Proposta de otimizador de código como um sistema multi-agentes em RP-Net

Bruno Ribeiro*
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
brunex.geek@gmail.com

ABSTRACT
Neste artigo será apresentada uma proposta de modelagem para otimizadores de código através de RP-Nets. Para tanto, o processo de otimização de código foi reorganizado em termos de sistemas multi-agentes, cada qual especializado num tipo de otimização.

1. INTRODUÇÃO
Este trabalho tem como objetivo propor uma nova modelagem para mecanismos de otimização de código, comumente partes integrantes de compiladores, utilizando os formalismos das RP-Nets. Sobretudo, intencionou-se reduzir a complexidade da construção de tais mecanismos tentando elaborar uma arquitetura mais genérica que pudesse ser modificada e expandida de forma mais fácil e modular. Nesse sentido, introduziu-se a noção de sistemas multi-agentes, no qual as técnicas de otimização são mapeadas em agentes que atuam sobre blocos básicos de um programa. Como premissa, considerou-se apenas as técnicas de otimização que atuam sobre código intermediário, pois além de ser um tipo de representação mais flexível, permite que os mecanismos elaborados sejam facilmente reusados independente da linguagem da máquina alvo para qual o compilador está gerando um executável.

Este artigo está organizado em quatro seções principais. Na seção 2 é apresentada uma introdução às RP-Nets, descrevendo os elementos que a compõe e seu funcionamento básico. Na seção 3 é tratado o processo de otimização de código. A seção 4 descreve a RP-Net proposta nesse trabalho através de diagramas e do detalhamento de sua estrutura e funcionamento. Por fim, são feitas algumas considerações finais na seção 5.

2. RP-NETS
Uma rede de processamento de recursos (*Resource Processing Network* ou RP-Net) é uma ferramenta matemática que se destina a modelar sistemas que envolvem o processamento de recursos de qualquer natureza, transformando e/ou consumindo tais recursos de acordo com a dinâmica do sistema. Essa rede pode ser definida como um modelo formal-computacional de sistemas à eventos discretos, uma vez que pode realizar o processamento de recursos de forma concorrente através de múltiplos agentes, tanto como uma técnica para a projeto de sistemas inteligentes, já que os recursos são definidos como entidades genéricas podendo assim ser caracterizados como representações de conhecimentos, ideias e signos [5]. Historicamente, a RP-Net é fruto da evolução de formalizações e conceitos da Rede de Objetos [3], da Rede de Agentes [2] e da Rede Semiônica [4].

Uma RP-Net é representada através de um diagrama formado por um conjunto de recursos situados em lugares, sendo esses últimos conectados por arcos. Os recursos podem ser ativos ou passivos, devem ser identificados por um nome exclusivo e interagem de acordo com as conexões entre lugares. Um recurso passivo é uma entidade a ser consumida de alguma forma e pode ser de dois tipos: material (peças, matérias-primas, pessoas, etc.) ou de informação (tabelas, arquivos, imagens, etc.). Já um recurso ativo são entidades capazes de processar outros recursos (ativos ou passivos), inclusive processar a si mesmo.

Todos os recursos de uma RP-Net estão situados em lugares interligados por arcos. Os lugares são entidades passivas capazes de armazenar um número arbitrário de recursos. Os pontos de contato dos arcos com os lugares são chamados de portas, que podem ser de entrada ou de saída, dependendo da direção do arco associado. Um recurso ativo armazenado em um determinado lugar somente poderá se servir dos recursos que estejam em lugares conectados ao lugar em que se encontra [5]. A figura 1 apresenta um diagrama RP-Net ilustrativo.

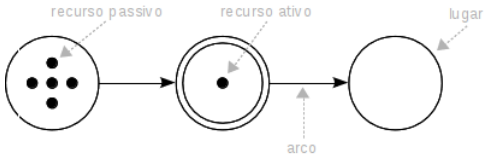

Figure 1: Exemplo de RP-Net

2.1 Recursos ativos
Os recursos ativos são os responsáveis por todo o processamento que uma RP-Net é capaz de realizar. Cada recurso ativo possui um conjunto de ações possíveis e a forma como as ações são constituídas define o tipo do recurso ativo. Os recursos ativos mecânicos são aqueles que não aplicam nenhum tipo de critério na escolha dos recursos que vai processar, a não ser a condição inerente de os recursos serem de um tipo compatível. Já os recursos ativos inteligentes

*Registro Acadêmico: 096149

são capazes de avaliar os recursos disponíveis e ponderar sobre quais deles são mais adequados/relevantes ao processamento.

As ações de um recurso ativo são determinadas através da definição de *funções de transformação*. Cada função está associada às portas de entrada, a partir da qual a função obterá os recursos a serem processados, e de saída, onde os recursos resultantes do processamento serão colocados. Uma função de transformação deve ser capaz tanto de realizar a seleção dos recursos de entrada, avaliando sua relevância de acordo com um conjunto de critérios, como de realizar as operações necessárias, gerando os recursos de saída. Para tanto, elas fazem uso de dois operadores especiais: os *operadores de avaliação* e os *operadores de transformação*.

Operador de avaliação determina, para um conjunto de recursos candidatos a serem processados, qual é o interesse do recurso ativo em selecionar cada um desses recursos. Esse interesse é representado por um valor de *utilidade* da função. Além disso, determina o *modo de acesso* com o qual a função deseja acessar determinada porta —se o recurso deve ser acessado com exclusividade ou compartilhado e se o recurso será consumido (deixará de existir).

Operador de transformação realiza as operações sobre os recursos selecionados previamente e gera as saídas, se necessário.

Retomando os dois tipos de recursos ativos apresentados anteriormente, temos que um recurso ativo mecânico não define um operador de avaliação, uma vez que não requer qualquer tipo de ponderação sobre os recursos potenciais a serem utilizados no processamento da função de transformação. Entretanto, o operador de avaliação é essencial para recursos ativos inteligentes, sendo utilizados para determinar uma ordem de preferência dos recursos disponíveis através da atribuição de uma nota/valor aos mesmos segundo critérios arbitrários, que podem ser tão complexos quanto se deseje.

Tatai [5] resume a definição de um recurso ativo como sendo um recurso que apresenta as seguintes características:

- é capaz de executar tarefas que envolvam ou não outros recursos, ou seja, executa atividade de processamento de recursos (os recursos processados podem ser passivos ou ativos);
- é capaz de selecionar outros recursos de um dado conjunto, dos quais depende para executar suas tarefas;
- é capaz de destruir ou preservar esses recursos escolhidos, dependendo da tarefa; uma vez de posse desses recursos, realiza operações internas, de forma a gerar outros recursos;
- é capaz de gerar novos recursos, a partir da assimilação e transformação do conteúdo dos recursos previamente selecionados;
- realiza incessantemente esta atividade de escolha, assimilação e geração de novos recursos, de maneira autônoma e independente;
- pode possuir diferentes maneiras de escolher e processar os recursos em seu ciclo operacional.

3. OTIMIZAÇÃO DE CÓDIGO

Comumente sendo última etapa no processo de geração de código em um compilador, a otimização de código consiste em efetuar transformações nas estruturas de um programa a fim de melhorá-lo em algum aspecto. Os otimizadores podem otimizar programas em vários estágios do processo de compilação e a diferença está na forma como o programa original está representado em dado estágio. A figura 2 apresenta uma visão geral do processo de compilação, as transformações no código do programa e alguns tipos de otimização possíveis em cada fase.

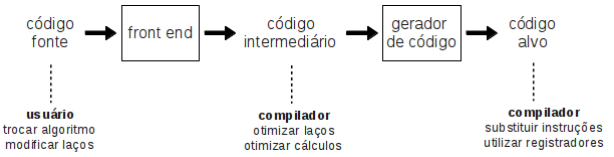

Figure 2: Oportunidades de otimização (adaptado de [1])

De uma forma geral, os otimizadores pode efetuar otimizações sobre representações intermediárias (árvores sintáticas, código intermediário, etc.) e sobre representações em linguagem de máquina. No primeiro caso, são realizadas otimizações ditas *independentes de máquina* ou *machine-independent*, uma vez que tais transformações independem da arquitetura de conjunto de instruções (ISA) da máquina alvo. O tipo de transformações que ocorrem nesse nível são mudanças na ordem de operações, substituição de construções algébricas por equivalentes mais eficientes, remoção de código redundante ou que nunca será executado, etc. No segundo caso, são realizadas otimizações ditas *dependentes de máquina* ou *machine-dependent*, pois elas são específicamente elaboradas em função da arquitetura de conjunto de instruções para qual o programa está sendo compilado. Os tipos de transformações que ocorrem nesse nível são substituições de instruções por equivalentes mais eficientes, substituição de um conjunto de instruções por uma possível única instrução que realiza o mesmo processamento (como uma operação binária de *rotate no carry*), etc.

O processo de otimização de código é norteado por três premissas fundamentais. Primeiramente, as transformações devem preservar o significado do programa original, ou seja, um programa otimizado deve produzir as mesmas saídas que o programa original geraria a partir das mesmas entradas. Em segundo lugar, o otimizador deve melhorar o programa original em uma quantidade mensurável, ou seja, deve ser possível avaliar se houve melhora efetiva e ser possível medi-la. Por fim, as possíveis transformações a serem aplicadas precisam compensar o esforço [1]. Esse último aspecto é válido tanto para o implementador do otimizador quanto para quem fará uso dele: não valerá a pena investir esforço em implementar um algoritmo de otimização complexo que fornecerá um ganho muito pequeno como resultado final; e

não valerá a pena um programador utilizar um algoritmo de seu otimizador se o tempo gasto na otimização é muito alto e o ganho é pequeno.

Apesar das técnicas de otimização de código permitirem um ganho significativo, é preciso atentar a alguns problemas inerentes do processo. Primeiramente é impossível construir um otimizador capaz obter um programa ótimo, isto é, um otimizador que recebe como entrada um programa P e gera um programa P' equivalente que é o melhor possível, segundo o aspecto considerado. Isso se deve ao fato de que um otimizador não consegue obter informações sobre, ou mesmo testar, todas as possibilidades de execução de um programa para que possa entender sua dinâmica ou o relacionamento entre suas partes. O que se consegue obter é um programa P' que tende a ser melhor do que o programa P original naquele aspecto, ou seja, um potencial ótimo local.

Em segundo lugar, as transformações que geralmente fornecem os maiores ganhos, em especial de desempenho, são mudanças de algoritmo. Entretanto, os otimizadores de código não são capazes de encontrar o melhor algoritmo para cada caso e, na maioria dos casos, nem mesmo serão capazes interpretar o que uma construção do código realiza. O que os otimizadores comumente farão é substituir instruções por outras potencialmente mais eficientes ou melhorar as construções existentes [1].

4. PROPOSTA DE REDE PARA OTIMIZAÇÃO DE CÓDIGO

Como descrito na seção 3, o processo de otimização de código compreende um conjunto de técnicas individuais, cada qual atuando na solução de um problema específico. A escolha da ordem de da frequência com a qual cada técnica será aplicada sobre o código intermediário tende a ser uma decisão particular dos implementadores do compilador, mas existem algumas premissas básicas que permitem orientar essa aplicação, dentre elas, podendo-se destacar duas: as técnicas que atuam localmente, ou seja, dentro do escopo de blocos básicos, devem ser aplicadas primeiro; e uma mesma técnica provavelmente precisará ser aplicada mais de uma vez sobre um mesmo trecho de código, pois uma técnica anterior pode ter como efeito colateral algum tipo de redundância que poderia ser otimizada por uma técnica diferente.

A partir dessas observações torna-se possível considerar que, no processo de otimização de código, as técnicas, de uma forma implícita, atuam em conjunto e de forma coordenada a fim de otimizarem o código de uma maneira que nenhuma técnica, individualmente, seria capaz. Sendo assim, foi aplicada uma abstração de forma a pensar no processo otimização como sendo um sistema multi-agentes, onde cada técnica de otimização atua como um agente. Do ponto de vista das RP-Nets, os agentes são representados por recursos ativos, denominados de *agentes otimizadores*, e os blocos de códigos intermediários a serem processados são representados por recursos passivos. A figura 3 apresenta um diagrama RP-Net modelando dois agentes otimizadores atuando sobre blocos básicos de código intermediário.

No modelo representado pela figura 3 temos um *buffer* centralizado contendo todos os blocos básicos de código intermediário a serem processados. Esse *buffer* deve preenchido

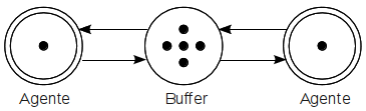

Figure 3: RP-Net com dois agentes otimizadores

pelo compilador com os blocos básicos gerados nas etapas anteriores do processo de compilação. A partir desse momento esses blocos tornam-se disponíveis para que os agentes otimizantes possam efetuar seus processamentos. Os agentes otimizantes sempre consomem os blocos básicos, gerando um novo após o processamento, e os blocos básicos estão sempre situados no *buffer* central.

No modelo proposto, os agentes otimizantes são recursos ativos inteligentes, pois em suas operações de avaliação devem efetuar avaliações que correspondam às premissas básicas de otimização, entre outros critérios:

- Blocos básicos que ainda não foram processados por todos os agentes otimizantes que atuam localmente não devem fazer parte do escopo habilitante de um agente otimizante que atue globalmente;
- Se algum agente otimizante conseguir efetuar alguma transformação em um bloco básico, todos os outros agentes devem processar tal bloco novamente. Pode-se adicionar algum mecanismo para evitar ciclos infinitos (caso dois ou mais agentes otimizantes criam efeitos colaterais otimizáveis entre si);
- Blocos básicos que já foram processados por todos os agentes otimizantes e em todos os casos não houve a necessidade, ou possibilidade, de transformação não devem fazer parte do escopo habilitando de qualquer agente. Essa regra é necessária para identificar blocos que não podem ser mais otimizados, bem como permitir que o otimizador consiga identificar quando o processo de otimização de código terminou. É importante destacar que esse critério requer algum tipo de memória no recurso passivo a fim de registrar essas informações.

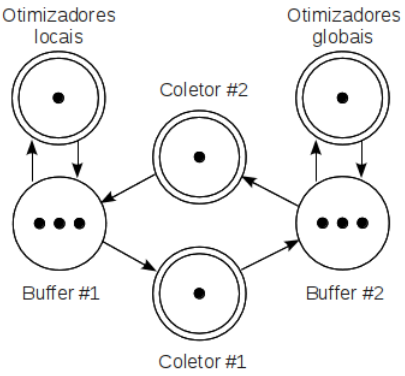

Figure 4: Versão melhorada da RP-Net proposta

Dependendo da quantidade de blocos básicos existentes no *buffer*, pode tornar-se custosa a execução da operação de avaliação de todos os agente otimizantes sobre todos os recursos. Como melhoria, pode-se separar os blocos básicos que já foram processados por técnicas locais, permitindo assim que os agentes otimizadores sempre avaliem um conjunto menor e mais coerentes de recursos. A figura 4 apresenta uma reestruturação da rede de forma a ter dois *buffers*. O primeiro *buffer* conterá, inicialmente, todos os blocos básicos do programa e na medida que os blocos básicos forem sendo processados, o primeiro coletor moverá os blocos que já receberam todas as otimizações locais para o segundo *buffer*. Os blocos básicos do segundo buffer, por sua vez, serão processados somente pelos agentes otimizadores de técnicas globais e qualquer um desses blocos que sofrer alguma transformação com esses otimizadores serão movidos de volta ao primeiro *buffer* pelo segundo coletor. O processo de otimização de um bloco termina quando o mesmo não sofrer mais transformações por nenhum agente otimizante e nesse instante tais blocos estarão no segundo *buffer*.

5. CONSIDERAÇÕES FINAIS

A RP-Net é uma ferramenta de formal poderosa, que permite expressar um variado repertório de tipos de sistemas. Nesse trabalho foram utilizados conceito de sistema multiagentes, aliados a representações em RP-Nets para tentar conceber uma nova modelagem para mecanismos de otimização de código. Dentre as vantagens do uso da abordagem apresentada, destacam-se:

- a facilidade em agregar novas técnicas de otimização, tornando o sistema mais modular;
- a facilidade de impor critérios sobre como o processo de otimização funcionará, através da adição de novos recursos ativos ou da modificação em regras que compõem o operador de avaliação dos agentes otimizadores;
- as técnicas de otimização global sempre atuarão sobre blocos básicos ótimos (dentro das capacidades das técnicas de otimização local);
- se expandido para todo o contexto do compilador, permitiria a criação de uma arquitetura padronizada de compilação, onde desenvolvedores de compiladores simplesmente criariam um conjunto de agentes, a serem utilizados na rede, destinados de compilar código de uma linguagem de alto nível para uma linguagem de máquina, quaisquer que fossem as mesmas.

Como melhoria, a RP-Net proposta poderia ser aprimorada inserindo-se a capacidade de interpretar recursos de *profiling*, ou seja, recursos contendo resultados de análises dinâmicas de programas, podendo ser descrições sobre o consumo de memória, uso de instruções em particular, frequência e duração de chamadas de função. Através dessas informações seria possível que rede proposta pudesse efetuar otimizações em um programa baseadas nas informações obtidas a partir de sua execução num cenário real ou simulado. Indo mais além, se a rede for capaz de armazenar e relacionar o histórico das transformações realizadas e as análises de execução correspondente a essas otimizações, seria possível introduzir o conceito de agentes otimizantes capazes de aprender através da experiência.

6. REFERENCES

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Prentice Hall, 2 edition, 2006.

[2] A. S. R. Gomes. *Contribuições ao Estudo da Rede de Agentes*. Tese de Mestrado, DCA/FEEC/UNICAMP, Campinas, 2000.

[3] R. R. Gudwin. *Contribuições ao Estudo Matemático de Sistemas Inteligentes*. Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, 1996.

[4] R. R. Gudwin. *Semiônica: Uma proposta de Contribuição à Semiótica Computacional*. Tese de Livre Docência, DCA/FEEC/UNICAMP, Campinas, 2003.

[5] V. K. Tatai. *Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador*. Tese de Mestrado, DCA/FEEC/UNICAMP, Campinas, 2003.