Otimização de Desempenho com Baixo Custo em Processadores Softcores via Reuso de Funções

Pedro Henrique Exenberger Becker
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Orientado por Prof. Dr. Antonio Carlos S. Beck Filho
phebecker@inf.ufrgs.br

Introdução

- Este trabalho considera os benefícios de processadores implementados em FPGA como:
  - Definição de arquiteturas específicas;
  - Inclusão de aceleradores de hardware;
  - *Time-to-market*;
  - Descrição de hardware mantém o projeto em par com a tecnologia atual;
- Observando os desafios de implementação em sistemas modernos:
  - Espera-se desempenho crescente nos dispositivos computacionais;
  - Processadores com múltiplos núcleos e múltiplos hardwares dedicados e, portanto, um projeto complexo;
  - Sistemas complexos podem exceder a capacidade dos FPGAs, devido a falta de LUTs ou registradores;
- Como solução, mapeiam-se as tarefas que seriam feitas em hardware, para o domínio do software, ao custo de performance.
- Pode-se acelerar a execução do software evitando reexecução de códigos iguais
  - Com Reuso de Funções, por exemplo;
    - Salvam-se os valores de funções computadas em uma tabela, para consulta futura caso a função e seus valores de entrada se repitam;

Proposta

- Alterar um processador de modo que este contenha uma unidade de reúso de funções.
- Aliviar o custo de performance com um módulo de baixo custo.
- Fazendo uso de BRAMs, que são recursos menos utilizados em FPGAs.

A tabela 1, abaixo, mostra que a utilização de BRAM cresce mais sutilmente que dos demais componentes em FPGAs, conforme a complexidade do design aumenta:

Tabela 1: *Utilização de Recursos de FPGAs para Diferentes Designs*

Design	% Slice LUT	% Slice Register	% BRAM	Complexidade
OpenRisc1200	5%	2%	7%	Baixa
Leon 3	27%	16%	15%	Média
OpenSparc T1	88%	56%	40%	Alta

Implementação do Sistema

Implementamos um sistema de reúso de funções no processador VLIW ρ-vex, descrito em linguagem de hardware VHDL. A seguir, na Figura 1, apresentamos o processador com uma Unidade de Reúso (RU), e explicamos o seu funcionamento. Observe os números indicados nas porções coloridas, que indicam fases de funcionamento do sistema.

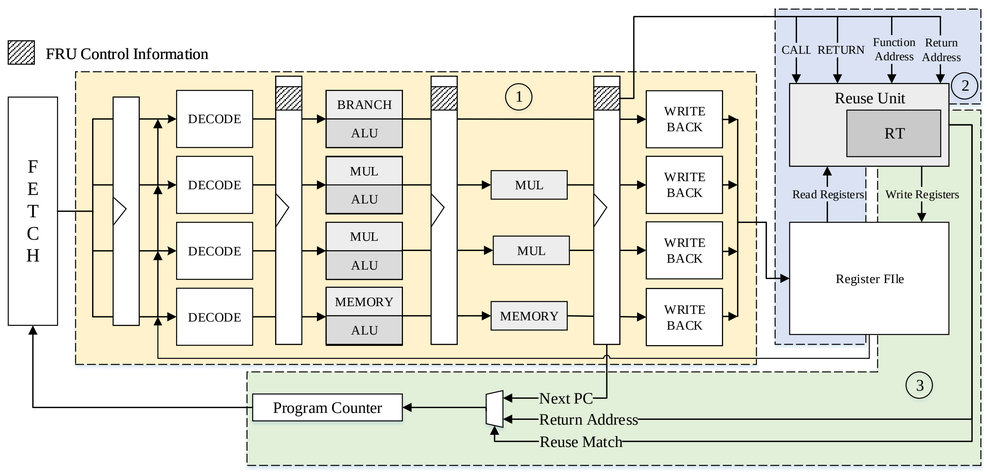

Figura 1: *Organização do sistema ρ-vex + RU*

- Fase 1: Na primeira fase algumas informações relevantes para o funcionamento do esquema são coletadas:
  - Se existe uma chamada de função (instrução *CALL*) ou retorno de função (instrução *RETURN*);
  - Se houver um *CALL*, captura-se qual é a função que se está chamando e qual o endereço de retorno;
- Fase 2: Na segunda fase a RU Consulta pode tomar dois caminhos
  - Se a função chamada e seus parâmetros nunca executaram previamente, espera a execução acontecer
    - Ao final da execução os valores de retorno da função (resultados) serão salvos em uma Tabela de Reúso
  - Se a função chamada e seus parâmetros já foram executados previamente, então o resultado já está na Tabela de Reúso
  - A fase 2, portanto, é capaz de verificar se os dados da função atual constam na tabela.
- Fase 3: Quando se descobre que uma função e seus parâmetros já foram executados previamente, a RU pode:
  - Atualizar o contexto do processador. Para isso, escreve-se no banco de registradores os valores de resultado da função, trocando a reexecução da mesma pela busca na Tabela de Reúso.
  - Ou, se não constam na Tabela de Reúso, salvar os valores atuais para futuro reúso (quando a função termina de executar).

Resultados

Como caso de estudo, imaginamos o cenário que um dado acelerador não cabe no design e precisamos utilizar a RU para acelerar o software. Os resultados a seguir, consideram que se substituiu uma unidade de ponto flutuante (FPU), por uma biblioteca de software que resolve operações de ponto flutuante em uma série de instruções inteiras (dentro de funções dessa biblioteca). A partir disso, a RU reutiliza as funções *add, sub, mul, div* dessa biblioteca, em diferentes benchmarks.

- Reusabilidade por Função
  - A figura 2 apresenta quanto cada uma das funções pode ser reusada, para diferentes tamanhos de Tabela de Reúso.
  - Tabelas maiores guardam mais valores de execuções passadas e, por isso, tendem a prover maior reusabilidade.

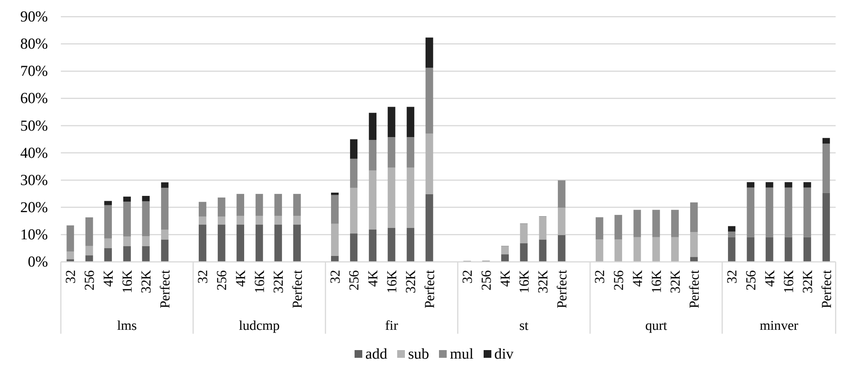

Figura 2: *Distribuição de Reusabilidade por Função*

- Aumento de Performance
  - A figura 3 apresenta o aumento de desempenho em 6 aplicações que utilizam as operações em ponto flutuante.
  - Em média geométrica, uma pequena tabela, com 32 linhas já consegue aumentar em 12% o desempenho do programa sobre o uso da biblioteca. Para tabelas maiores, os ganhos podem chegar a até 27%.

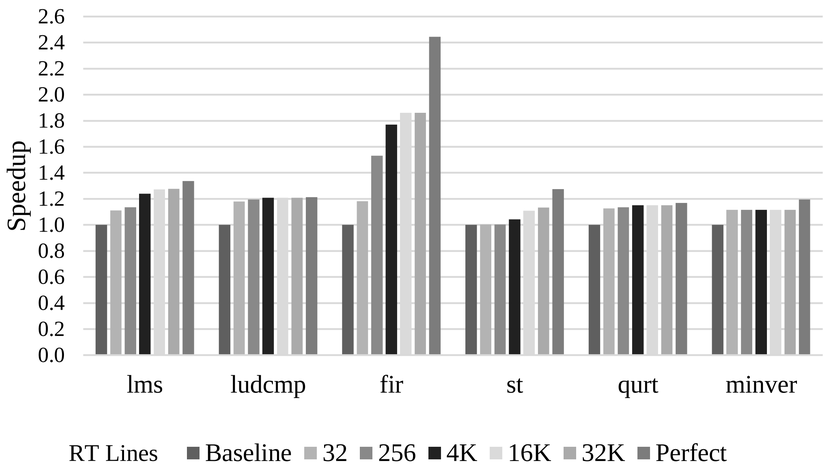

Figura 3: *Aumento de Performance para Diferentes Tamanhos de Tabela de Reúso*

- Uso de Recursos de FPGA
  - A tabela 2 abaixo apresenta o uso de recursos de FPGAs de diferentes modelos, onde vê-se que a RU é muito menos intrusiva do que um hardware complexo como uma FPU.
  - Por fim, observa-se que as BRAMs não são utilizadas nos cenários sem RU, reforçando o fato de que podemos usar nossa técnica para melhor utilizar o chip de FPGA.

Tabela 2: *Uso de Recursos em Diferentes Designs: ρ-vex original, ρ-vex com a unidade de reúso e ρ-vex com uma unidade de ponto flutuante*

Series	Model	Design	Used Slice Registers	% Used Slice Registers	Used Slice LUTs	% Used Slice LUTs	Used BRAM	% Used BRAM
Virtex 7	xc7vx690t	ρ-Vex	3,015	0%	14,675	3%	0	0%
		ρ-Vex + RU (32K lines)	3,494	0%	15,176	4%	226	15%
		ρ-Vex + FPU	7,275	1%	19,926	5%	0	0%
Virtex 5	xc5vsx95t	ρ-Vex	3,012	5%	15,200	26%	0	0%
		ρ-Vex + RU (32K lines)	3,516	6%	15,717	27%	226	93%
		ρ-Vex + FPU	7,061	12%	23,349	40%	0	0%
	xc5vsx50t	ρ-Vex	3,012	9%	15,200	47%	0	0%
		ρ-Vex + RU (16K lines)	3,516	11%	15,717	48%	113	86%
		ρ-Vex + FPU	7,061	22%	23,349	72%	0	0%
Virtex 4	xc4vsx55	ρ-Vex	3,008	6%	23,986	49%	0	0%
		ρ-Vex + RU (16K lines)	3,511	7%	24,820	50%	226	71%
		ρ-Vex + FPU	7,403	15%	35,888	73%	0	0%
	xc4vlx40	ρ-Vex	3,008	8%	23,986	65%	0	0%
		ρ-Vex + RU (4K lines)	3,511	10%	24,820	67%	57	59%
		ρ-Vex + FPU	7,403	20%	35,888	97%	0	0%

Conclusões

Apresentamos um mecanismo de reúso de funções acoplado a um processador real que é capaz de otimizar bibliotecas de software quando a adição de um hardware específico é custoso, em termos de recursos, em uma FPGA. Mostramos resultados de performance para um conjunto de benchmarks, bem como o pequeno impacto que o mecanismo implica para diferentes dispositivos.