Estudando Arquiteturas RISC–MIPS através de *Cross-Compilers*: Uma Análise Comparativa

Cristina Meinhardt, Odorico Machado Mendizabal,
Silvia Silva da Costa Botelho

Fundação Universidade Federal do Rio Grande - FURG
Av. Itália, km 8 - Rio Grande, Telefone: 53 233 8600
{cristina,odo}@ecomp.furg.br, silviacb@ee.furg.br

Introdução

A busca pelo alto desempenho fez com que surgissem duas tecnologias bem difundidas na área de arquitetura de computadores, as arquiteturas CISC (*Complex Instruction Set Computer*) e RISC (*Reduced Instrucion Set Computer*) [Stallings, 2002]. No ensino de arquiteturas de computadores deve-se apresentar e detalhar as características destas arquiteturas, contrastando e comparando as técnicas desenvolvidas para as implementações. Também é aconselhada a comparação da execução de códigos codificados em uma linguagem de alto nív el em arquiteturas distintas, podendo-se, assim, observar as diferenç as de comportamento diante os mesmos comandos. Entretanto, nem todas as arquiteturas estão à disposição para tais práticas. Neste contexto salienta-se a importância de simuladores de arquiteturas e *cross-compliers* (compiladores que geram código de máquina para as arquiteturas pretendidas, mas rodam sobre outra arquitetura). Porém, não são muitos os *cross-compilers* disponív eis. Mais ainda, percebe-se a carência de uma análise mais detalhada das características de tais ferramentas na sua utilização como material para apoio didático.

A proposta deste trabalho é analisar algumas ferramentas de apoio ao ensino de arquiteturas de computadores, como simuladores e *cross-compilers* para o processador RISC MIPS 2000. Pretende-se ressaltar a experiência obtida na utilização de *cross-compilers* para o MIPS em conjunto com o simulador SPIM do MIPS e demonstrar diferenç as encontradas entre a definição da arquitetura [Patterson and Hennessy, 1998] e as visualizadas e executadas nos códigos gerados.

A arquitetura MIPS

O conjunto de registradores do MIPS, mostrado na tabela 1, oferece ao programador ou compilador, benefícios quando operações intermediárias são executadas, pois possui vários registradores temporários. A passagem de parâmetros também é otimizada, pois contém dois tipos diferentes de registradores temporários, sendo que as variáveis locais aos procedimentos não precisam ter seus valores restaurado fora dos mesmos. Para este propósito, o MIPS possui registradores temporários (t0-t9) que não têm seus valores preservados em chamadas de rotina [Patterson and Hennessy, 1998].

Registrador	Número	Uso
ra	31	Endereç o de retorno de procedimentos
zero	0	Constante 0
at	1	Reservado ao *assembler*
v0-v1	2-3	Expressões e resultados de funções
a0-a3	4-7	Argumentos 1-4
t0-t9	8-15, 24-25	Valores temporários a descartar
s0-s7	16-23	Valores temporários a preservar
k0-k1	26-27	Reservados ao S.O (Kernel)
gp	28	Apontador para a área global de dados
sp	29	Apontador para a pilha
fp	30	Apontador para a estrutura

Tabela 1: Conjunto de Registradores do MIPS 2000

Simulando uma máquina MIPS

A simulação do processador MIPS foi feita através do SPIM [Larus, 2001]. O SPIM é um simulador da arquitetura MIPS R2000/R3000, que oferece um depurador simples, um conjunto reduzido dos serviç os do sistema operacional e executa programas em Assembly. Comparado com o MIPS, difere no tempo de execução das instruções, não possui caches, nem atrasos no acesso a memória, nem refletecorretamente o retardamento de operações de ponto flutuante,multiplicaç ões ou divisões [Patterson and Hennessy, 1998]. Outro conceito importante, presente no SPIM, é o de pseudo-instrução. Este expande uma instrução em várias sub-instruções de máquina. Quando executa-se passo-a-passo ou examina-se a memória, a instrução vista é ligeiramente diferente daquela do programa fonte. A correspondência entre os dois conjuntos de instruções é bastante simples pois o SPIM não reorganiza as instruções para preencher os espaç os de retardamento (*delay slots*) [Patterson and Hennessy, 1998].

Estes aspectos não chegam a comprometer a análise proposta neste artigo, pois será mostrado que aspectos de otimização de código muito mais previsív eis que técnicas como o *delay branch* não são desempenhadas por alguns *cross-compilers*.

A geração do código Assembly para o MIPS foi feita através do uso dos *cross-compilers* LCC [Navia, 2003] e o kit mips-linux-gcc [Morsiani and Davoli, 2002] capazes de gerar o código Assembly do MIPS a partir de programas escritos em C.

"Cross-Compilando" um código para o MIPS

Foram compilados códigos em C nos *cross-compilers* mips-linux-gcc e LCC e comparados com o comportamento do MIPS descrito em [Patterson and Hennessy, 1998]. Analisando os resultados observou-se que o código gerado pelo mips-linux-gcc não é otimizado para a arquitetura, pois utiliza os registradores de resultado ao invés dos registradores temporários utilizados na bibliografia.

A primeira coluna da tabela 2 mostra um exemplo de um programa recursivo escrito em C, as saídas Assembly geradas pelos *cross-compilers*, a partir do qual serão feitas algumas considerações. Nas linhas 60, 62 da segunda coluna da tabela 2 observa-se que os operandos são carregados da memória nos registradores de resultados. Devido à necessidade de mais registradores temporários, o mips-linux-gcc utiliza o registrador de

(a)	Código C	(b)	Assembly gcc			(c)	Assembly lcc
1	int sum(int, int);	14	main:	61	blez $2,.L3	7	main:
2	int main(void){	15	frame $fp,40,$31	62	lw $3,40($fp)	8	.frame $sp,24,$31
3	sum(3,0);	16	.mask 0xd0000000,-8	63	addu $2,$3,-1	9	addu $sp,$sp,-24
4	return 1;	17	.fmask 0x00000000,0	64	lw $3,44($fp)	10	.mask 0x80000000,-8
5	}	18	.set noreorder	65	lw $4,40($fp)	11	sw $31,16($sp)
6	int sum(int n, int acc){	20	.set reorder	66	addu $3,$3,$4	12	la $4,3
7	if (n > 0)	21	subu $sp,$sp,40	67	move $4,$2	13	move $5,$0
8	return sum(n -1, acc +n);	23	sw $31,32($sp)	68	move $5,$3	14	jal sum
9	else	24	sw $fp,28($sp)	69	jal sum	15	la $2,1
10	return acc;	25	sw $28,24($sp)	70	move $3,$2	16	L.1:
11	}	26	move $fp,$sp	71	move $2,$3	17	lw $31,16($sp)
		27	li $4,0x00000003	72	j .L2	18	addu $sp,$sp,24
		28	move $5,$0	73	j .L4	19	j $31
		29	jal sum	74	.L3:	20	.end main
		30	li $2,0x00000001	75	lw $2,44($fp)	21	.globl sum
		31	j .L1	76	j .L2	22	.text
		32	.L1:	77	.L4:	23	.align 2
		33	move $sp,$fp	78	.L2:	24	.ent sum
		34	lw $31,32($sp)	79	move $sp,$fp	25	sum:
		35	lw $fp,28($sp)	80	lw $31,32($sp)	26	.frame $sp,24,$31
		36	addu $sp,$sp,40	81	lw $fp,28($sp)	27	addu $sp,$sp,-24
		37	j $31	82	addu $sp,$sp,40	28	.mask 0x80000000,-8
		38	.end main	83	j $31	29	sw $31,16($sp)
		39	.Lfe1:	84	.end sum	30	sw $4,24($sp)
		41	.align 2	85	.Lfe2:	31	sw $5,28($sp)
		42	.globl sum			32	lw $24,0+24($sp)
		44	.ent sum			33	ble $24,$0,L.3
		45	sum:			34	lw $24,0+24($sp)
		46	.frame $fp,40,$31			35	subu $4,$24,1
		47	.mask 0xd0000000,-8			36	lw $15,4+24($sp)
		48	.fmask 0x00000000,0			37	addu $5,$15,$24
		49	.set noreorder			38	jal sum
		51	.set reorder			39	move $24,$2
		52	subu $sp,$sp,40			40	b L.2
		54	sw $31,32($sp)			41	L.3:
		55	sw $fp,28($sp)			42	lw $2,4+24($sp)
		56	sw $28,24($sp)			43	L.2:
		57	move $fp,$sp			44	lw $31,16($sp)
		58	sw $4,40($fp)			45	addu $sp,$sp,24
		59	sw $5,44($fp)			46	j $31
		60	lw $2,40($fp)			47	.end sum

Tabela 2: (a) *código em C*, (b) *cógido Assembly do gcc*, (c) *código Assembly do lcc*

passagem de parâmetros R4 na linha 65. A falta de otimização também se manifesta no fato do mips-linux-gcc sempre trabalhar referenciando a pilha do frame (registrador fp), ao invés de utilizar diretamente o registrador sp quando somente existe uma única pilha no programa. Por isto, o mips-linux-gcc realiza mais operações de acesso a memória na carga e armazenamento do endereç o da pilha do frame a cada chamada recursiva da rotina.

O LCC aborda o problema do mesmo modo que a referência, utilizando os registradores temporários para armazenamento dos operandos [Patterson and Hennessy, 1998], ver terceira coluna da tabela 2 linhas 32 e 36. Otimiza o trabalho com registradores armazenando o resultado das operações necessárias no cálculo de parâmetros diretamente nos registradores R4 e R5, ver terceira coluna da tabela 2.

Conclusões

Na análise do desempenho dos códigos gerados pelo *cross-compiler* mips-linux-gcc notou-se que o código produzido por esta ferramenta não usufrui das vantagens oferecidas pela arquitetura MIPS.

O compilador LCC mostrou tirar maior proveito da arquitetura proposta pelo MIPS. Alguns códigos simples foram testados com os dois *cross-compilers* e este foi o único que utilizou os registradores temporários para somas intermediárias em operações que utilizam mais que dois operadores. Para todos os códigos testados, o *cross-compiler* mips-linux-gcc não utilizou nenhuma vez os registradores temporários do MIPS (t0-t9 e s0-s7), mostrando ineficiência, pois em vez disso, os dados eram sempre alternados entre os registradores de argumento (a0-a3) e a pilha.

Uma peculiaridade foi percebida em todos os *cross-compilers* utilizados, quando uma atribuição de inteiro simples era convertida em Assembly, o valor a ser atribuído era carregado em um registrador e sempre, na instrução subseqüente , era colocado na pilha.

Em programas escritos em C, onde a variável é indicada como *register*, o compilador, não adiciona instruções de pilha para estes valores, mantendo eles apenas em registrador. Para estes casos em que o programador deixa explicito o uso de registradores para a alocação da variável, pode-se notar a otimização de código para ambos os *cross-compilers*.

Finalmente, conclui-se que, para o uso didático em curso de Arquitetura de Alto Desempenho, que averiguam o comportamento de máquinas MIPS, mais precisamente através do uso de simuladores e *cross-compilers*, o LCC apresentou resultados bastante semelhantes aos esperados para um compilador próprio para o MIPS.

Referências

[Larus, 2001] Larus, J. R. (2001). Spim s20: A mips r2000 simulator. Disponív el em: www.cs.wisc.edu/ larus/SPIM Acesso em: outubro 2003.

[Morsiani and Davoli, 2002] Morsiani, M. and Davoli, R. (2002). Mps: un simulatore general purpose per la ricerca e la didattica. Disponív el em:http://www.cs.unibo.it/mps/ Acesso em: outubro 2003.

[Navia, 2003] Navia, J. (2003). lcc-win32: A compiler system for windows. Disponív el em: http://www.cs.virginia.edu/ lcc-win32/ Acesso em: outubro 2003.

[Patterson and Hennessy, 1998] Patterson, D. A. and Hennessy, J. L. (1998). *Computer Organization and Design the Hardware/Software interface*. Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition.

[Stallings, 2002] Stallings, W. (2002). *Arquitetura e Organização de Computadores: Projeto para Desempenho*. Prentice Hall, São Paulo, BR, quinta edition.