

INVESTIGANDO A OTIMIZAÇÃO AUTOMÁTICA DE CÓDIGO ATRAVÉS DO PARADYN.

Evandro Augusto Marucci, Aleardo Manacero Junior. –
Ciência da Computação – Bacharelado em Ciência da Computação – Departamento de
Ciências de Computação e Estatística – Instituto de Biociências, Letras e Ciências Exatas –
Campus de São José do Rio Preto.

Nos últimos anos as tecnologias para a produção de compiladores se desenvolveram consideravelmente. Este desenvolvimento teve um papel fundamental quanto ao aumento de desempenho dos programas, obtido a partir de técnicas cada vez mais aprimoradas de otimização de código. Muitas delas são incorporadas aos compiladores e, em geral, são aplicadas sem levar em conta o tempo de execução em partes específicas do programa. Outras técnicas, no entanto, são aplicadas dinamicamente, e levam em consideração os dados coletados através de uma execução prévia do programa [1].

Nesta segunda classe de otimizações podemos incluir o alinhamento de funções e o desdobramento de laços. Essas técnicas, quando mal aplicadas, podem piorar o tempo de execução do programa, além de aumentar significativamente o tamanho do executável. Para que isto não ocorra, uma execução prévia deve investigar a viabilidade de aplicação dessas técnicas em trechos previamente identificados como gargalos de execução, o que é bastante útil em ferramentas para análise dinâmica de desempenho, como o Paradyn [2] [3].

Numa fase anterior desse projeto [2] obteve-se a identificação inicial de laços de repetição que representassem gargalos de execução, usando-se a estrutura original do Paradyn [3], uma ferramenta para análise de desempenho de sistemas paralelos e distribuídos, e a arquitetura do processador *sparc* como base. Esses trechos são laços de repetição com um tempo de execução maior que 20% do tempo total da função de que fazem parte. O projeto atual investiga técnicas de otimização de código possíveis de serem aplicadas nesses laços, de modo que o tempo de execução do mesmo diminua. No entanto, como já observado, as otimizações indicadas implicam no aumento no código gerado, o que pode diminuir o desempenho global por diminuir a taxa de acertos na cache, por exemplo. Isso obriga a que toda e qualquer otimização seja avaliada, o que é fácil de fazer dinamicamente através do Paradyn. Posteriormente, a partir de uma catalogação de resultados poderá ser feito um refinamento nas otimizações, evitando-se a aplicação em casos que notadamente levam a piora do desempenho [4].

Para o desenvolvimento desse projeto foi necessário portar o código anterior da plataforma *sparc* para a arquitetura x86, visto que essa tem sido uma das arquiteturas mais utilizadas em sistemas computacionais de alto desempenho. O porte foi realizado através de um processo de reconstrução, que envolveu as seguintes fases:

- **Identificação de laços de repetição**, que são caracterizados pela existência de uma instrução de salto para algum endereço anterior ao da própria instrumentação. Com isso podem ser identificados três tipos de laços, figura 1, que são laços simples, aninhados e entrelaçados, sendo o tratamento de cada um feito de forma diferente.
- **Identificação das modificações no código original do Paradyn e dos trechos dependentes de arquitetura**, buscando identificar quais seriam os pontos em que o uso de processadores diferentes implicaria em códigos também diferentes, como máscaras para identificação de instruções e métodos para geração do código instrumentado, entre outros.
- **Decodificação e manipulação de instruções x86 através de métodos existentes no Paradyn**, em que se construiu uma estrutura para manipular todas as instruções da função sob análise, sendo que enquanto na arquitetura Sparc, todas as instruções são de tamanho fixo (4 bytes), o que facilita muito esse processo, para o x86 esse procedimento não é viável, visto que as instruções têm tamanho variável, podendo ter de 1 a 15 bytes.
- **Identificação dos pontos de instrumentação**, que são os pontos de entrada ou saída de um laço ou função, armazenando-os em listas que permitem definir a quantidade de instrumentações de entrada e saída até a posição da n-ésima instrução da função. A figura 2 ilustra esses pontos para um laço de repetição em uma função gargalo.

- **Correção do deslocamento de instruções de salto e verificação da necessidade de alteração do tamanho das instruções de salto**, quando é preciso tratar o problema provocado pela inserção de instrumentações no código original e as alterações nos endereços destinos das instruções de salto, para que se contemple os novos valores de deslocamento.
- **Instrumentação de laços**, em que se implementou, para o x86, todas as formas de instrumentação de laços previstas.
- **Medição dos laços instrumentados**, em que se implementou as funções que determinam os tempos gastos nos laços e sua relação com o tempo de execução das funções, para a sua possível classificação como laço gargalo.

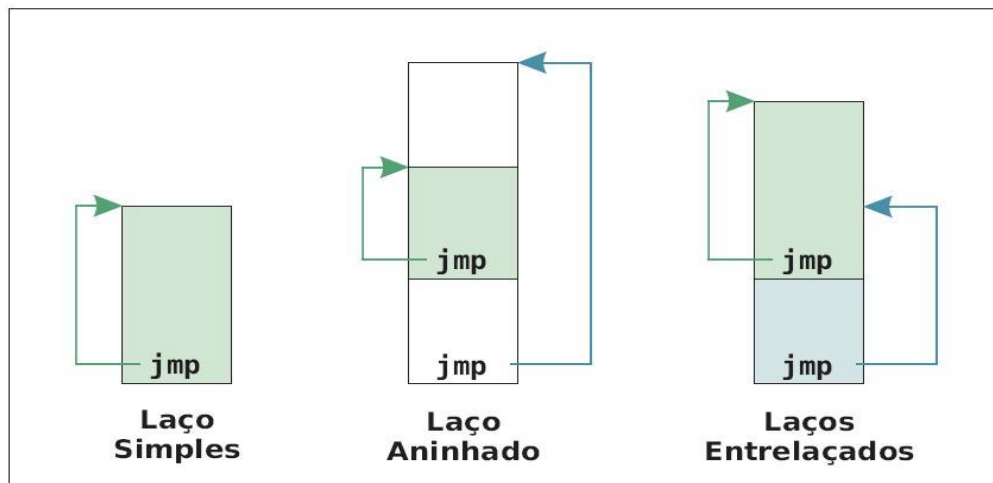


Figura 1. Tipos de laços encontrados na instrumentação

As otimizações dos laços de repetição identificados como gargalos são feitas através de algumas técnicas básicas, como o alinhamento de funções internas aos laços, propagação de constantes e eliminação de instruções LOAD e RESTORE redundantes. Essas técnicas foram escolhidas dentre as várias técnicas investigadas por questões de viabilidade de implementação. Estas três técnicas foram implementadas no projeto, automatizando a inserção dessas otimizações no código binário em análise pelo Paradyn.

O desdobramento de laços, que também é uma técnica importante, não foi implementado pois a verificação de dependência de dados entre as instruções de um laço, necessária para sua aplicação, apresenta uma complexidade que inviabilizou o desenvolvimento de um módulo de aplicação automática desta técnica.

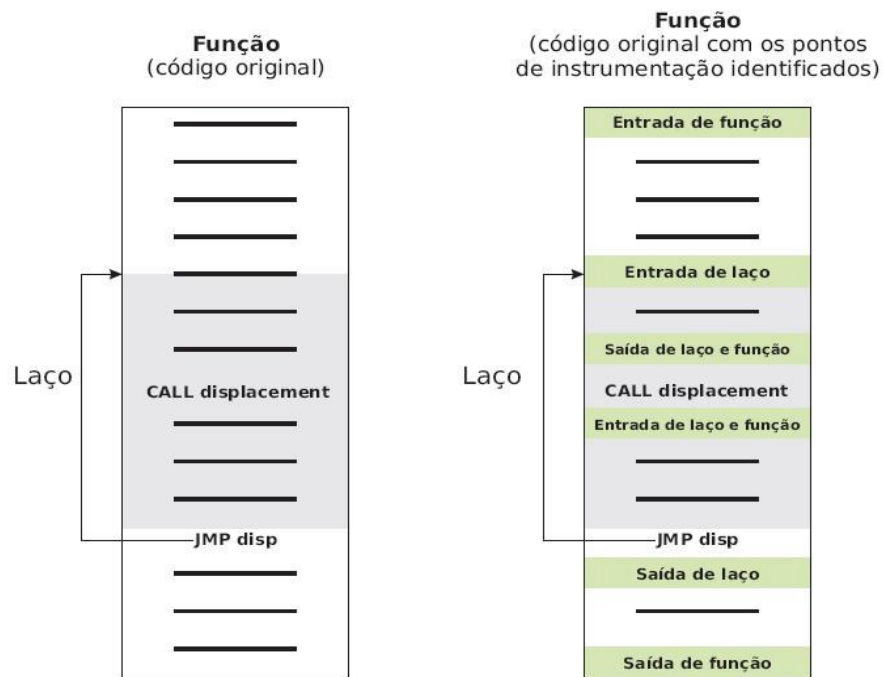


Figura 2. Identificação dos pontos de instrumentação

O processo de otimização começa com a aplicação do alinhamento de funções. Ela é implementada para produzir um código muito mais eficiente, o que é feito com a substituição da chamada CALL, no laço da função gargalo, com uma instância da própria função. A figura 3 mostra a aplicação do alinhamento de funções, com as devidas correções de deslocamento.

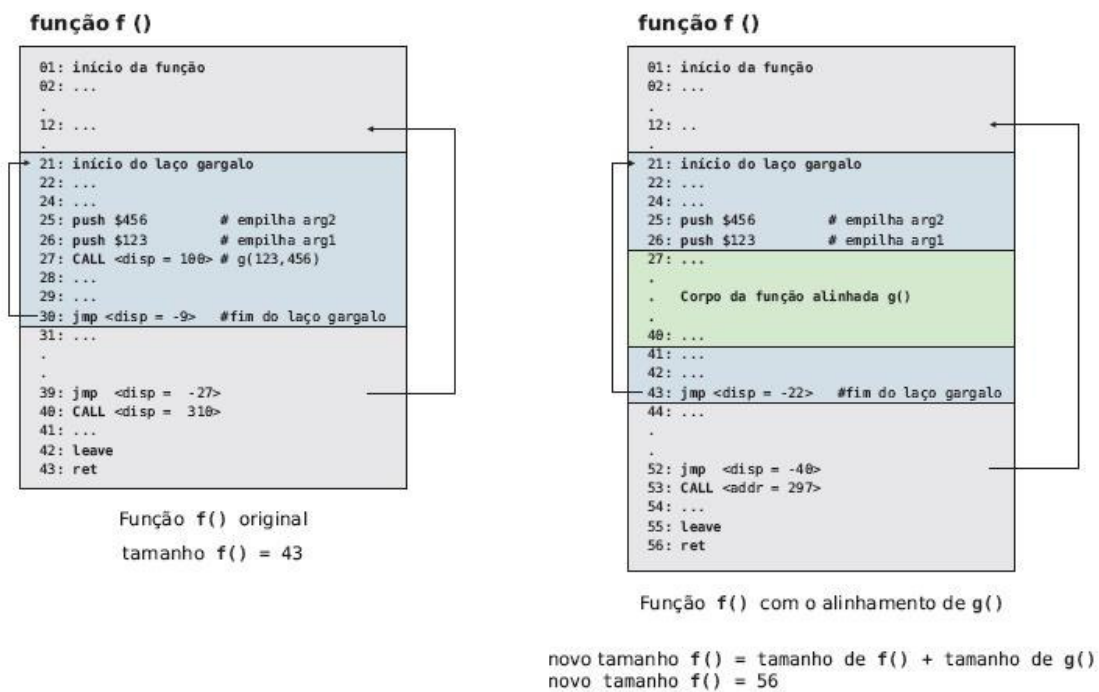


Figura 3. Exemplo da aplicação do alinhamento de função

A partir do alinhamento de função, as outras duas otimizações são então aplicadas. A propagação de constantes é a primeira delas e possui o objetivo de propagar argumentos constantes de uma função chamada para dentro de seu corpo. A otimização de remoção de instruções LOAD/RESTORE redundantes é aplicada em seguida. Nesse caso, procura-se encontrar múltiplas instruções LOAD de uma mesma localização e substituí-las por instruções MOV entre registradores. A idéia é encontrar duas instruções, I e J, que carreguem uma mesma variável para diferentes registradores da CPU. Para isso é preciso provar que a posição de memória não foi alterada entre a execução de I e J, e que o registrador carregado por I ainda contém o valor a ser passado para J.

Por fim, realizaram-se os testes para verificar se os módulos adicionados ao Paradyn estavam gerando, corretamente, as instrumentações para as funções reconhecidas como gargalo e se o código instrumentado executava da forma prevista. Verificou-se também eficiência das otimizações aplicadas. Em programas com laços gargalos, que possuem chamadas de funções de tamanho pequeno, porém repetidas um grande numero de vezes, a otimização obteve um aumento de desempenho razoável, sem aumentar consideravelmente o tamanho do código da aplicação. Isto ocorre pois as instruções de carregamento e encerramento de uma chamada de função foram eliminadas e os parâmetros que antes eram passados pela chamada de função foram propagados para o corpo da função.

Dessa forma, notou-se que os laços de repetição com chamadas de funções pequenas, e que executam uma grande quantidade de vezes, são pontos bons para a otimização. Assim, a implementação de um otimizador automático deve levar em conta esses resultados de forma a decidir um valor limitante para que as otimizações sejam aplicadas apenas nos casos em que o aumento de desempenho em relação ao aumento de código seja proveitoso.

Referências Bibliográficas

- [1] CHANG, P. P.; MAHLKE, S. A.; HWU, W. mei W. Using profile information to assist classic code optimizations. In: . New York, NY, USA: John Wiley & Sons, Inc., 1991. v. 21, n. 12, p. 1301-1321. ISSN 0038-0644.
- [2] BOCARDO, D.R.; NAKASHIMA, H.J.; JUNIOR, J.N.F. & CASAGRANDE, L.S.; "Medição de Desempenho de Programas Paralelos - Estendendo o Paradyn"; Monografia de projeto final de graduação, Ibilce/Unesp, 2004.
- [3] MILLER, B. P. et al. The paradyn parallel performance measurement tool. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 28, n. 11, p. 37-46, 1995. ISSN 0018-9162.
- [4] ZHAO, P.; AMARAL, J. N. To inline or not to inline ? enhanced inlining decisions. *16th Workshop on Languages and Compilers for Parallel Computing*, October 2003

Bolsa: FAPESP