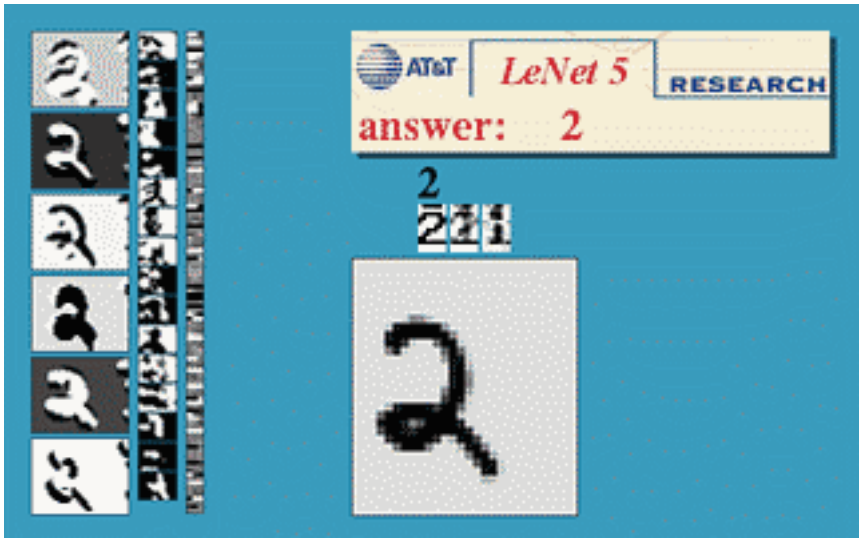
Projet Codesign - Plateform Based Design

Spécialité Electronique-Informatique
Parcours Systèmes Embarqués
Polytech Sorbonne



Profilage

Dans cette partie, vous allez effectuer le profil d'exécution d'un programme écrit en langage C qui exécute un réseau de neurones de reconnaissance de caractères. Ce réseau de neurones décrit par P.Y. Simard [Simard] est un réseau de type TDNN hérité des réseaux LeNet conçus par Y. Lecun [LeCun].

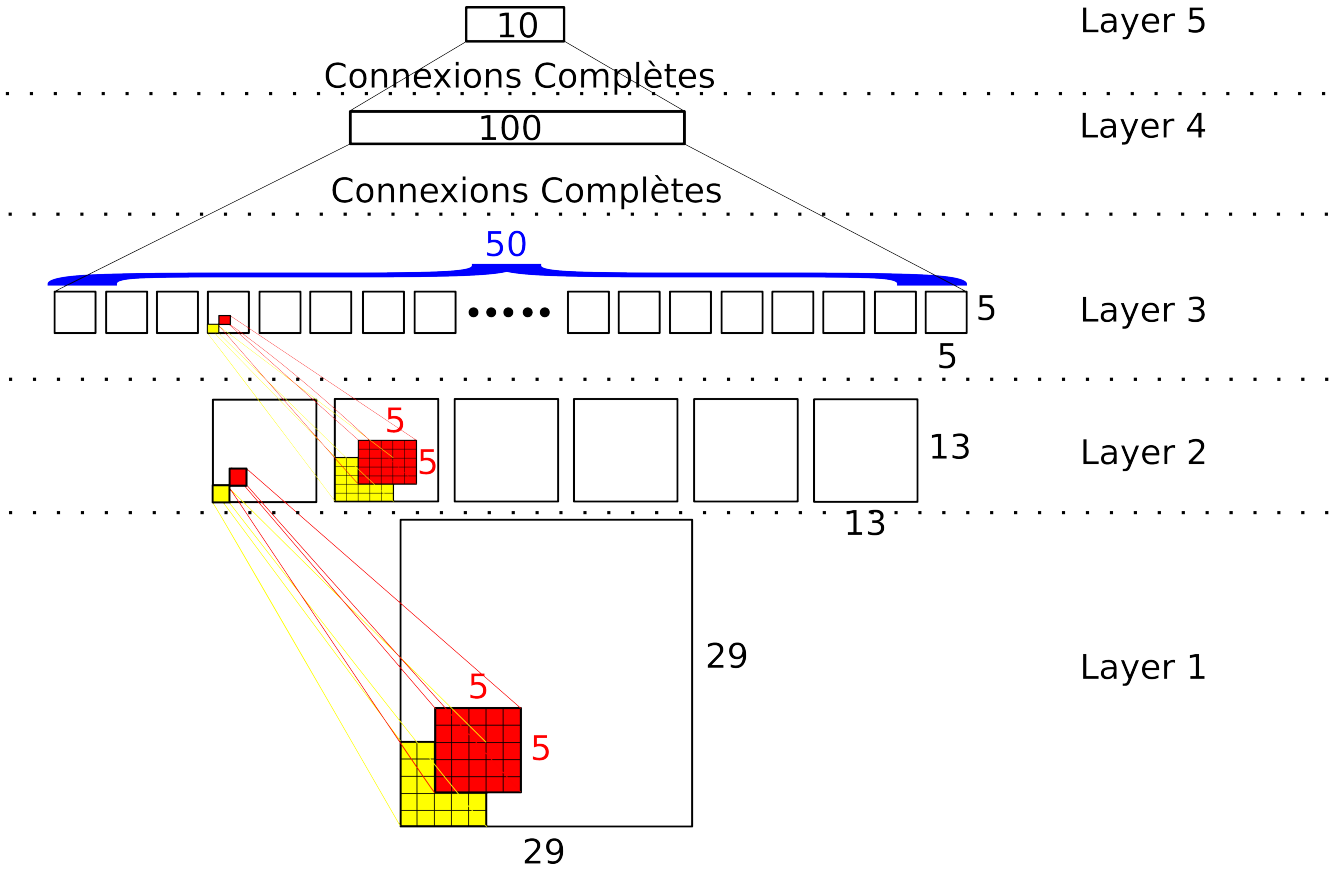


Figure 1: Schéma du réseau Lenet-Simard

Ce réseau visible sur la figure 1, est composé de cinq couches de neurones. Les trois premières couches de neurones sont des couches bidimensionnelles, les deux dernières des couches monodimensionnelles.

La structure du réseau est la suivante :

- Une couche d'entrée de 29*29 neurones que par la suite nous appellerons : `Layer1`
- Une couche cachée de 6*13*13 neurones que nous appellerons par la suite : `Layer2`. Dans cette couche on considérera 6 ensembles de 13*13 neurones.

- Une couche cachée de 50*5*5 neurones, appelée : `Layer3`. Dans cette couche on considérera 50 ensembles de 5*5 neurones.
- Une couche cachée de 100 neurones, du doux nom de : `Layer4`
- Une couche de sortie de 10 neurones, répondant au patronyme de : `Layer5`

le schéma de connexion entre ces couches est le suivant :

- entre la couche `Layer1` et la couche `Layer2` les neurones d'un ensemble de 13*13 neurones, sont connectés suivant un noyau de voisinage de 5*5 neurones, c'est à dire que chaque neurone de cet ensemble de la couche `Layer2` est connecté à un ensemble de 5*5 neurones de la couche `Layer1`. La relation entre les ces neurones est la suivante :

$$V_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X^1_{(i,j)} + W_{bias}$$

et

$$X_{(k,l)} = (1.7159 * tanhf(0.66666667 * V_{(k,l)}))$$

avec $k \in [0,12]$ et $l \in [0,12]$ qui indice un ensemble de 13*13 neurones de la couche `Layer2` et avec $i \in [0,28]$ et $j \in [0,28]$ qui indice les neurones de la couche `Layer1`. $V_{(k,l)}$ est le potentiel post-synaptique du neurone (k,l) d'un ensemble de 13*13 neurones de la couche `Layer2`. $X_{(k,l)}$ est l'état du neurone. Chaque neurone d'un même ensemble se partage le même jeu de poids synaptiques.

- entre la couche `Layer2` et la couche `Layer3` les neurones d'un ensemble de 5*5 neurones sont connectés suivant un noyau de voisinage de 5*5 neurones, c'est à dire que chaque neurone de la couche `Layer3` est connecté à un ensemble de 5*5 neurones des 6 ensembles de 13*13 neurones de la couche `Layer2`. La relation entre les neurones est la suivante :

$$V^0_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V^1_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V^2_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V^3_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V^4_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V^5_{(k,l)} = \sum_{\substack{i=2k \\ i=2k+4 \\ j=2l \\ j=2l+4}} W_{(k,l),(i,j)} * X_{(i,j)}$$

$$V_{(k,l)} = V^0_{(k,l)} + V^1_{(k,l)} + V^2_{(k,l)} + V^3_{(k,l)} + V^4_{(k,l)} + V^5_{(k,l)} + W_{bias}$$

et

$$X_{(k,l)} = (1,7159 * tanhf(0.66666667 * V_{(k,l)}))$$

avec $k \in [0,4]$ et $l \in [0,4]$ qui indice un ensemble de 5*5 neurones de la couche `Layer3` et avec $i \in [0,12]$ et $j \in [0,12]$ qui indice un ensemble de 13*13 neurones de la couche `Layer2`.

- $V^0_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le premier ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.
- $V^1_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le second ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.
- $V^2_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le troisième ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.

- $V^3_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le quatrième ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.
- $V^4_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le cinquième ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.
- $V^5_{(k,l)}$ est le potentiel synaptique partiel émanant des connexions entre le sixième ensemble de 13*13 neurones de la couche `Layer2` et un ensemble de 5*5 neurones de la couche `Layer3`.

$V_{(k,l)}$ est le potentiel post-synaptique du neurone (k,l) d'un ensemble de 5*5 neurones de la couche `Layer3`. $X_{(k,l)}$ est l'état du neurone. Chaque neurone d'un même ensemble se partage les mêmes 6 jeux de poids synaptiques.

- entre la couche `Layer3` et `Layer4` les neurones sont connectés suivant un schéma complet, c'est à dire que chaque neurone de la couche `Layer4` est connecté à tous les neurones de la couche `Layer3`

- entre la couche `Layer4` et `Layer5` les neurones sont connectés suivant un schéma complet, c'est à dire que chaque neurone de la couche `Layer5` est connecté à tous les neurones de la couche `Layer4`

On désire réaliser un profil d'exécution et une analyse permettant d'identifier les endroits à paralléliser, ces endroits devront être voraces en terme de puissance de calcul et se prêter à une parallélisation efficace.

Le programme qui vous est livré exécutant ce réseau respecte un certains nombres de contraintes de programmation :

- Les valeurs des états des neurones d'une couche sont stockées dans un tableau monodimensionnel nommé `Layer`N`_Neurons_CPU` où N est le numéro de la couche, par exemple pour la couche `Layer2` le tableau se nomme `Layer2_Neurons_CPU`

- Les valeurs des poids des neurones d'une couche sont stockées dans un tableau monodimensionnel nommé `Layer`N`_Weights_CPU` où N est le numéro de la couche, par exemple pour la couche `Layer2` le tableau se nomme `Layer2_Weights_CPU`

- Il y a 3 fonctions `InitHostMem`, `readIn` et `output` qui permettent d'initialiser les poids, les entrées et d'écrire le résultats dans des fichiers. Ces fonctions sont données en annexes.

- Vous disposez de 4 fichiers contenant les poids des connexions entres les couches,
  - `lw1.wei` pour les connexions entre `Layer1` et `Layer2`
  - `lw2.wei` pour les connexions entre `Layer2` et `Layer3`

  - `lw3.wei` pour les connexions entre les couches `Layer3` et `Layer4`
  - `lw4.wei` pour les connexions entre les couches `Layer4` et `Layer5`.

- Les poids et les états sont codés en flottant simple précision pour toutes les couches sauf la couche de sortie `Layer5`
- Les poids et les états sont codés en flottant double précision pour la couche de sortie `Layer5`

1. Identifiez les différentes fonctions du programme et leurs rôles.
2. Compilez le programme et testez-le.
3. En utilisant l'option `-pg` du compilateur `gcc` faites un profil d'exécution de ce réseau en identifiant les fonctions chronophages.
4. Il vous ait fourni une fonction `dtime()`, cette fonction dont un exemple d'utilisation est donnée en annexe, permet de mesurer le temps écoulé en allant lire les timers interne du CPU. Utilisez cette fonction pour mesurer le temps d'exécution de toutes les fonctions.

Fonctions utiles à la partie exhibition

fonction InitHostMem

```
void InitHostMem(float *Layer1_Weights_CPU,float *Layer2_Weights_CPU,
                              float *Layer3_Weights_CPU,float *Layer4_Weights_CPU)
{
// initial layer 1 weight
FILE * pFile1 = fopen ("lw1.wei","rb");
if (pFile1 != NULL)
{
for(int i=0;i<156;++i)
fread(&(Layer1_Weights_CPU[i]),sizeof(float),1,pFile1);
fclose (pFile1);
}

// initial layer 2 weight
FILE * pFile2 = fopen ("lw2.wei","rb");
if (pFile2 != NULL)
{
fread(Layer2_Weights_CPU,sizeof(float),7800,pFile2);
fclose (pFile2);
}
// initial layer 3 weight
FILE * pFile3 = fopen ("lw3.wei","rb");
if (pFile3 != NULL)
{
fread(Layer3_Weights_CPU,sizeof(float),125100,pFile3);
fclose (pFile3);
}
// initial layer 4 weight
FILE * pFile4 = fopen ("lw4.wei","rb");
if (pFile4 != NULL)
{
fread(Layer4_Weights_CPU,sizeof(float),1010,pFile4);
fclose (pFile4);
}
}
```

fonction readIn

```
void readIn(float *layer1)
{
FILE *fp;
fp=fopen("in.neu","rb");
if(fp)
{
fread(layer1,sizeof(float),29*29,fp);
fclose(fp);
}
}
```

fonction output

```
void output(double *final)
{
FILE *fp=0;
fp=fopen("out.res","wb");
if(fp)
{
fwrite(final,sizeof(double),10,fp);
fclose(fp);
}
}
```

fonction dtime

```
#include <stdio.h>

void permutation_cpu(char * a, char * b)
{
  char temp = *a;
  *a = *b ;
  *b = temp ;
}

void tribulle_cpu(char t[], int n)
{
  int changement=1, i, j ;

  for (i=1 ; i<=n && changement ; i++) {
    changement = 0 ;
    for (j=n ; j >= i+1 ; j--) {
      if (t[j] < t[j-1]) {
        permutation_cpu(&t[j], &t[j-1]) ;
        changement = 1 ;
      }
    }
  }
}

int main(void)
{
char mat[256];
double t1,t2,t3;
    t1=dtime();
    tribulle_cpu(mat,256);
    t2=dtime();

    t3=t2-t1;

    fprintf(stderr,"Le tri du tableau a duré %6f s \n",t3);
}
```

ARM et IP matérielle

Dans cette partie vous allez, d'abord exécuter le code uniquement sur le processeur *ARM*®. Il faudra apporter certaines modifications au programme. Pour cela il faudra d'abord instancier sur Vivado le processeur et après sur SDK développer la partie code source qui sera exécutée par le processeur. Une fois le code modifié et compilé vous pourrez l'exécuter et afficher sur un terminal (utiliser minicom pour l'affichage) le résultat de la reconnaissance du caractère. Une fois validé le bon fonctionnement du code et du réseau convolutionel vous pouvez faire le profilage et évaluer les performances de chaque fonction en terme de latence. Réaliser ensuite un système sur puce comprenant une partie logicielle exécutée par un processeur *ARM*® dual-core Cortex™-A9 MPCore™Processing System, et une partie matérielle accélérant les calculs (fonction) les plus coûteux identifiés lors du dernier profilage de l'application.

Pour réaliser ce système, il vous faudra prendre en compte les spécificités de la carte ZedBoard - Digilent comprenant un FPGA Zynq7000. Vous veillerez à utiliser efficacement les ressources à votre disposition : mémoire, blocs DSP, éléments logiques ....

1. Instancier le Soc Zynq en configurant le processeur ARM sur Vivado et générer la configuration matérielles (bitstream).
2. Apporter les modifications nécessaires à exécuter le programme sur le processeur ARM.
3. Compilez le programme et testez-le en le téléchargeant dans le ZYNQ de la carte Zedboard et en l'exécutant in-situ.
4. Profiler le code exécuté sur le processeur ARM avec des timers logiciels ou materiels.
5. Réalisez une IP avec une interface AXI(Light, Stream ou Burst) permettant d'afficher sur les leds en binaire le résultat de la reconnaissance, si rien n'est reconnu vous allumerez tous les leds indiquant une impossibilité de reconnaître le caractère.
6. Réalisez le partitionnement logiciel et matériel en concevant un synoptique (un schéma) de l'architecture du système sur puce. Vous veillerez à bien identifier les parties logicielles et matérielles, ainsi que le support de communication qui les relie (Interface AXI - une parmi les trois).
7. Proposez une architecture d'IP mettant en œuvre la partie matérielle de votre synoptique.
8. Décrivez, simulez et profilez votre architecture:
   (a) Décrire votre IP en VHDL.
   (b) Valider par simulation le comportement de votre IP.
   (c) Intégrer votre IP avec le processeur ARM, via l'interface AXI.

(d) Développer/modifier le programme d'origine afin que le processeur puisse communiquer avec votre IP.

(e) Exécutez-le afin de vérifier le bon fonctionnement et faire à nouveau le profilage.

(f) Comparez les résultats obtenus en terme de temps pour chaque fonction et en terme de taux d'occupation du FPGA. Vous utiliserez pour cela le timer décrit précédemment.

(g) Calculez l'accélération obtenue grâce à l'utilisation d'une IP matérielle

Timer de mesures des temps avec un processeur ARM sans OS (Bare-Metal)

https://forums.xilinx.com/t5/Embedded-Processor-System-Design/xtime-l-h-file-IP/td-p/799392

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xtime_l.h"

void print(char *str);

int main() {
    XTime tStart, tEnd;

    init_platform();

    XTime_GetTime(&tStart);
    print("Hello World\n\r");
    XTime_GetTime(&tEnd);

    printf("Output took %llu clock cycles.\n", 2*(tEnd - tStart));
    printf("Output took %.2f us.\n",
           1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000));

    return 0;
}
```

References

[LeCun] *Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning*, Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, **R. E. Howard and W. Hubbard , in Sanchez-Sinencio, E. and Lau, C. (Eds), Artificial Neural Networks, 463-468, IEEE press, 1992**

[Simard] *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*, Patrice Y. Simard, Dave Steinkraus, John C. Platt, **ICDAR archive Proceedings of the Seventh International Conference on Document Analysis and Recognition - 2003**