Estudo de algoritmos paralelos e aplicações de medidas fractais em máquina Multi-DSPs

Relatório Final

Bolsa de Iniciação Científica

Rodrigo Daniel Malara
Processo CNPq 520620-96/98-NV

Orientador: Dr. Hanumant Sawant

Departamento de Computação - UFSCar

São Carlos - SP
Julho de 1998

1.Introdução

A necessidade de sistemas computacionais mais robustos, e o fato de que a tecnologia de circuitos integrados está atingindo um ponto, em que sistemas processadores simples (conhecidos como máquinas de Von Newmann), não conseguirão oferecer o poder computacional necessitado pelas diferentes áreas do conhecimento humano.

A partir deste ponto de vista, se torna necessário buscar outros métodos, onde um maior poder computacional é possa ser alcançado. Dentre estas técnicas, temos o processamento concorrente, onde vários sistemas processadores trabalham paralelamente, para atingir um mesmo objetivo de forma mais rápida.

O trabalho desenvolvido teve por objetivo a implementação de algoritmos paralelos para medir a dimensão fractal de imagens, utilizando máquina paralela de alto desempenho, chamada Arquitetura Multiprocessadora usando DSPs (ArqMDSP) composta por 4 DSP's (Digital Signal Processor), modelo 320C40 da Texas Instruments, conectados por um barramento SUN e linguagem paralela Parallel C (3LC) e o software para apresentação gráfica IDL (Interactive Data Language).

A paralelização dos algoritmos de medida de dimensão fractal envolve aplicações como processamento de imagens microscópicas, imagens da coroa solar para aplicação da teoria do caos, etc.

Atualmente, os fractais e a teoria do caos, representam uma nova (estabelecida a menos de vinte anos) fonte de pesquisa, que diz respeito a praticamente todas as áreas do conhecimento humano, pelo simples fato que a geometria fractal descreve as formas e os fenômenos naturais de forma mais natural que a geometria euclidiana.

Na geometria tradicional (euclidiana), define-se que um ponto tem dimensão zero, uma linha tem dimensão igual a um, planos têm dimensão igual a dois e por meio de raciocínio matemático puro, pode-se chegar a dimensão n, mas a natureza não se resume a retas, quadrados ou cubos.

Supondo uma linha que se espalhe por um plano, de forma a nunca se cruzar, tem-se que no limite, ela preencherá o plano em que está contida. Neste caso, pode-se imaginar por pensamento intuitivo que ela possui uma dimensão "fracionária" entre 1 e 2, podendo-se definir o quanto a mudança de dimensão ocorreu de forma proporcional.

A idéia de dimensão fracionária foi concebida pelo matemático Felix Hausdorff, com base no fato de que a dimensão tradicional não descreve bem as formas dos elementos naturais e Benoit B. Mandelbrot aproveitou o conceito e introduziu o conceito de dimensão fractal.

2. Metodologias

2.1. Processamento Concorrente

2.1.1. Visão Geral

Multiprocessamento é um método utilizado para a realização de trabalhos concorrentemente a fim de se atingir um objetivo.

Pipelining: é um método que possibilita o processamento concorrente a nível de instrução dividindo a computação em um número de passos.

“Processamento concorrente é um tipo de processamento de informações que enfatiza a manipulação coincidente de elementos de dados pertencentes a um ou mais processos dividindo o processo inicial em vários outros que são executados paralelamente” [Quinn(1987)].
Existem várias formas de se classificar as arquiteturas paralelas, como o método de Händler: Este associa a cada arquitetura computacional uma expressão matemática que expressa o parelelismo.
Mas a mais utilizada é a taxonomia de Flynn que se baseia nos conceitos de instruções e dados. Estes dois conceitos levaram a quatro classes básicas:

a) **SISD:** (Single Instruction - Single Data ou Instrução Única e Dados Únicos) São os computadores seriais com processamento linear, podendo ser ou não em pipeline, mas possui apenas um processador

b) **SIMD:** (Single Instruction - Multiple Data ou Instrução Única e Múltiplos Dados). Computadores de processamento vetorial, onde se executa apenas uma instrução com vários fetch’s, sendo uma CPU que coordena as instruções e outras que manipulam os dados.

c) **MISD:** (Multiple Instruction - Single Data ou Múltiplas Instruções e Único Dado). Não é aplicável na prática.

d) **MIMD:** (Multiple Instruction - Multiple Data ou Múltiplas Instruções e Múltiplos Dados). Constitui a maioria dos sistemas de multiprocessamento e os mais consagrados. Os processadores trabalham isoladamente com um grau de interação razoável e um pequeno grau de dependência. Os algoritmos projetados para estes sistemas, visam atingir elevada performance e eficiente interação entre os processadores. Como são o alvo deste trabalho, será dedicada uma sessão para este modelo.

2.1.2. Organização dos Processadores

Existem formalmente seis importantes organizações dos processadores:

a) MESH (malha)
b) Pyramid (Pirâmide)
c) Shuffle exchange (trocas aleatórias)
d) Butterfly (borboleta)
e) Hypercube (conexão em cubo)
f) Cube Connected Cycles (Ciclos conectados como Cubos)

Aqui será discutido somente o MESH, pois este é a organização de interesse neste trabalho.

MESH: Os nós são arranjados como num quadrado. A comunicação é permitida somente entre os processadores vizinhos. Algumas variantes deste modelo permitem a conexão de processadores internos com os cantos da malha.

2.1.3. A arquitetura MIMD

As máquinas MIMD consistem num número de processadores totalmente programáveis, cada um capaz de executar seu próprio programa. Multiprocessadores são caracterizados por uma memória compartilhável e em contraste, cada processador possui sua própria memória. Todas as

comunicações e sincronizações entre os processadores são feitas via mensagens que trafegam por um barramento global. Distinções devem ser feitas examinando o padrão de interconexão entre os processadores.

a) Multiprocessadores Fortemente Acoplados

O padrão de comunicação entre os processadores mais simples assume que todos acessam uma memória compartilhável controlada por um mecanismo de trocas. Há uma variedade de formas de implementar este mecanismo, incluindo um barramento global, um sistema de barramentos cruzados (crossbar switches) e uma rede de trocas de pacotes (packet-switched networks).

b) Multiprocessadores Fracamente Acoplados

Como os fortemente acoplados, estes são caracterizados por um espaço de memória compartilhável, mas, este espaço é formado por uma combinação entre as memórias locais das CPU's, portanto o tempo de acesso a uma posição particular de memória, depende se esta é local ao processador ou não. Este acesso pode ser feito com a existência de um Map Bus (barramento mapeador).

2.1.4. Implementação de algoritmos em máquinas MIMD.

Deve-se considerar que perde-se mais tempo direcionando os dados para os processadores do que no processamento em si. As vezes, a complexidade deste processo é tão elevada que não há ganho de desempenho. Assim como, a performance de um algoritmo pode ser radicalmente diferente em uma arquitetura diferente. Algumas vezes é devido à sobrecarga de comunicação, problemas de sincronização, etc... o que leva à uma grande perda de desempenho em máquinas MIMD.

Quatro questões fundamentais devem ser levadas em consideração para o desenvolvimento de algoritmos em máquinas MIMD:

a) Dado um problema com certo grau de paralelismo, como dividir o trabalho entre um número de processadores de forma que eles concorram para a solução de um problema.

b) Quais são os mecanismos que permitem os processos trabalharem juntos e como estes mecanismos podem ser expressos em uma linguagem de programação?

c) O que é deadlock e como pode ser prevenido?

d) Como os processos são distribuídos nos processadores?

Portanto, máquinas MIMD são de propósito mais geral, desde que elas permitam a execução assíncrona de múltiplas instruções, conferindo maior flexibilidade ao programador destes sistemas.

2.1.5. Categorização de algoritmos em máquinas MIMD

a) Pipelined Algorithms: ou algoritmos em pipeline. É um conjunto de segmentos ordenados na qual a saída de cada segmento é a entrada do seu sucessor. A entrada para a execução funciona como a entrada para o primeiro, assim como a saída do primeiro segmento é a entrada para o segundo, até que a saída do último segmento seja produzida, levando ao final do algoritmo. Em todo processamento pipeline, todos os segmentos devem produzir resultados na mesma taxa, ou seja, o segmento mais lento é o gargalo de execução

b) Systolic Algorithms: ou algoritmos sistólicos. É um tipo especial de algoritmo em pipeline. Três atributos diferem no algoritmo sistólico:

1) O fluxo de dados é ritmado e regular
2) O fluxo de dados pode seguir mais de uma direção
3) A computação executada em cada segmento é praticamente idêntica.

Como no pipeline, o sistólico requer uma sincronização implícita entre o segmento que produz dados e o segmento que consome dados.

c) Partitioned Algorithms: ou algoritmos particionados. Neste caso, há a divisão da computação. Um problema é dividido em subproblemas, que são resolvidos individualmente pelos processadores. As solução dos subproblemas são então combinadas para formar a solução do problema. Algoritmos particionados podem ser divididos em duas categorias: Prescheduled algorithms e self-scheduled algorithms.

1) Prescheduled algorithms: ou algoritmos pré-determinados. O processo é dividido em sub-processos e designados para cada processador em tempo de compilação. Quando o tempo necessário para a execução de cada sub-processo é muito variável, se faz uso de uma das duas técnicas comuns. A primeira é ter certeza que existem muito mais sub-tasks (sub-tarefas) do que processos, assim, executa-se sub-tasks de forma aleatória e cada processo terá feito quase a mesma quantidade de trabalho. A segunda técnica é utilizar a segunda categoria de algoritmos.

No presente trabalho, foi utilizado este tipo de algoritmo para máquinas MIMD.

2) Self-Scheduled Algorithms: ou algoritmos auto-organizáveis. O processo é dividido apenas em tempo de execução.

Também são chamados de processor farms.

d) Relaxation Algorithms: ou algoritmos relaxados. É um algoritmo que trabalha sem sincronização. Todos os processos devem estar trabalhando em direção à mesma meta ou deve haver alguma especialização de propósito, mas essencialmente, o processador não necessita esperar por dados a serem fornecidos por outro processador, portanto algoritmos relaxados são caracterizados pela habilidade dos processadores de trabalharem com dados disponíveis mais recentemente. Faz com que a determinação de sai

performance seja difícil. Também podem ser chamados de Algoritmos Assíncronos.

2.2 A Arquitetura ArqMDSP

O presente trabalho faz a análise do uso de uma máquina paralela de alto desempenho de múltiplos DSPs, usando processador TMS320C40, que pode ser classificada como arquitetura MIMD dedicada denominada ArqMDSP. Esse processador tem 6 portas de comunicação de alto desempenho para comunicação e 6 portas canais de DMA, facilitando a construção de arquiteturas paralelas. Além disso, os principais fatores dos modernos DSPs são: hardware para um conjunto de instruções dedicadas, modos especiais de endereçamento, RAM ao invés de cache, e algumas instruções complexas especiais.

A implementação da máquina paralela é facilitada pelo uso de módulos TIM-40, que incorpora num cartão de circuito impresso, o processador, a memória e outros dispositivos. Estes módulos são incorporados em cartões portadores (motherboards ou carrier-cards) responsáveis pelas funções básicas de alimentação e reset, bem como fiações e conexões. A interface entre o hospedeiro e os módulos TIM-40 é implementada pelo cartão HESB40, que contém os circuitos de controle de acesso e circuito de emulação JTAG. Esse cartão é inserido no barramento interno ao SUN/SPARC denominado de SBUS, e a conexão com os módulos TIM-40 é feita através de um flat-cable (cabo plano) de 26 linhas [Saito(1986)].

3. Medidas Fractais

3.1. Introdução

A geometria euclidiana descreve um ponto como tendo dimensão zero. Uma reta é dita ter por dimensão, um. Um plano dois, um cubo três e assim por diante. Esta é uma boa técnica para a descrição de objetos manufaturados com superfícies perfeitas e formas regulares. Mas objetos naturais como montanhas e árvores tem características irregulares e fragmentadas e a geometria tradicional não se torna um modelo realístico nestes casos, mas os métodos da geometria fractal o fazem, onde procedimentos como equações são utilizados.

A geometria fractal é utilizada em vários campos do conhecimento humano para descrever formas naturais e irregulares, onde podemos citar áreas como a análise de imagens de microorganismos ou estudos de imagens da coroa solar, dentre outras aplicações.

Os fractais possuem diferenças fundamentais em relação às formas da geometria Euclidiana. Primeiro, enquanto as figuras tradicionais possuem pelo menos um tamanho ou uma escala característicos, como o raio de um círculo, os fractais não possuem tal tamanho característico. Os fractais são auto-similares (toda parte isolada assemelha-se ao conjunto) e independentes de escala (sob qualquer escala vemos figuras semelhantes). Em segundo lugar, as figuras Euclidianas são normalmente descritas por uma equação algébrica simples — por exemplo, a fórmula $r^2 = x^2 + y^2$ define um círculo de raio r —, ao passo que os fractais são definidos através de algoritmos, que normalmente são recursivos. Finalmente, a geometria tradicional é um meio conciso e eficaz para

descrever os objetos feitos pelo homem, mas é ineficiente na descrição de elementos naturais, ao passo que os fractais descrevem com grande exatidão os objetos encontrados na natureza.

Como um exemplo, suponha-se uma linha que se espalhe pela superfície de um plano, nunca cruzando a si mesma. No limite, ela preenche totalmente o plano. Pela geometria tradicional esta linha ainda é uma figura de uma dimensão, porém percebe-se intuitivamente que ela é *quase* bidimensional.

Tomando como verdadeira esta idéia, pode-se agora definir quando a mudança de dimensão um para dois ocorreu. O matemático Felix Hausdorff concebeu a idéia de dimensão fracionária, baseando-se na idéia de que a dimensão Euclidiana não descreve bem as formas dos elementos naturais. O matemático Benoit B. Mandelbrot aproveitou tal conceito e introduziu a idéia de dimensão fractal [Mandelbrot (1982)], definindo um fractal como um conjunto em que a dimensão de Hausdorff-Besicovitch excede a dimensão topológica.

De acordo com tais definições, a dimensão deve ser descrita como um número racional, e não simplesmente como um inteiro. Dessa forma, pode-se imaginar linhas que possuam dimensões de, por exemplo, 1,58 e 1,95 — neste caso, o maior valor denota a linha que mais preenche o espaço do plano em que elas estão contidas.

Mais tarde Mandelbrot modificou a sua definição inicial em favor de uma outra: um fractal é um elemento formado por partes similares ao todo de alguma maneira, que é ainda uma definição vaga, mas é a melhor atualmente disponível.

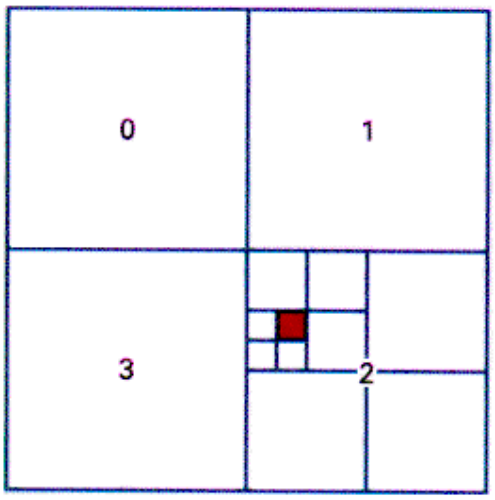
3.2. Medidas de dimensão fractal - método Box Counting

Em um primeiro momento, a imagem é dividida segundo uma grade composta de N_1^2 quadrados e determinando-se o número de quadrados que contém pelo menos um pixel da imagem. Posteriormente, são utilizadas grades mais finas compostas por $N_1^2 < N_2^2 < \cdots < N_m^2$ e calculam-se os respectivos números de quadrados $S(N_1) \cdots S(N_m)$ necessários para cobrir a imagem. Já que $S(N) \cong N^{-D}$, obtém-se a dimensão fractal D como o coeficiente angular da reta obtida, traçando-se $\log(S(N))$ contra $\log(1/N)$, ou simplesmente através da fórmula $D = \log(S(N)) / \log(1/N)$.

Há alguns cuidados que devem ser tomados na aplicação do Box Counting [Coelho & Costa (1995)]. Este método foi criado para ser aplicado estritamente em fractais, porém muitas vezes os objetos reais apresentam somente em parte as características de fractais, levando a um desvio da medida. Além disso, deve-se manter um ponto de equilíbrio entre o número de quadrados necessários para cobrir a imagem, sendo que o método é eficiente para uma grade fina, mas não chegando-se à unidade elementar da figura (pixel).

3.3. O algoritmo

Aqui será explicitado o algoritmo para a implementação do método Box-Counting que foi implementado em linguagem C paralelo (3LC).



As partes de 0 a 3 da imagem, serão as processadas individualmente por cada processador da Arquitetura ArqMDSP concorrentemente.

O algoritmo geral segue anexo no Apendice 1.

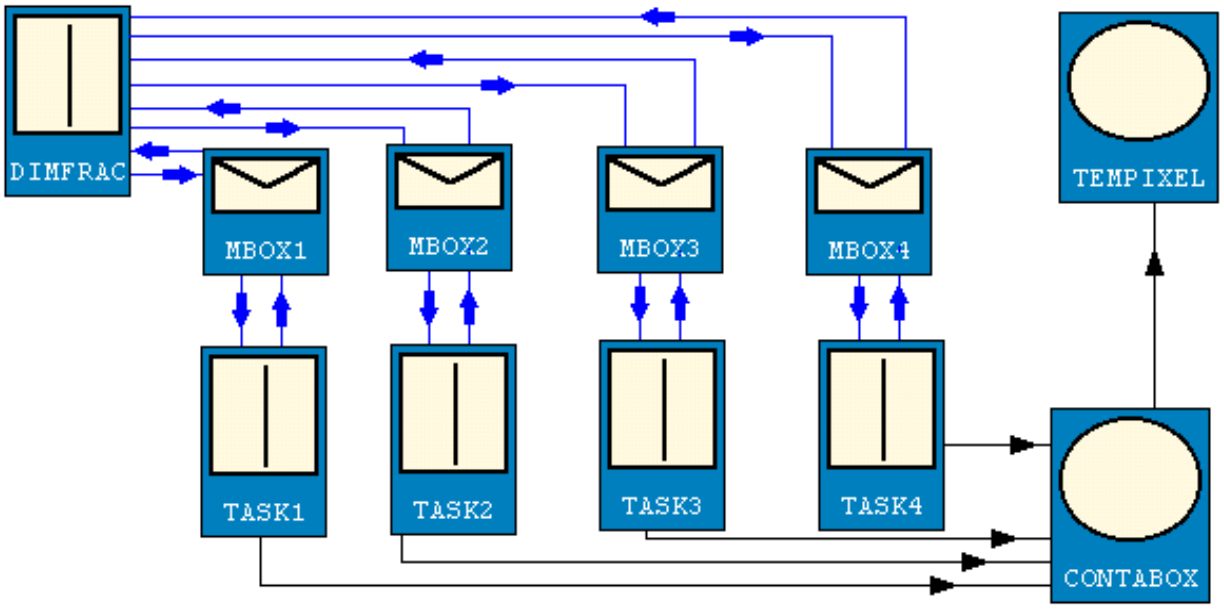
3.4 Teste na linguagem C

O algoritmo acima, foi num primeiro momento implementado no C++ Builder versão 1.0, onde foi implementada a classe SimFrac (simulação de medida fractal). O C++ Builder oferece as facilidades de possuir classes prontas para o acesso a imagens e um ambiente visual de trabalho, não deixando de lado toda a versatilidade da linguagem C++. A implementação funciona de forma seqüencial. O código se encontra no Apêndice B.

3.5 No kernel Virtuoso

Virtuoso é construído como uma camada de software acima da linguagem C, dando-lhe funcionalidades semelhantes a uma linguagem de programação paralela (como o C paralelo da 3L, por exemplo). Embora o Virtuoso implemente os recursos básicos para a construção de sistemas paralelos, ele não oferece um ambiente de alto nível para facilitar a programação. Para isto, foi construído o TEV (Teaching Environment for Virtuoso), onde foi feita uma segunda implementação de forma paralela em uma máquina serial (padrão IBM/PC).

Abaixo segue o modelo utilizado pelo TEV para auxiliar na construção e utilização do Kernel e ilustra a arquitetura da aplicação:



O módulo principal (DIMFRAC) implementa as chamadas aos processos filho. Isto é feito via MAILBOXes (caixas de correio) pertencentes a cada processo. Estas MAILBOX se encarregam de mandar e receber os dados para cada processo.

As tarefas (TASK1 a TASK4) se encarregam de fazer o cálculo concorrentemente. Para tanto, elas utilizam a mesma função CONTABOX (processa a imagem como uma grade) e esta por sua vez utiliza a função TEMPIXEL para um processamento da imagem pixel a pixel.

No Apêndice B, segue a implementação.

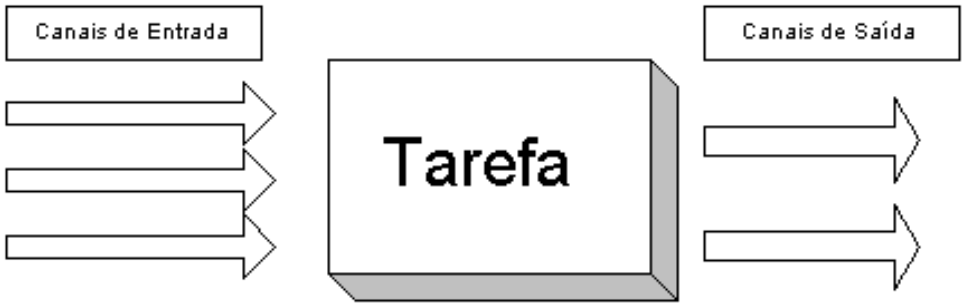
3.6. Na arquitetura ArqMDSP

3.6.1. O Parallel C da 3L

Visão Geral

O modelo seguido pelo 3LC é baseado na idéia de comunicação entre processos seqüenciais.

Desta forma, uma computação é um conjunto de processos ativos concorrentemente que podem se comunicar com outros processos apenas via canais. O modelo Parallel C para processamento paralelo segue o seguinte modelo:



Uma aplicação é um conjunto de uma ou mais tarefas concorrentemente em execução. Cada tarefa é um programa em C com seu próprio código main, sua própria região de memória para dados e pilha de execução.

Cada tarefa é um arquivo de tarefa simples (single task image file) gerado (neste caso pelo linker lkn30). Esta coleção de tarefas é combinada em uma aplicação simples por um utilitário chamado configurador (configurer). Possui um vetor de portas de entrada e um vetor de portas de saída, que são utilizados para conectar as tarefas. Uma tarefa é como uma caixa preta comunicando-se com o meio via portas.

Tarefas podem ser tratadas como blocos atômicos para sistemas paralelos. Elas podem ser juntadas como componentes eletrônicos conectando suas portas via canais.

Os vetores das portas de I/O são declarados como argumento na sua função main. Cada porta é do tipo “ponteiro para canal” (CHAN *). As bibliotecas do Parallel C provêm um número de funções para mandar e receber mensagens sobre os canais.

Portas são atribuídas aos canais pelo configurador, que provê formas de criar um canal de uma dada porta de saída de uma tarefa para a porta de entrada de outra tarefa.

Duas tarefas comunicantes não precisam estar no mesmo processador. Cada processador pode suportar qualquer número de tarefas, limitados apenas pela memória disponível.

Se duas tarefas não estão no mesmo processador, o software de comunicação providencia que as mensagens sejam transmitidas para o processador correto utilizando links inter-processadores.

O Parallel C permite a qualquer tarefa se comunicar com outra, não importando onde as duas estiverem localizadas na malha.

Canais Físicos ou Virtuais

Um canal conecta um processo a outro e pode carregar mensagens somente em uma direção. Se forem necessárias duas vias de comunicação entre dois processos, serão necessários dois canais. Cada processo pode ter qualquer numero de canais de I/O, mas novos canais não podem ser criados durante a execução.

Como padrão, o Parallel C provê de canais virtuais que carregam mensagens automaticamente entre processos distintos via nós intermediários.

Desde que múltiplos canais virtuais possam compartilhar o mesmo link físico, qualquer número de canais virtuais podem conectar tarefas em diferentes processadores.

Mas canais virtuais são mais lentos que canais físicos (links). O Parallel C provê de canais físicos que são mapeados diretamente por links físicos. A vantagem é a velocidade e a desvantagem é que o número de links físicos e o caminho que estes podem percorrer são limitados pelo hardware.

Para a implementação tanto de canais virtuais quanto de canais físicos, não são requeridas mudanças no programa escrito. Basta mudar a configuração do hardware. Portanto, novos canais não podem ser criados durante a execução.

A Estrutura de uma Aplicação

Tipicamente, a rede de processadores é controlada por outro processador: o *Host*. O *Hos*t comunica-se diretamente com apenas um processador da malha chamado *Root*.

Para possibilitar total autonomia para cada processador, em cada um deles é carregado o *Microkernel*, que tem as seguintes funções:

1) escalona todos os processos no processador.

2) controla links inter-processadores, portas de comunicação, timers, ...

3) provê primitivas para a implementação de semáforos.

4) gerenciamento interrupções.

5) scheduling: Permite que várias tarefas possam ser executadas num mesmo processador, provendo uma política de disponibilização de recursos.

O Servidor

O servidor é um programa que roda no host e que possui funções importantes:

1) Carregar a aplicação na malha de processadores.

2) Fazer I/O e outras operações no host e na rede de processadores.

Pela segunda função, o servidor roda no host o tempo todo.

Pode-se adicionar funções extras ao servidor ou então o usuário pode criar o seu próprio servidor.

Tarefas do Sistema

GFS: Global File Services: Arquivo de serviços globais. Ele suporta E/S padrão de qualquer nó na rede C40 pela requisição anterior para o servidor.

VCR: Virtual Channel Router: Roteador de canal virtual da tarefa. Ele fornece suporte para canais virtuais.

UPR: Universal Packet Router: Roteador universal de tarefas. Ele é usado pelo GFS e pelo VCR. Como o nome sugere, UPR roteia pacotes para toda a rede de processadores

Se qualquer nó requisitar I/O ou canais virtuais, o configurador carregará automaticamente o sistema de tarefas requerido para suportar aqueles serviços. Se o GFS ou o VCR é carregado, o UPR será carregado também.

Configurando a aplicação

Uma vez que a aplicação tenha sido designada e escrita com uma coleção de tarefas se comunicando, como é carregado dentro de um rede física de processadores?

Primeiro, cada tarefa individualmente é construída pela compilação dos arquivos fontes com o compilador C e usando o *linker* para combinar os arquivos objetos resultantes com os módulos necessários da biblioteca de tempo de execução do C paralelo.

Agora o arquivo de imagem da tarefa deveria ser combinado para formar um arquivo executável simples. O programa no qual é feito isto é chamado de configurador. Ele é dirigido pelo arquivo de configuração fornecida pelo usuário no qual especifica :

1) a configuração do hardware na qual a aplicação está rodando;
2) os nomes dos arquivos contidos nas tarefas que compõem a aplicação;
3) o lugar particular das tarefas dentro de um particular processador na rede física;
4) as conexões entre as várias portas das tarefas.

A saída do configurador é o arquivo de aplicação no qual pode ser inicializado dentro do *hardware* específico de rede e executar usando o mesmo servidor de programa *(tis)* usado por simples programas *stand-alone*.

A mudança dos processadores onde as tarefas executam, não implica em recompilação e "linkedição" das tarefas. Isto significa, por exemplo, que é possível desenvolver a aplicação em um único processador e depois de depurada executar nos processadores definitivos sem reconfiguração ou qualquer outra mudança. Canais físicos podem ser substituídos por canais virtuais transparentemente de maneira similar.

Links

Os *links* que são usados para a comunicação entre tarefas que estão em diferentes processadores, são usualmente as portas do C40. Em alguns casos, porém, os serviços de *hardware* são usados, assim como *link* tipo *transputer* ou memória compartilhada. Esta parte do *job* do microkernel faz com que certamente todos estes *links* se conduzam para o mesmo caminho, e que eles possam ser usados no mesmo caminho pelas tarefas.

Em algumas tarefas, os *links* são usados para interface com periféricos externos. Para estes casos, o C paralelo fornece funções para transmitir dados diretamente para o *link.* Funções DMA específicas do C40 são também fornecidas.

3.6.2. O IDL

O IDL é um ambiente computacional completo para a analise interativa e visualização de dados. Integra uma poderosa linguagem vetorial com numerosas técnicas de análise matemática e visualização gráfica. Possibilita a criação de aplicações pois possui uma linguagem própria.

Vantagens do IDL:

- Plotagem rápida em duas ou três dimensões, visualização de volumes, visualização de imagens (suporta uma enorme gama de formatos como BMP, GIF, DAT, JPEG e XWD, além de formatos científicos: CDF, HDF e NetCDF), rotinas embutidas para o processamento e análise de imagens.

- É uma interface multiplataforma, rodando sob Windows 95/NT ®, UNIX ® como Solaris ®, Linux, HP-UX, VMS ® e sistemas Macintosh ®.

- Possui recursos que permitem que rotinas escritas em Fortran ou C possam ser dinamicamente ligadas ao IDL, fornecendo outras funções

especializadas. Alternativamente, programas escritos em C ou Fortran podem chamar rotinas do IDL para utilizar seus recursos gráficos. Este trabalho utiliza este recurso do IDL.

3.6.3. Modelagem geral do sistema

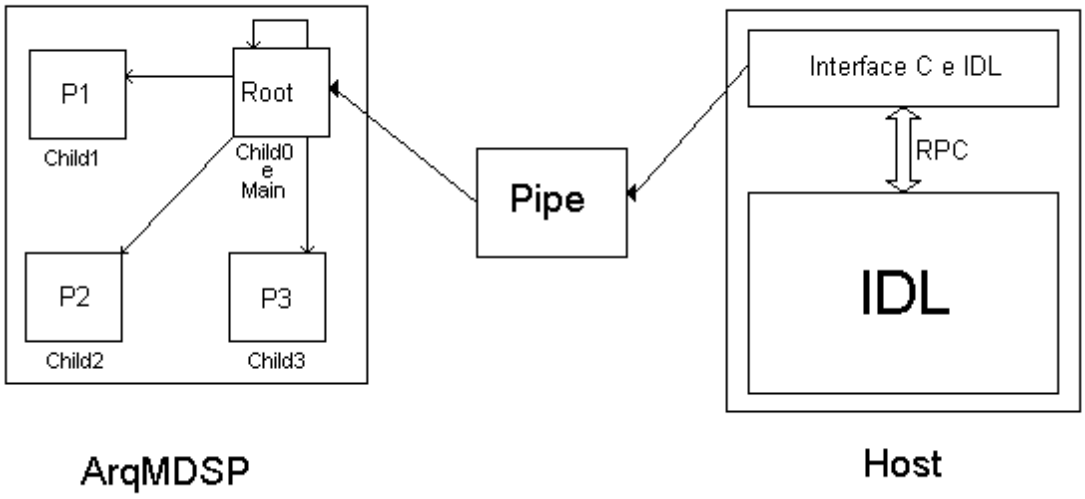
O sistema utiliza as vantagens do IDL para gerenciar e processar imagens de vários formatos diferentes e o poder de processamento de vários processadores trabalhando concorrentemente. Para isto, foram necessárias a implementação de mecanismos para a comunicação entre o Parallel C e o IDL, chamadas de técnicas de comunicação inter-processos.

Para tanto, foram utilizadas duas técnica de comunicação inter-processos:

1) *Remote Procedure Call* (Chamada Remota de Procedimento) : Permite que dois processos em execução se comuniquem e troquem dados, podendo estarem localizados inclusive em workstations diferentes. Esta técnica é oferecida na plataforma UNIX ®, não sendo oferecida para o Microsoft Windows ® ou MS-DOS ®. Através da utilização de bibliotecas, permite que a utilização deste recurso seja mais fácil, deixando para o Sistema Operacional, a tarefa de gerenciar os protocolos de comunicação necessários. Neste trabalho, foram utilizados recursos próprios do IDL (como manuais, bibliotecas e contatos com a empresa desenvolvedora do software: a RSI) para esta implementação.

2) Pipes: É um tipo de arquivo UNIX. São arquivos FIFO ("First-In / First-Out") com tamanhos definidos pelo programador que apresenta uma importante característica: Implementa a sincronização inter-processos. O acesso a pipes é feita da mesma forma que o acesso a arquivos normais usando a biblioteca padrão de E/S da linguagem C (stdio.h), mas o pipe não indica final de arquivo. Isto é feito através de um checksum, para saber se o número de valores recebidos é o mesmo que o número de valores esperados. Se não houverem dados no pipe, o processo consumidor é colocado para "dormir" (sleep) até que ele seja "acordado" (signal) pelo produtor que indica que há dados disponíveis no pipe. Foi utilizado apenas um pipe para a passagem de dados da imagem para a arquitetura paralela.

Abaixo está a modelagem do sistema:



3.6.3. Host - Interface entre C e IDL

Programa escrito em linguagem ANSI C (linguagem C padronizada), residente no Host, que tem como tarefa:

- Leitura da imagem
- Conversão dos dados da imagem para a forma vetorial
- Transferência da imagem para o programa em C padrão.
- Transferência seqüencial da imagem para o pipe.

Para isto, foram utilizadas bibliotecas e funções providas pelo IDL, RPC (Remote Procedure Call) para a comunicação inter-processos e também um pipe para o envio dos dados da imagem.

3.6.4. Arquitetura Paralela ArqMDSP - Root

Programa escrito em Parallel C (linguagem C padrão, com algumas modificações sutis), residente na arquitetura paralela, que tem como tarefa:

- Leitura dos dados da imagem diretamente do pipe
- Conversão dos dados lidos de string para matriz de pontos.
- Envio das "sub-imagens" para os processos-filhos através de canais virtuais.
- Recebimento dos valores processados pelos processos-filhos
- Cálculo da Dimensão Fractal

Foi utilizado um pipe para o recebimento dos dados da imagem.

3.6.5. Arquitetura Paralelo ArqMDSP - Child0 ... Child3

Programas escritos em Parallel C, residentes cada um em um processador da malha, que tem como tarefa:

- Recebimento dos dados das "sub-imagens" através de um canal virtual.
- Contagem de box que contém ao menos um pixel preto
- Envio do resultado da contagem por um canal virtual.

3.6.6. Observações da Implementação

- As imagens suportadas pelo método devem ser aquelas suportadas pelo IDL

- Foram utilizados canais virtuais, devido a presença de apenas um link físico em cada processador, sendo que o programa principal (main) precisava de no mínimo 4 links.

- O fato de que o processador root abriga duas tarefas (main e child0), não afeta a performance geral, visto que são executadas em momentos exclusivos e estão localizadas em posições totalmente distintas na memória.

A implementação na linguagem 3LC ainda não está terminada. Está sendo feita de forma a utilizar um dos processadores como *root* e os outros três como escravos. No processador *root* se encontrará o módulo principal, assim como uma parte o módulo que calculará um quarto da imagem, enquanto nos outros três estarão implementados apenas os módulos que irão calcular os outros três quartos da imagem.

Os processadores se interligarão através de um canal físico (physical channel) dos 3 processadores subordinados até o *root* para passagem do resultado da análise da imagem. Será utilizado um canal virtual (virtual channel) para a passagem do resultado do módulo que está residente no *root* para o programa principal.

Assim, o programa principal configura o hardware e o software, atribui um quarto de imagem para cada processador, recebe o resultado e retorna com a dimensão fractal. O programa subordinado, recebe a parte da imagem que lhe cabe processamento, processa e manda via canal (físico ou virtual) para o root.

4. Resultados obtidos

   A implementação seqüencial está completada, uma vez que esta é de extrema importância para a abordagem concorrente. Esta implementação seqüencial deverá ser implementada sem grandes modificações em 3LC, pois foi concebida para a linguagem paralela.

   Desta forma, os resultados desta implementação serão posteriormente discutidos com uma avaliação do aumento de desempenho que esta proporcionou para o método.

5. Dificuldades encontradas

   Uma das principais dificuldades enfrentadas foi a dificuldade de compreensão inicial da teoria fractal envolvida por se tratar de uma área do conhecimento humano ainda muito recente e de difícil disponibilidade de material bibliográfico.

   Além disso, também houve dificuldade iniciais no desenvolvimento de um algoritmo que pudesse ser implementada de forma seqüencial e também de forma paralela, assim como o estudo da teoria a respeito do desenvolvimento de aplicações para máquinas MIMD dedicadas [Quinn(1987)].

6. Cronograma final (ver o inicial)

   O plano de trabalho é o de um estudo bibliográfico sobre a implementação de algoritmos paralelos em máquinas MIMD [Quinn(1987)], seguido de implementação do modelamento browniano para a arquitetura ArqMDSP, usando a linguagem 3LC e o software gráfico IDL.

Meses	1 e 2	3 e 4	5 a 7	8 a 11	12
Atividade	eap	eab	imb	iig-iad	edr

onde:

eap - estudo de algoritmos paralelos para máquinas MIMD
eab - estudo do algoritmo do browniano para máquinas MIMD
imb - implementação do modelamento browniano utilizando a linguagem 3LC
iig - implementação da interface gráfica utilizando o IDL

iad - implementação na arquitetura ArqMDSP
edr - elaboração do relatório final

Com o andamento do cronograma, estamos no estágio *imb* e já em fase de término da implementação em linguagem 3LC.

7. Bibliografia

Hearn, Donald and Baker, M. Pauline - Computer Graphics (C version) - 2nd Ed. - Prentice Hall pg 362-366.

Apêndice A

```
funcao BoxCounting(Gridsize: inteiro) : DimenFrc
inicio
Total <-- 0
Grade <-- gridsize / 4 {divide grid para os 4 processad}
{ inicializacoes que dependem do software grafico }
XImag <-- Largura da Imagem
YImag <-- Altura da Imagem
cobegin
  Res0<--CountBox(0,0,YImag/2,XImag/2)
  Res1<--CountBox(0,XImag/2,YImag/2,XImag)
  Res2<--CountBox(YImag/2,0,YImag,XImag/2)
  Res3<--CountBox(YImag/2,XImag/2,YImag,XImag)
coend
Total <-- Res0 + Res1 + Res2 + Res3
DimenFrac <-- log(Total)/log(1/Gridsize)
retorna DimenFrac
fim-funcao


funcao CountBox(Top,Left,Bot,Right: inteiros): Totalbox
inicio
        Totalbox <-- 0
        XSum <-- (Rigth-Left)/Grade
        YSum <-- (Bottom-Top)/Grade
        X <-- Left
        Y <-- Top
        enquanto Y < Top faca
        inicio
                enquanto X < Rigth faca
                inicio
                        se TemPixel(X, Y, X+XSum, Y+YSum) entao
                        inicio
                                Conta <-- Conta + 1
                        fim-se
                        X <-- X + XSum
                fim-enquanto
                Y <-- Y + YSum
        fim-enquanto
        retorna Totalbox
```

fim-funcao

Apendice B

```
//---------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "SimFrac1.h"
#include <math.h>
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TSimFrac *SimFrac;
//---------------------------------------------------------------------
__fastcall TSimFrac::TSimFrac(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TSimFrac::Abrir1Click(TObject *Sender)
{
         if(OpenDialog1->Execute())
   {
        SimFrac->Caption = "Simulação Dimensão Fractal - " + ExtractFileName(OpenDialog1-
>FileName);
      Image1->Picture->LoadFromFile(OpenDialog1->FileName);
   }
}
//---------------------------------------------------------------------
void __fastcall TSimFrac::Sair1Click(TObject *Sender)
{
         Close();
}
//---------------------------------------------------------------------
int __fastcall TSimFrac::ContaBox(int Top, int Left, int Bott, int Right)
{
         int XSum = XImag/Grade, YSum = YImag/Grade, X = Left, Y = Top, TotalBox = 0;
   while (Y < Bott)
         {
         X = Left;
                  while (X < Right)
         {
                  if (ImagemTemPixel(X,Y,X+XSum,Y+YSum)) TotalBox++;
         X = X + XSum;
      }
      Y = Y + YSum;
   }

   return(TotalBox);
}
//---------------------------------------------------------------------
bool __fastcall TSimFrac::ImagemTemPixel(int Top, int Left, int Bott, int Right)
{
         int X, Y;
         bool Tem = False;
   for (X = Left; X <= Right; X++)
   {
         for (Y = Top; Y <= Bott; Y++)
      {
         if (Image1->Picture->Bitmap->Canvas->Pixels[X][Y] != clWhite)
        {
```

```
            Tem = True;
           }
        }
    }

    return Tem;
}
//---------------------------------------------------------------------
void __fastcall TSimFrac::CalcDimFrac(TObject *Sender)
{
            Gridsize = StrToInt(Edit1->Text);
    Grade = Gridsize / 4;
    XImag = Image1->Picture->Width; YImag = Image1->Picture->Height;
    int Res0, Res1, Res2, Res3;
    double DimenFrc, Total;
    // Cobegin
            Res0 = ContaBox(1,1,YImag/2,XImag/2);
            Res1 = ContaBox(1,XImag/2, YImag/2,XImag);
            Res2 = ContaBox(YImag/2,1,YImag,XImag/2);
            Res3 = ContaBox(YImag/2,XImag/2,YImag,XImag);
    //CoEnd
    Total = Res0 + Res1 + Res2 + Res3;
            DimenFrc = log(Total)/log(1/Gridsize);
    Label1->Caption = "Dimensão Fractal: " + FloatToStrF(DimenFrc,ffFixed,15,4);
}
//---------------------------------------------------------------------
```