

EI-SE4
Projet CoDesign
Compte Rendu

Laith ALKHAER - Alexandre VAUDELLE
25/05/2023

Profilage

On compile le programme avec gcc et on le teste, le programme reconnaît bien le chiffre 2 une fois les fichiers des poids sont bien configurés en des fichiers en-tête (.h).

On réalise un profil d'exécution du réseau en identifiant les fonctions chronophages en utilisant l'option -pg:

```
pCDesign/Projet_codesign/Lenet on main [?] via C v13.1.1-gcc took 4s
> gprof out
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self     total
time  seconds    seconds   calls  ms/call  ms/call  name
50.07      0.01      0.01         1     10.01    10.01  calculateLayer3
50.07      0.02      0.01         1     10.01    10.01  calculateLayer4
0.00      0.02      0.00         1         0.00     0.00  InitHostMem
0.00      0.02      0.00         1         0.00     0.00  calculateLayer1
0.00      0.02      0.00         1         0.00     0.00  calculateLayer2
0.00      0.02      0.00         1         0.00     0.00  calculateLayer5
```

On observe ainsi qu'un profilage software est trop lent pour mesurer certains temps d'exécution, mais que les couches 3 et 4 sont les plus lentes.

ARM 9 - Zynq

1. Instancier le processeur ARM sur Vivado et générer la configuration matérielles (bitstream).

On instancie le processeur ARM sur Vivado et on génère le bitstream.

2. Apporter les modifications nécessaires à exécuter le programme sur le processeur ARM.

Pour cette partie, on rencontre de nombreuses difficultés à cause des fonctions `xil_printf` et `printf` qui entrent en conflit, on réussit néanmoins à faire fonctionner le code et faire nos mesures (capture ci-dessous).

3. Compilez le programme et testez-le en le téléchargeant dans le ZYNQ de la carte Zedboard et en l'exécutant in-situ.

On compile le programme et on le teste. Le programme fonctionne !

4. Profiler le code exécuté sur le processeur ARM avec des timers logiciels ou matériels.

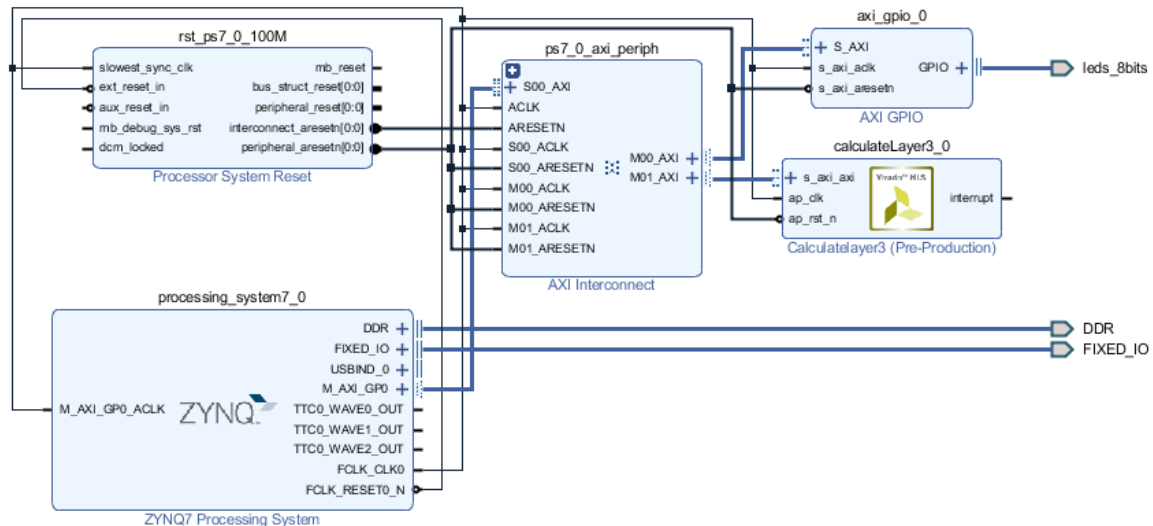
On profile le code exécuté sur le processeur ARM avec des timers matériels Xtime, et on a obtenu les résultats suivants:

```
Connected to: Serial ( /dev/ttyACM0, 115200, 0, 8 )
0 :
1 :
2 :
3 :
4 :
5 :
6 :
7 :
8 :
9 :
Le resultat est : 2
Couche 1: output took 8
, Couche 2: output took 4506
, Couche 3: output took 28438
, Couche 4: output took 19746
, Couche 5: output took 102
```

Et on arrive donc à la conclusion que la couche 3 est la couche la plus gourmande en temps de calcul, il conviendrait donc de la rendre matérielle en priorité.

High Level Synthesis - HLS

Nous avons vu à la partie précédente que la couche 3 est la couche la plus gourmande en temps de calcul, on va donc la matérialiser en priorité:



On cherche alors à atteindre la contrainte qui nous a été fixée, à savoir un fonctionnement à moins de 1 ms.

On commence donc par faire une intégration matérielle de la troisième couche sur Vivado HLS. En profilant on a d'après la simulation les temps estimés suivants :

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	12.00	11.49	1.50

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
1128101	1209351	1128101	1209351	none

On rappelle qu'une intégration matérielle sans optimisation par l'utilisateur est optimisée par défaut pour occuper le moins de surface possible sur la FPGA et non pour la latence, et c'est pour cette raison qu'on a ce résultat "lent".

En calculant, on a donc un temps d'exécution totale d'environ 13.8 ms.

Pour optimiser notre intégration matérielle de la couche 3, on essaye alors d'implémenter un pipeline afin de réduire ce temps.

Après de nombreux essais, on trouve à taton la solution la plus optimale suivante :

```
for( i=0;i<50;i++)
  //#pragma HLS pipeline II=1
  for(j=0;j<5;j++)
    //#pragma HLS pipeline II=2
    for(k=0;k<5;k++){
      #pragma HLS pipeline II=69
      somme = Layer2_Weights_CPU[26*6*i];
      for( m=0;m<5;m++){
        //#pragma HLS pipeline II=6
        for( n=0;n<5;n++){
          //#pragma HLS pipeline II=1
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)] * Layer2_Neurons_CPU[13*13*0+13*(2*j+m)+(2*k+n)];
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)+1] * Layer2_Neurons_CPU[13*13*1+13*(2*j+m)+(2*k+n)];
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)+2] * Layer2_Neurons_CPU[13*13*2+13*(2*j+m)+(2*k+n)];
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)+3] * Layer2_Neurons_CPU[13*13*3+13*(2*j+m)+(2*k+n)];
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)+4] * Layer2_Neurons_CPU[13*13*4+13*(2*j+m)+(2*k+n)];
          somme += Layer2_Weights_CPU[26*6*i+1+6*(n+5*m)+5] * Layer2_Neurons_CPU[13*13*5+13*(2*j+m)+(2*k+n)];
        }
      }
    }
  }
```

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	12.00	12.85	1.50

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
189437	189437	189437	189437	none

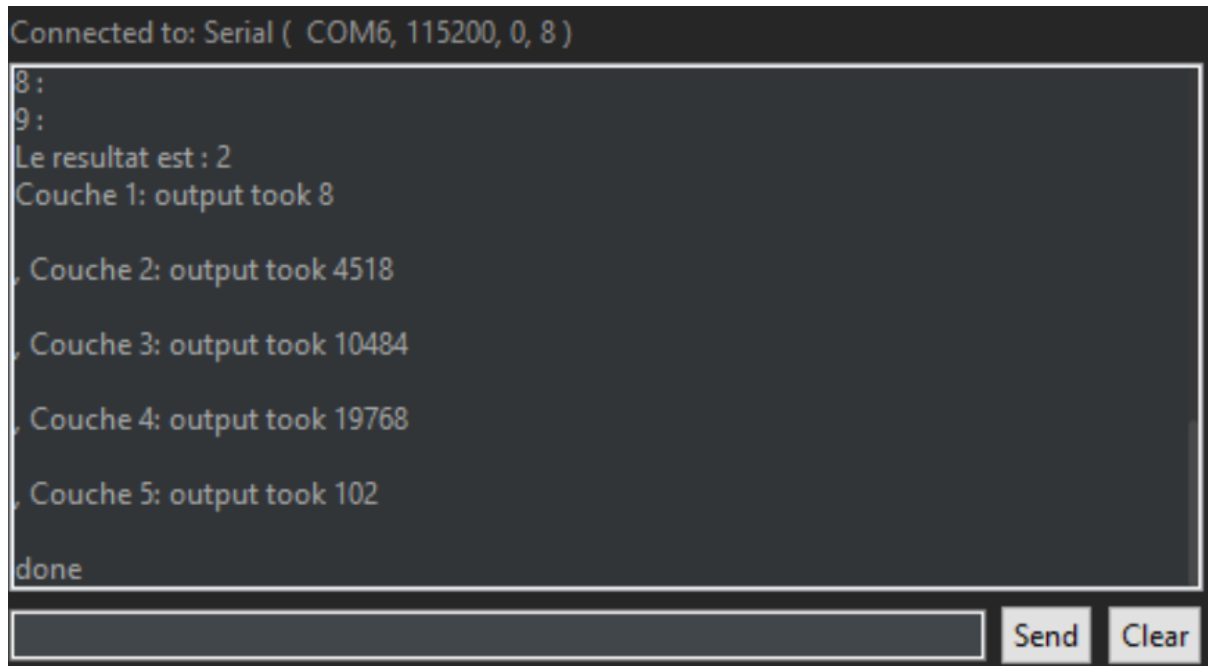
Le temps d'exécution est donc descendu à $12.85 \times 189437 \times 10^{-6} = 2.4$ ms. Il reste donc encore une petite optimisation supplémentaire à réaliser.

On vérifie tout d'abord que le code marche toujours avant de l'implémenter sur la carte:

```
Generating csim.exe
0 : -0.976698
1 : -0.997205
2 : 0.928655
3 : -0.991480
4 : -1.101828
5 : -0.986899
6 : -0.984240
7 : -0.964750
8 : -1.007396
9 : -0.991236
Le resultat est : 2
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.
```

La reconnaissance du chiffre fonctionne toujours !

On génère ensuite un bitstream avec Vivado de la version optimisée de l'architecture et on l'implémente sur la carte, et on observe les nouveaux résultats suivants:



```
Connected to: Serial ( COM6, 115200, 0, 8 )
8:
9:
Le resultat est : 2
Couche 1: output took 8
Couche 2: output took 4518
Couche 3: output took 10484
Couche 4: output took 19768
Couche 5: output took 102
done
```

On voit bien que cette fois on a un temps de calcul pour la couche 3 en HLS optimisée qui est trois fois inférieur à ce que l'on avait sur ARM, l'exécution du réseau a bien été accélérée.

Pour le niveau d'optimisation suivant, on a le choix entre la couche 4 et la 2. D'après le premier profilage réalisé, on sait que la couche 2 prend environ 0.4 ms, et la couche 4 en prend 2. Il conviendrait donc d'intégrer la couche 4 en priorité avec la couche 3.

On tente donc d'intégrer la couche 4 en suivant les mêmes étapes que pour la couche 3. Tout va bien jusqu'au moment de la génération du bitstream sous Vivado, là où on se rend compte qu'il est impossible de mettre ces deux couches sur la FPGA de la Zedboard car il n'y a pas assez d'éléments logiques, même lorsqu'on optimise les deux IPs (couche 3 et 4) en surface.

Il nous reste donc la couche 2 à intégrer dans la FPGA mais cette couche ne prend pas suffisamment de puissance de calcul pour justifier son intégration au dépens de l'optimisation temporelle de la couche 3. Nous décidons donc de conclure ce projet ici.
