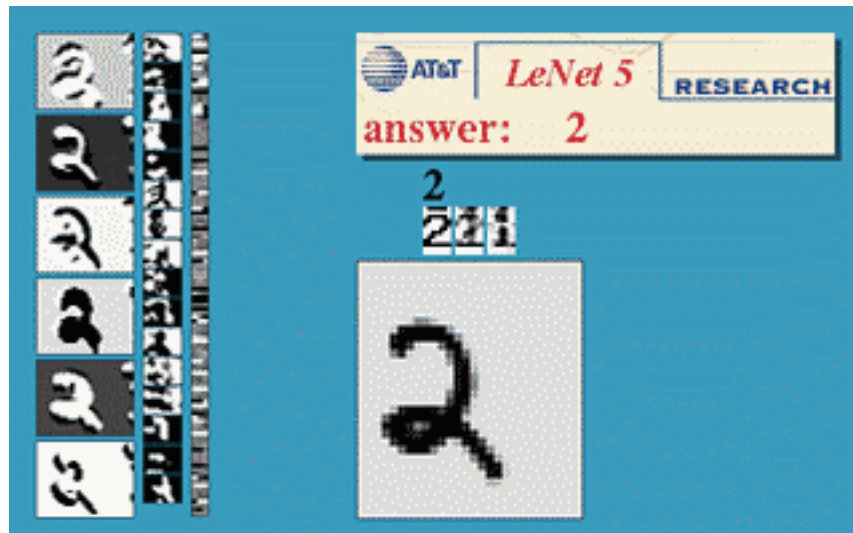


Platform Based Design - CoDesign

Université Pierre et Marie CURIE
Master d'Informatique Spécialité SESI



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].

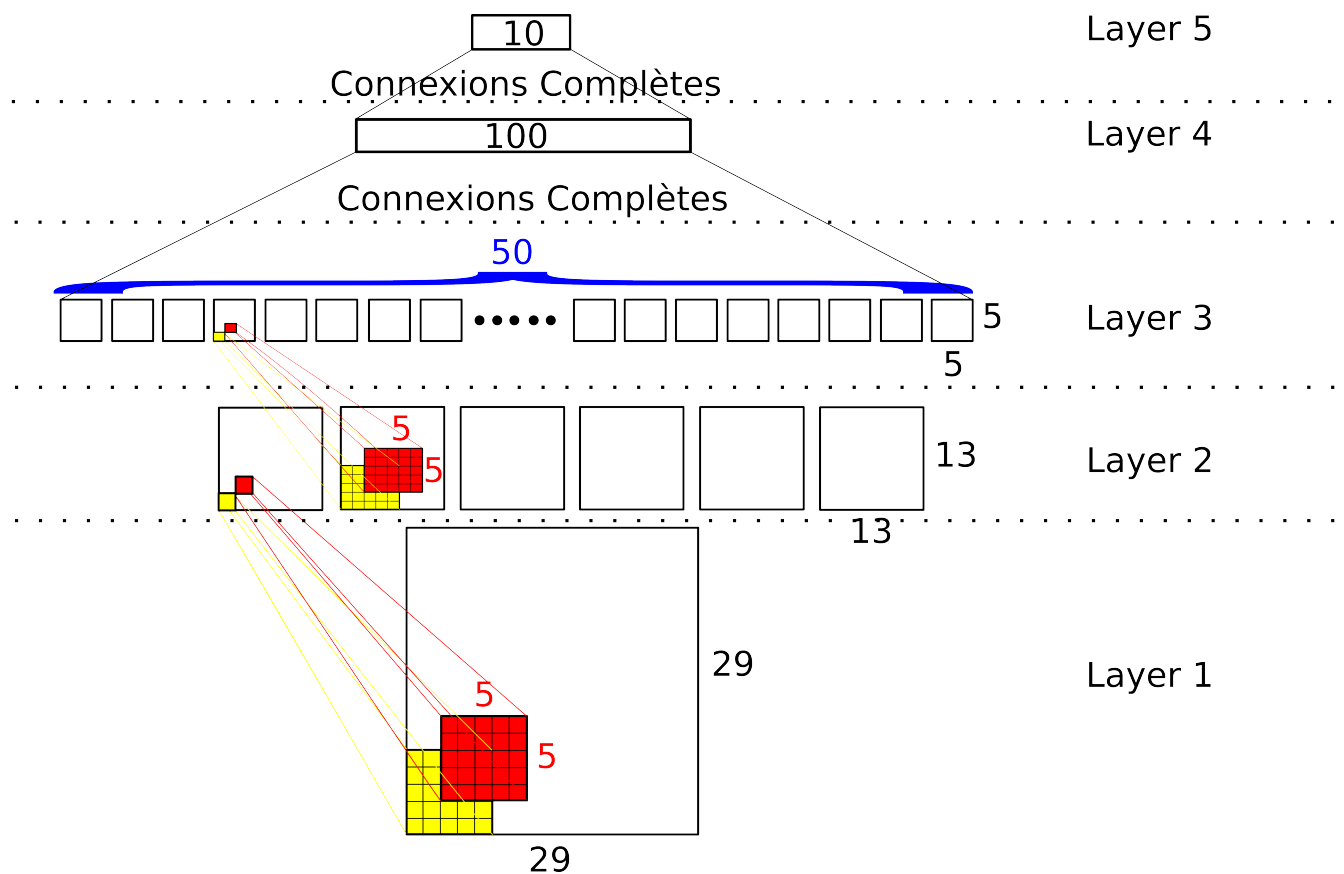


Figure 1: Schéma du réseau Lenet-Simard

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 $6 \times 13 \times 13$ 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 :

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

et

$$X_{(k,l)} = (1.7159 * \tanh f(0.66666667 * V_{(k,l)}))$$

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 :

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

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

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

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

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

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

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

et

$$X_{(k,l)} = (1, 7159 * \tanh f(0.66666667 * V_{(k,l)}))$$

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.

- $V_{(k,l)}^0$ 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.
- $V_{(k,l)}^1$ 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.
- $V_{(k,l)}^2$ 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.

- $V_{(k,l)}^3$ 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.
- $V_{(k,l)}^4$ 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.
- $V_{(k,l)}^5$ 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 `LayerN_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 `LayerN_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 entre 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.

1 Fonctions utiles à la partie exhibition

1.1 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);
    }
}
```


1.2 fonction readIn

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

1.3 fonction output

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

1.4 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);
}
```

Nios II

Dans cette partie vous allez réaliser un système sur puce comprenant une partie logicielle exécutée par un processeur 32bits, le processeur NiosII, et une partie matérielle accélérant les calculs les plus coûteux identifiés lors du profilage de l'application.

Pour réaliser ce système, il vous faudra prendre en compte les spécificités de la carte de développement FPGA, une carte DE2 comprenant un FPGA Cyclone II. Vous veillerez à utiliser efficacement les ressources à votre disposition : mémoire, blocs DSP, éléments logiques

1. Compilez le programme et testez-le en le téléchargeant dans le FPGA de la carte DE2 et en l'exécutant in-situ.
2. Réalisez une IP permettant d'afficher sur les afficheur 7 segments le résultat de la reconnaissance, si rien n'est reconnu vous afficherez un E indiquant une impossibilité de reconnaître le caractère.
3. 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
4. Proposez une architecture d'IP mettant en œuvre la partie matérielle de votre synoptique
5. Testez et simuler votre architecture d'IP
6. Intégrer votre IP avec le processeur NiosII
7. Implémenter votre système, Nios II et IP, et exécutez le afin de réaliser un nouveau profilage.
8. 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 dont l'utilisation est décrite dans la section 1.5
9. Calculez l'accélération obtenue grâce à l'utilisation d'une IP matérielle

1.5 Timer de mesures des temps avec un processeur Nios II

Dans l'environnement de développement du NiosII il est possible d'ajouter une IP d'un timer qui se nomme "Performance Counter Unit", si vous désirez profiler une application il suffit d'ajouter autant d'instances de ce timer que nécessaire.

Le code exemple en langage C qui suit vous explique comment utiliser ce timer. On suppose ici que "PERFORMANCE_COUNTER_0_BASE" est le nom du component timer instantié préalablement. Ce nom est déclaré dans le fichier "system.h"

```
#include "altera_avalon_performance_counter.h"

int main() {

    PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
    PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);

    PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 1);
    fonction_a_mesurer();
    PERF_END(PERFORMANCE_COUNTER_0_BASE, 1);

    PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 2);
    fonction_a_mesurer();
    PERF_END(PERFORMANCE_COUNTER_0_BASE, 2);

    perf_print_formatted_report((void *) PERFORMANCE_COUNTER_0_BASE,
    alt_get_cpu_freq(), 2, "timer1", "timer2");
    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**