Automação de Refatorações para Programas Fortran de Alto Desempenho

Bruno Batista Boniati, Andrea Schwertner Charão, Benhur de Oliveira Stein
Universidade Federal de Santa Maria
Programa de Pós-Graduação em Informática
Rio Grande do Sul, RS, Brasil
brunoboniati@gmail.com, {andrea, benhur}@inf.ufsm.br

Resumo

Refatoração é uma técnica de engenharia de software que objetiva aplicar melhorias internas no código-fonte de aplicações, sem que isso influencie no seu comportamento. É uma técnica amplamente empregada em código orientado a objetos e utilizada por algumas metodologias de desenvolvimento. Na computação de alto desempenho, a refatoração é uma técnica pouco explorada, sendo que grande parte código legado de programas de alto desempenho está escrita em linguagens não orientadas a objetos. Neste trabalho, explora-se a automatização de técnicas de refatoração e sua aplicação em códigos de alto desempenho escritos em linguagem imperativa Fortran. São identificadas oportunidades de melhorar a legibilidade e o design do código sem afetar o desempenho, além de reestruturações de código quem podem representar ganho de desempenho. As técnicas estudadas e automatizadas são utilizadas em aplicações reais, como meio de validar a abordagem. A automatização das técnicas é feita estendendo as funcionalidades do IDE Photran, um plugin para Eclipse, cujo funcionamento também será abortado neste texto.

1. Introdução

Aplicações computacionais de alto desempenho são comuns em áreas que unem física, matemática, engenharia e computação. Este tipo de *software* é muitas vezes resultante de anos de pesquisas e otimizações, realizadas com a colaboração de diferentes equipes multidisciplinares, em diferentes tempos. Do ponto de vista de sua funcionalidade e desempenho, o *software* atinge seu objetivo ao produzir resultados no menor tempo possível. Porém, aplicações de alto desempenho também são propensas a problemas combatidos nos ciclos de desenvolvimento de *software*, tais como o uso de construções de linguagem obsoletas ou de práticas desaconselhadas, que acabam dificultando a manutenção e a evolução do *software*.

Grande parte do código legado de aplicações de alto desempenho está escrito em Fortran, uma das primeiras linguagens de programação imperativas e com ampla utilização no meio científico. Embora a linguagem tenha evoluído ao longo do tempo, muitas aplicações ainda seguem antigos padrões de programação. Além disso, as técnicas e ferramentas de desenvolvimento em Fortran não tiveram o mesmo avanço daquelas destinadas a linguagens mais recentes, principalmente as orientadas a objetos.

Dado este cenário, o presente trabalho objetiva explorar a abordagem conhecida como refatoração, do inglês *refactoring*, para prover melhorias internas no código-fonte de aplicações Fortran de alto desempenho. Esta abordagem é amplamente empregada no desenvolvimento orientado a objetos e consiste na realização de transformações para aprimorar um código existente, sem que isso altere suas funcionalidades. Recursos automatizados de refatoração são disponíveis em modernos ambientes de desenvolvimento integrado (*Integrated Development Environments* - IDEs), mas são pouco explorados no contexto de aplicações Fortran de alto desempenho. Para alcançar o objetivo proposto, identificou-se algumas técnicas de refatoração aplicáveis no contexto em questão. Estas técnicas foram automatizadas e sua mecânica de funcionamento foi aplicada ao IDE Eclipse, a fim de facilitar a tarefa de refatoração. A implementação da automatização é feita como uma extensão à ferramenta Photran [19], um *plugin* para o Eclipse, de código aberto escrito em Java, voltado para o desenvolvimento de programas em Fortran.

O restante do texto será organizado da seguinte forma: a seção 2 aborda aspectos importantes relacionados à refatoração, sua aplicação a código imperativo escrito em Fortran e sua automatização a partir da ferramenta Photran. A seção 3 detalha as técnicas automatizadas durante a pesquisa, sua forma de atuação e eventuais limitações. A seção 4 é destinada à avaliação dos resultados da utilização das técnicas automatizadas em código Fortran. Por fim, na seção 5 são apresentadas as considerações finais e sugestões de trabalhos futuros.

2. Refatoração

De forma geral, a refatoração consiste em aplicar modificações no *software* sem afetar seu comportamento aparente. Este conceito começou a ser explorado em profundidade em 1992 por William Opdyke, em sua tese de doutorado [16], no qual foram catalogadas vinte e três técnicas primitivas de refatoração. Desde então, técnicas de refatoração vêm sendo agrupadas, catalogadas e documentadas de forma que possam ser aplicadas sistematicamente, melhoradas e compartilhadas [11, 1]. A refatoração pode ser realizada de forma manual pelo próprio desenvolvedor ou então automatizada através de ferramentas específicas.

Refatorar significa melhorar qualidades não funcionais do *software*, como simplicidade, flexibilidade e desempenho. Objetiva-se alcançar um código mais limpo, simples e elegante, permitindo que componentes mais simples ou expressões mais eficientes sejam utilizadas. São alguns exemplos de refatorações: mudança do nome de variáveis, alterações em interfaces dos subprogramas, encapsulamento de código repetido em novos subprogramas, generalização de subprograms muito específicos e evolução de código por meio de construções mais recentes.

Uma ação de refatoração deve ter a propriedade de equivalência semântica de operações e referências, isto é, um programa deve produzir a mesma saída para uma dada entrada, antes e depois de se aplicar a refatoração [7]. Para garantir essas propriedades é importante ter um conjunto sólido de testes para verificar o funcionamento correto da funcionalidade a ser refatorada. Testes automatizados auxiliam a detecção de erros que por acaso venham a ser adicionados ao *software*.

Técnicas de refatoração podem ser aplicadas a outros artefatos do *software* que não o código simplesmente. Alguns trabalhos abordam o uso de refatorações em diagramas UML [9, 13] utilizando refatorações automatizadas e integradas em IDEs. Na área de banco de dados, a refatoração tem sido utilizada no contexto da evolução de esquemas relacionais [5]. Um dos grandes desafios é definir mecanismos para manter a consistência entre diferentes artefatos (implementação e modelo) de *software* com o uso de refatoração.

Em relação ao **desempenho** da aplicação, ao tornar o código mais legível pode-se identificar ajustes na aplicação que venham a ser sensíveis ao desempenho [11]. Normalmente os gargalos de desempenho estão localizados em partes específicas de uma aplicação [14], de modo que as otimizações têm impacto significativo apenas em algumas partes do código-fonte. Técnicas de refatoração podem contribuir no sentido de manter o código mais claro e simples, possibilitando a identificação de tais oportunidades e permitindo ao programador concentrar esforços de otimização (geralmente custosos) em pontos em que realmente representarão ganho de desempenho.

Rieger et al. [20] caracterizam situações nas quais a refatoração pode contribuir com o ganho de desempenho. Seu trabalho explora refatorações que substituem construções puramente orientadas a objetos por uma solução híbrida, transformando algumas abstrações em tipos primitivos. As técnicas de refatoração aplicadas consideram também a arquitetura e o ambiente onde a aplicação será executada, buscando otimizar o uso de memória e diminuir o tempo de inicialização da aplicação alvo.

Algumas construções de código ligadas ao desempenho são complexas de serem aplicadas e gerenciadas manualmente. Neste caso, a utilização de ferramentas automatizadas ganha ainda mais importância. Overbey et al. [19] propõem a utilização de um *framework* e a extensão de um IDE para prover uma infraestrutura para a automatização de refatorações. A ferramenta proposta é utilizada em trabalhos subsequentes [18] para prover a evolução de código Fortran em aplicações de alto desempenho, considerando algumas refatorações automatizadas.

Em [17] é apresentado uma ferramenta para geração de árvores sintáticas abstratas manipuláveis (*Rewritable Abstract Syntax Trees*) um requisito fundamental para a automatização de técnicas de refatoração. Neste mesmo trabalho descreve-se a geração de um *parser* a partir de uma gramática formal, permitindo a partir dele manipular as entidades do código-fonte (incluíndo, removendo ou alterando elementos) e mantendo sua formatação original.

2.1. Refatoração e Fortran

Fortran (FORmula TRANslation) é uma linguagem de programação imperativa, cuja primeira versão foi apresentada em 1957, por uma equipe dirigida por John Backus. É a linguagem de programação pioneira em apresentar uma semântica de alto nível e utilizar um compilador. A aplicação principal de Fortran é a computação científica, predominante em áreas de aplicação como matemática, física, engenharia e meteorologia [3, 8]. Mesmo depois de mais de cinquenta anos de sua primeira versão, ainda é altamente utilizada em aplicações científicas de alto custo computacional.

Desempenho é uma palavra-chave que explica a perpetuação histórica de Fortran. A linguagem sofreu várias revisões ao longo do tempo, ganhou bibliotecas muito ricas e altamente otimizadas. Considerando a computação de alto desempenho, uma de suas vantagens é a existência de operações para tratamento de dados multidimensionais que normalmente não são encontradas de forma nativa em outras linguagens de programação [3]. Fortran também suporta a utilização de bibliotecas para programação concorrente e distribuída como os padrões OpenMP e MPI.

A perpetuação histórica de Fotran também lhe traz o

ônus em se manter viva a semântica e gramática de uma linguagem com mais de meio século de história. Fortran passou por várias mudanças de versões (Fortran I, II, III, IV, 66, 77, 90, 95, 2003, 2008) [18] e preserva procedimentos pouco recomendados (desvios rotulados, declarações implícitas, etc.). Construções obsoletas tornam complexo o entendimento do código, o que dificulta a alteração ou adição de funcionalidades sem comprometer o funcionamento da aplicação.

Fortran não utiliza o conceito de palavras reservadas, o que agrega dificuldades para ferramentas de análise de código [8]. É possível encontrar no código Fortran variáveis com nomes de comandos da linguagem como `if` ou `do`, o que faz com que o *parsing* de código tenha que considerar o contexto de utilização de um termo para definir a que entidade o mesmo pertence.

A refatoração de código-fonte escrito em linguagens não orientadas a objeto é mais difícil, pois fluxos de controle e de dados são fortemente interligados e, por causa disso, as reestruturações são limitadas ao nível de função ou bloco de código [15]. Entretanto há um considerável conjunto de técnicas que se aplicam de forma independente do paradigma de desenvolvimento, atuando principalmente sobre subprogramas, blocos condicionais e de repetição e nomes de entidades (variáveis, subprogramas, etc.).

2.2. Photran

Photran é uma ferramenta de apoio ao desenvolvimento de código Fortran, baseada na plataforma Eclipse e escrita em linguagem Java. Nasceu no grupo de pesquisas em arquiteturas de *software* (SAG, *Software Architecture Group*) que é coordenado pelo professor Ralph Johnson na Universidade Illinois em Urbana-Champaign (berço de muitas técnicas de refatoração e de aplicações para automatização de tais técnicas). Atualmente, Photran é mantido pelo projeto Eclipse, como um de seus *plugins* oficiais. Possui código-fonte livre, boa documentação e uma ativa comunidade de usuários.

Essa ferramenta estende as funcionalidades de outro *plugin* para Eclipse, o CDT (*C,C++ Development Tools*) [12], e provê recursos de um editor visual de código Fortran (indicação de sintaxe, sugestão de código, assistentes para criação de projetos, etc.) [19]. Photran permite a edição de código Fortran em formato fixo (versão 77 e anteriores) ou formato livre (versão 90 e posteriores).

Um dos objetivos do projeto é de disponibilizar um *framework* que possibilite o rápido desenvolvimento de ações de refatoração para código Fortran, reutilizando a infraestrutura provida pelo Eclipse [8]. Para isso, utiliza-se de *Rewritable ASTs*, ou seja, árvores sintáticas abstratas (AST, *Abstract Syntax Tree*) que podem ser manipuladas. Isso é possível em função do *parser* utilizado pelo *plugin* Photran, que além de navegar na estrutura da AST, permite a movimentação, remoção e adição de nós.

A viabilidade de disponibilização de ações de refatoração para código Fortran é comprovada pela própria ferramenta, que já disponibiliza algumas ações simples que podem ser utilizadas: *rename*, *introduce implit none*, *move saved variables to global common block* e *replace obsolete operators*. Na seção seguinte, descrevem-se como as ações de refatoração podem ser automatizadas utilizando-se o *framework* Eclipse/Photran.

3. Automatização de Refatorações

A automatização de ações de refatoração requer a existência de um analisador sintático específico para a linguagem de programação com a qual se deseja trabalhar. A análise sintática (*parsing*) se constitui de um algoritmo que decompõe as senteças do código-fonte a partir de uma gramática formal e constrói uma árvore gramatical [4]. Um analisador sintático gera uma sequência de derivação, ou seja, uma árvore sintática abstrata (AST). Uma AST é uma estrutura hierárquica em que seus nós decompõem-se em conjuntos de nós filhos, sendo que os nós terminais representam *tokens* (comandos, expressões, operadores, etc.) do código-fonte.

Além de prover métodos para construção e manipulação da AST, o Photran também disponibiliza uma estrutura denominada VPG (*Virtual Program Graph*). Pode-se entender a VPG como um conjunto de arestas que ligam os nós de uma AST de forma não hierarquizada. Na figura 1, é possível observar um pseudo-código e sua AST correspondente. As linhas que ligam a utilização da variável `i` à sua declaração e essa ao escopo (`PROGRAMA`) são exemplos de informações obtidas por meio da VPG. A consulta à VPG fornece as relações (ligações, dependências, derivações) dos nós da AST.

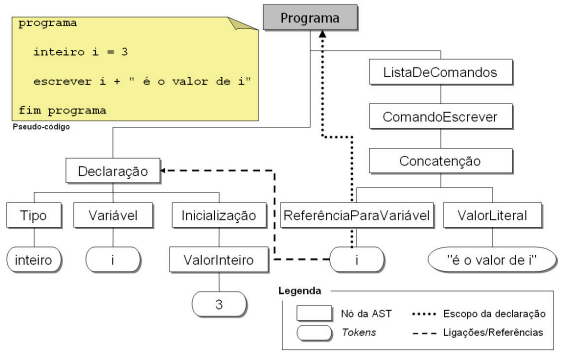

Figura 1. Pseudo-código, AST e VPG

A construção de ações de refatoração para a ferramenta

Photran é feita a partir de um conjunto base de classes que são estendidas e da implementação de métodos de pré-validação, manipulação e pós-validação da AST. A utilização deste tipo de ferramenta abstrai do programador a ação direta sobre o código-fonte, respeitando eventuais dependências e interligações entre entidades do mesmo.

São três as classes que precisam ser estendidas e implementadas para a disponibilização de uma ação de refatoração no Photran [6]:

- **AbstractFortranRefactoringActionDelegate**: é responsável por receber a chamada do usuário e associar a ação de refatoração com seu respectivo assistente.
- **AbstractFortranRefactoringWizard**: é o assistente (*wizard*) da refatoração. É comum que uma ação de refatoração solicite ao usuário alguma informação adicional para que possa ser executada. Neste caso, essa classe se responsabiliza pela construção gráfica da janela do assistente.
- **FortranRefactoring**: representa a ação de refatoração em si. É uma classe específica do Photran que por sua vez estende a classe *Refactoring*, que faz parte do *framework* do Eclipse.

Para a classe descendente de `FortranRefactoring`, quatro métodos precisam ser codificados. O primeiro, `getName()`, é responsável por fornecer o título da refatoração para que seja utilizado em janelas e caixas de diálogo (do Eclipse). O segundo, `doCheckInitialConditions` e o terceiro, `doCheckFinalConditions` são invocados antes de iniciar a ação e após sua conclusão, respectivamente. Podem ser utilizados para a realização de validações prévias e finais. Ambos podem propagar uma exceção da classe `PreconditionFailure`, indicando que alguma condição para a ação de refatoração não foi satisfeita e que a mesma não será realizada.

O método `doCreateChange` (o quarto método que precisa ser codificado) é onde se concentra a ação de refatoração em si. Nele são implementadas as ações que realizam modificações na AST do código-fonte. Ao término de tais modificações, este método chama dois outros métodos: `addChangeFromModifiedAST` (para adicionar as modificações feitas à AST utilizada pelo Photran) e `releaseAllASTs` (para forçar a atualização dos controles visuais do Photran).

Uma vez que as classes estão implementadas, a última etapa consiste em configurar o arquivo `manifest` (*plugin.xml*) adicionando a ele as novas classes e métodos que serão responsáveis pela ação de refatoração. Essas configurações informam ao Eclipse que seus pontos de extensão foram implementados, permitindo associar as ações de refatoração ao menu principal do Photran assim como aos menus de contexto e teclas de atalho.

Durante o desenvolvimento deste trabalho, foram automatizadas algumas técnicas de refatoração aplicadas a características da linguagem Fortran, utilizando-se o *framework* Photran. As técnicas selecionadas possuem diferentes objetivos:

- Evolução da linguagem (refatoração de código escrito para determinada versão de Fortran, substituindo-o por construções de uma versão mais recente);
- Organização do código (refatorações que afetam apenas aspectos ligados ao design do código-fonte, facilitando a manutenção de códigos complexos);
- Exploração de construções de código que oferecem melhor desempenho;

A técnica de substituição de operadores obsoletos (*Replace Obsolete Operators*), que foi desenvolvida durante essa pesquisa, já encontra-se integrada à versão oficial da ferramenta Photran. No restante dessa seção são detalhadas as implementações das demais técnicas de refatoração que foram automatizadas: introdução de `INTENT`, extração de subprograma e desenrolamento de laço de repetição.

3.1. Introduzir o atributo INTENT

O atributo `INTENT` é utilizado para indicar como um parâmetro é passado para um subprograma (`subroutine` ou `function`) em Fortran. Existem três formas de marcar um parâmetro com o atributo `INTENT`:

- `IN`: o atributo somente é referenciado;
- `OUT`: o atributo somente é alterado;
- `INOUT`: o atributo é referenciado e alterado;

No caso de `INTENT(IN)`, admite-se o uso de expressões para passagem de parâmetros, porém, a utilização de `INTENT OUT` ou `INOUT` somente deve ser feita com a indicação de uma variável. Informar o compilador como o programa pretende usar os parâmetros lhe permite captar erros de programação e utilizar melhores estratégias para geração do código executável (passagem por valor ou por referência). Um programa que utiliza `INTENT` possibilita ao desenvolvedor conhecer, ao ler sua declaração, a intenção de seus parâmetros, o que também favorece a legibilidade do código.

A automatização da refatoração *Introduce INTENT* possui dois requisitos básicos: o primeiro é a escolha por parte do programador de uma subrotina ou função na qual a refatoração irá atuar. O segundo é de que o escopo do

subprograma a ser refatorado seja `implict none`, ou seja, exija a declaração explícita dos tipos de dados utilizados[1]. A declaração explícita pode ser conseguida através da refatoração *Introduce Implict None* (disponível na ferramenta Photran).

A mecânica da refatoração deve detectar todas as declarações do subprograma que são argumentos do mesmo e verificar no corpo do subprograma que tipo de ação estes argumentos sofrem. As ações podem ser basicamente duas: referência (em expressões, por exemplo) ou alteração (atribuição e leitura). Uma vez detectados os parâmetros e as ações que os mesmos sofrem, é possível inferir o tipo de `INTENT`. Se o argumento somente é referenciado, é `INTENT(IN)`; se somente é alterado então é `INTENT(OUT)`; por fim, se o argumento é alterado e referenciado, é `INTENT(INOUT)`.

Quando existem chamadas a outros subprogramas (ou seja, um subprograma chama outro), é preciso respeitar os parâmetros `INTENT` de todos os subprogramas que são chamados por daquele que está sendo refatorado. Para tanto, antes de iniciar a refatoração (na fase de pré-validação), faz-se uma leitura em todos os arquivos envolvidos com o projeto no qual a refatoração está sendo aplicada e monta-se uma lista de interfaces de subprogramas, indicando para cada parâmetro o tipo de `INTENT` que o mesmo utiliza. Em não se detectando o tipo, o mesmo é considerado `INOUT`.

A figura 2 ilustra a refatoração *Introduce Intent* aplicada a um típico código de subprograma Fortran. No lado esquerdo da figura tem-se o código original, uma função `area`, que recebe três parâmetros: `h`, `b` e `r` (não declarados com o atributo `INTENT`). No lado direito da figura (código refatorado) é possível observar que a refatoração detectou o tipo de `INTENT` de cada parâmetro e realizou a alteração. A variável `b` poderia ser considerada `INTENT(OUT)` uma vez que é alterada antes de ser referenciada. Para detectar corretamente o tipo de `INTENT` seria necessária a utilização de um grafo de controle, que nessa versão da refatoração não existe.

3.2. Extrair Subprograma

A composição de subprogramas agrupa um conjunto de refatorações primitivas que em geral atuam sobre códigos longos em demasia, que são divididos em porções menores e empacotados em subprogramas. A principal problemática de longos trechos de código é a excessiva informação que fica ocultada pela complexidade da lógica associada [11].

A maior complexidade da técnica *extract subroutine* são as variáveis existentes no corpo do código a ser extraído. É preciso uma análise inicial para identificar se estas são utilizadas exclusivamente no trecho de código que está sendo extraído ou se são referenciadas além dele. No primeiro caso (mais simples) a variável é tratada como local e precisa apenas ser declarada na nova subrotina. Uma variável que é acessada fora do bloco que está sendo extraída precisa ser transformada em um parâmetro.

[1]Fortran permite que as variáveis utilizadas não sejam previamente declaradas, neste caso considerando que variáveis que iniciem com as letras I..N sejam do tipo INTEGER e as demais do tipo REAL.

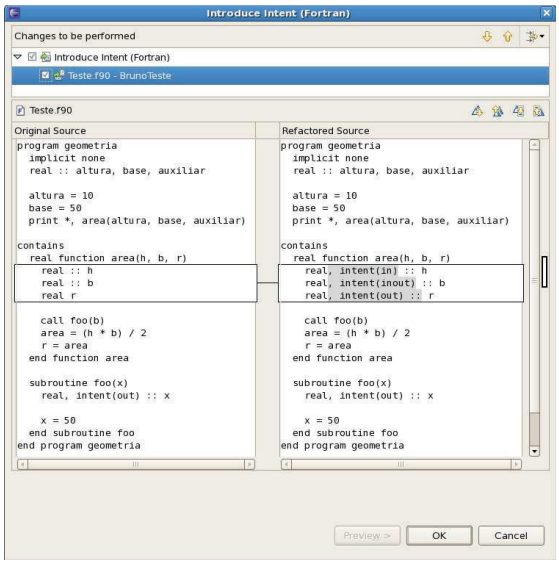

Figura 2. *Introduce INTENT* **no Photran**

Para extrair uma subrotina, o programador deverá selecionar no editor de código um conjunto de linhas e será questionado por um assistente acerca do nome da nova subrotina. O código a ser extraído deve respeitar os limites de blocos. Por exemplo, não é recomendado extrair um laço de repetição pela metade ou parte de uma subrotina e parte de outra. O bloco de código extraído é transformado em uma subrotina (comando `subroutine`) e em seu lugar é adicionado um comando `CALL` para fazer a chamada à mesma (informando os parâmetros, se necessário).

A extração de uma subrotina é uma técnica de refatoração que afeta diretamente a legibilidade do código, porém, dependendo da forma como é utilizada pode ter um impacto negativo sobre o desempenho. Não é recomendado, por exemplo, que pequenos blocos de instruções executadas dentro de laços de repetição sejam empacotadas em subprogramas, principalmente se existe a passagem de parâmetros.

É importante nomear bem as novas subrotinas para que estas permitam deduzir, mesmo sem comentários, seu propósito. A figura 3 ilustra a refatoração *Extract Subroutine* sendo utilizada para extrair do código a construção de um menu (à esquerda). Um subprograma `menu` é criado e adicionado ao código (logo após a instrução `CONTAINS`) e no lugar do código original é feita a chamada `CALL`

`menu(numero)`. Neste caso, o código extraído faz uso de uma variável (`numero`) que foi transformada em um parâmetro e que utiliza o atributo `INTENT(INOUT)`

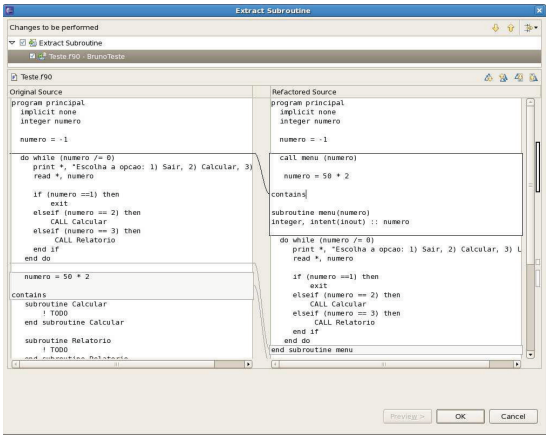

Figura 3. *Extract Subroutine* **no Photran**

3.3. Desenrolar Laço de Repetição

Aplicações de alto desempenho constantemente repetem instruções dentro de blocos de repetição e verificam se as mesmas devem ser novamente executadas. Otimizações no tempo de execução em tais estruturas podem representar resultados significativos ligados ao desempenho.

A técnica de desenrolar laços de repetição, *loop unrolling*, consiste em se replicar o corpo do laço por um fator de desenrolamento e alterar referências às variáveis de controle, que precisam ser incrementadas para cada instrução replicada. Sob o ponto de vista da legibilidade, um laço de repetição desenrolado é mais complexo de ser entendido. Porém, ao diminuir o número de iterações a uma quantidade apropriada, um laço desenrolado produz um significativo ganho de desempenho.

Compiladores geralmente realizam essas otimizações reestruturando o código em baixo nível [4]. Contudo, nem sempre o número de iterações de um laço pode ser detectado em tempo de compilação e há situações nas quais otimizações de compilador não podem ser utilizadas (em função de efeitos colaterais em outras regiões do código). Neste caso, a automatização da técnica tem uma importância ainda maior, dada a complexidade de se desenrolar um laço de repetição manualmente e a probabilidade de adição de erros involuntários ao código reestruturado.

Conforme a implementação mostrada na figura 4, o laço original manipula um vetor indexado pela variável de controle `i`, que vai de 1 a 100 e executa uma linha interna uma vez para cada posição do vetor. No laço desenrolado, 5

```
! Laço original
do i=1, 100
  c(i) = a(i) + b(i)
end do

!Laço desenrolado em 5 níveis
do i=1, 100, 5
  c(i) = a(i) + b(i)
  c(i+1) = a(i+1) + b(i+1)
  c(i+2) = a(i+2) + b(i+2)
  c(i+3) = a(i+3) + b(i+3)
  c(i+4) = a(i+4) + b(i+4)
end do
```

Figura 4. Código comparativo - laço original e laço desenrolado

execuções internas são realizadas e o incremento da variável de controle também acompanha este valor. Neste exemplo, onde o laço de repetição foi desenrolado para um fator de 5, o número de comparações feitas é reduzido de 100 para 20.

A mecânica de funcionamento do laço é simples. O usuário precisará inicialmente indicar o bloco completo do laço que deseja desenrolar e também o número de níveis em que ele será desenrolado. Partindo de tais informações, a ação de refatoração identifica a variável de controle do laço e repete o conteúdo do mesmo tantas vezes quantas necessárias. Naquelas instruções nas quais a variável de controle é referenciada, ela é então incrementada até o número de níveis em que o laço foi desenrolado. A técnica automatizada atua exclusivamente sobre laços de repetição indexados e que utilizam-se do bloco `DO..END DO`.

4. Avaliação

Para avaliar o impacto da utilização de técnicas de refatoração em aplicações de alto desempenho escritas em linguagem Fortran, o trabalho utilizou-se do código-fonte de uma aplicação cedida pelo Laboratório de Micrometeorologia vinculado à Universidade Federal de Santa Maria. O objetivo deste estudo de caso é avaliar de que forma a utilização de técnicas de refatoração podem influenciar no desempenho (de forma positiva ou negativa) da referida aplicação. O código-fonte foi compilado utilizando-se GCC (`gfortran`) e Intel Fortran (`ifort`), ambos com as otimizações desabilitadas.

A aplicação alvo foi escrita em linguagem Fortran 77 e tem grande importância para pesquisas no laboratório, sendo utilizada para analisar conjuntos numerosos de dados captados por sensores em estações meteorológicas. Dado seu tempo de vida e algumas construções obsoletas utilizadas no seu desenvolvimento o código-fonte apresenta uma alta complexidade de compreensão e manutenção. Para ge-

rar os resultados utilizando um computador AMD Atlhon XP 2 Ghz com 1 Gb de RAM e um arquivo de entrada com 100 Mb é necessário o tempo de 01:02:20 (compilador GCC) ou 00:59:19 (compilador Intel).

Antes da avaliação, o código da aplicação alvo foi preparado para receber as refatorações. Duas ações foram realizadas neste sentido: os laços de repetição que utilizavam-se de desvios rotulados foram manualmente transformados em laços do tipo `DO..END DO` e os tipos implícitos foram removidos (utilizando-se da refatoração *Introduce Implicit None*, disponível no Photran). O código-fonte original da aplicação (com a substituição dos laços rotulados e introdução explícita de declarações) possui em torno de 1130 linhas de código, sendo composto por um programa principal e 10 subrotinas.

A primeira etapa da avaliação consistiu em introduzir os atributos `INTENT` às 10 subrotinas existentes, além de explorar regiões de código que poderiam ser transformadas em novas subrotinas. Não era esperado qualquer ganho de desempenho nessa etapa da avaliação, uma vez que as técnicas aplicadas alteram características do design de código. Ao todo foram extraídas 17 novas subrotinas baseando-se em blocos de comentários que explicavam os principais objetivos do código-fonte. Todas as subrotinas existentes e as novas que foram extraídas receberam os atributos `INTENT`. A nova versão da aplicação comportou-se com um desempenho bastante semelhante à versão original. A versão compilada com o GCC foi 70 segundos mais lenta (o que equivale a 1,87% do tempo de execução) enquanto que a versão Intel foi apenas 26 segundos mais lenta (o equivalente a 0,73% do tempo).

Na segunda etapa, o objetivo foi demonstrar que algumas ações de refatoração podem potencializar o desempenho da aplicação, neste caso compensando a pequena queda de desempenho ocorrida durante a primeira etapa. Para tanto, utilizou-se a refatoração *Loop Unrolling*. Neste caso é importante considerar que a refatoração em questão (*Loop Unrolling*) pode ter um efeito negativo se utilizada em demasia, prejudicando a legibilidade do código.

Para minimizar este impacto, apenas o maior laço de repetição (com o maior número de iterações) foi desenrolado. O laço em questão poderia gerar (no pior caso) aproximadamente 1.073.741.824 iterações. A técnica *Loop Unrolling* automatizada na ferramenta Photran foi utilizada para desenrolar o laço de repetição em 32 níveis, reduzindo o número de iterações para 33.554.432. A execução da aplicação demonstra que o ganho de desempenho pode chegar a 4,35% (compilador Intel) e 4,25% (compilador GCC), considerando os tempos da primeira etapa. A tabela 1 contém todos os tempos das simulações e a figura 5 apresenta um gráfico comparativo dos tempos de execução de cada versão da aplicação.

Um segundo experimento foi realizado com o objetivo

Tabela 1. Avaliação do Tempo de Execução

Verão do Código	GCC	Intel
Original	01:02:20	00:59:19
Etapa 1 [1]	01:03:30	00:59:45
Etapa 2 [2]	01:00:48	00:57:09

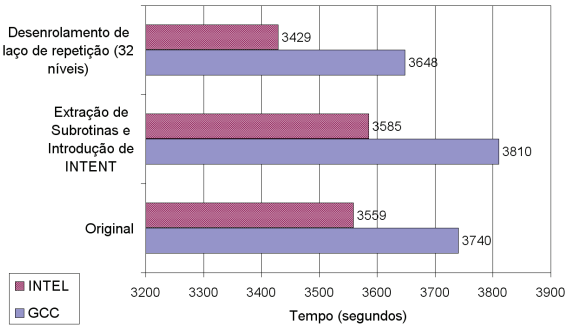

Figura 5. Gráfico comparativo dos tempos de execução

de comparar a precisão da automatização da técnica de introdução de `INTENT`. Neste caso, a aplicação utilizada foi cedida pelo CPTEC/INPE e constitui-se de um conjunto de subrotinas utilizadas para simular o processo de advecção do BRAMS [2]. O trabalho de Fazenda et al. [10] descreve um conjunto de transformações realizadas manualmente no código do processo de advecção do BRAMS, para torná-lo portável a diferentes arquiteturas de execução. Uma das etapas do trabalho foi a inclusão do atributo `INTENT` aos argumentos das subrotinas.

Refatorando-se o código original do processo de advecção, composto por 7 subprogramas (com 136 argumentos ao todo), utilizando a ferramenta Photran e a técnica *introduce `INTENT`*, apenas 15 argumentos ficaram com o `INTENT` diferente daquele que foi atribuído durante a refatoração manual. Desses 15 argumentos, 6 deles ficaram sem `INTENT` uma vez que a ferramenta não previa a não utilização de um argumento no corpo do subprograma. Os outros 9 referem-se a argumentos multidimensionais onde a ferramenta aplicou o `INTENT` de acordo com as regras estabelecidas na seção 3. Contudo, a ferramenta atualmente não consegue identificar, por exemplo, se um vetor é totalmente modificado ou parcialmente modificado. Uma análise de dependências pode ser uma alternativa para contornar este problema.

[1] Extração de subrotinas e introdução de `INTENT`

[2] Todas as melhorias da etapa 1 e também o desenrolamento de laço de repetição (em 32 níveis)

5. Considerações Finais

O presente trabalho explora a automatização e utilização de técnicas de refatoração sobre aplicações de alto desempenho escritas em Fortran, objetivando melhorar o design de código e detectar oportunidades de ganho de desempenho. A ferramenta Photran, um *plugin* para Eclipse, foi utilizado como IDE uma vez que permite estender funcionalidades de refatoração e aplicá-las sobre código Fortran.

Os resultados da avaliação demonstram que técnicas como *Introduce INTENT* e *Extract Subroutine*, que melhoram o design de código, não possuem influência negativa sobre o desempenho da aplicação. Dependendo da forma que são utilizadas ao mesmo tempo que tornam o código mais legível e melhor organizado, não introduzem gargalos que reduzem de forma significativa ou perceptível o desempenho.

A técnica *Loop Unrolling*, desenvolvida especialmente em função do ganho de desempenho, mostrou-se eficaz e comprovou o ganho de desempenho depois de utilizada. Neste caso é importante destacar que o uso deve ser moderado, pois ao passo que o desempenho é beneficiado a legibilidade do código pode ser prejudicada.

A utilização do Photran como ferramenta de apoio à codificação e automação de técnicas de refatoração também merece destaque. Ferramentas como Photran representam um importante avanço no sentido de se preencher a lacuna existente entre a enorme quantidade de código Fortran (em especial de aplicações científicas) e o limitado número de ferramentas de apoio ao desenvolvimento com técnicas de refatorações integradas. A continuidade dessa pesquisa objetiva detectar outras situações nas quais técnicas de refatoração podem ter influência direta no desempenho e também a automatização de outras técnicas ligadas à evolução de construções de código e aplicação de "melhores práticas" de desenvolvimento.

Referências

[1] Refactoring Home Page, 2008. http://www.refactoring.com.

[2] Brazilian Regional Atmospheric Modeling System Home Page, 2009. http://brams.cptec.inpe.br.

[3] J. C. Adams, W. S. Brainerd, R. A. Hendrickson, R. E. Maine, J. T. Martin, and B. T. Smith. *The Fortran 2003 Handbook: The Complete Syntax, Features and Procedures*. Springer Publishing Company, Incorporated, 2008.

[4] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2 edition, 2006.

[5] A. M. Boehm, D. Seipel, A. Sickmann, and M. Wetzka. Squash: A Tool for Analyzing, Tuning and Refactoring Relational Database Applications. In *17th International Conference on Applications of Declarative Programming and Knowledge Management*, pages 82–98, Berlin, Alemanha, 2007. Springer-Verlag.

[6] N. Chen and J. Overbey. *Photran 4.0 Developer's Guide*, 2008.

[7] M. L. Cornelio. *Refactorings as Formal Refinements*. Tese de doutorado, Universidade Federal de Pernambuco, Recife, Brasil, 2004.

[8] V. De. *A Foundation for Refactorin Fortran 90 in Eclipse*. Dissertação de mestrado, University of Illinois, Urbana-Champaign, EUA, 2004.

[9] L. Dobrzański and L. Kuźniarz. An Approach to Refactoring of Executable UML Models. In *ACM Symposium on Applied Computing*, pages 1273–1279, Dijon, Franca, 2006. ACM.

[10] A. L. Fazenda, E. H. Enari, L. F. Rodrigues, and J. Panetta. Towards Production Code Effective Portability among Vector Machines and Microprocessor-Based Architectures. In *18th International Symposium on Computer Architecture and High Performance Computing*, pages 11–20, Ouro Preto, Brasil, 2006. IEEE Computer Society.

[11] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[12] E. Graf, G. Zgraggen, and P. Sommerlad. Refactoring Support for the C++ Development Tooling. In *22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion*, pages 781–782, Montreal, Canada, 2007. ACM.

[13] S. Hosseini and M. A. Azgomi. UML Model Refactoring with Emphasis on Behavior Preservation. In *2nd International Symposium on Theoretical Aspects of Software Engineering*, pages 125–128, Nanjing, China, 2008. IEEE Computer Society.

[14] S. McConnell. *Code Complete: a Practical Handbook of Software Construction*. Microsoft Press, 1993.

[15] T. Mens, S. Demeyer, B. D. Bois, H. Stenten, and P. V. Gorp. Refactoring: Current Research and Future Trends. *Language descriptions, Tools and Applications*, 82(3):483–499, 2003.

[16] W. Opdyke. *Refactoring Object-Oriented Frameworks*. Tese de doutorado, University of Illinois, Urbana-Champaign, EUA, 1992.

[17] J. Overbey and R. Johnson. Generating Rewritable Abstract Syntax Trees. In *First International Conference on Software Language Engineering*, pages 114–133, Toulouse, Franca, 2009. Springer-Verlag.

[18] J. Overbey, S. Negara, and R. Johnson. Refactoring and the Evolution of Fortran. In *Second International Workshop on Software Engineering for Computational Science and Engineering*, pages 28–34, Vancouver, Canada, 2009. IEEE Computer Society.

[19] J. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for Fortran and High-Performance Computing. In *Second International Workshop on Software Engineering for High Performance Computing System Applications*, pages 37–39, St. Louis, EUA, 2005. ACM.

[20] M. Rieger, B. V. Rompaey, B. D. Bois, K. Meijfroidt, and P. Olievier. Refactoring for Performance: An Experience Report. In *Third International ERCIM Symposium on Software Evolution*, pages 206–214, Paris, Franca, 2007.