PBIW-SPARC: Uma Estratégia para Codificação de Instruções em Programas SPARC

Renato Santos*
*Instituto Federal de Mato Grosso do Sul
Coxim-MS, Brasil
renato.santos@ifms.edu.br

Renan Marks† e Ricardo Santos†
†Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Campo Grande-MS, Brasil
renan,ricardo@facom.ufms.br

***Resumo*—Esse trabalho apresenta o projeto e desenvolvimento da técnica de codificação de instruções PBIW sobre o conjunto de instruções SPARCv8. A técnica de codificação PBIW foi desenvolvida sobre uma infraestrutura de codificação de instruções em software que mapeia o código gerado pela saída de um compilador no esquema de codificação PBIW projetado para um processador alvo. A adoção da técnica PBIW para codificar programas SPARCv8 é denominada PBIW-SPARC. Instruções PBIW-SPARC possuem tamanho de 16 bits e os padrões codificados possuem tamanho de 24 bits. Experimentos estáticos e dinâmicos foram realizados de forma a caracterizar todos os efeitos da codificação PBIW-SPARC no código gerado e no processador alvo. Os resultados encontrados mostram que a codificação PBIW alcança ganhos na razão de compressão e desempenho: até 38% na redução do tamanho do programa e 1,75 de *speedup* em comparação aos programas SPARCv8.**

Palavras-chave*—*Codificação de Instruções, PBIW, Decodificadores de Instrução, SPARCv8.

I. INTRODUÇÃO

A partir das restrições de espaço em memória e desempenho impostas pelos sistemas embarcados, novas técnicas têm sido apresentadas visando reduzir o tamanho dos programas, de forma a também reduzir o impacto dos mesmos nos acessos à memória e, consequentemente, aumentar o desempenho final do código. Muitas dessas técnicas de redução de programas focaram em novos esquemas de compressão/codificação de instruções. Historicamente, tais técnicas têm sido propostas desde a década de 90 e são aplicadas tanto em arquiteturas de alto desempenho, de propósito geral, quanto em arquiteturas voltadas para domínios específicos de aplicações.

Os principais benefícios de um código menor são o menor consumo de energia, maior velocidade de execução e menor uso de memória. Tais características são especialmente desejáveis em sistemas embarcados, devido ao consumo de energia restrito, limitada capacidade de processamento e baixa capacidade de memória. No entanto, há de se considerar também a necessidade de balancear tais ganhos com os impactos gerados na microarquitetura do processador alvo, devido, inevitavelmente, à inclusão de hardware específico para decodificar, em tempo de execução, o programa que foi codificado em tempo de compilação.

A técnica de codificação de instruções PBIW [1] foi projetada, inicialmente, para arquiteturas que buscam instruções longas na memória como processadores baseados em arquiteturas EPIC, VLIW, arquiteturas reconfiguráveis com várias unidades funcionais e arquiteturas de alto desempenho baseadas em matrizes de unidades funcionais [2]. A técnica é composta por um algoritmo que realiza a fatoração de operandos [3] e por uma memória *cache* chamada de P-*Cache*.

Este artigo apresenta a implementação de um codificador de instruções, baseado na técnica PBIW, para o conjunto de instruções SPARC revisão 8 (SPARCv8) [4]. O presente trabalho utiliza uma infraestrutura de codificação flexível para aplicar, validar e avaliar o algoritmo PBIW sobre o conjunto de instruções SPARCv8. A aplicação de PBIW sobre programas SPARC será doravante denominada PBIW-SPARC. Apresenta-se também um conjunto de experimentos visando validar e avaliar a técnica comparando com programas gerados para o conjunto de instruções SPARCv8. Alguns resultados são comparados com outra técnica de codificação de instruções denominada SPARC16 [5]. O intuito principal é mostrar ganhos sobre a área de memória ocupada e desempenho de programas SPARC ao adotar um esquema de codificação como PBIW. Além disso, objetiva-se também mostrar que o projeto e implementação de codificadores de instruções pode ser facilitado com uma infraestrutura de codificação.

Neste artigo, a Seção II apresenta trabalhos relacionados com esta proposta; a Seção III detalha a infraestrutura de codificação PBIW e o algoritmo de codificação; a Seção IV apresenta o projeto e implementação da estratégia de codificação PBIW para o conjunto de instruções SPARCv8 e o projeto do decodificador PBIW-SPARC; os experimentos e resultados obtidos são apresentados na Seção V; por fim, as conclusões e propostas de trabalhos futuros estão presentes na Seção VI.

II. TRABALHOS RELACIONADOS

O intuito de codificar instruções consiste na perspectiva de reduzir o tamanho das instruções, mantendo-as com informações semânticas, com o objetivo de reduzir o espaço ocupado em memória pelo programa e, possibilitando que a instrução possa interagir com os estágios iniciais da via de dados do processador, minimizando os impactos de desempenho oriundos da decodificação. Apresenta-se nos próximos parágrafos, algumas propostas de codificação de instruções e considera-se que taxa de compressão/codificação (TC) é:

$$TC = \frac{\text{Tamanho Programa Codificado + Overhead}}{\text{Tamanho Programa Não - Codificado}} \quad (1)$$

O Overhead é devido à criação de palavras de dicionário ou padrões de código.

A técnica de codificação Thumb [6] aumenta a densidade de código ARM sem grandes perdas de desempenho. O conjunto de instruções Thumb é um subconjunto das instruções ARM codificado em 16 bits. A codificação Thumb resultou em uma taxa de compressão de 70%. Em 2003 a ARM anunciou a tecnologia Thumb-2 [7], que aumenta a densidade de código misturando instruções de 16 bits e 32 bits, ambas no mesmo fluxo de instruções. Isso foi possível pela incorporação de acessos a endereços não alinhados no projeto da arquitetura. Thumb-2 tem desempenho superior (ou próximo) ao do conjunto de instruções ARM e densidade de código semelhante ao da ISA Thumb original. A codificação Thumb/Thumb-2 necessita de um decodificador na via de dados do processador.

A técnica MIPS16 [8] é um mecanismo de codificação para instruções MIPS. Cada instrução MIPS16 corresponde a exatamente uma instrução MIPS. Para reduzir o tamanho da instrução para 16 bits, são utilizados três campos da instrução: opcodes, número de registradores e valores imediatos. Há duplo fluxo de instruções (codificadas e originais) e a necessidade de um decodificador de instruções MIPS16 na via de dados do processador. Em testes realizados pelos autores, a taxa de compressão alcançada foi de 60%.

A técnica SPARC16 [5] representa uma escolha entre oportunidade de codificação e impacto do resultado. Para reduzir instruções da ISA SPARCv8 para 16 bits, observou-se que os campos mais importantes para redução, são os de valores imediatos que podem ocupar mais da metade da instrução. A tradução é realizada por meio de um decodificador localizado entre a *cache* de instruções e o *pipeline* do SPARCv8. Conforme experimentos realizados pelos autores, a taxa de codificação média obtida foi de 60% em relação ao SPARCv8 nativo. Também, houve redução significativa no número de *cache misses* para *caches* de vários tamanhos.

A técnica PBIW sobre o processador ρ-VEX [9] codifica todas as instruções da ISA VLIW VEX. A codificação PBIW é baseada em padrões de instruções, sendo que o processo de codificação percorre o programa realizando a fatoração de operandos redundantes e opcodes em dois elementos: instruções codificadas e padrões. PBIW explora a sobrejeção entre o conjunto de instruções e o conjunto de padrões. Os padrões são armazenados em uma memória ou tabela de padrões. A reconstituição da instrução original é realizada por um decodificador na via de dados do processador, que recombina campos da instrução codificada e seu padrão correspondente. Conforme experimentos realizados pelos autores, obteve-se taxa de compressão de 60,97% e aumento na taxa de acertos de até 53,93%.

As técnicas Thumb/Thumb-2, MIPS16 e SPARC16 são restritas a conjuntos de instruções específico. A técnica PBIW é independente de ISA/processador, podendo ser adequada e aplicada em várias outras ISAs alvo. PBIW realiza o mapeamento de diferentes instruções para um mesmo padrão, como uma função sobrejetora entre o conjunto de instruções codificadas e padrões.

III. Infraestrutura de Codificação Baseada em Padrões

A codificação PBIW tem como principal característica ser aplicável em várias *Instruction Set Architectures* (ISAs) diferentes. Com base nessa premissa, foi desenvolvida uma infraestrutra de software [10], com o intuito de facilitar o uso do algoritmo PBIW em diversos tipos de ISA. Essa infraestrutura agiliza a integração entre o algoritmo PBIW e a ISA alvo da codificação.

A. *Técnica de Codificação PBIW*

A técnica de codificação PBIW [1] é focada na exploração da sobrejeção entre conjuntos de instruções codificadas e seus respectivos padrões, com o intuito de reduzir o tamanho do programa em memória. Esta técnica foi inicialmente projetada para atender arquiteturas *Very Long Instruction Word* (VLIW) [11]. A técnica é composta por um algoritmo baseado em fatoração de operandos [3] e por uma tabela de padrões *Pattern cache* (P-*Cache*). O algoritmo percorre as instruções do programa e extrai operandos redundantes. O algoritmo mantém a posição original e o opcode dos operandos em um padrão da instrução. As instruções codificadas são armazenadas como as instruções originais (memória principal ou *cache* de instruções) enquanto que os padrões são armazenados em uma tabela de padrões (dicionário) junto ao hardware decodificador ou mesmo podem ser armazenados em uma memória *cache* de padrões.

O projeto do formato do padrão e instrução codificada PBIW para uma ISA ou processador específico, devem levar em consideração algumas características da arquitetura alvo [1]:

- Número de registradores de leitura;
- Número de registradores de escrita;
- Número de imediatos;
- Tamanho do índice (ponteiro) para a tabela de padrões na P-*Cache*.

Ao projetar a instrução codificada, além das características mencionadas, é necessário que os operandos de uma instrução sejam mantidos na instrução codificada, enquanto que os campos que possuem sinais de controles devem ser armazenados no padrão. Isso se justifica pela possibilidade dos operandos serem lidos no banco de registradores, paralelamente ao processo de codificação de instruções. Essa é uma das características que diferenciam PBIW de outras técnicas de codificação uma vez que a instrução PBIW codificada mantém parte da semântica da instrução original e, com isso, pode ser parcialmente executada em paralelo com a decodificação.

O padrão é uma estrutura de dados que armazena os campos que, combinados aos operandos armazenados na

instrução codificada, reconstitui a instrução original. Cada operação representada no padrão, contém um opcode e um número fixo de campos que representam os operandos. Os campos que representam os operandos no padrão conservam suas posições originais e armazenam o índice referente ao campo na instrução codificada que armazena seu respectivo operando.

B. *Infraestrutura de Codificação PBIW*

Com o intuito de facilitar o uso do algoritmo PBIW, foi desenvolvida uma infraestrutra de software modular e extensível [10] a diversos tipos de ISAs e representações específicas de arquivos de entrada e saída. A infraestrutura de codificação pode ser estendida para suportar alguns tipos de formatos de programas (entradas para a infraestrutura):

- Texto em linguagem de montagem (.s, .asm, etc.);
- Estruturas de dados finais providos pelo *backend* de um compilador;
- Binários *Executable and Linkable Format* (ELF).

Exemplos de algoritmos PBIW genéricos, especificamente otimizados e inicialmente disponíveis na infraestrutura de codificação incluem:

- Codificação PBIW genérica: algoritmo parametrizável capaz de se adequar a diversas ISAs de entradas;
- Codificação *VLIW Example* (VEX) PBIW [9], [10].

A infraestrutura de codificação PBIW, por padrão, está preparada para realizar a codificação PBIW sobre o conjunto de instruções baseado na ISA VEX [12]. Assim, visando a utilização dessa infraestrutura para prover uma ferramenta de codificação para a ISA SPARCv8 foi necessário estender e implementar interfaces de software dessa infraestrutura para adequá-la às entradas e saídas necessárias para codificação PBIW-SPARC.

A arquitetura da infraestrutura é composta por três módulos [10]: o contexto de montagem, o contexto de codificação e o contexto de otimização. O fluxo de dados e controle do processo de codificação PBIW pode ser visto na Figura 1.

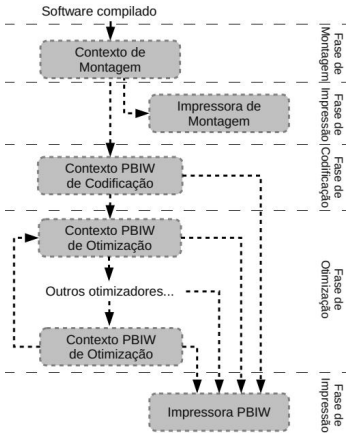

Figura 1. Fluxo de dados da infraestrutura de codificação PBIW [10].

O programa de entrada é processado pelo contexto de montagem que gera uma representação interna. Esta representação interna é totalmente dependente do montador, tipo de arquitetura e da estrutura de montagem. A infraestrutura de codificação, por meio de interfaces que devem ser implementadas, torna a representação interna acessível ao contexto de codificação. As interfaces fornecem métodos de acesso aos dados internos que estão encapsulados, através de uma camada de abstração entre diferentes implementações possíveis dos contextos de montagem e os algoritmos de codificação PBIW. Após o processo de codificação é possível aplicar otimizações sobre o conjunto de padrões e instruções codificadas resultantes.

Depois das etapas de montagem e/ou codificação, pode-se realizar a impressão dos dados gerados, utilizando classes denominadas impressoras. As classes impressoras têm acesso às instruções, padrões e demais dados disponíveis e podem imprimir desde informações de depuração, passando por arquivos de descrição de *hardware* e até mesmo gerar arquivos binários ELF completos.

IV. Esquema de Codificação PBIW Sobre a Arquitetura SPARC

Apesar da larga utilização do conjunto de instruções SPARC [13] tanto em projetos do meio acadêmico quanto comercial, seja no âmbito de sistemas embarcados quanto alto desempenho, a exploração de técnicas de codificação ainda é limitada [5]. Outra motivação para aplicação de técnicas de codificação é atribuída ao processador Leon3 [14], que implementa a arquitetura SPARCv8 e seu código fonte é disponibilizado completamente sob a licença GNU.

A. *Arquitetura SPARCv8*

A arquitetura *Scalable Processor Architecture* (SPARC) está publicada como padrão IEEE 1754-1994. Ela define registradores de propósito geral, de ponto flutuante, de controle/status e 72 instruções, todas codificadas em formatos de 32 bits. Um processador SPARC, logicamente compreende uma Unidade de Inteiro (UI), uma Unidade de Ponto Flutuante (UPF) e opcionalmente, um coprocessador, cada com seus respectivos registradores. A seguir são listadas algumas características da arquitetura SPARCv8 [4]:

1) Alinhamento: Espaço de endereços linear de 32 bits.
2) Poucos modos de endereçamento: Formados por *registrador* + *registrador* ou *registrador* + *imediato*.
3) Tríade de endereços de registradores: A maioria das instruções opera em dois ou três operandos e atribui o resultado em um terceiro registrador.
4) Banco de registradores em janelas *"windowed"*: A qualquer instante estão visíveis ao programa oito registradores inteiros globais, mais uma janela de 24 registradores, que podem ser descritos como uma *cache* de argumentos de procedimentos, valores locais e endereços de retorno.

As instruções da ISA SPARCv8 são organizadas em três formatos, conforme apresentado na Figura 2.

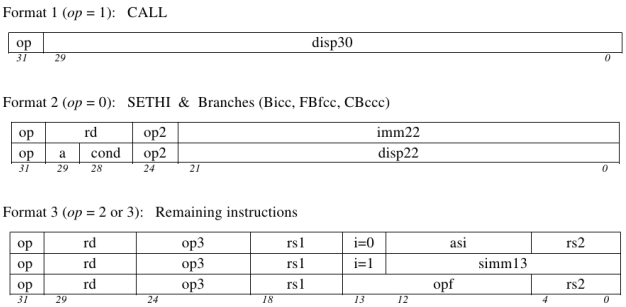

Figura 2. Formatos de instrução SPARC [4].

O campo *op* é o único comum aos três formatos, sendo ele o identificador do formato da instrução. O campo *op* com valor 1, indica o formato 1, utilizado apenas pela instrução CALL, que além do campo *op* possui o campo *disp30*, um imediato de 30 bits. Esse campo é deslocado dois bits à esquerda e somado ao *Program Counter* (PC) quando a instrução CALL é executada. Dessa forma, todos os endereços de memória podem ser alcançados pela instrução CALL. Quando *op* é zero, o formato 2 é utilizado. O formato 3 é utilizado, quando *op* é 3, por instruções de memória ou quando *op* é igual a 2, por instruções aritméticas, lógicas, de deslocamentos e demais instruções.

B. Codificação PBIW-SPARC

Seguindo a especificação de projeto baseada no algoritmo PBIW, a técnica PBIW-SPARC permite a codificação de todas as instruções da ISA SPARCv8. Para cada instrução codificada PBIW-SPARC há exatamente uma correspondente SPARCv8. Nenhuma instrução adicional foi criada. A codificação PBIW-SPARC realiza sua codificação sobre um arquivo binário ELF gerado por uma compilador com *back-end* para o conjunto de instruções SPARCv8.

Após a codificação são gerados um conjunto de instruções codificadas e um conjunto de padrões, armazenados nas seções *.text* e *.pattern* do arquivo binário ELF. O leiaute de padrão e instrução codificada utilizados são armazenados na seção *.layout*.

Para definição do tamanho do padrão e instrução codificada PBIW-SPARC, optou-se por reduzir o tamanho da instrução com relação ao padrão, tendo em vista explorar o reúso de padrões. No entanto, há de se considerar a existência de um campo/ponteiro de índice de padrões junto aos demais campos da instrução codificada. Esse campo indica o endereço/índice de memória em que o respectivo padrão da instrução pode ser encontrado.

Pode-se observar no leiaute dos três formatos de instrução da ISA SPARCv8, que a quantidade de bits dos campos de operandos varia de 15-30 bits, sendo inviável mantê-los integralmente na instrução codificada. Optou-se por armazenar apenas a parte mais significativa dos operandos no padrão e a parte menos significativa e mais susceptível a mudanças, na instrução codificada. A única exceção é o campo *rs* (utilizado pelo formato 3) que é armazenado totalmente na instrução codificada. O leiaute do padrão e da instrução utilizados pela codificação PBIW-SPARC foram definidos com tamanhos de 24 bits e 16 bits respectivamente e são mostrados nas Figuras 3 e 4.

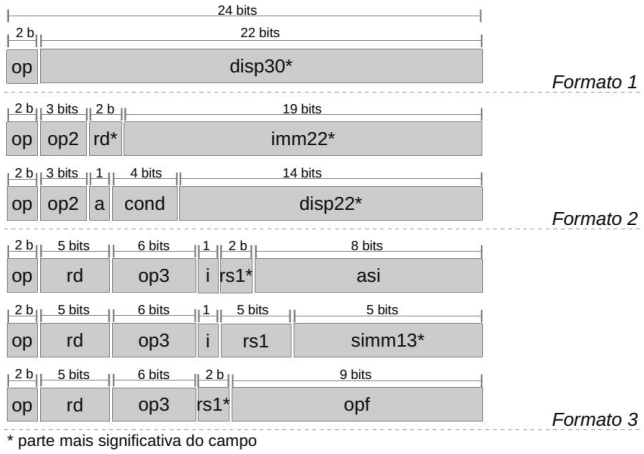

Figura 3. Leiaute do padrão PBIW-SPARC.

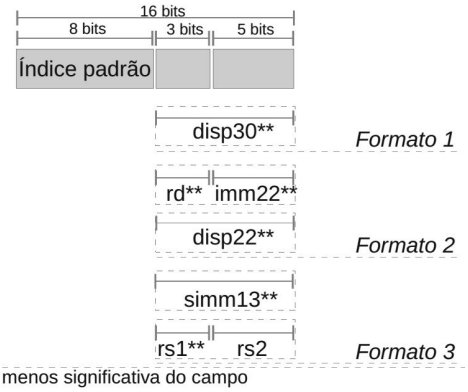

Figura 4. Leiaute de instrução PBIW-SPARC.

A Figura 3 mostra o leiaute do padrão que possui 24 bits e a oganização lógica dos campos de acordo com cada formato de instrução da ISA SPARCv8. Os campos cujo nome contém o símbolo * armazenam apenas a parte mais significativa do campo. A instrução codificada (Figura 4) tem 16 bits e campos com os símbolos ** no nome armazenam a parte mais significativa do campo. O formato possui um campo de 8 bits para o índice do padrão e mais um ou dois campos nos 8 bits restantes. Por exemplo, nas duas situações possíveis para o Formato 2, além do campo de índice do padrão, a instrução codificada pode armazenar, no primeiro caso, os 3 bits menos significativos do campo *rd* e os 5 bits menos significativos do imediato *imm22*. No segundo caso, pode armazenar os 8 bits menos significativos do imediato *simm1*.

A codificação com instruções de 16 bits está limitada a endereçar no máximo 256 padrões (campo de índice de padrão de 8 bits). Para superar essa limitação e permitir a execução de códigos que geram mais que 256 padrões, utilizou-se uma codificação híbrida (parte do programa é codificada e parte permanece original) em alguns programas. Para tanto, utilizou-se o bit 21 da instrução NOP para indicar ao decodificador se a próxima instrução deve ou não ser decodificada.

Após o projeto e formato da instrução codificada e padrão PBIW-SPARC, realizou-se a integração dessa especificação junto à infraestrutura de codificação PBIW. Ao incluir esse novo projeto de codificação na infraestrutura PBIW objetiva-se minimizar o tempo de projeto de um codificador, utilizar mecanismos para validação e avaliação da técnica de codificação e, principalmente, demonstrar a flexibilidade dessa infraestrutura quanto à possibilidade de ser estendida para outros domínios de projetos codificadores. A inclusão do codificador PBIW-SPARC junto à infraestrutura pode ser vista na Figura 5.

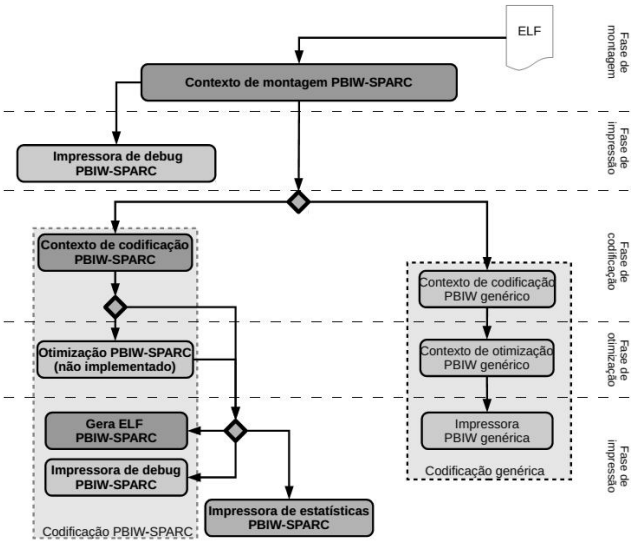

Figura 5. Infraestrutura de codificação PBIW após contexto de codificação PBIW-SPARC.

C. Decodificador PBIW-SPARC

Baseando-se na organização da via de dados e controle do processador *soft-core* Leon3 [14], o circuito decodificador PBIW-SPARC foi implementado no início do estágio de *decode* do *pipeline* desse processador. O diagrama de blocos do circuito decodificador PBIW-SPARC é apresentada na Figura 6.

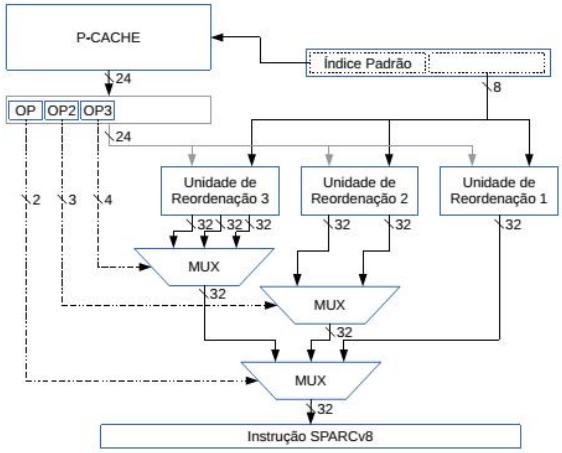

Figura 6. Diagrama de blocos do decodificador PBIW-SPARC para Leon3.

Na parte superior da Figura 6 há a instrução PBIW que pode se encaixar em qualquer combinação de bits dos 3 formatos. A partir da busca da instrução na *cache* de instruções, realiza-se a leitura do campo de índice de padrões e então, o padrão endereçado é buscado na tabela de padrões (P-*cache*). De acordo com o campo *op* presente no padrão define-se o formato da instrução, que em combinação com o campo opcode (exceto o formato 1, que não possui este campo), respectivo ao formato, configura os multiplexadores para selecionar a instrução decodificada. Observe que as entradas de dados dos multiplexadores são oriundas das unidades de reordenação que atuam, paralelamente, reorganizando os dados e sinais de controle de acordo com cada formato específico. Após esse passo, a instrução original já está recomposta e pode ser despachada para o estágio de *decode* do *pipeline*.

V. Experimentos e Resultados

Os experimentos visando a validação e avaliação da técnica de codificação PBIW-SPARC foram realizados utilizando programas dos *benchmarks* MiBench [15] e MediaBench [16]. Todos os programas foram compilados com o *cross-compiler* GCC 3.3.1 para arquitetura SPARCv8, disponibilizado pelo projeto ArchC [17]. Para validação dos resultados e avaliação de desempenho foram utilizados simuladores para o conjunto de instruções SPARCv8 projetados em ArchC. Um desses simuladores SPARCv8, com suporte às instruções codificadas PBIW-SPARC, foi desenvolvido neste trabalho.

A. Experimentos Estáticos

As Figuras 7 e 8 mostram a taxa de compressão para cada programa dos *benchmarks* MiBench e MediaBench, respectivamente. Para os programas desses *benchmarks*, houve a necessidade de utilizar uma codificação híbrida (parte do código possui instruções codificadas) uma vez que, na codificação total o número de padrões gerados extrapola o alcance do campo de índice de padrões.

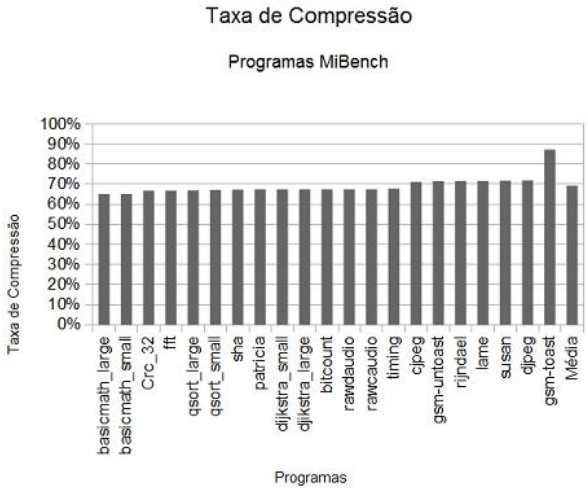

Figura 7. Taxa de compressão para o conjunto de programas do MiBench.

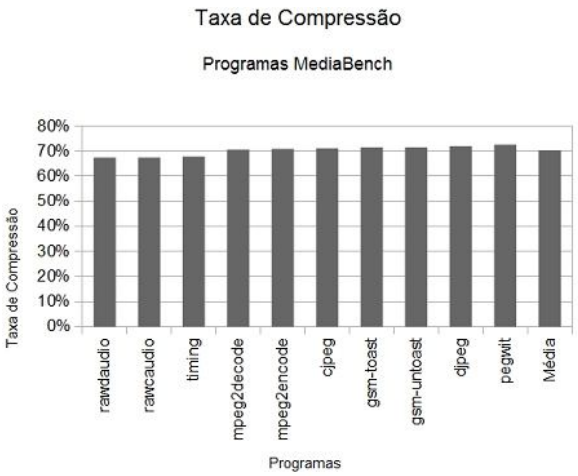

Figura 8. Taxa de compressão para o conjunto de programas do MediaBench.

No *benchmark* Mibench, as taxas de compressão dos programas variaram de 62%-88%. No *benchmark* MediaBench, as taxas de compressão variaram entre 67%-72%. Essa pequena variação acontece como consequência imediata da variação nos tamanhos originais dos programas. O maior valor de reúso de padrões ($TR = \frac{\text{Número de instruções codificadas}}{\text{Número de padrões}}$) obtido foi de 74 padrões por instrução.

Considerando a taxa de compressão realizada apenas pelas instruções codificadas, as taxas de compressão alcançáveis são próximas do limite ótimo de 50%, uma vez que a variação foi de 51%-53% para os programas de ambos os *benchmarks*.

B. Experimentos Dinâmicos

O desempenho dos programas foi avaliado com base nas seguintes métricas: quantidade de instruções (e padrões) executadas; quantidade total de *misses* de instruções e padrões; ciclos totais de execução e *speedup* PBIW-SPARC sobre SPARC. Para obter o impacto do código sobre as *caches* de instruções e padrões, utilizou-se o simulador de *caches* Dinero [18]. Para obter os resultados de ciclos e instruções executadas utilizou-se o simulador SPARCv8 implementado em ArchC [17]. A validação e avaliação de desempenho dos programas codificados PBIW-SPARC também foram realizadas com a adoção de um simulador SPARCv8 implementando em ArchC. Esse simulador foi desenvolvido exclusivamente neste trabalho.

Os experimentos foram realizados considerando *caches* organizadas com conjuntos associativos de 2 vias, blocos de 2 e 4 bytes e tamanho de 1 kbyte. Como a codificação PBIW-SPARC utiliza uma *cache* para padrões e outra para instruções, projetou-se uma *cache* de instruções codificadas de 768 bytes e uma *cache* de padrões de 256 bytes objetivando manter a mesma área de memória dos programas originais.

O *speedup* foi calculado a partir da quantidade de ciclos obtidos pelos programas PBIW-SPARC e SPARC (original). A ocorrência de *cache misses* é penalizada em 10 ciclos para a instrução PBIW-SPARC, para o padrão e para instrução SPARCv8. Portanto, o número de ciclos (CP) gastos na execução de um programa é dado por:

$$\text{CP} = \left(\begin{matrix} \text{Nº instruções} \\ \text{executadas} \end{matrix} + \text{Nº de } \textit{misses} \times \text{Penalidade} \right)$$

Se o programa for codificado, o Número de *misses* deve ser a soma dos *misses* de instrução e padrão PBIW-SPARC.

As Tabelas I e II mostram a análise da quantidade de ciclos de *clock* para execução e *speedup* para os *benchmarks* MiBench e MediaBench.

Tabela 1. Ciclos de execução e desempenho de programas do MiBench.

Ciclos de Execução			
Programa	**SPARCv8**	**PBIW-SPARC**	***speedup***
basicmath_large	77483455	67428934	1,15
basicmath_small	77601804	68171258	1,14
bitcount	46178898	46163377	1,00
crc_32	30241274	30239003	1,00
dijkstra_small	16492	9426	1,75
dijkstra_large	51354962	46720541	1,10
fft	52424195	47847744	1,09
lame	138341008	122425671	1,13
patricia	181867041	153602843	1,18
qsort_large	62399581	49699138	1,25
qsort_small	15529680	14782546	1,05
rijndael	145194341	112322740	1,29
sha	15659525	15358909	1,01
susan	12632	11563	1,09

Tabela 2. Tempo de execução e desempenho de programas do MediaBench.

Tempo de Execução (ciclos)			
Programa	**SPARCv8**	**PBIW-SPARC**	***speedup***
cjpeg	15177774	14466009	0,95
djpeg	4838476	4948036	1,02
gsm-toast	7970	6794	1,17
gsm-untoast	8128	6960	1,18
mpeg2decode	68812369	61893189	1,11
mpeg2encode	61585502	64257217	0,96
pegwit	14233452	14033227	1,01
rawcaudio	7407556	7406485	1,00
rawdaudio	6412012	6410991	1,00
timing	1602161	1596845	1,00

Para o *benchmark* MiBench, o melhor desempenho foi alcançado pelo programa *dijkstra*, com *speedup* de 1,75. O pior desempenho foi obtido com os programas *crc* e *bitcounts*, com *speedup* 1,00. O *speedup* médio para os programas do MiBench foi 1,16. No *benchmark* MediaBench, o pior desempenho foi do programa *cjpeg* e o melhor desempenho foi do programa *gsm-untoast* com 1,18 de *speedup*. Os programas do MediaBench alcançaram um *speedup* médio de 1,04 (1,10 com média ponderada).

Dentre os programas analisados quanto ao *speedup*, observa-se que o programa *rijndael* do *benchmark* MiBench foi avaliado sob a técnica de codificação SPARC16 [5], obtendo *speedup* de 1,12 considerando que cada *miss* na *cache* de instruções tem penalidade de 10 ciclos e *speedup* de 1,33 com penalidade de 50 ciclos. A técnica PBIW-SPARC, considerando penalidade de 10 ciclos para instrução ou padrão, obteve *speedup* de 1,29.

VI. Comparações das codificações PBIW-SPARC e SPARC16

A codificação PBIW-SPARC obteve, para os *benchmarks* avaliados, taxas de compressão entre 62% e 88%, com média de 65%. Os resultados da compressão de código foram obtidos a partir da codificação de cada programa avaliado. A técnica de codificação SPARC16 apresenta taxa de compressão para os *benchmarks* MiBench e Mediabench, variando entre 56,4% e 63,9% e, com média de 61,02%. A taxa de compressão apresentada por SPARC16 é uma estimativa e desconsidera a limitação de referenciar apenas 08 registradores assim como a sobrecarga gerada no código devido a trocas de contexto (troca de seções de código de 16 bits para 32 bits). Ao caracterizar os programas gerados pelos *benchmarks* Mibench e Mediabench com o compilador GCC 3.3.1 e sem o uso de otimizações, todos os programas utilizaram os 32 registradores disponíveis. Tal resultado já era esperado uma vez que a manipulação de registradores em janela da arquitetura SPARC estimula a utilização de vários registradores distintos ao longo do código.

Na comparação da área ocupada pelos decodificadores PBIW-SPARC e SPARC16, realizada sobre um dispositivo FPGA Altera Cyclone II, o decodificador PBIW-SPARC gerou um aumento de área do processador Leon3 de 6,1% enquanto que o decodificador SPARC16 aumentou a área em 9,4%. O circuito decodificador e a técnica de codificação SPARC16 não oferecem suporte a instruções de ponto flutuante, sendo necessário adaptações em ambos, o que pode aumentar a área e número de elementos lógicos do processador. Mesmo sendo um circuito consideravelmente simples e com pouco impacto na área ocupada pelo processador, o decodificador PBIW-SPARC está apto para realizar a decodificação de qualquer programa codificado com a técnica PBIW-SPARC.

VII. Conclusões e Trabalhos Futuros

Este trabalho apresentou o projeto e implementação de um codificador de instruções, baseado na técnica de codificação PBIW, para o conjunto de instruções SPARCv8 (PBIW-SPARC). A aplicação do algoritmo de codificação PBIW-SPARC foi realizada utilizando-se da infraestrutura de codificação PBIW.

Devido a limitada capacidade de endereçamento de padrões da instrução codificada de 16 bits, houve a necessidade de projetar mecanismos para codificação híbrida de programas SPARC. Assim, um programa codificado com PBIW-SPARC suporta tanto instruções codificadas de 16 bits quanto instruções originais de 32 bits. Os programas codificados sob a técnica de codificação PBIW-SPARC obtiveram redução de tamanho do código que variaram entre 38%-11% em todos os programas avaliados. Na experimentação dinâmica, foram avaliados a quantidade total de ciclos executados, baseando-se na taxa de *cache miss*, e desempenho (*speedup*). A variação de *speedup* entre os programas avaliados foi de 0,95-1,75.

Dentre os trabalhos futuros, apresenta-se a possibilidade de avaliar os efeitos do decodificador PBIW-SPARC quanto à potência dissipada, energia e frequência máxima alcançável pelo processador.

VIII. Agradecimentos

Os autores agradecem as agências de fomento de pesquisa CAPES, CNPq e Fundect-MS, além das instituições UFMS e IFMS pelo financiamento de vários projetos de pesquisa desenvolvidos no Laboratório de Sistemas Computacionais de Alto Desempenho (LSCAD) da Faculdade de Computação da UFMS.

References

[1] R. Batistella, "PBIW: Um Esquema de Codificacão Baseado em Padrões de Instrucão," Master's thesis, Unicamp, Brazil, Fevereiro 2008.

[2] R. Santos, R. Batistella, and R. Azevedo, "A Pattern Based Instruction Encoding Technique For High Performance Architectures," *IJHPSA*, vol. 2, no. 2, pp. 71–80, 2009.

[3] G. Araujo, P. Centoducatte, M. Cortes, and R. Pannain, "Code Compression Based on Operand Factorization," in *Proceedings of the 31st Annual ACM/IEEE MICRO*. IEEE Computer Society, 1998, pp. 194–201.

[4] I. SPARC International, "The SPARC Architecture Manual," SPARC International, Inc., Tech. Rep., 1992.

[5] L. L. Ecco, B. C. Lopes, E. C. Xavier, R. Pannain, P. Centoducatte, and R. J. de Azevedo, "SPARC16: A New Compression Approach for the SPARC Architecture," in *Proceedings of the 21st SBAC-PAD*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 169–176.

[6] L. Goundge and S. Segars, "Thumb: Reducing the Cost of 32-bit RISC Performance in Portable and Consumer Applications," *Proceedings of Computer Society Conference*, 1996.

[7] A. Holdings, "ARM1156T2-S Technical Reference Manual," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0338g/index.html, Tech. Rep., 2005-2007, pp. 35-36.

[8] K. D. Kissell, "Mips16: High-density MIPS for the Embedded Market." Real Time Systems, 1997.

[9] R. Marks, F. Araújo, R. Santos, F. Yonehara, and R. Santos, "Design and Implementation of the PBIW Instruction Decoder in a Softcore Embedded Processor," in *13th WSCAD-SSC*. IEEE, 2012, pp. 110–117.

[10] R. A. MARKS, "Infraestrutura para Codificação de Instrucões Baseada em Fatoracão de Padrões," Master's thesis, UFMS, Brazil, Novembro 2012.

[11] M. Len and I. Vaitsman, "VLIW: Old Architecture of the New Generation," Mar. 2011, http://ixbtlabs.com/articles2/vliw/.

[12] J. A. Fisher, P. Faraboschi, and C. Young, Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. Elsevier, 2005. [Online]. Available: http://www.vliw.org/

[13] SPARC International, Inc., http://www.sparc.org, Fevereiro 2012.

[14] Glaiser, "LEON 3 Processor," http://www.gaisler.com/cms/index.php?option=com\content\&task=view\&id=13n&Itemid=53, Mar. 2011.

[15] T. M. A. T. M. Matthew R. Guthaus; Jeffrey S. Ringenberg; Dan Ernst, "MiBench Version 1.0," [online], 2001, http://www.eecs.umich.edu/mibench/.

[16] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench Consortium," [online], 1997, http://euler.slu.edu/~fritts/mediabench/.

[17] L. Computers Systems Laboratory IC UNICAMP, "ArchC - Architechture Description Language," [online], http://archc.sourceforge.net/.

[18] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," [online], 1995, http://www.cs.wisc.edu/~markhill/DineroIV/.