel processing on the Convex  
Supercomputer. In: Convex computer technology review. Jul,  
1990.

MERCER, Randall The Convex Fortran 5.0 Compiler. Convex  
Corporation.

OLIVEIRA, V.D; SCHWEITZER, A. Curso de UNIX. Florianopolis, NPD  
UFSC, 1990. (Apostila)