**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

# 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

Índice

[Objetivos 1](#_Toc163813818)

[Introdução 1](#_Toc163813819)

[Testes 2](#_Toc163813820)

[Sem operações HLS 2](#_Toc163813821)

[Pipeline off in all loops 2](#_Toc163813822)

[Pipeline off and unroll factor of 2 3](#_Toc163813823)

[Perfect unroll 3](#_Toc163813824)

[Partition type complete e Perfect unroll 4](#_Toc163813825)

[Partition type complete e pipeline off 4](#_Toc163813826)

[Conclusões 4](#_Toc163813827)

# 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

A screenshot of a computer

Description automatically generated

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

Resultados:

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

## Pipeline off in all loops

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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

A grey rectangular sign with white text

Description automatically generated

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



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

A screenshot of a computer

Description automatically generated

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.