AUTOMATIC LINK-EDITOR GENERATION FOR EMBEDDED CPU CORES

Daniel C. Casarotto e Luiz C. V. dos Santos

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis, SC, Brasil

{casaroto, santos}@inf.ufsc.br

ABSTRACT

SoC design space exploration requires code generation for several CPU core alternatives. However, an embedded software code generation toolkit cannot be developed from scratch for every target CPU under exploration. Nor can it always be reused from standard packages, especially when the CPU core is an ASIP. That's why automatically retargetable tools are required. This paper describes a retargetable technique for link-editor automatic generation from a formal description of the target CPU core. The implementation of the technique relies on the well-known GNU `binutils` package. To make it retargetable, the key is to reuse the architecture-independent libraries and automatically generate the architecture-dependent ones. The technique's correctness and robustness were verified for two target CPUs (MIPS and SPARC) running programs from the *benchmark* MiBench. For experimental validation, we have successfully compared the executable files produced by the generated tools with those produced by available tools from the GNU `binutils` package.

1. INTRODUÇÃO

Como resultado da Lei de Moore, milhões de portas lógicas podem ser implementadas em um único *chip*. Esta imensa oferta de hardware, combinada com a demanda crescente de sistemas embarcados deu origem a sistemas dedicados de hardware e software em um único circuito integrado, os assim-chamados *Systems-on-Chip* (SoCs) [1]. SoCs podem ser construídos com CPUs de propósitos gerais ou CPUs dedicadas (ASIPs).

A exploração do espaço de projeto de SoCs requer a geração de código para diversas alternativas de CPUs. Entretanto, um kit de ferramentas de geração de código para software embarcado não pode ser completamente desenvolvido para cada CPU-alvo explorada. Tampouco pode um tal kit ser reutilizado a partir de um pacote de utilitários padrão, especialmente se a CPU é dedicada (ASIP). Portanto, precisa-se de ferramentas com redirecionamento automático.

A Figura 1.1 ilustra o processo de exploração de soluções alternativas com o auxílio de ferramentas redirecionáveis, geradas automaticamente. Dada a descrição de uma CPU, suas ferramentas são geradas automaticamente. Um código aplicativo escrito em linguagem de alto nível pode desta forma, ser compilado, montado, ligado e simulado para verificar se os requisitos especificados são satisfeitos. Em caso contrário, faz-se uma modificação no conjunto de instruções da CPU ou escolhe-se uma nova CPU a ser explorada. O fluxo é então repetido até que os requisitos sejam satisfeitos.

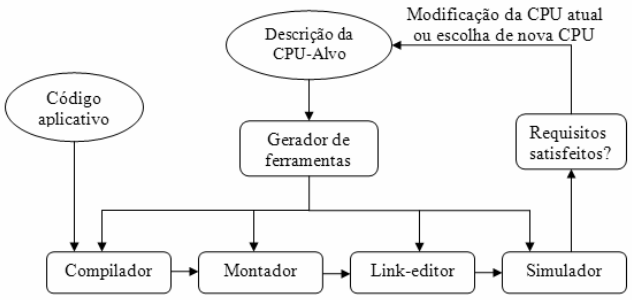

Figura 1.1 – Fluxo de exploração de soluções

Este artigo aborda a geração automaticamente redirecionável de linkeditores. O linkeditor gerado é então utilizado como parte da cadeia de ferramentas para a geração de código durante a exploração de soluções alternativas.

O restante deste artigo é organizado da seguinte forma. A Seção 2 discute trabalhos correlatos. A Seção 3 revisa os conceitos de relocação e ligação. A Seção 4 descreve a técnica proposta e sua implementação. Os resultados experimentais da validação da técnica são apresentados na Seção 5. As principais conclusões e perspectivas de trabalhos futuros são resumidas na Seção 6.

2. TRABALHOS CORRELATOS

Linkeditores são ferramentas de desenvolvimento de software quase tão antigas quanto os próprios computadores; suas primeiras versões datam da década de 40 [2]. Assim, as técnicas para sua construção são bem conhecidas. O desenvolvimento de linkeditores envolve o conhecimento e manipulação de utilitários binários e é bastante dependente da arquitetura-alvo. Por um lado, como o mercado de desktops e servidores é dominado por umas poucas arquiteturas, embora o desenvolvimento de linkeditores seja trabalhoso (pois seu redirecionamento para novas arquiteturas é feito manualmente), o esforço de seu desenvolvimento é amortizado pelo seu reuso intensivo. Isto se traduz na disponibilidade de linkeditores para várias CPUs de propósitos gerais [3].

Por outro lado, no mercado de computação embarcada e, em especial, em aplicações que demandam SoCs, há uma variada gama de processadores dedicados. Neste caso, a cada nova arquitetura explorada, não se justificaria o esforço de desenvolvimento de um novo linkeditor em face das restrições de *time-to-market*.

Assim, no contexto de projeto de SoCs orientados a plataforma, tem sido apontada a necessidade de se gerar automaticamente, para cada CPU-alvo explorada, uma série de utilitários binários, inclusive o linkeditor [4].

Em [4], propõe-se um modelo formal abstrato para a geração de ferramentas redirecionáveis, utilizando como substrato o pacote `binutils`. Dentre as ferramentas descritas, estão um montador e um linkeditor. Para viabilizar a geração do linkeditor, o modelo associa uma informação de relocação a cada campo do formato de instrução. Cada informação de relocação tem os seguintes componentes:

- *Id*: Um número de identifica a relocação.
- *ExpCode*: Indica como é calculado o valor da relocação.
- *RightShift*: Indica o valor do deslocamento à direita que será aplicado ao resultado da relocação.
- *BitSize*: Indica o número de n bits menos significativos que serão considerados do valor final da relocação.
- *BitPos*: Sua função não é explicada no artigo.
- *Complain*: Contém o tipo de verificação de *overflow* a ser implementada pelo linkeditor.

A validação da técnica de geração utilizou o *benchmark* SPECInt2000 [5], e a arquitetura SPARC. Assim, embora o modelo tenha sido proposto para ser genérico, sua validação restringiu-se a uma única CPU-alvo.

Embora várias linguagens de descrição de arquiteturas (ADLs) sejam reportadas na literatura, poucas oferecem ferramentas de domínio público. Este é o caso da ADL ArchC [6]. Dentre suas características destacam-se a capacidade de descrição do conjunto de instruções, precisão de ciclos, suporte à execução multi-ciclo, *pipeline*, hierarquia de memória, descrição do *assembly*, etc. Junto com ArchC, estão disponibilizadas ferramentas de geração automática de simuladores interpretados (`acsim`), simuladores compilados (`accsim`) e montadores (`acasm`). Em particular, o gerador de montadores [1] utiliza a infra-estrutura do pacote `binutils`, numa abordagem similar à descrita em [4].

O pacote `binutils` [7], é composto de ferramentas para a geração, manipulação e visualização de arquivos-objeto, tais como montador (`gas`), linkeditor (`ld`), ferramentas para manipular arquivos-objeto (`ar`, `ranlib`, `strip`, `objcopy`) e ferramentas de visualização dos arquivos (`objdump`, `mn`, `strings`).

Este trabalho é motivado pela oportunidade de adicionar uma nova ferramenta a uma cadeia de ferramentas de domínio público, baseadas na ADL ArchC. A ferramenta proposta, denominada `aclink`, é um gerador redirecionável de linkeditores, obtido através da modificação automática do linkeditor (`ld`) disponível no pacote `binutils`.

Para testar o caráter redirecionável de `aclink`, a ferramenta será validada para mais do que uma CPU-alvo, ao contrário de [4].

3. FUNDAMENTOS

3.1. A linkedição e seus principais conceitos

O papel de um linkeditor é o de viabilizar a manipulação modular de programas, em vez de se ter que tratar um grande bloco monolítico. Cada módulo é compilado e montado como se seu código iniciasse no endereço 0. Ademais, como os módulos não são estanques, um módulo pode conter referências externas para outros módulos. Essencialmente, a função de um linkeditor é a de criar um único bloco de código, realizando duas tarefas principais:

- *Ligação*: consiste na resolução de *referências externas*. Requer a conversão de endereços simbólicos em endereços numéricos.
- *Relocação*: consiste no reposicionamento dos módulos em espaços de endereçamento contígüos. Requer a edição de endereços absolutos utilizados dentro de um módulo, ajustando-os ao seu novo espaço de endereçamento.

A Figura 3.1 ilustra um exemplo de linkedição. Os módulos M1, M2 e M3 representam três arquivos-objeto e E1 representa o arquivo executável (embora seus conteúdos estejam esquematicamente mostrados em *assembly* para melhor visualização do processo). Como cada módulo é posicionado imediatamente após o outro, o endereço inicial de um módulo é o endereço final do módulo anterior mais 1. Esse endereço é conhecido como *endereço base*. Note que há três instruções referenciando endereços absolutos (que correspondem a endereços simbólicos já resolvidos pelo montador no escopo de um módulo). Tais instruções precisam ter seus endereços editados para fins de relocação. O endereço relocado é

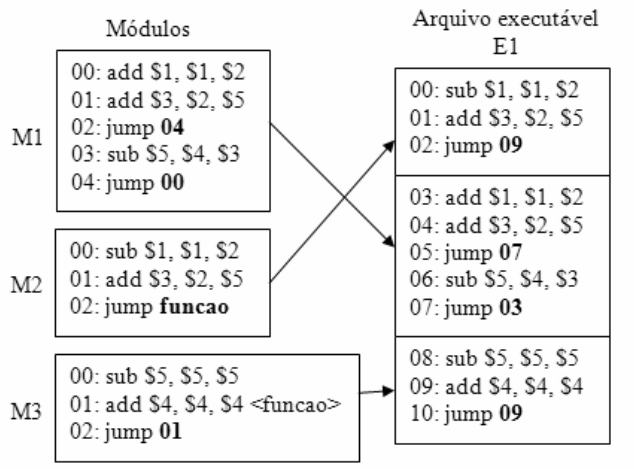

Figura 3.1 – Exemplo do processo de linkedição

obtido pela soma do valor original com o valor do endereço-base do módulo onde reside. Por exemplo, no módulo M1 a instrução `j 04` torna-se `j 07`, pois o endereço-base de M1 em E1 é 3. Note que no módulo M2 há uma referência externa (`j label`) a uma instrução especificada no módulo M3, cujo endereço simbólico precisa ser convertido em numérico. Assim a instrução `j label` do módulo M2 torna-se `j 9` no arquivo executável, uma vez que a função marcada com label no módulo M3 foi relocada para o endereço 09 no arquivo E1.

3.1. Informações de relocação

Para que essas modificações sejam possíveis, é necessário prover informações de relocação, descrevendo *quais* campos da instrução devem ser editados, e *como* devem ser modificados. Por exemplo, considere a instrução "j" do MIPS. O campo a ser editado corresponde aos 26 bits menos significativos da instrução. O novo valor desse campo é obtido da seguinte forma: dado um endereço-alvo de 32 bits, deve-se descartar os 4 bits mais significativos e os 2 bits menos significativos, pois tais bits são concatenados pela CPU, em tempo de execução, para formar o endereço efetivo de memória [8].

As informações de relocação são inseridas nos arquivos-objeto, que são então chamados de arquivos-objeto relocáveis. Um arquivo-objeto relocável é subdividido em seções, sendo que cada seção deve ser ligada com sua congênere em outro arquivo objeto. Um arquivo-objeto relocável consiste de pelo menos três seções:

- *Seção de código*: contém o código executável do programa e é denominada `.text`. É uma seção apenas de leitura.
- *Seção de dados*: contém dados inicializados estaticamente e é denominada `.data`. Contém, por exemplo, dados declarados com a diretiva .word em que um valor de 32 bits é passado como parâmetro. Por exemplo, "`int temp[] = {10, 20, 30};`" irá gerar 3 entradas nessa seção.
- *Seção de dados não inicializados*: é denominada `.bss` e contém, por exemplo, dados declarados com a diretiva `.comm`, que descreve apenas o nome e o tamanho do dado.

4. A FERRAMENTA PROPOSTA

Para viabilizar a geração automática de um linkeditor, pode-se re-utilizar primitivas da ADL para descrever as informações de relocação, mas deve-se implementar algumas extensões no montador e unificar as informações de relocação para viabilizar o redirecionamento.

4.1. Reuso, extensões e restrições visando a automação

4.1.1. Reuso de formatadores para relocação

Procurou-se reusar ao máximo o suporte já existente na ADL para descrever as informações de relocação. Por exemplo, a ADL ArchC possui a palavra reservada `set_asm` que associa uma dada instrução com sua sintaxe assembly. A sintaxe adotada é similar à da função `scanf()` da linguagem C, conforme exemplo a seguir:

```
addi.set_asm("addi %reg, %reg, %exp", rt, rs, imm);
```

No *string* delimitado entre aspas, os campos precedidos por `%` são os formatadores e a cada formatador corresponde um campo da instrução (`rt`, `rs` e `imm`), representados após o *string*. Os dois primeiros formatadores referem-se a registradores e o último (`exp`), a uma expressão aritmética.

É possível reusar modificadores já definidos na ADL para possibilitar a descrição de linkedição de valores, antes mesmo de serem inseridos em seus respectivos campo de instrução. Esses modificadores são listados abaixo, onde n representa um número de bits e onde s|u representa se o n-ésimo bit é considerado como bit de sinal (s) e com isso estendido antes que o valor seja codificado.

- L[n][s|u] - Seleciona os n bits menos significativos do valor associado ao formatador;
- H[n][s|u] - Seleciona os n bits mais significativos do valor associado ao formatador
- R[n] - Adiciona o valor do PC ao valor associado ao formatador, pode-se usar um adendo (n), que é somado ao PC.
- A[n][u|s] - indica que o valor associado ao formatador usa um endereçamento alinhado em n bytes.

Vamos ilustrar o uso dos modificadores com o seguinte exemplo:

```
beq.set_asm("beq %reg, %reg, %expR4A", rs, rt, imm);
```

A construção acima modela o fato de a instrução `beq` possuir um campo que corresponde a uma expressão `%exp`, à qual são aplicados os modificadores `R4A`. Isso significa que ao valor associado a esse formatador (no caso o valor correspondente ao campo imm da instrução) será adicionado o valor do `PC+4`, com alinhamento no tamanho da palavra.

O gerador de montadores `acasm` utiliza esses modificadores para descrever informações para a resolução de labels. Entretanto, como informações de relocação precisam também ser inseridas no arquivo objeto para torná-lo relocável, o `acasm` foi estendido para gerar montadores capazes de inserir tais informações como insumos para o linkeditor.

Há uma exceção que precisa ser tratada: quando a expressão utilizar o modificador R (relativo ao PC), o montador deve gerar as informações de relocação apenas se a instrução contiver referência externa ao módulo onde reside, exclusivamente para fins de ligação. Isso se deve ao fato de que o modificador R é utilizado para desvios relativos ao PC, que não requerem relocação.

4.1.2. Unificação das seções

Como explicado na Seção 3, existem três seções essenciais em um arquivo objeto (`.text`, `.bss`, `.data`), porém algumas arquiteturas possuem outras seções que são variantes destas. No MIPS, por exemplo, são definidas seções denominadas `.rdata` para dados somente de leitura, `.sdata` para dados de pequeno tamanho, e `.sbss` para dados não inicializados de pequeno tamanho, entre outras. O montador `acasm` somente aceita as seções `.data` e `.text`. Da mesma forma, o linkeditor gera apenas a seção `.bss`, não sendo permitidas outras variações. Esta unificação em apenas três seções garante a redirecionalidade para arquiteturas-alvo distintas, sem perda de generalidade.

4.1. Implementação da ferramenta

Basicamente, o linkeditor GNU (`ld`) possui um núcleo contendo suas principais funções e utiliza a biblioteca BFD para acesso a arquivos. A biblioteca BFD por sua vez, possuiu um núcleo para acesso a arquivos, que utiliza duas bibliotecas, uma dependente do formato de arquivo (ELF, COFF, etc.), outra dependente da arquitetura, como esquematizado na Figura 4.1.

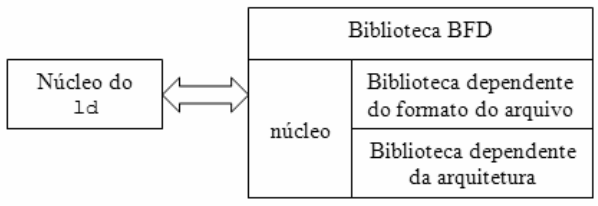

Figura 4.1 – Estrutura do pacote GNU `binutils`

Para a implementação da ferramenta `aclink`, escolheu-se utilizar a biblioteca do formato ELF, que é mantida inalterada, enquanto a biblioteca dependente de arquitetura é gerada automaticamente.

Essencialmente, a biblioteca dependente de arquitetura suporta o tratamento de relocações, na forma de uma tabela contendo *diretivas de relocação* (que descrevem a forma de resolução das relocações). Os passos necessários para se gerar automaticamente a biblioteca dependente de arquitetura são descritos a seguir.

- Passo 1: Para cada instrução encontrada na descrição ADL, verificar se seu formato *assembly* possui algum campo de expressão (exp).
- Passo 2: Em caso positivo, gerar uma diretiva de relocação, utilizando os dados dos modificadores descritos na Seção 4.1. Verificar se uma diretiva de relocação equivalente já existe na tabela de relocação da biblioteca BFD. Se a diretiva não existir, inserí-la na tabela.
- Passo 3: No código do `acasm`, modificar as informações específicas daquela instrução de forma que, quando a instrução for montada, seja também gerada sua informação de relocação, conforme a diretiva gerada no Passo 2.

5. RESULTADOS EXPERIMENTAIS

5.1. Configuração experimental

O pacote GNU `binutils` [7] disponibiliza montadores e ligadores para diversas CPUs-alvo. Tais montadores e ligadores, doravante denominados *originais*, serão utilizados para produzir código executável a ser usado como referência durante a validação. Os montadores e ligadores gerados pelas ferramentas `acasm` e `aclink`, respectivamente, serão doravante denominados *gerados*. Para a validação foi utilizado o *benchmark* Mibench[9], que é representativo de aplicações típicas do mercado de sistemas embarcados.

5.1.1. Validação da correção do ligador gerado

Dada uma CPU-alvo, o montador e o ligador originais são utilizados para gerar um código executável de referência, a ser comparado com o código executável produzido pelo montador e ligador gerados automaticamente. Como o montador gerado já foi validado anteriormente [1], a igualdade dos códigos executáveis é uma evidência experimental da correção do ligador gerado.

5.1.2. Validação da ferramenta geradora

Para verificar a redirecionalidade da ferramenta geradora, o procedimento da Seção 5.1.1 deve ser repetido para várias CPUs. Por enquanto, as CPUs MIPS e SPARC foram utilizadas para a validação, uma vez que os

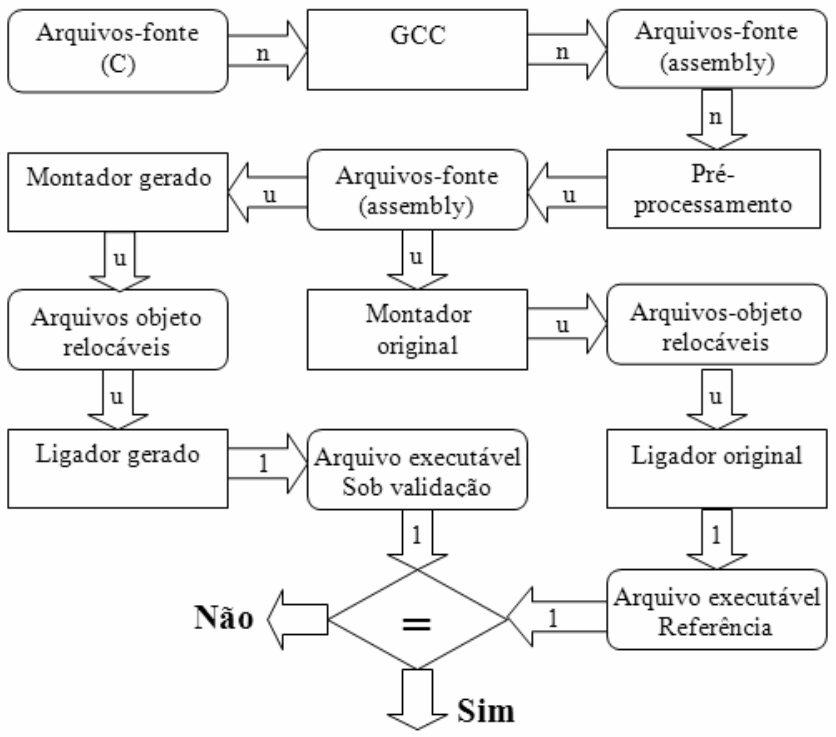

Figura 5.1 – Fluxo de validação do gerador de linkeditores

respectivos modelos são os mais robustos disponíveis em [10]. O procedimento deverá ser repetido para mais CPUs no futuro.

5.1.3. Fluxo de validação

A Figura 5.1 ilustra o fluxo de validação da ferramenta `aclink`. Dado um programa composto de n arquivos-fonte, o compilador cruzado GNU `gcc` produz n arquivos com código *assembly*. Para compatibilidade com o montador, esses n arquivos são então passados por um filtro, similar ao descrito em [1], que elimina diretivas de montagem dependentes de arquitetura, ou as substitui por diretivas genéricas suportadas pelo `acasm`, resultando em um conjunto de u arquivos. Em seguida, o fluxo bifurca em dois ramos distintos, um que utiliza as ferramentas originais, outro que utiliza as ferramentas geradas. Em cada ramo, os arquivos *assembly* são montados, gerando arquivos relocáveis e, depois, linkeditados, dando assim origem a dois arquivos executáveis, um que servirá de referência e outro sob validação. Ao final, verifica-se a igualdade dos arquivos executáveis produzidos em ambos os ramos do fluxo.

A comparação dos arquivos executáveis foi implementada da seguinte forma. Para cada arquivo executável, o conteúdo de cada seção (`.text, .data e .bss`) foi extraído para um arquivo texto (em formato hexadecimal) através do utilitário `readelf` (que é independente de arquitetura). Em seguida, cada arquivo texto correspondente a uma seção foi comparado com seu congênere, através do utilitário `diff`. Como os módulos podem ser ligados em diferentes ordens, diferentes códigos executáveis equivalentes podem ser gerados para um mesmo código-fonte. Ora, para garantir que a comparação seja feita de forma simples é preciso garantir que ambos os ramos do fluxo de validação usem a mesma ordem, de forma que o teste de equivalência possa ser implementado através de uma mera comparação de igualdade. Para isso, em ambos os ramos definiram-se endereços comuns de início de cada seção (`.text, .data e .bss`), para que os símbolos fiquem na mesma posição em ambos os ramos. Devido à incompatibilidade do linkeditor gerado com as bibliotecas que foram utilizadas pelo linkeditor original (por exemplo, a biblioteca `newlib`), e para evitar a necessidade de nova compilação e linkedição dessas bibliotecas com o montador e linkeditor gerados, os símbolos (contidos no *assembly*) que correspondem a essas bibliotecas externas foram adicionados em um arquivo separado, podendo desta forma serem resolvidos pelo linkeditor. Como o objetivo é permitir uma mera comparação de arquivos gerados, este procedimento não altera a validade do experimento.

5.2. Análise dos resultados

O fluxo de validação ilustrado na Figura 5.1 foi repetido para cada um dos programas do *benchmark* adotado e para cada uma das CPUs-alvo escolhidas, dando origem aos resultados mostrados nas Tabelas 5.1 e 5.2. Nas tabelas, as duas primeiras colunas identificam o programa do

benchmark e a terceira mostra o número de arquivos objeto a serem ligados. A quarta coluna apresenta o número de relocações necessárias durante a ligação. As três últimas colunas apresentam o tamanho de cada uma das seções do código executável, expresso em bytes.

Para todos os programas das Tabelas 5.1 e 5.2, o código executável produzido pelas ferramentas geradas coincidiu com código de referência.

Observe que o número de arquivos ligados varia entre 1 e 60, gerando arquivos executáveis que chegam à ordem de 400KB. Note também que há casos em que mais de 8000 relocações foram necessárias. O fato de se ter obtido resultados corretos para tamanha diversidade de programas reais, com uma grande variação na faixa de relocações, número de arquivos e tamanho de código é uma evidência experimental da robustez dos ligadores gerados pela ferramenta `aclink`.

9. CONCLUSÕES E PERSPECTIVAS

Este artigo apresenta uma técnica para a geração automática de linkeditores a partir da descrição da CPU-alvo usando a ADL ArchC. A implementação da técnica baseia-se no bem conhecido pacote `GNU binutils`, onde as bibliotecas independentes da CPU são mantidas e as dela dependentes são geradas automaticamente pela ferramenta `aclink`. Para tanto, foram realizadas extensões na ADL e modificações no `acasm`, o gerador de montadores original do pacote ArchC. A corretude e a robustez da técnica proposta foram comprovadas ao se comparar os arquivos linkeditados pela ferramenta gerada com os da ferramenta original para as arquiteturas MIPS e SPARC, utilizando o *benchmark* MiBench [9]. Como trabalho futuro pretende-se investigar a eficiência da ferramenta ao se utilizar de linkedição dinâmica e as respectivas modificações para viabilizá-la.

10. REFERÊNCIAS

[1] A. Baldassin. “Geração Automática de Montadores em ArchC.” Master thesis, Instituto de Computação, UNICAMP, Campinas, Março 2005.

[2] John R. Levine, *Linkers and Loaders*, Morgan Kufmann Publishers, 2000

[3] GNU Unix Toolset. Informações e arquivos disponíveis em http://www.gnu.org

[4] M. Abbaspour and J. Zhu. “Retargetable binary utilities”. In Proceedings of the 39th Conference on Design Automation, pages 331-336. ACM Press, June 2002.

[5] John L. Henning. “SPEC CPU 2000: Measuring CPU performance in the new millennium”. IEEE Computer, Vol. 33, Nº 7, pp. 28-35, July 2000.

Benchmark	Nº arq.	Relo-cações	Tamanho das seções (bytes) .text	.data	.bss
basicmath_s	4	346	5056	528	0
basicmath_l	4	442	6.244	664	0
bitcount	9	119	4900	964	0
qsort_small	1	26	1032	64	0
qsort_large	1	53	1804	120	0
susan	1	620	64628	1752	0
dijkstra	1	169	2472	204	40840
blowfish	7	97	19116	4416	0
sha	2	40	3232	68	0
crc32	1	22	1288	1044	0
adpcm	2	50	2500	460	2516
fft	3	212	5872	392	0
djpeg	60	3815	285500	11184	16
cjpeg	60	3707	284776	11056	16
typeset	1	8426	29668	426884	7685
ispell	1	45	1140	376	0
stringsearch_s	4	406	5604	2800	2072
stringsearch_l	4	2956	5604	15384	2072

Tabela 5.1 – Resultados obtidos com a CPU MIPS

Benchmark	Nº arq.	Relo-cações	Tamanho das seções (bytes) .text	.data	.bss
basicmath_s	4	278	5268	568	0
basicmath_l	4	353	6324	704	0
bitcount	9	106	4092	1000	0
qsort_small	1	22	1172	80	0
qsort_large	1	44	2064	136	0
susan	1	837	59444	1576	0
dijkstra	1	161	2444	232	40840
blowfish	7	76	16484	4352	0
sha	2	25	2844	72	0
crc32	1	17	1256	1048	0
adpcm	2	39	2388	472	2516
fft	3	189	5552	440	0
djpeg	60	2135	240684	11504	16
cjpeg	60	2022	239872	11360	16
typeset	1	8217	32652	428448	7688
ispell	1	39	1172	384	0
stringsearch_s	4	370	5328	3000	4140
stringsearch_l	4	2920	5596	15792	4140

Tabela 5.2 – Resultados obtidos com a CPU SPARC

[6] S. Rigo. “ArchC: Uma linguagem de descrição de arquiteturas”. PhD thesis, Instituto de Computação, UNICAMP, Campinas, Julho 2004.

[7] R. H. Pesch and J. M. Osier. *The GNU binary utilities*. Free Software Foundation, Inc., May 1993.

[8] Hennessy, J. L. and Patterson, D. A. 1998 *Computer organization and design (2nd ed.): the hardware/software interface*. Morgan Kaufmann Publishers Inc.

[9] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. “MiBench: A free, commercially representative embedded benchmark suite”. In Proceedings of the 4th Annual IEEEWorkshop on Workload Characterization, pages 3-14, December 2001.

[10] The ArchC Architecture Description Language web-page. Informações e arquivos disponíveis em www.archc.org