Otimização de Localidade de Dados em Aplicações Reais e Estratégias de Alocação Dinâmica de Memória *

Edson Toshimi Midorikawa
emidorik@lsi.usp.br
Laboratório de Sistemas Integráveis
Departmento de Engenharia Eletrônica
Escola Politécnica da Universidade de São Paulo
São Paulo, SP 05508-900

RESUMO

A localidade de dados é um dos fatores determinantes nos modernos sistemas de computação de alto desempenho. Face a ineficácia das estratégias atuais, é proposto um novo enfoque para a otimização de programas: a estratégia *Communion*. A palavra chave em nossa estratégia é cooperação; todos os programas de sistema responsáveis pela criação e execução de aplicações devem interagir entre si de forma a garantir uma gerência eficiente da memória e um alto grau de localidade de dados. Entre as interações possíveis, abordamos duas delas: a primeira, chamada *Resurgence*, enfoca a interação dos componentes do sistema de compilação para garantir a criação de um programa executável eficiente. A segunda, chamada *Adjacence*, aborda aspectos relacionados à alocação dinâmica de memória, com a cooperação do sistema de execução e do sistema operacional.

ABSTRACT

Data locality is one of the key factors in modern high-performance computing systems. Due to the inefficacy of current strategies, we propose a new approach to program optimization: the *Communion* strategy. The keyword is cooperation; all system programs that are responsable to generate or execute application programs must interact among themselves in order to guarantee an efficient memory management and an high degree of data locality. Among all possible interaction, we examined two of them: the first, named *Resurgence*, focuses on the interaction among the components of the compilation system in order to get an efficient executable program. The second, named *Adjacence*, deals with the aspects related to dynamic memory allocation, with the cooperation between the run-time system and the operating system.

1. Introdução

Com o agravamento da disparidade de velocidades do processador e da memória, os sistemas de computação modernos têm introduzido uma complexa hierarquia de memória. A relação do desempenho de pico dos modernos processadores e o desempenho real alcançado pelos computadores atuais reflete a grande diferença de velocidades dos elementos internos (registradores, cache) e os elementos externos (vários níveis de memória principal) da hierarquia de memória. Diversas estratégias em hardware e software têm sido empregadas para solucionar este problema.

Infelizmente, as soluções empregadas atualmente focalizam suas atenções em alcançar alto desempenho mantendo-se inalteradas as aplicações. Isto é, dado o comportamento atual do padrão de acessos à memória, procura-se encontrar uma estratégia para explorar tal característica. Apresentamos assim a proposta de um nova estratégia de melhora de desempenho de sistemas de memória. São abordados alguns aspectos complementares: a otimização do padrão de acessos à memória, com a participação conjunta dos programas de sistema e, finalmente, a proposta de um sistema de execução que leva em consideração o sistema de memória.

Este trabalho é estruturado da seguinte forma: apresentamos a aplicação de nossa estratégia de otimização da localidade de referências de duas aplicações na seção 2. A seção seguinte apresenta a nossa estratégia para melhorar a alocação dinâmica de memória. E finalmente, colocamos nossas conclusões na seção 4.

* Este trabalho de pesquisa foi financiado em parte pela Finep (Projeto Hipersistemas) e pelo programa RHAE/CNPq.

2. Otimização de Localidade em duas Aplicações Reais

Resurgence é uma estratégia para otimização de localidade de dados que emprega um ponto de vista alternativo [MIDO97]. Ao invés de procurar minimizar o tempo de execução dos programas, Resurgence propõe mudar o foco da otimização para a demanda de memória. Esta estratégia propõe a interação entre os componentes do sistema de compilação (compilador, montador, ligador e carregador) no processo de otimização do programa gerado.

Apresentamos aqui os resultados[1] obtidos na aplicação da estratégia *Resurgence* em duas aplicações reais nas áreas de codificação de voz e de processamento de imagens médicas tridimensionais. Ambas as aplicações foram desenvolvidas por professores do Departamento de Engenharia Eletrônica da Escola Politécnica da USP.

2.1. VSELP

Uma das aplicações utilizadas para aplicação das técnicas de otimização foi um programa de codificação de sinais de voz que implementa o algoritmo VSELP ("*vector sum excited linear prediction*") [RAMI95]. O programa inicialmente foi desenvolvido em linguagem C para uma placa DSP com processador TMS320C30. O código foi recodificado para as plataformas SGI e Sun, tornando o software mais modular, facilitando assim a otimização das diversas funções internas.

O programa original (`v0`) apresenta como saída um conjunto de tabelas apresentando o estado atual da codificação e dos parâmetros calculados que não representam informação útil do processo de codificação, servindo apenas para indicar ao usuário sobre a estado da execução do programa. Como primeira otimização (`v1`), foi eliminada a apresentação destas tabelas durante a execução do programa.

Para orientar a otimização, uma vez que o programa foi dividido em vários módulos, cada um contendo um certo conjunto de funções correlacionadas. O perfil de execução do programa original foi analisado para verificarmos os tempos gastos em cada módulo, usando-se o comando `profile`. Diversas versões foram geradas (`v2` a `v5`), com a aplicação de otimizações em diversas rotinas da aplicação. Abaixo são apresentados os tempos de execução das diversas versões executadas em quatro estações de trabalho diferentes (tabela 1).

Tabela 1 - Tempos de execução das versões do VSELP.

	Tempo de execução (s)			
versões VSELP	SGI PowerSeries 4D/480	SGI Indigo	SUN SPARCstation 10	SUN SPARCstation 20
v0	267,167	163,691	254,646	148,969
v1	267,166	158,032	250,531	148,920
v2	142,338	75,172	130,511	71,148
v3	132,379	70,191	123,706	68,226
v4	132,729	70,150	122,219	67,448
v5	132,712	69,334	120,961	67,468

Verifica-se que os tempos de execução das versões `v3`, `v4` e `v5` são apenas pouco menores ao da versão `v2`. Isto pode ser explicado pelo fato da versão `v2` ser aquela onde foi otimizada a função `VSl_Lag` responsável por 36% to tempo total de execução. Portanto, com sua otimização o tempo total de execucão diminuiu em quase 50% (*speedup* de 2 vezes). O mesmo não ocorre nas versões `v3`, `v4` e `v5`, pois estas versões representam otimizações de funções que representam apenas de 10% a 1% do tempo total de execução. A otimização realizada na rotina `VSl_Lag` incluíram a aplicação das transformações de *unroll-and-jam*, *scalar replacement* e a remoção de comandos if.

Concluímos então que, para programas mais extensos, não é vantajoso otimizar todas as funções possíveis pois muitas delas não são executadas com muita freqüência, enquanto que algumas são responsáveis pela maior parte do tempo de processamento.

[1] Uma versão mais detalhada deste trabalho pode ser obtida no endereço `http://www.lsi.usp.br/~emidorik`.

2.2. 3D Region Filtering: filtragem de volumes por vizinhança adaptativa

Uma outra aplicação analisada foi um programa de filtragem de imagens volumétricas por vizinhança adaptativa [LOPE95]. O processo de otimização se concentrou nas rotinas `grow_region_*`. Utilizando um arquivo de entrada com um volume de `128x64x64` voxels e executando os estudos em um SGI Power Series 4D/480VGX, os tempos de execução das versões originais e otimizadas foram os seguintes.

Tabela 2 - Comparação entre as versões originais e otimizadas.

	grow_region	transposição	tempo total	melhora
xy	46,89	0,00	46,89	
xy otimizado	42,88	0,00	42,88	1,094
xz	49,22	0,00	49,22	
xz otimizado	42,90	2,57	45,47	1,082
yz	51,73	0,00	51,73	
yz otimizado	43,78	3,05	46,83	1,105
3D	135,28	0,00	135,28	
3D otimizado	125,36	0,00	125,36	1,079

Da tabela 2 acima, observamos que o ganho de desempenho com relação ao tempo de crescimento das regiões ficou da ordem de 9% para a filtragem por planos (`xy`, `xz`, `yz`) e da ordem de 8% para a filtragem por região adaptativa tridimensional (`3D`). Uma análise mais aprofundada no padrão de acessos nos mostrou que não há localidade temporal nos acessos aos pontos e um mesmo ponto é acessado várias vezes desnecessariamente apenas devido à recursão adotada. Uma forma possível de melhorar a localidade seria transformar o algoritmo recursivo num equivalente iterativo onde não se faz acesso aos pontos já referenciados. Uma outra forma de melhorar o desempenho seria otimizar o padrão de acessos de uma algoritmo de filtragem com vizinhança fixa, pois seu padrão de acessos é muito mais simples e previsível.

Uma conclusão deste estudo é a de que não basta apenas realizar uma análise do padrão de acessos de um programa e em seguida otimizá-lo, pois os resultados dependem da natureza da aplicação. É necessário ir além, como por exemplo com novas formas de codificação de um mesmo algoritmo ou até com a adoção de outro algoritmo para solucionar o problema.

3. Adjacence: localidade em alocação dinâmica de memória

Um componente normalmente esquecido nas pesquisas atuais é o sistema de execução ("run time system") nos sistemas de alto desempenho. O sistema de execução é composto basicamente pelas rotinas de biblioteca do sistema que são chamadas durante a execução dos programas. Exemplos de tais rotinas são aquelas responsáveis pela alocação dinâmica de memória (`malloc`), alocação de memória compartilhada, criação e gerência de processos, obtenção de informações do sistema operacional (`getpid`, `getrusage`, `pacct`) e controle de recursos.

Normalmente a alocação dinâmica de memória é realizada como uma extensão da área de dados do programa. De forma a que se tenha acessos mais localizados é importante que também estas rotinas de sistema tenham "conhecimento" do relacionamento dos dados, de forma a melhorar a colocação dos dados no espaço virtual do processo requisitante. A biblioteca `malloc` faz uso das chamadas `sbrk` ou `brk` para alocar um espaço extra no segmento de dados do processo e faz a gerência dele. Conforme a dinâmica de alocação e dealocação de áreas de memória, os blocos de memória gerenciados pela biblioteca podem ficar muito fragmentados. Normalmente, é adotado um algoritmo de alocação como o first-fit, best-fit ou next-fit, ou ainda uma variante do algoritmo buddy [KNUT97].

Uma solução para melhorar a localidade nas alocações dinâmicas é possibilitar a interação do programa com o sistema de execução de forma a alterar o comportamento interno da alocação. De forma a aumentar a localidade espacial das alocações, requisições correlacionadas poderiam utilizar uma chave de identificação que indicasse esta correspondência mútua. Desta forma, o alocador poderia executar a alocação em regiões de memórias próximas.

A localidade entre alocações pode ser obtida com uma pequena alteração na interface padrão da biblioteca `malloc`. Uma proposta de modificação da interface padrão é o acréscimo de um parâmetro adicional que identificaria cada área de memória alocada dinamicamente. Esta chave poderia ser utilizada em requisições posteriores para garantir uma proximidade em relação a uma dada área alocada anteriormente. Convém ressaltar que o uso da chave é opcional, fazendo com que os programas atuais permaneçam válidos.

```
#include <stdlib.h>
void *malloc  (size_t size [, key_t key]);
void *calloc  (size_t nmemb, size_t el_size [, key_t key]);
void *realloc (void *ptr, size_t new_size [, key_t key]);
void  free    (void *ptr);

key: chave de alocação de memória (uso opcional)
```

Algumas considerações devem ser levadas em conta quando se propõe uma nova interface de programação: similaridade com a interface padrão, modificações para incorporação da nova funcionalidade e portabilidade do código da aplicação. Esta questão sobre a localidade de referências é abordada também em [PART97][TSEN97].

4. Conclusão e Trabalhos Futuros

O problema do acesso à memória principal nos modernos sistemas de computação de alto desempenho não tem sido trabalhado de uma forma global e integrada. Este trabalho mostrou alguns trabalhos na estratégia *Communion*, resultado de nossas pesquisas em métodos de gerência de memória para sistemas de alto desempenho. Dentre as técnicas propostas na estratégia, detalhou-se duas delas: *Resurgence* e *Adjacence*. Os estudos e avaliações em aplicações reais mostraram bons resultados. Apesar destes resultados promissores, sentimos que é necessário um estudo complementar com a extensão das propostas: por exemplo, uma integração entre o compilador e o sistema operacional na *Resurgence* poderia ser aplicada em outras situações como no escalonamento de programas paralelos e na distribuição de dados pelos módulos de memória de um sistema distribuído.

Exemplos de trabalho futuro incluem a integração da estratégia *Resurgence* em um sistema automático de compilação e otimização de programas (como o Suif ou Polaris) e a investigação de alternativas de implementação da estratégia *Adjacence* sobre a biblioteca GNU malloc.

Referências Bibliográficas

[KNUTH97] KNUTH, D. E. **The art of computer programming**: v.1 - fundamental algorithms. 3rd ed., Addison Wesley, 1997.

[LOPE95] LOPES, R. D. & RANGAYYAN, R. N. *Three dimensional region based filters for noise removal.* In: IASTED Int. Conf. in Signal and Image Processing. Las Vegas. **Proceedings**. p.424-427. November 1995.

[MIDO97] MIDORIKAWA, E. T.; SATO, L. M.; ZUFFO, J. A. *Communion: a new strategy for memory management in high-performance computer systems.* In: CACIC'97, La Plata, Argentina. **Anales**. Octubre 1997.

[RAMI95] RAMÍREZ, M. A. et alii. *Filter structures for a DSP VSELP speech coder.* In: International Conference on Signal Processing Applications & Technology, Boston. **Proceedings**. v.II, p.1854-1858. 1995.

[TSEN97] TSENG, C.-W. **Data layout optimizations for high-performance architectures**. Technical report CS-TR-3818. Department of Computer Science, University of Maryland. 1997.

[WILS95] WILSON, P. R. et alii. *Dynamic storage allocation: a survey and critical review.* In: 1995 International Workshop on Memory Management, Kinross, Scotland, UK. 1995. **Proceedings**. Springer Verlag, Lecture Notes in Computer Science.