UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CRISTIAN FERNANDO FLORES CASTAÑEDA

Otimização Unroll and Jam através da refatoração

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Nicolas Bruno Maillard
Orientador

Porto Alegre, Agosto de 2011

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Castañeda, Cristian Fernando Flores

Otimização Unroll and Jam através da refatoração / Cristian Fernando Flores Castañeda. – Porto Alegre: PPGC da UFRGS, 2011.

76 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2011. Orientador: Nicolas Bruno Maillard.

1. Otimização. 2. Unroll and Jam. 3. Refatoração. 4. Fortran. I. Maillard, Nicolas Bruno. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus pela oportunidade que me proporcionou na realização deste mestrado. Também quero agradecer a minha família, especialmente meus pais Fernando e Marisol por todo o suporte prestado durante estes anos, e também as minhas irmãs Daniela e Leticia pelo respaldo e toda ajuda oferecida durante esta etapa. Também gostaria de agradecer a minha namorada Médelin por todo o apoio e companhia.

Gostaria também de agradecer ao meu orientador Nicolas Maillard pelo apoio e pela confiança na realização deste trabalho, e ao CNPq pela concessão da bolsa de estudos. Também gostaria de agradecer aos meus colegas de sala e de grupo que de alguma forma me auxiliaram durante todo este período de estudos e pesquisa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AST	*Abstract Syntax Tree*
BLAS	*Basic Linear Algebra Subprograms*
BRAMS	*Brazilian Atmospheric Modeling System*
CDT	*C, C++ Development Tools*
CFG	*Control Flow Graph*
CPTEC	Centro de Previsão de Tempo e Estudos Climáticos
FSF	*Free Software Foundation*
GCC	*GNU Compiler Collection*
GPL	*General Public License*
GPPD	Grupo de Processamento Paralelo e Distribuído
GNU	*Gnome Not Unix*
HPC	*High Performance Computing*
IDE	*Integrated Development Environment*
RI	Representação Intermediária
IMSL	*International Mathematics and Statistics Library*
INPE	Instituto Nacional de Pesquisas Espaciais
LAPACK	*Linear Algebra Package*
MPI	*Message-Passing Interface*
NAG	*Numerical Algorithms Group*
OLAM	*Ocean Land Atmosphere Model*
OpenMP	*Open Multi-Processing*
PTP	*Parallel Tools Platform*
RTL	*Register Transfer Language*
SAG	*Software Architecture Group*
SSA	*Static Single Assignment*
UFRGS	Universidade Federal do Rio Grande do Sul

UML	*Unified Modeling Language*
VPG	*Virtual Program Graph*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

As otimizações de um programa podem ser efetuadas no código intermediário gerado na fase de compilação, ou através da *Performance Refactoring* que consiste na inserção de otimizações diretamente no código fonte da aplicação.

Na estrutura do código fonte, os laços de iteração possuem um importante impacto no desempenho da aplicação, pois consomem elevados tempos de execução, sendo assim, é fundamental que estes sejam alvo de otimizações. Uma das otimizações de laço é o *Unroll and Jam*, que aplica operações de reestruturação de laços aninhados, permitindo utilizar eficientemente a hierarquia de memória. Os passos desta otimização consistem em primeiramente desenrolar os laços a serem otimizados, e posteriormente fusioná-los formando um bloco único de instrução.

Neste trabalho se propôs usar refatoração de código para automatizar o *Unroll and Jam* de laços em nível do código fonte, o que possibilita acumular os efeitos da otimização no programa original com as melhorias tradicionais propiciadas pelo compilador na fase de otimização do código intermediário. A implementação foi realizada com base nas estruturas da ferramenta Photran 7.0, que consiste em um ambiente de desenvolvimento integrado para a linguagem Fortran 77-2008.

Os resultados de desempenho obtidos demonstraram que a utilização de *Unroll and Jam* através da *Performance Refactoring*, com diferentes níveis de otimização, obteve ganhos significativos de desempenho em comparação a otimização realizada pelo compilador da Intel. Deve-se destacar que a aplicação conjunta de ambas as técnicas de otimização, originou resultados melhores que a utilização de uma técnica somente. Como trabalhos futuros, é interessante que novas pesquisas possam ser realizadas na obtenção de índices de desenrolamento ótimos e na aplicação conjunta de outras abordagens de otimização. Também pode haver estudos que permitam que outras otimizações sejam aplicadas pela *Performance Refactoring*.

Palavras-chave: Otimização, Unroll and Jam, Refatoração, Fortran.

Unroll and Jam optimization through refactoring

ABSTRACT

The optimizations of a program can be performed on the intermediate code, generated during compilation time, or through the Performance Refactoring that consists of insertion of optimizations directly in the source code of the application.

In the source code structure, iteration loops have a major impact in the performance of the application because they are time-consuming code and, thus, an essential target of optimizations. One loop optimization is the Unroll and Jam that applies restructuring operations in nested loops allowing efficient usage of memory hierarchy. Its optimization steps are: to unroll the target loops, and to join them forming a single block of instruction.

This work proposes the usage of refactoring to automate Unroll and Jam loops in source code level, allowing to accumulate the effects of optimization in the original program with the optimizations in the intermediate code by traditional compilers. The implementation was based on structures of Photran tool 7.0, which consists of an integrated development environment for Fortran 77-2008.

The performance results using Unroll and Jam by Performance Refactoring with different levels of unrolling, showed significant gains compared to the optimization performed by the Intel compiler. It should also be noted that the joint of both optimization techniques has led to better results than the use of an individual technique. As future works, new researches can be performed to obtain optimal rates of unrolling and to join the application of other optimization approaches. Furthermore, studies may search for optimizations to be applied by the Performance Refactoring.

Keywords: Optimization, Unroll and Jam, Refactoring, Fortran.

1 INTRODUÇÃO

1.1 Contexto e Motivação

O *software* ao longo do ciclo de vida necessita de novas funcionalidades de acordo com as necessidades do usuário e do desenvolvedor. A inserção destas funcionalidades pode ocasionar modificações estruturais e funcionais no código fonte, podendo comprometer os resultados e a manutenção da aplicação. Estes problemas podem acontecer quando a aplicação não está apta para receber novas funcionalidades, ou quando estas não são inseridas corretamente no código fonte.

Um modo a fim de evitar estas consequências indesejáveis, é a utilização da refatoração que consiste na aplicação de técnicas de melhorias na estrutura interna do *software* sem afetar os resultados produzidos pelo mesmo (FOWLER, 1999). A refatoração modifica o código fonte da aplicação, deixando o código mais reusável e flexível para futuras alterações no código da aplicação (GARRIDO; JOHNSON, 2003).

A refatoração começou a ser utilizada na programação orientada a objetos, e tornou-se muito utilizada nas aplicações deste paradigma. Porém, durante os últimos anos, novas pesquisas apareceram visando a inclusão da refatoração em outros paradigmas.

Em relação a computação de alto desempenho (HPC), a aplicação da refatoração é pouco explorada, porém é possível destacar dois grandes campos de aplicação nesta área:

- Manutenção de código.
- Otimização de código.

A manutenção de código é uma área relativamente nova que pode ser aplicada a sistemas de HPC que possuam códigos legados. Neste sentido, a refatoração é um ponto fundamental para que programas de HPC legados possam ser atualizados e novos recursos possam ser integrados. Sobre este aspecto, é importante que estas refatorações sejam realizadas através de ferramentas integradas de desenvolvimento, pois assim é possível unir a manutenção de recursos legados e o desenvolvimento de novos recursos em uma única ferramenta.

Aplicações implementadas com versões antigas da linguagem Fortran servem como exemplo de manutenção de código, pois os novos recursos disponíveis nas últimas versões do Fortran provavelmente não podem ser inseridos nas aplicações implementadas pelas antigas versões da linguagem (RISSETI; CHARAO, 2011).

Por outro lado, na otimização de código, as otimizações que visam aumentar o desempenho das aplicações são denominadas de *Performance Refactoring*. A aplicação destas refatorações deve ser automatizada através de ferramentas de desenvolvimento a fim de

evitar a inserção de erros pelo programador no momento da reestruturação do código fonte.

A utilização de otimizações em Fortran é fundamental, pois o desempenho pode ser melhorado em aplicações em Fortran que implementem algoritmos matemáticos e físicos através de laços de iteração aninhados que demandem alto poder computacional.

Sobre o ponto de vista do desempenho do código fonte da aplicação, os laços de iteração são mais propensos a otimização, pois consomem elevados tempos de execução. Entre estas otimizações, *Unroll and Jam* se destaca pela eficiência na utilização da hierarquia de memória através de operações de reestruturação de laços aninhados (CRAWFORD; WADLEIGH, 2000).

Aplicações otimizadas para uma determinada arquitetura podem usufruir de modo mais eficiente do poder computacional disponível. Estas otimizações podem ser inseridas na aplicação através de otimizações no código intermediário por meio de compiladores, ou através de otimizações no código fonte por meio de ferramentas de desenvolvimento.

Em muitos casos alcançar um bom nível de desempenho através de otimizações no código fonte não é uma tarefa trivial, pois reconfigurar todo o código da aplicação para ser otimizado para uma determinada arquitetura, pode tornar-se uma tarefa custosa e propensa a erros. Por isso, é importante que haja ferramentas que possam pré-processar o código fonte de aplicação, otimizando os blocos de instruções e permitindo que o programador possa posteriormente inserir diretivas de paralelização ou novas otimizações pelo compilador.

Sendo assim, a automatização de otimizações no código fonte através de ferramentas de refatoração torna a ação de otimizar menos custosa e mais segura. Além disso, a portabilidade das otimizações das aplicações entre diferentes arquiteturas é realizada de modo mais confiável. Do ponto de vista da fase de desenvolvimento da aplicação, através destas ferramentas é fornecido ao programador a possibilidade de realizar as tarefas de otimização e programação ao mesmo tempo. O Photran pode ser citado como uma ferramenta para automatização de refatorações para a linguagem Fortran 77-2008 (BEATON; RIVIERES, 2006). Esta ferramenta implementa diversas refatorações para esta linguagem e fornece um núcleo sofisticado para inserção de novas refatorações.

1.2 Objetivos e Contribuição

Sobre estes aspectos, o objetivo principal do trabalho foi a aplicação da otimização *Unroll and Jam* no código fonte através da *Performance Refactoring*. Para isto, a implementação foi realizada através das estruturas de dados do Photran 7.0.

A fim de avaliar a implementação do presente trabalho, aplicações científicas em Fortran implementadas através de diferentes laços de iteração foram utilizadas como estudo de casos.

Em relação a contribuição deste trabalho, os resultados obtidos das aplicações demonstraram que melhores índices de desempenho podem ser atingidos a partir da otimização *Unroll and Jam* automatizada pela ferramenta desenvolvida. Também é importante destacar que novas pesquisas sobre refatoração podem ser aplicadas por outras otimizações.

1.3 Organização do Texto

Este trabalho está organizado do seguinte modo: o capítulo 2 apresenta a revisão bibliográfica sobre os conceitos e diferentes trabalhos relacionados à refatoração.

Na capítulo 3 são descritos as diferentes otimizações existentes na literatura para laços de iteração. Neste capitulo é apresentado a otimização *Unroll and Jam* e seus benefícios.

No capítulo 4 é descrito a aplicação de otimização tanto em nível de código intermediário como em nível de código fonte. Para a otimização no código intermediário é apresentado o compilador GCC para análise e otimização de laços. Enquanto para a otimização no código fonte é apresentada a ferramenta Photran, nas quais são descritas as estruturas *Abstract Syntax Tree* e *Virtual Program Graph* para alteração do código fonte.

No capítulo 5 é apresentada a implementação de *Unroll and Jam* realizada através das estruturas de dados da ferramenta Photran 7.0.

No capítulo 6 são apresentados os resultados obtidos da presente implementação através do algoritmo de multiplicação de matrizes e da subrotina SGTSV da biblioteca Lapack.

Por fim, no capítulo 7 são descritas as conclusões deste trabalho em relação aos resultados obtidos e sua importância para a comunidade científica.

2 REFATORAÇÃO

Este capítulo tem o objetivo de introduzir um breve histórico sobre as origens e conceitos relacionados à refatoração, relatar suas principais áreas de pesquisa, demonstrando a sua importância nos diferentes campos da computação.

Também neste capitulo são apresentados os conceitos da refatoração na área de HPC, e os benefícios para as aplicações.

2.1 Conceitos

A refatoração é o processo de alteração de um sistema de *software* de modo que o comportamento externo do código não mude, mas que sua estrutura interna seja melhorada. A utilização de técnicas de refatoração reestrutura o *software*, aplicando uma série de refatorações sem alterar o seu comportamento observável (FOWLER, 1999). Podem-se destacar cinco benefícios primários na utilização da refatoração (FOWLER, 1999):

- Melhorar o projeto de *software*.
- Tornar o código fonte mais legível.
- Aprimorar a detecção de falhas no código.
- Agilizar o desenvolvimento de *software*.
- Otimizar o código fonte pela *Performance Refactoring*.

Sobre o ponto de vista da gerência de projetos, um projeto de *software* tende a se deteriorar durante o tempo a medida que diversos programadores envolvidos no projeto alteram o código fonte do *software*. Este deterioramento do projeto, decorrente da alteração contínua do código fonte, ocasiona dificuldades na legibilidade do código fonte durante a fase de desenvolvimento. Sobre este problema, a refatoração organiza o código fonte da aplicação de modo a simplificar e remover partes de código que não estejam em locais apropriados, garantindo uma boa consistência do projeto.

A fase da manutenção do *software* também é beneficiada, pois a refatoração permite gerar códigos melhores estruturados para que eles possam ser modificados pelo próprio programador e por demais profissionais envolvidos na manutenção. Outro ponto importante a ser destacado, é o aprimoramento na detecção de falhas do código, pois a partir de um código bem estruturado as falhas são percebidas e corrigidas mais rapidamente.

Todos os benefícios citados anteriormente tornam o desenvolvimento de *software* mais ágil, pois através de um projeto consistente com códigos fontes simples e sem falhas é possível evitar o deterioramento e qualificar o projeto final de *software*.

Sobre o modo de aplicação da refatoração, o processo de refatorar deve ser realizado em fases pequenas e contínuas de modo a prevenir a introdução de defeitos no código. A grande maioria das refatorações pode demorar segundos ou minutos para serem aplicadas, porém grandes refatorações podem demorar dias, semanas, ou meses para as transformações serem concluídas (KERIEVSKY, 2004).

2.2 Breve Histórico

Os trabalhos iniciais em relação à refatoração foram realizados por William F. Opdyke (OPDYKE, 1992), orientado pelo professor Ralph Johnson da Universidade de Illinois em Urbana-Champaign, no qual trabalhava com o objetivo de pesquisar e desenvolver técnicas para tornar o processo da evolução do software menos custoso e mais simples (BONIATI; CHARAO, 2008).

Posteriormente, John Brant e Don Roberts, desenvolveram a primeira ferramenta para automatização de técnicas de refatoração chamada de *Refactoring Browser* (ROBERTS et al., 1996) (ROBERTS; BRANT; JOHNSON, 1997) voltada para código Smalltalk que introduzia pela primeira vez a automação de refatorações primitivas, simplificando e incentivando a utilização da técnica de refatorar (BONIATI; CHARAO, 2008).

Desde então, muitas pesquisas foram conduzidas nas diversas áreas da computação, algumas com o objetivo de catalogar e documentar as refatorações, outras como automatizar tais técnicas em ferramentas ou IDEs.

2.3 Estado da arte

Em relação às pesquisas sobre refatoração, estas podem ser classificadas em quatro áreas (MENS et al., 2003):

- Formalismo.
- Aplicação de técnicas de refatoração.
- Suporte a ferramentas.
- Linguagens de programação.

Na área de formalismo existem pesquisas relacionadas a divisões de programas (ETTINGER, 2007), transformações de grafos (MENS; TAENTZER; RUNGE, 2007), e métricas de software (BOIS, 2006).

Na área de aplicação de técnicas de refatoração, estas são organizadas de forma que orientem sua aplicação sistemática em diferentes áreas. Neste caso, na engenharia de *software* a reestruturação é utilizada para atualizar códigos legados, ou para migrar códigos a outras linguagens ou paradigmas de programação (FANTA; RAJLICH, 1999) (OVERBEY; NEGARA; JOHNSON, 2009).

A visualização de software é outra técnica que pode ser beneficiada pela refatoração através de diagramas (GRISWOLD et al., 1996), onde ferramentas gráficas são utilizadas para detectar códigos duplicados (DUCASSE; RIEGER; DEMEYER, 1999). Nesta área também há estudos para identificar quais tipos de técnicas de refatoração podem ser aplicadas dependendo da natureza dos sistemas (MENS; TOURWÉ, 2004).

Na área de suporte, estudam-se ferramentas de apoio ao desenvolvimento que possam automatizar técnicas de refatoração, atuando especificamente sobre um paradigma

e uma linguagem de programação (WATSON; DEBARDELEBEN, 2006) (ROBERTS et al., 1996) (GRAF; ZGRAGGEN; SOMMERLAD, 2007). Neste sentido, há estudos na geração de ferramentas de refatoração (MURPHY-HILL; BLACK, 2008), e trabalhos recentes que estudam o impacto e a forma de refatorar programas funcionais (LI, 1992) (LI; THOMPSON, 2008) e lógicos (SEREBRENIK; SCHRIJVERS; DEMOEN, 2008).

Nas linguagens de programação, o conceito principal se aplica igualmente independente do paradigma de programação, mas a construção de ferramentas e a própria sistematização de técnicas possuem características específicas. Destacam-se também os estudos na tentativa de evoluir aplicações legadas (LIU; BATORY; LENGAUER, 2006). Há trabalhos que abordam o uso de refatorações em diagramas UML (ASTELS, 2002) assim como a automatização de refatorações integradas a editores UML (BOGER; STURM; FRAGEMANN, 2002).

Alguns trabalhos mais recentes abordam problemas estruturais de linguagens de programação específicas, como trabalhos relacionados a automatização de ações de refatoração para código da linguagem C (GARRIDO; JOHNSON, 2002) (GARRIDO; JOHNSON, 2003), pesquisas referentes a linguagem Java (KIEZUN et al., 2007) (CHEN et al., 2008), e alguns trabalhos no contexto da computação de alto desempenho (HPC) para linguagem Fortran (OVERBEY et al., 2005) (OVERBEY; NEGARA; JOHNSON, 2009) (MATTHIAS RIEGER BART VAN ROMPAEY; LIEVIER, 2007).

Na área de banco de dados, há trabalhos que discutem a refatoração no contexto da evolução de esquemas relacionais (BOEHM et al., 2009) (AMBLER; SADALAGE, 2006). Também é possível destacar estudos sobre que aspectos a refatoração pode ser benéfica segundo a natureza do projeto de *software* (STROGGYLOS; SPINELLIS, 2007) (RATZINGER; FISCHER, 2005).

Alto desempenho e refatoração constituem-se de uma área de pesquisa relativamente recente. Observa-se a publicação de estudos de caso relacionados a técnicas de refatoração para evolução de aplicações ligadas ao alto desempenho (OVERBEY; NEGARA; JOHNSON, 2009) (BEVERLY SANDERS ERIK DEUMENS; PONTON, 2008), utilização de técnicas para melhorar o desempenho de aplicações (MATTHIAS RIEGER BART VAN ROMPAEY; LIEVIER, 2007) e em especial a automatização de técnicas em ferramentas ou *frameworks* específicos (OVERBEY et al., 2005) (JOHNSON et al., 2005) (WATSON; DEBARDELEBEN, 2006).

Em relação a otimização *Unroll and Jam*, na literatura é possível encontrar trabalhos que implementam algoritmos baseados na álgebra linear (CARR; GUAN, 1997), estudos sobre métodos analíticos (ZINGIRIAN; MARESCA, 1999), e algoritmos baseados em grafos (SONG; LIN, 2000) para determinar índices de desenrolamento ideais para os laços. Alem disso, também há estudos sobre aplicações que demonstrem o impacto da otimização *Unroll and Jam* em *software pipelining* (CARR; DING; SWEANY, 1996).

Em relação as ferramentas de refatoração para a linguagem Fortran, é possível destacar o PIPS (F. IRIGOIN; TRIOLET, 1991) para Fotran 77, o Nestor (SILBER; DARTE, 1998) para Fortran 77, 90 e o Photran também para Fortran 77-2008 (BEATON; RIVIERES, 2006). O Photran é uma ferramenta de desenvolvimento baseada na plataforma Eclipse (BEATON; RIVIERES, 2006) pertencente ao macro projeto *Parallel Tools Platform* (PTP), que visa prover ao desenvolvedor um ambiente de desenvolvimento com recursos de programação e refatoração. Por ser código aberto, o Photran permite que diversas pesquisas possam ser elaboradas com o objetivo de implementar novas refatorações para o núcleo do Photran.

2.4 Refatoração em HPC

Na área de HPC a refatoração possui um papel fundamental na obtenção de desempenho através da otimização de aplicações. Estas refatorações exclusivas para a obtenção de desempenho são denominadas de *Performance Refactoring*. Apesar das técnicas de compilação implementarem otimizações no código intermediário (AHO; SETHI; ULLMAN, 1986), estas otimizações manuais realizadas pelo programador são importantes na obtenção de desempenho em aplicações quando executadas para arquiteturas específicas, ou na migração destas otimizações entre diferentes arquiteturas.

As otimizações da *Performance refactoring* são específicas para cada arquitetura e usualmente tornam o código fonte menos legível, pois os blocos de instruções das aplicações devem ser reordenados para alcançar o desempenho desejado (OVERBEY et al., 2005), sendo assim estas otimizações podem estar sujeitas a erros por parte do programador. Por este motivo é importante a utilização de ferramentas que possam automatizar todo o processo de refatoração do código fonte.

Para a obtenção de desempenho por parte das otimizações, os melhores resultados de desempenho são obtidos através da otimização das regiões mais executadas do programa, neste caso os laços de iteração são os mais indicados a otimização (KOSEKI; KOMASTU; FUKAZAWA, 1997). A otimização de laços consiste na reestruturação dos blocos de instruções das iterações, sendo assim para a reestruturação ser válida é necessário que haja mecanismos de detecção de dependências entre as iterações, uma vez que as dependências estarem estabelecidas, a reestruturação pode ser concretizada (WOLFE, 1996).

A reestruturação do programa na refatoração deve acontecer diretamente na *Abstract Syntax Tree* (AST) do código fonte, pois ela fornece toda a estrutura de representação do código fonte do programa. Ela é composta por uma raiz do qual são derivados diversos nós que por sua vez compõem outros nós. O último nível dessa árvore geralmente representa os *tokens* da linguagem de programação. Desta forma, a refatoração consiste na remoção, edição e alocação de novos nós da AST do código fonte a ser refatorado (OVERBEY; JOHNSON, 2009).

Outra área importante no uso da refatoração em HPC é a atualização de código legado de linguagens de alto desempenho. A evolução das linguagens de programação possui um custo elevado, pois a introdução de novas funcionalidades adiciona certos graus de complexidade na linguagem, ao mesmo tempo em que a remoção de funções na linguagem pode tornar obsoletos os programas existentes.

Neste aspecto, a atualização de código legado através de ferramentas integradas de desenvolvimento (OVERBEY; NEGARA; JOHNSON, 2009) torna-se importante para que aplicações legadas possam ser atualizadas. Este é o caso de Fortran, cuja linguagem possui mais do que cinco décadas, sendo utilizada frequentemente em aplicações que demandam alto poder computacional.

O escopo deste trabalho está relacionado a primeira área descrita sobre a otimização de laços de iteração em aplicações Fortran.

2.5 Fortran

Fortran é uma linguagem de programação que permite a criação de programas que primam pela velocidade de execução, sendo importante na utilização em aplicações científicas computacionalmente intensivas. Uma das suas principais vantagens, considerando

a computação de alto desempenho, é a existência de operações para tratamento de dados multidimensionais que normalmente não são encontradas de forma nativa em outras linguagens de programação (KOFFMANN; FRIEDMAN, 2006). Pelo fato de que o domínio de aplicação de Fortran é bastante específico e ligado à pesquisa científica e não ao mercado, há uma lacuna entre a quantidade de código legado escrita em Fortran e ferramentas de refatoração existentes para a linguagem Fortran (BONIATI; CHARAO, 2008).

A principal aplicação de Fortran desde seu advento é a computação científica. Este foco de atuação fez com que a linguagem tenha se perpetuado ao longo do tempo e, mesmo depois de mais de cinquenta anos de sua primeira versão, ainda seja altamente utilizada em aplicações científicas de alto custo computacional.

A linguagem Fortran é fundamental para aplicações e bibliotecas como BLAS (*Basic Linear Algebra Subprograms*), LAPACK (*Linear Algebra Package*), *Benchmark* LINPACK (DONGARRA; LUSZCZEK; PETITET, 2003), IMSL (*International Mathematics and Statistics Library*), NAG (*Numerical Algorithms Group*) e pode ser considerada a linguagem predominante em áreas de aplicação como matemática, física, engenharia e análises científicas (DE, 2004) (KOFFMANN; FRIEDMAN, 2006) (NYHOFF; LEESTMA, 1997).

Fortran também oferece suporte a utilização de bibliotecas para programação concorrente e distribuída como MPI (RASMUSSEN; SQUYRES, 2005) e OpenMP (HERMANNS, 2002). Além disso, é fundamental para aplicações meteorológicas como o BRAMS (FAZENDA et al., 2011) e o OLAM (WALKO; AVISSAR, 2008), ambos mantidos e gerenciados pelo CPTEC (Centro de Previsão de Tempo e Estudos Climáticos) do INPE (Instituto Nacional de Pesquisas Espaciais).

Devido a longevidade da linguagem Fortran, a refatoração em Fortran pode ser vastamente utilizada tanto para atualização de sistemas legados, quanto para melhorar o desempenho dos sistemas a partir da reestruturação do código fonte. Sendo assim, a escolha da linguagem Fortran neste trabalho, é devido ao fato desta linguagem estar ligada diretamente a aplicações que envolvam cálculos multidimensionais implementados com laços de iteração que possam ser otimizados.

3 OTIMIZAÇÃO DE LAÇOS

A otimização de laços possui diversos objetivos como o aumento no aproveitamento da memória cache, a utilização de modo eficiente dos recursos de processamento paralelo disponíveis, e a diminuição do *overhead* gerado durante a execução dos laços (CRAWFORD; WADLEIGH, 2000).

Para as otimizações serem realizadas sem afetar a semântica do código, é necessário analisar as dependências de dados (WOLFE, 1996). Nas seções a seguir, é descrito a execução da análise de dependência e os tipos de otimizações disponíveis na literatura.

3.1 Análise de dependência

A otimização no código é efetuada através da reordenação da ordem de execução das instruções. Para isto, é preciso executar três passos (BACON; GRAHAM; SHARP, 1994):

- Decidir o tipo de otimização e o local onde será aplicada no código.
- Verificar se a transformação não altera a semântica do código.
- Transformar o código.

A detecção de dependências de dados no laço é necessária para verificar se o laço é válido para reordenação. Após estas dependências estarem estabelecidas, é possível realizar a alteração.

A dependência de dados no laço é analisada através do grafo *Iteration Space* (WOLFE, 1996). Cada eixo do grafo representa uma variável de iteração, e cada ponto representa um valor de índice das iterações. Se alguma variável *A* possui uma dependência de outra variável *B*, é gerado uma aresta do ponto de iteração da variável *B* para o ponto de iteração da variável *A*.

Nas figuras a seguir é possível observar a construção do grafo *Iteration Space* para um laço. Na Fig. 3.1 é possível observar a dependência da variável de leitura *b(i)* da instrução *b(j) = b(j) - a(i,j) * b(i)* com a variável de escrita *b(i)* da instrução *b(i) = b(i) / a(i,i)*. Na Fig. 3.2 pode-se observar a dependência da variável de leitura *b(i)* da instrução *b(i) = b(i) / a(i,i)* com a variável de escrita *b(j)* da instrução *b(j) = b(j) - a(i,j) * b(i)*. Na Fig. 3.3 é possível observar a dependência da variável de escrita *b(j)* da instrução *b(j) = b(j) - a(i,j) * b(i)* com a variável de leitura *b(j)* da mesma instrução.

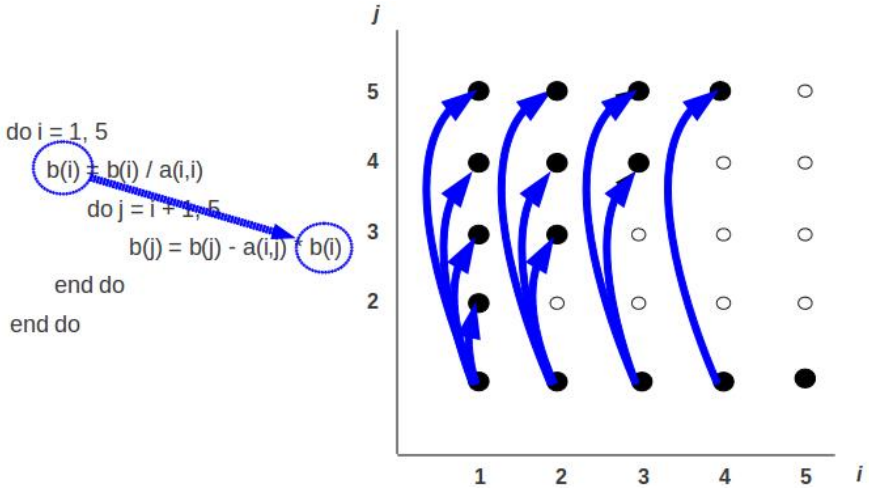


Figura 3.1: *Iteration Space*. Primeira dependência detectada.

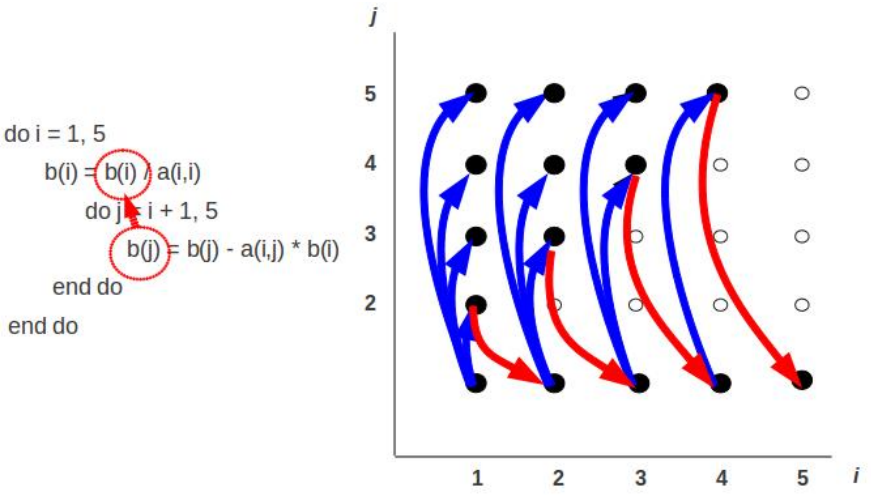


Figura 3.2: *Iteration Space*. Segunda dependência detectada.

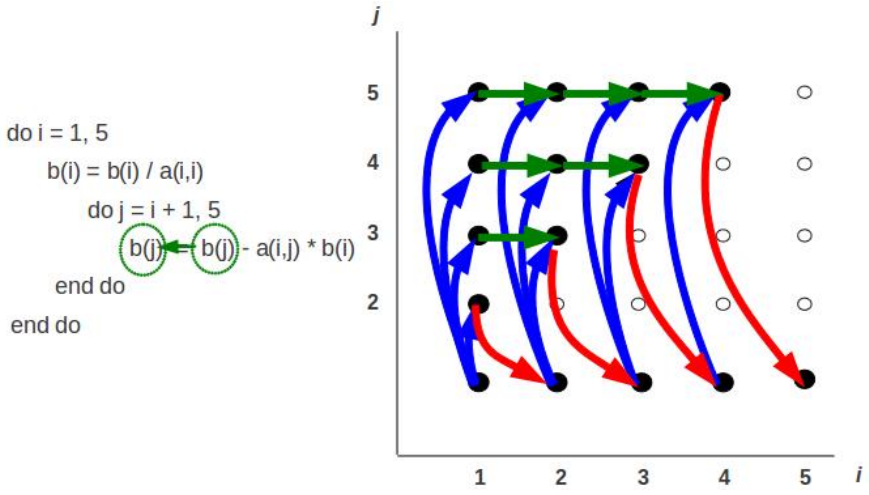


Figura 3.3: *Iteration Space*. Terceira dependência detectada.

3.2 Tipos de Otimizações

A execução dos laços de iteração possui um impacto importante no desempenho de uma aplicação, por este motivo diversos algoritmos de otimização de laços foram desenvolvidos com o objetivo de melhorar o tempo de execução dos laços, através do reuso temporal e espacial de dados e instruções das aplicações. Nas subseções a seguir são descritas as principais otimizações de laços.

3.2.1 Loop Fusion

Esta otimização consiste em transformar diversos laços de iteração em um único laço. No código 3.1 é ilustrado o código original dos laços, e no código 3.2 é possível verificar o laço otimizado, nas quais as instruções 2 e 4 são unificadas em um único laço. Esta otimização somente é válida se ambos os laços tiverem o mesmo número de iterações.

Código 3.1: Laço original

```
1 do i = 0, 100
2       a(i) = 1;
3 enddo
4 do i = 0, 100
5       b(i) = 2;
6 enddo
```

Código 3.2: Loop fusion

```
1 do i = 0, 100
2       a(i) = 1;
3       b(i) = 2;
4 enddo
```

3.2.2 Loop Fission

A otimização *Loop fission* consiste em quebrar um laço de iteração em diversos laços com o mesmo número de iterações do laço original, porém com porções de instruções diferentes do laço original. O objetivo desta otimização é transformar um laço de iteração com várias instruções em diversos laços menores e assim utilizar melhor a localidade de referencia. Esta técnica é vantajosa em arquiteturas que contenham unidades de processamento paralelo que possam executar os diversos laços criados. Esta otimização é valida se as instruções do laço não tiverem dependências entre elas. No código 3.3 é mostrado o código com o laço original, e no código 3.4 são gerados dois laços de iteração nos quais contêm as instruções 2 e 3 do código original.

Código 3.3: Laço original

```
1 do i = 0, 100
2       a(i) = 1;
3       b(i) = 2;
4 enddo
```

Código 3.4: Loop fission

```
1 do i = 0, 100
2       a(i) = 1;
3 enddo
4 do i = 0, 100
```

```
5       b(i) = 2;
6   enddo
```

3.2.3 Loop Unrolling

A otimização *loop unrolling* tem o objetivo de incrementar o *speed-up* da aplicação através da diminuição das instruções que controlam o laço, diminuindo o número de iterações. Esta otimização é considerada válida se a divisão entre o número de iterações e o índice de desenrolamento for inteira. No código 3.5 é ilustrado o código com o laço original, e no código 3.6 o respectivo laço desenrolado (*unrolling*).

Código 3.5: Laço original

```
1   do i = 0, 100
2       insert(i);
3   enddo
```

Código 3.6: Loop unrolling

```
1   do i = 0, 100
2       insert(i);
3       insert(i+1);
4       insert(i+2);
5       insert(i+3);
6       insert(i+4);
7   enddo
```

3.2.4 Loop Invariant Code Motion

A otimização *Loop invariant code motion* consiste em instruções que podem ser movidas para fora do corpo do laço sem afetar a semântica original do laço. Esta otimização é aplicada quando o valor final de uma expressão interna do laço não é alterado durante as iterações, sendo assim, sua remoção do bloco de instruções interno do laço é válida. No código 3.7 é mostrado o código original, e no código 3.8 a otimização é aplicada, onde as instruções 2 e 3 são excluídas do laço, pois o resultado de suas expressões não é alterado durante as iterações.

Código 3.7: Laço original

```
1   do i = 0, n
2       x = y + z
3       a(i) = 6 * i + x * x;
4   enddo
```

Código 3.8: Loop invariant code motion

```
1   x = y + z;
2   t1 = x * x;
3   do i = 0, n
4       a(i) = 6 * i + t1;
5   enddo
```

3.2.5 Loop Unswitching

Esta otimização consiste em mover a expressão condicional interno do laço para fora do laço através da duplicação do corpo do laço, colocando uma versão do laço para cada

cláusula *if* e *else*. Esta otimização pode melhorar a paralelização do laço. No código 3.9 é possível observar o laço original e sua otimização no código 3.10.

Código 3.9: Laço original

```
1  do i = 0, 1000
2          x(i) = x(i) + y(i);
3          if(w){
4                  y(i) = 0;
5          }
6  enddo
```

Código 3.10: Loop unswitching

```
1  if(w){
2          do i = 0, 1000
3                  x(i) = x(i) + y(i);
4                  y(i) = 0;
5          enddo
6  }
7  else{
8          do i = 0, 1000
9                  x(i) = x(i) + y(i);
10         enddo
11 }
```

3.2.6 Loop Interchange

Esta otimização consiste no processo de mudar a ordem de duas variáveis de iteração. Esta otimização possui a finalidade de melhorar a localidade espacial, pois em geral arrays são organizados por linha pelos compiladores. No código 3.11 é ilustrado o código com os laços originais, e no código 3.12 os índices e a condição de parada dos laços são alterados entre eles.

Código 3.11: Laço original

```
1  do i = 0, n
2          do j = 0, m
3                  a(i,j) = i + j;
4          enddo
5  enddo
```

Código 3.12: Loop interchange

```
1  do i = 0, m
2          do j = 0, n
3                  a(i,j) = i + j;
4          enddo
5  enddo
```

3.2.7 Loop Unroll and Jam

Unroll and Jam é uma técnica de otimização de laços aninhados que visa reduzir o número de desvios e operações na memória, aproveitando os registradores disponíveis na arquitetura (CARR; KENNEDY, 1994).

A otimização *Unroll and Jam* consiste em dois passos: desenrolar e fusionar os laços de iteração. Esta otimização é válida se a divisão entre o número de iterações e o índice

de desenrolamento for inteira, e se entre os laços não existam dependências. No presente trabalho não foram consideradas as dependências.

A partir de laços originais (código 3.13), o bloco de instruções do laço mais externo é mantido, e as instruções do laço mais interno são replicadas em tantas vezes quanto o índice de desenrolamento for indicado para este laço (código 3.14). No código 3.15, finalmente as instruções dos laços internos replicados são fusionados em um bloco único de instrução.

Código 3.13: Laço original

```
do k = 0, n
      do j = 0, n
            do i = 0, n
                  a(i,j) = a(i,j) + b(i,k) * c(k,j)
            enddo
      enddo
enddo
```

Código 3.14: Unroll

```
do k = 0, n, 2
      do j = 0, n, 4
            do i = 0, n
                  a(i,j) = a(i,j) + b(i,k) * c(k,j)
                  a(i,j) = a(i,j) + b(i,k+1) * c(k+1,j)
            enddo
            do i = 0, n
                  a(i,j+1) = a(i,j+1) + b(i,k) * c(k,j+1)
                  a(i,j+1) = a(i,j+1) + b(i,k+1) * c(k+1,j+1)
            enddo
            do i = 0, n
                  a(i,j+2) = a(i,j+2) + b(i,k) * c(k,j+2)
                  a(i,j+2) = a(i,j+2) + b(i,k+1) * c(k+1,j+2)
            enddo
            do i = 0, n
                  a(i,j+3) = a(i,j+3) + b(i,k) * c(k,j+3)
                  a(i,j+3) = a(i,j+3) + b(i,k+1) * c(k+1,j+3)
            enddo
      enddo
enddo
```

Código 3.15: Jam

```
do k = 0, n, 2
      do j = 0, n, 4
            do i = 0, n
                  a(i,j) = a(i,j) + b(i,k) * c(k,j)
                  a(i,j) = a(i,j) + b(i,k+1) * c(k+1,j)
                  a(i,j+1) = a(i,j+1) + b(i,k) * c(k,j+1)
                  a(i,j+1) = a(i,j+1) + b(i,k+1) * c(k+1,j+1)
                  a(i,j+2) = a(i,j+2) + b(i,k) * c(k,j+2)
                  a(i,j+2) = a(i,j+2) + b(i,k+1) * c(k+1,j+2)
                  a(i,j+3) = a(i,j+3) + b(i,k) * c(k,j+3)
                  a(i,j+3) = a(i,j+3) + b(i,k+1) * c(k+1,j+3)
            enddo
      enddo
enddo
```

A otimização *Unroll and Jam* diminui o número de incrementos e testes realizados nos laços de iteração. Para isto, o índice de desenrolamento está ligado diretamente aos recursos de *hardware* disponíveis, ou seja, a obtenção de desempenho pode ser comprometida caso os laços desenrolados necessitem de um maior número de registradores disponíveis (CRAWFORD; WADLEIGH, 2000).

4 FERRAMENTAS PARA OTIMIZAÇÃO

As otimizações nas aplicações podem ocorrer no código intermediário através de compiladores, ou no código fonte por meio de ferramentas de desenvolvimento. Neste capítulo são descritos as características e diferenças entre ambas através do compilador GCC e da ferramenta Photran.

4.1 GNU Compiler Collection

O *GNU Compiler Collection* (GCC) é um conjunto de compiladores de linguagens de programação produzido pelo projeto GNU. Ele é caracterizado por ser de caráter software livre distribuído pela Free Software Foundation (FSF) sob os termos da GNU GPL, e por ser uma componente chave do conjunto de ferramentas GNU. O GCC pode ser considerado o compilador padrão para sistemas operativos UNIX, Linux, e Mac OS (VICHARE, 2008), e representa uma peça fundamental no desenvolvimento de aplicações científicas de caráter software livre (JONES, 2008).

4.1.1 Breve Histórico

O compilador GCC foi escrito por Richard Stallman em 1987 com o objetivo de servir de compilador para o Projeto GNU, a criação do GCC foi um marco importante, pois foi o primeiro compilador software livre otimizado para linguagem C, desde então o GCC tornou-se uma importante ferramenta no desenvolvimento de software (GOUGH; STALLMAN, 2004).

Logo após em 1992, um versão mais robusta veio com a série 2.0, nesta versão surgiu a possibilidade de compilação C++. No ano de 1997, com a idéia de juntar várias bifurcações experimentais num único projeto e assim aumentar no compilador o processo de otimização e suporte ao C++, surgiu um novo projeto chamado de EGCS. Desde então em Abril de 1999, o projeto ECGS foi adotado como versão inicial do GCC (GOUGH; STALLMAN, 2004).

Na versão 3.0 surgida em 2001, foram obtidas melhorias nas otimizações e suporte ao C++. Atualmente o GCC é estendido para o suporte a outras linguagens, incluindo Fortran, ADA, Java e Objective-C, e seu desenvolvimento é guiado pelo GCC Steering Committee, um grupo composto por representantes de comunidades de usuários do GCC presentes na indústria, pesquisa, e academia (GOUGH; STALLMAN, 2004).

4.1.2 Arquitetura

O GCC é essencialmente um pipeline que converte uma representação de um programa em outra, nele há três componentes principais: front-end, middle-end, e back-end.

O front-end é responsável por ler o código fonte, analisá-lo, e convertê-lo em uma árvore sintática abstrata (AST), neste aspecto, existe um front-end e uma AST específica para cada linguagem. Após isto, a árvore sintática abstrata é convertida em uma árvore chamada Generic, cuja representação é uniforme para todas as linguagens. As árvores Generic são utilizadas para ser uma representação intermediária entre o *parser* e o código (NOVILLO, 2006).

No middle-end a árvore Generic é convertida em uma nova representação chamada de Gimple. Neste formato cada expressão não contém mais do que três operandos, o que facilita os processos de otimização do código fonte. No código 4.1 é possível observar um código no formato Generic, e no código 4.2 no formato Gimple.

Código 4.1: Código no formato Generic

```
a = foo ();
b = a + 10;
c = 5;
if (a > b + c)
      c = b++ / a + (b * a);
bar (a, b, c);
```

Código 4.2: Código no formato Gimple

```
a = foo ();
b = a + 10;
c = 5;
T1 = b + c;
if (a > T1){
      T2 = b / a;
      T3 = b * a;
      c = T2 + T3;
      b = b + 1;
}
bar (a, b, c);
```

Logo após a geração do Gimple, esta é convertida em uma representação SSA (*static single assignment*). O formato SSA permite que uma determinada variável possa ser atribuída uma única vez, porém a mesma variável pode ser utilizada no lado direito da expressão inúmeras vezes, caso uma variável seja atribuída varias vezes, é criada uma nova versão desta variável e armazenada o novo valor nela (NOVILLO, 2006). No código 4.3 é possível observar um código no formato SSA.

Código 4.3: Código no formato SSA

```
baz ()
{
      int i, j;
      {
            int k;
            k = foo ();
            i = k + 2;
            j = i * k;
      }
      return j;
}
```

O formato SSA é alvo de otimizações (opt1, opt2, opt3) por parte do GCC, e convertido na representação intermediária (RI) chamada de *Register Transfer Language* (RTL).

A RTL é uma representação próxima a linguagem Assembly que corresponde a arquitetura alvo, esta representação também pode ser alvo de otimizações (NOVILLO, 2006).

Finalmente o back-end do GCC gera o código Assembly para a arquitetura alvo utilizando a representação RTL. Todos os componentes da arquitetura do GCC podem ser vistos na Figura 4.1 (NOVILLO, 2006).

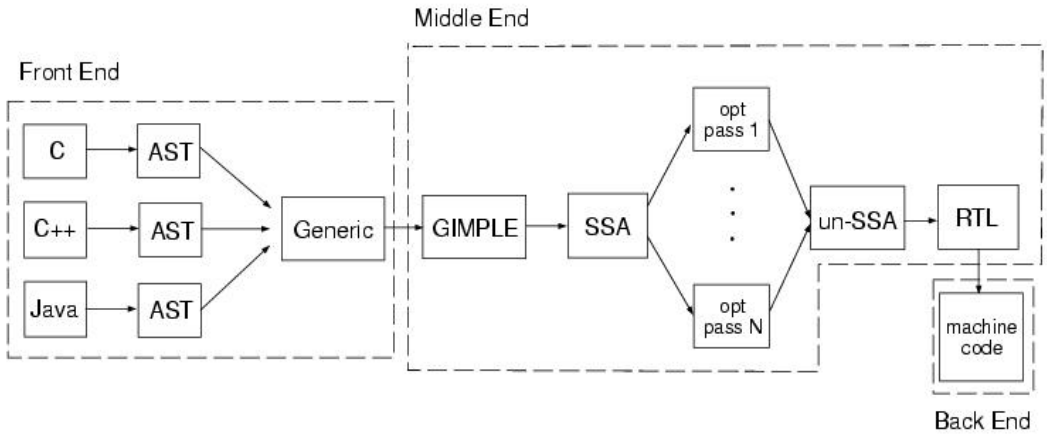


Figura 4.1: Arquitetura do GCC.

No compilador GCC existem três blocos básicos que compõem a infraestrutura de otimização: o Gimplifier, *Control Flow Graph* (CFG), e o módulo SSA. O Gimplifier é responsável por prover funções que convertem as árvores Generic no formato Gimple (NOVILLO, 2006). Os motivos da utilização do Gimple se devem por três aspectos:

- Facilitar a implementação de novas análises e otimizações próximas a representação do código fonte.
- Simplificar o trabalho realizado pelos otimizadores da RTL, melhorando o código fonte e desempenho dos programas compilados.
- Otimizar aplicações independentes da linguagem do código e da arquitetura alvo, isto é importante para o GCC visto que ele comporta diferentes linguagens.

As versões anteriores do GCC operavam otimizações na representação intermediaria Register Transfer Language (RTL). As árvores geradas pelo front-end eram imediatamente convertidas para a RTL e otimizadas (Figura 4.2). A RTL é uma boa estrutura para otimizações em baixo-nível, porém muitas otimizações necessitam informações de mais alto nível sobre o programa, o que é muito difícil de obtê-las através da RTL (NOVILLO, 2006).

O *Control Flow Graph* (CFG) é uma estrutura de dados construída juntamente com a árvore Gimple, o CFG é um grafo direcionado que modela a execução do programa, no qual cada bloco básico representa instruções que devem ser executadas sequencialmente, e as arestas do grafo representam possíveis caminhos de execução no fluxo de controle (NOVILLO, 2006).

O módulo SSA é o *framework* de otimização que opera sobre as árvores Gimple (Figura 4.3) gerando árvores SSA. O formato SSA é projetado para ser independente de linguagem e arquitetura, e permite análises e transformações de alto nível que são difíceis de implementar utilizando a RTL (NOVILLO, 2006). Neste aspecto, um dos objetivos

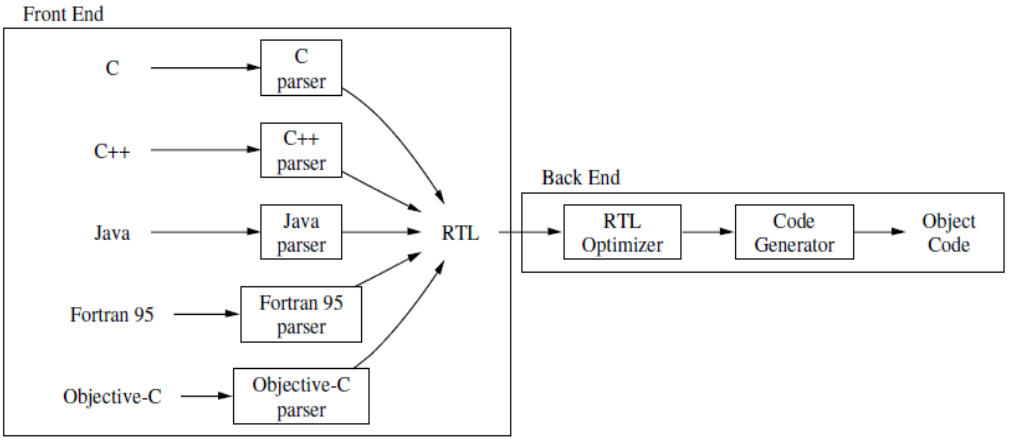


Figura 4.2: Otimização no RTL.

da utilização de árvores SSA é produzir para o GCC uma infraestrutura compacta para análises e algoritmos de otimização disponíveis na literatura.

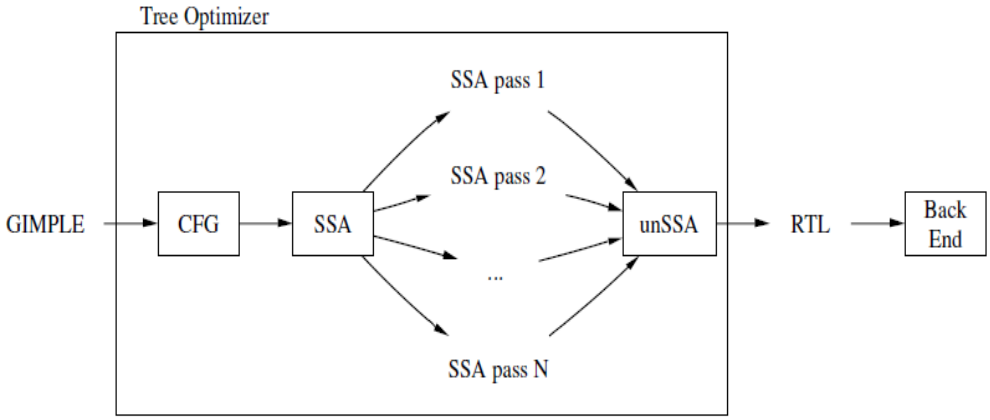


Figura 4.3: Otimização na árvore SSA.

Além disso, a modularização das otimizações a partir da árvore SSA pode melhorar o desempenho do GCC e facilitar o aprendizado por parte dos desenvolvedores externos ao GCC que desejem adicionar novas otimizações. Outro ponto fundamental na utilização da árvore SSA, é a redução do código RTL gerado, e do tempo de compilação do programa, melhorando a qualidade de código final obtido (NOVILLO, 2006).

4.1.3 Representação e Otimização de Laços

Um laço de iteração possui um bloco de entrada (cabeçalho) e diversas arestas que dirigem-se do cabeçalho para os blocos de instruções do respectivo laço. No GCC o *Control Flow Graph* é responsável por obter as informações necessárias dos laços, e organizá-los em uma estrutura de hierarquia de árvore (NOVILLO, 2006).

Todos os laços são representados por uma estrutura chamada de *struct loops*, onde para cada laço é atribuído um valor de índice, e um ponteiro para o laço é armazenado em um vetor de laços chamado de *larray* contido na *struct loops* (STALLMAN, 2011).

Cada laço interno contém a referência para o laço mais externo do qual ele pertence, por esta razão, somente é possível ter ao mesmo tempo uma estrutura do tipo *struct lo-*

ops inicializada no CFG. A variável global *current-loops* é responsável por referenciar a estrutura *struct loops* (STALLMAN, 2011).

Os acessos aos laços através dos índices não devem ser realizados diretamente do *larray*, deve-se usar a função *get-loop* para retornar o laço referenciado. Outra função chamada de *number-of-loops* retorna o número de laços em uma determinada função (STALLMAN, 2011).

Para realizar a iteração sobre os laços, é utilizada a macro *FOR-EACH-LOOP*, onde as *flags* da macro indicam a direção que a iteração deve realizar, e quais laços devem ser percorridos. O GCC garante a visita nos laços pelo menos uma vez através da macro *FOR-EACH-LOOP*, alocando variáveis temporárias, porém caso ela seja interrompida por uma instrução *break* ou *goto*, as variáveis não serão desalocadas, sendo necessário a utilização da macro *FOR-EACH-LOOP-BREAK* para desalocá-las. Durante a iteração, os laços podem ser adicionados, alterados, ou removidos, porém as mudanças não afetam o processo inicial de iteração (STALLMAN, 2011).

Uma vez descrito como os laços são representados, é necessário verificar como o GCC realiza a análise dos laços no CFG. A inicialização da análise dos laços de iteração é realizada através da função *loop-optimizer-init(flags)*, onde as informações sobre os laços são acessadas através da variável *current-loops* encontrado no arquivo *loop-init.c* (STALLMAN, 2011).

É importante salientar que tais informações dos laços de iteração são preservadas através das otimizações e atualizadas quando a otimização é finalizada. O argumento *flags* desta função permite que um conjunto de propriedades das estruturas dos laços seja calculado ou preservado. Uma vez finalizada a análise, os dados armazenados durante a análise dos laços devem liberados da memória através da função *loop-optimizer-finalize* (STALLMAN, 2011).

Algumas informações de mais alto nível sobre os laços podem ser obtidas diretamente das estruturas dos laços. Alguns campos da *struct loop* são utilizados para obter informações (STALLMAN, 2011):

- header: referência ao cabeçalho do laço.
- num-nodes: indica número de nós do laço (incluindo os blocos básicos dos laços internos).
- depth: indica a profundidade do laço na árvore.
- outer, inner, next: referência ao laço mais externo, interno, e o próximo laço na árvore.

As otimizações de laços no GCC utilizam as informações dos campos da *struct loops* citados acima através das seguintes funções declaradas no arquivo *cfgloop.h* e utilizadas no arquivo *cfgloop.c* (STALLMAN, 2011):

- verify-loop-structure: verifica a consistência do laço.
- flow-loop-nested-p: testa se um determinado laço é interno a outro laço.
- flow-bb-inside-loop-p: verifica se um bloco básico de instruções pertence a um determinado laço.
- get-loop-body: retorna um determinado bloco de instruções no laço.

- single-exit: verifica se o laço finaliza normalmente.

Finalmente a manipulação dos laços é realizada através das seguintes funções (STALLMAN, 2011):

- flow-loop-tree-node-add: adiciona um nó na árvore.
- flow-loop-tree-node-remove: remove um nó da árvore.
- add-bb-to-loop: adiciona um bloco de instrução na árvore.
- remove-bb-from-loops: remove um bloco de instrução da árvore.

As otimizações no GCC utilizam todas as estruturas de dados descritas anteriormente, o arquivo onde se encontram as funções principais de otimização pode ser encontrado em *tree-ssa-loop.c* e as otimizações são descritas através de *pass-optimizationName*, as funções que manipulam os laços podem ser encontradas nos arquivos *tree-ssa-loop-manip.c, cfgloop.c, cfgloopanal.c e cfgloopmain.c*.

Como dito anteriormente, as otimizações no GCC podem ocorrer no RTL e na árvore SSA. Em relação à otimização *Loop Unroll*, a partir da versão do GCC 4.2 esta otimização pode ser implementada tanto no formato SSA no arquivo *tree-ssa-ivcanon.c* através de *pass-complete-unroll*, e na RTL no arquivo *loop-unroll.c*.

No caso da otimização *Loop Unroll* o otimizador implementado no RTL não reconhece o *Control Flow Graph* e a estrutura *struct loops*. Sendo assim quando o otimizador em nível RTL é executado, as informações contidas no CFG são desconsideradas, e após o otimizador ter concluído a transformação, o CFG deve ser reconstruído ocasionando perda de desempenho durante a otimização.

Sendo assim, pesquisas tem se concentrado em desenvolver otimizações no nível das árvores SSA, eliminando otimizações possíveis no RTL, pois desse modo é possível aproveitar as informações de alto nível do CFG, reduzir o código em nível RTL, e facilitar a inserção de novas otimizações de laços.

4.2 Photran

O Photran é um ambiente integrado de desenvolvimento (IDE) para Fortran 77-2008 baseado na plataforma Eclipse (BEATON; RIVIERES, 2006), o qual organiza em torno do IDE do Eclipse ferramentas para desenvolvimento, depuração, distribuição e monitoramento de aplicações.

O Photran pertence ao macro projeto denominado *Parallel Tools Platform* (PTP), cujo objetivo é explorar a deficiência de ferramentas para a computação científica (WATSON; DEBARDELEBEN, 2006) e de alto desempenho (VAN DE VANTER; POST; ZOSEL, 2005).

4.2.1 Breve Histórico

Os primeiros trabalhos desenvolvidos em relação ao Photran foram realizados por Vaishali De, Julia Dtragan-Chirila e Rohit Eipe no SAG (*Software Architectute Group*), coordenados por Ralph Johnson na Universidade de Illinois em Urbana Champaign (CHEN N., 2010).

O desenvolvimento do Photran foi realizado através da linguagem Java, e sua arquitetura pode ser composta por subprojetos em que cada um constitui-se de plugins ou

recursos. Os plugins visam adicionar funcionalidades ao Eclipse e os recursos oferecem unidades de desenvolvimento para os programadores (CHEN N., 2010).

4.2.2 Recursos

A edição de código no Photran permite o suporte de edição em formato livre através da classe *FreeFormFortranEditor*, e em formato fixo pela classe *FixedFormFortranEditor* (CHEN N., 2010).

Os processos e assistentes para criação, edição de projetos Fortran são similares aos processos necessários para criação de projetos em outras linguagens no Eclipse. Porém, uma funcionalidade destacada durante a edição do código fonte, é a possibilidade de visualizar através do recurso *Outline View* a estrutura hierárquica da AST do código fonte. Na Figura 4.4 pode ser observado um exemplo de visualização da AST do Photran no *Outline View*.

Figura 4.4: AST no *Outline View* do Photran.

A verificação de erros sintáticos é possível ser detectada através da AST exibida no *Outline View* do Eclipse. Também é possível destacar que ao selecionar algum elemento da *Outline View*, o cursor se posiciona sobre o elemento mais significativo do código fonte.

Além dos recursos citados anteriormente, um dos recursos mais importantes no ambiente Photran é a possibilidade de refatoração do código fonte. O mecanismo de refatoração no Photran permite alterar o código fonte em alto nível, visualizar as diferenças antes e depois da refatoração, e poder cancelar as refatorações inseridas no código fonte. O escopo deste trabalho aborda a utilização deste recurso.

4.2.3 Arquitetura

A arquitetura do Photran é baseada e adaptada a partir de outro plugin para o Eclipse, o CDT (*C, C++ Development Tools*) (LEE B., 2004) (GRAF; ZGRAGGEN; SOMMERLAD, 2007). O CDT é um projeto de código aberto implementado na linguagem Java baseado em um conjunto de plugins para a plataforma do Eclipse SDK, este plugin adiciona uma perspectiva C/C++ a área de trabalho do Eclipse permitindo a edição, compila-

ção, e debugação de códigos em C/C++. Na Figura 4.5 é possível observar a interface de programação do CDT (CHEN N., 2010).

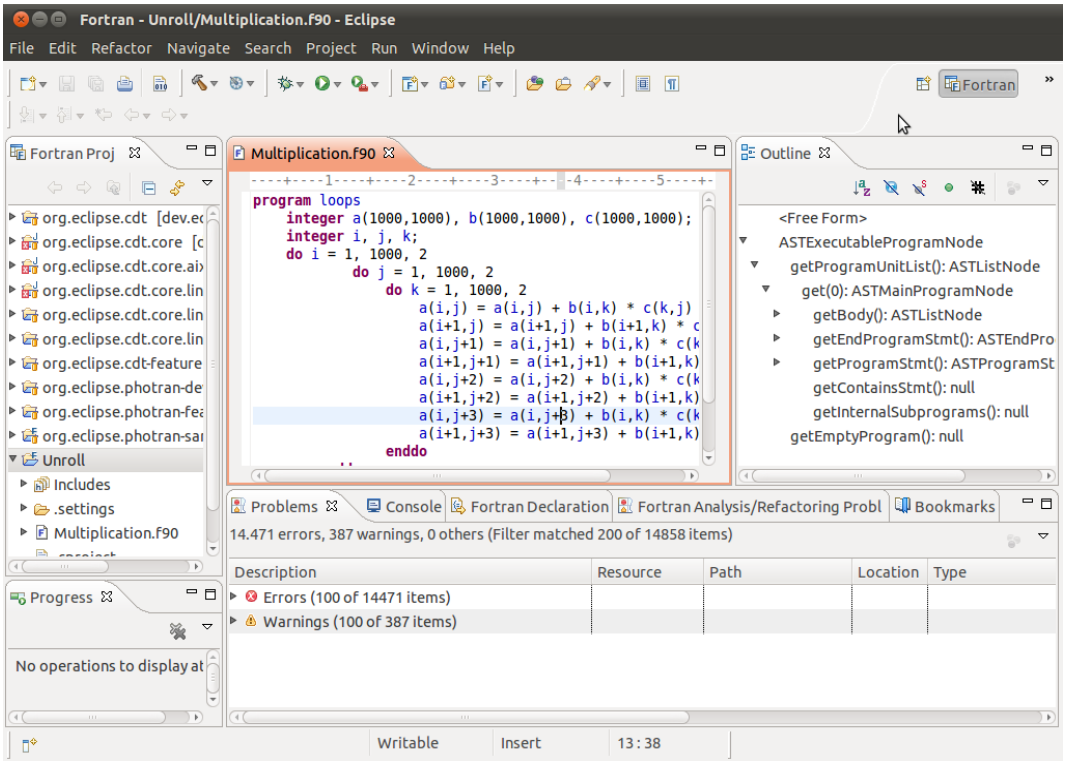


Figura 4.5: Interface Eclipse/CDT.

O Photran é construído sobre o CDT, adicionando extensões do pacote *org.eclipse.cdt.core.language* para validar o reconhecimento da linguagem Fortran no CDT, e permitir que o Photran possa emitir estruturas visuais de saída sobre o código fonte (CHEN N., 2010).

Sobre a interface gráfica, elementos da interface do CDT são estendidos para fornecer ao Photran uma IDE similar ao CDT. O Photran também contribui com *error parsers* que permitem ao CDT reconhecer mensagens de erros de diversos compiladores Fortran e exibi-los no console de erros. Além disso, o Photran adiciona novos projetos de *template* e ferramentas para o projeto CDT e para o próprio Photran (CHEN N., 2010).

Em relação aos diversos componentes/projetos do Photran, estes podem ser decompostos do seguinte modo (CHEN N., 2010):

- *Rephraser Engine*: Este projeto representa uma infraestrutura independente de linguagem, ela contém as classes bases para as classes da refatoração do Photran, as quais são herdadas das classes do *Rephraser Engine*. A idéia básica desta infraestrutura é criar um mecanismo comum que possa ser reutilizado em ferramentas de refatoração para diferentes linguagens. Atualmente há pesquisas realizadas pelo professor Ralph Johnson para desenvolvimento de ferramentas de refatoração para PHP, Lua, BC, que utilizam o *Rephraser Engine* como estrutura base para a refatoração.

- *Base Photran*: Dentro deste projeto está o pacote *org.eclipse.photran.cdtinterface* que contém a maioria dos componentes (*core* e UI) relacionados com a integração com o CDT. O pacote *org.eclipse.photran.core* é considerado o núcleo do Photran,

pois contem as funcionalidades que permitem que o eclipse suporte projetos em Fortran. Outro pacote fundamental neste projeto, é o *org.eclipse.photran. ui* que possui os componentes específicos do Fortran da interface do usuário que não são derivados do CDT e que não dependem do *parser* do Fortran. Finalmente os pacotes *org.eclipse.photran.managedbuilder.core*, *org.eclipse.photran.managedbuilder.gnu .ui*, *orc.eclipse.photran.managedbuilder.ui* oferecem suportes para projetos que utilizem o GNU Fortran (gfortran).

- *Virtual Program Graph*: Neste projeto está o pacote *org.eclipse.photran.core.vpg* que contém a infraestrutura para análise, *parsing* (neste pacote encontra-se a AST), e o pacote org.eclipse.photran.internal.core.refactoring que contém a refatoração. Os pacotes *org.eclipse.photran.core.vpg.tests* e *org.eclipse.photran.core. vpg.tests. failing* fornecem testes para o núcleo VPG. Também neste projeto encontra-se o pacote *org.eclipse.photran.ui.vpg* onde são implementadas as interfaces gráficas das refatorações. Outro pacote encontrado é o *org.eclipse.photran.cdtinterface. vpg*, o qual constrói as visualizações do código apresentadas no *Outline View*.

- *XL Fortran Compiler*: Neste projeto, os pacotes *org.eclipse.photran.core. errorparsers.xlf* e *orc.eclipse.photran.managedbuilder.xlf.ui* fornecem funcionalidades para suporte ao compilador XL Fortran.

- *Intel Fortran Compiler*: Este projeto dispõe suporte para o compilador Intel Fortran. Os pacotes encontrados *org.eclipse.photran.core.intel* e *org.eclipse. photran. managedbuilder.intel.ui* são os responsáveis por este suporte.

4.2.4 Abstract Syntax Tree

A *Abstract Syntax Tree* é a representação interna do código fonte gerada diretamente pelo analisador sintático utilizado. No caso do Photran o analisador sintático foi produzido pelo Ludwig, o qual possui a característica de produzir analisadores sintáticos e ASTs que possam ser manipuladas (*Rewritable* AST), ou seja, cujos nós possam removidos, inseridos, ou realocados de lugar, permitindo modificar o código fonte e manter o formato original do código não alterado. O Ludwig utiliza a notação EBNF para a especificação da gramática e produz o analisador sintático em código Java, no caso do Photran a gramática está contida no arquivo *fortran2003.bnf* (CHEN N., 2010).

A AST é peça chave para a automatização de ações de refatoração (OVERBEY; JOHNSON, 2009), pois possibilita ao desenvolvedor navegar pela estrutura do código fonte, detectando correlações entre seus nós e possibilitando as inserções, exclusões e posicionamentos destes nós. A Fig 4.6 ilustra na esquerda um laço em Fortran e na direita a sua correspondente AST, nesta figura é possível verificar que para cada *token* ou expressão no código fonte, existe o seu nó correspondente na AST.

Mais especificamente sobre a estrutura da AST, existe um *ASTNode* para cada elemento não-terminal e um método de acesso para cada elemento da direita da produção. No exemplo a seguir é ilustrada uma especificação:

<DataStmtSet> ::= <DataStmtObjectList> | <DataStmtValueList>

A convenção adotada no Photran é gerar para o não-terminal uma classe com o nome *AST<nonterminal name>Node* que estende a classe *ASTNode*. Dessa forma na especificação acima a classe gerada é *ASTDataStmtSetNode*, e métodos *setters* e *getters* são gerados para os elementos *DataStmtObjectList* e *DataStmtValueList (getDataStmtObjectList, setDataStmtObjectList, getDataStmtValueList, setDataStmtValueList)* (CHEN N., 2010).

Figura 4.6: *Abstract Syntax Tree*.

Em relação aos *tokens* da AST, estes formam as folhas da AST e mantém informações como (CHEN N., 2010):

- O elemento terminal do *token* na gramática que pode ser acessado pelo método *getTerminal()*.
- O texto do *token* que pode ser coletado pelo método *getText()*.
- A linha, coluna, *offset*, e tamanho do *token* no código fonte, os quais podem ser obtidos através dos métodos *getLine(), getCol(), getFileOffset(), getLength()*.

4.2.5 Virtual Program Graph

No Photran também existe a estrutura *Virtual Program Graph* (VPG), responsável por adicionar informações de análise para a *Abstract Syntax Tree* do programa. De modo geral, as análises a serem realizadas no código devem ser realizadas através de métodos do VPG.

Um objeto *singleton* do *Virtual Program Graph* é responsável pela criação da respectiva AST do código do programa em Fortran. As ASTs são geradas de duas maneiras: invocando ASTs permanentes, onde a árvore é armazenada na memória até que um método é chamado para eliminar a respectiva AST, ou ASTs transitórias, onde a árvore pode ser eliminada através do *garbage collector* assim que os nós desapareçam (CHEN N., 2010).

Ambos os métodos construtores da AST retornam um objeto *IFortranAST*, o qual possui métodos para acessar os nós da AST, obter informações de escopo da AST, visualizar logs de erros durante a análise. Na Figura 4.7 é possível verificar as linhas pontilhadas que indicam informações do VPG na AST (CHEN N., 2010).

Em relação aos diferentes modos de aplicação das otimizações descritas anteriormente, o presente trabalho foca na implementação de otimizações no código fonte da aplicação, pois através da refatoração é possível obter um controle mais alto nível na otimização e tornar o código já otimizado apto para futuras paralelizações e otimizações.

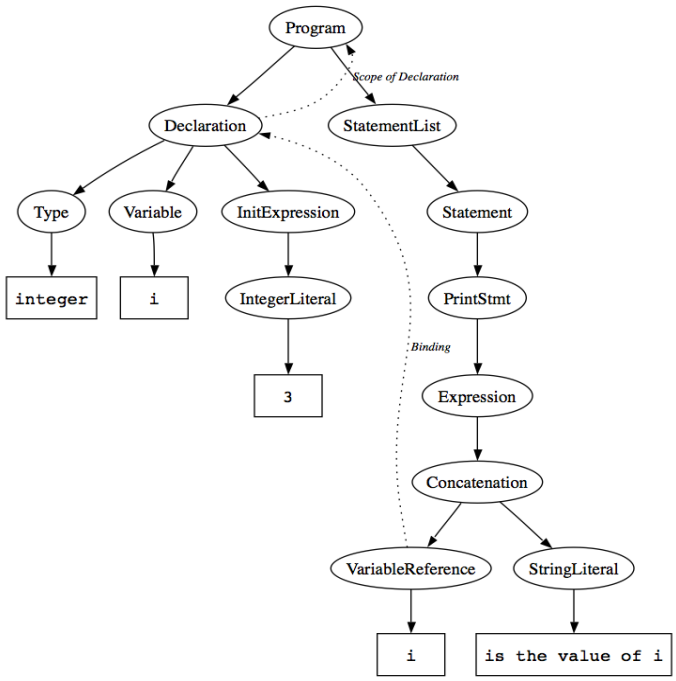


Figura 4.7: *Virtual Program Graph*.

Outro aspecto importante, é a eficiência na portabilidade das otimizações aplicadas no código entre diferentes arquiteturas, pois se uma aplicação for otimizada em uma arquitetura diferente, somente é necessário refatorar o código.

Em relação ao ponto de vista do programador, este pode usufruir dos benefícios que o ambiente de desenvolvimento do Photran fornece ao próprio programador, pois através desta ferramenta é possível desenvolver, refatorar e debugar aplicações paralelas na mesma ferramenta.

5 IMPLEMENTAÇÃO

Esta capítulo é responsável por descrever a implementação realizada neste trabalho, sendo implementada através da linguagem Java devido a integração das refatorações ao núcleo do Photran 7.0. Nas seções a seguir são relatados os diversos atributos, métodos e classes que formam a base da implementação.

A refatoração de *Unroll and Jam* no Photran foi realizada dentro do pacote *org.eclipse.photran.internal.core.refactoring*. A sua implementação através do Photran é realizada em três diferentes etapas:

- Pré-validação.
- Manipulação.
- Pós-validação.

Na fase da pré-validação, o Photran utiliza métodos que permitem modificar a estrutura do laço para deixá-la apta para alteração. Também nesta fase, é verificado se os índices dos laços de iteração podem ser alterados.

Na etapa de manipulação, através de métodos pertencentes ao *Virtual Program Graph* é possível navegar sobre os diferentes nós da *Abstract Syntax Tree*, coletar as informações acerca dos nós que compõem a AST e alterá-los.

Finalmente na etapa de pós-validação, após a reestruturação estar finalizada, métodos são utilizados para reindentar e atualizar toda a AST. Na Figura 5.1 através da UML é possível observar o diagrama de atividades do algoritmo de refatoração *Unroll and Jam*, e nas seções a seguir são relatadas mais especificamente as três fases destacadas anteriormente.

5.1 Pré-validação

Os laços de iteração no *Virtual Program Graph* são implementados em um pacote chamado *org.eclipse.photran.internal.core.analysis*, cujo objetivo é fornecer uma representação adequada aos laços para que estes estejam aptos para visualização e alteração dos nós na AST.

Neste pacote, o método *replaceAllLoops(ast)* transforma os nós *ASTNode* dos laços em nós *ASTProperLoopConstructNode*. A classe *ASTProperLoopConstructNode* fornece diversos atributos para a identificação das partes dos laços e métodos que percorrem as estruturas dos laços:

- ASTLabelDoStmtNode: nó cabeçalho do laço.

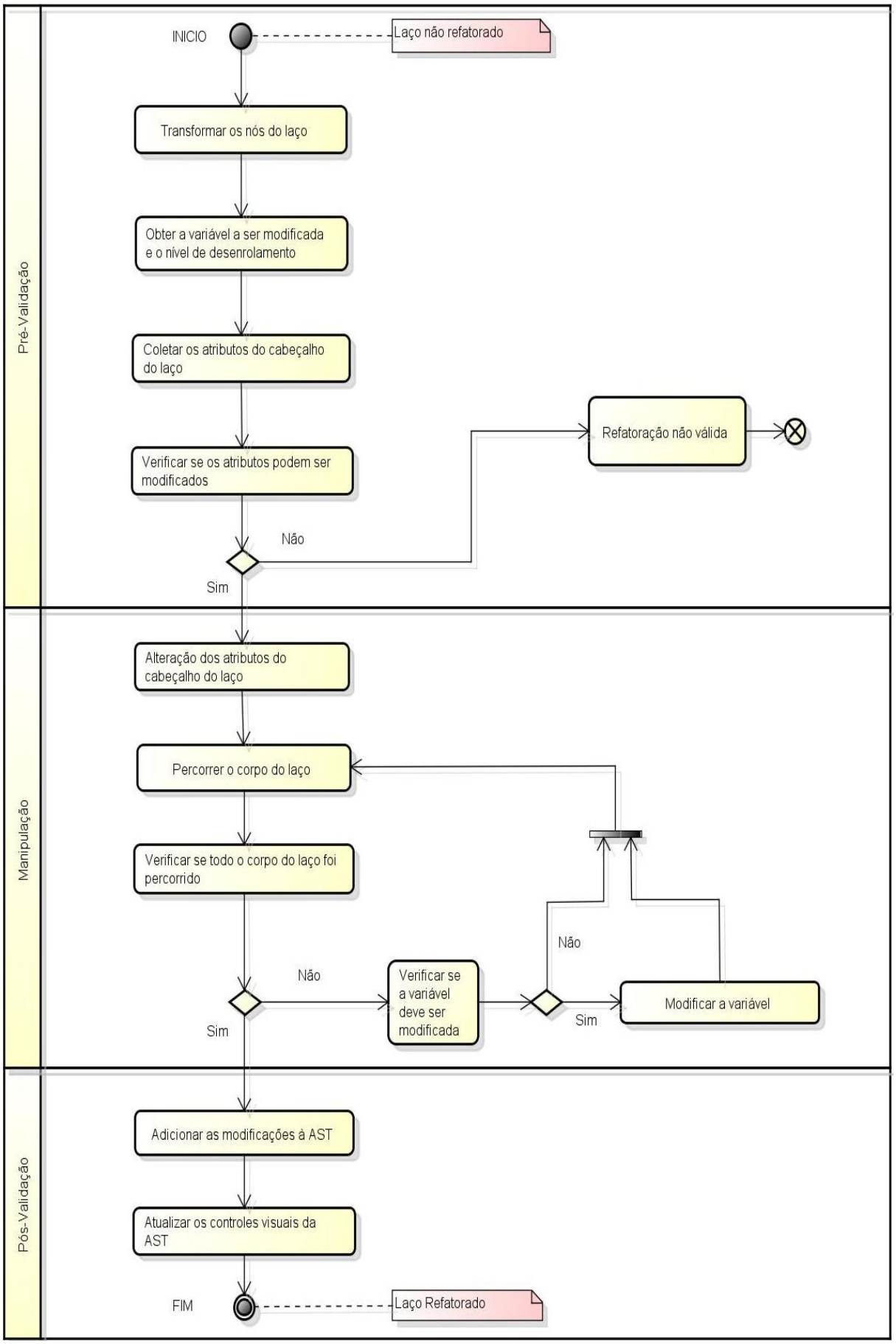


Figura 5.1: Diagrama de atividades do algoritmo de refatoração *Unroll and Jam*.

- IASTListNodeExecutionPartConstruct: nó corpo do laço constituído por uma lista de nós.

- ASTEndDoStmtNode: nó finalizador do laço.

O nó raiz do laço de iteração é obtido através do método *getLoopNode()* e os índices dos cabeçalhos dos laços são coletados através do método *getLoopHeader()*. Para cada índice é utilizado o método *getLoopControl().getUb().findFirstToken().getText()* para obter o valor do índice e verificar através de *checkHeaders()* se o valor do índice do laço de iteração pode ser modificado. O método *getFactor()* é responsável por obter o índice de desenrolamento desejado na refatoração.

5.2 Manipulação

Após as etapas da fase de pré-validação estarem finalizadas, se a validação da alteração do índice de iteração for positiva, então o método *getLoopControl().getUb().findFirstToken().setText()* é utilizado para armazenar o novo valor do índice de iteração. Posteriormente, através do método *modifyTheBody()* é realizado a alteração das variáveis que devem ser refatoradas no bloco de instruções do laço.

Dentro do método *modifyTheBody()*, o corpo do laço é obtido através do método *getBody()* e o método *accept(new GenericASTVisitorWithLoops())* é utilizado para percorrer os *tokens* do corpo do laço e identificar as variáveis que devem ser modificadas de acordo com o índice de desenrolamento obtido na fase de pré-validação. No algoritmo 5.1 abaixo é possível verificar como é realizado as mudanças das variáveis:

Código 5.1: Algoritmo de alteração dos *tokens*

```
1  for(i = 0; i < uf1; i++){
2        oldBody.accept(new GenericASTVisitorWithLoops(){
3              @Override public void visitToken(Token token){
4                    sb.append(SourcePrinter.getSourceCodeFromASTNode(token));
5                    if(i != 0)
6                          if(token.getText().contentEquals(sb1))
7                                sb.append(" + "+i);
8              }
9        });
10       sb.append("\n");
11 }
```

Neste trecho de código é possível observar que o método *getSourceCodeFromASTNode()* é utilizado para obter o token, e o método *append()* é utilizado para armazenar o novo valor do token.

5.3 Pós-validação

Ao término da refatoração, o método *addChangeFromModifiedAST()* é utilizado para adicionar as modificações realizadas na AST utilizada pelo Photran, e o método *releaseAllASTs()* é executado para forçar o Photran a atualizar os controles visuais da AST assim como sua própria validação.

5.4 Arquitetura

A implementação e integração de ações de refatoração no Photran necessita estender comportamentos de três classes (CHEN et al., 2008), na Figura 5.2 é possível observar o diagrama de classes da ação de refatoração *Unroll and Jam*.

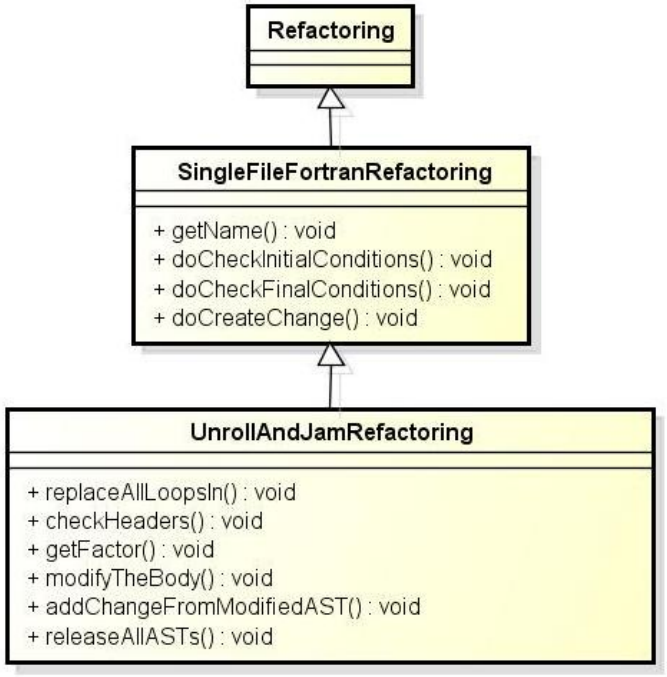


Figura 5.2: Diagrama de classes da refatoração *Unroll and Jam*.

A primeira classe responsabiliza-se por receber a chamada do usuário e associar a ação de refatoração com seu respectivo assistente. Essa classe deve estender a classe *AbstractFortranRefactoringActionDelegate* e deve implementar os métodos de duas interfaces: *IWorkbenchWindowActionDelegate*, a qual permite que a chamada do usuário seja feita a partir do menu principal do Eclipse, e a *IEditorActionDelegate* que permite que a chamada do usuário seja feita a partir de um menu de contexto no próprio editor.

A segunda classe a ser estendida é a classe assistente da refatoração, cujo objetivo tem de solicitar alguma ação do usuário sobre alguma informação adicional para que possa ser executada na refatoração. Essa classe é estendida de *AbstractFortranRefactoringWizard* e basicamente se responsabiliza pela construção gráfica da janela do assistente. O método construtor desta classe recebe como parâmetro um objeto para a terceira classe necessária: a ação de refatoração.

A terceira e principal classe a ser codificada é a ação da refatoração. Essa classe estende a superclasse *SingleFileFortranRefactoring* que por sua vez estende a classe *Refactoring*. Nessa classe, quatro métodos precisam ser codificados:

- getName(): responsável por fornecer o título da refatoração *Unroll and Jam*.
- doCheckInitialConditions(): método utilizado para iniciar a ação de refatoração, o qual é utilizado para verificar pré-validações da modificação dos laços. Pode propagar uma exceção da classe *PreconditionFailure* indicando que alguma condição para a ação de refatoração não foi satisfeita e que a mesma não pode ser realizada.

- doCheckFinalConditions(): método utilizado para finalizar a ação de refatoração, o qual pode ser utilizado para realizar pós-validações na alteração dos laços. Da mesma forma que o método anterior, também pode utilizar exceções da classe *PreconditionFailure* para indicar que a refatoração não foi realizada.

- doCreateChange(): método principal que implementa a ação de refatoração *Unroll and Jam*.

Uma vez que as classes estão implementadas, é necessário alterar o arquivo *plugin.xml* do pacote *org.eclipse.photran.ui.vgp* para indicar ao Eclipse que existem novos pontos de extensão que precisam ser disponibilizados (BONIATI; CHARAO, 2008). Na Tabela 5.1 é possível observar os métodos utilizados para cada fase da implementação.

Pré-validação	Manipulação	Pós-validação
replaceAllLoopsIn()	*modifyTheBody()*	*addChangeFromModifiedAST()*
checkHeaders()		*releaseAllASTs()*
getFactor()		

Tabela 5.1: Métodos utilizados em cada fase da implementação.

5.5 Usabilidade

A refatoração *Unroll and Jam* pode ser aplicada a qualquer tipo de estrutura de laços de repetição, no qual o programador deverá fornecer um laço de repetição através de uma seleção de código começando com uma instrução DO e encerrando por uma instrução END DO, com isto formando um bloco de código único. O laço selecionado deverá conter uma variável de controle com uma expressão de inicialização e um limite.

O índice de desenrolamento deverá ser informado pelo programador para cada laço de iteração. É importante salientar que a variável refatorada é modificada dentro de todo o escopo do laço, e que o ambiente Eclipse fornece toda a base necessária de interação com o programador.

6 AVALIAÇÃO DE DESEMPENHO

O capítulo anterior descreveu a implementação da otimização *Unroll and Jam* através da refatoração pelo Photran. A fim de validar o presente trabalho, este capítulo apresenta os resultados de desempenho da implementação.

Na seção 6.1 é descrita a metodologia experimental utilizada para a realização dos experimentos, na seção 6.2 são ilustrados os resultados obtidos, na seção 6.3 é mostrado o código em Assembly do algoritmo de multiplicação de matrizes clássico, e na seção 6.4 é realizada a análise dos resultados obtidos.

6.1 Metodologia Experimental

A metodologia utilizada neste trabalho consiste em determinar o desempenho da otimização *Unroll and Jam* através da refatoração implementada e de diretivas de compilação.

6.1.1 *Benchmarks*

Os *benchmarks* utilizados na avaliação foram: multiplicação de matrizes clássico com matrizes de 2400x2400, e a subrotina SGTSV da biblioteca Lapack com uma matriz de 22000x22000. Para cada *Benchmark* foram utilizadas 9 versões de algoritmo os quais são:

- Algoritmo original sem otimização.
- 4 Algoritmos otimizados com *Unroll and Jam* pelo compilador intel através do pragma no código DEC UNROLL_AND_JAM, com índices de desenrolamento 2, 4, 8 e 16.
- 4 Algoritmos otimizados com *Unroll and Jam* pela refatoração com índices de desenrolamento 2, 4, 8 e 16.

Para o algoritmo da subrotina SGTSV da biblioteca Lapack, foi gerada uma versão de algoritmo que implementa para o mesmo laço a otimização pelo compilador e pela refatoração. O pragma de otimização do compilador é aplicada mais externamente ao laço, enquanto que a otimização da refatoração é realizada mais internamente.

Para cada versão são realizadas 30 execuções, onde é avaliado o tempo de execução (segundos), o número de leituras na memória cache L1 através do Vtune Amplifier, e o *speed-up* (*S*) das otimizações para cada índice de desenrolamento (ID).

6.1.2 Arquitetura

Os experimentos foram realizados em uma arquitetura *multi-core* com a seguinte configuração:

- Processador Intel Core 2 Duo T8100 2.10 GHz.
- 64KB(32KB para dados e para instruções) de Cache L1 por core, 3MB Cache L2, 4GB RAM.
- Sistema Operacional GNU-Linux Kernel 2.6.35-24.
- Compilador Intel 12.0.2.
- Analisador Vtune Amplifier XE 2011.

6.2 Resultados

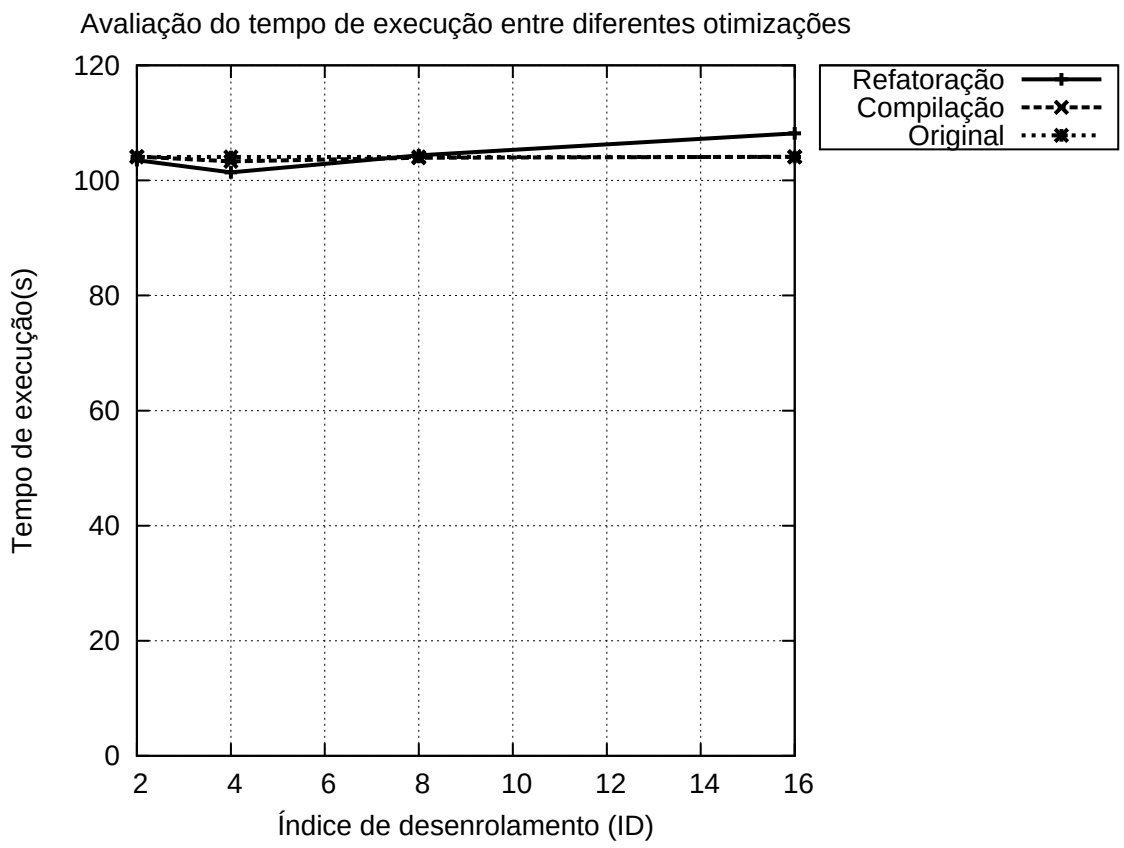
Nesta seção são descritos os resultados dos *benchmarks* citados anteriormente.

6.2.1 Multiplicação de matrizes

O algoritmo de multiplicação de matrizes clássico consiste em um bloco de instruções com três níveis de laços aninhados. Nesta versão, a otimização consistiu no desenrolamento do laço mais externo do algoritmo. Na Fig. 6.1 é possível observar o tempo de execução, e na Fig. 6.2 o número de requisições de leitura na memória cache L1 para as diferentes versões otimizadas. Na Fig 6.1 observa-se que de modo geral, a refatoração obteve um melhor desempenho em relação ao uso das opções de otimização do compilador.

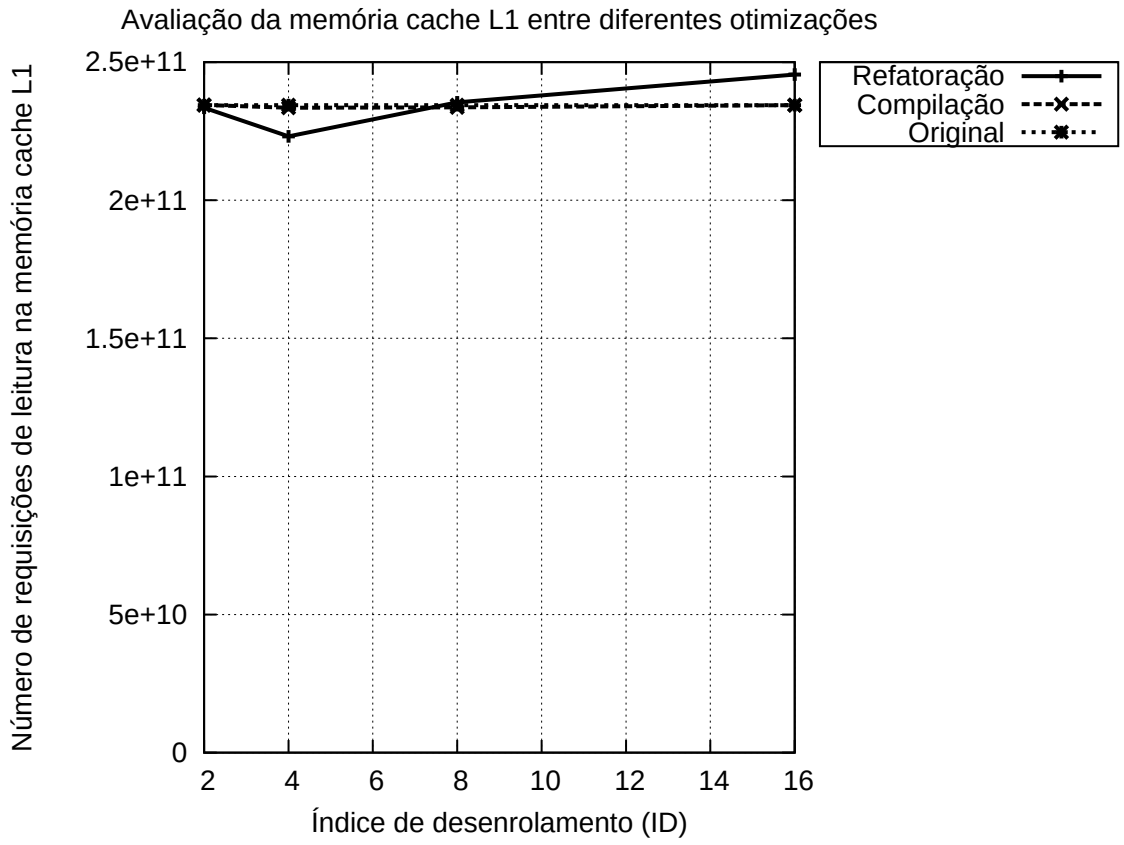
Mais especificamente para a versão refatorada, utilizando índices de desenrolamento 2 e 4 foi possível obter os melhores índices de desempenho, obtendo um *speed-up* de 1,01 e 1,03 respectivamente. Porém ao aumentar o índice de desenrolamento (8 e 16), ocorre uma perda de desempenho. Da mesma forma, na otimização pelo compilador, também houve uma queda de desempenho ao utilizar o índice de desenrolamento 16. Esta perda de desempenho em ambos tipos de otimizações, é devido à superação do limite ideal de desenrolamento para os laços.

Nota-se que o mesmo comportamento do gráfico da Fig. 6.1 também é obtido no número de requisições de leitura na memória cache L1 obtido pelo Vtune Amplifier (Fig. 6.2).



Multiplicação de matrizes, 30 amostras, $\alpha = 0,95$								
	Refatoração			Compilação			Original	
ID	$\overline{X}$	σ	S	$\overline{X}$	σ	S	$\overline{X}$	σ
2	103.52	0,052	1,01	104.16	0,015	1	104.06	0,041
4	101.40	0,061	1,03	103.31	0,012	1,01	104.06	0,041
8	104.35	0,015	1	103.95	0,051	1	104.06	0,041
16	108.19	0,026	0,96	104.11	0,052	1	104.06	0,041

Figura 6.1: Tempo de execução do algoritmo multiplicação de matrizes



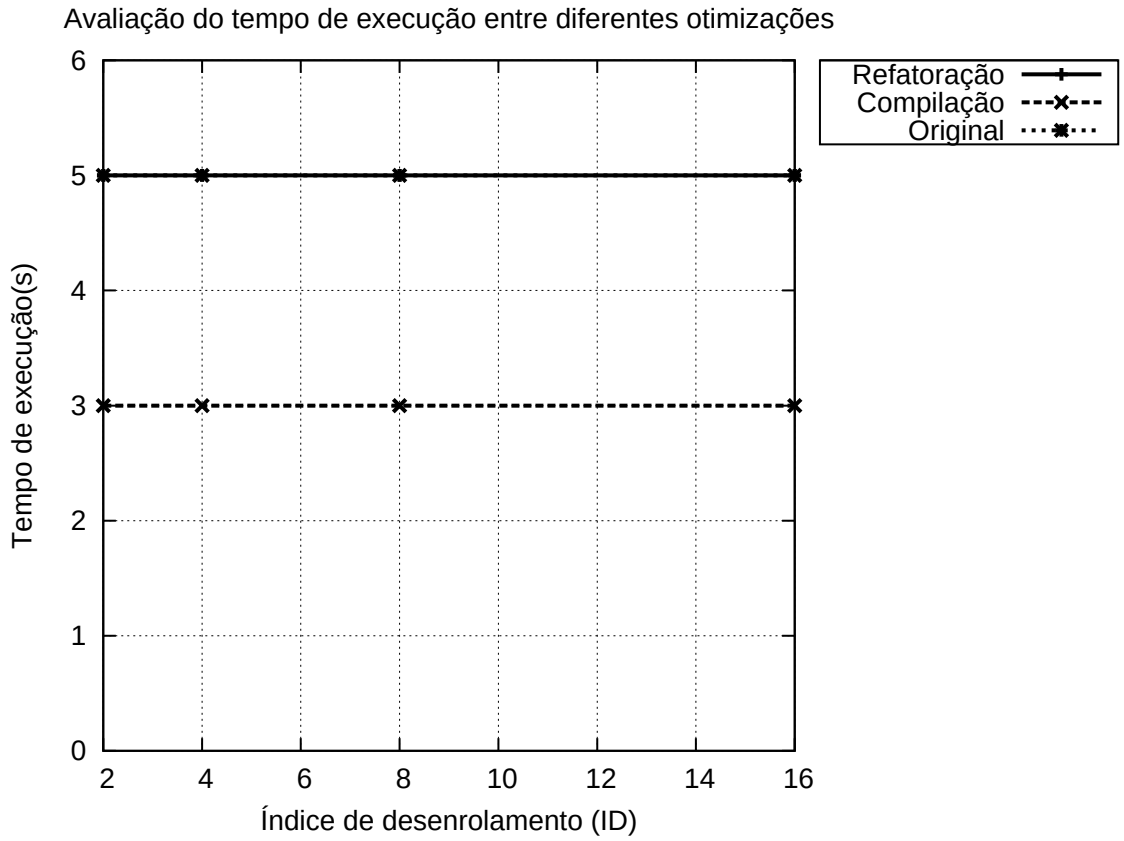
Multiplicação de matrizes, 30 amostras, $\alpha = 0,95$						
	Refatoração		Compilação		Original	
ID	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
2	2334895x10^5	0,021	2344675x10^5	0,094	2343975x10^5	0,023
4	2231585x10^5	0,041	2334795x10^5	0,075	2343975x10^5	0,023
8	2354325x10^5	0,053	2335992x10^5	0,052	2343975x10^5	0,023
16	2455175x10^5	0,048	2344219x10^5	0,041	2343975x10^5	0,023

Figura 6.2: Número de requisições de leitura na memória cache L1 do algoritmo multiplicação de matrizes

6.2.2 Subrotina SGTSV da biblioteca Lapack v.1

Esta aplicação consiste na subrotina SGTSV da biblioteca Lapack (versão 3.3.1) que resolve problemas de equações lineares do tipo $A * X = B$, onde A é uma matriz tridiagonal nXn, através de eliminação gaussiana com pivotamento parcial.

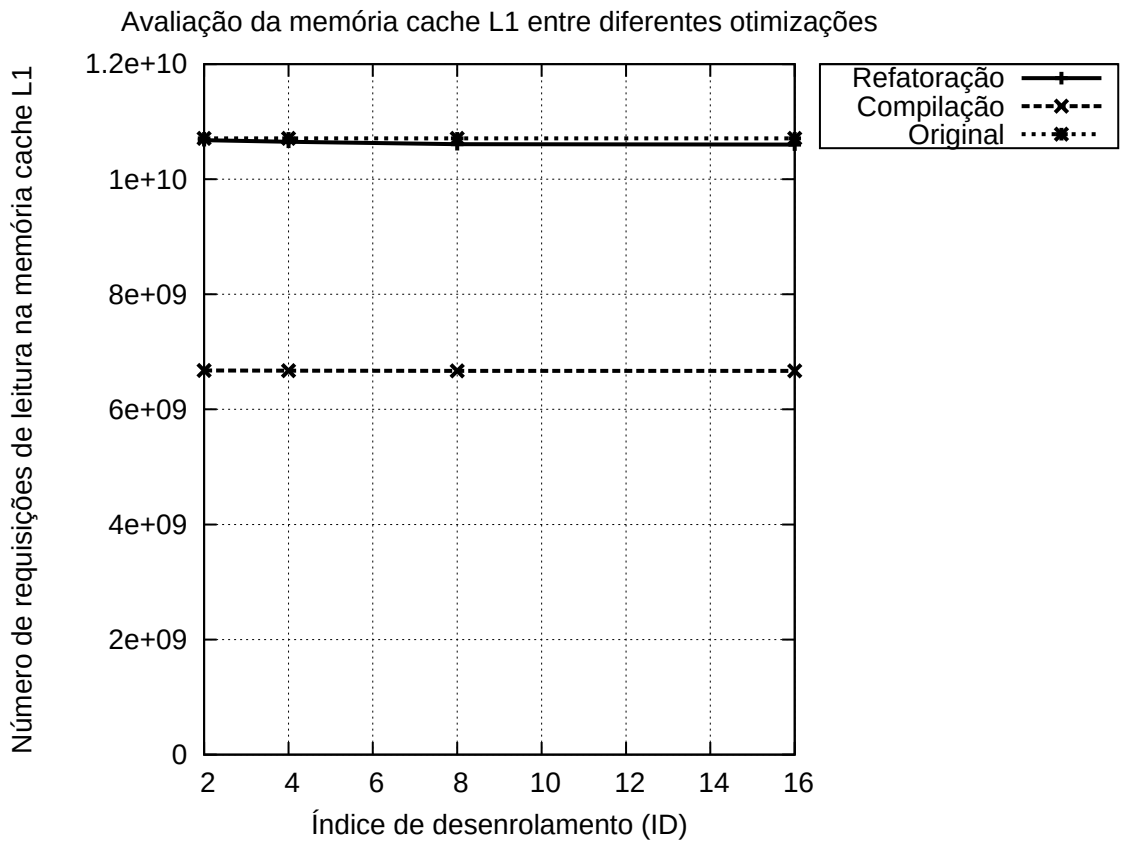
Nesta subrotina os laços que percorrem as colunas da matriz foram otimizados. Na Fig. 6.3 é possível observar o tempo de execução, e na Fig. 6.4 o número de requisições de leitura na memória cache L1 para as diferentes versões.



Subrotina SGTSV da biblioteca Lapack v.1, 30 amostras, $\alpha = 0,95$

	Refatoração			Compilação			Original	
ID	$\overline{X}$	σ	S	$\overline{X}$	σ	S	$\overline{X}$	σ
2	5,42	0,121	1,04	3,71	0,024	1,52	5,64	0,083
4	5,42	0,013	1,04	3,71	0,067	1,52	5,64	0,083
8	5,40	0,015	1,04	3,70	0,098	1,52	5,64	0,083
16	5,39	0,071	1,05	3,69	0,088	1,53	5,64	0,083

Figura 6.3: Tempo de execução da subrotina SGTSV da biblioteca Lapack v.1.



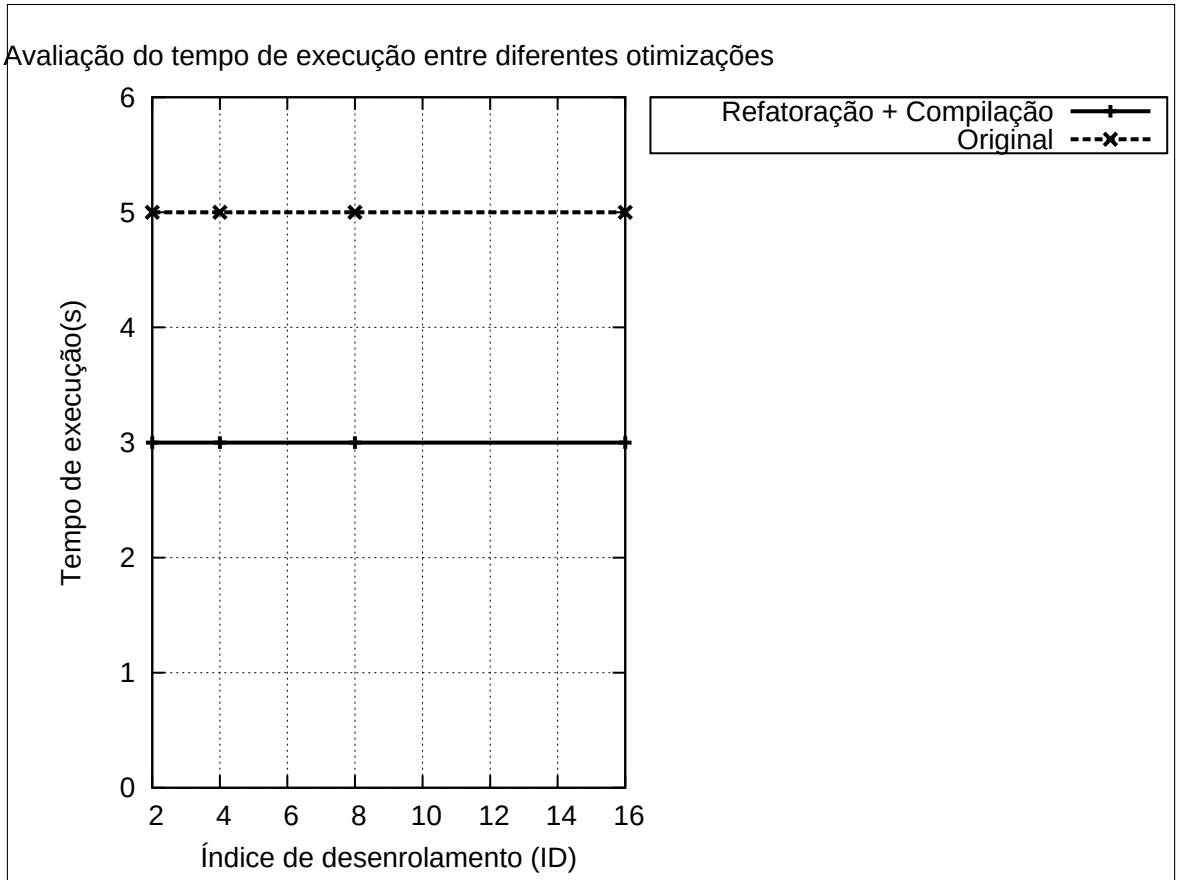
Subrotina SGTSV da biblioteca Lapack v.1, 30 amostras, $\alpha = 0,95$						
	Refatoração		Compilação		Original	
ID	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
2	106800x10^5	0,054	66750x10^5	0,035	107100x10^5	0,035
4	106508x10^5	0,032	66731x10^5	0,012	107100x10^5	0,035
8	106100x10^5	0,010	66710x10^5	0,145	107100x10^5	0,035
16	106020x10^5	0,102	66690x10^5	0,062	107100x10^5	0,035

Figura 6.4: Número de requisições de leitura na memória cache L1 da subrotina SGTSV da biblioteca Lapack v.1.

Nesta versão os melhores índices de desempenho foram obtidos pelos algoritmos otimizados pelo compilador. O tempo de execução foi praticamente similar para os diferentes índices de desenrolamento (Fig. 6.3), o mesmo comportamento é obtido pelo número de requisições de leitura na memória cache L1 (Fig. 6.4). O *speed-up* máximo alcançado pela compilação foi de 1,53, enquanto que pela refatoração o maior *speed-up* foi de 1,05. A partir destes resultados, na seguinte versão foi avaliada a utilização de ambas as técnicas de otimização.

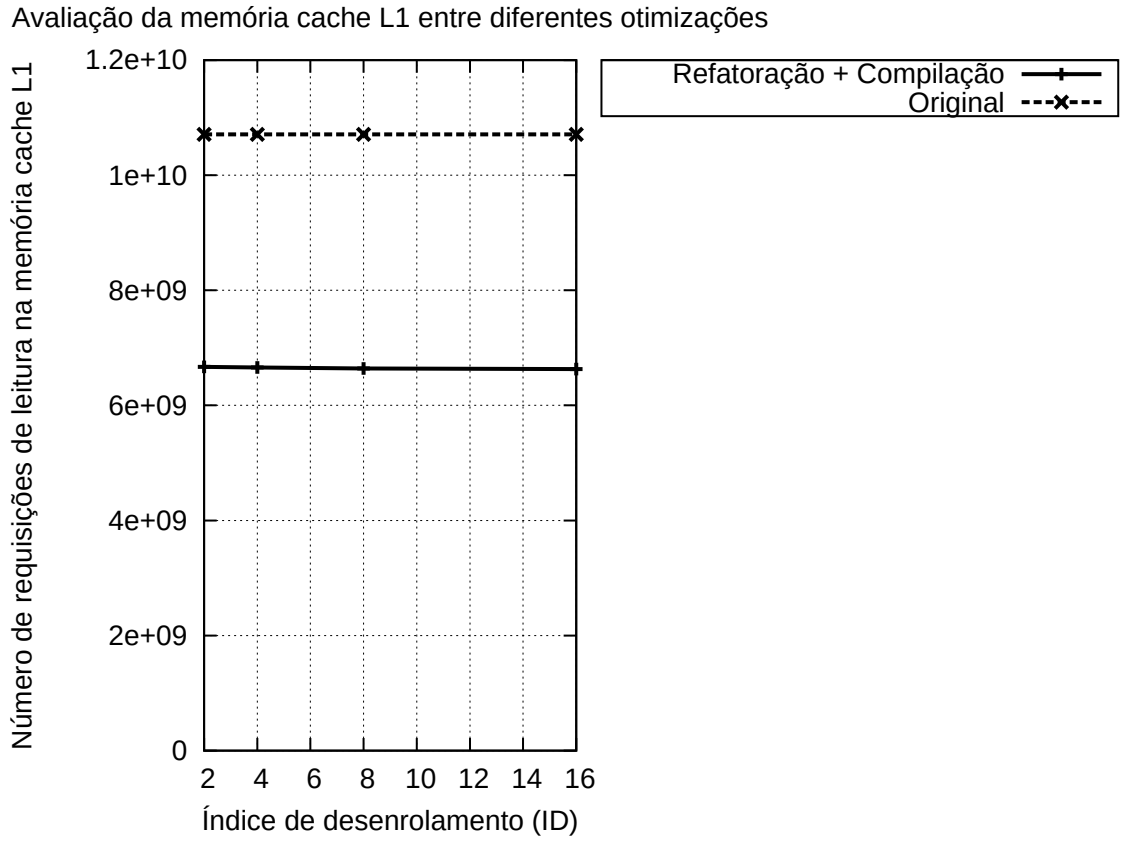
6.2.3 Subrotina SGTSV da biblioteca Lapack v.2

Nesta versão foi aplicada a otimização conjunta pelo pragma de otimização DEC UNROLL_AND_JAM do compilador e pela refatoração. Na Fig. 6.5 é possível observar o tempo de execução, e na Fig. 6.6 o número de requisições de leitura na memória cache L1 para as diferentes versões.



Subrotina SGTSV da biblioteca Lapack v.2, 30 amostras, $\alpha = 0,95$					
	Refatoração + Compilação			Original	
ID	$\overline{X}$	σ	S	$\overline{X}$	σ
2	3,54	0,091	1,59	5,64	0,083
4	3,45	0,126	1,63	5,64	0,083
8	3,42	0,091	1,65	5,64	0,083
16	3,42	0,104	1,65	5,64	0,083

Figura 6.5: Tempo de execução da subrotina SGTSV da biblioteca Lapack v.2.



Subrotina SGTSV da biblioteca Lapack v.2, 30 amostras, $\alpha = 0,95$				
	Refatoração + Compilação		Original	
ID	$\overline{X}$	σ	$\overline{X}$	σ
2	66710x10^5	0,077	107100x10^5	0,035
4	66590x10^5	0,106	107100x10^5	0,035
8	66400x10^5	0,091	107100x10^5	0,035
16	66310x10^5	0,093	107100x10^5	0,035

Figura 6.6: Número de requisições de leitura na memória cache L1 da subrotina SGTSV da biblioteca Lapack v.2.

Como pode ser visto na Fig. 6.5, nesta versão foram obtidos os melhores índices de desempenho para a subrotina SGTSV da biblioteca Lapack . A combinação de ambas as técnicas gerou um *speed-up* de 1,65, sendo superior ao *speed-up* de 1,53 obtido pela otimização da compilação. O mesmo ganho de desempenho é refletido no número de requisições de leitura na memória cache L1 (Fig. 6.6).

6.3 Geração de código em Assembly

Para avaliar o desempenho do algoritmo multiplicação de matrizes com índice de desenrolamento 2, foi analisado o código Assembly gerado pelo compilador da Intel através do pragma DEC UNROLL_AND_JAM e o código Assembly gerado pela refatoração. No

Apêndice, no código 7.1 é possível observar que o código gerado pela refatoração teve menor número de acessos à memória, enquanto que no código 7.2 obtido pelas diretivas de otimização do compilador, ocorre um maior número de acessos à memória.

Este menor número de acesso à memória reflete diretamente no tempo de execução das aplicações. Sendo assim, estas avaliações no código Assembly justificam os resultados de desempenho obtidos pelo algoritmo citado na subseção anterior.

6.4 Análise dos resultados

Em relação aos resultados obtidos nos *benchmarks* anteriores, é possível afirmar que através do aumento do índice de desenrolamento, houve um impacto positivo no desempenho nas versões otimizadas pela refatoração.

Os melhores resultados obtidos pela refatoração foram na versão refatorada da multiplicação de matrizes. Estes ganhos de desempenho são devidos à menor necessidade de acesso à memória cache L1 para obtenção de dados, fazendo utilização de modo mais eficiente dos registradores disponíveis.

Porém a partir do índice de desenrolamento 8 na versão refatorada da multiplicação de matrizes, houve uma perda de desempenho, sendo assim, é possível afirmar que o índice de desenrolamento nos laços possui um limite para obtenção de bons índices de desempenho. Isto significa que ao superar este limite, o desempenho é comprometido. Neste sentido, é necessário avaliar até que ponto o índice de desenrolamento da refatoração pode ser benéfico para os laços.

Também é possível destacar que a utilização dos pragmas de otimização do compilador e dos desenrolamentos da refatoração, indicaram que melhores índices de desempenho podem ser obtidos se ambas as técnicas foram aplicadas juntamente ao laço.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho implementou o emprego da otimização *Unroll and Jam* através de mecanismos de refatoração do código fonte, por meio da inserção, e realocação dos blocos de instruções. Os benefícios da utilização de *Unroll and Jam* são consideráveis para o desempenho das aplicações, pois esta técnica reestrutura os laços de iteração fazendo que as aplicações utilizem os recursos da arquitetura disponíveis de modo eficiente.

Os resultados de desempenho validaram o presente trabalho e demonstraram que é possível obter melhores índices de desempenho através dos índices de desenrolamento de laços realizados pela refatoração. Porém é necessário avaliar a natureza da aplicação, para saber melhor qual técnica de otimização deve ser utilizada.

Os resultados obtidos na utilização conjunta da otimização realizada pela refatoração e pela compilação foram superiores aos resultados encontrados nos *Benchmarks* que foram otimizados por uma única técnica de otimização. Isto quer dizer que o pré-processamento do código fonte possibilita que o otimizador do compilador faça um trabalho mais eficiente de desenrolamento.

Sob um ponto de vista mais amplo, a otimização assistida pela refatoração possibilita que durante a fase de desenvolvimento o programador tenha um controle mais preciso das otimizações no código fonte.

Ambientes de desenvolvimento integrado que implementam a otimização assistida pela refatoração, podem fornecer funcionalidades que indiquem ao programador quais são os melhores índices de desenrolamento para cada laço de iteração do código fonte. A partir desta interatividade, é possível obter um código fonte otimizado pronto para futuras otimizações do compilador e paralelizações.

Como trabalho futuro é necessário que novas pesquisas possam ser realizadas para a obtenção de índices de desenrolamento ótimos na refatoração. Também é interessante que haja estudos na aplicação conjunta de diferentes abordagens de otimização.

Outros trabalhos futuros podem ser destacados, como o estudo de outros tipos de otimização que possam ser aplicados através da refatoração.

APÊNDICE

Os códigos a seguir ilustram os diferentes números de acesso à memória do algoritmo de multiplicação de matrizes, com índice de desenrolamento 2, otimizado pela refatoração (código 7.1) e pelo compilador (código 7.2).

Código 7.1: Código Assembly gerado pelo compilador Intel através da refatoração

```
# mark_begin;
        .align    2,0x90
       .globl MAIN__
MAIN__:
..B1.1:                         # Preds ..B1.0
        pushl     %ebx                                          #1.9
        movl      %esp, %ebx                                    #1.9
        andl      $-16, %esp                                    #1.9
        pushl     %ebp                                          #1.9
        pushl     %ebp                                          #1.9
        movl      4(%ebx), %ebp                                 #1.9
        movl      %ebp, 4(%esp)                                 #1.9
        movl      %esp, %ebp                                    #1.9
        subl      $88, %esp                                     #1.9
        movl      %esi, -20(%ebp)                               #1.9
        movl      $.L_2_LITPACK_0.0.1, (%esp)                   #1.9
        call      for_set_reentrancy                            #1.9
                                # LOE
..B1.19:                        # Preds ..B1.1
        addl      $16, %esp                                     #1.9
                                # LOE
..B1.2:                         # Preds ..B1.19
        movl      $1, -12(%ebp)                                 #5.2
                                # LOE
..B1.3:                         # Preds ..B1.5 ..B1.2
        movl      $1, -8(%ebp)                                  #6.10
                                # LOE
..B1.4:                         # Preds ..B1.4 ..B1.3
        movl      -8(%ebp), %eax                                #7.5
        imull     $9600, %eax, %eax                             #7.5
        movl      $matrix_multiplication_$A.0.1, %edx           #7.5
        addl      %eax, %edx                                    #7.5
        addl      $-9600, %edx                                  #7.5
        movl      -12(%ebp), %eax                               #7.5
        imull     $4, %eax, %eax                                #7.5
        addl      %eax, %edx                                    #7.5
        addl      $-4, %edx                                     #7.5
        movl      $1, (%edx)                                    #7.5
        movl      -8(%ebp), %eax                                #8.5
```

```
40          imull     $9600, %eax, %eax                               #8.5
41          movl      $matrix_multiplication_$B.0.1, %edx             #8.5
42          addl      %eax, %edx                                      #8.5
43          addl      $-9600, %edx                                    #8.5
44          movl      -12(%ebp), %eax                                 #8.5
45          imull     $4, %eax, %eax                                  #8.5
46          addl      %eax, %edx                                      #8.5
47          addl      $-4, %edx                                       #8.5
48          movl      $1, (%edx)                                      #8.5
49          movl      -8(%ebp), %eax                                  #9.5
50          imull     $9600, %eax, %eax                               #9.5
51          movl      $matrix_multiplication_$C.0.1, %edx             #9.5
52          addl      %eax, %edx                                      #9.5
53          addl      $-9600, %edx                                    #9.5
54          movl      -12(%ebp), %eax                                 #9.5
55          imull     $4, %eax, %eax                                  #9.5
56          addl      %eax, %edx                                      #9.5
57          addl      $-4, %edx                                       #9.5
58          movl      $1, (%edx)                                      #9.5
59          movl      $1, %eax                                        #10.3
60          addl      -8(%ebp), %eax                                  #10.3
61          movl      %eax, -8(%ebp)                                  #10.3
62          movl      -8(%ebp), %eax                                  #10.3
63          cmpl      $2400, %eax                                     #10.3
64          jle       ..B1.4        # Prob 50%                        #10.3
65                                    # LOE
66  ..B1.5:                           # Preds ..B1.4
67          movl      $1, %eax                                        #11.2
68          addl      -12(%ebp), %eax                                 #11.2
69          movl      %eax, -12(%ebp)                                 #11.2
70          movl      -12(%ebp), %eax                                 #11.2
71          cmpl      $2400, %eax                                     #11.2
72          jle       ..B1.3        # Prob 50%                        #11.2
73                                    # LOE
74  ..B1.6:                           # Preds ..B1.5
75          addl      $-16, %esp                                      #12.8
76          lea       -32(%ebp), %eax                                 #12.17
77          movl      %eax, (%esp)                                    #12.17
78          call      for_cpusec                                      #12.8
79                                    # LOE
80  ..B1.20:                          # Preds ..B1.6
81          addl      $16, %esp                                       #12.8
82                                    # LOE
83  ..B1.7:                           # Preds ..B1.20
84          movl      $1, -4(%ebp)                                    #13.3
85                                    # LOE
86  ..B1.8:                           # Preds ..B1.12 ..B1.7
87          movl      $1, -8(%ebp)                                    #14.4
88                                    # LOE
89  ..B1.9:                           # Preds ..B1.11 ..B1.8
90          movl      $1, -12(%ebp)                                   #15.7
91                                    # LOE
92  ..B1.10:                          # Preds ..B1.10 ..B1.9
93          movl      -8(%ebp), %eax                                  #16.6
94          imull     $9600, %eax, %eax                               #16.6
95          movl      $matrix_multiplication_$A.0.1, %edx             #16.15
96          addl      %eax, %edx                                      #16.6
97          addl      $-9600, %edx                                    #16.6
```

```
98          movl    -12(%ebp), %eax                 #16.15
99          imull   $4, %eax, %eax                  #16.15
100         addl    %eax, %edx                      #16.6
101         addl    $-4, %edx                       #16.6
102         movl    -8(%ebp), %eax                  #16.24
103         imull   $9600, %eax, %eax               #16.24
104         movl    $matrix_multiplication_$C.0.1, %ecx #16.33
105         addl    %eax, %ecx                      #16.6
106         addl    $-9600, %ecx                    #16.6
107         movl    -4(%ebp), %eax                  #16.33
108         imull   $4, %eax, %eax                  #16.33
109         addl    %eax, %ecx                      #16.6
110         addl    $-4, %ecx                       #16.6
111         movl    -4(%ebp), %eax                  #16.15
112         imull   $9600, %eax, %eax               #16.15
113         movl    $matrix_multiplication_$B.0.1, %esi #16.24
114         addl    %eax, %esi                      #16.6
115         addl    $-9600, %esi                    #16.6
116         movl    -12(%ebp), %eax                 #16.24
117         imull   $4, %eax, %eax                  #16.24
118         addl    %eax, %esi                      #16.6
119         addl    $-4, %esi                       #16.6
120         movl    (%ecx), %eax                    #16.33
121         imull   (%esi), %eax                    #16.31
122         addl    (%edx), %eax                    #16.6
123         movl    -8(%ebp), %edx                  #16.22
124         imull   $9600, %edx, %edx               #16.22
125         movl    $matrix_multiplication_$A.0.1, %ecx #16.6
126         addl    %edx, %ecx                      #16.6
127         addl    $-9600, %ecx                    #16.6
128         movl    -12(%ebp), %edx                 #16.6
129         imull   $4, %edx, %edx                  #16.6
130         addl    %edx, %ecx                      #16.6
131         addl    $-4, %ecx                       #16.6
132         movl    %eax, (%ecx)                    #16.6
133         movl    -8(%ebp), %eax                  #17.7
134         imull   $9600, %eax, %eax               #17.7
135         movl    $matrix_multiplication_$A.0.1, %edx #17.16
136         addl    %eax, %edx                      #17.7
137         addl    $-9600, %edx                    #17.7
138         movl    -12(%ebp), %eax                 #17.16
139         imull   $4, %eax, %eax                  #17.16
140         addl    %eax, %edx                      #17.7
141         addl    $-4, %edx                       #17.7
142         movl    -8(%ebp), %eax                  #17.25
143         imull   $9600, %eax, %eax               #17.25
144         movl    $matrix_multiplication_$C.0.1, %ecx #17.36
145         addl    %eax, %ecx                      #17.7
146         addl    $-9600, %ecx                    #17.7
147         movl    $1, %eax                        #17.7
148         addl    -4(%ebp), %eax                  #17.36
149         imull   $4, %eax, %eax                  #17.36
150         addl    %eax, %ecx                      #17.7
151         addl    $-4, %ecx                       #17.7
152         movl    $1, %eax                        #17.7
153         addl    -4(%ebp), %eax                  #17.16
154         imull   $9600, %eax, %eax               #17.16
155         movl    $matrix_multiplication_$B.0.1, %esi #17.25
```

```
156         addl      %eax, %esi                                    #17.7
157         addl      $-9600, %esi                                  #17.7
158         movl      -12(%ebp), %eax                               #17.25
159         imull     $4, %eax, %eax                                #17.25
160         addl      %eax, %esi                                    #17.7
161         addl      $-4, %esi                                     #17.7
162         movl      (%ecx), %eax                                  #17.36
163         imull     (%esi), %eax                                  #17.34
164         addl      (%edx), %eax                                  #17.7
165         movl      -8(%ebp), %edx                                #17.23
166         imull     $9600, %edx, %edx                             #17.23
167         movl      $matrix_multiplication_$A.0.1, %ecx           #17.7
168         addl      %edx, %ecx                                    #17.7
169         addl      $-9600, %ecx                                  #17.7
170         movl      -12(%ebp), %edx                               #17.7
171         imull     $4, %edx, %edx                                #17.7
172         addl      %edx, %ecx                                    #17.7
173         addl      $-4, %ecx                                     #17.7
174         movl      %eax, (%ecx)                                  #17.7
175         movl      $1, %eax                                      #18.6
176         addl      -12(%ebp), %eax                               #18.6
177         movl      %eax, -12(%ebp)                               #18.6
178         movl      -12(%ebp), %eax                               #18.6
179         cmpl      $2400, %eax                                   #18.6
180         jle       ..B1.10       # Prob 50%                      #18.6
181                                   # LOE
182 ..B1.11:                         # Preds ..B1.10
183         movl      $1, %eax                                      #19.4
184         addl      -8(%ebp), %eax                                #19.4
185         movl      %eax, -8(%ebp)                                #19.4
186         movl      -8(%ebp), %eax                                #19.4
187         cmpl      $2400, %eax                                   #19.4
188         jle       ..B1.9        # Prob 50%                      #19.4
189                                   # LOE
190 ..B1.12:                         # Preds ..B1.11
191         movl      $2, %eax                                      #20.3
192         addl      -4(%ebp), %eax                                #20.3
193         movl      %eax, -4(%ebp)                                #20.3
194         movl      -4(%ebp), %eax                                #20.3
195         cmpl      $2400, %eax                                   #20.3
196         jle       ..B1.8        # Prob 50%                      #20.3
197                                   # LOE
198 ..B1.13:                         # Preds ..B1.12
199         addl      $-16, %esp                                    #21.8
200         lea       -28(%ebp), %eax                               #21.17
201         movl      %eax, (%esp)                                  #21.17
202         call      for_cpusec                                    #21.8
203                                   # LOE
204 ..B1.21:                         # Preds ..B1.13
205         addl      $16, %esp                                     #21.8
206                                   # LOE
207 ..B1.14:                         # Preds ..B1.21
208         movss     -28(%ebp), %xmm0                              #22.3
209         movss     -32(%ebp), %xmm1                              #22.3
210         subss     %xmm1, %xmm0                                  #22.3
211         movss     %xmm0, -24(%ebp)                              #22.3
212         movl      $0, -72(%ebp)                                 #23.3
213         movl      $10, -40(%ebp)                                #23.3
```

```
214         movl      $.L_2_STRLITPACK_0, -36(%ebp)                 #23.3
215         addl      $-32, %esp                                    #23.3
216         lea       -72(%ebp), %eax                               #23.3
217         movl      %eax, (%esp)                                  #23.3
218         movl      $-1, 4(%esp)                                  #23.3
219         movl      $-2088435968, 8(%esp)                         #23.3
220         movl      $.L_2_STRLITPACK_1.0.1, 12(%esp)              #23.3
221         lea       -40(%ebp), %eax                               #23.3
222         movl      %eax, 16(%esp)                                #23.3
223         movl      $32, 20(%esp)                                 #23.3
224         call      for_write_seq_lis                             #23.3
225                                       # LOE
226 ..B1.22:                             # Preds ..B1.14
227         addl      $32, %esp                                     #23.3
228                                       # LOE
229 ..B1.15:                             # Preds ..B1.22
230         movss     -24(%ebp), %xmm0                              #23.3
231         movss     %xmm0, -16(%ebp)                              #23.3
232         addl      $-16, %esp                                    #23.3
233         lea       -72(%ebp), %eax                               #23.3
234         movl      %eax, (%esp)                                  #23.3
235         movl      $.L_2_STRLITPACK_2.0.1, 4(%esp)               #23.3
236         lea       -16(%ebp), %eax                               #23.3
237         movl      %eax, 8(%esp)                                 #23.3
238         call      for_write_seq_lis_xmit                        #23.3
239                                       # LOE
240 ..B1.23:                             # Preds ..B1.15
241         addl      $16, %esp                                     #23.3
242                                       # LOE
243 ..B1.16:                             # Preds ..B1.23
244         movl      $1, %eax                                      #24.1
245         movl      -20(%ebp), %esi                               #24.1
246         leave                                                   #24.1
247         movl      %ebx, %esp                                    #24.1
248         popl      %ebx                                          #24.1
249         ret                                                     #24.1
250         .align    2,0x90
251                                       # LOE
252 # mark_end;
```

Código 7.2: Código Assembly gerado pelo compilador Intel através das diretivas de otimização

```
1  # mark_begin;
2         .align    16,0x90
3        .globl MAIN__
4  MAIN__:
5  ..B1.1:                             # Preds ..B1.0
6           pushl     %ebp                                          #1.9
7           movl      %esp, %ebp                                    #1.9
8           andl      $-128, %esp                                   #1.9
9           pushl     %ebx                                          #1.9
10          subl      $632, %esp                                    #1.9
11          pushl     $3                                            #1.9
12          call      __intel_new_proc_init                         #1.9
13                                        # LOE esi edi
14 ..B1.65:                             # Preds ..B1.1
15          stmxcsr   128(%esp)                                     #1.9
```

```
16          orl       $32768, 128(%esp)                             #1.9
17          ldmxcsr   128(%esp)                                     #1.9
18          addl      $4, %esp                                      #1.9
19          pushl     $.L_2_LITPACK_0.0.1                           #1.9
20          call      for_set_reentrancy                            #1.9
21                                    # LOE esi edi
22  ..B1.2:                           # Preds ..B1.65
23          movdqa    .L_2il0floatpacket.0, %xmm0                   #
24          xorl      %eax, %eax                                    #
25                                    # LOE eax esi edi xmm0
26  ..B1.3:                           # Preds ..B1.3 ..B1.2
27          movntdq   %xmm0, matrix_multiplication_$A.0.1(,%eax,4)  #7.5
28          movntdq   %xmm0, matrix_multiplication_$B.0.1(,%eax,4)  #8.5
29          movntdq   %xmm0, matrix_multiplication_$C.0.1(,%eax,4)  #9.5
30          addl      $4, %eax                                      #6.10
31          cmpl      $5760000, %eax                                #6.10
32          jb        ..B1.3          # Prob 99%                    #6.10
33                                    # LOE eax esi edi xmm0
34  ..B1.4:                           # Preds ..B1.3
35          addl      $4, %esp                                      #12.8
36          lea       164(%esp), %eax                               #12.17
37          pushl     %eax                                          #12.8
38          call      for_cpusec                                    #12.8
39                                    # LOE esi edi
40  ..B1.6:                           # Preds ..B1.4
41          movdqa    .L_2il0floatpacket.1, %xmm0                   #17.32
42          xorl      %edx, %edx                                    #14.3
43          movl      %esi, 136(%esp)                               #17.32
44          movl      $512, %eax                                    #17.32
45          movl      %edi, 140(%esp)                               #17.32
46                                    # LOE eax edx xmm0
47  ..B1.7:                           # Preds ..B1.53 ..B1.6
48          movl      %edx, %ebx                                    #14.3
49          xorl      %esi, %esi                                    #14.3
50          shll      $9, %ebx                                      #14.3
51          negl      %ebx                                          #14.3
52          addl      $2400, %ebx                                   #14.3
53          cmpl      $512, %ebx                                    #14.3
54          movl      %esi, 200(%esp)                               #14.3
55          cmova     %eax, %ebx                                    #14.3
56          movl      %ebx, %ecx                                    #14.3
57          shrl      $1, %ecx                                      #14.3
58          movl      %ecx, 144(%esp)                               #14.3
59          movl      %ebx, 128(%esp)                               #14.3
60          movl      %edx, 164(%esp)                               #14.3
61                                    # LOE xmm0
62  ..B1.8:                           # Preds ..B1.52 ..B1.7
63          movl      200(%esp), %edx                               #15.4
64          movl      %edx, %esi                                    #15.4
65          shll      $9, %esi                                      #15.4
66          movl      $512, %ebx                                    #15.4
67          negl      %esi                                          #15.4
68          xorl      %eax, %eax                                    #14.3
69          addl      $2400, %esi                                   #15.4
70          imull     $4915200, %edx, %edx                          #
71          cmpl      $512, %esi                                    #15.4
72          movl      164(%esp), %edi                               #
73          movl      %edi, %ecx                                    #
```

```
74          cmova     %ebx, %esi                                    #15.4
75          imull     $4915200, %edi, %edi                          #
76          movl      %esi, %ebx                                    #15.4
77          shll      $11, %ecx                                     #
78          shrl      $2, %ebx                                      #15.4
79          addl      %edx, %ecx                                    #
80          movl      %edi, 148(%esp)                               #
81          movl      %ecx, 152(%esp)                               #
82          movl      %edx, 156(%esp)                               #
83          movl      %ebx, 172(%esp)                               #
84          movl      %esi, 196(%esp)                               #
85                                    # LOE eax xmm0
86  ..B1.10:                          # Preds ..B1.8 ..B1.51
87          cmpl      $0, 144(%esp)                                 #14.3
88          jbe       ..B1.62         # Prob 0%                     #14.3
89                                    # LOE eax xmm0
90  ..B1.11:                          # Preds ..B1.10
91          movl      %eax, %ecx                                    #16.7
92          movl      $512, %ebx                                    #16.7
93          shll      $9, %ecx                                      #16.7
94          xorl      %edx, %edx                                    #
95          negl      %ecx                                          #16.7
96          addl      $2400, %ecx                                   #16.7
97          cmpl      $512, %ecx                                    #16.7
98          movl      156(%esp), %esi                               #
99          cmovae    %ebx, %ecx                                    #16.7
100         movl      %eax, %ebx                                    #
101         shll      $11, %ebx                                     #
102         movl      %ecx, 244(%esp)                               #
103         movl      %eax, 160(%esp)                               #
104         addl      %ebx, %esi                                    #
105         addl      148(%esp), %ebx                               #
106         movl      %esi, 220(%esp)                               #
107         movl      %ebx, 176(%esp)                               #
108         movl      152(%esp), %esi                               #
109                                   # LOE edx esi xmm0
110 ..B1.12:                          # Preds ..B1.36 ..B1.11
111         cmpl      $0, 172(%esp)                                 #15.4
112         jbe       ..B1.61         # Prob 0%                     #15.4
113                                   # LOE edx esi xmm0
114 ..B1.13:                          # Preds ..B1.12
115         imull     $19200, %edx, %ecx                            #
116         xorl      %edi, %edi                                    #
117         addl      176(%esp), %ecx                               #
118         lea       (%esi,%edx,8), %ebx                           #
119         movl      %edi, 236(%esp)                               #
120         movl      %ecx, 520(%esp)                               #
121         movl      %ebx, 232(%esp)                               #
122         movl      %edi, 188(%esp)                               #
123         movl      %edx, 184(%esp)                               #
124                                   # LOE
125 ..B1.14:                          # Preds ..B1.22 ..B1.13
126         cmpl      $4, 244(%esp)                                 #16.7
127         jb        ..B1.60         # Prob 10%                    #16.7
128                                   # LOE
129 ..B1.15:                          # Preds ..B1.14
130         movl      236(%esp), %ebx                               #
131         xorl      %ecx, %ecx                                    #16.7
```

```
132         movl      232(%esp), %esi                                 #17.34
133         movl      220(%esp), %edx                                 #
134         movl      244(%esp), %eax                                 #16.7
135         movd      matrix_multiplication_$C.0.1(%ebx,%esi), %xmm7 #17.34
136         movd      4+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm0 #17.34
137         addl      %ebx, %edx                                      #
138         pshufd    $0, %xmm7, %xmm7                                #17.34
139         pshufd    $0, %xmm0, %xmm0                                #17.34
140         movd      9600+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm1 #17.34
141         movdqa    %xmm7, 496(%esp)                                #17.34
142         psrlq     $32, %xmm7                                      #17.32
143         movdqa    %xmm7, 480(%esp)                                #17.32
144         movdqa    %xmm0, %xmm7                                    #17.32
145         pshufd    $0, %xmm1, %xmm1                                #17.34
146         psrlq     $32, %xmm7                                      #17.32
147         movd      9604+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm2 #17.34
148         movdqa    %xmm7, 464(%esp)                                #17.32
149         movdqa    %xmm1, %xmm7                                    #17.32
150         pshufd    $0, %xmm2, %xmm2                                #17.34
151         psrlq     $32, %xmm7                                      #17.32
152         movd      19200+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm3 #17.34
153         movdqa    %xmm7, 448(%esp)                                #17.32
154         movdqa    %xmm2, %xmm7                                    #17.32
155         pshufd    $0, %xmm3, %xmm3                                #17.34
156         psrlq     $32, %xmm7                                      #17.32
157         movd      19204+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm4 #17.34
158         movdqa    %xmm7, 432(%esp)                                #17.32
159         movdqa    %xmm3, %xmm7                                    #17.32
160         pshufd    $0, %xmm4, %xmm4                                #17.34
161         psrlq     $32, %xmm7                                      #17.32
162         movd      28800+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm5 #17.34
163         movdqa    %xmm7, 416(%esp)                                #17.32
164         movdqa    %xmm4, %xmm7                                    #17.32
165         pshufd    $0, %xmm5, %xmm5                                #17.34
166         psrlq     $32, %xmm7                                      #17.32
167         movd      28804+matrix_multiplication_$C.0.1(%ebx,%esi), %xmm6 #17.34
168         movl      %eax, %esi                                      #17.32
169         movdqa    %xmm7, 400(%esp)                                #17.32
170         movdqa    %xmm5, %xmm7                                    #17.32
171         pshufd    $0, %xmm6, %xmm6                                #17.34
172         psrlq     $32, %xmm7                                      #17.32
173         movdqa    %xmm7, 384(%esp)                                #17.32
174         movdqa    %xmm6, %xmm7                                    #17.32
175         psrlq     $32, %xmm7                                      #17.32
176         movdqa    %xmm7, 368(%esp)                                #17.32
177         movdqa    %xmm0, 352(%esp)                                #17.32
178         movdqa    %xmm1, 336(%esp)                                #17.32
179         movdqa    %xmm2, 320(%esp)                                #17.32
180         movdqa    %xmm3, 304(%esp)                                #17.32
181         movdqa    %xmm4, 288(%esp)                                #17.32
182         movdqa    %xmm5, 272(%esp)                                #17.32
183         movdqa    %xmm6, 256(%esp)                                #17.32
184         movdqa    .L_2il0floatpacket.1, %xmm7                     #17.32
185         movl      520(%esp), %ebx                                 #17.32
186                                   # LOE eax edx ecx ebx esi xmm7
187 ..B1.16:                          # Preds ..B1.16 ..B1.15
188         movdqa    matrix_multiplication_$B.0.1(%ebx,%ecx,4), %xmm6 #17.25
189         movdqa    %xmm6, %xmm5                                    #17.32
```

```
190         movdqa    496(%esp), %xmm2                                      #17.32
191         psrlq     $32, %xmm5                                            #17.32
192         movdqa    480(%esp), %xmm0                                      #17.32
193         pmuludq   %xmm6, %xmm2                                          #17.32
194         pmuludq   %xmm5, %xmm0                                          #17.32
195         movdqa    9600+matrix_multiplication_$B.0.1(%ebx,%ecx,4), %xmm4 #17.25
196         pand      %xmm7, %xmm2                                          #17.32
197         psllq     $32, %xmm0                                            #17.32
198         movdqa    %xmm4, %xmm3                                          #17.32
199         movdqa    464(%esp), %xmm1                                      #17.32
200         por       %xmm0, %xmm2                                          #17.32
201         movdqa    352(%esp), %xmm0                                      #17.32
202         psrlq     $32, %xmm3                                            #17.32
203         pmuludq   %xmm4, %xmm0                                          #17.32
204         pmuludq   %xmm3, %xmm1                                          #17.32
205         pand      %xmm7, %xmm0                                          #17.32
206         psllq     $32, %xmm1                                            #17.32
207         paddd     matrix_multiplication_$A.0.1(%edx,%ecx,4), %xmm2 #17.7
208         por       %xmm1, %xmm0                                          #17.32
209         paddd     %xmm0, %xmm2                                          #17.7
210         movdqa    %xmm2, matrix_multiplication_$A.0.1(%edx,%ecx,4) #17.7
211         movdqa    336(%esp), %xmm1                                      #17.32
212         movdqa    448(%esp), %xmm2                                      #17.32
213         pmuludq   %xmm6, %xmm1                                          #17.32
214         pmuludq   %xmm5, %xmm2                                          #17.32
215         pand      %xmm7, %xmm1                                          #17.32
216         psllq     $32, %xmm2                                            #17.32
217         movdqa    432(%esp), %xmm0                                      #17.32
218         por       %xmm2, %xmm1                                          #17.32
219         movdqa    320(%esp), %xmm2                                      #17.32
220         pmuludq   %xmm4, %xmm2                                          #17.32
221         pmuludq   %xmm3, %xmm0                                          #17.32
222         pand      %xmm7, %xmm2                                          #17.32
223         psllq     $32, %xmm0                                            #17.32
224         paddd     9600+matrix_multiplication_$A.0.1(%edx,%ecx,4), %xmm1 #17.7
225         por       %xmm0, %xmm2                                          #17.32
226         paddd     %xmm2, %xmm1                                          #17.7
227         movdqa    %xmm1, 9600+matrix_multiplication_$A.0.1(%edx,%ecx,4) #17.7
228         movdqa    304(%esp), %xmm0                                      #17.32
229         movdqa    416(%esp), %xmm1                                      #17.32
230         pmuludq   %xmm6, %xmm0                                          #17.32
231         pmuludq   %xmm5, %xmm1                                          #17.32
232         pmuludq   272(%esp), %xmm6                                      #17.32
233         pmuludq   384(%esp), %xmm5                                      #17.32
234         pand      %xmm7, %xmm0                                          #17.32
235         psllq     $32, %xmm1                                            #17.32
236         movdqa    400(%esp), %xmm2                                      #17.32
237         por       %xmm1, %xmm0                                          #17.32
238         movdqa    288(%esp), %xmm1                                      #17.32
239         pand      %xmm7, %xmm6                                          #17.32
240         pmuludq   %xmm4, %xmm1                                          #17.32
241         psllq     $32, %xmm5                                            #17.32
242         pmuludq   256(%esp), %xmm4                                      #17.32
243         pmuludq   %xmm3, %xmm2                                          #17.32
244         pmuludq   368(%esp), %xmm3                                      #17.32
245         pand      %xmm7, %xmm1                                          #17.32
246         psllq     $32, %xmm2                                            #17.32
247         por       %xmm5, %xmm6                                          #17.32
```

```
248             pand      %xmm7, %xmm4                                  #17.32
249             psllq     $32, %xmm3                                    #17.32
250             por       %xmm2, %xmm1                                  #17.32
251             paddd     19200+matrix_multiplication_$A.0.1(%edx,%ecx,4), %xmm0 #17.7
252             por       %xmm3, %xmm4                                  #17.32
253             paddd     28800+matrix_multiplication_$A.0.1(%edx,%ecx,4), %xmm6 #17.7
254             paddd     %xmm1, %xmm0                                  #17.7
255             paddd     %xmm4, %xmm6                                  #17.7
256             movdqa    %xmm0, 19200+matrix_multiplication_$A.0.1(%edx,%ecx,4) #17.7
257             movdqa    %xmm6, 28800+matrix_multiplication_$A.0.1(%edx,%ecx,4) #17.7
258             addl      $4, %ecx                                      #16.7
259             cmpl      %esi, %ecx                                    #16.7
260             jb        ..B1.16       # Prob 99%                      #16.7
261                                     # LOE eax edx ecx ebx esi xmm7
262     ..B1.18:                        # Preds ..B1.16 ..B1.60
263             cmpl      244(%esp), %eax                               #16.7
264             jae       ..B1.22       # Prob 0%                       #16.7
265                                     # LOE eax
266     ..B1.19:                        # Preds ..B1.18
267             movl      236(%esp), %esi                               #
268             movl      232(%esp), %ebx                               #17.34
269             movl      220(%esp), %ecx                               #
270             movl      28804+matrix_multiplication_$C.0.1(%esi,%ebx), %edi #17.34
271             movl      %edi, 528(%esp)                               #17.34
272             lea       (%ecx,%esi), %edx                             #
273             movl      %edx, 240(%esp)                               #
274             movl      28800+matrix_multiplication_$C.0.1(%esi,%ebx), %ecx #17.34
275             movl      19200+matrix_multiplication_$C.0.1(%esi,%ebx), %edx #17.34
276             movl      9604+matrix_multiplication_$C.0.1(%esi,%ebx), %edi #17.34
277             movl      %ecx, 540(%esp)                               #17.34
278             movl      %edx, 536(%esp)                               #17.34
279             movl      %edi, 532(%esp)                               #17.34
280             movl      19204+matrix_multiplication_$C.0.1(%esi,%ebx), %ecx #17.34
281             movl      9600+matrix_multiplication_$C.0.1(%esi,%ebx), %edx #17.34
282             movl      4+matrix_multiplication_$C.0.1(%esi,%ebx), %edi #17.34
283             movl      matrix_multiplication_$C.0.1(%esi,%ebx), %ebx #17.34
284             movl      %ecx, 516(%esp)                               #17.34
285             movl      %edi, 524(%esp)                               #17.34
286             movl      %ebx, 252(%esp)                               #17.34
287             movl      %edx, 512(%esp)                               #17.34
288             movl      240(%esp), %ecx                               #17.34
289                                     # LOE eax ecx
290     ..B1.20:                        # Preds ..B1.20 ..B1.19
291             movl      520(%esp), %edi                               #17.25
292             movl      252(%esp), %esi                               #17.32
293             movl      matrix_multiplication_$B.0.1(%edi,%eax,4), %ebx #17.25
294             imull     %ebx, %esi                                    #17.32
295             movl      9600+matrix_multiplication_$B.0.1(%edi,%eax,4), %edx #17.25
296             movl      524(%esp), %edi                               #17.32
297             imull     %edx, %edi                                    #17.32
298             addl      matrix_multiplication_$A.0.1(%ecx,%eax,4), %esi #17.7
299             addl      %edi, %esi                                    #17.7
300             movl      512(%esp), %edi                               #17.32
301             imull     %ebx, %edi                                    #17.32
302             movl      %esi, matrix_multiplication_$A.0.1(%ecx,%eax,4) #17.7
303             movl      532(%esp), %esi                               #17.32
304             imull     %edx, %esi                                    #17.32
305             addl      9600+matrix_multiplication_$A.0.1(%ecx,%eax,4), %edi #17.7
```

```
306         addl    %esi, %edi                                      #17.7
307         movl    %edi, 9600+matrix_multiplication_$A.0.1(%ecx,%eax,4) #17.7
308         movl    536(%esp), %edi                                 #17.32
309         imull   %ebx, %edi                                      #17.32
310         imull   540(%esp), %ebx                                 #17.32
311         movl    516(%esp), %esi                                 #17.32
312         imull   %edx, %esi                                      #17.32
313         imull   528(%esp), %edx                                 #17.32
314         addl    19200+matrix_multiplication_$A.0.1(%ecx,%eax,4), %edi #17.7
315         addl    28800+matrix_multiplication_$A.0.1(%ecx,%eax,4), %ebx #17.7
316         addl    %esi, %edi                                      #17.7
317         addl    %edx, %ebx                                      #17.7
318         movl    %edi, 19200+matrix_multiplication_$A.0.1(%ecx,%eax,4) #17.7
319         movl    %ebx, 28800+matrix_multiplication_$A.0.1(%ecx,%eax,4) #17.7
320         incl    %eax                                            #16.7
321         cmpl    244(%esp), %eax                                 #16.7
322         jb      ..B1.20       # Prob 99%                        #16.7
323                               # LOE eax ecx
324 ..B1.22:                      # Preds ..B1.20 ..B1.18
325         movl    188(%esp), %edx                                 #15.4
326         incl    %edx                                            #15.4
327         movl    236(%esp), %eax                                 #15.4
328         addl    $38400, %eax                                    #15.4
329         movl    %eax, 236(%esp)                                 #15.4
330         movl    %edx, 188(%esp)                                 #15.4
331         cmpl    172(%esp), %edx                                 #15.4
332         jb      ..B1.14       # Prob 99%                        #15.4
333                               # LOE edx dl dh
334 ..B1.23:                      # Preds ..B1.22
335         movl    %edx, %edi                                      #
336         lea     1(,%edi,4), %ebx                                #15.4
337         movl    184(%esp), %edx                                 #
338         movl    152(%esp), %esi                                 #
339         movdqa  .L_2il0floatpacket.1, %xmm0                     #
340                               # LOE edx ebx esi xmm0
341 ..B1.24:                      # Preds ..B1.23 ..B1.61
342         movl    220(%esp), %ecx                                 #
343         lea     -1(%ebx), %edi                                  #15.4
344         imull   $9600, %edi, %eax                               #
345         cmpl    196(%esp), %ebx                                 #15.4
346         lea     (%ecx,%eax), %ecx                               #
347         ja      ..B1.36       # Prob 0%                         #15.4
348                               # LOE eax edx ecx esi edi xmm0
349 ..B1.25:                      # Preds ..B1.24
350         movl    %edi, 180(%esp)                                 #
351         lea     (%esi,%edx,8), %ebx                             #
352         addl    %ebx, %eax                                      #
353         imull   $19200, %edx, %ebx                              #
354         addl    176(%esp), %ebx                                 #
355         movl    %ebx, 228(%esp)                                 #
356         movl    %edx, 184(%esp)                                 #
357         movl    244(%esp), %ebx                                 #
358                               # LOE eax ecx ebx xmm0
359 ..B1.26:                      # Preds ..B1.34 ..B1.25
360         cmpl    $8, %ebx                                        #16.7
361         jb      ..B1.59       # Prob 10%                        #16.7
362                               # LOE eax ecx ebx xmm0
363 ..B1.27:                      # Preds ..B1.26
```

```
364         movd      matrix_multiplication_$C.0.1(%eax), %xmm5      #17.34
365         movl      %ebx, %esi                                     #16.7
366         movd      4+matrix_multiplication_$C.0.1(%eax), %xmm1    #17.34
367         xorl      %edx, %edx                                     #16.7
368         pshufd    $0, %xmm5, %xmm5                               #17.34
369         pshufd    $0, %xmm1, %xmm4                               #17.34
370         movdqa    %xmm5, %xmm3                                   #17.32
371         movdqa    %xmm4, %xmm6                                   #17.32
372         psrlq     $32, %xmm3                                     #17.32
373         movl      228(%esp), %edi                                #17.32
374         psrlq     $32, %xmm6                                     #17.32
375                                 # LOE eax edx ecx ebx esi edi xmm0 xmm3 xmm4 xmm5 xmm
376 ..B1.28:                        # Preds ..B1.28 ..B1.27
377         movdqa    matrix_multiplication_$B.0.1(%edi,%edx,4), %xmm7 #17.25
378         movdqa    %xmm5, %xmm2                                   #17.32
379         pmuludq   %xmm7, %xmm2                                   #17.32
380         psrlq     $32, %xmm7                                     #17.32
381         pmuludq   %xmm3, %xmm7                                   #17.32
382         pand      %xmm0, %xmm2                                   #17.32
383         psllq     $32, %xmm7                                     #17.32
384         movdqa    9600+matrix_multiplication_$B.0.1(%edi,%edx,4), %xmm1 #17.25
385         por       %xmm7, %xmm2                                   #17.32
386         movdqa    %xmm4, %xmm7                                   #17.32
387         pmuludq   %xmm1, %xmm7                                   #17.32
388         psrlq     $32, %xmm1                                     #17.32
389         pmuludq   %xmm6, %xmm1                                   #17.32
390         pand      %xmm0, %xmm7                                   #17.32
391         psllq     $32, %xmm1                                     #17.32
392         paddd     matrix_multiplication_$A.0.1(%ecx,%edx,4), %xmm2 #17.7
393         por       %xmm1, %xmm7                                   #17.32
394         paddd     %xmm7, %xmm2                                   #17.7
395         movdqa    %xmm5, %xmm1                                   #17.32
396         movdqa    %xmm2, matrix_multiplication_$A.0.1(%ecx,%edx,4) #17.7
397         movdqa    16+matrix_multiplication_$B.0.1(%edi,%edx,4), %xmm2 #17.25
398         pmuludq   %xmm2, %xmm1                                   #17.32
399         psrlq     $32, %xmm2                                     #17.32
400         pmuludq   %xmm3, %xmm2                                   #17.32
401         pand      %xmm0, %xmm1                                   #17.32
402         psllq     $32, %xmm2                                     #17.32
403         movdqa    9616+matrix_multiplication_$B.0.1(%edi,%edx,4), %xmm7 #17.25
404         por       %xmm2, %xmm1                                   #17.32
405         movdqa    %xmm4, %xmm2                                   #17.32
406         pmuludq   %xmm7, %xmm2                                   #17.32
407         psrlq     $32, %xmm7                                     #17.32
408         pmuludq   %xmm6, %xmm7                                   #17.32
409         pand      %xmm0, %xmm2                                   #17.32
410         psllq     $32, %xmm7                                     #17.32
411         paddd     16+matrix_multiplication_$A.0.1(%ecx,%edx,4), %xmm1 #17.7
412         por       %xmm7, %xmm2                                   #17.32
413         paddd     %xmm2, %xmm1                                   #17.7
414         movdqa    %xmm1, 16+matrix_multiplication_$A.0.1(%ecx,%edx,4) #17.7
415         addl      $8, %edx                                       #16.7
416         cmpl      %ebx, %edx                                     #16.7
417         jb        ..B1.28       # Prob 99%                       #16.7
418                                 # LOE eax edx ecx ebx esi edi xmm0 xmm3 xmm4 xmm5 xmm
419 ..B1.30:                        # Preds ..B1.28 ..B1.59
420         cmpl      %ebx, %esi                                     #16.7
421         jae       ..B1.34       # Prob 0%                        #16.7
```

```
422                                     # LOE eax ecx ebx esi xmm0
423 ..B1.31:                            # Preds ..B1.30
424         movl      4+matrix_multiplication_$C.0.1(%eax), %edi    #17.34
425         movl      %edi, 248(%esp)                               #17.34
426         movl      matrix_multiplication_$C.0.1(%eax), %edx      #17.34
427         movl      %eax, 224(%esp)                               #17.34
428         movl      228(%esp), %edi                               #17.34
429                                     # LOE edx ecx esi edi xmm0
430 ..B1.32:                            # Preds ..B1.32 ..B1.31
431         movl      matrix_multiplication_$B.0.1(%edi,%esi,4), %ebx #17.25
432         imull     %edx, %ebx                                    #17.32
433         movl      9600+matrix_multiplication_$B.0.1(%edi,%esi,4), %eax #17.25
434         imull     248(%esp), %eax                               #17.32
435         addl      matrix_multiplication_$A.0.1(%ecx,%esi,4), %ebx #17.7
436         addl      %eax, %ebx                                    #17.7
437         movl      %ebx, matrix_multiplication_$A.0.1(%ecx,%esi,4) #17.7
438         incl      %esi                                          #16.7
439         cmpl      244(%esp), %esi                               #16.7
440         jb        ..B1.32       # Prob 99%                      #16.7
441                                     # LOE edx ecx esi edi xmm0
442 ..B1.33:                            # Preds ..B1.32
443         movl      224(%esp), %eax                               #
444         movl      244(%esp), %ebx                               #
445                                     # LOE eax ecx ebx xmm0
446 ..B1.34:                            # Preds ..B1.30 ..B1.33
447         movl      180(%esp), %edx                               #15.4
448         addl      $9600, %eax                                   #15.4
449         incl      %edx                                          #15.4
450         addl      $9600, %ecx                                   #15.4
451         movl      %edx, 180(%esp)                               #15.4
452         cmpl      196(%esp), %edx                               #15.4
453         jb        ..B1.26       # Prob 99%                      #15.4
454                                     # LOE eax ecx ebx xmm0
455 ..B1.35:                            # Preds ..B1.34
456         movl      184(%esp), %edx                               #
457         movl      152(%esp), %esi                               #
458                                     # LOE edx esi xmm0
459 ..B1.36:                            # Preds ..B1.24 ..B1.35
460         incl      %edx                                          #14.3
461         cmpl      144(%esp), %edx                               #14.3
462         jb        ..B1.12       # Prob 99%                      #14.3
463                                     # LOE edx esi xmm0
464 ..B1.37:                            # Preds ..B1.36
465         movl      160(%esp), %eax                               #
466         lea       1(%edx,%edx), %edx                            #14.3
467         movl      %edx, 132(%esp)                               #14.3
468                                     # LOE eax xmm0
469 ..B1.38:                            # Preds ..B1.37 ..B1.62
470         movl      132(%esp), %edx                               #14.3
471         cmpl      128(%esp), %edx                               #14.3
472         ja        ..B1.51       # Prob 0%                       #14.3
473                                     # LOE eax edx dl dh xmm0
474 ..B1.39:                            # Preds ..B1.38
475         cmpl      $0, 196(%esp)                                 #15.4
476         jbe       ..B1.51       # Prob 0%                       #15.4
477                                     # LOE eax edx dl dh xmm0
478 ..B1.40:                            # Preds ..B1.39
479         movl      %eax, %ecx                                    #16.7
```

```
480         movl      $512, %esi                                    #16.7
481         shll      $9, %ecx                                      #16.7
482         xorl      %ebx, %ebx                                    #
483         negl      %ecx                                          #16.7
484         addl      $2400, %ecx                                   #16.7
485         movl      152(%esp), %edi                               #
486         cmpl      $512, %ecx                                    #16.7
487         cmovae    %esi, %ecx                                    #16.7
488         movl      %eax, 160(%esp)                               #
489         lea       (%edi,%edx,4), %esi                           #
490         movl      %esi, 216(%esp)                               #
491         movl      %eax, %esi                                    #
492         shll      $11, %esi                                     #
493         imull     $9600, %edx, %edx                             #
494         movl      156(%esp), %edi                               #
495         addl      %esi, %edi                                    #
496         addl      148(%esp), %esi                               #
497         addl      %edx, %esi                                    #
498         xorl      %edx, %edx                                    #
499         movl      %edi, 212(%esp)                               #
500         movl      %esi, 208(%esp)                               #
501                               # LOE edx ecx ebx xmm0
502 ..B1.41:                      # Preds ..B1.49 ..B1.40
503         cmpl      $16, %ecx                                     #16.7
504         jb        ..B1.58     # Prob 10%                        #16.7
505                               # LOE edx ecx ebx xmm0
506 ..B1.42:                      # Preds ..B1.41
507         movl      216(%esp), %edi                               #17.34
508         movl      %ecx, %eax                                    #16.7
509         movl      %ebx, 192(%esp)                               #17.32
510         xorl      %esi, %esi                                    #16.7
511         movl      208(%esp), %ebx                               #17.32
512         movd      -4+matrix_multiplication_$C.0.1(%edx,%edi), %xmm1 #17.34
513         movl      212(%esp), %edi                               #
514         pshufd    $0, %xmm1, %xmm1                              #17.34
515         movdqa    %xmm1, %xmm2                                  #17.32
516         psrlq     $32, %xmm2                                    #17.32
517         addl      %edx, %edi                                    #
518                               # LOE eax edx ecx ebx esi edi xmm0 xmm1 xmm2
519 ..B1.43:                      # Preds ..B1.43 ..B1.42
520         movdqa    -9600+matrix_multiplication_$B.0.1(%ebx,%esi,4), %xmm3 #17.25
521         movdqa    %xmm1, %xmm4                                  #17.32
522         pmuludq   %xmm3, %xmm4                                  #17.32
523         psrlq     $32, %xmm3                                    #17.32
524         pmuludq   %xmm2, %xmm3                                  #17.32
525         movdqa    -9584+matrix_multiplication_$B.0.1(%ebx,%esi,4), %xmm5 #17.25
526         movdqa    %xmm1, %xmm6                                  #17.32
527         pmuludq   %xmm5, %xmm6                                  #17.32
528         psrlq     $32, %xmm5                                    #17.32
529         pmuludq   %xmm2, %xmm5                                  #17.32
530         pand      %xmm0, %xmm4                                  #17.32
531         psllq     $32, %xmm3                                    #17.32
532         por       %xmm3, %xmm4                                  #17.32
533         pand      %xmm0, %xmm6                                  #17.32
534         paddd     matrix_multiplication_$A.0.1(%edi,%esi,4), %xmm4 #17.7
535         psllq     $32, %xmm5                                    #17.32
536         movdqa    %xmm4, matrix_multiplication_$A.0.1(%edi,%esi,4) #17.7
537         por       %xmm5, %xmm6                                  #17.32
```

```
538         movdqa    -9568+matrix_multiplication_$B.0.1(%ebx,%esi,4), %xmm7 #17.25
539         movdqa    %xmm1, %xmm3                                    #17.32
540         movdqa    -9552+matrix_multiplication_$B.0.1(%ebx,%esi,4), %xmm4 #17.25
541         movdqa    %xmm1, %xmm5                                    #17.32
542         pmuludq   %xmm7, %xmm3                                    #17.32
543         psrlq     $32, %xmm7                                      #17.32
544         pmuludq   %xmm4, %xmm5                                    #17.32
545         pmuludq   %xmm2, %xmm7                                    #17.32
546         psrlq     $32, %xmm4                                      #17.32
547         pand      %xmm0, %xmm3                                    #17.32
548         pmuludq   %xmm2, %xmm4                                    #17.32
549         psllq     $32, %xmm7                                      #17.32
550         pand      %xmm0, %xmm5                                    #17.32
551         psllq     $32, %xmm4                                      #17.32
552         por       %xmm7, %xmm3                                    #17.32
553         por       %xmm4, %xmm5                                    #17.32
554         paddd     16+matrix_multiplication_$A.0.1(%edi,%esi,4), %xmm6 #17.7
555         paddd     32+matrix_multiplication_$A.0.1(%edi,%esi,4), %xmm3 #17.7
556         paddd     48+matrix_multiplication_$A.0.1(%edi,%esi,4), %xmm5 #17.7
557         movdqa    %xmm6, 16+matrix_multiplication_$A.0.1(%edi,%esi,4) #17.7
558         movdqa    %xmm3, 32+matrix_multiplication_$A.0.1(%edi,%esi,4) #17.7
559         movdqa    %xmm5, 48+matrix_multiplication_$A.0.1(%edi,%esi,4) #17.7
560         addl      $16, %esi                                       #16.7
561         cmpl      %ecx, %esi                                      #16.7
562         jb        ..B1.43       # Prob 99%                        #16.7
563                                 # LOE eax edx ecx ebx esi edi xmm0 xmm1 xmm2
564 ..B1.44:                        # Preds ..B1.43
565         movl      192(%esp), %ebx                                 #
566                                 # LOE eax edx ecx ebx xmm0
567 ..B1.45:                        # Preds ..B1.44 ..B1.58
568         cmpl      %ecx, %eax                                      #16.7
569         jae       ..B1.49       # Prob 0%                         #16.7
570                                 # LOE eax edx ecx ebx xmm0
571 ..B1.46:                        # Preds ..B1.45
572         movl      216(%esp), %esi                                 #17.34
573         movl      212(%esp), %edi                                 #
574         movl      %ebx, 192(%esp)                                 #
575         movl      -4+matrix_multiplication_$C.0.1(%edx,%esi), %esi #17.34
576         movl      %edx, 204(%esp)                                 #
577         addl      %edx, %edi                                      #
578         movl      208(%esp), %ebx                                 #
579                                 # LOE eax ecx ebx esi edi xmm0
580 ..B1.47:                        # Preds ..B1.47 ..B1.46
581         movl      -9600+matrix_multiplication_$B.0.1(%ebx,%eax,4), %edx #17.25
582         imull     %esi, %edx                                      #17.32
583         addl      %edx, matrix_multiplication_$A.0.1(%edi,%eax,4) #17.7
584         incl      %eax                                            #16.7
585         cmpl      %ecx, %eax                                      #16.7
586         jb        ..B1.47       # Prob 99%                        #16.7
587                                 # LOE eax ecx ebx esi edi xmm0
588 ..B1.48:                        # Preds ..B1.47
589         movl      204(%esp), %edx                                 #
590         movl      192(%esp), %ebx                                 #
591                                 # LOE edx ecx ebx xmm0
592 ..B1.49:                        # Preds ..B1.48 ..B1.45
593         incl      %ebx                                            #15.4
594         addl      $9600, %edx                                     #15.4
595         cmpl      196(%esp), %ebx                                 #15.4
```

```
596         jb          ..B1.41       # Prob 99%                      #15.4
597                                   # LOE edx ecx ebx xmm0
598 ..B1.50:                          # Preds ..B1.49
599         .byte       141                                           #
600         .byte       118                                           #
601         .byte       0                                             #
602         movl        160(%esp), %eax                               #
603                                   # LOE eax xmm0
604 ..B1.51:                          # Preds ..B1.50 ..B1.38 ..B1.39
605         incl        %eax                                          #14.3
606         cmpl        $5, %eax                                      #14.3
607         jb          ..B1.10       # Prob 99%                      #14.3
608                                   # LOE eax xmm0
609 ..B1.52:                          # Preds ..B1.51
610         movl        200(%esp), %eax                               #14.3
611         incl        %eax                                          #14.3
612         movl        %eax, 200(%esp)                               #14.3
613         cmpl        $5, %eax                                      #14.3
614         jb          ..B1.8        # Prob 99%                      #14.3
615                                   # LOE xmm0
616 ..B1.53:                          # Preds ..B1.52
617         movl        164(%esp), %edx                               #
618         movl        $512, %eax                                    #
619         incl        %edx                                          #14.3
620         cmpl        $5, %edx                                      #14.3
621         jb          ..B1.7        # Prob 99%                      #14.3
622                                   # LOE eax edx xmm0
623 ..B1.54:                          # Preds ..B1.53
624         movl        136(%esp), %esi                               #
625         movl        140(%esp), %edi                               #
626         addl        $4, %esp                                      #21.8
627         lea         168(%esp), %eax                               #21.17
628         pushl       %eax                                          #21.8
629         call        for_cpusec                                    #21.8
630                                   # LOE esi edi
631 ..B1.55:                          # Preds ..B1.54
632         movss       172(%esp), %xmm0                              #22.3
633         movl        $0, 128(%esp)                                 #23.3
634         movl        $10, 160(%esp)                                #23.3
635         movl        $.L_2_STRLITPACK_0, 164(%esp)                 #23.3
636         subss       168(%esp), %xmm0                              #22.3
637         movss       %xmm0, 176(%esp)                              #22.3
638         addl        $24, %esp                                     #23.3
639         lea         104(%esp), %ebx                               #23.3
640         lea         136(%esp), %eax                               #23.3
641         pushl       $32                                           #23.3
642         pushl       %eax                                          #23.3
643         pushl       $.L_2_STRLITPACK_1.0.1                        #23.3
644         pushl       $-2088435968                                  #23.3
645         pushl       $-1                                           #23.3
646         pushl       %ebx                                          #23.3
647         call        for_write_seq_lis                             #23.3
648                                   # LOE ebx esi edi
649 ..B1.56:                          # Preds ..B1.55
650         movss       176(%esp), %xmm0                              #23.3
651         movss       %xmm0, 184(%esp)                              #23.3
652         addl        $12, %esp                                     #23.3
653         lea         172(%esp), %eax                               #23.3
```

```
654         pushl     %eax                                          #23.3
655         pushl     $.L_2_STRLITPACK_2.0.1                        #23.3
656         pushl     %ebx                                          #23.3
657         call      for_write_seq_lis_xmit                        #23.3
658                                   # LOE esi edi
659 ..B1.57:                         # Preds ..B1.56
660         movl      $1, %eax                                      #24.1
661         addl      $636, %esp                                    #24.1
662         popl      %ebx                                          #24.1
663         movl      %ebp, %esp                                    #24.1
664         popl      %ebp                                          #24.1
665         ret                                                     #24.1
666                                   # LOE
667 ..B1.58:                         # Preds ..B1.41                 # Infreq
668         xorl      %eax, %eax                                    #16.7
669         jmp       ..B1.45       # Prob 100%                     #16.7
670                                   # LOE eax edx ecx ebx xmm0
671 ..B1.59:                         # Preds ..B1.26                 # Infreq
672         xorl      %esi, %esi                                    #16.7
673         jmp       ..B1.30       # Prob 100%                     #16.7
674                                   # LOE eax ecx ebx esi xmm0
675 ..B1.60:                         # Preds ..B1.14                 # Infreq
676         xorl      %eax, %eax                                    #16.7
677         jmp       ..B1.18       # Prob 100%                     #16.7
678                                   # LOE eax
679 ..B1.61:                         # Preds ..B1.12                 # Infreq
680         movl      $1, %ebx                                      #
681         jmp       ..B1.24       # Prob 100%                     #
682                                   # LOE edx ebx esi xmm0
683 ..B1.62:                         # Preds ..B1.10                 # Infreq
684         movl      $1, 132(%esp)                                 #
685         jmp       ..B1.38       # Prob 100%                     #
686         .align    16,0x90
687                                   # LOE eax xmm0
688 # mark_end;
```

ASSINATURAS

_____
Cristian Fernando Flores Castañeda

_____
Prof. Dr. Nicolas Maillard

REFERÊNCIAS

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compilers. Principles, Techniques and Tools**. [S.l.]: Addison Wesley, 1986.

AMBLER, S. W.; SADALAGE, P. J. **Refactoring Databases**: evolutionary database design. [S.l.]: Addison-Wesley, 2006.

ASTELS, D. Refactoring with UML. In: INT'L CONF. EXTREME PROGRAMMING AND FLEXIBLE PROCESSES IN SOFTWARE ENGINEERING XP, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.67–70. Alghero, Sardinia, Italy.

BACON, D. F.; GRAHAM, S. L.; SHARP, O. J. Compiler transformations for high-performance computing. **ACM Comput. Surv.**, New York, NY, USA, v.26, p.345–420, December 1994.

BEATON, W.; RIVIERES, J. **Eclipse Platform Technical Overview**. [S.l.]: The Eclipse Foundation, 2006.

BEVERLY SANDERS ERIK DEUMENS, V. L.; PONTON, M. Refactoring a Language for Parallel Computational Chemistry. In: SECOND WORKSHOP ON REFACTORING TOOLS, 2008. **Anais...** [S.l.: s.n.], 2008.

BOEHM, A. M.; SEIPEL, D.; SICKMANN, A.; WETZKA, M. Squash: a tool for analyzing, tuning and refactoring relational database applications. In: APPLICATIONS OF DECLARATIVE PROGRAMMING AND KNOWLEDGE MANAGEMENT: 17TH INTERNATIONAL CONFERENCE, INAP 2007, AND 21ST WORKSHOP ON LOGIC PROGRAMMING, WLP 2007, WüRZBURG, GERMANY, OCTOBER 4-6, 2007, REVISED SELECTED PAPERS, 2009, Berlin, Heidelberg. **Anais...** Springer-Verlag, 2009. p.82–98.

BOGER, M.; STURM, T.; FRAGEMANN, P. Refactoring Browser for UML. In: OBJECTS, COMPONENTS, ARCHITECTURES, SERVICES, AND APPLICATIONS FOR A NETWORKEDWORLD: INTERNATIONAL CONFERENCE NETOBJECTDAYS, NODE 2002, SPRINGER-VERLAG LNCS 2591/2003, 2002. **Anais...** [S.l.: s.n.], 2002. p.77–81.

BOIS, B. D. **A Study of Quality Improvements by refactoring**. 2006. Tese (Doutorado em Ciência da Computação) — University of Antwerp.

BONIATI, B. B.; CHARAO, A. S. Refatoração de Programas Fortran de Alto Desempenho. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 2008, Santa Cruz do Sul, Brasil. **Anais...** [S.l.: s.n.], 2008.

CARR, S.; DING, C.; SWEANY, P. Improving Software Pipelining With Unroll-and-Jam. In: IN PROCEEDINGS OF THE 29TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, MAUI, HI, 1996. **Anais...** [S.l.: s.n.], 1996. p.183–192.

CARR, S.; GUAN, Y. Unroll-and-Jam Using Uniformly Generated Sets. In: IN PROCEEDINGS OF THE 30TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE (MICRO, 1997. **Anais...** IEEE Computer Society, 1997. p.349–357.

CARR, S.; KENNEDY, K. Improving the ratio of memory operations to floating-point operations in loops. **ACM Trans. Program. Lang. Syst.**, New York, NY, USA, v.16, n.6, p.1768–1810, 1994.

CHEN, L.; XU, B.; ZHOU, T.; ZHOU, Y. Applying Generalization Refactoring to Java Generic Programs. In: WSCS '08: PROCEEDINGS OF THE IEEE INTERNATIONAL WORKSHOP ON SEMANTIC COMPUTING AND SYSTEMS, 2008, Washington, DC, USA. **Anais...** IEEE Computer Society, 2008. p.35–39.

CHEN N., O. J. **Photran 7.0 Developer's Guide**. [S.l.: s.n.], 2010.

CRAWFORD, I. L.; WADLEIGH, K. R. **Software Optimization for High Performance Computers**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

DE, V. **A Foundation for Refactoring Fortran 90 in Eclipse**. EUA: Urbana-Champaign, 2004.

DONGARRA, J.; LUSZCZEK, P.; PETITET, A. The LINPACK Benchmark: past, present and future. **Concurrency and Computation: Practice and Experience**, [S.l.], v.15, n.9, p.803–820, 2003.

DUCASSE, S.; RIEGER, M.; DEMEYER, S. A language independent approach for detecting duplicated code. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 1999. **Proceedings...** [S.l.: s.n.], 1999. p.109–118.

ETTINGER, R. Refactoring via Program Slicing and Sliding. In: ICSM, 2007. **Anais...** [S.l.: s.n.], 2007. p.505–506.

F. IRIGOIN, P. J.; TRIOLET, R. **Semantical Interprocedural Parallelization**: an overview of the pips project. Cologne, 1991.

FANTA, R.; RAJLICH, V. Restructuring legacy C code into C++. In: INT'L CONF. SOFTWARE MAINTENANCE (ICSM), 1999. **Proceedings...** IEEE Computer Society Press, 1999. p.77–85.

FAZENDA, A. L.; DEMERVAL, S. M.; ENARI, E. H.; PANETTA, J.; RODRIGUES, L. F. **First Time User Guide (BRAMS Version 4.2)**. 2011.

FOWLER, M. **Refactoring**: improving the design of existing code. Boston, MA, USA: Addison-Wesley, 1999.

GARRIDO, A.; JOHNSON, R. Challenges of refactoring C programs. In: IWPSE '02: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON PRINCIPLES OF SOFTWARE EVOLUTION, 2002, New York, NY, USA. **Anais...** ACM, 2002. p.6–14.

GARRIDO, A.; JOHNSON, R. Refactoring C with Conditional Compilation. **Automated Software Engineering, International Conference on**, Los Alamitos, CA, USA, v.0, p.323, 2003.

GOUGH, B. J.; STALLMAN, R. M. **An Introduction to GCC**. [S.l.]: Network Theory Ltd., 2004.

GRAF, E.; ZGRAGGEN, G.; SOMMERLAD, P. Refactoring support for the C++ development tooling. In: COMPANION TO THE 22ND ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS AND APPLICATIONS COMPANION, 2007, New York, NY, USA. **Anais...** ACM, 2007. p.781–782. (OOPSLA '07).

GRISWOLD, W. G.; CHEN, M. I.; BOWDIDGE, R. W.; MORGENTHALER, J. D. Tool support for planning the restructuring of data abstractions in large systems. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.21, p.33–45, October 1996.

HERMANNS, M. **Parallel Programming in Fortran 95 using OpenMP**. [S.l.: s.n.], 2002.

JOHNSON, R.; FOOTE, B.; OVERBEY, J.; XANTHOS, S. **Changing the Face of High-Performance Fortran Code**. 2005.

JONES, T. **Conhecendo o GCC 4**. [S.l.]: Emulex Corporation, 2008.

KERIEVSKY, J. **Refactoring to Patterns**. [S.l.]: Addison-Wesley, 2004.

KIEZUN, A.; ERNST, M. D.; TIP, F.; FUHRER, R. M. Refactoring for Parameterizing Java Classes. In: ICSE '07: PROCEEDINGS OF THE 29TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2007, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.437–446.

KOFFMANN, E.; FRIEDMAN, F. **Fortran**. [S.l.]: Addison Wesley, 2006.

KOSEKI, A.; KOMASTU, H.; FUKAZAWA, Y. A method for estimating optimal unrolling times for nested loops. In: ISPAN '97: PROCEEDINGS OF THE 1997 INTERNATIONAL SYMPOSIUM ON PARALLEL ARCHITECTURES, ALGORITHMS AND NETWORKS (ISPAN, 1997. **Anais...** IEEE CS Press, 1997. p.376.

LEE B., S. D. **Eclipse Project CDT (C/C++) Plugin Tutorial**. 2004.

LI, H. **Refactoring Haskell Programs**. 1992.

LI, H.; THOMPSON, S. Tool support for refactoring functional programs. In: PEPM '08: PROCEEDINGS OF THE 2008 ACM SIGPLAN SYMPOSIUM ON PARTIAL EVALUATION AND SEMANTICS-BASED PROGRAM MANIPULATION, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.199–203.

LIU, J.; BATORY, D.; LENGAUER, C. Feature oriented refactoring of legacy applications. In: ICSE '06: PROCEEDINGS OF THE 28TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.112–121.

MATTHIAS RIEGER BART VAN ROMPAEY, K. M.; LIEVIER, P. Refactoring for Performance: an experience report. In: THIRD INTERNATIONAL ERCIM SYMPOSIUM ON SOFTWARE EVOLUTION, 2007. **Anais...** [S.l.: s.n.], 2007.

MENS, T.; DEMEYER, S.; BOIS, B. D.; STENTEN, H.; GORP, P. V. Refactoring: current research and future trends. **Electronic Notes in Theoretical Computer Science**, [S.l.], v.82, n.3, p.483–499, December 2003.

MENS, T.; TAENTZER, G.; RUNGE, O. Analyzing Refactoring Dependencies Using Graph Transformation. **Software and Systems Modeling**, [S.l.], 2007.

MENS, T.; TOURWÉ, T. A Survey of Software Refactoring. **IEEE Trans. Softw. Eng.**, Piscataway, NJ, USA, v.30, n.2, p.126–139, 2004.

MURPHY-HILL, E.; BLACK, A. P. Breaking the barriers to successful refactoring: observations and tools for extract method. In: ICSE '08: PROCEEDINGS OF THE 30TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.421–430.

NOVILLO, D. GCC - An Architectural Overview, Current Status, and Future Directions. In: LINUX SYMPOSIUM, 2006, Ottawa, Canada. **Anais...** [S.l.: s.n.], 2006. p.187–199.

NYHOFF, L.; LEESTMA, S. **Fortran 90 for Engineers and Scientists**. [S.l.]: Prentice-Hall, 1997.

OPDYKE, W. F. **Refactoring object-oriented frameworks**. 1992. Tese (Doutorado em Ciência da Computação) — , Champaign, IL, USA.

OVERBEY, J. L.; JOHNSON, R. E. Generating Rewritable Abstract Syntax Trees. In: SOFTWARE LANGUAGE ENGINEERING: FIRST INTERNATIONAL CONFERENCE, SLE 2008, TOULOUSE, FRANCE, SEPTEMBER 29-30, 2008. REVISED SELECTED PAPERS, 2009, Berlin, Heidelberg. **Anais...** Springer-Verlag, 2009. p.114–133.

OVERBEY, J. L.; NEGARA, S.; JOHNSON, R. E. Refactoring and the evolution of Fortran. In: SECSE '09: PROCEEDINGS OF THE 2009 ICSE WORKSHOP ON SOFTWARE ENGINEERING FOR COMPUTATIONAL SCIENCE AND ENGINEERING, 2009, Washington, DC, USA. **Anais...** IEEE Computer Society, 2009. p.28–34.

OVERBEY, J.; XANTHOS, S.; JOHNSON, R.; FOOTE, B. Refactorings for Fortran and high-performance computing. In: SE-HPCS '05: PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR HIGH PERFORMANCE COMPUTING SYSTEM A APPLICATIONS, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.37–39.

RASMUSSEN, C. E.; SQUYRES, J. M. A Case for New MPI Fortran Bindings. In: EUROPEAN PVM/MPI USERS' GROUP MEETING, 12., 2005, Sorrento, Italy. **Proceedings...** [S.l.: s.n.], 2005.

RATZINGER, J.; FISCHER, M. Improving evolvability through refactoring. In: IN MSR '05: PROCEEDINGS OF THE 2005 INTERNATIONAL WORKSHOP ON MINING SOFTWARE REPOSITORIES, 2005. **Anais...** ACM Press, 2005. p.1–5.

RISSETI, G.; CHARAO, A. S. **Catálogo de Refatorações para a Evolução de Programas em Linguagem Fortran**. 2011. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Maria.

ROBERTS, D.; BRANT, J.; JOHNSON, R. A refactoring tool for Smalltalk. **Theor. Pract. Object Syst.**, New York, NY, USA, v.3, n.4, p.253–263, 1997.

ROBERTS, D.; BRANT, J.; JOHNSON, R.; OPDYKE, B. An Automated Refactoring Tool. In: ICAST 1996, CHICAGO, IL, 1996. **Proceedings...** [S.l.: s.n.], 1996.

SEREBRENIK, A.; SCHRIJVERS, T.; DEMOEN, B. Improving prolog programs: refactoring for prolog. **Theory Pract. Log. Program.**, New York, NY, USA, v.8, n.2, p.201–215, 2008.

SILBER, G.-A.; DARTE, A. **The Nestor library**: a tool for implementing fortran source to source transformations. 1998.

SONG, Y.; LIN, Y. Unroll-and-jam for imperfectly-nested loops in DSP applications. In: COMPILERS, ARCHITECTURE, AND SYNTHESIS FOR EMBEDDED SYSTEMS, 2000., 2000, New York, NY, USA. **Proceedings...** ACM, 2000. p.148–156. (CASES '00).

STALLMAN, R. **The GCC Internals manual**. [S.l.]: Free Software Foundation, 2011.

STROGGYLOS, K.; SPINELLIS, D. Refactoring: does it improve software quality? In: INTERNATIONAL WORKSHOP ON SOFTWARE QUALITY, 5., 2007. **Anais...** ACM Press, 2007. p.1–6.

VAN DE VANTER, M. L.; POST, D. E.; ZOSEL, M. E. HPC needs a tool strategy. In: SE-HPCS '05: PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR HIGH PERFORMANCE COMPUTING SYSTEM APPLICATIONS, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.55–59.

VICHARE, A. **The Conceptual Structure of GCC**. [S.l.]: Indian Institute of Technology, 2008.

WALKO, R. L.; AVISSAR, R. The Ocean-Land-Atmosphere Model (OLAM). Part I: shallow-water tests. **Monthly Weather Review**, [S.l.], v.136, n.11, p.4033–4044, 2008.

WATSON, G. R.; DEBARDELEBEN, N. A. Developing Scientific Applications Using Eclipse. **Computing in Science and Engineering**, Los Alamitos, CA, USA, v.8, n.4, p.50–61, 2006.

WOLFE, M. **High Performance Compilers For Parallel Computing**. Redwood City, CA, USA: Addison-Wesley Publishing Company, 1996.

ZINGIRIAN, N.; MARESCA, M. Finding the Optimal Unroll-and-Jam. In: HPCN EUROPE '99: PROCEEDINGS OF THE 7TH INTERNATIONAL CONFERENCE ON HIGH-PERFORMANCE COMPUTING AND NETWORKING, 1999, London, UK. **Anais...** Springer-Verlag, 1999. p.633–642.