31830-6

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL UFRGS

INSTITUTO DE INFORMÁTICA

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Máquina CONVEX C210

primeiros passos:

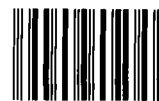
utilização e programação

por

Tiaraju Asmuz Diverio

Relatório do estágio de aperfeiçoamento tecnológico,
realizado em abril e maio na UFSC.
Projeto 46.0139/91-6 Rhae INFO CNPq

Porto Alegre, junho de 1991.



UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Informática - SBU
CONVEX
Software numérico

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

Nº CHAMADA		Nº REG.:
FL 2052		36231
		DATA: 25/06/91
ORIGEM: D	DATA: 20/6/91	PREÇO: Cr$ 3000,00
FUNDO: II	FORN.: PROF. TIARAJÚ DIVÉRIO	

AGRADECIMENTOS

Agradeço

Ao Programa do CNPq de Formação de Recursos Humanos em Áreas Estratégicas (RHAE), que financiou este estágio na Universidade Federal de Santa Catarina, em Florinópolis;

Ao Departamento de Ciências Estatísticas e Computação da UFSC, pelo acolhimento e pelos recursos que me foram colocados a disposição, em especial, ao chefe do departamento Prof Jovelino Falqueto e ao Professor Luiz Fernando Maia;

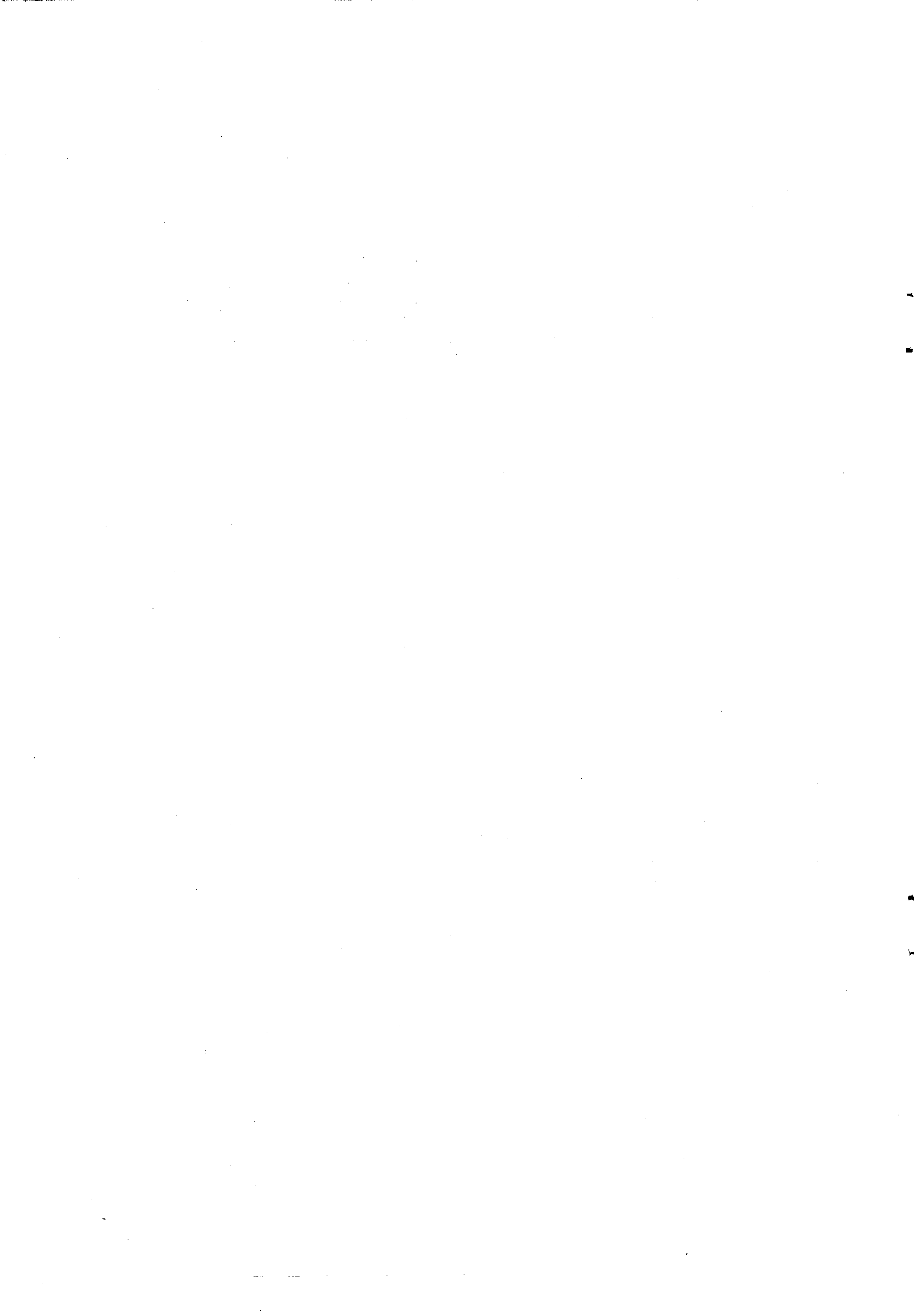
Ao Núcleo de Processamento de Dados, em especial ao pessoal do suporte: Edison Melo, Elvis e Valmira, pela paciência nos momentos de esclarecimentos, pela compreensão nos momentos de greve e pela amizade em todos os momentos.

Ao Prof Dr Clovis Barcelos, ao Jun Fonseca e ao Armando do Departamento de Engenharia Mecânica (GRANTE), pelas dicas sobre o uso do Convex e pela orientação na resolução de problemas por eles estudados.

Aos colegas do Departamento de Computação, aos professores Fernando, Elizabeth, Carmen, Marta, Zancanella, Julio, Olinto, Pimenta, Friedrich, De Lucca entre outros, pela amizade, simpatia e pela calorosa acolhida;

Ao amigo, orientador, colega e chefe do Departamento de Informática Teórica da UFRGS, Prof Dr Dalcidio Moraes Claudio pela liberação para estágio e pela aulas ministradas em minha substituição;

Ao Prof Dr Philippe Navaux pela amizade, pelo empenho em conseguir o estágio e pela orientação.



SUMÁRIO

LISTA DE FIGURAS 4

INTRODUÇÃO 5

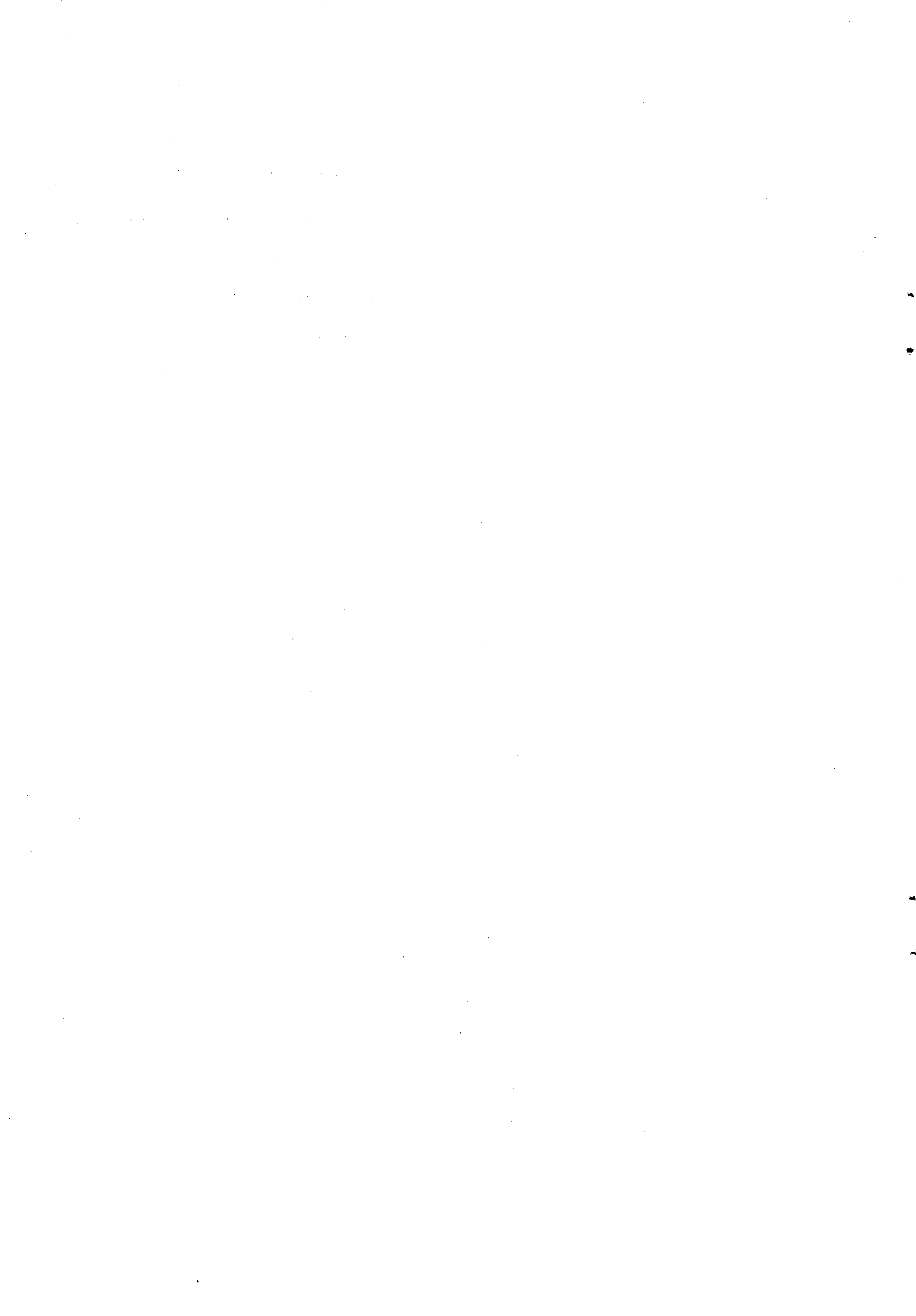
1 A UNIVERSIDADE FEDERAL DE SANTA CATARINA 6
1.1 Histórico e organização da Universidade 6
1.2 Departamento de Ciências Estatísticas e da Computação. 7
1.3 Núcleo de Processamento de Dados - NPD 8

2 O SISTEMA CONVEX 11
2.1 Descrição do sistema Convex 11
2.1.1 O Convex da UFSC 14
2.2 Sistema Operacional ConvexOS 14
2.2.1 Comandos de manipulação de diretórios 16
2.2.2 Comandos para mostrar o conteúdo dos arquivos . 17
2.2.3 Comandos para manipulação de arquivos 17
2.2.4 Comandos para impressão de arquivos 17
2.2.5 Outros comandos úteis 17
2.3 Compiladores do Convex 18

3 USANDO O CONVEX 21
3.1 Formas de acesso e login/logout 21
3.2 Criando e editando arquivos 24
3.2.1 Sintaxe dos comandos 27
3.2.2 Recuperando de enganos e cancelando comandos .. 27
3.2.3 Movendo através de arquivos 27
3.2.4 Adicionando e retirando textos 29
3.2.5 Outras ações com o editor vi 30
3.2.6 Término de edição - saindo do vi 31
3.3 Usando o compilador Fortran 31
3.3.1 O processo de compilação 32
3.3.2 Chamada do compilador fc 34
3.3.3 Definições padrões do compilador fc 35
3.3.4 Controlando mensagens e listando 35
3.3.5 Chamando o depurador/ferramentas profile 37
3.3.6 Controlando o nível de otimização 37
3.3.7 Acréscimo de capacidade 37
3.3.8 Controlando fpp e arquivos objetos 37
3.4 Um exemplo completo 38
3.5 Biblioteca de rotinas vetoriais VECLIB 41
3.5.1 Operações com vetores densos 42
3.5.2 Operações com vetores esparços 42
3.5.3 Operações com matrizes 43
3.5.4 Solução de equações lineares 43
3.5.5 Convulsões e FFT 43
3.5.6 Outras rotinas 43

ANEXOS 45

BIBLIOGRAFIA 57



LISTA DE FIGURAS

Figura 2.1 Lista de filas do Convex C210 da UFSC 13

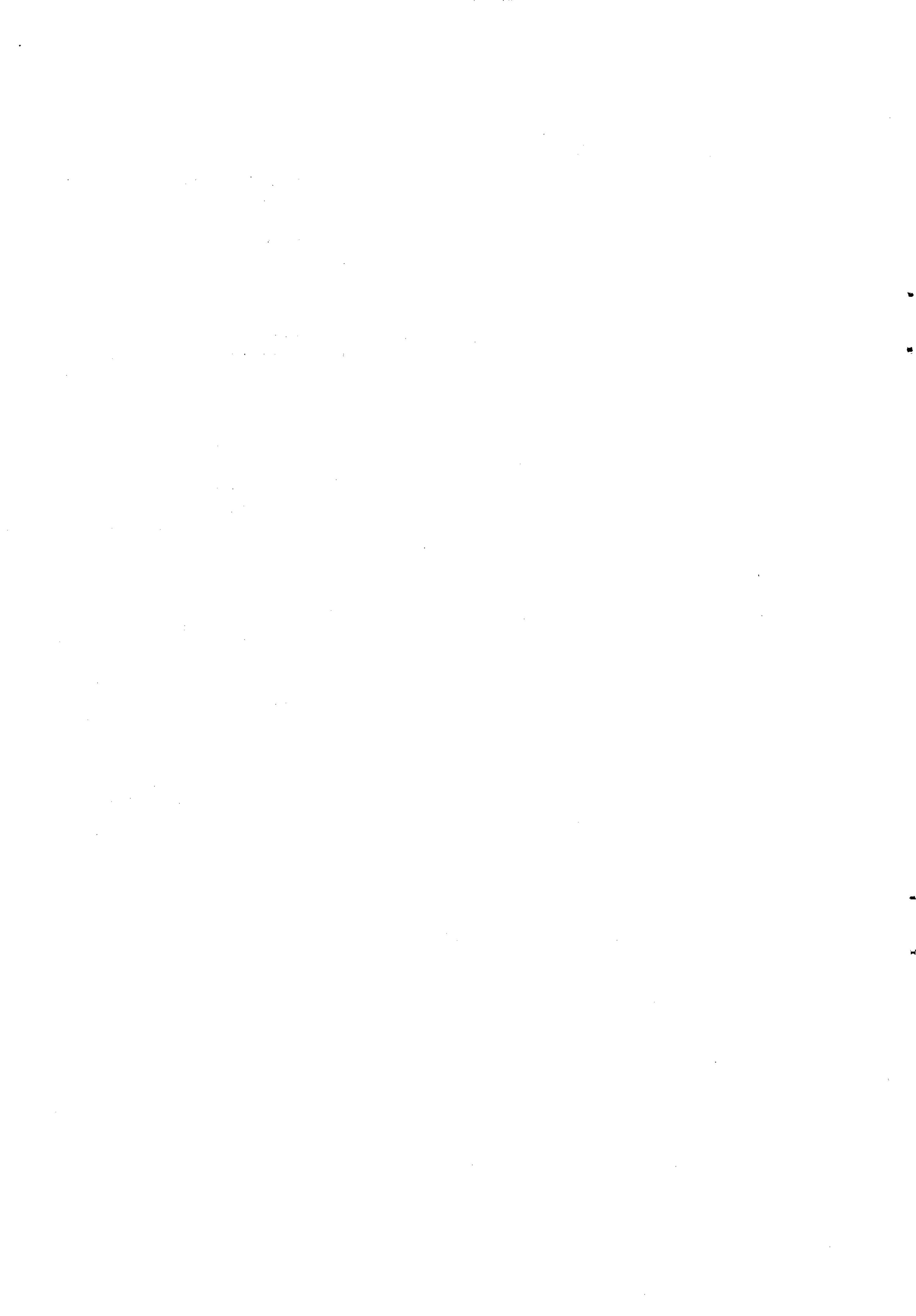
Figura 2.2 Estrutura do Compilador 18

Figura 3.1 Modos do editor vi 26

Figura 3.2 Tabela de comandos de movimentação do cursor .. 29

Figura 3.3 Processo de compilação 33

Figura 3.4 Exemplo-forma de obter o tempo de processamento 44



INTRODUÇÃO

Este documento é um dos resultados do 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 (RHAE), cujo projeto foi entitulado: **Aperfeiçoamento em processamento vetorial, uso da máquina Convex C210 e intercâmbio Cultural**, (processo Nro 46.0139/91-6 RHAE INFO) e que se realizou nos meses de abril e maio de 1991.

Este documento inicia com uma caracterização da UFSC e dos recursos computacionais disponíveis que estão localizados no NPD e nos demais centros. Apesar deste capítulo não ser de relevância científica para o trabalho, achou-se necessário apresentar a UFSC, mais específicamente o DCEC e seus recursos computacionais, onde o trabalho foi desenvolvido.

No segundo capítulo existe 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 vetorizaçãc.

Por fim, 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.

1 A UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC

1.1 Histórico e Organização da Universidade

A Universidade Federal de Santa Catarina foi criada pela Lei Nro 3.849 de 18/12/60. Ela era composta de seis faculdades. Em 1969, as faculdades foram extintas e foram criados os centros com departamentos.

A UFSC tem seu campus na cidade de Florianópolis, no sub-distrito da Trindade (na Ilha) em uma área de 1.007.420m2, distando do centro aproximadamente 7 Km. Nele se encontram dez dos onze centros, além das demais unidades administrativas e de apoio como restaurante universitário, biblioteca, bancos, DCEs e Imprensa Universitária.

A administração universitária a nível superior efetiva-se através de Orgãos Deliberativos Centrais (Conselho Universitário, Conselho de Ensino, Pesquisa e Extensão e Conselho de Curadores) e Orgãos Executivos Centrais (Reitoria, Vice-Reitoria e Pró-Reitorias).

A administração a nível de Unidades, Sub-unidades se efetiva através de Orgãos Executivos Setoriais (Conselhos Departamentais e Departamentos) e Orgãos Executivos Setoriais (Diretoria dos Centros e Chefia de Departamento).

Atualmente na UFSC existem onze centros com 56 departamentos. Entre os centros destaco o Centro Tecnológico, no qual existem oito departamentos, a saber:

Departamento de Ciências Estatisticas e da Computação-DCEC
Departamento de Arquitetura e Urbanismo
Departamento de Engenharia Civil
Departamento de Engenharia Elétrica
Departamento de Engenharia Mecânica
Departamento de Engenharia de Produções e Sistemas
Departamento de Engenharia Química
Departamento de Engenharia Sanitária

e o Centro de Ciências Fisicas e Matemáticas, onde existem três departamentos: o da Física, o da Matemática e o da Química.

A UFSC oferece cerca de 65 cursos de graduação, contando opções de licenciatura e bacharelado e as diferentes habilitações dos cursos.

A UFSC possui o seu acervo bibliográfico na Biblioteca Central, situada dentro do campus com um acervo estimado de 195.000 volumes. Ela está em processo de informatização desde 1986 e faz parte da rede nacional de dados bibliográficos (BIBLIODATA/CALCO) coordenada pela Fundação Getúlio Vargas.

Na área de informática a Biblioteca possui 836 títulos de livros e um total de 78 assinaturas de periódicos correntes em várias áreas.

1.2 Departamento de Ciências Estatísticas e da Computação

O Departamento de Ciências Estatísticas e da Computação (DCEC) foi criado em 1970. Atualmente é coordenado pelo Prof Jovelino Falqueto. O DCEC é um dos oito departamentos do Centro de Tecnologia. Nele estão reunidos 52 professores e pesquisadores, 4 com título de doutor e 36 com titulo de mestre (dentro destes 10 estão cursansdo doutorado).

Ao DCEC está vinculado o curso de graduação de Bacharelado em Ciência da Computação e, desde 1987, vem sendo oferecido um curso de Pós-graduação, a nível de especialização. Está previsto para março de 1992 o inicio do curso de mestrado em Computação na UFSC.

Entre as linhas de Pesquisa destacam-se Rede de Computadores, Microcomputadores Dedicados, Inteligência Artificial e Linguagens Formais e Compiladores. Os principais projetos de pesquisa em andamento, são:

SOPISE - Software para implementação de sistemas especialistas, consiste em desenvolver uma concha LISP para sistemas que usam regras de produção, com inferência não monotônica e fuzzy (Profs Renato e Marcia Rabuske).

Uma interface em linguagem natural de aplicação genérica, consiste em implementar no computador um software que reconhece linguagem natural. A parte de análise sintática e léxica estão prontas, a análise semântica vem sendo implementada (Profs Renato e Marcia Rabuske).

Microcomputadores Dedicados. O desenvolvimento de hardware e software básico; a implementação da linguagem Forth para desenvolvimento de aplicativos em microcomputadores dedicados; a implementação de uma rede de computadores em base do padrão RS485. (Prof Hermann Lucke).

Ambiente para ensino e desenvolvimento de compiladores, o trabalho visa dar suporte às disciplinas de compiladores nos programas de graduação e pós-graduação, bem como implementar ferramentas visando a criação de uma estrutura básica de desenvolvimento e pesquisa na área de linguagens de programação e compiladores (Prof Olinto Furtado).

Implantação de um sistema de mensagens baseado em computador. O sistema de mensagens desenvolvido no LISHA compreende a implantação de um agente de usuário (AU), um agente de transferência de mensagem (ATM) e do protocolo de submissão e entrega (P3). O AU fornece os serviços necessários para a criação, manutenção, recuperação e emissão das mensagens do usuário. O ATM implementa as funções de distribuição e transmissão das mensagens do sistema. O P3 estabelece e gerencia a comunicação entre o AU e um ATM. Todas as especificações basearam-se na recomendação X.400 do CCITT.(Profs Elizabeth Specialski e Gur Dial).

Sistema de desenvolvimento de protocolos em redes de computadores. Ampliação da rede do laboratório de integração de software e hardware (LISHA) e o desenvolvimento de um analisador de redes de Petri por invariantes (Profs Elizabeth Specialski, Julio Szaremeta e Gur Dial).

Ambiente para ensino de programação concorrente, visa a criação de um ambiente para uso no ensino de programação concorrente. Tal ambiente visará auxiliar o aluno na formalização dos conceitos de programação concorrente e dar ao mesmo experiências no desenvolvimento de sistemas concorrentes.(Luis Carlos Zancanella, Olinto Furtado e Jose Mazzucco Jr).

Entre os recursos computacionais disponíveis no DCEC, em particular, destacam-se os oito laboratórios: de Inteligência Artificial, de Integração Software e Hardware, de Pesquisa e Pós-Graduação, de Software Educacional e Gráfico, de Circuitos Digitais, os dois de sistemas e a sala de terminais. Eles possuem cerca de trinta microcomputadores e 14 terminais IBM e umas seis impressoras.

Entre as expansões previstas para os laboratórios do DCEC está a aquisição de algumas estações de trabalho Sun Parc de 12Mb ligadas a um potente servidor de rede de 32Mb e conectado via rede aos mainframe Convex C210 e IBM 3090, ambos com facilidades de processamento vetorial.

1.3 Núcleo de Processamento de Dados - NPD

O NPD é o local onde os recursos computacionais da Universide são congregados. A Divisão de Suporte Técnico do NPD é quem gerencia o acesso de usuários aos recursos computacionais da UFSC e fornece treinamentos de utilização destes recursos.

Entre os recursos computacionais disponíveis destacam-se:

- Convex C210 com processamento vetorial;
- IBM 3090 modelo 170J com processamento vetorial;
- Estações de trabalho de alta resolução gráfica;
- Impressora Laser e plotter;
- Gateway para Rede Bitnet;
- Gateway para Rede Renpac.

A divisão de suporte vem trabalhando para implantação da rede local da UFSC que tem por objetivo a integração de todos os computadores existentes no campus e a integração dessa rede local com as redes públicas e de pesquisa existentes no Brasil e no exterior.

Com este projeto, um usuário poderá se logar, via terminal remoto localizado em qualquer ponto da UFSC, em qualquer um dos computadores localizados no NPD. Outra vantagem é o correio eletrônico local e com redes públicas e de pesquisa.

A UFSC é um nó adjacente a FAPESP (Fundação de Amparo a Pesquisa do Estado de São Paulo) e o **acesso a rede Bitnet** é possível através de qualquer terminal ou micro ligado ao computador IBM 3090.

Todos os docentes, pesquisadores, pós-graduandos e administradores da UFSC podem ser usuários da Bitnet, para tanto necessitam solicitar o código de acesso a Divisão de Suporte Técnico do NPD. O código do usuário, que permite acesso a rede Bitnet, possui uma convenção, na qual o código indica o departamento e as iniciais de usuário. Para recebimento de mensagens na rede, é necessário acrescentar "@BRUFSC.BITNET". Um exemplo é o E-mail: CEC1TAD@BRUFSC.BITNET, que significa que o usuário é de departamento de Ciencias Estatísticas e da Computação e as iniciais dso usuário são T A D (Tiaraju Asmuz Diverio).

O **Mainframe IBM 3090** é do modelo 170 J, possui uma memória de 32 Mb e uma arquitetura 370/XA. Possui 18.5 Gigabytes em discos, 4 unidades de fita carretel (IBM 3420), duas unidades de fita cartucho (IBM 3480) e duas impressoras locais (uma de 1100 e outra de 600 linhas por minuto).

O IBM 3090 possui 4 controladoras de terminais locais com capacidade de ligação de até 32 terminais cada e três controladoras de terminais remotos com capacidade de 8 terminais cada. Existem 97 terminais e/ou micros espalhados entre 11 departamentos do campus com acesso ao IBM.

Os sistemas operacionais disponíveis no IBM são o VM/XA - Virtual Machine/ Extended Architecture e o MVS/XA - Multiple Virtual Storage/ Extended Architecture.

Entre as linguagens de programação destacam-se: os compiladores Fortran (VS V2, IV e o WATFIV), Linguagem C, Mumps VM, Lisp, Prolog, Pascal, Cobol, PL/1, Assembler 370 e 8080 e a linguagem REXX.

Existem várias bibliotecas estatísticas, matemáticas e gráficas disponíveis, algumas delas são: SAS, SPSS, SHAZAN, IMSL, LINPACK, WATLIB e GKS.

O **Mainframe CONVEX C210** tem um processador, memória de 64 Mb, 5.4 Gigabytes de disco, uma unidade de fita carretel e uma impressora com capacidade de 600 linhas por minuto. Existem duas controladoras de terminais locais assincronos com capacidade de ligação de até 16 terminais cada. Atualmente, existem 8 terminais remotos entre cinco departamentos da UFSC e 8 terminais no NPD.

O sistema operacional é o CONVEXOS, que é a implementação do sistema UNIX BSD 4.2 de Berkeley. É um sistema multiusuário, com capacidade para multiprocessamento, memória virtual de até 2 Gigabytes (um giga para usuários).

O Convex pode gerenciar processamento paralelo de até 4 CPU's e processamento vetorial. A equipamento disponível na UFSC dispõe de apenas uma CPU com capacidade de utilizar processamento vetorial. Existem as linguagens C, Fortran e Ada para o Convex, na UFSC estão disponíveis as linguagens C e Fortran com vetorização automática, além da biblioteca vetorial VECLIB.

Neste capítulo, procurou-se dar uma caracterização do ambiente acadêmico da UFSC, do DCEC e dos recursos computacionais disponíveis no NPD. A seguir, será abordado com mais detalhes o sistema Convex C210.

2 O SISTEMA CONVEX

2.1 Descrição do sistema Convex

A introdução da série dos supercomputadores C200, da Convex é tido como uma importante contribuição para o mundo do processamento paralelo.

A série Convex C200 inclue quatro sistemas, dependendo do número de processadores que possue. Eles são:

- CONVEX C210 uma CPU (processamento vetorial);
- CONVEX C220 duas CPUs (processamento paralelo);
- CONVEX C230 três CPUs;
- CONVEX C240 quatro CPUs.

A arquitetura destas máquinas estão estritamente relacionadas com máquinas de memória compartilhada MIMD (Multiple Instructions Stream, Multiple Data Stream).

No Brasil, atualmente, existem três supercomputadores Convex instalados; um no Rio de Janeiro na UNYSIS e outro em Florianópolis na Universidade Federal de Santa Catarina (UFSC), ambos C210. O terceiro está instalado em São Paulo na USP, é um C220, com dois processadores possibilitando o paralelismo.

Esta máquina da USP encontra-se ligada a Rede Bitnet, através do nó da Fapesp. Na verdade, o Convex está ligado como um terminal servidor do computador da Fapesp, o que, no momento ainda é uma limitação para usuários remotos, que tem que se logar na Fapesp, para então, conectar-se ao terminal servidor (Convex).

O sistema da UFSC não está ainda ligada a rede, mas existem o projeto de ligação com a Bitnet. Este sistema será tomado de exemplo, em virtude do estágio em Florianópolis, que permitiu este estudo do sistema Convex e das potencialidades de seu uso na Matemática Computacional.

Define-se por **throughput** o número de tarefas que podem ser completadas em uma unidade de tempo. Este número reflete o poder computacional de uma máquina.

O maior objetivo do processamento paralelo do Convex é aumentar o **throughput** em um ambiente de multiusuários enquanto reduz o tempo de espera para executar um único programa.

A série Convex C200 combina uma arquitetura inovadora, suportando um sistema operacional e uma paralelização automática do compilador Fortran, que sempre trabalha para obter um eficiente atendimento a multiusuários e processamento paralelo.

O sistema é composto de uma memória central compartilhada de 32 a 2048 Mbytes. A memória pode ser acessada simultaneamente por cinco portas de 200 Mbytes/segundos (cada porta tem o tamanho de 64 bits).

O banco de memória arbitrário é controlado por uma memória **crossbar** que funciona como uma central telefônica; cada processador poder receber qualquer banco de memória desempedido. Pode haver até 64 bancos na configuração da memória. Uma das cinco portas é dedicada ao trafego I/O, as outras quatro são para processamento.

Os processadores são completamente simétricos, cada um pode ser interrompido e o sistema operacional prevê primitivas de sincronização para permitir a execução em paralelo de qualquer número de processadores.

A **Automatic Self-Allocation Processor - ASAP** é a pedra angular da arquitetura da série Convex C200, que foi projetada para que cada processador, independentemente, procure e execute o próximo pedaço de trabalho disponível, assim que possível. Portanto, a arquitetura de alocação-propria automática dos processadores, executa dinamicamente jobs assim que existem CPUs disponíveis. Devido o esquema de hardware trabalhar sem a intervenção do sistema operacional, o tempo para alocar uma CPU é minimo. O tempo em que a CPU fica ociosa é reduzido e possibilita um paralelismo com granularidade fina.

Para melhor descrever uma alocação ótima dos recursos, mostrar-se-á o mecanismo providenciado pelo processador, para solicitar a outro processador que o ajude na execução da sequência de instruções do fluxo de instruções.

A requisição para um processador auxiliar na execução de uma sequência de instruções é feita por um conjunto de instruções de hardware, as quais desempenham a operação chamada FORK. A instrução FORK é gerada pelo compilador sobre determinadas porções de código (geralmente loops) para ser paralelizada. Durante a execução, a instrução FORK faz uma requisição de processador ao conjunto de registradores de comunicação. Esta instrução é feita apenas em poucos ciclos de máquina na arquitetura do Convex; portanto este processo é disparável imediatamente após a requisição do Fork ser atendida, outro processo no sistema tem a oportunidade de começar a executar seu código paralelo. O processador que executa a instrução Fork continua executando a próxima instrução na sequência.

Um processador pode estar em dois estados; disponível ou executando. No estado disponível o processador sucessivamente verifica com o registrador de comunicação, procurando por uma requisição de Fork, que indica a existência de trabalho para ele fazer. Quando esta requisição é encontrada, uma tarefa começa a ser executada.

O processador continua operando o fluxo de instruções. Para cada iteração de um loop, a variável de indução é carregada, incrementada e armazenada em uma única operação indivisível. Se, depois de carregada e incrementada a variável de indução for

maior que o valor terminal do loop, o loop será extinto. Na saída o processador executa a instrução de junção, JOIN. Ela checa para ver se existe alguma outra tarefa sendo executada e avisa o processador. Se existir, o processador torna-se disponível e inicia checagem de requisição fork no registrador de comunicação. Se não há outra tarefa executando para este processo, então o processador simplesmente continuará a execução serial com a próxima instrução do fluxo de instruções.

O esquema de habilitação do uso de hardware do sistema operacional é eficiente, pois é baseado numa organização dos jobs em filas de execução por prioridades. A CPU processa os jobs a partir do topo das filas.

Um exemplo destas filas é dado na figura 2.1, onde são mostradas as possíveis filas de execução de jobs no Convex C210 da UFSC.

FILA	RUN	CPU min	WORKING SET Mbytes	NICE [-20,20]	PERMFILE (Mbytes)	PR [0,63]
q1	4	1	5	0	20	43
q3	3	3	15	0	20	47
q5	2	5	25	0	20	51
q9	2	9	45	0	20	55
qE	1	30	100	0	20	59
qF	1	60	100	0	20	61
qG	1	90	100	0	20	61
qZ	1	24h	100	0	20	61

RUN: Número máximo de pedidos que podem rodar simultaneamente
CPU: Limite de tempo máximo de CPU por processo
WORKING-SET: Limite de memória
NICE: Determina a proporção de tempo de CPU alocado para um processo relativo a todos os outros processos do sitema.
PERMFILE: Limite de tamanho de um arquivo criado por qualquer processo dentro do pedido.

Fig. 2.1 - Listas de filas do Convex C210 da UFSC

O compilador automaticamente identifica sequências de código que podem ser executados simultaneamente e insere instruções para habilitar o código a ser executado em paralelo. Quando o programa paralelizado atinge uma seção paralela, ele

requere CPU adicional. Se existir disponível, o job as usa; caso contrário, o job continua executando com uma única CPU. Quando o programa termina, a porção paralela desta execução torna-se disponível para serem utilizadas por outros jobs. Desta maneira, vários jobs podem fazer um uso mais eficiente da arquitetura paralela do Convex.

O Convex tem disponível os compiladores ADA, C e Fortran. Os dois últimos tem a habilidade de paralelizar e vetorizar automaticamente. No sistema Convex da UFSC, estão disponíveis as linguagens C e Fortran com opção de vetorização e automática, além da biblioteca de instruções vetoriais VECLIB.

A função mais importante do compilador Fortran 5.0 do Convex é a habilidade para paralelizar e vetorizar loops e iterações sem modificar o código. O compilador Fortran paraleliza os programas pela geração de código que requer multiplas CPUs para processar diferentes iterações de um loop escalar. Devido ao Convex ter alocação própria de CPUs e o número de CPUs que trabalham em um pedaço ser dinâmico; a extensão para que o código execute em paralelo depende do número de CPUs disponíveis no momento da execução do código.

O compilador entretanto gera código paralelo que é independente do números de CPUs disponíveis. Se executar o código paralelizado com uma única CPU (Convex C210), a execução paralela é impossível, entretanto o código inserido pelo computador para admitir a execução é simplesmente retirado.

2.1.1 O Convex da UFSC

O computador Convex da UFSC é do modelo C210 tem um processador, memória de 64 Mb, 5.4 Gigabytes de disco, uma unidade de fita carretel e uma impressora com capacidade de 600 linhas por minuto. Existem duas controladoras de terminais locais assincronos com capacidade de ligação de até 16 terminais cada. Atualmente, existem 8 terminais remotos entre cinco departamentos da UFSC e 8 terminais no NPD.

2.2 Sistema Operacional ConvexOS

O sistema operacional do Convex é o CONVEXOS, uma versão do UNIX derivada do UNIX 4.2 BSD de Berkeley, que suporta até quatro processadores. Cada processador é um 40-nanosegundos ECL/CMOS do tipo Cray.

O modelo C210, existente na UFSC, possui apenas um processador. Pode-se utilizá-lo de dois modos; um único usuário (single user) ou múltiplos usuários (multi-user). Com múltiplos usuários o sistema verifica periodicamente o espaço de arquivo dos usuários, o número de usuários e o tempo de processamento dos usuários.

É um sistema multiusuário, com capacidade para multiprocessamento, memória virtual de até 2 Gigabytes (um giga para usuários).

O sistema UNIX é composto por vários programas que o acompanham. Eles podem ser divididos em duas classes; a de utilitários integrados e de ferramentas. A classe de utilitários integrados é necessária para a execução, enquanto que as ferramentas são opcionais. Entre as ferramentas pode-se citar planilhas eletrônicas, processadores de palavras e correio eletrônico.

Entre as principais características do sistema ConvexOS destacam-se as capacidades multitarefas, multiusuários, transportabilidade, comunicação e correio eletrônico e as bibliotecas de softwares aplicativos.

A **capacidade multitarefa** indica que se pode executar mais de uma tarefa ao mesmo tempo, ou seja, permite que tarefas que eram executadas sequencialmente sejam executadas simultaneamente, possibilitando que o conjunto de tarefas seja executado mais rapidamente e que a máquina esteja disponível para executar mais coisas. Possibilita a vetorização eo paralelismo de tarefas.

A **capacidade multi-usuário** permite que vários usuários usem o computador simultaneamente através de diferentes terminais conectados ao computador. Cada usuário pode executar programas, acessar arquivos e imprimir documentos, uma vez que o sistema operacional gerencia os pedidos que os vários usuários fazem ao sistema, evitando que um interfira a outro e atribui prioridades aos pedidos dos vários usuários.

A **transportabilidade** do sistema reside na propria portabilidade do Unix entre diferentes máquinas.

O software de **comunicação** é parte integrante do sistema operacional do Convex. As comunicações poden ocorrer entre terminais de uma máquina, entre máquinas diferentes de uma instalação ou de diferentes intalações (outros lugares).

Como o sistema ConvexOS é uma versão do sistema Unix, a seguir será dado uma breve caracterização dos módulo ou programas que o compõem. Ele pode ser dividido em **Kernel, Shell e os aplicativos e ferramentas.**

O **Kernel** é o administrador dos recursos do sistema. Ele administra interrupções, memória, armazenagem de dados, dispositivos de entrada e saída (I/O) e a gerência de processos.

O **Shell** é o programa que conecta e interpreta os comandos digitados por um usuário. Ele interpreta os pedidos do usuário, chama programas em memoria e executa-os individualmente.

O programa Shell interpreta os comandos do sistema operacional digitados e os traduz para o Kernel, funcionando como um tradutor.

A seguir é apresentado uma relação de comandos do Convex com a descrição de sua função. Para saber maiores detalhes consultar o manual ou solicitar ajuda ao sistema operacional pelos comandos **info** ou **man**.

É importante se ter idéia da estrutura de arquivos do sistema, para saber onde se está trabalhando. A estrutura de arquivos é baseadas em diretórios. O diretório inicial é o **RAIZ**, representado pela barra /. Deste se derivam seis subdiretórios principais, que são:

- **bin** Onde ficam armazenados os comandos mais utilizados do sistema operacional Unix;
- **etc** Onde ficam armazenados os programas de administração do sistema;
- **usr** Onde ficam todas as contas dos usuários. Em cada conta pode haver seus diretórios, sub-diretórios e arquivos;
- **lib** Onde ficam os arquivos da biblioteca C;
- **dev** Onde ficam os arquivos de dispositivos;
- **tmp** Onde ficam os arquivos armazenados temporariamente.

A seguir são fornecidos alguns conjuntos de comandos de manipulação de diretórios, de verificação de conteúdo de arquivos, de manipulação de arquivos, de impressão de arquivos e outros comandos úteis.

2.2.1 Comandos de manipulação de diretórios

ls <opção> arquivo - lista os arquivos de um diretório;
opções: -a : lista os arquivos invisíveis;
-l : lista os arquivos juntamente com suas caracteristicas (permissão, tamanho, dono, data de alteração);
-s : mostra o tamanho do arquivo em Kbytes;
-F : mostra o tipo de arquivo (* executáveis e / diretórios).

mkdir <nome-diretorio> - cria um novo diretório;

rmdir <nome-diretorio> - remove um diretório vazio;

rm -ri <nome-diretorio> - apaga todos os arquivos do diretório, solicitando a confirmação a cada arquivo a ser deletado;

pwd - mostra o diretório corrente;

cd <nome-diretorio> - move para outro diretório.

2.2.2 Comandos para mostrar o conteúdo dos arquivos

cat -n <nome-arq> - mostra um ou mais arquivos na tela, para numerar a linha usar opção n.

more (-c) (+num-linha) <nome-arq> - mostra o conteúdo do arquivo por telas. As opções -c mostra topo da tela e +num-linha a linha inicial que deve ser mostrada.

less <nome-arq> - mostra o arquivo na tela.

2.2.3 Comandos para manipulação de arquivos

cp (-i) <arq-origem> <arq-destino> - Copia o arquivo origem para o arquivo destino. A opção i resguarda, no caso de já existência de arquivo com o nome do destino, solicitando confirmação.

rm (i) <nome-arq> - deleta o arquivo, com a opção de confirmação;

mv (i) <arq-origem> <arq-destino> - move arquivos entre diretórios

2.2.4 Comandos de impressão de arquivos

lpr (-#n) (-p) <nome-arq> - Imprime arquivos. A opção n indica o número de cópias e a opção p é de paginação.

lpq - checa o estado da impressora fornecendo a sua posição na lista de espera.

2.2.5 Outros comandos úteis

script (-a) <nome-arq> - salva em um arquivo tudo que aparecer na tela durante uma determinada sessão. É necessário digitar ctrl-d antes de encerrar a sessão.

chmod DGO <nome-arq> - Muda a permissão de acesso de arquivos e diretórios. O acesso pode ser em relação ao dono D, ao grupo G ou a outros usuários O. O estado é marcado pela soma dos modos: 4 para leitura; 2 para escrita e 1 para execução. exemplo: chmod 764 programa.f

psswd - troca o password secreto de acesso.

info - ativa sistema informativo.

finger - informações detalhadas sobre um ou mais usuários.

vi <nome-arq> - cria ou edita arquivos. Ativa o editor de texto do Unix.

fc <nome-arq>.f -O n -o<nom-modulo> -lveclib - ativa o compilador fortran. O arquivo deve ser um ponto f, ou seja código fonte fortran. A opção n indica o nível de otmização (0 sem; 1 escalar; 2 vetorial e 3 paralela). A opção o é para pré-compilação.

Maiores informações sobre os comandos podem ser obtidos nos manuais de sistema operacional ConvexOS/Unix.

2.3 Compiladores do Convex

O compilador consiste de três fases principais, uma linguagem específica denominada **Front End**, um otimizador e um gerador de código. O otimizador e o gerador de código formam juntos uma linguagem comum denominada **Para o fim (Back End)**. O otimizador é largamente independente de máquina. As dependências são isoladas no Gerador de código. O Front End para as Lingugens C, ADA e Fortran já foram implementadas. Cada compilador tem a mesma habilidade para vetorizar e paralelizar programas, exceto o compilador ADA Front End, que não tem a paralelização automática. A estrutura da família do compilador é ilustrada na figura 2.2.

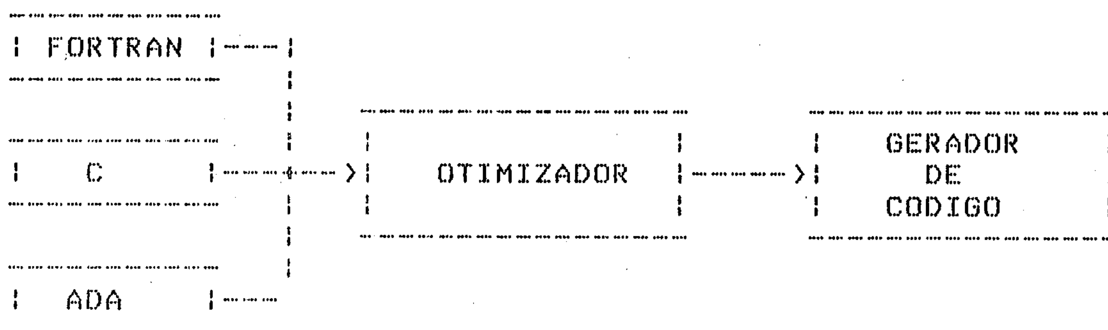


Fig. 2.2 - Estrutura do Compilador

A mais importante transformação desempenhada pelo Compilador 5.0 Back End são as transformações de reordenação. Transformação de reordenação não eliminam operações de um programa e nem substitui operações simples, mas as rearranja de modo que possam ser mais eficientemente executadas.

Estas transformações são feitas graças a uma sofisticada análise de dependência, que determina que limitações são requeridas de forma que as operações sejem executadas para garantir a fidelidade do intuito do programa original.

Transformações de reordenação planejadas pelo otimizador inclue: vetorização, vetorização parcial, paralelização, distribuição de loops e mudanças internas de loops. Estes ítens serão exemplificados no capítulo 3. Estas transformações são orquestradas pelo compilador 5.0 para obter alto grau de eficiência na geração de código.

O compilador permite vetorização parcial pela reestruturação de um loop, onde a porção que inibiu a vetorização pode ser separada em um loop que será executado escalarmente.

Mudanças interiores no loop consistem na reestruturação de um par aninhado de loops, onde o loop externo se torna o interno, ou o interno se torna o externo. Estas mudanças são permitidas pelo compilador quando é verificada recorrência no loop interior ou se for verificado um aumento da quantidade de iterações no loop interior vetorizável ou, ainda se verrificado que para o loop intero o passo de vetorização é mais eficiente.

A distribuição de loops consiste da reestruturação de um par de loop aninhado para que o cálculo vetorizável seja removido do loop externo e que o novo loop vetorizado seja gerado para manipular os cálculos do loop externo. A distribuição de loops ajuda a minimizar o efeito negativo de se poder vetorizar somente o loop mais interior.

Estruturas condicionais não necessariamente inibem a vetorização quando encontradas em loops, pois todas as tarefas de cada ramo da condição são mantidas e atendidas nos registradores vetoriais.

Alguns loops podem ser vetorizáveis mesmo que se o resultado das operações não são armazenados em aarays de dados. A redução vetorial de instruções na linguagem assembler permite que cálculos sejam realizados em vetores de operandos e os resultados sejam armazenados em registradores escalares, como por exemplo: soma de todos os elementos de um vetor, multiplicação de um escalar por todos os elementos de um vetor, cálculo do máximo e mínimo de um vetor. O compilador usa redução vetorial sempre que possível para possibilitar a vetorização.

A análise de dependência é um fator importante, pois a dependência é uma limitação na hora de executar duas operações. A transformação de reordenação é válida se a nova ordem corfirmar todas as dependências da velha ordem. Análise de dependência descobre e representa a dependência e dependências potenciais de um programa.

No Fortran existem quatro tipos de representações de inteiros, essas formas de representação variam com o núnero de bytes que são utilizados para representar o número. Estes formatos usam um, dois, quatro e oito bytes, sendo descritos por 1x, 2x, 4x ou 8x. A representação em ponto-flutuante pode usar 32 ou 64 bits. Na representação de 32 bits, o intervalo de representação é 2.9387358*10^(-38) a 1.7014117*10^38, uma precisão aproximada de sete dígitos decimais. Já a representação de ponto-flutuante de 64 bits, tem um intervalo de representação de 5.562684646268003*10^(-309) a 8.988465674311584*10^307, o que implica numa precisão aproximada de 15 dígitos decimais.

O compilador 5.0 usa a Desigualdade de Banerjee e o Teste GCD na análise de dependências como descrito em [ALL83] com mínimas modificações. Estes testes são montados com poucos **ad-hoc** testes e complementados por manipulação simbólica.

3 USANDO O CONVEX

Até este momento, já foram apresentadas diversas informações do sistema Convex, seu sistema operacional e o que se pode fazer com eles, mas restam perguntas práticas para que se possa começar a trabalhar, a programar nesse supercomputador. Entre estas perguntas, destacam-se:

- Como se acessa ao Convex?

- Como se cria e edita programas em Fortran e C?

- Como se compila os programas?

- Como ter acesso a lista de erros da compilação e uma listagem do código fonte?

- Como se executa os programas compilados?

- Como se utiliza a opção de vetorização, o otimizador?

Nos próximos itens, pretende-se responder a estas perguntas, sem muito formalismo, dando os principais comandos e alguns exemplos.

3.1 Formas de acesso e login/logout

O acesso ao Convex da UFSC, atualmete só pode ser feito através de terminais próprios do Convex ou de micros Pc ligados por cabos ao Convex, que emulam terminais. Existe o projeto de colocação do Convex na Rede, o que permitirá seu acesso por outras instituições.

Para se ter acesso, tem-se que ter um **login**, um código de acesso (a máquina virtual do IBM ou o usercode do VAX), o qual é fornecido pelo administrador ou gerente do sistema, que no caso é o Departamento de Suporte Técnico do NPD. Na definição do login do usuário está embutido o departamento em que ele trabalha e suas iniciais, análogamente como foi definido no IBM3090.

O ConvexOS diferencia os caracteres maiúsculos dos minúsculos, por isto é importante deixar o teclado sempre em minúsculo. Todos os comandos devem ser digitados em minúsculo.

No caso de utilização de um terminal do Convex, ao ligar aparecerá, automatifcamente, a mensagem informando a versão do sistema operacional e solicitando que voce introduza seu login.

Se voce estiver num dos microcomputadores PC que estão ligados ao Convex, será necessário acionar o programa VTKermit (do winchester ou de disquete) que define o protocolo de comunicação entre o PC e o Convex, de forma que o PC se comporte como um terminal.

O programa VTKermit, é uma versão local do protocolo Kermit, que serve para previnir a adulteração dos dados por interferências da linha de comunicação e sincronizar a comunicação de computadores que se entendem, podendo trocar informações de controle, ao mesmo tempo em que estão transferindo dados.

O Kermit é um protocolo para transferências confiáveis de arquivos entre computadores, através das linhas comuns de telecomunicações usadas para conectar os terminais aos computadores. Para que se estabelea um protocolo Kermit, um programa Kermit deve estar rodando em cada ponta da linha de comunicação (uma ponta no host-Convex e outra no micro-PC).

Para se efetivar uma comunicação, deve-se verificar se a velocidade de transmissão em bit por segundo, está definida de forma que seja a mesma velocidade nas duas pontas. Esta velocidade de transmissão é chamada de BAUD-RATE, ou simplesmente BAUD. No caso do Convex é de 9600.

Deve-se verificar e definir por qual das portas de comunicação do micro, a comunicação se realizará. No caso específico, o modem foi instalado na Com2.

A paridade é uma técnica empregada pelos equipamentos de comunicação para detectar erros numa base caracter-por-caracter. O Kermit, geralmente, não usa paridade para carcteres. Deve-se então defini-la como NONE, que indica que os 8 bits são de dados e não tem paridade.

Para se transferir arquivos de um equipamento para outro, como por exemplo, do Convex para um PC (que emula terminal), tem-se que ativar o kermit nas duas máquinas e, então,

da máquina que se quer enviar digitar o comando SEND <nome-arq>. Voltar a máquina que deve receber e digitar RECEIVED e, a transferência se efetuará. Maiores detalhes podem ser encontrados no manual do Kermit.

Na comunicação do Convex com um terminal emulado(PC), para efetivar a comunicação, deve-se ativar o programa VTKermit digitando **vtkermit**, quando aparecerá a mensagem:

```
VTKermit 1.0 running under MS - DOS --  Type ? or Help for help
VTKermit>
```

É necessário definir algumas coisas, como foi visto, para possibilitar a conecção. Basta digitar os comandos:

```
VTKermit>    set port com2
VTKermit>    set parity none
VTKermit>    set baud 9600
```

Caso voce queira verificar se foram definidas ou como estão definidas as diretrizes de comunicação, poderá usar o comando **status**, que produzirá a seguinte tela:

```
VTKermit>    status


BAUD rate is 9600                          PARITY is NONE
Communications  PORT: 2                    ESCAPE character is ^]
LOCAL-ECHO is OFF                          LOG is OFF
FLOW-CONTROL is XON/XOFF                   HANDSHARE is NONE
file DESTINATION is DISK                   DEFAULT-DISK is A:
WARNING is ON                              Ring BELL after transfer
EOF mode is NDCTRL-Z                       INCOMPLETE file: DISCARD
TIMER is ON                                TAKE-ECHO is OFF
8-bit quoting done only or request         END-OF-LINE character is ^M
7171-MODE is OFF                           DISPLAY permitted to be color
BLOCK-CHECK is 1-CHARACTER CHECKSUM        AUTOWRAP-MODE is OFF
SEND entrl char prefix:        #           RECEIVE entrl char prefix:  #
SEND start-of-packet char:    ^A           RECEIVE start-of-packet char:   ^A
SEND TIMEOUT (seconds):       10           RECEIVE TIMEOUT (seconds):      10
SEND PACKET-LENGTH:           80           RECEIVE PACKET-LENGTH:          94
SEND # of PAD chars:           0           RECEIVE # of PAD chars:          0
MODE IS COMMAND
VTKermit>
```

Observa-se nas duas primeiras linhas as diretrizes BAUD, Communications e Parity como foram definidas.

A conexão com o Convex é feita pelo comando **connect**. Ao se digitar, a tela será trocada e depois de alguns segundos, pressione a tecla **enter**, então aparecerá, em caso de conexão a tela do Convex.

VTKermit> connect
<enter>

ConvexOS, Release V8.1 (uxnpd01)

login:

Voce deverá então, introduzir seu código de acesso seguido de sua senha, que por ser secreta não aparecerá na tela, como mostra a sequência abaixo.

login: cecitad

password: xxxxxxxxx

Após ter introduzido seu login e sua senha corretamente, será mostrado a data e a hora da última vez que voce usou seu login e aparecerá o comando prompt %, indicando que voce pode utilizar os recursos do computador, ou seja, está aguardando que voce digite os comandos.

Poderá aparecer outras mensagens informando a existência de mensagens ou de mail para voce.

Após o uso, quando voce tiver terminado de utilizar o computador, deve-se digitar o comando que desliga a sua sessão no Convex (**logout**) e, ainda, retornar ao VTKermit (**CTRL-] C**) e desativá-lo através do comando **quit**.

Resumindo:

% logout

CTRL] C

VTKermit> quit

> (de volta ao DOS)

3.2 Criando e editando arquivos

Sabe-se que um arquivo é uma coleção de informações, as quais são armazenadas em um disco e referenciada por um nome de arquivo. Arquivo texto é tipicamente constituído de memorandos, artigos, programa em código fonte, mensagens ou qualquer coisa constituída como uma série de caracteres (alfabéticos, numéricos, etc). Qualquer arquivo no qual você digita dados é um arquivo texto.

O editor de texto proporciona uma maneira fácil de colocar um texto em um arquivo ou mudam texto já armazenado em um arquivo. Os arquivos podem ser criados pelo comando **cat**, mas não é muito prático, especialmente, para quem não é um bom digitador, pois uma vez digitado <enter>, usando o comando cat, a linha é introduzida e não pode ser modificada, a não ser que você delete o arquivo e o digite novamente.

Os editores de texto resolvem este problema. Eles permitem que se crie e modifique arquivos de textos. Pode-se não apenas acrescentar textos, como: deletar, mudar e inserir textos em arquivos já existentes. Se você se enganar no digitar, basta utilizar o editor para mudar o caracter específico, ou palavra, ou linha.

Todos os editores de texto fazem uma cópia temporária de um arquivo e permitem que você faça mudanças na cópia. Esta cópia temporária é chamada de **buffer de edição**. Um buffer de edição armazena uma cópia do arquivo na memória do computador. Qualquer coisa que se faça no arquivo, mesmo adicionar texto ou retirar parte não desejada é feito no buffer, na memória. O arquivo original somente é modificado quando você escrever o conteúdo do buffer de volta no arquivo (salvar o buffer editado).

O uso de buffer para editor de texto traz algumas importantes vantagens como o aumento da velocidade de edição, pois o computador pode trabalhar rapidamente com dados armazenados na memória. Outra vantagem é a preservação do conteúdo do arquivo original no caso de algum problema. Também, isto leva a você salvar as mudanças quando você quer escrever o mesmo ou diferente arquivo. Uma preocupação é salvar as mudanças periódicamente, pois se não forem salvas e você sair do editor sem salvar as alterações, elas serão todas perdidas.

O editor vi opera em dois modos; o modo de comandos e o modo de edição. Quando estiver no modo de comandos, o editor assume que tudo que for digitado é um comando para ser executado sobre o arquivo. Alguns exemplos de comandos são: pesquisa por um texto, substituição de textos, troca de lugar de texto, deletar parte do texto.

Quando no modo de edição, o editor assume que tudo que for editado deve ser armazenado no arquivo. No canto esquerdo, abaixo da tela, aparece um caracter I, indicando o modo de edição (Input), para trocar de modo deve ser precionada a tecla ESC. Para retornar ao modo de edição, deve-se precionar a tecla I. Durante uma sessão de edição, voce precisará trocar frequentemente de modos.

O modo de comandos é o modo de operações, onde o editor interpreta tudo que for digitado como um comando a ser executado no buffer. O modo de edição é o modo de operação onde o editor armazena tudo o que for digitado no buffer como um texto.

Os editores são divididos em três categorias: editores de linha, editores de tela e batch editor. Os dois primeiros são editores interativos, ou seja, quando deseja um comando, você vê o resultado imediatamente. Você pode executar quantos comandos quiser e depois sair. Um editor por batch por outro lado, opera com um arquivo com um conjunto específico de comandos e sai sem qualquer intervenção. São pouco utilizados.

Editor orientado por tela mostra uma tela do arquivo por vez. Eles permitem que se veja uma porção do arquivo na tela do terminal e se modifique caracteres, palavras e linhas, simplesmente pelo digitar nela, na posição corrente do cursor. Deve-se posicionar o cursor na tela para que o texto seja modificado e então, adicionado ou alterado no texto.

Editores orientados por telas introduzem a característica de rolagem (scroll), o que permite que se mova uma simples linha do texto. Mudanças são visíveis na tela assim que são feitas, verificando o efeito causado.

O editor vi é um editor de texto orientado por telas e está disponível em todos os sistemas Convex. Para auxiliar a aprendizagem do uso do vi, é visto como acessar o vi, uma noção dos modos, a sintaxe dos comandos, como mover-se no texto, como modificar textos, como recuperar o texto de enganos e como sair do editor vi.

Para iniciar uma sessão de edição, deve-se ser digitado

% vi <nome-arq> [enter], onde nome-arq é o nome de um arquivo já existente ou de um arquivo que se quer criar.

Em baixo da tela, pode aparecer a linha estatus. Ela não é parte do texto e permite que você saiba qual o arquivo que está editando, o número de linhas e o número de caracteres no arquivo.

Quando é ativado o editor vi, o modo de operação disponível é o de comandos. Enquanto estiver neste modo, não se pode adicionar textos ao arquivo, é necessário passar para o modo de edição. A figura 3.1 resume as tarefas que podem ser realizadas em cada um desses modos.

Modo de Comandos	Modo de Edição
mover o cursor	adicionar novo texto
procurar por certo caracter	corrigir erros de digitação
mudar caracter,palavra,linha	inserir textos
remover caracter,palavra,linh	abrir nova linha p/edição
fazer mudanças globais	mudar ou trocar textos
cancelar comandos	

Fig.3.1 Modos do editor vi

A tecla ESC serve para retornar do modo de edição para o modo de comandos e para cancelar um comando parcial, retornando ao modo de comandos.

3.2.1 Sintaxe dos Comandos

A sintaxe dos comandos do vi é:

[quantidade]operador [quantidade]objeto,

onde operador é o comando, a tarefa a ser feita e o objeto é a entidade que sofrerá a ação. O argumento [quantidade] é opcional e tem como efeito repetir certo número de vezes a ação ou o objeto afetado pela operação. Se não for especificado o operador o cursor se posicionará no objeto. Exemplos de operadores são: d (delete), c (change), f (find), r (replace). Exemplos de objetos incluem: w palavras,) sentenças, } paragrafos. Um comando para deletar a próxima palavra seria feito por dw.

Exemplos usando quantidade são dados à seguir, em todos eles o efeito final sobre o texto será a retirada de 12 palavras do texto.

12dw - deletar uma palavra 12 vezes;
d12w - deletar 12 palavras;
3d4w - três vezes deletar 4 palavras;
4d3d - quatro vezes deletar 3 palavras.

O operador é identificado por uma letra, em geral a primeira letra da ação em inglês.

3.2.2 Recuperando de enganos e cancelando comandos

O editor de texto vi providência um comando que cancela a execução de comandos, restaurando o arquivo no estágio antes da mudança. O comando é undo, invocador por u no modo de comandos. Ele só restaura do último comando. Invocado por U será restaurada a linha corrente, independente de quantas modificações haviam sido feitas nela.

3.2.3 Movendo através de arquivo

Uma das coisas básicas quando se edita um arquivo com o editor vi é como mover o cursor pela tela e através do arquivo. Pode-se mover o cursor e acrescentar caracteres, palavras, linhas, frases e parágrafos. Só é necessário mover o cursor para a posição desejada e executar o comando desejado que altera o arquivo.

O movimento mais básico é por caracteres. O movimento por caracteres é controlado pelas letras abaixo ou pelas setas.

k - movimenta para cima;
j - movimenta para baixo;
h - moviemnta para esquerda;
l - movimenta para direita.

Estas letras podem ser precedidas por uma quantidade, o que indicará que deverá mover esta quantidade de linhas ou caracteres na direção da operação desejada.

O editor vi permite, também, que se mova entre palavras num simples passo. Uma palavra é definida como um conjunto de caracteres separdos por espaços ou pontuações. O movimento por palavras é ativado pelas letras abaixo.

w - move o cursor para o início da próxima palavra no arquivo;
e - move o cursor para o final da próxima palavra no arquivo;
b - move o cursor para o início da palavra anterior no arquivo;
W - move o cursor para o início da próxima palavra, sem parar na pontuação;
B - move o cursor para o início da palavra anterior, sem parar na pontuação;

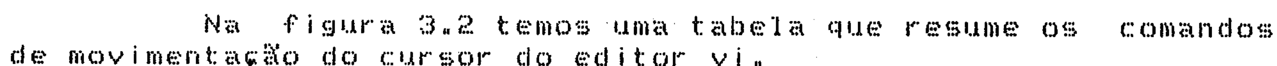
O editor também permite que se mova pelo arquivo, através do início e final de linhas. Os comandos que permitem a movimentação por linhas é ativado por:

<return> - move o cursor para o início da nova linha;
0 - move o cursor para o início da linha corrente;
$ - move o cursor para o final da linha corrente;

Quando o cursor está em cima ou em baixo da tela, ao se tentar mover por linhas pode ocasionar na rolagem da tela. O editor vi possui seis comandos para manipular a rolagem da tela.

ctrl-E rola uma linha para baixo;
ctrl-Y rola uma linha para cima;
ctrl-d rola meia tela para baixo;
ctrl-u rola meia tela para cima;
ctrl-f rola uma tela adiante;
ctrl-B rola uma tela para trás.

Na figura 3.2 temos uma tabela que resume os comandos de movimentação do cursor do editor vi.



OBJETO	COMANDO	AÇÃO
caracter	l	um caracter a direita
	j	um caracter para baixo
	k	um caracter para cima
	h	um caracter a esquerda
palavra	w	uma palavra a direita (inicio)
	e	uma palavra a direita (fim)
	b	uma palavra a esquerda (inicio)
	W	uma palavra a direita (sem pontuação)
	B	uma palavra a esquerda (sem pontuação)
linha	return	inicio da proxima linha
	+	inicio da proxima linha
	-	inicio da linha anterior
	0	inicio da linha corrente
	$	final da linha corrente
sentenca	(	inicio da sentenca anterior
	)	inicio da nova sentenca
paragraf	}	inicio do novo paragrafo
	{	inicio do paragrafo anterior
	[[	inicio da nova rotina C
	]]	inicio da rotina anterior
rolagem	ctrl-E	abaixa uma linha
	ctrl-Y	sobe uma linha
	ctrl-D	abaixa meia tela
	ctrl-U	sobe meia tela
	ctrl-F	abaixa uma tela
	ctrl-B	sobe uma tela
outros	H	Home cursor no canto superior esquerdo
	M	cursor no meio da tela
	L	cursor na ultima linha da tela

Fig. 3.2 Tabela de comandos de movimentação do cursor

3.2.4 Adicionando e retirando textos

A principal razão de se utilizar um editor é para criar novos arquivos ou para adicionar texto a um arquivo já existente. Para adicionar novo texto usando vi, voce necessita estar no modo de edição. No modo de edição tudo o que for digitado é colocado no arquivo, até que se precione a tecla ESC.

As três formas de adicionar texto, são: append, insert e open. Append adiciona o texto depois da posição corrente do cursor, na mesma linha. Insert adiciona o texto antes da posição corrente do cursor, na mesma linha. E open cria uma linha em branco para adicionar texto, onde o cursor é posicionado no inicio da linha. As formas de ativar os comandos de adição de texto são dados pelas letras abaixo.

a - insere o texto após a posição do cursor;
A - insere o texto no final da linha que contem o cursor;
i - insere o texto antes da posição do cursor;
I - insere o texto no inicio da linha que contém o cursor;
O - cria uma linha em branco antes da linha corrente, possibilitando a adição de texto;
o - Cria uma linha em branco abaixo a linha corrente possibilitando a adição de texto;
ESC - Termina a edição do texto, voltando ao modo de comandos.

Para remover caracteres, palavras, sentenças, linhas ou blocos de textos, devem ser utilizados os comandos:

x - deleta o caracter abaixo do cursor;
d - deleta o objeto seguinte ao cursor;
D - deleta do cursor até o final da linha;
dd - deleta a linha corrente.

3.2.5 Outras ações com o editor vi

Em arquivos grandes, pode-se querer mover o cursor para um local específico do arquivo, através da procura e posicionamento em um caracter ou grupo de caracteres chamados de string. Para a localização de strings o vi proporciona dois comandos: /[string] que procura para frente pelo string e o comando ?[string] que procura para atrás o string.

As duas funções básicas para modificações são substituição (replace) e troca(change). O editor vi as utiliza para distinguir operações com caracteres de operações com objetos. Os comandos para substituição de caracteres são:

r - substitui um caracter pelo próximo digitado;
R - substitui n caracteres por n novos caracteres digitados
s - substitui n caracteres por m novos caracteres. Necessita da marca $ para delimitar o final do string a ser trocado;
~ - troca maiúsculo por minúsculo e vice-versa;

Os comandos de troca de objetos são similares ao comando s. Eles substituem um número específico de objetos do texto pelo que for digitado. É necessário marcar o final do string a ser substituído pelo caracter $. Ao sair do modo de edição a troca se efetuará.

O editor vi providência um buffer temporário especial que armazena o texto removido do arquivo utilizado no último comando. Os comando que se valem disto são:

d - deleta o texto do arquivo e coloca no buffer;
y - copia o texto do arquivo e coloca no buffer;
p - coloca o texto armazenado no buffer no arquivo após a posição do cursor;
P - coloca o texto armazenado no buffer na arquivo antes da posição do cursor.

3.2.6 Término de edição - saindo do vi

Portanto o editor vi armazena o arquivo temporariamente num buffer, isto necessita que você salve, periódicamente, o conteúdo do buffer. O vi providência comandos para realizar isto e para sair do editor, esses comandos são:

:w - escreve o conteúdo do buffer para um arquivo. Se for especificado o nome do arquivo, será escrito nele, caso contrário é escrito no arquivo corrente.
ZZ - escreve o conteúdo do buffer para o arquivo corrente e sai do editor.
:q! - sai do editor sem salvar o conteúdo do buffer.
:wq - salva e termina a sessão.

3.3 Usando o compilador Fortran

Neste item é apresentado uma visão geral sobre o compilador Fortran, que é análogo aos compiladores C e Ada e, é mostrado os comandos básicos para invocar o compilador.

Quando se invoca o compilador, tem que indicar na própria linha de comando os nomes dos arquivos a serem compilados. Estes arquivos tem extensões, que são padrões e indicam o tipo de arquivo que se está trabalhando, por exemplo:

<nome-arq>.f - arquivos contendo programas fontes em Fortran

<nome-arq>.c - arquivos contendo programas fontes em C

<nome-arq>.a - arquivos contendo programas fontes em Ada

<nome-arq>.s - arquivos contendo programas fontes na linguagem Assembler

<nome-arq>.o - arquivo contendo código objeto. Pode ter sido gerado pela compilação de Fortran, C ou Ada. Isto facilita a junção de programas ("lincagem").

O compilador Fortran (assim como o C e o Ada), transforma um arquivo fonte contendo um ou mais programas em um arquivo objeto. O arquivo objeto pode ser unido com uma biblioteca de programas ou com outros arquivos objetos (extensão ponto o) e executados num computador Convex. Arquivos previamente compilados em Fortran podem ser unidos (lincados) a outros programas, gerados por outras linguagens e, executados.

A forma de invocar o compilador Fortran é:

fc <nome-arq>.f -O n -o<nom-modulo> -lveclib - ativa o compilador fortran. O arquivo deve ser um ponto f, ou seja código fonte fortran. A opção n indica o nível de otmização (0 sem; 1 escalar; 2 vetorial e 3 paralela). A opção o é para pré-compilação.

Os arquivos processados pelo compilador Fortran fc são: os de código fonte em fortran (arquivo.f), os módulos objetos (arquivo.o), os códigos fontes em linguagem assembler (arquivo.s), os módulos objetos de bibliotecas (arquivo.a) e os de código intermediário (arquivo.fil). O compilador pode processar arquivos especificados pelo comando #include.

Na criação e digitação de um arquivo em fortran, deve-se atentar as colunas. As colunas de 1 a 5 são para labels, rótulos; a coluna seis só deve ser utilizada para indicar que a linha é continuação da anterior. As colunas úteis para os camandos são da 7 a 72. As colunas finais (73 a 80) são para numerar as linhas, técnica especialmente útil em programas grandes. Uma linha de comentário deve iniciar com o caracter "d" na primeira coluna.

3.3.1 O processo de compilação

Arquivos contendo código fonte em fortran e arquivos especificados pelo comando #include podem ser pré-processados pelo pré-processador **fpp**. Se for especificado pela opção -E é gerado uma saída.

O compilador **fc** recebe arquivos contendo códigos fontes pré-processados, arquivos intermediários e arquivos assembler e, pode gerar ou desencadear funções, dependendo da opção utilizada, por exemplo: <opção - ação>

-sc	Verificação sintática;
-il	Geração de arquivo intermediário (Inlining) .fil;
-S	Geração de arquivos assember .s;
-c	Geração de arquivo executável .o.

O código objeto é transmitido ao carregador **ld**, que pode colocar a disposição os arquivos de bibliotecas (.a) gerando código executável em **a.out**.

A figura 3.3 mostra o processo de compilação do compilador fc.

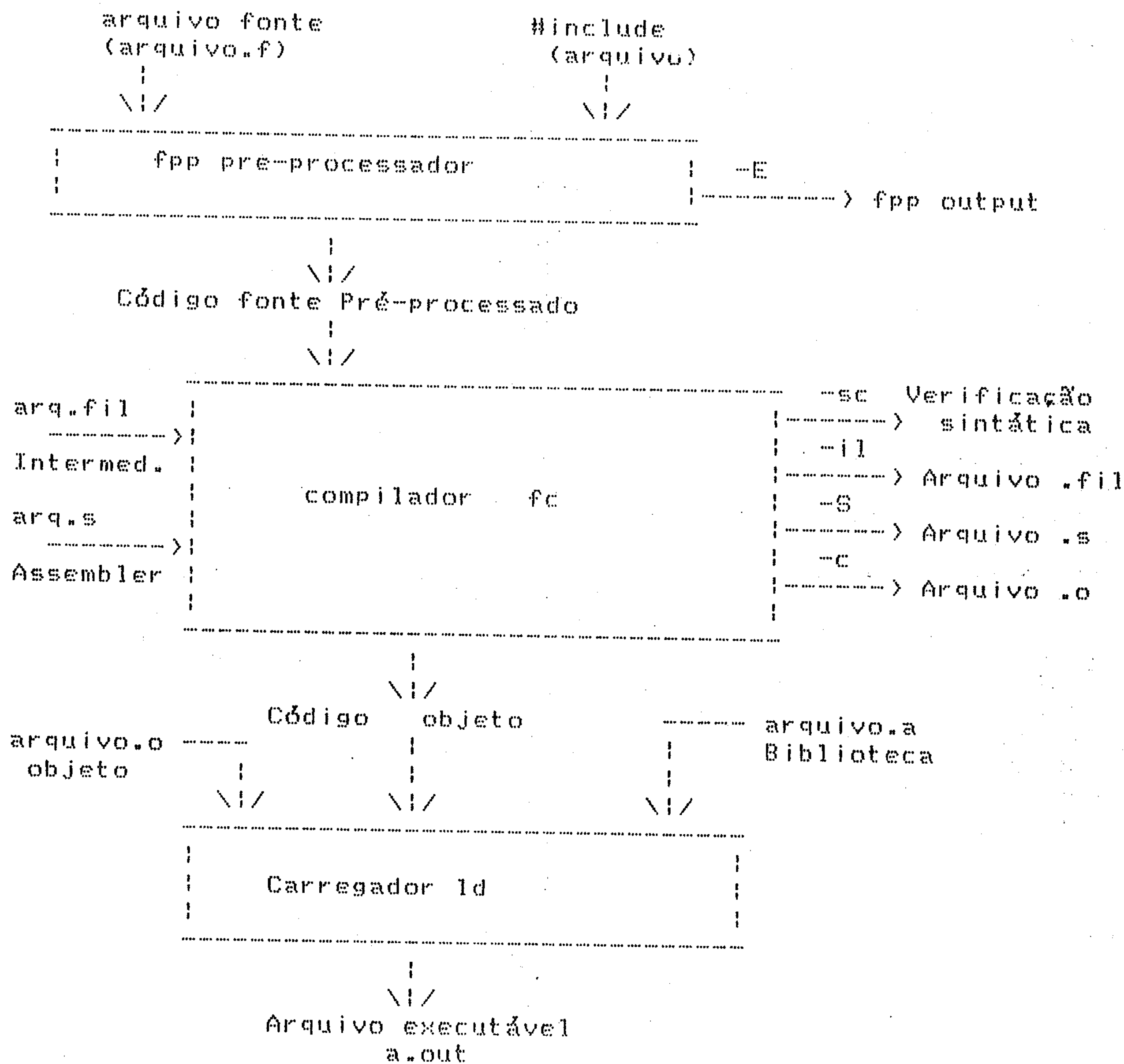


Fig. 3.3 Processo de compilação

Arquivos fontes individuais podem ser compilados com opções diferentes e então lincados posteriormente. Rotinas com referências externas podem ser compiladas separadamente usando as opções -c ou -S na linha de comando do fc. O comando **split** pode ser utilizado para separar o programa por unidades, em rotinas.

A opção -c causa que o compilador gere arquivos objetos com a extensão .o; a opção -S causa que o compilador gere código assembler com a extensão .s. Invocando o fc com arquivos .o ou .s ocasionará que o código seja reunido, lincado e produzido o módulo executável a.out.

3.3.2 Chamada do compilador fc

A forma completa da invocação do compilador fc é dada por:

% fc [opção] <arquivos> [opção-de-carga]

onde,

[opção]	é opcional e consiste de opções do compilador, do pré-processador e do assembler;
<arquivos>	consiste da lista de arquivos a ser processada;
[opções-de-carga]	é opcional e são usadas para avisar ao compilador que deve lincar com que biblioteca, para que ela fique disponível.

A maneira mais simples de compilar um programa em Fortran, supondo que o arquivo fonte seja teste.f, é digitando:

% fc teste.f

O programa será compilado e o código objeto executável será armazenado no diretório corrente, com o nome de a.out. Caso você queira definir o nome para o programa executável, deve digitar:

% fc teste.f -o teste1

Para compilar e juntar com módulos já previamente compilados, digita-se:

% fc teste.f test-mod1.o test-mod2.o -o teste2

Para executar um programa compilado, basta digitar o nome da versão executável.

3.3.3 Definições padrões do compilador fc

Linhas de comandos iniciadas com o caracter "d" na primeira coluna são tratados como comentário.

As declarações sem especificação de tamanho de variáveis, são tomadar por: integer = integer*4; real = real*4 e logical = logical*4.

Arquivos objetos (arq.o) só são salvos se os arquivos fontes são compilados e lincados.

3.3.4 Controlando mensagens e listando

Entre as opções do compilador encontram-se as opções que controlam mensagens, entre elas se destacam:

-E	Executa somente fpp e envia o código pré-processado para stdout;
-S	Gera código assembler (.s) para cada unidade do programa;
-na	Suprime todas as mensagens diagnósticas;
-nv	Suprime todas mensagens de resumo de vetorização;
-nw	Suprime todas as mensagens diagnósticas de espera;
-vn	Identifica a versão do compilador utilizada;
-xr	Gera tabela de referência cruzada e envia para o stdout;
-iw N	(implica em xr) N especifica o tamanho da coluna para o identificador (padrão: 16);
-sl	(implica em xr) Produz uma listagem fonte com o numero de linha;
-pw N	(implica em xr) N especifica o tamanho da página (padrão: 132).

As mensagens geradas durante a compilação podem ser de sete tipos: do fpp: mensagem de erro ou de advertência; do fc: mensagem de erro, de advertência, de aconselhamento, de resumo de vetorização e de erro interno do compilador.

O formato das mensagens do fpp e do fc são:

fpp: <nome-arq>: <num-linha>: <mensagem>

fc: <tipo> on line <num-linha>.<pos> of <nome-arq>:<mensagem>

onde,

<tipo>	erro, aviso, conselho;
<num-linha>	número da linha onde se localiza o erro;
<pos>	posição do caracter depois da compressão do fpp;
<mensagem>	diagnóstico.

Mensagem de resumo de vetorização é gerada para cada unidade do programa, ela contém: o número da linha do início do loop; a variável de iteração; o rótulo de comando por loop; o inicio, a parada e valores de passo para cada variável de iteração e; uma mensagem informando que foi vetorizada.

Os erros são identificados pelo compilador e o diagnóstico com as mensagens de erro são mostrados na tela do terminal. Pode-se redirecionar estas mensagens de erro para um arquivo usando o comando >&, como no exemplo abaixo;

% fc teste.f >& erros

os erros ficarão armazenados no arquivo erros, que pode ser listado ou impresso.

Cabe aqui, lembrar os comandos **cat** e **cp** e sua sintaxe, pois eles são utilizados para criar e para copiar arquivos. O comando **cat** (concatena) mostra o conteúdo de um ou mais arquivos na tela. Quando são especificados dois ou mais arquivos, eles são concatenados na sequência em que foram especificados. A sintaxe do comando é:

% cat -n <nome-arq> a opção -n coloca o número da linha antes de cada linha mostrada. Esta opção é útil para listar o código fonte de um programa. A listagem via comando cat só é conveniente para arquivos pequenos, por que ele lista todo o arquivo antes de parar.

O comando cp duplica um arquivo. Isto é útil para fazer cópias backup de arquivos. A sintaxe do comando é:

% cp -i <arq-origem> <arq-destino> A opção -i solicita uma confirmação, quando já existir o arquivo destino.

Existem duas maneiras de se criar arquivos com as mensagem de erro de compilação. Na primeira constará apenas as mensagens de erro. Na segunda forma as mensagens de erro intercalam o texto, facilitando as correções. As duas formas são dadas a seguir:

```
% cat -n <nome-arq>.f > <nome-arq-erros>
% fc <nome-arq>.f >>& <nome-arq-erros>
```

é produzido uma listagem com a numeração das linhas e os erros, as mensagens de erro são concatenados no final do arquivo;

```
% cp <nome-arq>.f <nome-arq-erros>.f
% fc <nome-arq-erros>.f >& erros
```

as mensagens de erro são intercaladas a listagem do código fonte com numeração de linhas. Como é feito cópia do fonte, não se perde o arquivo original.

3.3.5 Chamando o depurador e feramentas profile

-cs	verifica constantes subscritas no tempo de compilação e gera código que verifica em tempo de execução onde variáveis subscritas são usadas em referências a arrays com os limites dos arrays.
-dc	Trata linhas de código iniciadas com d na primeira coluna como código fonte, apesar de serem comentários;
-db	produz as informações necessárias para o código fonte depurar csd;
-sc	providencia a verificação sintática sem usar o tempo de compilação completo;
-p	Habilita a execução da capacidade que conta o número de vezes que cada rotina é chamada;
-pb	Habilita a execução da capacidade que conta o número de vezes que cada linha é executada;
-pg	Habilita a execução da capacidade que similarmente a opção -p produz mais estatísticas extensivas;
-tl N	N especifica o número maximo de minutos de CPU que será permitido a compilação.

3.3.6 Controle do nível de otimização

-oN	Realiza otimização no nínel N 0 => Otimização escalar local; 1 => Mais otimização escalar global; 2 => Mais vetorização; 3 => Mais paralelização (só com mais de 1 CPU);
-no	Não realiza otimização independente de máquina;
-uo	Realiza otimizações não garantidas;
-il	Gera arquivos intermediários (.fil).

3.3.7 Acréscimo de capacidade

-B[dir]	Usa um compilador substituto residente em dir;
-F66	Usa Fortran66, nas regras de interpretação;
-cft	Usa linguagem como definida no Cray;
-iN	Avisa ao compilador interpretar variáveis inteiras e lógicas do tamanho N (2,4,8);
-rN	Avisa ao compilador interpretar variáveis reais pelo tamanho N (4,8);
-72	Trunca código fonte na coluna 72;
-fi	Usa o formato de ponto flutuante IEEE;
-fr	Usa o formato de ponto flutuante original do Convex;
-fx	Adimite programação nos dois formatos.

3.3.8 Controlando fpp e arquivos objetos

-E Executa somente fpp e envia o código pré-processado para stdout;
-I[dir] Avisa ao fpp onde procurar pelos arquivos incluídos por #include;
-c Cria código objeto sem invocar o carregador;
-o nome Cria arquivo executável em nome no lugar de a.out.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

3.4 Um exemplo completo

Neste item apresenta-se um exemplo completo, para ilustrar desde a chamada do editor de texto vi, até a execução do exemplo e, opcionalmente, da forma vetorizada.

O exemplo abordado é o cálculo de um determinante, por cofatores, de uma matriz 4x4. A matriz exemplo adotada é:

```
        [  3   2   1   -2 ]
  A =   [  9   4  -5   -4 ]
        [ -6  -3  -4   12 ]
        [  3   4   3    9 ]
```

Para criar o arquivo, deve-se chamar o editor vi e atribuir um nome ao arquivo. Como se trata de um código fonte em fortran, deve ter o sufixo .f. Para iniciar a digitação deve-se digitar a letra i. (Ver figura 2.4 e item 2.4.2)

```
% vi execomp.f
      Program main
      real a(4,4),c(3,3)
      real d3,d4
      integer n
      common /data/a,c,n,d3,d4
      data n/4/
      data a/3.,9.,-6.,3.,2.,4.,-3.,4.,1.,-5.,-4.,3.,-2.,-4.,12.,9./
      i=1
      d4=0.0
      do j=1,n
         call mresult(i,j)
         d3=det(c)
         d4=d4+a(i,j)*((-1)**(i+j))*d3
      enddo
      write(*,*) "Matriz A"
      do i=1,n
         write(*,*) (a(i,j),j=1,n)
      enddo
      write(*,*) "determinante =";d4
      end
d
      Function det(c)
      real a(4,4),c(3,3)
      real d3,d4
      integer n
      common /data/a,c,n,d3,d4
      p1=c(1,1)*c(2,2)*c(3,3)
      p2=c(1,2)*c(2,3)*c(3,1)
      p3=c(1,3)*c(2,1)*c(3,2)
      n1=c(1,3)*c(2,2)*c(3,1)
      n2=c(1,1)*c(3,2)*c(2,3)
      n3=c(1,2)*c(2,1)*c(3,3)
      det3=p1+p2+p3-n1-n2-n3
      end
```

```
d
      Subroutine mresult(ti,tj)
      real a(4,4),c(3,3)
      real d3,d4
      integer n,ti,tj
      common /data/a,c,n,d3,d4
      t=1
      z=0
      do f=2,4
         z=z+1
         l=1
         do g=1,4
            if (g.ne.tj) then
               c(z,l)=a(f,g)
               l=l+1
            endif
         enddo
      enddo
      end
```

ESC <para sair do modo de edição>
:wq <para gravar o arquivo e sair do editor, irá aparece uma mensagem dizendo que o arquivo é novo, o número de linhas voltando o prompt do UNIX,

%

Antes de compilar, é aconselhável fazer a cópia, para que a listagem do código e a listagem dos erros sejam gravadas em um arquivo, para que possam ser impressas.

```
% cat -n exemplo.f > teste.f
% fc teste.f >>& teste.f -o testei
```

Em teste.f teremos a listagem com numeração de linhas e os eventuais erros. Caso não tenha ocorrido erros, o código executável será gerado e guardado com o nome de testei. Isto pode ser verificado pela listagem do diretório (% ls) ou pela listagem do arquivo fonte, para verificar a existência de erros.

Como o programa não possui erros, existe o código executável. Quando não é especificado o nome do executável pela opção -o ele é armazenado com o nome a.out. Para executa-lo basta digitar o nome do arquivo a.out ou, como no caso, testei. A execução aparecerá na tela:

```
Matriz A
  3    2    1   -2
  9    4   -5   -4
 -6   -3   -4   12
  3    4    3    9

determinante = 144.0000
```

Pode-se direcioná-la de forma que fique armazenada em um arquivo, para sua posterior impressão:

```
% teste1 > saida

% cat saida

% lpr teste.f saida
```

ou mandá-la diretamente para a impressora:

```
%% teste1|lpr
```

A forma de compilar um arquivo invocando a otmização vetorial, como ja foi visto, é na própria linha do fc:

```
% fc teste.f -o2 teste2
```

quando se utiliza a vetorização é fornecido um relatório estatístico da vetorização.

```
% teste2 >saida2
% cat saida2
% lpr saida2
```

Na figura 3.4 é apresentado a forma de se ter o tempo de execução no programa. Esta forma se utiliza de uma das rotinas da biblioteca Veclib, por isto será apresentada a seguir.

3.5 Biblioteca de rotinas vetoriais VECLIB

Veclib é um conjunto de subrotinas Fortran altamente otimizadas. Esta biblioteca providência um software matemático básico e um núcleo computacional para programas de aplicação. Este conjunto contém subrotinas otimizadas nas áres: operações vetoriais densas e esparças, operações entre matrizes, solução de equações e sistemas de equações lineares, transformadas de Fourier discreta, convulsões, correlações, filtragem e uma miscelânia de rotinas.

A Veclib é um produto opcional da linha Convex. Para se verificar se ela está disponível no sistema, tem-se que mudar para o diretório usr/lib e listar seu conteúdo:

```
% cd /usr/lib
% ls libveclib.a
```

Para ter acesso a biblioteca Veclib em um programa, usa-se a opção -l na linha do comando fc de compilação. Isto instrui ao carregador tornar disponível as rotinas da biblioteca.

Entre as convenções de nomes das rotinas da biblioteca Veclib, destaca-se o tipo de resultados. Eles podem ser produzidos em cinco tipos de dados numéricos:

integer*4	i
real*4	s
real*8	d
complex*8	c
complex*16	z

No uso de alguma das rotinas, o primeiro caracter do nome indica o tipo de dado operado e que ira retornar. Em algumas rotinas deve-se usar dois caracteres, um para o resultado e outro para caracterizar o tipo do operando.

A seguir é apresentado as rotinas da biblioteca Veclib. Na chamada da rotina, a letra T representa o tipo de dado manipulado, que pode ser <i,s,d,c,z> indicando um dos cinco tipos numéricos.

3.5.1 Operações com vetores densos

iTamax	Achar o índice do elemento do vetor de maior magnitude;
iTamin	Achar o índice do elemento do vetor de menor magnitude;
iTmax	Achar o índice do maior elemento do vetor;
iTmin	Achar o índice do menor elemento do vetor;
Tamax	Achar o elemento do vetor de maior magnitude;
Tamin	Achar o elemento do vetor de menor magnitude;
Tmax	Achar o maior elemento do vetor;
Tmin	Achar o menor elemento do vetor;
iTctX	Conta elementos do vetor (X:eq,ge,gt,le,lt,ne) a um escalar;
iTsvX	Acha oíndice do primeiro elementos do vetor (X:eq,ge,gt,le,lt,ne) a um escalar;
TlstX	Cria um array de índices dos elementos (X:eq,ge,gt,le,lt,ne) a um escalar;
Tasum	Calcula a soma em valor absoluto dos elementos do vetor;
Tsum	Calcula a soma dos elementos do vetor;
Tcopy	Copia um vetor para outro;
Tswap	Troca o conteúdo de dois vetores;
Tscal	Calcula o produto de um escalar por um vetor;
Tfrac	Retorna a parte fracionária de um vetor;
Tdot	Calcula o produto interno;
Twdot	Calcula o produto interno ponderado;
Taxpy	Calcula a multiplicação de um escalar por um vetor somado com outro vetor;
Tnrm2	Calcula a norma euclidiana do vetor;
Tnrsq	Calcula a raiz quadrada da norma euclidiana do vetor;
Tramp	Gera uma função linear de salto;
Trot	Aplica uma dada rotação;
Trotg	Constroi uma rotação;
Trotm	Aplica a rotação modificada;
Trotmg	Constroi uma rotação modificada;
Tclip	Realiza um recorte vetorial nos dois lados
Tclipl	Realiza um recorte vetorial no lado esquerdo
Tclipr	Realiza um recorte vetorial no lado direito

3.5.2 Operações com vetores esparços

Taxpyi	Calcula o produto de um escalar por um vetor esparço somando com outro vetor;
Tdoti	Calcula o produto interno de um vetor esparço;
Tgthr	Reune um vetor esparço;
Tgthrz	Reune e zera um vetor esparço;
Troti	Realiza uma rotação esparça;
Tsctr	Dispersa uma matriz esparça;

3.5.3 Operações com matrizes

Tgemm	Realiza uma multiplicação de matrizes;
Tgemv	Realiza uma multiplicação de matriz por vetor;
Tger	Realiza um rank-1 update;
Ttrsv	Resolve sistema triangular;

3.5.4 Operações para solução de equações lineares

Tgeco	Computa fatorização triangular de uma matriz densa;
Tgedi	Calcula o determinante pelo resultado de geco;
Tgefa	Determina fatorização triangular dada elo resultado de gedi;
Tgesl	Resolve sistema de equações lineares;
Tpbco	Calcula a fatorização de Choleski de uma matriz banda;
Tpbdi	Dada fatorização de Choleski, avalia o determinante;
Tpbfa	Calcula fatorização de Choleski de uma matriz banda;
Tpbsl	Resolve o sistema de equações lineares dada a fatorização de Choleski;
Tpoco	Calcula a fatorização de Choleski de uma matriz definida;
Tpodi	Dada uma fatorização de Choleski, avalia o determinante;
Tpofa	Calcula a fatorização de Choleski de uma matriz definida;
Tposl	Resolve um sistema de equações lineares dada a fatorização de Choleski;
Ttri	Resolve um sistema de equações lineares dada uma matriz tridiagonal;

3.5.5 Convulsões e FFT

T1dfft	Realiza 1-dimensão complexa-complexa FFT;
T2dfft	Realiza 2-dimensão complexa-complexa FFT;
T3dfft	Realiza 3-dimensão complexa-complexa FFT;
Tffts	Realiza 1-dimensão simultânea complexa-complexa FFT;
Trcifft	Realiza 1-dimensão real-complexa FFT;
Tconv	Realiza correlação e convulsão;

3.5.6 Outras rotinas

ran	Gerador de número randômico - versão escalar;
ranv	Gerador de número randômico - versão vetorial;
cputime	Tempo da Cpu em microsegundos.

Juntamente com a biblioteca Veclib, estão disponíveis outras bibliotecas, como por exemplo a LSQPACK, que é um pacote de subrotinas altamente otimizadas, em dupla precisão, para resolver problemas de mínimos quadrados.

Pode-se obter o tempo de execução no programa, basta para tanto utilizar a rotina cputime, como no exemplo apresentado na figura 3.4, onde a variável time apresenta o tempo em segundos.

```
real*4 t0,time,cputime

   - - -

t0 = cputime(0.0)

   - - -

time = cputime(t0)
print *, 'Tempo =',time,'segundos'

   - - -

stop
end.
```

Fig. 3.4 Exemplo - forma de obtenção do tempo de processamento

Maiores informações sobre outras bibliotecas podem ser obtidas nos manuais das Bibliotecas, ou utilizando o comando man, nas formas:

```
% man 3f intro              lista das subotinas
% man 3f <nome-rot>         características de uma rotina.
```

ANEXOS

Nos anexos são apresentados seis programas desenvolvidos no Convex C210, para a resolução de sistemas de equações lineares. Juntamente com o programa é apresentado a linha do comando de compilação do programa com e sem otimização vetorial.

Anexo 1: Programa do Método de eliminação de Gauss com pivotamento simples. Programa MEGPS.f

```
      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

      Subroutine init
d Rotina que inicializa os valores
      real a(4,4),b(4),x(4),ab(4,5)
      integer n
      common /data/a,b,x,ab,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

      Subroutine Concab
d Rotina que concatena A com B
      integer n
      real a(4,4),b(4),x(4),ab(4,5)
      common /data/a,b,x,ab,n

      do i=1,n
         ab(i,n+1)=b(i)
         do j=1,n
            ab(i,j)=a(i,j)
         enddo
       enddo
       end
```

```
      Subroutine Tngab
d Triangulariza a matriz ab(n,n+1)

      real a(4,4),b(4),x(4),ab(4,5)
      integer n
      common /data/a,b,x,ab,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

      Subroutine retro
d Rotina que efetua a retrosubstituição
      real a(4,4),b(4),x(4),ab(4,5)
      integer n
      common /data/a,b,x,ab,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

      Subroutine out
d Impressão de resultados
      real a(4,4),b(4),x(4),ab(4,5)
      integer n
      common /data/a,b,x,ab,n

      write(*,*) "Matriz A"
      do i=1,4
         write(*,10) (a(i,j),j=1,4)
 10   format(4(f12.6,3x))
      enddo
      write(*,*)
      write(*,*) "Vetores B e X"
      do i=1,4
         write(*,*) b(i),x(i)
      enddo
      end
```

```
d
d Escolha do pivo e troca de linhas
d
        Subroutine Piv(k)
        real a(4,4),b(4),x(4),ab(4,5)
        integer n,lmaior
        common /data/a,b,x,ab,n
        lmaior=k
        maior=abs(ab(k,k))
        do m=k+1,n
           if (maior .le. abs(ab(m,k))) then
              maior=abs(ab(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
```

Comandos de compilação:

```
fc megps.f -o pmeg >& erro1
fc megps.f -O2 -o vmeg >& erro2

less erro2    - verificar listagem dos erros e de vetorização
lpr erro2     - impressão da listagem de otimização vetorial
```

Anexo 2:

Programa do método de Gauss-Jordan com matriz inversa. Arquivo é MGJOD.f.

```
      Program main
d metodo de gauss-jordan com inversa e piv simples
d
      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
d
d  dados
d
      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
d Concatena A e a Identidade
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+j)=0.0
         enddo
         ainv(i,n+i)=1.0
      enddo
      end
d
d  Triangularizacao da matriz AINV
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

      det=1
      do i=1,n
         det=det*ainv(i,i)
         e1=ainv(i,i)
         do j=1,8
            ainv(i,j)=ainv(i,j)/e1
         enddo
      enddo
      write(*,*) det
      end
c
c  Montagem da matriz inversa
c
      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
c
c Calculo do vetor solucao X por INV * B
c
      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
```

```
d
d Impressao dos resultados
d
       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
       subroutine out3
       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(*,*)
       write(*,*) "teste da inversa"
       do i=1,n
          write(*,10) (ainv(i,j),j=1,n+n)
       enddo
   10  format(8f8.4)
       end
```

Comandos de compilação:

```
fc mgjod.f -o pjod >& erro1
fc mgjod.f -O2 -o vjod >& erro2

less erro2    - verificar listagem dos erros e de vetorização
lpr erro2     - impressão da listagem de otimização vetorial
```

Anexo 3:

Programa do método compacto de Banachieviski para a solução de sistemas de equações. Arquivo MBANC.f.

```
      Program main
      real a(4,4),b(4),x(4),lu(4,4),y(4)
      integer n
d
d Inicializacao de dados
d
       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
d Montagem da LU
d
       do i=1,n
          do j=1,n
             lu(i,j)=a(i,j)
          enddo
       enddo
d
d  Decomposicao LU
d
       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
d Resolucao do sistema LY=B
d
       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
d Resolucao do sistema UX=Y
d
        do i=n,1,-1
           sum=0.0
           do j=1+i,n
              sum=sum+lu(i,j)*x(j)
           enddo
           x(i)=y(i)-sum
        enddo
d
d Impressao do resultados
d
        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
```

Comandos de compilação:

```
fc mbanc.f -o pban >& erro1
fc mbanc.f -O2 -o vban >& erro2

less erro2    - verificar listagem dos erros e de vetorização
lpr erro2     - impressão da listagem de otimização vetorial
```

Anexo 4:

Programa do método compacto de Choleski para sistemas de equações. Arquivo MCHOL.f.

```
d metodo de cholesky
      Program main
      real a(4,4),b(4),x(4),lu(4,5)
      integer n
d inicializacao do 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 matriz 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=1,m-1
               sum1=sum1+lu(i,k)*lu(k,m)
            enddo
            lu(i,m)=lu(i,m)-sum1
         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 dos 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(*,*) "vetores B,X"
      do i=1,n
         write(*,*) b(i),x(i)
      enddo
      end
```

Comandos de compilação:

```
fc mchol.f -o pchol >& erro1
fc mchol.f -O2 -o vchol >& erro2

less erro2   - verificar listagem dos erros e de vetorização
lpr erro2    - impressão da listagem de otimização vetorial
```

Anexo 5: Programa do método iterativo de Gauss-Jacobi. Arquivo MJAC.f.

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

Comandos de compilação:

```
fc mjac.f -o pjac   >& erro1
fc mjac.f -O2 -o vjac   >& erro2
```

Anexo 6: Programa do Método Iterativo de Gauss Seidel.

```
d metodo de gauss-seidel
       Program main
       real a(6,6),b(6),x(6),asc(6)
       real xnovo,ascmin,ascreq,mi
       integer it,limit,n
      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/
       n=6
       ascmin=0.0
       it=0
       mi=0.0000000005
       ascreq=8.0
       limit=50
       do i=1,n
          write(*,10) (a(i,j),j=1,n),b(i)
       enddo
   10  format(6f8.4,f12.4)
       write(*,*)
       do while ((ascmin.lt.ascreq).and.(it.le.limit))
        write(*,20)it,(x(i),i=1,n),ascmin
   20   format(i2,6f12.8,f6.2)
          do i=1,n
             sum=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) asc(i)=0.0
             x(i)=xnovo
             if (i.eq.1) then
                               ascmin=asc(i)
                           else if (ascmin.gt.asc(i)) then
                                    ascmin=asc(i)
                           end if
          enddo
          it=it+1
       enddo
       write(*,20) it,(x(i),i=1,n),ascmin
       end
```

Comandos de compilação:

```
fc msei.f -o psei  >& erro1
fc msei.f -O2 -o vsei  >& erro2
```

BIBLIOGRAFIA

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, Brasilia, 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)