Linguagens para Descrição de Arquiteturas de Computadores

Mariza Andrade da Silva Bigonha[1]
José Lucas Mourão Rangel Netto[2]

RESUMO

Arquiteturas mais modernas de computadores motivam pesquisas por técnicas mais eficazes de implementação de compiladores capazes de gerar código de alta qualidade. As arquiteturas superescalares têm conjuntos reduzidos de instruções, cuja combinação para execução eficiente deve ser feita pelos "back-ends" dos compiladores usualmente de maior complexidade que seus correspondentes para arquiteturas CISC. Os objetivos deste trabalho são o estudo das características de linguagens formais de descrição de arquiteturas, apropriadas para uso com geradores de geradores de código, e o projeto e a validação semântica de uma linguagem de descrição de arquiteturas apropriada para uso com o gerador de geradores de código projetado.

ABSTRACT

Modern computer architectures lead to research looking for better techniques for effective implementation of compilers which must be able to produce high-quality code. In the special case of superscalar architectures we have reduced instruction sets, and instructions must be combined to warrant efficient execution by the compilers, which have more complex back-ends than their CISC counterparts. Objectives of this work are the study of the characteristics of formal languages for the description of architectures, meant for use with code generator generators and the project, and the semantic validation of an architecture description language adequate for use with the generator system.

1 Introdução

Qualquer discussão sobre geração de código deve considerar a arquitetura alvo e, isso introduz a necessidade de um mecanismo adequado para especificá-la. As linguagens de descrição

[1]Mariza Andrade da Silva Bigonha é mestre em Ciência da Computação pela UFMG e doutora em Informática: Ciência da Computação pela Pontifícia Universidade Católica do Rio de Janeiro. É analista de sistemas do Departamento de Ciência da Computação da UFMG. Áreas de interesse: linguagens de programação, compiladores, arquitetura.
E-mail:mariza@dcc.ufmg.br

[2]José Lucas Mourão Rangel Netto é mestre em Engenharia Elétrica pela COPPE/UFRJ e doutor em Ciência da Computação pela Univ. of Wisconsin, E.U.A. É professor do Instituto de Matemática da UFRJ e Professor Associado do Dep. de Informática da PUC/RJ. Áreas de interesse: engenharia de software, compiladores, linguagens de programação, teoria da computação.
E-mail: rangel@inf.puc-rio.br

para descrever, além das instruções e conjuntos de registradores, as unidades funcionais, os estágios da *pipeline* e outras propriedades do escalonamento como por exemplo, a latência e custo das instruções etc.;
- incorporar informações sobre a geração e otimização de código nesta linguagem;
- identificar classes de arquiteturas que podem ser descritas com a nova linguagem de descrição de máquina.

Um dos objetivos deste trabalho é apresentar soluções para estas questões. Precisamente, pretendemos demonstrar a viabilidade de definir uma linguagem de descrição de máquina concisa que seja capaz, contudo, de especificar diferentes famílias de processadores superescalares. Para demonstrar que alcançamos nosso objetivo, propusemos e definimos formalmente a sintaxe e semântica de uma linguagem para descrever o conhecimento embutido nas arquiteturas de computadores e projetamos um gerador de código para a arquitetura SPARC [SUN Microsystems, 1987, SUN Microsystems, 1989], escolhida como exemplo para demonstrar o poder de expressão da linguagem desenvolvida.

2 As Famílias de LDAs

Sucintamente, apresentamos um estudo comparativo das diversas linguagens existentes para descrever arquiteturas de máquinas CISCs, tendo em vista o projeto de uma nova LDA para arquiteturas superescalares. Iniciamos apresentando os principais trabalhos dentro de quatro famílias de geradores de código (veja Figura 1), encontrados atualmente na literatura, que se utilizam de linguagens de descrição de arquiteturas para obter o resultado almejado.

Glanville e Graham [Graham and Glanville, 1978, Graham et al., 1982] construíram o primeiro sistema redirecionável de gerador de código. Seu sistema tem como entrada a descrição da máquina em forma de gramática e produz como resultado um gerador de código que utiliza técnicas de análise sintática LR para efetuar o casamento de padrão com a linguagem intermediária. As produções da gramática são compostas de padrões, que englobam os símbolos terminais e não-terminais, e de um lado esquerdo que é formado por um símbolo não-terminal. Ações associadas às produções dirigem a geração do código de máquina.

O sistema CODEGEN de Robert Henry [Aigrain et al., 1984, Henry and Damron, 1989a, Henry and Damron, 1989b] é um sucessor do sistema de Graham-Glanville. Esse sistema tem como entrada PPRACs (padrão, predicado, substituição (*replacement*), ação, custo) e produz uma árvore de reconhecimento de padrão. PPRACs se assemelham às produções do sistema de Graham-Glanville, mas são acrescidas de predicados e custos para dirigir o casamento (*matching*).

O trabalho de Paulo Costa [Costa, 1990], intitulado *AutoCode*, é um sistema de produção de geradores de código baseado em reconhecimento de padrões e dirigido por tabelas geradas automaticamente, a partir de uma descrição da arquitetura da máquina alvo na forma de gramática livre do contexto. Seu sistema se assemelha à abordagem de Glanville e Graham.

geral, uma notação para a descrição de máquina real, baseada em *ISP* [Bell and Newell, 1971], além da semântica para tal descrição, que é apresentada em termos do modelo abstrato. Este enfoque resulta em uma descrição de instruções simples, entretanto a forma utilizada para descrever os modos de endereçamento é complicada e trabalhosa.

O projeto do PQCC (*Production-Quality Compiler-Compiler*) é um descendente do compilador Bliss-11 de Wulf [Wulf et al., 1975]. Este projeto foi muito ambicioso: PQCC tentou incluir uma gama de detalhes muito minuciosa na linguagem para descrever a arquitetura da máquina alvo, de tal forma que compromissos entre espaço e tempo na otimização, seleção de código e alocação de registradores pudessem ser examinados. Devido ao fato de ter tentado incorporar muitos detalhes da máquina alvo, o sistema ficou difícil de ser utilizado.

A abordagem utilizada por Warfield e Bauer [Warfield and Bauer, 1988] para descrever arquiteturas de máquinas é compacta, preenche todos os requisitos necessários em uma linguagem de descrição, inclusive mecanismos para definir regras. Entretanto, depende de uma ferramenta, *MRS* [Russell, 1985], que está disponível somente em três máquinas: *DEC-20* rodando *MacLISP*, *VAX* rodando *FranzLISP* sobre o *UNIX* de Berkeley e "*Symbolics LISP machine LM-2/3600*", o que torna o seu uso inviável.

A tabela a seguir mostra uma avaliação destes métodos.

	descrição máquina	método usado	redirecionamento
GLANVILLE	muito extensa incompleta	gramáticas livres do contexto LR	difícil avaliar dependência das linguagens:fonte e descritiva
GANAPATHI	longa, difícil e incompleta	gramáticas de atributos	difícil avaliar arquiteturas semelhantes
CATTELL	mais complexa, equipara-se a Ganapathi em complexidade	definição na forma de asserções de entrada e saída associada a cada instr.	distribuição de etapas em outras partes do compilador
COSTA	muito longa	gramáticas livres do contexto	dependência de máquina da repres. intermediária embutida no frontend
DAVIDSON	fácil de ler, flexível	expressão regular gramáticas livre contexto-lex,yacc	mais fácil
WARFIELD	compacta, mas depende de MRS	regras	difícil avaliar
GIEGERICH	simples porem forma utilizada para descrever modos de endereçamento complicada, trabalhosa	adaptação de ISP	restringe somente a MC86000, 8086
HENRY	razoável	enumeração de cinco tuplas PPRACS ou regras	difícil avaliar

Para especificar as características dos processadores RISC, linguagens de descrição de arquiteturas com as propriedades descritas até agora não são suficientes. Uma linguagem para

3 A LDA para Arquiteturas Superescalares

Considerando que uma linguagem de descrição de máquina serve como veículo para especificar o comportamento de uma máquina alvo, é importante que a mesma possua mecanismos para definir o máximo de informações possível sobre uma dada arquitetura. Dependendo da aplicabilidade da descrição de máquina, é fundamental que ela seja capaz de especificar outras características da máquina, além de seu conjunto de instruções e modos de endereçamento. Por exemplo, a inclusão de facilidade para definir custo relativo a tempo e espaço é característica importante, quando o objetivo é otimizar o código gerado. O conhecimento do custo é necessário para dirigir o processo de otimização. Outras informações, como por exemplo, mecanismos para atualizar o código de condição utilizado nas operações lógico-aritméticas, tornam as notações mais poderosas.

A LDA para a especificação de arquiteturas de máquinas superescalares que apresentamos neste trabalho leva em consideração as questões levantadas na Seção 1 e incorpora, de forma que consideramos satisfatória, mecanismos para definição dos modos de endereçamento, facilidades para descrever as unidades funcionais, os estágios do *pipeline* e outras propriedades para o escalonamento; permite definição completa da semântica das instruções; incorpora informações sobre a geração e otimização de código; identifica as classes de arquiteturas que podem ser descritas e é concisa e possui alto poder de expressão. Ela é uma adaptação da linguagem proposta por Bradlee [Bradlee et al., 1991], incorpora as características básicas desta linguagem, mas inclui novas facilidades, amplia seu leque de abrangência e a torna mais robusta. Ela é capaz de especificar os pontos abaixo relacionados necessários para descrever arquiteturas superescalares.

1. Características gerais das arquiteturas:
   - Registradores.
   - Estágios de *pipeline*.
   - Unidades funcionais.
   - Memória.
2. Modelo da máquina virtual.
   - Definição do uso dos registradores.
   - Passagem de parâmetros.
3. Lista de Instruções:
   - Instruções padrão para cada instrução de máquina deve ser especificado:
     - Mnemônico da instrução.
     - Restrições de tipo dos operandos.
     - Semântica da instrução.
     - Recursos de *pipeline* necessários.
     - Outras propriedades de escalonamento, como por exemplo. latência, custo e número de *slots* associado às operações.
   - Instruções especiais.
4. Detalhes específicos de arquitetura:

- Definição de Classes: define classe para ser o conjunto de elementos previamente definidos.
- Associação de Instruções a Relógios: define um nome que representa uma instrução como um elemento pertencente a um conjunto classe.

```
declarations
{
  %reg r[0:31] (charlintlshortlpointer);
  %reg i[0:7]  (charlintlshortlpointer);
  %reg f[0:31] (floatlint);
  %reg d[0:15] (double);
  %clock clk-cc;
  %reg cc(int; clk-cc) +temporal;
  %equiv i[0]    r[24];
  /* Integer Unit Stages  (CY7C601) */
       %resource IF;  /* instruction fetch */
       %resource ID;  /* decode */
  %memory m[0:4294967295];
  %def uns-const[65536:2147483647] +relocatable;
   %label rel-lab[-4194304:4194303] +relative
}
```

Figura 2: Trecho da Seção de Declarações para uma dada Arquitetura

Informações coletadas das diretivas da seção de declarações são colocadas em uma matriz e posteriormente utilizadas nas rotinas para construir o DAG de código[7], para gerar código, para efetuar reconhecimento de padrões, para alocar registradores, para fazer transformações e para escalonar instruções.

3.1.2 Seção de Definição da Máquina Alvo

A seção (`vm`) descreve o modelo de execução da máquina à qual o código gerado deve corresponder. Seus objetivos são, basicamente: especificar as convenções das chamadas de procedimento; definir o uso dos registradores. Por exemplo, na SPARC e na maioria das arquiteturas, durante a entrada de um procedimento, um registrador, o apontador de *frame*, é estabelecido para apontar para a base do registro de ativação corrente na pilha. Todas as referências a registradores locais utilizam índices a partir dele, de modo que os registradores locais mantêm

[7] DAG de código é a estrutura de dados usada no processo de escalonamento de instruções. Representa-se nesta estrutura de dados os blocos básicos[8] em que foi dividido o programa.

- Declaração de registradores com valores pré-definidos: %hard define registradores que contêm valores especificados pelo *hardware* da arquitetura em questão.
- Definição dos registradores preservados: define registradores com a diretiva %calleesave que devem ser preservados pelo procedimento chamado. Na arquitetura SPARC corresponde aos registradores (*out*) declarados pela diretiva %arg.
- Definição dos registradores preservados pelo procedimento chamador: define registradores com a diretiva %callersave que devem ser preservados pelo procedimento chamador. Na arquitetura SPARC corresponde aos registradores (*in*) declarados pela diretiva %par.
- Definição de argumentos específicos: define os argumentos do tipo especificado em %reg.
- Definição de parâmetros específicos: define os parâmetros do tipo especificado em %reg.
- Definição do resultado da operação: %result define o registrador que contém o resultado de acordo com o tipo especificado.
- Definição do endereço de retorno: %retaddr define o registrador que contém o endereço de retorno.
- Definição do método de avaliação de argumento: a idéia da diretiva associada a este item é poder especificar quando expressões devem ser avaliadas ou não. A diretiva usada neste caso é %evalargs.

As informações coletadas das diretivas %calleesave, %callersave, %allocable, %args, %par, %result e %equiv são colocadas em uma matriz para serem usadas na geração de código, durante o escalonamento de instruções, na construção do DAG de interferência[9] de registradores e essencialmente na passagem de parâmetros. As demais diretivas são utilizadas durante a execução da máquina alvo.

3.1.3 Seção de Definição de Instruções

O objetivo desta seção é descrever as instruções da máquina alvo e suas funções. A seção de definição de instruções é composta de dois tipos de instruções: as instruções comuns e as instruções especiais. Cada diretiva %instr descreve uma instrução em cinco partes. A Figura 4 apresenta a sintaxe de uma instrução. A primeira parte especifica o mnemônico e os operandos da instrução. A segunda parte, entre parênteses, especifica, opcionalmente, as restrições de tipos dos operandos. A terceira parte, entre chaves, define a semântica da instrução. A quarta parte, entre colchetes, define os recursos utilizados pela instrução. A quinta parte, entre parênteses, define informações sobre o custo e os ciclos gastos pela instrução, etc.

Incluem-se na categoria de instruções especiais as instruções para movimentação entre registradores, as declarações de instruções sem operação, as declarações das instruções "glue" e as definições de instruções para especificar uma nova latência. As instruções especiais auxiliam

[9] DAG de interferência é a estrutura de dados usada no processo de alocação de registradores.

```
instructions
{
 %instr  cmp  r, r, r[0]   (;  clk-cc)
   {cc = ($1 :: $2); }
   [IF; ID; IE; IW;]
   (1, 1, 0)

 %instr be  #rel-lab
    {if cc == 0
          goto $1;}
    [IF; ID; IE;]
    (1, 1, 1)

 %move  mov  i, r
    {$2 = $1;}
    [IF; ID; IE; IW;]
    (1, 1, 0)

  %glue  r,  #uns-const13          (int)
       { ($! == $2) ==> (($1 :: $2) == 0);}
}
```

Figura 5: Trecho da Especificação de Instruções em uma dada Arquitetura

Janela de registradores:
A janela de registradores, visível por uma função em particular, é um subconjunto do conjunto total de registradores. O projetista do compilador usa as diretivas %arg e %par para especificar separadamente os argumentos e parâmetros, o que modela a renomeação de registradores, e, utiliza as diretivas %calleesave e %callersave para modelar o salvamento dos registradores especificados.

Registradores com valores pré-definidos por *hardware*:
Os registradores com valores pré-definidos por *hardware* são especificados pela diretiva %hard.

Dependência Estrutural:
O projetista do compilador especifica as dependências de recursos ou dependências estruturais explicitamente durante a especificação de cada instrução da arquitetura em questão. Durante o escalonamento de instruções verificam-se estes recursos. Isto evita atrasos por dependência estrutural. Os recursos devem ter sido préviamente definidos na seção de declarações através da diretiva %resource.

Esquema de prioridade por *hardware* para contenção de recursos:
O projetista do compilador especifica as prioridades dos recursos associando valores numéricos aos recursos pertinentes na seção de declarações e a instrução indica sua pri-

5 Conclusões

Com o objetivo de projetar uma linguagem para descrição de arquitetura capaz de tratar as construções próprias dos processadores comercialmente disponíveis foi realizado um estudo sobre as linguagens existentes visando definir suas diretivas. Chegamos a conclusão de que deveriam ser incorporada à linguagem primitivas para auxiliar o escalonamento de instruções, como por exemplo, facilidades para descrever as unidades funcionais, os estágios da *pipeline*, informações sobre a geração e otimização de código, além daquelas diretivas para descrever as instruções e conjuntos de registradores.

Dentre os processadores comercialmente disponíveis, procuramos cobrir a classe de arquitetura de computadores denominada superescalar, subtende-se aqui, também os processadores RISCs. Consideramos como base para a especificação da linguagem para descrição de arquitetura o processador SPARC. Como resultado deste estudo, projetamos um gerador de geradores de código otimizado que inclui uma linguagem para descrever tais arquiteturas. Esta linguagem, LDA, possui além das primitivas necessárias à descrição de características gerais destes processadores, mecanismos para modelar aquelas peculiaridades de algumas arquiteturas superescalares, como por exemplo, diretivas para manipular janelas de registradores, *pipelines* com avanço explícito, latências de tempo de execução, latências auxiliares, recursos, etc. Enfim, LDA oferece vários mecanismos que auxiliam o processo de escalonamento de instruções. Mediante esta ferramenta, o projetista de compilador pode especificar o gerador de código para a máquina que deseja obter, bastando para isso que ele descreva as características do processador nesta linguagem. Estas informações são obtidas diretamente de manuais de usuário.

O projeto de LDA faz parte de um trabalho [Bigonha, 1994] no qual apresentamos um sistema de geração de código otimizado e definimos formalmente a a sintaxe e semântica de uma linguagem para descrever arquiteturas de computadores, cujos principais resultados obtidos foram: (1) projeto e validação semântica de uma linguagem de descrição de arquiteturas, apropriada para uso com o gerador de geradores de código projetado; (2) identificação dos atributos mais relevantes que devem ser considerados em uma LDA tendo em vista arquiteturas superescalares; (3) especificação da parte dependente de máquina do processador SPARC para seu respectivo gerador de código para demonstrar o poder de expressão da linguagem LDA proposta.

Referências

[Aho et al., 1989] Aho, A. V., Mahadevan, G., and Tjiang, S. W. K. (1989). Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems*, 11(4).

[Aigrain et al., 1984] Aigrain, P. et al. (1984). Experience with a graham-glanville code generator. In *Proceedings of the Sigplan'84 Symposium on Compiler Construction*, pages 13–24. ACM Sigplan Notices 19(6).

[Henry and Damron, 1989a] Henry, R. R. and Damron, P. C. (1989a). Algorithms for table-driven code generators using tree-pattern matching. Technical Report # 89-02-03, Computer Science Department, University of Washington, FR-35 Seattle, WA 89195 USA.

[Henry and Damron, 1989b] Henry, R. R. and Damron, P. C. (1989b). Performance of table-driven code, generators using tree-pattern matching. Technical Report # 89-02-02, Computer Science Department, University of Washington, FR-35 Seattle WA 89195 USA.

[Russell, 1985] Russell, S. (1985). The compleat guide to mrs. Technical Report KSL-85-12, Stanford Knowledge Systems Laboratory, Stanford University.

[Stallman, 1989] Stallman, R. M. (1989). Using and porting GNU C. Free Software Foundation Incorporation. Cambridge Massachusetts.

[SUN Microsystems, 1987] SUN Microsystems, I. (1987). *The SPARC Architectural Manual.* Mountain View, California.

[SUN Microsystems, 1989] SUN Microsystems, I. (1989). *The SPARC Architectural Manual.* Version 8, Part No. 800-1399-09.

[Tiemann, 1989] Tiemann, M. D. (1989). The gnu instruction scheduler. Class Report CS 343, Stanford University.

[Wall and Powell, 1987] Wall, D. W. and Powell, M. L. (1987). The mahler experience: Using an intermediate language as the machine description. In *ASPLOS-II Proceedings - Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, California.

[Warfield and Bauer, 1988] Warfield, J. W. and Bauer, H. R. (1988). An expert system for a retargetable peephole optimizer. *ACM Sigplan Notices*, 23(10).

[Wulf et al., 1975] Wulf, W. A., Johnson, R., Weinstock, C., Hobbs, S., and Geschke, C. (1975). *The Design of an Optimizing Compiler.* American Elsevier.