Implementing CNN using Zynq device - Hardware Software Co-Design- Final Project

João Lopes e Gonçalo Carvalho

June 2019

1 Introduction

Ao contrário do que foi feito na primeira parte do trabalho onde optimizamos apenas a convolução, pretende-se agora, que além disso se faça a classificação da imagem, utilizando os pesos fornecidos pelo professor. Posto isto a convolução de matrizes mantem-se, teremos a introdução de uma rede neural com diversas layers as quais precisam de ser optimizadas de modo a acelarar o processo de classificação das imagens.Para esta segunda parte do projecto diferentes soluções são propostas, usando diferentes IPs, Direct Memory Access (DMA) e ambos ARM Cores como forma de acelarar este processo.

2 Rede Neural

O objectivo é acelarar uma rede neural convolucional simples que recebe uma imagem de 28x28 floating point onde na ultima layer a imagem será classificada com um número de 0 a 9 consoante a a previsão da rede neural. A rede já está treinada, e os pesos deste treino prévio vão ser usados no processo de classificação. Este processo de classificação está dividido em 4 partes fundamentais:

- Forward Convolutional Network
- Pooling Layer
- Fully Connected Layer
- Softmax Layer

Cada layer tem um papel distinto na classificação da imagem. Na forward convolution layer a imagem e convolvida com 22 kernels diferentes sendo obtido um total de 22 imagens das quais diferentes features foram extraidas. As imagens resultantes terão o tamanho de 24x24.
Na pooling layer as 22 imagens anteriores serão reduzidas a uma única imagem de tamanho 24x24.

Na fully connected layer os pixeis resultantes são multiplicados como um vetor único pelos vetores de pesos, resultando em 10 floating-point outputs. Na ultima layer (Softmax) estes 10 resultados são escalados exponencialmente e cujo resultado é a probabilidade de cada digito. O digito que apresentar a maior probabilidade será o digito predicto.

3 Software Only

O primeiro passo para acelarar a classificação da imagem é correr o código como SW only em primeira instância. O código foi providenciado pelo professor e realiza os pontos atrás mencionados. Através da tabla seguinte :

Layer	Convolutional	Pooling	Fully-Connected	Softmax	Tempo Total
Time (μs)	32839	1300	2998	5	37142
Maior fracção de execução (%)	88.41	3.50	8.077	0.013	

Tabela 1: Tempos de execução de cada layer e respectiva fracção

Fazendo uso da tabela acima indicada e bem como a lei de Amdhal é possivel "prever"o speed-up que conseguimos obter em hardware. A lei de Amdhal é dada pela seguinte equação:

$$S_{hwsw} = \frac{1}{F + \frac{1-F}{S_{cs}}} \tag{1}$$

Pela tabela 5 verifica-se que a maior fracção é a correspondente à convolutional layer. A prioridade para o speed-up será nesta layer. Pela equação 1 vem que :

$$Speed_{up} \leq \frac{1}{(1 - 0.8841) + \frac{1-(1-0.8841)}{S_{cs}}} = 8.62 \tag{2}$$

Esta convolução como referido é calculada como uma multiplicação de matrizes usando floating-points. Como estamos a lidar com floating points é possivel acelarar este processo com Ips especiais de multiplicação de matrizes de floating points. Todas as outras layers podem ser acelaradas. Onde a fully connected layer pode também ser acelarada tendo em conta o mesmo processo referido, dado que se trata também esta de uma multiplicação de matrizes. Para a fully-connected layer o speed-up tem o valor de:

$$Speed_{up} \leq \frac{1}{(1 - 0.08077) + \frac{1-(1-0.08077)}{S_{cs}}} = 1.088 \tag{3}$$

As outras layers são também possiveis de optimizar com um speed-up máximo de:

- Pooling Layer

$$Speed_{up} \leq \frac{1}{(1 - 0.035) + \frac{1-(1-0.035)}{S_{cs}}} = 1.036 \tag{4}$$

- Softmax

$$Speed_{up} \leq \frac{1}{(1-0.00013) + \frac{1-(1-0.00013)}{S_{cs}}} = 1.00 \tag{5}$$

4 Step by step Optimization

4.1 DMA e IP

A primeira otimização foi incluir um DMA no design, juntamente com o IP que faz a multiplicação de matrizes de floating point. O IP está "condicionado"por dois conversores. Um float2fixed (ligado à entrada) e um fixed2float (ligado à saida). O IP faz a multiplicação recorrendo a multiplicações e somas. A adição de números com virgula fixa com o mesmo factor de escala é directa, contudo se estes apresentarem fatores de escala diferente, os pontos binários precisam de estar alinhados para a soma produzir o resultado correto. Não obstante a aritmética com numeros de virgula fixa é mais rápida, dado que, é implementada usando operações de inteiros regulares. A utilização do conversor float2fixed antes do IP vai facilitar a rapidez de cáculo da multiplicação no ip originando um melhoramento geral na velocidade calssificação de uma imagem.

No ponto anterior foram calculados os speed-up que são possivel obter teoricamente. Apesar de alguns valores serem bastante promissores é muito complicado atingir os valores teóricos máximos. A multiplicação da matriz ainda leva um tempo significativo para ser fefectuada, e ainda há que ter em atenção ao overhead associado à comunicação de/para o hardware. Este overhead pode ser minimizado usando como recurso uma DMA para ler e escrever a informação diretamente de/para a memória principal sem que tenha de passar pelo processador. O AXI DMA comunica com o processador através da AXI-Lite Interface (GP0), com o processador como master, para transmissão dos sinais de controlo. Algumas definições foram desativadas por simplificação, sendo usado o simple pooling mode. O DMA conecta diretamente ao PS através do AXI-HP port, que interage diretamente com o mapeamento de memória AXI4 e a data é transmitida de/para o IP através do AXI4-Stream. Após estas implementações é utilizado o seguinte design:

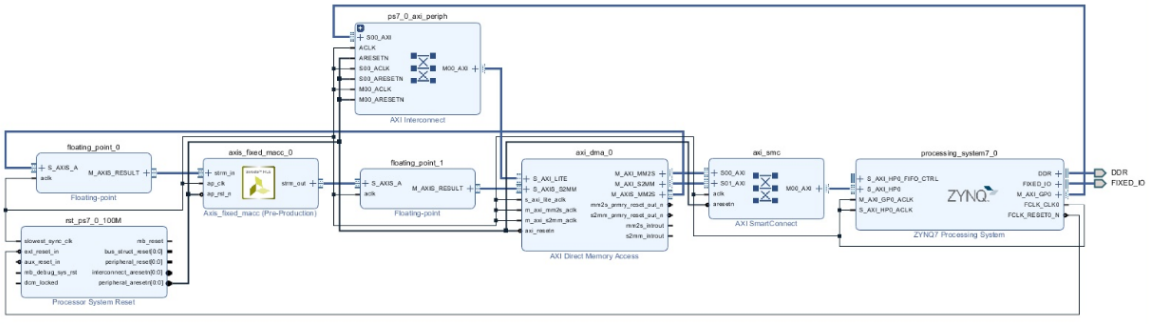

Figura 1: Design for Software-Hardware Implementation

Os resultados obtidos para o respectivo design foram os seguintes:

Layer	Convolutional	Pooling	Fully-Connected	Softmax	Tempo Total
Time (μs)	6612	1342	2998	5	10957

Tabela 2: Tempos de execução de cada layer e respectiva fracção

Como previsto, a utilização do IP acompanhado dos conversores fixed2float e float2fixed, e a DMA levou a uma diminuição significativa do tempo que o programa demora a correr. Olhando para o tempo em cada layer verifica-se que a convolutional layer teve um speed-up de 4.96. Enquanto que as restantes layers mantiveram aproximadamente o seu valor. Esta inalteração nas restantes layers pode ser explicado pelo **não uso** da DMA e do IP para realizar as operações que lhes dizem respeito. A forward convolutional layer assenta na preparação da matrix A (efectuada em software) e na multiplicação dessa matriz pelos 22 kernels (feito com recurso ao IP e usando o DMA para acelarar as tranferências de dados). Assim as restantes layers continuam a ser feitas unicamente em software enquanto que a convolutional está repartida em:

- Software (prepare_matrixA)
- Hardware (dmammBT)

Explicando assim o speed-up verificado.

4.2 Mudança no IP (Bias)

A segunda otimização foi realizada no IP. Dentro da Convolutional Layer após a convolução da imagem de entrada com cada um dos 22 Kernels é somado ao resultado, matriz C, um valor bias em cada uma das suas entradas. Esta tarefa era originalmente realizada por uma função chamada add_bias em que através de dois ciclos for se somava a cada entrada da matriz C o valor correspondente guardado numa matriz previamente guardada em memória. De forma a não perdermos tempo a somar o bias alteramos o IP para que esta adição passe a ser realizada em HW.

O ip foi alterado de maneira a receber na stream de entrada um inteiro que é utilizado para inicializar o acumulador que guarda o resultado de B por cada uma das linhas da matriz A. Na aplicação SW apenas foi realizada uma pequena alteração. Agora após o envio da coluna da matriz B (linha de B transposto) é necessário enviar um inteiro bias correspondente à linha da matriz que C que se está a calcular.

Com esta pequena otimização obtiveram-se os seguintes resultados.

Layer	Convolutional	Pooling	Fully-Connected	Softmax	Tempo Total
Time (μs)	6038	1325	2997	5	10365

Tabela 3: Tempos de execução de cada layer após alteração do IP

4.3 Paralelização pelos 2 cores

Enquanto que até aqui usou-se sempre uma versão sequencial agora passar-se-á a paralelizar o processamento dos dados de maneira a acelerar todo o processo. Uma vez que a FPGA

usada possui um processador com dois ARM cores tentou-se distribuir o processamento dos dados de forma igual por cada um dos dois cores.

Uma vez que existem 2 cores se fosse possível dividir a execução de forma igual por cada um deles o speed-up que se conseguiria atingir seria 2. No entanto existem operações que apesar de poderem ser paralelizadas o speed-up realizado não seria suficiente para compensar o overhead criado pela sincronização dos dados entre cores.

Uma vez que cada core possui a sua própria cache para que os dados de um processador estejam disponíveis quando necessários no outro processador é necessário que seja realizado um flush destes dados para a memória principal. Sempre que um flush da cache de um core é realizado o outro tem de invalidar a cache na região correspondente. Para da coerência entre caches é necessário garantir sincronização entre cores, isto é realizado com flags que são verificadas após cada tarefa que está a ser paralelizada. Estas flags são guardadas numa zona de memória onde a cache está sempre desativada.

Uma vez que se utilizou dois cores tiveram de ser criados dois BSP, um para cada core, e para cada um deles uma aplicação diferente. Estas aplicações correm assim em paralelo em cada um dos cores.

Foram paralelizadas a Forward Connected Layer e a Forward Maxpool Layer.

A Forward Maxpool Layer realiza o pooling das 22 imagens resultantes da Convolutional Layer. A operação realizada consiste em percorrer cada uma das imagens e torna-las em imagens 12*12 escolhendo para cada pixel da nova imagem o valor máximo na região 4x4 correspondente na imagem original. Esta tarefa foi dividida igualmente pelos dois cores sendo que cada um deles realiza esta operação em metade das 22 imagens resultantes da Convolutional Layer.

A Forward Connected Layer condensa as 20 imagens de dimensão 12x12 num vetor com dimensão dez. Esta diminuição de tamanho é realizada multiplicando cada uma das imagens individualmente por dez diferentes vetores de pesos. Mais uma vez a divisão é feita dividindo metade das imagens que resultantes da layer anterior por cada um dos cores.

Com esta divisão obtiveram-se os seguintes tempos em cada Layer.

Layer	Convolutional	Pooling	Fully-Connected	Softmax	Tempo Total
Time (μs)	6038	712	1528	5	8238

Tabela 4: Tempos de execução após alteração do IP e paralelização das layers 2 e 3

Verifica-se como esperado que a paralelização das layers reduz o tempo de cada uma para praticamente metade. Uma vez que tal redução no tempo foi conseguida podemos concluir que o overhead resultante da sincronização entre cores é desprezável face ao ganho em tempo resultante da divisão de tarefas pelos cores. Isto mostra que as tarefas foram bem divididas e que as esperas para sincronização foram implementadas corretamente.

Esta foi a melhor optimização conseguida sendo o speed-up neste caso 4.5.

Para além das paralelizações realizadas tentou-se ainda dividir pelos dois cores a preparação da matriz A. Conseguiu-se com que a preparação fosse feita em ambos os cores no entanto o resultado final da classificações era sistematicamente errado algo que provavelmente é explicado por problemas de coerência nas zonas de memória que devem ser partilhadas após a preparação da matriz. Ainda assim, obtendo um resultado errado, foi possível medir que a Convolutional Layer gastaria 5153us nessa situação. Este valor corresponderia a um speed-up total de 5.0 em vez dos 4.5 obtidos pela solução implementada. No entanto, uma vez que esta implementação não produzia um resultado correto foi retirada

da implementação final e assim sendo apenas as layers 2 e 3 correm em paralelo sendo o speed-up 4.5 como já foi referido.

5 Resultados

Implementação	L1 (μs)	L2 (μs)	L3 (μs)	L4 (μs)	Total (μs)	Speed-Up
SW-Only	32839	1340	2998	5	37182	
SW-HW	6612	1342	2998	5	10957	3.4
SW-HW (add bias no IP)	6038	1325	2997	5	10365	3.6
Dual Core SW-HW (add bias no IP)	6038	712	1528	5	8238	4.5

Tabela 5: Tempos de execução para cada implementação

Comparando as várias otimizações realizadas percebemos que aquela que teve mais impacto no speed-up final foi utilização do IP como acelerador. Este resultado é o esperado uma vez que como foi mostrado na secção Introdução a Convoluitonal Layer que é realizada como multiplicação de matrizes ocupa 88% do tempo total de execução do programa.

Destaca-se ainda que a utilização de uma implementação dual core reduziu para cerca de metade os tempos de execução nas Layers em que foi implementada. A utilização de dual core poderia ser utilizada ainda na primeira Layer para preparar a matriz A, como foi explicado na secção anterior, o que levaria a uma aceleração de todo o processo ainda maior. A paralelização da SoftMax Layer não foi equacionada uma vez que o seu tempo de execução é quase insignificante no geral de todo o processo.

5.1 Possiveis Melhoramentos

Poder-se-ia ter-se usado o IP para realizar a Fully Connected Layer uma vez que a operação realizada nesta Layer consiste na multiplicação de matrizes. Na tabela 6 mostra-se a percentagem do tempo de execução de cada uma das Layers com a implementação final. A Fully Connected Layer que inicialmente representava cerca de 8% do tempo total de execução representa agora 18.4% o que é justificado pelo facto de o "peso" da Convolutional Layer ter descido bastante a partir do momento em que esta passou a ser realizada em HW. Seria de esperar uma redução de tempo na layer 3 se esta também tivesse sido realizada em HW ainda assim como o tempo gasto nesta layer é relativamente pouco e considerando o overhead das comunicações o ganho de tempo não seria tão substancial como foi para o caso da Convolutional Layer.

Layer	Convolutional	Pooling	Fully-Connected	Softmax	Tempo Total
Time (μs)	6038	712	1528	5	8238
Fracção da execução(%)	72.8	8.5	18.4	0.06	

Tabela 6: Tempos de execução de cada layer e respectiva fracção

A implementação realizada apenas inclui um IP para realização da multiplicação de matrizes em HW. A utilização de recursos por parte desta implementação está representada na imagem 2.

Como se pode observar a quantidade de recursos de lógica programável é ainda muito inferior à total capacidade da FPGA utilizada. A percentagem de Look-Up Tables é apenas

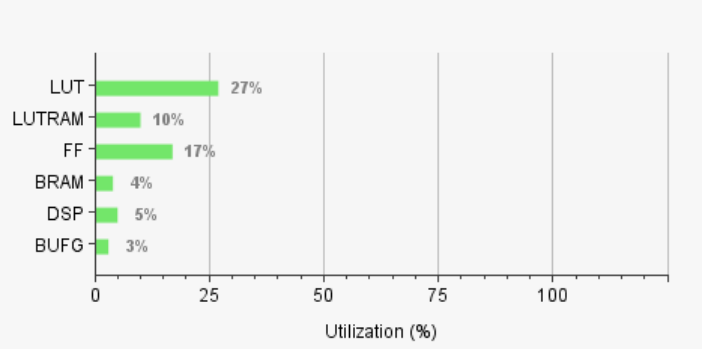

Figura 2: Recursos da FPGA utilizados

27% e por essa razão uma maior utilização dos recursos da FPGA seria possível e traria melhores resultados. Uma vez que os recursos utilizados são sensivelmente um quarto dos totais seria possível implementar um sistema com 4 IP e 4 DMA utilizando assim as 4 AXI HP interfaces (high performance port) existentes no PS (processing system). Desta forma e uma vez que estamos a utilizar ambos os cores do processador seria possível dividir a utilização dos IP pelos cores do processador sendo que cada core utilizaria dois IP para realizar as multiplicações em paralelo.

Ao fazermos isto poderiamos não só preparar a matriz A na primeira Layer em paralelo mas também realizar a propria da multiplicação em paralelo. Isto siginifica dividir toda a primeira Layer por ambos os cores o que em teoria levaria à diminuição do tempo de execução desta etapa para metade.

Estas possíveis futuras implementações resultariam numa diminuição do tempo total de execução melhorando o speed-up no entanto as implementações realizadas já aceleraram a parte mais substancial do programa.

6 Conclusão

No final de todas as optimzações efetuadas, um speed-up de 4.5 é uma boa melhoria (usando um DMA e Dual-core) quando comparada com o abrandamento obtido na primeira parte. Apesar do desempenho extraido da DMA é impressionante quando as layers fazem uso do IP, a lógica programavél ainda está áquem do seu potencial total. A versão final de hardware utiliza menos de metade das portas lógicas, havendo espaço para muito mais utilização destes componentes podendo ser possivel aumentar a velocidade da classificação das imagens. Não só isto mas também há espaço para escalar o hardware para utilizar 4 IPs e 4DMAs. Este escalamento ainda estaria limitado pela overhead do acesso à data. A alteração do IP para tratar da matrix A seria uma boa forma de aumentar a velocidade de classificação, já que esta estaria guardada em memória e não haveria necessidade de enviar diversas esta matriz. Uma outra forma de melhorar isto seria usar os comandos non-blocking DMA e começar a processar a pooling layer logo após a convoução estar feita. Num modo geral explorámos diferentes abordagens e formas de optimizar cada uma das layers sendo que certas certas optimizações não produziram os resultados esperados, para os quais se teve de tentar novas abordagens. Viu-se a flexibilidade e adaptabilidade da placa para diferentes problemas, graças, em grande parte ao hardware adaptável.