NINA

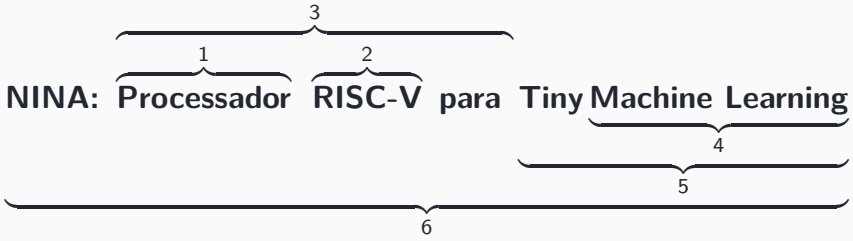
Processador RISC-V para Tiny Machine Learning

Guilherme Moreno Silva
Orientador: Prof. Dr. Alfredo Goldman

Instituto de Matemática e Estatística
Universidade de São Paulo

IME USP

Em um Mundo Ideal...



...Mas a Vida É Curta

Seção 2.1 Seção 2.2

NINA: Processador RISC-V para Tiny Machine Learning

Seção 3

Overview

Overview

Introdução

- **A miniaturização de transistores é essencial para a computação**
  - Transistores menores consomem menos energia
  - Com maior quantidade, podemos construir processadores mais complexos
  - Permitem construir dispositivos ainda mais compactos
- **Houve desaceleração no ritmo dessa miniaturização**
- **Coloca novas dificuldades para arquitetos de computadores**
  - Como melhorar o desempenho?

Hardware-Software Co-design

Introdução

Hardware-Software Co-design

Abordagem de projetar *hardware* especificamente para determinada *software*

Overview

Processadores

Métricas de Performance

Instruções por Segundos

$$\text{IPS} = \frac{Frequência}{Ciclos\ por\ Instrução}$$

Performance

Queremos aumentar a frequência de operação e diminuir a quantidade de ciclos por instrução (CPI)

Pipelining

Pipelining de instruções é uma técnica que consiste em dividir a execução de instruções em múltiplas etapas, denominadas estágios. Isso possibilita que todas as partes do processador estejam ocupadas simultaneamente.

Pipelining

Utilizar *pipelining* permite aumentar a frequência de operação do processador, pois a frequência é determinada pelo período do estágio mais lento, enquanto cria a ilusão de executarmos uma instrução por ciclo.

Pipelining

O que nos impede de aumentar o número de estágios?

Hazards

- **Instruções podem possuir dependências**
  - Dependências que limitam o *pipelining* são chamadas *hazards*
- **Existem dois de *hazards***
  - De dados (*Data*)
  - De controle (*Control*)

Hazards

- **Data Hazards**
  - Instrução depende do resultado de outra instrução não terminada
- **Control Hazards**
  - Escolha da próxima instrução depende do resultado de outra instrução não terminada

Resolvendo Hazards

- **Data Hazards**
  - *stalling* ou *forwarding*

Resolvendo Hazards

- **Stalling**
  - Atrasar a execução das instruções até que o *hazard* tenha se resolvido

Stalling

- **Stalling**
  - Atrasar a execução das instruções até que o *hazard* tenha se resolvido
  - Impacta diretamente o CPI

Forwarding

- **Stalling**
  - Atrasar a execução das instruções até que o *hazard* tenha se resolvido
  - Impacta diretamente o CPI
- **Forwarding**
  - "Encaminhar" o resultado de um estágio para outro

Forwarding

- **Stalling**
  - Atrasar a execução das instruções até que o *hazard* tenha se resolvido
  - Impacta diretamente o CPI
- **Forwarding**
  - "Encaminhar" o resultado de um estágio para outro
  - Resolve a dependência antes do término de execução
  - Não impacta o CPI
  - Aumenta a complexidade do processador

Resolvendo Hazards

- **Data Hazards**
  - *stalling* ou *forwarding*
- **Control Hazards**
  - *stalling*

Control Hazards

- **Ocorrem quando temos desvios condicionais**
  - `if (x == y) x++;`

Control Hazards

- **Ocorrem quando temos desvios condicionais**
  - `if (x == y) x++;`
- **Desvios condicionais são não determinísticos**
  - Devemos executar `x++`?
- **Não podemos determiná-los *a priori***

Control Hazards

- **Ocorrem quando temos desvios condicionais**
  - `if (x == y) x++;`
- **Desvios condicionais são não determinísticos**
  - Devemos executar `x++`?
- **Não podemos determiná-los *a priori***
- **Podemos prever seu resultado?**

Branch Prediction

Branch Predictor

O *branch predictor* é um componente de processadores responsável por tentar "adivinhar" se um desvios condicional será executado ou não.

Resolvendo Hazards

- **Stalling**
  - Atrasar a execução das instruções até que o *hazard* tenha se resolvido
  - Impacta diretamente o CPI
- **Forwarding**
  - "Encaminhar" o resultado de um estágio para outro
  - Resolve a dependência antes do término de execução
  - Não impacta o CPI
  - Aumenta a complexidade do processador
- **Branch Prediction**
  - Conjectura o resultado de um desvio condicional
  - Se correto, não impacta o CPI
  - Caso contrário, impacta o CPI (necessário desfazer instruções)
  - Aumenta (muito) a complexidade do processador

Tiny Machine Learning

(Tiny) Machine Learning

Tiny Machine Learning

É um subcampo de aprendizado de máquina com foco em habilitar *deep learning* em dispositivos com recursos limitados

Tiny Machine Learning

- **Por que *Tiny Machine Learning*?**
  - Redes neurais *feedforward* são um produto escalar
  - Podemos implementar aninhando laços e condicionais
  - Laços causam *control hazards* e estressam *branch predictors*

Overview

Metodologia

- **Pesquisa experimental e quantitativa**
- **Implementa-se um *soft-core* em três variantes, controlando-se as variáveis**
- **Implementa-se o *software* usado como *benchmark***
- **Objetivos**
  - Estudar o efeito e aplicabilidade de *hardware-software co-design* no projeto de processadores
  - Comparar o desempenho com abordagens tradicionais de otimização

Metodologia

Software:

- **Rede neural *feedforward***
  - Camadas: Dense
  - Ativações: ReLu e Softmax
- **Implementação sem *frameworks***
  - Camadas implementadas como multiplicação de matrizes

Metodologia

Hardware:

- **Processador A**
  - Baseline
  - Data hazards: *forwarding*
  - Control hazards: *stalls*
- **Processador B**
  - Data hazards: *forwarding* aprimorado
  - Control hazards: *branch prediction*
- **NINA**

NINA

NINA **I**s **N**ot **A**RM

NINA

NINA **I**s **N**ot **A**RM

NINA

- **Processador RISC-V**
  - Implementa o *instruction set* RV32I
- **Três estágios de *pipeline***
- **Arquitetura Harvard**
  - 16KB de IMEM
  - 16KB de DMEM

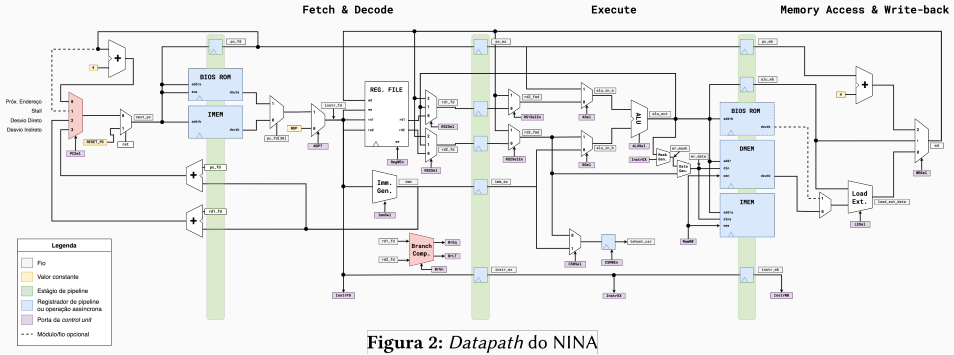


Figura 2: *Datapath* do NINA

Overview

Resultados

	A	B	NINA
CPI	1.18	1.1	1.0
Freq. Máx	45	53	62.5
MIPS	38,13	48,18	62.5

Tabela 1: *Comparativo de desempenho. Sintetizados em FPGA Gowin GW2A-18*

Resultados

	A	B	NINA
CPI	1.18	1.1	1.0
Freq. Máx	45	53	62.5
MIPS	38,13	48,18	62.5
Uplift	1,63x	1,29x	1x

Tabela 2: *Comparativo de desempenho com ganho relativo*

Overview

Conclusão

- **Abordagem de *co-design* possibilitou ganhos expressivos de desempenho em relação à abordagem tradicional**
- **O tempo de projeto e implementação entre as diferentes abordagem foi semelhante**

Overview

Referências i

- Hennessy, John L. **and** David A. Patterson (2011). *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. Morgan Kaufmann Publishers Inc.
- Lin, Ji **andothers** (2023). **?**Tiny Machine Learning: Progress and Futures [Feature]**?** *IEEE Circuits and Systems Magazine* 23.3, **pages** 8–34. DOI: 10.1109/MCAS.2023.3302182.
- Patterson, David A. **and** John L. Hennessy (2017). *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. 1st. Morgan Kaufmann Publishers Inc.
- Theis, Thomas N. **and** H.-S. Philip Wong (2017). **?**The End of Moore's Law: A New Beginning for Information Technology**?** *Computing in Science & Engineering* 19.2, **pages** 41–50. DOI: 10.1109/MCSE.2017.29.

Obrigado!