X Simpósio Brasileiro de Arquitetura de Computadores

Paralelização de Geração de Regras de Associação

B. Pôssas, F. Peligrinelli, W. Meira Jr., M. Carvalho, R. Resende
DCC - UFMG
Caixa Postal 702 - 30161-970
Belo Horizonte - MG
{bavep,flavia,meira,mlbc,rodolfo}@dcc.ufmg.br

Resumo

Mineração de dados é uma área de pesquisa emergente, cujo objetivo principal é extrair padrões e regras implícitos em banco de dados. Muitos algoritmos para mineração de regras de associação foram propostos. Entretanto, a pesquisa tem dado atenção principalmente à algoritmos seqüenciais.

Neste artigo apresentamos a paralelização de um algoritmo para determinação de regras de associação, utilizando o paradigma de memória compartilhada. Os resultados indicam que a nossa paralelização é escalável até oito processadores, independentemente das características da massa de dados.

Abstract

Data mining is an emerging research area, whose goal is to extract significant patterns or interesting rules implicit in databases. Many algorithms have been proposed for data mining of association rules. However, research so far has mainly focused in sequential algorithms.

In this paper we present an association rule algorithm paralelization which uses the shared memory paradigm. The results show that our parallelization scales up to eight processors, regardless of the characteristics of the transactions' database.

1 Introdução

Com o crescente volume de dados armazenados, organizações cada vez mais voltam-se para a extração de informações úteis em seus banco de dados. Mineração de dados é uma área de pesquisa emergente, cujo objetivo principal é extrair padrões e regras implícitos nestes bancos de dados. Mineração de dados na verdade combina pesquisadores de diversas áreas, como inteligência artificial, estatística e banco de dados. As técnicas de mineração de dados podem ser divididas em grupos de acordo com o padrão resultante da sua aplicação [1]: (1) classificação; (2) agrupamento; (3) regras de associação; e (4) previsão. Neste artigo enfocamos uma técnica para determinação de regras de associação, que explicitam correlações entre itens do arquivo em termos da sua ocorrência simultânea. Regras de associação são aplicadas na solução de vários problemas, desde suporte a decisão em telecomunicações até planejamento. Um exemplo popular de regras de associação é a determinação de produtos que são comprados simultaneamente em supermercados, sendo utilizados para fins de *marketing*.

Apesar do aumento do poder computacional disponível, algoritmos para determinação de regras de associação são ainda capazes de exaurir recursos de computadores atuais. Uma forma de minimizar o tempo de resposta destes algoritmos é a sua paralelização. Neste artigo investigamos a paralelização do algoritmo *Apriori* [4] utilizando o paradigma de memória compartilhada na máquina Sun Starfire ENT10000 [8]. A utilização desse modelo de computação para a nossa implementação paralela se justifica pela programação mais intuitiva, embora a abstração representada pelo seu uso possa comprometer o desempenho da aplicação. Analisamos o grau de paralelismo, as necessidades de sincronização e os problemas de localidade de referência relacionados à paralelização de aplicações em mineração de dados.

O restante do artigo é organizado da seguinte maneira. Na próxima seção descrevemos o problema de geração de regras de associação e apresentamos o algoritmo *Apriori*. Na Seção 3 apresentamos uma discussão sobre os possíveis pontos de paralelização do algoritmo. A Seção 4 apresenta o ambiente de execução da implementação paralela do algoritmo e os resultados experimentais obtidos. A Seção 5 apresenta alguns trabalhos correlatos. As conclusões são apresentadas na Seção 6.

2 Determinação de Regras de Associação

Nesta seção descrevemos formalmente regras de associação e o algoritmo *Apriori*, cuja paralelização será discutida.

Seja $I = \{i_1, i_2, ..., i_n\}$ um conjunto de itens. Seja D um conjunto de transações, onde cada transação T é um conjunto de itens tal que $T \subseteq I$. Associado a cada transação existe um identificador único (TID). Uma regra de associação é uma implicação da forma $X \rightarrow Y$, onde $X \subset I$, $Y \subset I$ e $X \cap Y = \emptyset$. A regra $X \rightarrow Y$ é válida para o conjunto de transações D com confiança c se $c\%$ das transações em D que contêm X também contêm Y. A regra $X \rightarrow Y$ tem suporte s no conjunto de transações D se $s\%$ das transações em D contêm $X \cup Y$. Dado um conjunto de transações D, o problema da determinação de regras de associação é gerar todas as regras que tenham suporte e confiança maiores que os valores mínimos especificados (*minsup* e *minconf*, respectivamente). Este problema pode ser dividido em duas etapas: (1) encontrar os conjuntos de itens que possuam suporte maior do que o suporte mínimo especificado (chamados de conjuntos freqüentes); e (2) gerar as regras de associação a partir dos conjuntos freqüentes de itens.

Uma aplicação dessas regras de associação é a análise de compras feitas em um supermercado, onde cada compra corresponde a uma transação. As transações são armazenadas em um banco de dados e consistem dos itens comprados. Após a aplicação de algoritmos como o *Apriori* às transações, obtemos regras da forma "80% das transações que contêm cerveja também contêm batatas fritas". Essas regras podem ser usadas para definir estratégias de *marketing* ou caracterização do perfil dos compradores.

Há dois problemas básicos a serem tratados quando implementamos algoritmos para regras de associação:

Associabilidade: quais são os itens entre os quais devem ser investigadas as associações.

Custo de enumeração: uma vez definidos os itens que potencialmente podem participar de uma regra, é necessário enumerar os possíveis conjuntos de itens que devem ser verificados, tarefa cujo custo cresce exponencialmente com o número de itens. Desta forma, é normalmente necessário adotar técnicas que reduzam a quantidade de conjuntos a serem enumerados.

O algoritmo *Apriori* [4] minimiza o custo de enumeração gerando somente conjuntos de itens cujos subconjuntos são freqüentes. O algoritmo se inicia com a contagem da freqüência dos itens no arquivo, determinando quais deles são freqüentes. Esses itens frequentes são utilizados para a geração dos conjuntos de itens potencialmente frequentes de tamanho 2, chamados de conjuntos candidatos. A seguir, verifica-se qual a freqüência de conjuntos candidatos na base de transações, descartando os conjuntos que não satisfaçam o critério de freqüência (*minsup*) e determinando os conjuntos freqüentes de tamanho 2, os quais são utilizados para determinar os conjuntos candidatos de tamanho 3. O processo continua até que não haja mais conjuntos freqüentes. Um vez determinados todos os conjuntos freqüentes, as regras de associação são derivadas.

2.1 O Algoritmo *Apriori*

Assumimos que em cada transação os itens estão ordenados em sua ordem lexicográfica. Chamamos de *tamanho* o número de itens em uma transação, e chamamos de $k-itemsets$ um conjunto de itens de tamanho k. Usamos a notação $c[1], c[2], ..., c[k]$ para representar um $k-itemset$ c que contém os itens $c[1], c[2], ..., c[k]$ com $c[1] < c[2] < ... < c[k]$.

Durante cada iteração do algoritmo somente os conjuntos determinados como freqüentes na iteração anterior são usados para gerar conjuntos candidatos, cujo suporte é determinado na iteração corrente. Um passo de corte elimina qualquer conjunto candidato que tenha um subconjunto que não seja freqüente. O algoritmo termina no passo t, se não há nenhum conjunto candidato de tamanho t ($t-itemsets$). A estrutura geral do algoritmo é apresentada na Figura 1. Uma breve discussão sobre cada etapa do algoritmo é dada nas próximas subseções.

```
1.  L1 = {frequent 1-itemsets}
2.  for (k=2; Lk-1<>0; k++) {
3.    Ck = generate_candidates(Lk-1); // Ver Secao 2.1.1
4.    for all transactions T in DB
5.      for all subsets t in T
6.        if (c is in Ck: c=t) c.count++;
7.    Lk = {c in Ck | c.count >= minsup};
8.  }
9.  for all Lk, k>2
10.    generate_rules(Lk, Lk); // Ver Secao 2.1.2
```

Figura 1: O Algoritmo *Apriori*

2.1.1 Geração de Conjuntos Candidatos

A função **generate_candidates** tem como argumento L_{k-1}, o conjunto freqüente de itens de tamanho $k-1$. Esta função retorna conjuntos candidatos de tamanho k e consiste de dois passos: enumeração e corte.

Como mencionado, a enumeração dos conjuntos candidatos de tamanho $k+1$ é baseada nos conjuntos de tamanho k. O processo se inicia pela determinação de todos os pares de conjuntos que compartilhem os primeiros $k-1$ itens. O conjunto candidato é gerado pela união desses dois conjuntos, formando um conjunto de tamanho $k+1$.

Durante o corte são removidos todos os conjuntos de itens candidatos $c \in C_k$ se algum subconjunto de c com tamanho $k-1$ não pertencer a L_{k-1}. Nesse passo é verificada

a existência de todos os subconjuntos do conjunto candidato C_{k+1} gerado no passo da combinação. Se um subconjunto $c \in C_k$ não pertencer ao conjunto freqüente de itens L_k então esse candidato é descartado. Este algoritmo é apresentado na Figura 2.

```
1.  for (i=1; i<=k-1; i++)
2.    if (p.item[i] > q.item[i])
3.      return;  // Nao gera candidato
4.  if (p.item[k] < q.item[k]) {
5.    for all itemsets c in Ck  // Passo do corte
6.      for all (k-1)subsets s in c
7.        if (s is not in Lk-1)
8.          Remove c de Ck;
9.    Ck = p U q;
10. }
```

Figura 2: Algoritmo para geração de conjuntos candidatos

Por exemplo, seja L_3={{1 2 3},{1 2 4},{1 3 4},{1 3 5},{2 3 4}}. Depois da fase de combinação, C_4={{1 2 3 4},{1 3 4 5}}. O passo de corte irá remover o conjunto de itens {1 3 4 5} porque o conjunto {1 4 5} não está em L_3, resultando em L_4={1 2 3 4}.

2.1.2 Geração das Regras

Na geração de regras, para cada conjunto freqüente L_k, encontramos todos os subconjuntos não nulos de L_k. Para cada subconjunto s, escrevemos a regra da forma $s \rightarrow (L_k - s)$ se a razão $conf = suporte(L_k)/suporte(s)$ é maior ou igual à confiança mínima ($minconf$). Consideramos todos os subconjuntos de L_k para gerar regras com múltiplos tamanhos.

2.1.3 Implementação

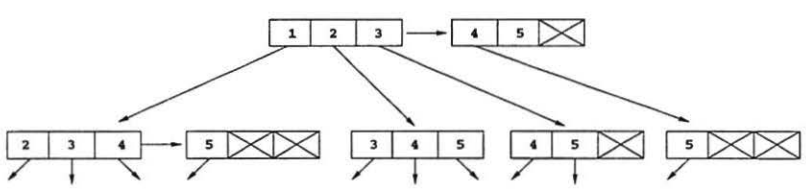

Figura 3: Estrutura da árvore

Para tornar a contagem do suporte dos conjuntos de itens mais eficiente, armazenamos os conjuntos na forma de uma árvore, onde cada nível n da árvore representa conjuntos de itens de tamanho n. Um nível é composto por um ou mais nodos, e cada nodo contém um número pré-fixado de itens. Além do identificador do item, são também armazenados um contador, que contém o suporte do conjunto representado pela entrada, e um apontador para um nodo no próximo nível (possivelmente nulo para os nodos folha). Se o número de itens em um nodo excede o valor pré-fixado, um novo nodo é criado e encadeado ao nodo cujo número máximo de itens foi excedido.

Por exemplo, a Figura 3 mostra uma árvore contendo os seguintes conjuntos frequentes: {{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4,5}}.

Esta estrutura se mostra eficiente tanto para geração de conjuntos candidatos quanto para contagem, uma vez que pode-se localizar qualquer conjunto em tempo proporcional ao tamanho do conjunto. Uma vez que o algoritmo opera em apenas um nível por iteração, a estrutura também se mostra adequada a paralelização, permitindo acesso a vários conjuntos simultaneamente.

3 Paralelização do Algoritmo *Apriori*

Nesta seção apresentamos a paralelização do algoritmo *Apriori* para um ambiente de memória compartilhada. Nossa opção por essa paralelização se deve não apenas a necessidade da redução do custo computacional, mas também pelo fato de ser mais intuitiva a programação em memória compartilhada.

Analisando o algoritmo da Figura 1 determinamos duas etapas que poderiam ser paralelizadas: (1) geração de conjuntos candidatos (linha 3) e (2) contagem do suporte (linhas 4 a 7). Como discutido na Seção 2.1.3, a estrutura de dados utilizada permite plena exploração do paralelismo para estas duas etapas.

Na geração de conjuntos candidatos, os nodos folhas da árvore (ou seja, os últimos conjuntos freqüentes determinados) são particionados entre os processadores, e cada processador gera os candidatos relativos à sua partição. Como os conjuntos estão particionados não é necessária a sincronização durante o processo, apenas ao fim da geração, o que é feito por uma barreira.

A contagem do suporte é dividida em dois passos, tendo em vista a necessidade de leitura das transações do arquivo, o que tem de ser feito serialmente tendo em vista a Starfire não efetuar operações de entrada e saída em paralelo. Assim, no primeiro passo são lidas e armazenadas p transações, onde p é igual ao número de processadores. Durante o segundo passo cada processador conta o suporte para os conjuntos de uma dessas transações. Como dois ou mais processadores podem estar contando o suporte para um mesmo conjunto simultaneamente foi necessária a inclusão de um semáforo que faz a exclusão mútua entre os acessos à estrutura.

Entre as seções paralelas e seriais, incluindo as operações de entrada e saída (E/S) há barreiras que fazem a sincronização dos processadores. Na Figura 4 podemos observar a porção paralelizada do algoritmo, correspondente às linhas 2 a 8 do algoritmo sequencial (Figura 1).

```
1.  for (k=2; Lk-1<>0; K++) {
2.    Pk-1 = (Lk-1)itemsets handled by processors;
3.    Ck = generate_cadidates (Pk-1);
4.    for all transactions T in DB
5.      read p transactions T1...Tp in T
6.        for all subsets t in Ti
7.          if (c is in Ck: c=t) c.count++;
8.    Lk = {c in Ck | c.count >= minsup};
9.  }
```

Figura 4: O Algoritmo *Apriori* Paralelo

4 Resultados

Nesta seção descrevemos o ambiente de execução, a geração de banco de dados sintéticos e apresentamos os resultados da paralelização implementada.

4.1 Ambiente de Execução

Para relatar a performace relativa à execução da versão paralela do algoritmo *Apriori*, realizamos vários testes na Máquina Sun Starfire ENT10000 [8], cujas principais características são:

- Gabinete com 32 processadores ULTRASPARC 250MHz
- 1MB cache por processador
- 8GB mémoria RAM, em módulos de 512MB
- SMP escalável permitindo reconfiguração dinâmica, com partição flexível do sistema
- Largura de banda de I/O de 6,4 GB/s (pico)
- System Boards com 4 processadores e 1GB de memória RAM
- Interconexão Gigaplane-XB, com largura de banda de 12,5 GB/s
- Desempenho de pico (Linpack Parallel) de 16 GFLOPS

A nossa implementação é baseada em Posix Threads, e o escalonamento de *threads* a processadores fica a cargo do sistema operacional, que é o Solaris, versão 5.5.1.

4.2 Geração de Dados Sintéticos

De forma a poder explorar os efeitos de variações nas características das massas de dados a serem processadas, utilizamos um gerador de transações sintético [4], cujas cargas sintéticas simulam transações do ambiente real. Estas cargas levam em conta que pessoas tendem a comprar itens de forma correlacionada, ou seja, cada conjunto de itens é um conjunto freqüente em potencial. Os tamanhos das transações são tipicamente agrupados em torno de uma média e poucas transações têm muitos itens. Os tamanhos típicos para conjuntos freqüentes de itens são também agrupados em torno de uma média, sendo que poucos conjuntos freqüentes possuem um número elevado de itens.

Para gerar um banco de dados, o programa de geração dos dados sintéticos recebe os seguintes cinco parâmetros: (1) $|D|$ número de transações, (2) $|T|$ tamanho médio para as transações, (3) $|I|$ tamanho máximo para os conjuntos de itens candidatos, (4) $|L|$ número máximo de conjuntos de itens candidatos e (5) $|N|$ número de itens.

Desta forma geramos seis conjuntos sintéticos de transações, todos com 10.000 transações num universo de 2.500 itens, tendo até 50.000 conjuntos de itens candidatos. Essas massas de dados se diferenciam pelo tamanho médio da transação e tamanho máximo para os conjuntos de itens candidatos, que variaram de 200 a 700 e 20 a 70, respectivamente. A Tabela 1 apresenta as características desses conjuntos de transações.

BD	T	D	I	L	N	Suporte	Tamanho
A	200	10k	20	50k	2500	0.8%	8.78MB
B	300	10k	30	50k	2500	1.2%	13.13MB
C	400	10k	40	50k	2500	1.5%	17.45MB
D	500	10k	50	50k	2500	2.0%	21.81MB
E	600	10k	60	50k	2500	2.4%	26.14MB
F	700	10k	70	50k	2500	2.7%	30.50MB

Tabela 1: Propriedades das Massas de dados

4.3 Análise dos Resultados

Nesta seção analisamos os resultados obtidos pela nossa paralelização do algoritmo *Apriori*. Nossa análise se concentra em duas dimensões: (1) escalabilidade e (2) sensibilidade às características dos dados. Para avaliar escalabilidade, executamos o programa paralelo utilizando até 16 processadores. No que tange à sensibilidade às características dos dados, comparamos os custos computacionais e o comportamento da nossa implementação para execuções sobre as massas de dados descritas na Seção 4.2. O suporte utilizado para cada massa de dados é também apresentado na Tabela 1, enquanto que todos os experimentos foram realizados com confiança mínima igual a 80%.

O perfil de desempenho da implementação para as várias combinações de processadores e massas de dados pode ser visto na Figura 5, que mostra o tempo médio por processador em cada configuração. Desta forma, as barras que compõem o gráfico foram divididas em nove grupos, de acordo com o número de processadores utilizados, sendo cada grupo composto de seis barras, correspondentes às seis massas de dados (A, B, C, D, E e F, respectivamente). Cada barra, por sua vez, é dividida conforme as seguintes seis categorias:

Conta: Tempo associado à contagem do suporte dos conjuntos candidatos.

E/S: Tempo associado às operações de entrada e saída, mais especificamente leitura das transações e escrita das regras.

Cand: Tempo associado à determinação dos conjuntos candidatos.

PI (Paralelismo Insuficiente): Tempo durante o qual *threads* ficaram esperando em barreiras porque não havia tarefas que elas pudessem executar, como ocorre durante operações de E/S.

DC (Desbalanceamento de Carga): Tempo durante o qual *threads* ficaram esperando em barreiras porque receberam menos "trabalho" que outras, tendo em vista o escalonamento estático do programa (como descrito na Seção 3).

Sinc: Tempo associado à sincronização em semáforos, ou seja, quanto tempo uma *thread* esperou, tendo em vista que outra *thread* estava dentro da seção crítica. Estes tempos não estão representados no gráfico da Figura 5 por não serem maiores que 1% do tempo total de execução.

Analisando este gráfico, observamos que o melhor desempenho foi obtido com oito processadores para todas as massas de dados. Neste caso, os *speedups* variaram de 5.71

a 7.50 e, conforme esperado, a maior massa de dados (F) teve o maior ganho resultante do paralelismo. Para as execuções com 2, 4 e 6 processadores, o ganho foi próximo de linear, com decrescimento do tempo associado a desbalanceamento de carga e paralelismo insuficiente. É interessante notar que o tempo associado a desbalanceamento de carga e paralelismo insuficiente não cresce significativamente com a utilização de mais de 8 processadores. Por outro lado, o tempo para contagem do suporte cresce, sendo o fator determinante para a degradação de desempenho observada para todas as massas de dados. Isto ocorre devido aos custos de comunicação entre os processadores, que, tendo em vista limitações dos mecanismos de medição utilizados, não puderam ser fatorados dos tempos de computação propriamente dita.

Um fenômeno particularmente interessante ocorre com relação aos *speedups* quando são utilizados dez processadores ou mais, e execuções que processem a massa de dados A apresentam menor degradação de desempenho (e.g., 4.79 em dezesseis processadores), enquanto que a massa de dados F apresentou o pior *speedup* (4.10 em 16 processadores). Podemos explicar esse fenômeno com base na observação de que o custo computacional associado à contagem do suporte cresce mais com o tamanho da massa de dados que os custos das outras operações, como por exemplo E/S. Esta situação pode ser facilmente observada nas barras relativas à execução com um processador (Figura 5), pois a despeito do aumento do tamanho da massa de dados, os tempos de E/S não variaram mais que 10%, enquanto que os tempos para contagem do suporte cresceram uma ordem de magnitude. Mais ainda, o gráfico da Figura 6 mostra que há uma queda súbita entre oito e dez processadores. Neste último caso, com base na documentação da Starfire [8], não conseguimos uma explicação para essa degradação sistemática de desempenho quando da utilização de mais de duas *system boards*, uma vez que a quantidade de dados manipulada é a mesma para cada massa de dados (A-F). Finalmente, deve-se também notar que o tempo de E/S, quando amortizado entre vários processadores, deixa de ser um custo relevante para a nossa análise.

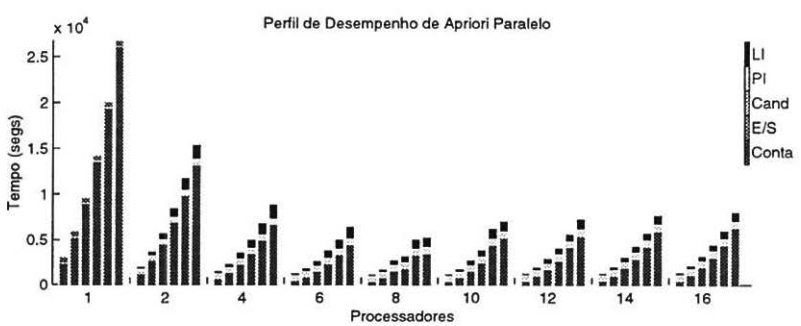

Figura 5: Perfis de Desempenho do *Apriori* Paralelo

5 Trabalhos Correlatos

Muitos algoritmos para determinção de regras de associação foram propostos na literatura desde a introdução deste problema em [2] (Algoritmo AIS).

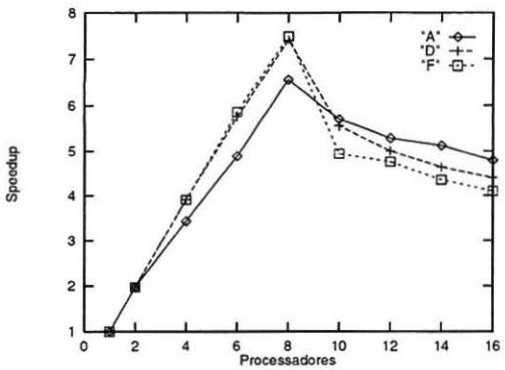

Figura 6: *Speedup* para as massas de dados A, D e F

O algoritmo *Apriori* [4] usa a propriedade que qualquer subconjunto de um conjunto freqüente de itens também é freqüente. Este algoritmo tem desempenho superior ao AIS, mas ambos são polinomiais. O algoritmo DHP [6] usa uma hashtable num passo k para tornar mais eficiente o passo de corte para conjuntos de itens de tamanho $k+1$. O algoritmo partition [7] minimiza o tempo de E/S, pois o banco de dados é acessado somente duas vezes. Na primeira passada são gerados todos os conjuntos de itens candidatos e na segunda o suporte destes conjuntos é contado.

Existem poucos trabalhos voltados à paralelização de algoritmos para regras de associação. Implementações paralelas do algoritmo *Apriori*, num ambiente de memória distribuída como na IBM SP2 são apresentadas em [3] e num ambiente de memória compatilhada são apresentadas em [9]. Em ambos os trabalhos, os autores reforçam que as implementações ainda são ineficientes e há muito o que melhorar.

6 Conclusões

Neste artigo, apresentamos uma implementação paralela do algoritmo *Apriori* para o paradigma de memória compartilhada. Discutimos também os principais pontos de paralelização do algoritmo, tais como a fase de contagem do suporte e a geração dos conjuntos candidatos. Apresentamos os resultados experimentais para a execução do algoritmo para diversos bancos de dados sintéticos.

Os resultados indicam que a nossa implementação tem boa escalabilidade utilizando até oito processadores, quando então a comunicação passa a ser um fonte de degradação de desempenho significante. Como esperado, execuções sobre massas de dados maiores permitiram a obtenção de melhores *speedups*, embora essas mesmas execuções tenham sido as mais afetadas pelo aumento dos custos de comunicação, principalmente quando foram utilizados dez ou mais processadores.

Pretendemos continuar este trabalho explorando a variação de outras características das massas de dados e investigando os efeitos da comunicação, de forma a explicar o fenômeno relatado na Seção 4. Também pretendemos investigar a utilização de estruturas de dados alternativas como *hash trees* [1] e formas de otimizar as operações de entrada e saída.

Referências

[1] R. Agrawal, T. Imielinski, and A.Swami. Database mining: A performance perspective. In *IEEE Trans. on Knowlegde and Data Engineering.*, 1993.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Intl. Conf. Management of Data*, May 1993.

[3] R. Agrawal and J. Shafer. Parallel mining association rules: Design, implementation, and experience. Technical Report 10004, IBM Almaden Research Center, San Jose, CA, Jan 1996.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *The 20th International Conference on Very Large Data Bases*, September 1994.

[5] G. John. *Enhancements to the Data Mining Process.* PhD thesis, Stanford University, March 1997.

[6] J. Park, M. Chen, and P. Yu. An effective hash based algorithm for mining associative rules. In *ACM SIGMOD.* ACM, May 1995.

[7] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *The 2lst International Conference on Very Large Data Bases*, 1995.

[8] Sun Microsystems. The Ultra Enterprise 10000 Server, 1997. Technical White Paper.

[9] M. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. Technical Report 618, The University of Rochester Computer Science Department, Rochester, NY, 1996.