Unioeste – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Informática

Curso de Bacharelado em Informática

Programação Paralela Aplicada a Banco de Dados

Luciano Trevisan Alberti

CASCAVEL

2001

Luciano Trevisan Alberti

Programação Paralela Aplicada a Banco de Dados

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Marcio Seiji Oyamada

CASCAVEL

2001

DEDICATÓRIA

A Deus, por guiar nossos passos, aos meus pais que sempre servirão de base em nossas conquistas, meus irmãos (segundos pais) e à Ângela, meu bem querer.

AGRADECIMENTOS

Aos mestres que nos orientaram nestes “primeiros degraus”, aos quais devemos todo o respeito e, aos amigos, que nos deram a mão nas horas “onde o degrau era um pouco maior”.

LISTA DE FIGURAS

LISTA DE TABELAS E QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CPU	*Central Process Unit*
MPI	*Message Passing Interface*
PVM	*Parallel Virtual Machine*
Mbyte	*Mega Byte*
Gbyte	*Giga Byte*
Mbits	*Mega Bits* por Segundo
SGBD	Sistema Gerenciador de Banco de Dados
SGBDP	Sistema Gerenciador de Banco de Dados Paralelo
PC	*Personal Computer*
SISD	*Single Instruction Single Data*
SIMD	*Single Instruction Multiple Data*
MISD	*Multiple Instruction Single Data*
MIMD	*Multiple Instruction Multiple Data*
I/O	*Input/Output*
E/S	Entrada/Saída
LSDP	Laboratório de Sistemas Paralelos e Distribuídos

SUMÁRIO

RESUMO

O propósito deste trabalho é o estudo de técnicas de programação paralela aplicada a banco de dados. Tais aplicações abrangem algoritmos de ordenação, junção, seleção, eliminação de duplicata, projeção e agregação. Estes algoritmos têm como principal objetivo o aumento no desempenho de um SGBD (Sistema Gerenciador de Banco de Dados) através da extração de paralelismo nas operações. Comparações entre o modelo seqüencial e o modelo paralelo são apresentadas no estudo de caso para algoritmos de busca e contagem.

Palavras-chave: MPI, Banco de Dados, Paralelismo

1. INTRODUÇÃO

No início, a implementação dos Gerenciadores de Banco de Dados era baseada na utilização em ambientes monoprocessados (apenas um computador com um processador). Com a evolução, o conceito de Banco de Dados Distribuídos passou a ser um ponto de interesse para muitos pesquisadores. Özsu, M. (1999, p. xvii) cita "a tecnologia dos sistemas de banco de dados distribuídos é uma das mais recentes áreas em desenvolvimento dentro dos sistemas de banco de dados". Torna-se crescente a necessidade de sistemas que consigam processar grandes volumes de informações de maneira mais rápida e eficiente. A manipulação destes dados de forma paralela sobre vários computadores é uma alternativa, descartando a necessidade de um único computador com grande poder de processamento. Para que tal manipulação possa ser realizada é necessário o suporte tanto em *software* (através das linguagens de programação paralela) quanto em *hardware* (através de ambientes multiprocessados).

O objetivo deste trabalho é demonstrar as técnicas de paralelismo em banco de dados assim como a explanação da programação paralela, sua importância e funcionamento.

O capítulo 2 trata sobre a programação paralela e os tipos de arquiteturas existentes que suportam tal aplicação. Algumas linguagens de programação paralela, como o PVM e o MPI, são apresentadas no capítulo 3. O capítulo 4 refere-se aos Banco de Dados Paralelos, as técnicas de distribuição de dados e algoritmos de aplicação paralela para os mesmos. Algoritmos paralelos para contagem de tuplas são estudados e comparados com os modelos seqüenciais no capítulo 5.

2. PARALELISMO

Segundo Gavilan, J. (2000, p. 1): “o uso de vários processadores, como os utilizados em máquinas seqüenciais, executando um algoritmo simultaneamente, possibilita uma solução mais veloz de um problema do que somente com um processador resolvendo o mesmo algoritmo”. Mesmo se este único processador possuir maior capacidade de processamento do que os outros em paralelo, o resultado obtido pode ser melhor. A principal comparação que pode-se fazer entre um sistema paralelo e um sistema seqüencial é a de uma tarefa ser executada em um computador de grande porte (*mainframe*) e, a mesma tarefa, em vários computadores de pequeno porte (*personal computer*). Este tipo de comparação torna-se interessante pelo fato da primeira categoria citada (*mainframe*) possuir custo unitário relativamente maior que os PCs, o que leva ao crédito dos sistemas paralelos em utilizar vários computadores pessoais a executar uma tarefa que somente um grande, e caro, computador poderia fazer.

Enquanto que na programação seqüencial o projetista pensa de forma linear (uma instrução após a outra), na programação paralela outra forma de raciocínio é exigida. É necessário ter em mente que a tarefa será dividida em vários módulos e que cada módulo é executado por um processo que fará parte do sistema paralelo. A idéia básica do paralelismo é dividir tarefas em processos para que os mesmos executem-nas paralelamente.

Um exemplo para a divisão de tarefas é o de calcular uma integral utilizando o paradigma da programação paralela, citado em Pacheco, P. (1997, p. 53). O autor transcreve a Regra do Trapezoidal como segue:

a) uma integral definida de a para b de uma função $f(x)$ não-negativa, pode ser vista como uma área limitada pelo eixo x, as linhas verticais $x=a$ e $x=b$, e o gráfico da função $f(x)$, como na figura 1;
b) uma maneira de estimar esta área, ou integral, é particionar a região em formas geométricas regulares e então adicionar o resultado de cada forma à grande área (área total). Na regra do trapezoidal, a forma geométrica regular é em forma de trapézio; cada trapézio tem sua base no eixo x, o lado vertical e suas arestas são unidas em dois pontos no gráfico de $f(x)$, como na figura 2.

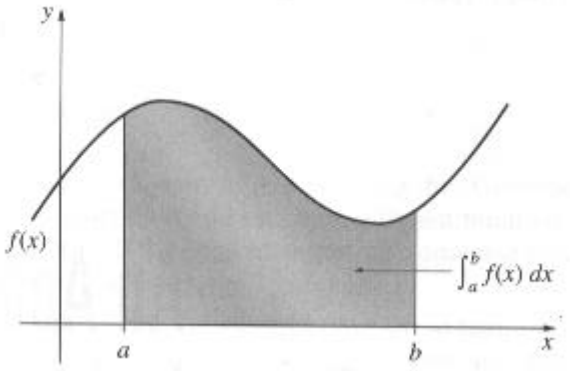

Figura 1 - Integral a ser calculada (PACHECO, P., 1997, p.54)

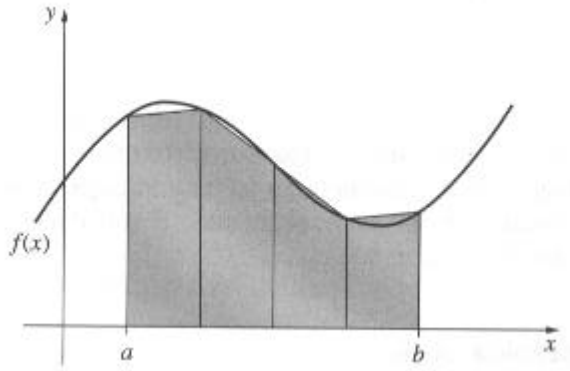

Figura 2 - Divisão da integral a ser calculada (PACHECO, P., 1997, p.54)

De maneira resumida, o algoritmo proposto atribui a cada processo criado uma parte da integral para ser calculada. Cada processo ao terminar de efetuar seus cálculos retorna o resultado para um processo "mestre" que estará coletando as respostas e somando os resultados parciais para se obter o resultado final. Neste modelo, conforme aumenta o número de processos, maior será a precisão do cálculo obtido, pois os polígonos conseguirão chegar mais próximos da linha da função integral.

2.1. Comunicação por Troca de Mensagens

Como dito anteriormente, vários processos podem ser criados para que a tarefa seja efetuada paralelamente. A comunicação entre os mesmos pode ser efetuada de duas formas:

a) através de memória compartilhada, onde a comunicação entre os processos é feita através de um *buffer* comum entre eles;
b) através de um mecanismo de comunicação entre processos, onde não existe um recurso compartilhado e a comunicação entre processos é garantida através da troca de mensagens (*message passing*).

Silberchatz, A. (1997, p. 108) descreve as facilidades da comunicação entre processos através de um sistema de mensagens: "A função de um sistema de mensagens é garantir a comunicação para os processos sem a necessidade de um recurso de compartilhamento de memória". Esta garantia é obtida através de duas operações: *send(mensagem)* – enviar mensagem – e *receive(mensagem)* – receber mensagem. As mensagens que trafegam entre os processos podem ser de tamanho fixo ou variável. A segunda opção torna a implementação física mais complexa porém, simplifica a tarefa da programação.

2.2. Mecanismos de comunicação

De acordo com Merkle, C. (1996, p. 35), a comunicação entre os processos pode ser provida de duas formas:

a) Comunicação Síncrona: onde os canais de comunicação não possuem capacidade de armazenamento e as comunicações requerem a participação simultânea dos processos envolvidos (sincronização entre os processos);
b) Comunicação Assíncrona: o canal serve de transporte bidirecional de mensagens o qual possui capacidade de armazenamento e não requer a participação simultânea entre os processos envolvidos.

A utilização tanto da comunicação síncrona quanto da assíncrona depende diretamente do ambiente de programação e da aplicação para qual é destinada.

2.3. Tipos de Paralelismo

O paralelismo pode ser classificado segundo o objeto a ser paralelizado e, de acordo com CENAPAD-NE (2000, p. 1), é dividido em Paralelismo Funcional ou Paralelismo de Dados.

2.3.1. Paralelismo funcional

O paralelismo funcional é aplicado em programas dinâmicos e modulares onde cada tarefa será um programa diferente. Os dados, não necessariamente diferentes, sofrem a execução de instruções distintas, conforme a figura 3.

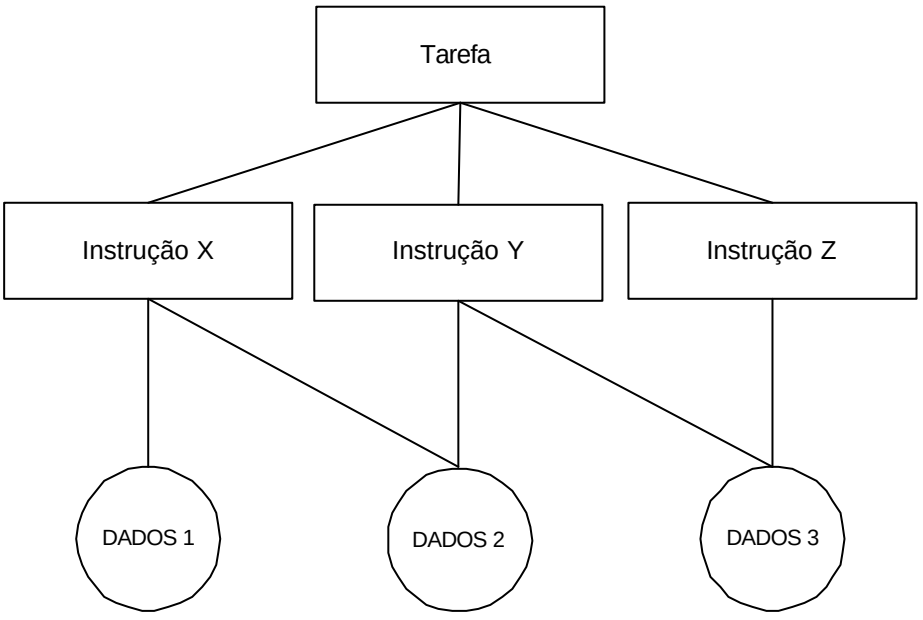

Figura 3 - Exemplo de Paralelismo Funcional

Barreto, R. (1993., p. 33) cita o paralelismo funcional como particionamento de algoritmo (ou particionamento de função). Segundo o mesmo autor, “esta estratégia é característica de aplicações onde a computação pode ser dividida em estágios, de forma que as saídas (*output*) de um estágio alimente (*input*) uma nova etapa de processamento, perfazendo uma linha-de-montagem.”

2.3.2. Paralelismo de dados

Dados diferentes sofrem a execução das mesmas instruções dos processadores, isto é, existe apenas um tipo de instrução e é aplicada, paralelamente, em dados distintos, conforme a figura 4.

Segundo Preuss, E. (1998, p. 26), “o paradigma de programação paralela baseada no paralelismo de dados representa uma forma de exploração de operações simultâneas sobre grandes conjuntos de dados, ao invés de especificar várias tarefas paralelas de controle.”

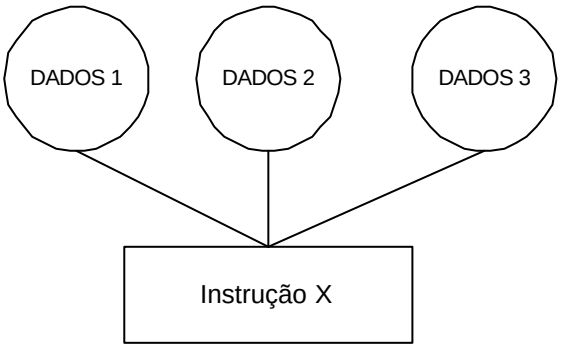

Figura 4 - Exemplo de Paralelismo de Dados

Um exemplo para o paralelismo de dados é a aplicação em programas que utilizam grandes matrizes, como no algoritmo de Divisão e Conquista.

2.4. Granularidade

Para Gavilan, J. (2000, p. 2), a granularidade (ou tamanho do grão) de um sistema paralelo condiz ao tamanho médio das unidades de trabalho ou à quantidade de instruções de um segmento de programa submetida aos processadores, podendo esta ser fina, média ou grossa. Se, por exemplo, está previsto alocar grandes processos a um pequeno número de processadores, diz-se que a arquitetura tem granularidade grossa (baixa granularidade, grão grosso) e, se ocorrer o contrário – pequenas quantidades de unidades de trabalho alocadas para um grande número de processadores – a arquitetura tem uma granularidade fina (alta granularidade, grão fino).

O critério de medida de granularidade é algo subjetivo, segundo Barreto, R. (1993., p. 32) "costuma-se chamar paralelismo a nível de processo de granularidade grossa, a nível (*sic*) de blocos de granularidade média e a nível (*sic*) de blocos pequenos, ou instruções, de granularidade fina."

Exemplos práticos para arquiteturas que utilizam granularidade grossa são os multiprocessadores convencionais, para granularidade fina são as máquinas de fluxo de dados (GAVILAN, J., 2000, p.2). Para granularidade média (ou grossa) os ambientes de memória global são bons exemplos (BARRETO, R., 1993, p. 33).

"Nos ambientes de memória distribuída, o grão do paralelismo tende a ser de média para grande (*sic*). O paralelismo de grão fino torna-se inviável pois o tempo dispensado com comunicação e criação remota de processos geralmente é mais significativo do que o ganho teórico pela execução em paralelo" (BARRETO, R., 1993, p. 33).

Pode-se dizer então, que o objetivo da programação paralela (melhor desempenho que a programação seqüencial), não é alcançado se sua utilização em memória distribuída for no nível de blocos pequenos, devido a grande quantidade de comunicação existente entre os processos.

2.5. Arquiteturas de Máquinas Paralelas

Sistemas computacionais podem ser classificados de acordo com a taxonomia de Flynn, que divide-os em 4 (quatro) categorias de acordo com o número de fluxo de dados e instruções a serem processadas simultaneamente, o qual é listado abaixo (PACHECO, P., 1997, p. 12):

a) SISD (*Single Instruction Single Data*) – Única Instrução e Único Dado: onde cada instrução manipula um dado por vez;
b) SIMD (*Single Instruction Multiple Data*) – Única Instrução e Múltiplos Dados: onde cada instrução pode manipular mais de um dado por vez;
c) MISD (*Multiple Instruction Single Data*) – Múltiplas instruções e Único Dado: várias instruções operando sobre o mesmo dado (alguns autores consideram inexistente a utilização prática desta categoria);
d) MIMD (*Multiple Instruction Multiple Data*) – Múltiplas Instruções e Múltiplos Dados: onde várias instruções operam sobre vários dados simultaneamente. Aqui são caracterizados os sistemas distribuídos.

Dentre os citados, os sistemas MIMD receberão maior atenção. Devido a sua diversidade de arquiteturas, que podem ser subdivididos em Arquiteturas MIMD de Memória Compartilhada e Arquiteturas MIMD de Memória Distribuída.

O sistema MIMD de Memória Compartilhada também subdivide-se em classificações, tais como:

a) Arquiteturas Baseadas em Barramento: um barramento interliga os processadores para o compartilhamento da memória. É o tipo de interconexão mais simples porém menos escalável e tem o problema da sobrecarga do barramento. Esta arquitetura é mostrada na figura 5. Para contornar o problema da sobrecarga do barramento, utiliza-se uma arquitetura de coerência de *cache* (ou consistência de *cache*). Esta possui um protocolo para manter a consistência entre as memórias privadas com o objetivo de igualar cada memória pertencente no grupo. Dentre os protocolos existentes pode-se destacar o Protocolo *Snoopy*, que é projetado para máquinas com arquitetura baseada em barramento e

sua idéia básica é monitorar o tráfego no barramento (quando um processador atualiza uma variável compartilhada o monitor também atualiza a correspondência na memória principal e *caches*). Esta arquitetura evita a sobrecarga no barramento apesar da complexidade da atualização de cache.

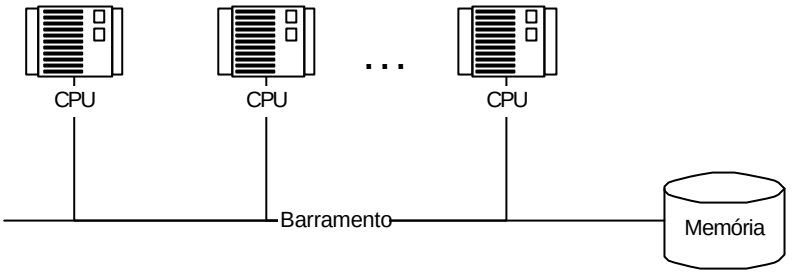

Figura 5 - Arquitetura Baseada em Barramento

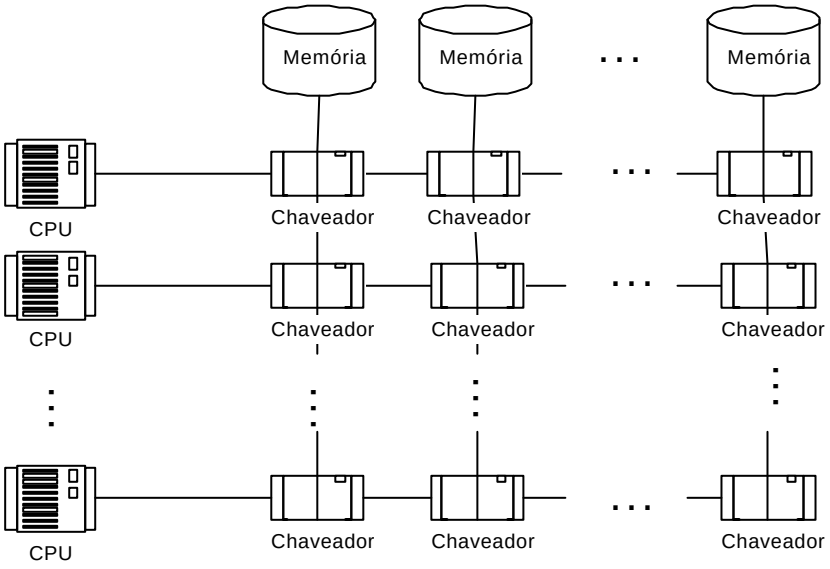

Figura 6 - Arquitetura Baseada em Chaveamento

b) Arquiteturas Baseadas em Chaveamento: a definição pode ser acompanhada com o auxílio da figura 6. Os processadores, ou os módulos de memória, podem se conectar através de um chaveamento

feito por sinais horizontais e verticais e possibilita a combinação entre acessos, não correndo o risco de sobrecarga. Seu maior problema é o tamanho da matriz de chaveamento que aumenta em grande escala de acordo com o número de módulos acrescentados.

Exemplos de sistemas de banco de dados paralelos com memória compartilhada são: *XPRS*, o *DBS3* e o *Volcano*, onde todos compartilham memória principal (ÖZSU, M., 1999, p. 425, p. 426).

Como nas Arquiteturas MIMD de Memória Compartilhada, as de Memória Distribuída possuem suas subclasses:

a) Rede de Interconexão Dinâmica: existe um chaveamento entre cada máquina e pode ser efetuado a qualquer momento. Qualquer tipo de interconexão entre os participantes da rede é permitido, conforme demonstrado na figura 7;

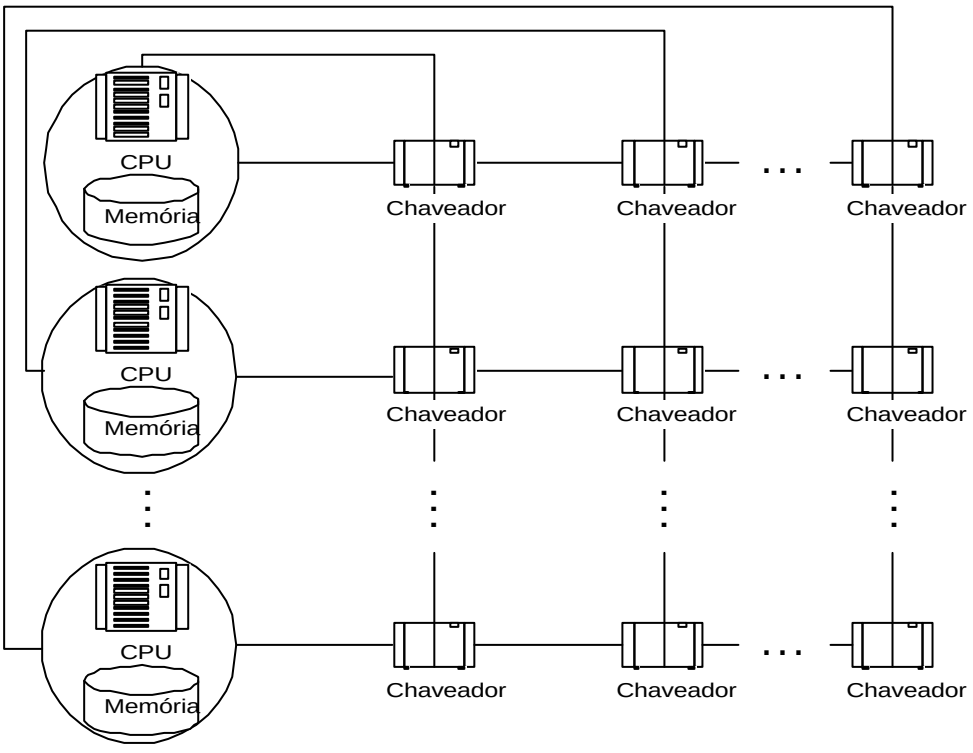

Figura 7 - Rede de Interconexão Dinâmica

b) Rede de Interconexão Estática: conforme visto na figura 8, a ligação entre as máquinas não é feita por chaveamento e permanece a mesma caso o projeto de rede não seja mudado;

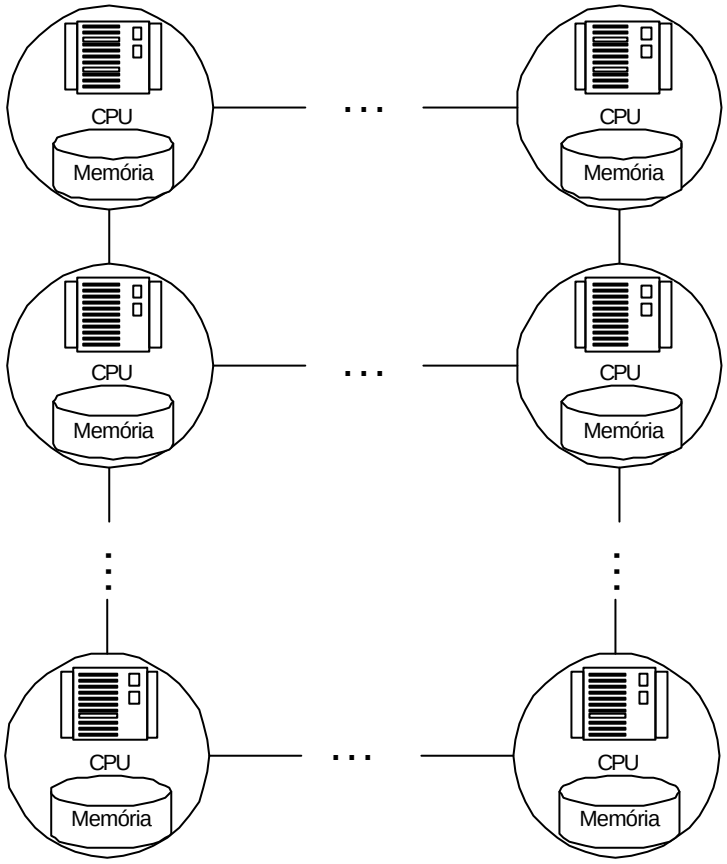

Figura 8 - Rede de Interconexão Estática

c) Rede Baseada em Barramento: se comparada às anteriores é o mais simples e de menor custo, onde pode-se anexar, como exemplo, um grupo de computadores em uma Intranet. Possibilita grande escalabilidade porém com problema da sobrecarga do barramento de comunicação comum, como ilustra a figura 9.

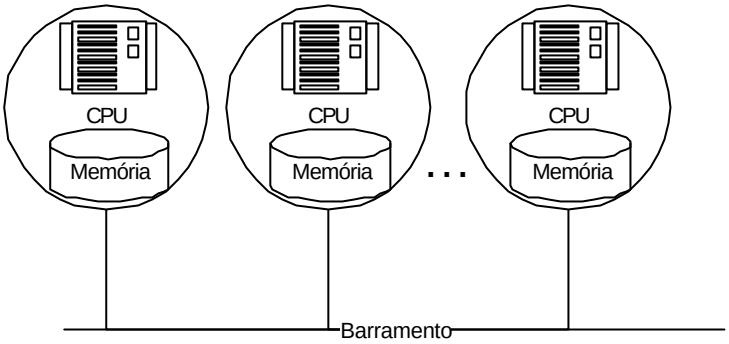

Figura 9 - Rede Baseada em Barramento

Como exemplos de sistemas de banco de dados paralelos que utilizam arquitetura de memória distribuída pode-se citar os sistemas *DBC* da *Teradata* e *NonStopSQL* da *Tandem* (ÖZSU, M., 1999, p. 427).

Tabela 1 - Comparação entre Arquiteturas MIMD

Tipo de Arquitetura	Vantagens	Desvantagens
Memória Compartilhada	• Compartilhamento de dados entre processo é mais rápida.	• Custo elevado de equipamento; • Existe limite de número de processadores; • São necessárias técnicas de sincronização para leitura e gravação.
Memória Distribuída	• Acesso à memória local sem interferência; • Sem limite, teórico, para o número de processadores.	• Dificuldade para mapear informações; • O usuário é responsável pelo sincronismo e recebimento de dados; • Elevado *overhead* devido à comunicação.

(Fonte: CENAPAD-NE 2000, p. 5)

Tanto as arquiteturas de memória compartilhada como as de memória distribuída possuem suas vantagens e suas desvantagens conforme demonstrado na tabela 1, construída a partir dos dados de CENAPAD-NE (2000, p. 5).

2.6. Problemas da Programação Paralela

Existem alguns pontos a considerar sobre a viabilidade da utilização dos sistemas paralelos na computação, que segundo CENAPAD-NE (2000, p. 3) são:

a) sincronização entre os processos: a coordenação dos resultados ou a troca de dados entre os processos é necessária, assim, deve-se atentar ao fato de que a sincronização dos processos e da comunicação entre os mesmos seja menor do que o processamento em si, para não prejudicar o desempenho dos processos em execução;
b) conversão de algoritmos seqüenciais em paralelos: a revisão do programa seqüencial para, futuramente, paralelizá-lo demanda grande parte do tempo. Analisar o programa origem, avaliar como e o que será particionado, quais os pontos de sincronização e qual a forma de comunicação empregada são algumas das principais partes de tal revisão;
c) nem tudo é paralelizável: deve-se fazer um levantamento da quantidade de comunicação em relação ao processamento do algoritmo a ser paralelizado, para se obter resultados sobre a real viabilidade do sistema;
d) portabilidade: existe uma perda de portabilidade quanto aos programas adaptados a arquiteturas de comunicação baseada em memória compartilhada, pois os mesmos não utilizam os mecanismos de troca de mensagens necessários nas arquiteturas de comunicação baseada em memória distribuída;
e) depuração do código: como os processos são distribuídos e executados em vários processadores simultaneamente, não existe uma forma eficiente de acompanhar o exato decorrer do programa, alteração em variáveis por exemplo, durante o processamento das várias tarefas em paralelo;

f) arquitetura de rede: em determinadas redes de computadores, onde existe uma deficiência de velocidade devido a precárias condições dos equipamentos (cabeamento, *hosts* ou dispositivos de chaveamento – comutadores) a utilização do paralelismo com memória distribuída torna-se totalmente inviável.

Conforme Ricarte, I. (1996, p. 71) "a tendência atual na área de computação distribuída aponta para um novo tipo de 'supercomputador'. São *clusters* de estações de trabalho ligadas por redes de alta velocidade, com taxas de transmissão já muito próximas daquelas alcançadas nas redes de interconexão de sistemas supercomputadores".

Então, apesar dos obstáculos, as áreas dos sistemas paralelos e distribuídos estão na constante busca de aperfeiçoamento.

2.7. Métricas de Desempenho de um Sistema Paralelo

As propriedades que um sistema paralelo deve possuir estão relacionadas a escalabilidade e velocidade de execução, sendo ambas lineares. De acordo com Meyer, L. (1997, p. 19), tais propriedades são:

a) Aceleração Linear (*Linear Speedup*) – é medida através da equação definida pela razão entre o tempo de execução com 1 processador (processamento seqüencial) e o tempo de execução com n processadores (processamento paralelo). Isto é, se o número de processadores for duas vezes maior, a tarefa deverá ser executada na metade do tempo;
b) Crescimento Linear (*Linear Scaleup*) – é a capacidade de aumentar o *speedup* a medida que é aumentado o número de processadores, ou melhor, se o número de processadores é duplicado, o sistema deverá ser capaz de executar uma tarefa duas vezes maior no mesmo espaço de tempo.

Porém, segundo Meyer, L. (1997, p. 20): "(...) ganhos superlineares também podem ser alcançados, em particular nas aplicações de banco de dados, quando se consegue

tirar proveito de um bom balanceamento de carga e da disponibilização dos dados em memória e *cache*."

Como pode ser visto, o paralelismo pode ser alcançado pelo particionamento de E/S entre diversos discos e pelo particionamento de trabalho de CPU entre os múltiplos processadores. Um problema em potencial com estas técnicas é o desbalanceamento de carga, que está associado com o particionamento dos dados entre os discos, já descrito anteriormente. Outro problema refere-se a aceleração e crescimento linear. Em relação a estes dois últimos citados alguns fatores devem ser considerados, tais como:

a) Custo de Inicialização: tempo gasto para inicialização da operação paralela em diversos processadores;
b) Custo de Interferência: tempo de resposta no acesso a recursos compartilhados, como memória, disco e rede comunicação;
c) Custo de Comportamento: caso haja um desbalanceamento de carga ou um computador mais lento, o resultado fica comprometido.

Sintetizando, para se encontrar o verdadeiro ganho de uma determinada implementação paralela, deve-se utilizar o quociente da razão do tempo para executar uma tarefa seqüencialmente pelo tempo levado para executar a mesma tarefa em paralelo. Com isto tem-se o *speedup*, isto é, quantas vezes o processamento paralelo foi superior/inferior ao seqüencial. O desempenho é superior caso o resultado obtido seja maior que 1, inferior se menor que 1 e igual se o *speedup* for igual a 1.

3. LINGUAGENS DE PROGRAMAÇÃO PARALELA

Merkle, C. (1996, p.31) descreve que um modelo de programação paralela se baseia na utilização de extensões de linguagens do tipo *C*, *Fortran*, *Pascal*, entre outras, denominadas clássicas. Como por exemplo, tanto o MPI quanto o PVM são bibliotecas que podem ser utilizadas em linguagens como *C*, *C++* e *Fortran*, com o intuito de fornecer suporte a programação paralela.

3.1. MPI (*Message Passing Interface*)

"O padrão para *Message Passing*, denominado MPI, foi projetado em um fórum aberto constituído de pesquisadores, acadêmicos, programadores, usuários e vendedores, representando 40 organizações ao todo" (CENAPAD-NE, 2000, p. 12). Segundo Pacheco, P. (1997, p. vii), o padrão MPI define sintática e semanticamente as rotinas de biblioteca mais utilizadas, tanto pelo domínio público como comercial, para os desenvolvedores de sistemas que utilizam o paradigma da troca de mensagens em código nas linguagens *C* ou *Fortran 77*.

O *MPI_Send* e *MPI_Recv* são as primitivas básicas para comunicação entre processos, sendo utilizadas respectivamente, para enviar e receber mensagens. Outro tipo de comunicação que os processos podem ter entre si é o da comunicação em grupo (*broadcast*), efetuado pela primitiva *MPI_Bcast*. Um conjunto completo das funções MPI pode ser encontrado em Pacheco, P. (1997).

3.2. PVM (*Parallel Virtual Machine*)

De acordo com Rímolo, G. (1997, p.415), "PVM é um *software* que permite a execução de programas paralelos em um ambiente heterogêneo. Esse *software* atua interligando várias máquinas da rede, formando um ambiente preestabelecido pelo usuário. Esse ambiente passa a ser visto de uma forma transparente, como se fosse uma máquina".

O PVM utiliza diretivas de troca de mensagens cooperativas entre processos onde a operação de envio deve estar associada a uma operação de recebimento, como no MPI.

3.3. Comparações Entre os Modelos MPI e PVM

A Tabela 2, descreve as principais diferenças entre os dois ambientes de programação paralela.

Tabela 2 - Diferenças entre MPI e PVM

Características	MPI	PVM
Mecanismos para criação dinâmica de tarefas	Não	Sim
Mecanismos de gerência do ambiente de execução	Não	Sim
Controle de *buffer* para troca de dados	Feito pelo usuário	Pelo Ambiente
Gerenciamento de *buffer* seguro	Sim	Não
Suporte a *threads*	Disponível	Em Desenvolvimento

(Fonte: adaptada de Rímolo, G., 1997, p.416)

Além disso, o MPI, segundo Rímolo, G. (1997, p.416), dispõe de outras vantagens sobre o PVM no que diz respeito à sua implementação, comunicação assíncrona, portabilidade, conjunto de primitivas de programação paralela, entre outras. Ainda, comparando MPI e PVM através de execuções de programas paralelos, Rímolo conclui que "os resultados vão depender muito da forma em que as aplicações sejam implementadas, porém, os custos das primitivas de comunicação MPI são inferiores às equivalentes PVM".

4. BANCO DE DADOS PARALELO

A tecnologia de Banco de Dados Distribuído pode ser naturalmente estendida para implementar um Sistema de Banco de Dados Paralelo, isto é, um Sistema de Banco de Dados em computadores paralelos. (DEWITT, D., 1992, p. 85-98)

De acordo com Özsu, M. (1999, p. 420), "Sistemas de Banco de Dados Paralelo combinam o Gerenciamento de Banco de Dados e o processamento paralelo para aumentar o desempenho e a disponibilidade", características almejadas na manipulação de banco de dados. E Hsiao, D.(1983) **apud** Özsu, M. (1999, p. 421), notifica que "desempenho já era, também, o objetivo das máquinas de banco de dados (DBMs – *Database Machines*) nos anos 70 e 80".

O grande "gargalo" nas máquinas de banco de dados é o problema da entrada/saída (I/O – *Input/Output*) que pode ser resolvido com o aumento da largura de banda que o paralelismo disponibiliza. Özsu, M. (1999, p. 421) exemplifica esta situação da seguinte forma: "se nós armazenarmos um banco de dados de tamanho *D* em um único disco com processamento (*throughput*) *T*, este será limitado por *T*. Em contrapartida, se nós particionarmos o banco de dados em *N* discos, cada um com capacidade de *D/N* e processamento (*throughput*) *T'* (esperando-se a equivalência a *T*), nós obteremos um processamento ideal de *N*T'* que pode ser melhor consumido por múltiplos processadores (idealmente *N*)."

Korth, H. (1999, p. 567) relata algumas tendências as quais atribui ao sucesso dos Banco de Dados Paralelos. Tais tendências estão abaixo citadas:

a) as exigências das organizações em relação ao processamento de transações vêm aumentando com o crescente uso de computadores;
b) sistemas de processadores simples não são capazes de tratar grandes volumes de dados – na faixa dos *terabytes* – necessários no processamento das organizações;
c) como os microprocessadores se tornaram economicamente mais acessíveis, os equipamentos paralelos passaram a ser mais comuns e relativamente econômicos.

Os itens relacionados acima demonstram o interesse na implementação de banco de dados paralelos para suportar as necessidades de manipulação de grandes bases de dados utilizando-se do processamento paralelo.

4.1. Particionamento de Dados

"Não haverá nenhum benefício real no desempenho de um sistema de banco de dados quando uma arquitetura paralela for utilizada mas os dados não estiverem bem distribuídos entre os diversos nós do sistema" (RICARTE, I., 1996, p. 43). De acordo com Korth, H. (1999, p. 568) pode-se mostrar três técnicas de particionamento de dados, considerando-se *n* discos (D_0, D_1, ..., D_{n-1}) entre os quais os dados devem estar particionados:

a) *Round-Robin*: as tuplas são distribuídas pela "ordem de chegada" – a *i*-ésima tupla é enviada ao $D_{i \mod n}$ disco. Cada disco terá, aproximadamente, o mesmo número de tuplas;

b) Particionamento *Hash*: uma função *hash* é utilizada para mapear o domínio de um (ou mais) atributo para um valor inteiro maior, ou igual, a zero e menor que o número de nós (*n*), e cada tupla da relação original é separada pelo atributo de particionamento;

c) Particionamento por Faixa de Valores: valores próximos podem ser alocados a um mesmo nó. Um bom exemplo, para este tipo de particionamento, citado em Ricarte, I. (1996, p. 45) e adaptado para uma lista de telefones em um sistema de cinco discos, é demonstrado na Tabela 3.

Tabela 3 - Exemplo para o Particionamento por Faixa de Valores

Partição (Disco)	Faixa (Pela primeira letra do Sobrenome)
1	A – E
2	F – J
3	K – O
4	P – T
5	U – Z

O grau de eficiência entre as técnicas de particionamento recém descritas, resumido da Tabela 4, pode ser considerado um comparativo entre os métodos para percorrer uma relação inteira, localizar uma tupla associativamente em específico ou localizar todas as tuplas em determinados intervalos de valor de um determinado atributo.

Tabela 4 - Comparação entre os Métodos de Particionamento de Dados.

Tipo de Busca / Técnica	Percorrer a relação Inteira	Consulta Pontual	Consulta por faixa
Round-Robin	Adequada	Inadequada	Inadequada
Hash	Adequada	Adequada	Inadequada
Faixa de Valores	Inadequada	Adequada	Adequada

Nos métodos de *hash* e por faixa de valores deve-se tomar cuidado com o balanceamento de carga, pois pode ocorrer um acúmulo de tuplas em determinadas partições e, em conseqüência, poucas tuplas em outras. Através de uma escolha cautelosa da função *hash* ou do vetor de partição, para os métodos de *hash* e faixa de valores, respectivamente, pode-se evitar perdas em desempenho no paralelismo (KORTH, H., 1999, p. 570).

4.2. Exemplos de Algoritmos Paralelos para Banco de Dados

A paralelização das tarefas pertinentes a um Sistema Gerenciador de Banco de Dados pode ser classificada através das operações de Classificação, Junção, Seleção, Eliminação de duplicatas, Projeção e Agregação, descritas na seqüência.

4.2.1. Classificação (ordenação)

É o processo onde uma tabela é ordenada com base em um ou mais de seus campos. A ordenação total dos elementos contidos no conjunto que deseja-se a ordem pode ser crescente ou decrescente. (VELOSO, P., 1997, p. 178).

Na ordenação paralela, pode-se considerar alguns pontos, tais como:

a) o número de elementos da lista é divisível pelo número de processos que estarão executando sobre a máquina virtual;
b) os elementos da lista são distribuídos, entre os processos, em blocos. Por exemplo, se a lista possuir 100 elementos e 4 processos para executar o algoritmo de ordenação, então o 1º processo será responsável pela faixa de elementos 1-25, o 2º processo de 26-50 e assim sucessivamente.

O algoritmo inicia com cada processo recebendo elementos aleatoriamente. Tendo em vista as considerações anteriores, utiliza-se uma função de redistribuição onde cada processo conterá apenas os elementos que lhe competem – item b, e cada processo irá ordenar sua lista de dados, individualmente. A operação final é a concatenação dos conjuntos ordenados.

4.2.2. Junção

A junção trabalha com dois ou mais conjuntos de elementos realizando uma interseção entre os mesmos, tendo como resultado um conjunto de elementos onde cada elemento está contido nos conjuntos da operação de junção. O algoritmo de junção paralela tenta dividir entre vários processadores os pares a serem testados. Cada processador computa parte da junção localmente e, após isto, é obtido um resultado de cada processador e agrupado em um único conjunto final (KORTH, H.,1999, p. 575).

4.2.3. Seleção

Constitui-se a tarefa de encontrar um determinado elemento sobre um ou mais conjuntos. Para realizar uma operação de seleção em paralelo, é necessário particionar o conjunto em faixas onde cada processador tenha seu domínio de busca para o elemento que se deseja encontrar. Tendo isto paralelizado, cada processador ao realizar sua busca irá retornar o resultado de tal tarefa, finalizando a operação de seleção.

4.2.4. Eliminação de duplicatas e projeção

A eliminação de duplicatas pode ser alcançada através de algoritmos de classificação ou pelo particionamento das tuplas, necessitando apenas uma otimização para a eliminação de duplicatas. No caso da projeção, sem a eliminação de duplicatas, pode ser realizada a medida que as tuplas são lidas em paralelo. Se realizada a eliminação das duplicatas na projeção, as técnicas já citadas podem ser aplicadas. (KORTH, H.,1999, p. 575)

4.2.5. Agregação

Cada processo realiza a operação pelos atributos de agrupamento localmente. Ao fim desta etapa pode-se computar os resultados de cada processador para a obtenção de uma única resposta ao sistema. (KORTH, H.,1999, p. 580)

4.3. Nível de Paralelismo em Consultas

Para se obter um melhor desempenho, no caso de consultas, para sistemas paralelos pode-se obter o paralelismo em duas formas, descritas a seguir.

4.3.1. Paralelismo entre consultas

De acordo com Korth, H. (1999, p. 571), o paralelismo entre consultas é a forma de paralelismo mais fácil de manter em um sistema de banco de dados, principalmente em um sistema paralelo de memória compartilhada. Sua principal aplicação é melhorar o sistema de processamento de transações de modo a aceitar um número maior de transações processadas por segundo. Consiste em executar consultas ou transações distintas em paralelo umas com as outras.

4.3.2. Paralelismo interno a consulta

Uma única consulta é executada paralelamente em diversos processadores e discos. Segundo Korth, H. (1999, p. 572), isto é importante para melhorar o tempo de execução de consultas com grande tempo de duração, pois pode-se paralelizar as operações relacionais sobre diferentes conjuntos de relações existentes. Meyer, L. (1997, p. 27) descreve que este tipo de paralelismo "(...) consiste da execução paralela de diferentes operadores dentro de uma única consulta. Se um operador pode enviar o resultado obtido para o seguinte, ambos podem executar em paralelo, diminuindo o tempo de resposta da consulta".

Este paralelismo pode ser canalizado (*pipeline*) ou independente. Na canalização, por exemplo, as tuplas resultantes em *A* serão utilizadas para a computação em *B* (como em uma linha de produção). No caso da independência, por exemplo, *A* e *B* não possuem uma dependência para se obter um resultado final de uma operação entre eles, como acontece no *pipeline*. Apesar da utilização do *pipeline* em um sistema paralelo, a princípio, ter a mesma finalidade de utilização em um sistema seqüencial, Korth, H.(1999, p.582) diz que pode-se extrair o paralelismo no *pipeline*, da mesma forma que os canais de instrução são usados como uma das fontes de paralelismo em um projeto de *hardware*.

Como exemplo de paralelismo *pipeline,* considere uma junção entre três conjuntos de relações (r1, r2 e r3). O processamento da junção de r1 com r2 será efetuado por A. B computará a junção do resultado de A com r3. O paralelismo será obtido ao passo que as

tuplas já computadas em A estarão sendo enviadas a B para a computação com r3 sem que A tenha terminado completamente a operação de junção entre r1 e r2.

5. UM ESTUDO DE CASO PARA APLICAÇÃO DO PARALELISMO

Como visto no capítulo anterior, o paralelismo pode ser aplicado em diversas tarefas para tentar melhorar o desempenho de acesso a uma base de dados. Este capítulo tem como finalidade complementar a idéia da aplicação do paralelismo em banco de dados e, para representar melhor estes conceitos, demonstrar o resultado da implementação dos algoritmos paralelos para busca e contagem de tuplas e seus modelos seqüenciais. Suas implementações e os resultados obtidos são detalhados e, ao final do capítulo, tem-se um comparativo geral entre eles e o algoritmo seqüencial.

5.1. Definição das Técnicas Para Contagem e Busca de Tuplas

A contagem de elementos é um dos procedimentos realizado pelos operadores de agregação, contidos nas consultas. Dois algoritmos paralelos foram implementados: o primeiro com vários processos pesquisando em um único arquivo (Modelo Replicado) e o segundo com vários processos pesquisando em vários arquivos (Modelo Particionado).

A busca de elementos é o procedimento para seleção e projeção de dados em consultas relacionais. Um algoritmo paralelo foi desenvolvido para realizar este tipo de operação sobre os arquivos, o qual irá utilizar o Modelo Particionado para a busca de elementos.

Para os testes, foram utilizados arquivos textos e cujos conteúdos serão explicados na seqüência. Os tempos são medidos através do comando *time* fornecido pelo Sistema Operacional (o Sistema Operacional utilizado é descrito no item 5.2).

Em um sistema de banco de dados paralelo real, quando um usuário realiza uma consulta, o sistema deve ser capaz de tornar transparente esta consulta para o utilizador do sistema como se o mesmo estivesse executando sua consulta em um sistema convencional monoprocessado, com uma grande diferença: o tempo de resposta reduzido. Um exemplo de consulta realizada por um usuário pode ser a seguinte: "*pesquise quantos dos clientes residem*

na cidade de Curitiba" (em SQL: "*select count(CLI_Codigo) from Cliente where CLI_Cidade = 'Curitiba' "*). Paralelamente o sistema irá computar, ressaltando, de forma transparente ao usuário, a mesma consulta em vários locais, caso os dados estejam particionados ou dividirá uma porção da base de dados para cada processador computar localmente, caso os dados estejam replicados, retornando o resultado obtido ao usuário.

5.2. Ambiente de execução

Foram utilizadas, para a execução do algoritmo paralelo, três estações, totalmente homogêneas, PA-RISC da HP® (*Hewlett-Packard*) série 700 modelo C200 monoprocessadas, Sistema Operacional *HP-UX* 11, com disco rígido *Wide-SCSI* de 9 *Gbytes* e 128 *Mbytes* de memória principal em uma rede *Ethernet* de 10 *Mbits*. Um comutador simples de dados do tipo *hub* de 12 portas e cabos par-trançado categoria 5 foram utilizados como infra-estrutura de interligação. Estes equipamentos constituem a máquina virtual paralela, contida no LSDP do Colegiado de Informática, UNIOESTE – Campus Cascavel.

A distribuição da biblioteca MPICH (2000) do MPI é utilizada sobre o sistema *Unix* da HP (*HP-UX*). Todas as implementações utilizam as diretivas da linguagem de programação *C*.

5.3. Algoritmos para Contagem de Tuplas

As seções seguintes demonstrarão os modelos implementados para a simulação da contagem de tuplas, suas metodologias e seus resultados. Os modelos seqüenciais também serão descritos e servirão como base de comparação com os modelos paralelos.

5.3.1. Modelo simulando replicação de dados

Este modelo consiste em um algoritmo paralelo que irá distribuir seus processos sobre faixas de um mesmo arquivo que poderá estar replicado em outras máquinas e sofrerá a pesquisa individual por cada processo contido na máquina virtual paralela. Como cada processo terá uma faixa do arquivo para realizar a pesquisa, cada réplica terá, no mínimo, um processo contendo um ponteiro em determinada posição do arquivo. Considere o ambiente descrito no item 5.2, a execução deste modelo utilizando 3 (três) processos realizando a contagem em um arquivo de 30 *Mbytes* de tamanho. Como demonstrado na Figura 10, a faixa de atuação para cada processo será: de 0 a 10 *Mbytes*, de 10 *Mbytes* a 20 *Mbytes* e de 20 *Mbytes* a 30 *Mbytes* deste arquivo, respectivamente.

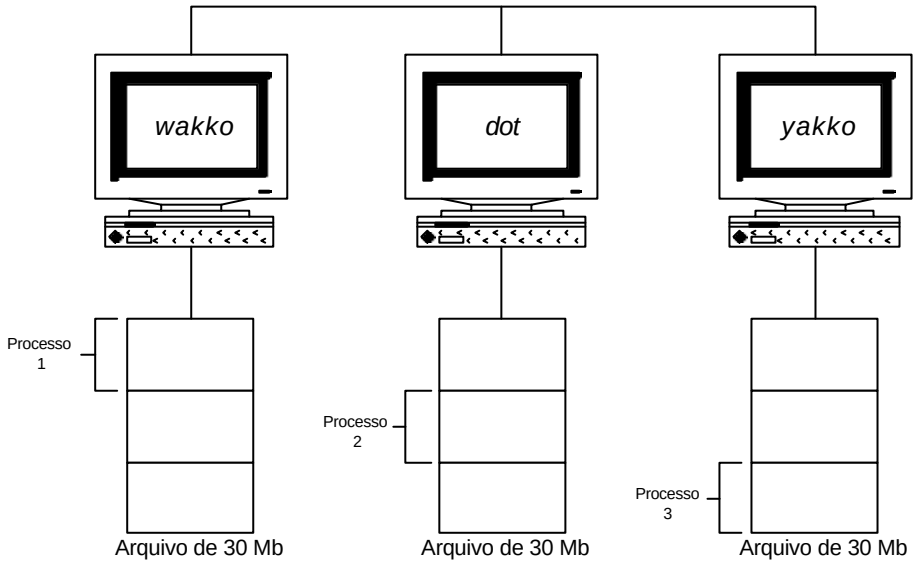

Figura 10 - Exemplo do Modelo Simulando Replicação de Dados

5.3.1.1. Metodologia

Foram utilizados os testes em dois tamanhos de arquivos diferentes. Um arquivo contendo 79,31 *Mbytes* (Teste 1) e outro contendo 922,35 *Mbytes* (Teste 2).

Cada nó da máquina virtual paralela continha uma cópia dos arquivos de Teste 1 e Teste 2 em seus discos locais, caracterizando-se a replicação de dados. Isto foi realizado pois o *overhead* na transferência dos dados no ambiente descrito inviabilizaria o experimento. O arquivo relativo ao Teste 1 contém campos de 5 (cinco) dígitos que, intercalando-se, formam chaves únicas. A intercalação obtida da letra *a* até a letra *z* segue na seguinte forma: *aaaaa*, *aaaab*, *aaaac*, ... , *bbbba*, ... , *zzzzx*, *zzzzy*, *zzzzz*. O arquivo concernente ao Teste 2 contém campos de 6 (seis) dígitos seguindo a mesma idéia do Teste 1, excluindo o fato de alcançar a letra *z*, onde, neste caso a chave única incide-se até a letra *l*.

Cada processo ao iniciar irá calcular sua faixa de busca, obtendo um início e um final de partição no arquivo. O primeiro processo (processo 0) será o coordenador e responsável pela coleta dos resultados. A tarefa de distribuição dos outros processos é efetuada por *Round-Robin*, isto é, dos computadores existentes na máquina virtual paralela, é criada uma fila circular onde os processos serão executados. Por exemplo, suponha que existem 7 processos para executar este procedimento de contagem, a ordem que as máquinas estão declaradas e seus respectivos nomes são, *dot*, *wakko* e *yakko*. O primeiro processo (zero) ficará na máquina que originou a chamada da contagem, o segundo processo irá para *dot*, o terceito para *wakko*, o quarto para *yakko*, o quinto para *dot*, o sexto para *wakko* e o sétimo processo para *yakko*, obtendo-se 3 (três) processos por máquina realizando efetivamente a operação de contagem.

Para ambos os testes, o algoritmo foi executado com 4, 7, 13, 16 e 19 processos, tendo-se, respectivamente, 1, 3, 4, 5 e 6 processos computando a contagem em cada máquina.

5.3.1.2. Resultados obtidos

A obtenção dos resultados foi alcançada através da média de cinco execuções para cada um dos modelos de processos, descrita no último parágrafo do item anterior. A tabela 5 demonstra os resultados do Teste 1 e a tabela 6 do Teste 2. Nota-se um decréscimo do tempo de execução a medida que aumenta-se o número de processos, porém, o aumento exagerado do número de processos poderá prejudicar o bom desempenho e o tempo

de execução poderá vir a crescer diretamente. Isto ocorre devido ao tempo de comunicação e controle entre processos e pela sobrecarga de processos na mesma máquina. A inclusão de outros processadores na máquina virtual paralela poderá melhorar o desempenho dos algoritmos.

Tabela 5 - Contagem utilizando o Modelo Simulando Replicação de Dados (Teste 1)

Nº Processos	Tempo (minutos)
4	1,214533
7	0,843133
13	0,710183
16	0,709167
19	0,752683

Tabela 6 –Contagem utilizando o Modelo Simulando Replicação de Dados (Teste 2)

Nº Processos	Tempo (minutos)
4	38,20428
7	21,84437
13	13,39817
16	11,54975
19	10,67448

5.3.2. Modelo simulando particionamento horizontal de dados

O particionamento horizontal, descrito no capítulo 4, foi simulado através de vários arquivos de tamanhos idênticos e a contagem consiste no fato de cada processo ser responsável por apenas um arquivo.

5.3.2.1. Metodologia

O número de processos executados é exatamente o número de arquivos mais um, isto é, caso se pretenda realizar a contagem em 3 (três) arquivos, ter-se-á 3 (três) processos, sendo cada um destinado a um arquivo. Um quarto processo será o coordenador, que não executará a consulta porém ficará responsável por coletar os resultados da contagem. Arquivos contendo 67,98 *Mbytes* foram utilizados para a execução com 4, 7 e 13 processos. Para facilitar a execução e evitar o *overhead* da transferência dos arquivos pela rede de comunicação estes são replicados para todos os discos dos computadores contidos na máquina virtual paralela. A distribuição de processos por máquina é realizada da mesma forma descrita no segundo parágrafo do item 5.3.1.

5.3.2.2. Resultados obtidos

A obtenção dos resultados também foi alcançada por meio da média de cinco execuções para cada um dos modelos de processos, conforme o número de arquivos utilizados. Estes resultados representam algumas diferenças em relação aos resultados descritos no item 5.3.1.2. Observando a tabela 7 pode-se notar que à medida que se aumenta o número de arquivos, aumenta-se também o tempo. Utilizando-se 3 (três) arquivos para contagem, o resultado em minutos foi 2,98. Considerando o fato que a máquina paralela virtual contém somente 3 (três) processadores, dobrando o número de arquivos (utilizando seis arquivos) esperava-se um resultado de 5,96 (o dobro do tempo em relação a execução com três arquivos), porém o resultado foi de 4,01 minutos. E com doze arquivos (quatro vezes mais), o valor obtido de 6,09 é 95 % mais rápido que o esperado de 11,93 minutos (quatro vezes a utilização de três arquivos). Este fato pode ser explicado devido ao sistema de contagem estar executando mais processos na mesma máquina, portanto, a porção dos recursos do sistema (especificamente o tempo de CPU), é maior para os algoritmos de contagem, devido a política *Round-Robin* de escalonamento entre processos do Sistema Operacional utilizado.

Tabela 7 –Contagem utilizando o Modelo Simulando Particionamento Horizontal de Dados

Nº Arquivos	Tempo (minutos)
3	2,983633
6	4,010533
12	6,090817

5.3.3. Modelos seqüenciais

Para motivar uma comparação entre os modelos apresentados anteriormente, algoritmos seqüenciais similares foram implementados utilizando os mesmos conceitos. Foram realizadas execuções para o modelo com um único arquivo e para o modelo particionado (vários arquivos).

5.3.3.1. Metodologia

Três algoritmos seqüenciais foram implementados, também nesta seção, uma nomenclatura será utilizada para os modelos. *Teste A* identifica o algoritmo seqüencial que irá utilizar os vários arquivos para a contagem (Modelo Particionado), sendo cada arquivo contendo 67,98 *Mbytes*. *Teste B* e *Teste C* nomeia-se os algoritmos seqüenciais que utilizarão o Modelo Replicado (apenas um arquivo), sendo que o Teste B designou sua contagem em um arquivo de 79,31 *Mbytes* de tamanho e o Teste C um arquivo contendo 922,35 *Mbytes*. Os três algoritmos foram executados sobre apenas uma máquina, com a mesma descrição técnica detalhada no item 5.2.

No caso do Teste A, o número de arquivos utilizados foi 3, 6 e 12, para melhor se assemelhar ao modelo paralelo.

5.3.3.2. Resultados obtidos

Novamente, a média de cinco execuções para cada algoritmo foi calculada. Neste modelo (seqüencial) é comum o crescimento exponencial do tempo, isto é, caso aumenta-se o número de arquivos ou o tamanho do arquivo que será efetuada a contagem, aumenta-se, também o valor temporal. A tabela 8 mostra, resumidamente, os resultados dos três modelos seqüenciais.

Tabela 8 –Modelos Seqüenciais para Contagem

Teste A		**Teste B**	**Teste C**
Nº Arquivos	**Tempo (minutos)**	**Tempo (minutos)**	**Tempo (minutos)**
3	2,9629	1,152417	13,52163
6	5,93025		
12	11,8627		

5.3.4. Comparação dos resultados da contagem

Como descrito o item 5.3.3, a comparação entre os modelos paralelos e seqüenciais são de suma importância para a visualização dos resultados. Tem-se três tipos de comparações, a primeira relacionando o Modelo Particionado Paralelo com o Seqüencial; a segunda o Modelo Replicado Paralelo com o Seqüencial, utilizando um arquivo contendo, aproximadamente, 80 *Mbytes* e a terceira o Modelo Replicado Paralelo com o Seqüencial, utilizando um arquivo contendo, aproximadamente, 920 *Mbytes*.

No Modelo Particionado, como demonstrado no gráfico da figura 11, observa-se um ganho crescente se for utilizado o modelo paralelo. Particularmente, no caso da utilização do número de arquivos igual a 12 (doze), tem-se um *speedup* de 1,95 (isto é, a versão paralela foi 95 % mais rápida que a seqüencial – quase 2 vezes mais rápida). No caso de utilizar-se 3 (três) arquivos, vê-se uma semelhança de tempos entre as duas

implementações, isto é decorrente do tempo gasto em comunicação entre processos e latência de rede, pois estas afetaram o desempenho do modelo paralelo.

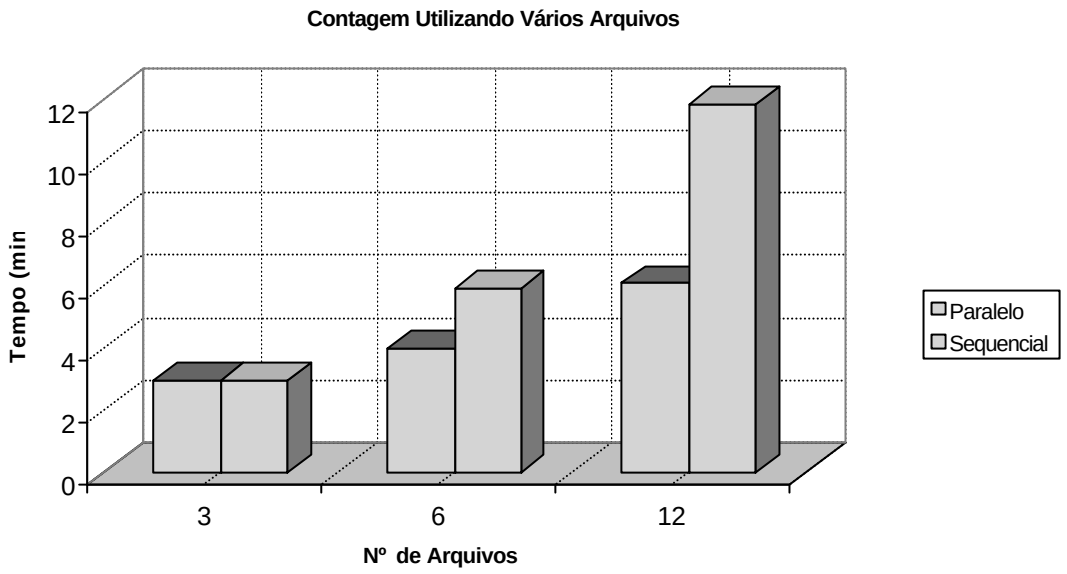

Figura 11 – Gráfico de Comparação dos Resultados (Modelo Particionado)

No primeiro Modelo Replicado (que utiliza um arquivo de quase 80 *Mbytes*), foi comparado o algoritmo paralelo, variando-se o número de processos, com o seqüencial. Como no seqüencial o número de processos é constante, como ilustra a figura 12, as barras que o representam possuem tamanho constante. De início o *speedup* é de 0,95, significando que a implementação paralela utilizando quatro processos para a contagem é 5% mais lenta que a implementação seqüencial. Nestes testes, o *speedup* mais satisfatório é o de 1,62, correspondendo a utilização de treze processos para a contagem. Conforme o número de processos é incluído ao algoritmo paralelo, o tempo de execução aumentará, devido ao excesso de pacotes trafegando em rede e o alto número de colisões. Por se tratar de um comutador simples e o número de máquinas ser apenas três, este incremento ao número de processos pode tornar-se inviável.

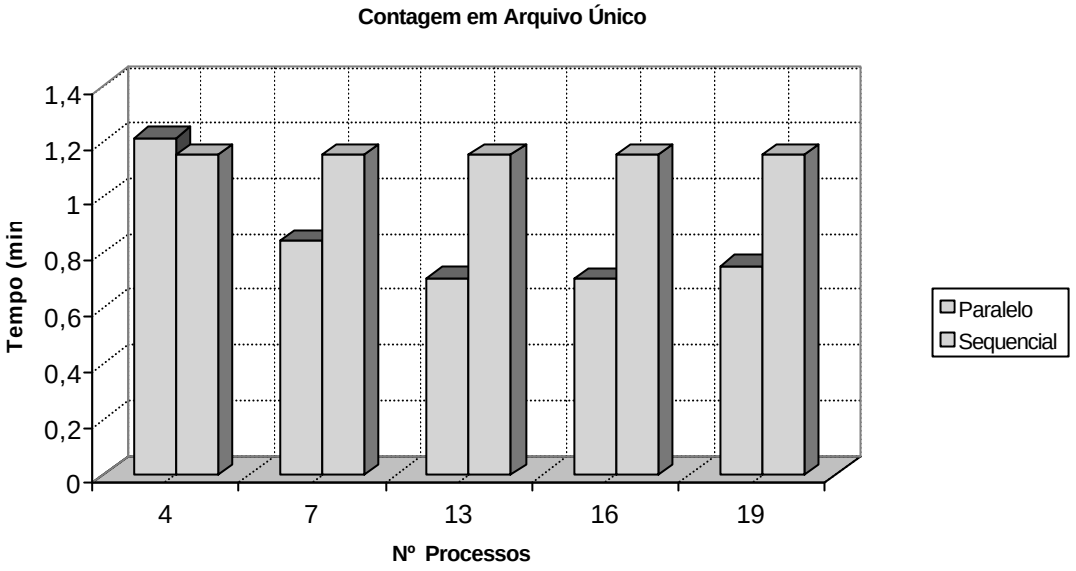

Figura 12 - Gráfico de Comparação dos Resultados (primeiro Modelo Replicado)

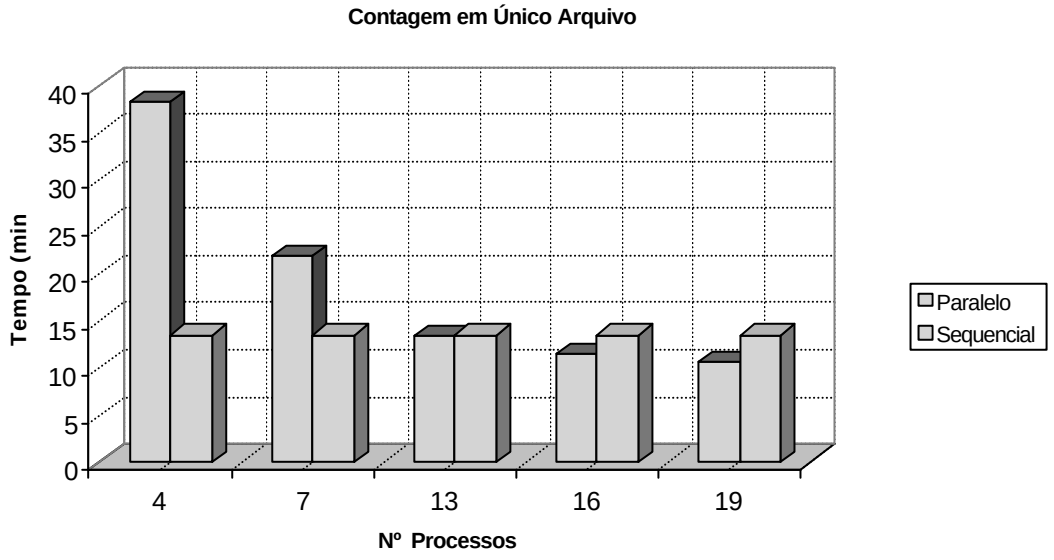

Figura 13 - Gráfico de Comparação dos Resultados (segundo Modelo Replicado)

A terceira comparação é dada com a segunda implementação do Modelo Replicado (arquivo de 922,31 *Mbytes*) . Visto na figura 13, o *speedup* inicial é de 0,35 e resulta em uma queda de 65% no desempenho do algoritmo paralelo sobre o seqüencial. Neste

teste o *speedup* de 1,27 representa o melhor desempenho do paralelo sobre o seqüencial. Observa-se no gráfico um decremento de tempo de execução da implementação paralela a medida que aumenta-se o número de processos, porém, após um determinado limite, o aumento do número de processos na sobrecarga do meio de comunicação e das unidades de execução, já descrito anteriormente.

5.4. Algoritmos para Busca de Tuplas

As seções seguintes demonstrarão os modelos implementados para a simulação da busca de tuplas, suas metodologias e seus resultados. Os modelos seqüenciais também serão descritos e servirão como base de comparação com os modelos paralelos.

5.4.1. Modelo simulando particionamento horizontal de dados

A modelo para simular o particionamento horizontal de dados, como descrito no item 5.3.2, utiliza vários arquivos para a execução dos testes paralelos e seqüenciais.

5.4.1.1. Metodologia

Foram utilizados três e doze arquivos tomando-se o melhor e o pior caso para a execução da busca paralela. Nenhuma simulação foi realizada para cobrir casos aleatórios para a operação de busca sobre os dados, sendo que tal avaliação poderá ser realizada em trabalhos futuros. O melhor caso caracteriza-se na presença da chave de busca no primeiro arquivo. Já o pior caso, pela presença da chave de busca no último arquivo. Em um procedimento seqüencial a caracterização do melhor e do pior caso é visível no tempo de execução. No caso de um algoritmo paralelo, o melhor e o pior caso possuem resultados idênticos. Atribui-se a esta afirmação o fato de que a execução paralela irá percorrer os arquivos paralelamente, sendo então, indiferente a localização da chave de busca em

determinado arquivo. O conteúdo dos arquivos e a distribuição dos processos entre as máquinas é o mesmo conforme descrito no item 5.3.1.1.

O algoritmo consiste na utilização de um arquivo por processo somando-se com um processo (coordenador) que fica ocioso aguardando o envio de informações de algum dos processos. Considerando a busca em três arquivos, sendo o primeiro deles contendo a chave de busca, quatro processo serão necessários para a execução. Os processos realizarão a pesquisa nos arquivos específicos e a cada dez tuplas percorridas, verificará se o coordenador lhe enviou alguma mensagem (através de uma comunicação não bloqueante). No momento em que algum processo encontrar a chave de busca, o mesmo irá finalizar a busca e enviará uma mensagem para o coordenador que será responsável em enviar uma mensagem de término para cada um dos outros processo que ainda estão realizando a busca, finalizando, assim, a busca paralela.

5.4.1.2. Resultados

Como a localização da chave de busca não altera o resultado final da execução da busca paralela, a tabela 9 mostra a média de tempo das cinco execuções utilizando o algoritmo de busca, notando-se o aumento do tempo de execução diretamente proporcional ao acréscimo de arquivos para ser realizada a busca. O tempo de execução é incrementado devido ao crescimento de comunicação entre os processos para a verificação de chave de busca encontrada.

Tabela 9 – Modelo Paralelo para Busca

Nº de Arquivos	Tempo (minutos)
3	1,536916667
12	6,337483333

5.4.2. Modelo seqüencial

Motivou-se a implementação do modelo seqüencial similar ao paralelo descrito no item 5.4.1 para uma comparação entre os mesmos.

5.4.2.1. Metodologia

Também utilizou-se, para a busca seqüencial, três e doze arquivos tomando a média de cinco execuções com cada número de arquivos. O melhor e o pior caso, citados no primeiro parágrafo do item 5.4.1.1 foram utilizados obtendo-se grandes diferenças entre resultados.

5.4.2.2. Resultados obtidos

A tabela 10 mostra os resultados obtidos para a execução do algoritmo seqüencial de busca em vários arquivos. Observa-se que, os melhores casos, utilizando três e doze arquivos os tempos são iguais. Se a chave de busca está contida no primeiro arquivo, o algoritmo ao encontra-la irá finalizar a busca, portanto o número de arquivos utilizados para este caso é irrelevante. Porém, o crescimento do tempo de execução é exponencial na utilização do pior caso para a busca da chave.

Tabela 10 - Modelo Seqüencial para Busca

Nº de Arquivos	Tempo (minutos)	Caso
3	2,896316667	Pior
3	0,560383333	Melhor
12	13,31123333	Pior
12	0,560383333	Melhor

5.4.3. Comparação dos resultados da busca

Tomando-se o melhor caso, onde a chave de busca está contida no primeiro arquivo, o algoritmo seqüencial mostra resultados superiores ao paralelo. A figura 14 expressa esta diferença através do gráfico em colunas, utilizando três e doze arquivos para tal fim.

A sobrecarga do meio de comunicação no algoritmo paralelo foi o fator mais relevante para a obtenção destes resultados. Considerando o melhor caso, como no modelo seqüencial não necessita de comunicação entre processos, seu resultado foi melhor. Calculou-se um *speedup* de 0,36 para a utilização de três arquivos e 0,09 para doze arquivos, significando, no segundo caso uma queda de desempenho de 91 % do algoritmo paralelo sobre o seqüencial.

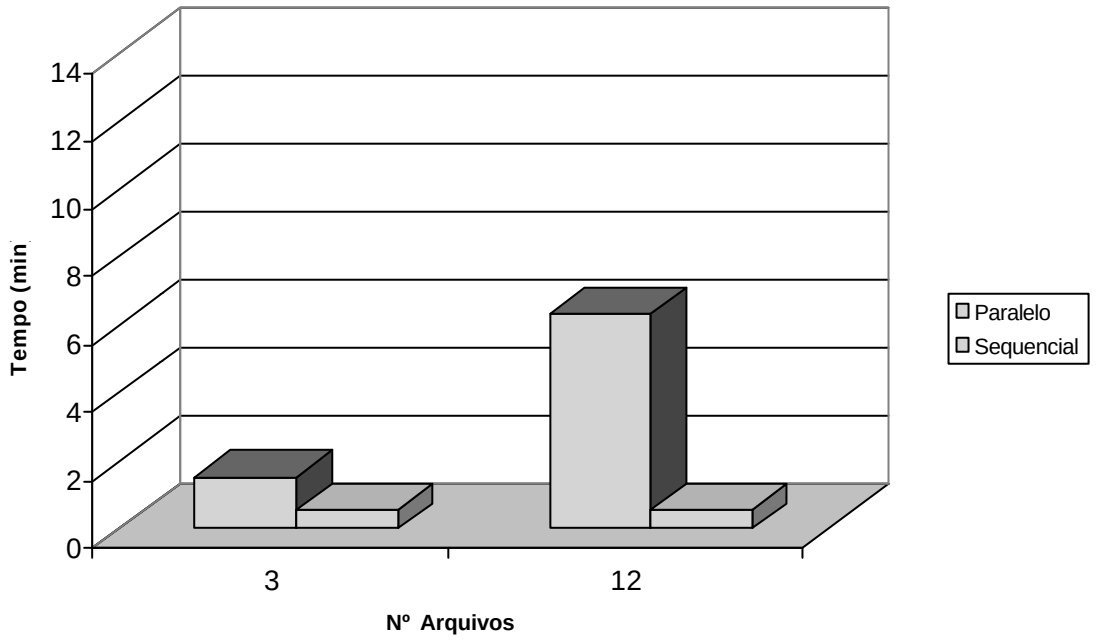

Figura 14 – Gráfico do Melhor Caso para Busca

No pior caso, onde a chave de busca encontra-se no último arquivo, o algoritmo paralelo mostrou melhores resultados que o seqüencial. O resultado do algoritmo

paralelo permanece o mesmo no pior e no melhor caso e, no algoritmo seqüencial o aumento é exponencial relacionando-se o melhor e o pior caso, então, a comparação favorece ao modelo paralelo. O gráfico da figura 15 mostra esta comparação.

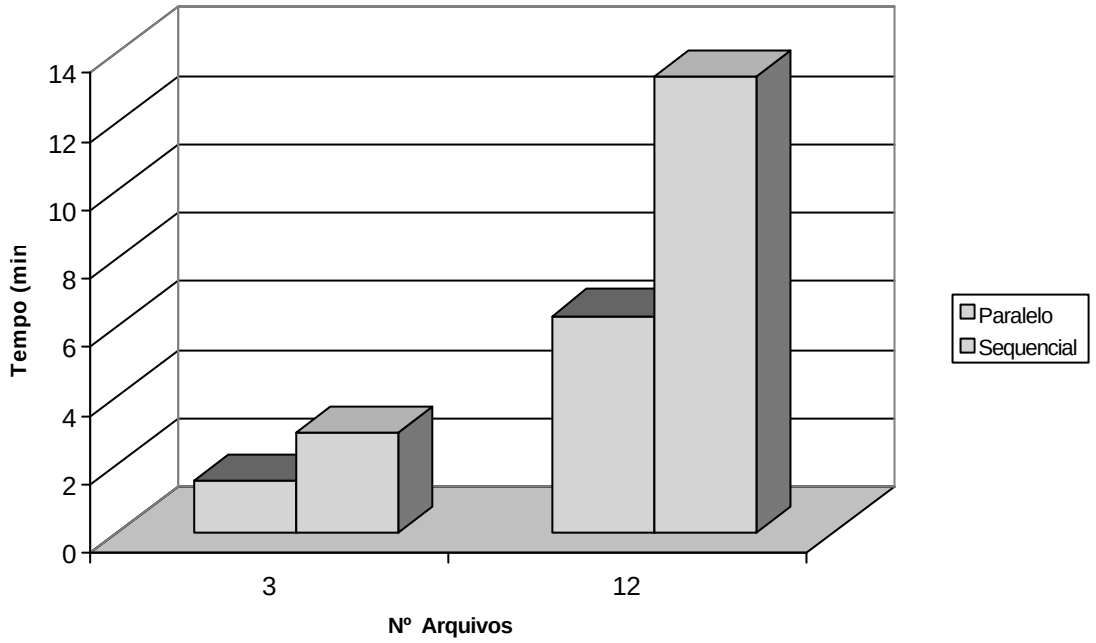

Figura 15 - Gráfico do Pior Caso para Busca

Os custos em comunicação no modelo paralelo são menos onerosos do que os custos da pesquisa em todos os arquivos no modelo seqüencial. O *speedup* encontrado na utilização de três arquivos foi de 1,88 e de 2,10 para a utilização de doze arquivos. O segundo resultado (*speedup* = 2,10) representa a execução do algoritmo paralelo no mínimo duas vezes mais rápida se comparado com o modelo seqüencial.

6. CONCLUSÕES

Este trabalho tem como objetivo o estudo da programação paralela aplicada a banco de dados, concluindo que a melhoria de certas atividades que um SGBD convencional efetua, pode ser alcançada por meio do paralelismo destas tarefas.

É importante observar que os algoritmos utilizados foram baseados na estrutura dos arquivos seqüenciais, sem considerar a utilização de estruturas mais elaboradas como árvores binárias, AVL ou outros recursos utilizados por banco de dados modernos.

Os resultados obtidos demonstram que a execução simultânea para determinadas tarefas em um sistema de banco de dados é viável. Porém, é necessário um tratamento transparente com o suporte do SGBD para a paralelização de consultas.

Em grande parte dos resultados obtidos neste trabalho, o algoritmo paralelo demonstrou melhor desempenho sobre o algoritmo seqüencial, considerando-se o tempo de execução como métrica de desempenho. Entretanto, o projeto de um modelo paralelo levanta questões sobre a viabilidade de se paralelizar tarefas. O projeto de um algoritmo paralelo depende das características do ambiente paralelo tais como a máquina paralela que será utilizada, a comunicação existente entre os processos, a estrutura da rede de interconexão utilizada, o balanceamento da carga de trabalho e a quantidade de processos necessários para que se possa alcançar o melhor desempenho possível. Fatores estes descartados no projeto de um modelo seqüencial. A quantidade de dados que será utilizada também é um fator fundamental no projeto de um sistema paralelo pois com um número pequeno de registros, por exemplo, o sistema seqüencial terá um resultado superior ao paralelo.

A tarefa de pré-projeto é importante para decidir entre o modelo paralelo ou seqüencial. Portanto, deve-se ponderar todos os pontos críticos para saber se, realmente, o projeto de um modelo paralelo será viável ou não.

A avaliação do impacto da inclusão de mais nós na máquina virtual paralela e o desenvolvimento de novos algoritmos (como classificação e ordenação) para fim de simulação e coleta de resultados com o objetivo de comparação com a implementação

seqüencial também poderão ser realizados como trabalhos futuros, além da implementação de um processador de consultas capaz de extrair o paralelismo de linguagens de consultas.

E, a união destes futuros trabalhos poderá a vir gerar uma coletânea de especificações em relação ao desempenho de algoritmos paralelos, para utilização conjunta na implementação de um SGBD paralelo.

REFERÊNCIAS BIBLIOGRÁFICAS

BARRETO, Ricardo M.; 1993. *Avaliação de Desempenho de Programas Paralelos*. Porto Alegre. Tese (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul.

CENAPAD-NE; Apostila de Paralelismo, Workshop Virtual de MPI. http://www.cenapadne.br/cursos. consultado na INTERNET em 15 de nov. 2000.

DEWITT, David; GRAY, Jim; 1992. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of The ACM*, Jun. 1992, Vol. 35, No. 6, p. 85-98.

ELMASRI, Ramez; SHAMKANT, Navathe B.; 2000. Fundamentals of Database Systems. 3ª Edição. Vancouver, Canadá: Addison-Wesley.

GAVILAN, César J.; 2000. *Síntese em Alto Nível de uma Rede de Interconexão Dinâmica para Multicomputador*. Florianópolis. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina.

HSIAO, D.; 1983. Advanced Database machine Architectures. Englewood Cliffs, N.J.: Prentice-Hall.

KORTH, Henry F.; SILBERCHATZ, Abraham; SUDARSHAN, S.; 1999. Sistema de Banco de Dados. 3ª Edição. São Paulo, SP: Makron Books.

MERKLE, Carla; 1996. *Ambiente para Execução de Programas Paralelos Escritos na Linguagem SUPERPASCAL em um Multicomputador com Rede de Interconexão Dinâmica*. Florianópolis. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina.

MEYER, Luiz A. V. C.; 1997. Paralelismo em SGBDOO com Memória Distribuída: Uma Implementação no PARGOA. Rio de Janeiro. Tese (Mestrado em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro.

MPICH; 2000; 1.2.1. Biblioteca para Programação Paralela. Disponível na INTERNET pelo endereço: http://www.mcs.anl/gov/mpi/mpich/download.html. consultado na INTERNET em 04 de fev. 2001.

ÖZSU, M. Tamer; VALDURIEZ, Patrich; 1999. Principles of Distributed Database Systems. 2ª Edição. Upper Saddle River, New Jersey: Prentice Hall

PACHECO, Peter S.; 1997. Parallel Programming with MPI. 1ª Edição. San Francisco, California: Morgan Kaufmann Publishers, Inc.

PREUS, Evandro; 1998. *MDX: Um Ambiente de Programação Paralela Baseada em Memória Virtual Distribuída*. Porto Alegre. Tese (Mestrado em Ciência da Computação) – Pontifício Universidade Católica / RS.

RICARTE, Prof. Ivan L. M.; 1996. Sistemas Paralelos de Banco de Dados: Arquiteturas e Algoritmos. Campinas, 1996. Apostila, Engenharia Elétrica e Computação, PET Informática, Universidade Estadual de Maringá.

RÍMOLO, Gustavo S.; ÁRABE, José, N. C.;1997. PVM x MPI: Um Estudo Comparativo em Redes de Workstations. In Simpósio Brasileiro de Arquitetura de Computadores – Processamento de Alto Desempenho (9 : Outubro-1997 : Campos do Jodão-SP). *Anais*. São Paulo: Escola Politécnica da USP. P.143-429.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; 1997. Operating System Concepts. 5ª Edição. New York, NY: John Wiley & Sons, Inc.

VELOSO, Paulo; SANTOS, Clesio dos; AZEREDO, Paulo; FURTADO, Antonio; 1997. Estrutura de Dados. 1ª Edição. Rio de Janeiro, Brasil: Editora Campus Ltda.