Projeto 2
Laboratório de Sistemas Computacionais

Victor Accarini D'Lima
RA105753
Grupo 15

9 de maio de 2014

1 Introdução e Objetivos

Esse projeto tem como objetivo verificar as diferenças de desempenho entre diferentes implementações de um processador MIPS utilizando o simulador ArchC. Para verificar essas diferenças foram utilizadas 7 configurações diferentes, variando entre implementações de um *pipeline* de 5 estágios sem *forwards*, sem *branch predictor*, para um pipeline de 5 estágios superescalar e um pipeline de 7 estágios ambos com controle de *hazards*, para cada uma dessas configurações foram utilizadas também 4 configurações de cache.

Os 3 programas utilizados para a avaliação de desempenho foram: Um algoritmo para encontrar o menor caminho entre dois pontos (Dijkstra), um encoder de jpeg (JPEG) e um gerador de chaves criptográficas SHA (SHA), todos disponibilizados no repositório do professor.

Com os resutados obtidos conseguimos verificar as diferenças que técnicas simples como forward podem gerar uma grande diferença na velocidade de execução de um programa, assim como a importância de um branch prediction para a diminuição de penalidades na execução de um programa utilizando assim o máximo da capacidade do processador.

2 Metodologia

Considerando que o simulador disponibilizado executa uma instrução por vez, foram implementadas estruturas para simular um pipeline e seus dispositivos de controles de *hazards*. Ao rodar os programas de teste no simulador os valores das penalidades e ganhos em ciclos de cada configuração simulada eram mostrados no final da execução.

Para a simulação dos dispositivos de cache foi utilizado o programa DineroIV, que dado a configuração de cache pedida e as instruções, gravações e leituras da memória retornava a porcentagem de falhas de acesso a cache.

2.1 Cache

As quatro configurações de cache selecionadas para a avaliação foram:

Cache 1
Cache de instruções:
L1 - Tamanho: 2k Blocos: 4bytes 1-Associativo
Cache de Dados:
L1 - Tamanho: 4k Blocos: 16bytes 2-Associativo

Cache 2
Cache de instruções:
L1 - Tamanho: 1k Blocos: 4bytes 2-Associativo
Cache de Dados:
L1 - Tamanho: 8k Blocos: 16bytes 4-Associativo

Cache 3
Cache de instruções:
L1 - Tamanho: 2k Blocos: 4bytes 1-Associativo
L2 - Tamanho: 8k Blocos: 1bytes 2-Associativo
Cache de Dados:
L1 - Tamanho: 4k Blocos: 16bytes 4-Associativo
L2 - Tamanho: 32k Blocos: 64bytes 4-Associativo

Cache 4
Cache de instruções:
L1 - Tamanho: 1k Blocos: 4bytes 2-Associativo
L2 - Tamanho: 8k Blocos: 1bytes 2-Associativo
Cache de Dados:
L1 - Tamanho: 16k Blocos: 8bytes 1-Associativo
L2 - Tamanho: 32k Blocos: 64bytes 4-Associativo

Foram testadas diversas outras configurações para as caches, porém essas configurações foram as que retornaram os resultados mais plausíveis, ou seja, com *miss* entre 2-8% o que é próximo da margem encontrada na literatura.

2.2 Arquiteturas

Durante a simulação observamos diversos eventos, o primeiro é o nº de instruções executadas, seguido pela quantidade de *load-use hazards(LUH)*, *forwards*, *branchs w/ static predictor(BSP)*, *branchs w/ local saturating counter predictor(BLSCP)*, *branchs w/ local two-level adaptive predictor(BLTAP)* e no caso da arquitetura superescalar o nº de instruções em paralelo.

Na arquitetura superescalar a divisão do pipeline ocorre do estágio ID para o EX onde um pipeline trata de instruções de load-store e outro do resto. A arquitetura com o pipeline de 7 estágios adiciona 2 novos estágios um depois do estágio IF e um estágio antes do estágio MEM, o que permite um aumento nos ciclos por segundo do processador devido a maior granularidade do pipeline.

Na tabela abaixo é mostrada cada arquitetura considerada e suas funcionalidades:

Arquiteturas/Funcionalidades	Modelo 1	Modelo 2	Modelo 3	Modelo 4	Modelo 5	Modelo 6	Modelo 7
Estágios de *Pipeline*	5	5	5	5	5	7	7
LUH	Não	Sim	Sim	Sim	Sim	Sim	Sim
Forwards	Não	Sim	Sim	Sim	Sim	Sim	Sim
BSP	Sim	Sim	Não	Não	Não	Não	Não
BLSCP	Não	Não	Sim	Não	Não	Sim	Não
BLTAP	Não	Não	Não	Sim	Sim	Não	Sim
Instruções em Paralelo	Não	Não	Não	Não	Sim	Não	Não

Tabela 1: Modelos analisados.

2.3 *Branch Predictors*

Algumas ressalvas precisam ser feitas com relação ao BLSCP e BLTAP pois na simulação foi considerado que as tabelas utilizadas para guardar os valores referentes a cada branch possui capacidade para alocar todos os branchs dos programas utilizados.

Nos modelos o branch só é verificado no estágio EX do pipeline portanto para o pipeline de 5 estágios há uma penalidade de 3 ciclos e para o de 7 há uma penalidade de 4 ciclos. O branch predictor fica posicionado no inicio do estágio ID podendo alterar o PC antes de se buscar a próxima instrução. No pipeline de 7 estágios o branch prediction fica no início estágio depois do IF e também altera o PC antes da busca da próxima instrução.

2.3.1 Saturating Counter

Esse branch predictor é bom quando há uma repetição no comportamento dos branchs como , por exemplo, em loops.

Cada instrução de branch no código possui um registrador de 2 bits que funciona como uma máquina de estado com 4 estados, strongly not taken(SNT), not taken(NT), taken(T) e strongly taken(ST). Caso esteja em um dos tres primeiros estágios(SNT, NT e T) e ocorrer um pulo então anda uma posição em direção ao estado ST e caso esteja nos dois primeiros estágios paga uma penalidade por ter previsto errado um branch. O mesmo ocorre no inverso caso o branch não pule andasse uma posição na direção do SNT e se estiver prevendo que irá pular ocorre uma penalidade.

2.3.2 Two-Level Adaptative

Esse branch predictor permite identificar padrões no comportamento de um branch, por exemplo, um 'if' que sempre pula na terceira vez para o 'else'.

Ele utiliza um registrador de 2 bits que guarda a história recente do branch. Para cada pulo ele faz um shift-left e guarda o valor 1 no bit menos significativo, e no caso de não pular ele faz o shift só que guarda o valor 0. Portanto há 4 possibilidades de valor no registrador: 00, 01, 10, 11.

O registrador é usado como index de uma tabela de 4 posições, onde em cada posição há um saturating counter que guarda o que ocorre com mais frequencia depois de uma determinada sequencia de ações, por exemplo, digamos que um branch segue a sequência 0110110110... onde 1 é pulo e 0 é não-pulo, depois de 4 iterações o registrador teria o valor 10 e acessaria a terceira posição da tabela onde o contador estaria em SNT porém depois do 10 há sempre um pulo então ele soma um no counter e continua, depois de 2 iterações o valor do counter estará como T e a partir dai todas a vezes, que aparecer no registrador 10 ele irá prever um pulo.

3 Análise de Dados

3.1 Caches

Utilizando o simulador DineroIV com as configurações apresentadas acima foi calculado o número de ciclos causados por *cache misses*, para o cálculo foi considerada uma penalidade de 1 ciclos no acesso a cache L1, 10 ciclos no acesso a cache L2 e 1000 ciclos no acesso a memória RAM. Portanto o número de ciclos gastos é dados por $L1 + L1_{Miss} * 1000$ para as duas primeiras configurações e $L1 + L2 * 10 + L2_{Miss} * 1000$ para as duas últimas.

	Dijkstra				JPEG				SHA			
	Cache 1	Cache 2	Cache 3	Cache 4	Cache 1	Cache 2	Cache 3	Cache 4	Cache 1	Cache 2	Cache 3	Cache 4
Nº Instruções L1	285278937	285278937	285278937	285278937	29474602	29474602	29474602	29474602	13035973	13035973	13035973	13035973
Nº Dados L1	61259116	61259116	61259116	61259116	7279180	7279180	7279180	7279180	2498746	2498746	2498746	2498746
Cache L1 I Miss	1,74%	2,65%	1,74%	2,65%	5,39%	7,22%	5,39%	7,22%	0,08%	0,68%	0,08%	0,68%
Cache L1 D Miss	8,01%	3,46%	7,86%	6,06%	4,50%	2,47%	4,18%	4,75%	1,74%	1,07%	1,71%	0,88%
Cache L2 I Miss			2,68%	1,76%			5,23%	3,78%			20,07%	2,25%
Cache L2 D Miss			3,91%	5,06%			8,25%	6,46%			0,29%	0,63%
Cache L1 Instrução	4963854	7559892	4963854	7559892	1588681	2128066	1588681	2128066	10429	88645	10429	88645
Cache L1 Dados	4906855	2119565	4814967	3712302	327563	179796	304270	345761	43478	26737	42729	21989
Cache L2 Instrução	0	0	133031	133054	0	0	83088	80441	0	0	2093	1995
Cache L2 Dados	0	0	188265	187843	0	0	25102	22336	0	0	124	139
Ciclos L1	10217246748	10025995297	765622718	780156595	1952997930	2344615792	163873561	164269124	69441678	130915918	18283261	18774089

Tabela 2: Penalidades causadas pela cache.

Como resultado obtivemos que a melhor configuração de cache para os programas é a Cache 3.

3.2 Modelos

Com os três programas selecionados Dijkstra, JPEG e SHA obtivemos os seguintes resultados para os modelos de 1 a 4 e considerando um processador que faz $50.000.000 ciclos/segundos$, temos:

	Modelo 1			Modelo 2			Modelo 3			Modelo 4		
	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA
Nº de Instruções	285280151	29474823	13036287	285280151	29474823	13036287	285280151	29474823	13036287	285280151	29474823	13036287
Pipeline	4	4	4	4	4	4	4	4	4	4	4	4
LUH	0	49232	0	0	49232	0	0	49232	0	0	49232	0
Forwards	289129896	20216111	7415403	0	0	0	0	0	0	0	0	0
BSP	74772918	6473388	2389284	74772918	6473388	2389284	0	0	0	0	0	0
BLSCP	0	0	0	0	0	0	990717	772785	104520	0	0	0
BLTAP	0	0	0	0	0	0	0	0	0	882597	750516	119184
Instruções em Paralelo	0	0	0	0	0	0	0	0	0	0	0	0
Penalidade Cache	765622718	163873561	18283261	765622718	163873561	18283261	765622718	163873561	18283261	765622718	163873561	18283261
Total de Ciclos	649182969	56213558	22840978	360053073	35997447	15425575	286270872	30296844	13140811	286162752	30274575	13155475
Tempo(s)	28,296	4,402	0,822	22,514	3,997	0,674	21,038	3,883	0,628	21,036	3,883	0,629

Tabela 3: Modelos com pipeline de 5 estágios.

Como podemos ver os modelos que apresentam melhor resultado são aqueles que possuem os branch predictors, o modelo 4 possui uma leve vantagem devido à capacidade de detectar padrões na execução do branch. Porém a maior expressividade de resultados se deve a implementação do forward.

Agora comparando uma implementação do Modelo 4 superescalar temos:

	Modelo 4			Modelo 5		
	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA
Nº de Instruções	285280151	29474823	13036287	285280151	29474823	13036287
Pipeline	4	4	4	4	4	4
LUH	0	49232	0	0	49232	0
Forwards	0	0	0	0	0	0
BSP	0	0	0	0	0	0
BLSCP	0	0	0	0	0	0
BLTAP	882597	750516	119184	882597	750516	119184
Instruções em Paralelo	0	0	0	-91113214	-6179927	-2114996
Penalidade Cache	765622718	163873561	18283261	765622718	163873561	18283261
Total de Ciclos	1051785470	194148136	31438736	960672256	187968209	29323740
Tempo(s)	21,036	3,883	0,629	19,213	3,759	0,586

Tabela 4: Comparação com superescalar.

Percebemos uma diminuição clara no tempo de execução, devido a instruções de load/store que podem ser feitas em paralelo com outros tipos de intruções.

No final vamos verificar as diferenças de desempenho do melhor modelo de pipeline de 5 estágios com dois modelos de pipeline de 7 estágios:

	Modelo 5			Modelo 6			Modelo 7		
	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA	Dijkstra	JPEG	SHA
Nº de Instruções	285280151	29474823	13036287	285280151	29474823	13036287	285280151	29474823	13036287
Pipeline	4	4	4	6	6	6	6	6	6
LUH	0	49232	0	41406040	3237863	393486	41406040	3237863	393486
Forwards	0	0	0	0	0	0	0	0	0
BSP	0	0	0	0	0	0	0	0	0
BLSCP	0	0	0	1320956	1030380	139360	0	0	0
BLTAP	882597	750516	119184	0	0	0	1176796	1000688	158912
Instruções em Paralelo	-91113214	-6179927	-2114996	0	0	0	0	0	0
Penalidade Cache	765622718	163873561	18283261	765622718	163873561	18283261	765622718	163873561	18283261
Total de Ciclos	960672256	187968209	29323740	1093629871	197616633	31852400	1093485711	197586941	31871952
Tempo(s)	19,213	3,759	0,586	15,623	2,823	0,455	15,621	2,823	0,455

Tabela 5: Comparação com pipeline de 7 estágios, há diferenças no fator utilizado no cálculo do tempo.

A primeira diferença aparente é o aumento no número de *load-use hazards* que é causado pelo aumento de um estágio de pipeline entre o EX e o MEM portanto adicionando uma bolha a mais do que o processador de 5 estágios e um aumento na penalidade de um *branch prediction miss* pois há mais um estágio antes do IF. No caso dos modelos de 5 estágios há valores 0 no LUH devido a otimizações do compilador.

Há também uma alteração no fator de divisão pois com um pipeline mais granular podemos aumenta o número de ciclos por segundo, utilizando um fator de proporção baseado no pipeline temos $50000000 * 7/5 = 70000000 ciclos/s$.

4 Conclusão

Analisando as informações obtidas pelas tabelas, percebemos que há 3 grandes influências no desempenho do processador, a primeira é devido à data hazards como o forward que pela Tabela 3 é possível notar um grande aumento no desempenho só pela implementação desse mecanismo simples.

A segunda grande diferença é dada pela implementação de um branch predictor o que também pela Tabela 3 é possível perceber um aumento de quase 10% de desempenho.

A terceira são recursos mais sofisticados como superescalarizar um pipeline, ou quebrar partes de um estágio em uma ou mais partes para poder aumentar o número de ciclos por segundo. Porém foi possível perceber durante a implementação das estruturas no simulador que esses recursos podem ao ínvez de aumentar o desempenho diminuí-lo, por exemplo, aqui foi utilizada a hipótese de que é possível aumentar o fator de ciclos por segundo proporcionamente ao tamanho do pipeline, porém se isso não foi verdade na prática a implementação dessa característica pode vir a trazer prejuízos.