Instituto Superior de Engenharia de Lisboa

Mestrado em Engenharia Informática e Computadores

Semestre de Verão 2023/2024

Projeto 1

Arquiteturas Avançadas de Computação

Trabalho realizado por:

Pedro Carvalho nº47113 G02

Nuno Bartolomeu nº47233 G02

12/04/2024


ISEL
INSTITUTO SUPERIOR DE
ENGENHARIA DE LISBOA

ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Índice

Objetivos

Este relatório tem como objetivo relatar a aprendizagem feita pelos alunos sobre o primeiro trabalho de AAC.

Introdução

Neste trabalho foram realizados testes de design para tentar compreender a relação entre performance (ciclos de clock) e área (número de recursos). Estes testes foram feitos em cima de uma aplicação que com uma imagem e um kernel cria um output, todos estes elementos são representados por arrays bidimensionais.

Com o auxílio de operações HLS é possível otimizar a aplicação. As operações utilizadas neste projeto foram o Pipeline, Loop Unroll e Array Partition.

- **Pipeline**
  Sem auxílio ao pipeline, o código necessita utilizar um ciclo para reaver os dados da memoria e outro para realizar operações lógicas. Com o pipeline este processo pode ser realizado no mesmo ciclo o que pode reduzir o número de ciclos pela metade.

- **Loop unroll**
  O objetivo do loop unroll é semelhante ao pipeline, fazer mais ações por cada clock. Com o loop unroll é possível diminuir o número de iterações de cada loop até que o loop seja realizado por completo em apenas uma iteração, deixando o código como se fosse serializado.

- **Array Partition**
  Com o array partition é possível dividir um array em várias memorias, aumentando o número de portos. Isto pode ajudar o pipeline e o unroll que são limitados pelos acessos a memoria.
  - **Block**

  A partição em bloco divide o array em F outros, sendo F o valor do fator. Estes arrays vão ser preenchidos sequencialmente, ou seja, o primeiro vetor, o V0, contém os indexes de 0 a N/F-1.

  - **Cyclic**

  A partição em ciclo preenche F arrays de forma alternada, ou seja, V0 contém o index 0, F+1, F+2 … N-F.

- **Complete**

A partição completa divide cada index no seu próprio vetor. Esta partição pode ser atingida usando o bloco e a cíclica se F for igual a N. V0 contem apenas o index 0.

Testes

Para os testes foram utilizadas diversas combinações de ajustes entre:

- Tamanhos de imagem
- Pipeline on ou off
- Fatores de unroll diferentes
- Fatores de partição diferentes

Sem operações HLS

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Interval	Pipelined	BRAM	DSP	FF	LUT	URAM
conv2D0		-0.27	39	390.000	40	no	0	5	502	920	0
conv2D0_Pipeline_readImg		-	18	180.000	18	no	0	0	140	65	0
conv2D0_Pipeline_readweights		-	11	110.000	11	no	0	0	83	64	0
conv2D0_Pipeline_loop_orow_loop_ocol	Timing Violation	-0.27	10	100.000	10	no	0	5	264	681	0
conv2D0_Pipeline_writeImg		-	6	60.000	6	no	0	0	5	71	0

Sem as operações é possível identificar uma “Violação de Tempo”.
Resultados:

- Ciclos - 39
- DSP - 5
- FF - 502
- LUT - 920

Pipeline off in all loops

Modules & Loops	Issue Type	Slack	Latency(cycles)	Interval	Trip Count	Pipelined	DSP	FF	LUT
conv2D0		-	274	275	-	no	1	123	403
readImg		-	32	-	16	no	-	-	-
readweights		-	18	-	9	no	-	-	-
loop_orow		-	216	-	2	no	-	-	-
loop_ocol		-	106	-	2	no	-	-	-
loop_k1		-	51	-	3	no	-	-	-
loop_k2		-	15	-	3	no	-	-	-
writeImg		-	4	-	4	no	-	-	-

Ao desativar o pipeline foi possível identificar um aumento significativo na latência, mas isto é complementado com uma diminuição de recursos. Isto era esperado devido ao fato de que o pipeline permite realizar várias operações no mesmo ciclo, o que reduz o número total de ciclos, mas gasta mais recursos.

Resultados:

- Ciclos – 274 (7x mais elevado)
- DSP – 1 (-5x)
- FF – 123 (-4x)
- LUT – 403 (-2x)

Pipeline off and unroll factor of 2

Modules & Loops	Issue Type	Slack	Latency(cycles)	Interval	Trip Count	Pipelined	DSP	FF	LUT
conv2D0		-	130	131	-	no	16	275	1481
readImg		-	16	-	8	no	-	-	-
readweights		-	13	-	4	no	-	-	-
loop_k1		-	22	-	2	no	-	-	-
loop_k1		-	22	-	2	no	-	-	-
loop_k1		-	22	-	2	no	-	-	-
loop_k1		-	22	-	2	no	-	-	-
writeImg		-	2	-	2	no	-	-	-

Esta experiência mostrou algo interessante, em comparação com a última, houve uma redução de ciclos, mas o número de recursos ficou extremamente mais elevado.

Resultados:

- Ciclos – 130 (-2x)
- DSP – 16 (16x)
- FF – 275 (2x)
- LUT – 1481 (4x)

Perfect unroll

Modules & Loops	Issue Type	Slack	Latency(cycles)	Iteration Latency	Interval	Trip Count	Pipelined	DSP	FF	LUT
conv2D0		-	12	-	13	-	no	20	301	989

O “perfect unroll” quer dizer que o fator é igual ao número de iterações de cada loop, isto faz com que os loops deixem de ser loops e passem a ser uma sequência corrida de operações. Este facto faz com que o pipeline seja inútil e não houve qualquer diferença deixa-lo on ou off.
A comparação foi feita com o teste inicial, devido a serem mais semelhantes e os resultados foram esperados a exceção do número de flip flops que diminui devido a já não estar a ser usado o pipeline.

Resultados:

- Ciclos - 12 (-3x)
- DSP - 20 (4x)
- FF - 301 (-1.5x)
- LUT – 989(semelhante)

Partition type complete e Perfect unroll

Modules & Loops	Issue Type	Slack	Latency(cycles)	Iteration Latency	Interval	Trip Count	Pipelined	DSP	FF	LUT
conv2D0		-	5	-	6	-	no	20	38	817

Esta foi a melhor solução que conseguimos alcançar a nível de tempo e recursos. Em comparação com o teste sem pipeline (o que utiliza menos recursos), houve uma redução de tempo extremamente significativa ao custo de duplicar o número de LUT.
A comparação de resultados em baixo é relativa ao perfect unroll.

Resultados:

- Ciclos - 5 (-2x)
- DSP - 20 (igual)
- FF - 38 (-8x)
- LUT – 817(-1.2x)

Partition type complete e pipeline off

Modules & Loops	Issue Type	Slack	Latency(cycles)	Iteration Latency	Interval	Trip Count	Pipelined	DSP	FF	LUT
conv2D0		-	233	-	234	-	no	1	304	544
readImg		-	32	2	-	16	no	-	-	-
readweights		-	18	2	-	9	no	-	-	-
loop_orow		-	180	90	-	2	no	-	-	-
loop_ocol		-	88	44	-	2	no	-	-	-
loop_k1		-	42	14	-	3	no	-	-	-
loop_k2		-	12	4	-	3	no	-	-	-

Por curiosidade tentamos remover o unroll e o pipeline e obtemos resultados comparáveis com os valores sem partição que estão comparados em baixo. Notável apenas o aumento nos flip flops com uma ligeira diminuição no número de ciclos. Não mostramos os valores com o pipeline ativo uma vez que são basicamente iguais aos valores do primeiro teste realizado tal como esperado.

Resultados:

- Ciclos – 233(-1.2x)
- DSP – 1 (igual)
- FF – 304(2.5x)
- LUT – 544(1.3x)

Conclusões

Nós notamos que a melhor forma de não usar recursos é não fazer nenhum tipo de otimizações e a melhor forma de aumentar a performance é ativar todas as otimizações no máximo.
Reparamos que o pipeline não tem efeito se o unroll for completo, mas em qualquer outra situação diminui imenso o número de ciclos.

Também concluímos que as partições do tipo block ou cyclic com fatores iguais ao tamanho do vetor é o mesmo que fazer complete.
Em testes que fizemos com o block e o cyclic com fatores diferentes do tamanho do vetor, não encontramos nenhuma diferença de performance ou área entre os dois.
Quando aumentamos o tamanho o número de ciclos não aumenta com otimizações máximas mas gastamos mais área. Se não usarmos as otimizações todas então vemos um aumento tanto da performance como o número de recursos.
Também encontramos uma corelação entre flip flops e o pipeline que revela parte do seu funcionamento.
Por último, os unrolls reduzem o número de ciclos e aumentam os recursos como já era esperado.