Stages M2 2025, Equipe MADMax, Inria/TIMA

28 novembre 2024

Table des matières

1 Analysis of sharing patterns in multicore applications and new instructions to accelerate such patterns 1

2 Introducing New RISC-V Instructions for Memory Accesses in RISC-V 5

3 Reducing The Cost of Branch Mispredictions with Software-hinted Hardware Predication 7

4 High-Level Modelling of a High-Performance L1 Data Cache 12

5 Integration of a High-Performance L1 Data Cache with a GPU 15

6 Hardware implementation of quantized LLM networks on FPGA 18

7 Support of library ONNX to implement pre-trained neural networks on FPGA 20

8 Evaluation de solutions PCI Express et intégration dans un générateur de réseaux de neurones 22

9 Evaluation des mémoires HBM et intégration dans un générateur de réseaux de neurones 23

10 Evaluation de solutions Ethernet et intégration dans un générateur de réseaux de neurones 24

11 Analyse de l'impact de la réorganisation des kernels sur la compressibilité des Réseaux Neuronaux Convolutifs (CNN) 25

11.1 Contexte 25
11.2 Objectifs 25
11.3 Compétences requises 25

12 Analyse non-linéaire des poids des Réseaux Neuronaux Convolutifs (CNN) dans le but d'améliorer leur compressivité 26

12.1 Contexte 26
12.2 Objectifs 26
12.3 Compétences requises 26

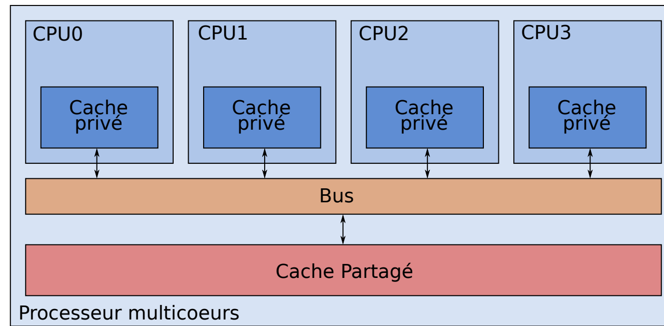

Figure 1 – Processeur multicoeur avec 4 coeurs, chacun possédant sa mémoire cache privée, ainsi qu'une mémoire cache partagée.

1 Analysis of sharing patterns in multicore applications and new instructions to accelerate such patterns

Français

Mots-clés : *processeurs multicoeurs, cohérence de caches, RISC-V*

Contexte : Afin d'améliorer la performance, un processeur généraliste associe à chacun de ses coeurs une mémoire cache privée (utilisable uniquement par ce coeur), qui va garder un sous-ensemble des données utilisées proche des unités d'exécution afin d'en accélérer l'accès. La puce possède de plus un cache de plus grande taille, partagé par tous les coeurs. Cette architecture est illustrée dans la Figure 1.

Afin de faciliter le développement d'applications parallèle, les différentes mémoires caches situées sur la puce assurent la cohérence des données, c'est à dire qu'un mot mémoire peut résider dans plusieurs caches privés uniquement si les processeurs ne font que lire la donnée. Toute écriture entraîne l'invalidation de la donnée dans les caches privés la possédant (excepté l'écrivain), et un cache ayant perdu sa copie suite à une écriture devra recharger la nouvelle version de la donnée. Dans les processeurs généralistes, la gestion de la cohérence est faite par le matériel, et est complètement transparente au programmeur. Dans le cas contraire, le logiciel devrait explicitement gérer la cohérence des données ne serait-ce que pour obtenir un résultat correct, ce qui ralentirai significativement le développement d'applications multithreadées.

Cependant, le matériel gérant la cohérence est obligé de faire certains choix sous-optimaux car il n'a pas une vision globale des motifs d'accès et de partage des données. Par exemple, lors d'un chargement depuis la mémoire, un processeur peut demander à charger une donnée depuis la mémoire et l'insérer dans son cache privé avec uniquement la permission de lire la donnée, ou avec la permission de la lire et de l'écrire. Suivant le motif d'accès et de partage, la bonne décision (du point de vue de la performance et du traffic sur la puce) n'est pas toujours la même. Si la donnée n'est pas partagée, le processeur a tout intérêt à obtenir la permission d'écrire même si il ne fait que lire la donnée, afin d'éviter d'envoyer une seconde requête dans le futur demandant la permission d'écrire sur la ligne déjà présente en lecture seule. Si la donnée est partagée et que le processeur ne va pas l'écrire, alors il vaut mieux au contraire ne pas demander la permission d'écrire, car cela impliquerait d'invalider les autres copies présentes dans le système (i.e., dans les caches privés des autres processeurs). De plus, de manière générale, suivant l'intervalle de temps entre la lecture d'une donnée et son écriture par un processeur, il peut-être soit plus intéressant

d'obtenir la permission d'écrire au plus tôt (à la première lecture) afin de ne pas ralentir l'écriture si elle est proche, ou au plus tard (lors de l'écriture) afin de permettre aux autres lecteurs de garder leur copie aussi longtemps que possible.

Ces motifs d'accès et de partage sont cependant connu par la personne ayant conçu le programme. Il serait donc intéressant de pouvoir exprimer ces motifs afin que le matériel soit en capacité de prendre les bonnes décisions lors de l'exécution, plutôt que de tenter de deviner quelle est la bonne décision. Le jeu d'instruction RISC-V étant ouvert et facilement extensible, il nous donne une opportunité d'étudier cette possibilité, via l'ajout d'instructions indiquant avec quel motif une donnée est accédée ou partagée.

Objectifs : Le stage s'articule en trois temps. Premièrement, il conviendra de faire une revue de l'état de l'art portant sur les différents motifs de partages observés dans les programmes multithreads et les heuristique matériels permettant de les identifier (notamment leur précision). Cela permettra à l'étudiant ou l'étudiante de formuler les motifs que l'on souhaiterai exprimer via des instructions dédiées. Dans un second temps, il conviendra de quantifier la fréquence de tels motifs dans des programmes multithreadés classiques (suite de benchmarks PARSEC [1]), afin d'une part de raffiner les motifs à considérer et d'autre part d'identifier lesquels prioriser. Enfin, il conviendra d'estimer le gain en performance que pourrait apporter l'ajout de telles instructions en i) Ajoutant le support de nouvelles instructions dans gcc (via intrinsic) ii) Ajout des instructions dans les benchmarks PARSEC et iii) Simulation des benchmarks PARSEC sur un modèle de processeur généraliste multicoeur auquel l'étudiant ou l'étudiante aura ajouté le support pour les nouvelles instructions non seulement dans le processeur mais dans le bloc matériel responsable de la gestion de la cohérence. Pour cette dernière étape, on se place dans un simulateur haut niveau (gem5 [2]) et il ne sera donc **pas** question de développer du matériel en VHDL ou Verilog.

Compétences attendues/mises en œuvre :

- Maîtrise des concepts de synchronisation pour la programmation parallèle (threads, verrous...)
- Des bases en C++
- Maîtrise d' (au moins) un langage de scripting (*e.g.*, Bash, Python)
- Notion de jeu d'instruction (des notions de RISC-V sont bienvenues)
- Des notions sur les protocoles de cohérence sont un plus

English

Keywords : *Multicore, cache coherency, RISC-V*

Context : To improve performance, a general purpose processor associates a private cache memory to each of its cores, in order to keep a subset of data close to the execution units of the core and thereby accelerate data access. The chip also features a larger cache shared by all cores. This architecture is depicted in Figure 1.

To facilitate the development of parallel applications, the various cache memories implemented on chip provide data coherency. That is, a given memory address may be cached in several private caches only if the associated cores are only reading the data. Any write to the data requires invalidating all existing copies in the private caches (except for that of the writer). A cache that lost its copy following a write will have to reload the new version of the data (e.g., from memory). In general purpose processors, coherency is handled by hardware and is completely

transparent to the programmer. If that were not the case, software would have to manage coherency explicitly for the program to be correct, which would significantly slow parallel application development down.

However, the hardware handling coherency is forced to make sub-optimal choices as it does not have a global vision of access and sharing patterns accross the system. For instance, when data is loaded from memory, a core can aks to insert it into its private cache either with read permission or both read and write permissions. Depending on the access and sharing patterns, the correct decision (for performance or chip traffic) is not always the same. If the data is not shared, the core should obtain write permission in order to avoid sending a second request asking for write permission in the future. If the data is shared but the core is only going to read it, then, the core should ask for read permission only. Asking for write permission would imply invalidating the other copies and would be wasteful in this case. In addition, generally speaking, depending on the number of cycles between when a data is read by a core and when it is written by that core, it can be more interesting to obtain the write permission early (when reading) in order to not slow down the write if it is close in time, or late (when performing the write), in order to allow other cores to keep their copies as long as possible.

Those access and sharing patterns are known by the developer. It would therefore be interesting to express those patterns in the source code to help the hardware make correct decisions at runtime, rather than just trying to guess what that decision might be. The RISC-V instruction set being open source and extensible, it provides us with an opportunity to study this technique, by adding instructions conveying with what access and sharing patterns a data is being manipulated.

Objectives : The internship is built around three items. First, reviewing the litterature on sharing patterns in multicore programs as well as hardware techniques to identify them will allow the candidate to identify patterns that we would want to express via dedicated instructions. Second, quantifying the frequency at which such patterns occur at runtime in typical multicore workloads (PARSEC [1]) will be needed, in order to confirm the usefulness of such patterns and prioritize which patterns to support. Finally, the candidate will study the performance gain that the introduction of new instructions will bring by i) Adding support for those instructions in gcc (through intrinsic) ii) Adding those instructions in the PARSEC benchmarks and iii) By simulating PARSEC benchmarks on a multicore processor model to which the candidate will have added support for the new RISC-V instructions, both in the processor core and the blocks handling cache coherency. For this last step, a high level simulator will be used (gem5 [2]), and hardware will not be developped using VHDL or Verilog.

Expected Skills :

— Understanding of synchronization concepts for parallel programming (threads, locks...)
— Basics in C++
— Proficiency of (at least) one scripting language (*e.g.*, Bash, Python)
— Understanding of concept of instruction set (knowledge of RISC-V is welcome)
— Basics on coherency protocols are a plus

References

[1] `https://parsec.cs.princeton.edu/parsec3-doc.htm`
[2] `https://www.gem5.org/`

Location and Contact :

The internship takes place at the TIMA Lab, in the center of Grenoble in the historical premises of Grenoble INP, 46, avenue Félix Viallet. **Contacts :** Julie Dumas (Julie.Dumas@univ-grenoble-alpes.fr), Arthur Perais (Arthur.Perais@univ-grenoble-alpes.fr)

2 Introducing New RISC-V Instructions for Memory Accesses in RISC-V

Français

Mots-clés : *RISC-V, compilateur LLVM, microarchitecture en clusters*

Contexte : Pour continuer à augmenter les performances des processeurs modernes, on augmente la taille de la fenêtre d'instructions en vol dans le processeur, c'est à dire le nombre d'instructions du programme étant traitées en parallèle au sein du processeur. Le souci est que certaines scructures qui composent cette fenêtre ne passent pas à l'échelle en terme de latence et de consommation énergétique. Il est donc de plus en plus difficile de continuer à augmenter la performance de cette manière.

Il est cependant possible d'utiliser un processeur divisé en "clusters" d'exécutions qui sont presque indépendants. Ainsi, une fenêtre d'instructions de 200 instructions peut être implémentée avec deux "clusters", chacun pouvant traiter 100 instructions en parallèle. Cela permet d'implémenter des structures matérielles plus petites, mais en double, ce qui pose moins de soucis de latence et de consommation.

Dans cette organisation, chaque instruction doit être envoyée à un "cluster" en particulier, pour qu'elle soit exécutée. Nous étudions pour le moment une microarchitecture à deux "clusters" : Le premier traite les instructions qui participent au calcul d'adresses mémoires, et le second traite les autres instructions. Cela necéssite que le processeur apprenne quelles instructions participent au calcul d'addresse via du matériel dédié, ce qui a un coût.

Dans ce stage, on se propose de faire cette analyse à la compilation et de rajouter notamment des instructions *load ptr* qui indique que la valeur chargée depuis la mémoire. Toute instruction qui dépend d'un *load ptr* manipule une addresse par construction, ce qui rend très facile le choix du "cluster" pour chaque instruction. Une fois l'instruction insérée, on pourra en mesurer l'impact via simulation du processeur en utilisant par exemple le logiciel gem5 [1].

Objectifs : Le stage s'articule en trois temps. Premièrement, il faudra se familiariser avec l'infrastructure LLVM et les concepts utilisés. A noter que l'équipe n'a pas d'expertise particulière avec LLVM, et il faut donc s'attendre à devoir progresser par soi-même. Deuxièmement, il s'agira de modifier LLVM pour insérer automatiquement les nouvelles instructions préalablement définies. Troisièmement, et si le temps le permet, il s'agira d'implémenter le support de ces nouvelles instructions dans gem5 et d'en vérifier le bon fonctionnement d'un point de vue fonctionnel mais aussi d'un point de vue performance du programme simulé.

Compétences attendues/mises en œuvre :

- Bases solides en C++ (LLVM et gem5 sont écrits en C++)
- Maîtrise d' (au moins) un langage de scripting (*e.g.*, Bash, Python)
- Notion de jeu d'instruction (des notions de RISC-V sont bienvenues)
- Notion de ce qu'est un compilateur et des abstractions utilisées (e.g., ASA)
- Forte capacité à apprendre par soi-même et autonomie

English

Keywords : *RISC-V, LLVM compiler, clustered microarchitecture*

Context : To keep improving the performance of modern processors, the number of instructions that can be processed in parallel within the processor keeps increasing. However, some hardware structures within the processor do not scale with this increased parallelism in terms of latency and power consumption. It has therefore become harder and harder to keep increasing performance in this fashion.

Fortunately, it is possible to pursue a design where the processor is divided in several execution clusters that are almost independent. In this fashion, being able to process 200 instructions in the processor can be achieved by having two clusters processing 100 each. This allows implementing smaller hardware structures, which solves the runaway latency and power consumption problem, while still increasing performance because those resources are duplicated (multiple clusters).

In a clustered processor, each instruction must be steered to a cluster to be executed. We are currently studying a microarchitecture with two clusters : The first processes instructions that participate in memory address calculation, and the second processes the rest. This requires that the processor learn which instructions participate to address calculation using dedicated hardware, which has a cost.

In this internship, we propose to perform this analysis at compile time, and to insert *load ptr* instructions that indicate that the data being loaded is an address. Any instruction that depends on a *load ptr* therefore manipulates an address and participates to address calculation. This makes the choice of which cluster to send the instruction to trivial for the hardware. Once this instruction has been inserted, the candidate will study the impact on performance using the gem5 simulator [1].

Objectives : The internship is articulated around three steps. First, the candidate will familiarize themselves with the LLVM compiler framework and the concepts it uses. It should be note that the group has no LLVM expertise and the candidate should be able to progress on their own. Second, LLVM should be modified so that the newly defined instructions are automatically inserted at code generation. Third, and it time permits, the candidate will implement support for these new instructions in gem5 in order to verify their behavior from a functionnal point of view but also from a simulated performance point of view.

Expected Skills :

— Solid foundations in C++ (LLVM and gem5 are written in C++)
— Proficiency of (at least) one scripting language (*e.g.*, Bash, Python)
— Understanding of concept of instruction set (knowledge of RISC-V is welcome)
— Understanding of what a compiler is and the abstractions in use (e.g., AST)
— Strong capabilities in learning by oneself and be autonomous

References

[1] https ://www.gem5.org/

Location and Contact :

The internship takes place at the TIMA Lab, in the center of Grenoble in the historical premises of Grenoble INP, 46, avenue Félix Viallet. **Contacts :** Arthur Perais (Arthur.Perais@univ-grenoble-alpes.fr)

3 Reducing The Cost of Branch Mispredictions with Software-hinted Hardware Predication

Français

Mots-clés : *RISC-V, compilateur LLVM, prédication, simulation de processeur*

Contexte : Les processeurs généralistes utilisent des algorithmes matériels avancés pour prédire la direction des branchements conditionnels avant qu'ils ne soient exécutés, ce qui permet de "prendre de l'avance" dans l'exécution du programme et donc d'augmenter la performance.

Cependant, certains branchements sont difficile à prédire, ce qui pose problème car une mauvaise prédiction de branchement implique de jeter toutes les instructions qui sont rentrées dans le processeur après le branchement mal prédit, car elles sont sur le "mauvais chemin", ce qui à un coût en performance. Une solution est la prédiction logicielle, qui consiste à remplacer le branchement par des instructions prédiquées, i.e., qui calculeront leur résultat uniquement si un prédiquat associé vaut "vrai" lors de leur exécution.

Par exemple, considérons l'exemple ci-dessous. En utilisant des branchements conditionnels, on va tester *cond* puis sauter vers B si *cond* est fausse, et continuer en séquence si *cond* est vraie. En d'autres termes, selon la valeur de *cond*, le processeur exécutera soit A, soit B, mais jamais A et B pour un seul appel de la fonction *foo*.

```
foo:
  if(cond)
    A;
  else
    B;
```

Si on implémente la structure *if-else* avec des instructions prédiquées (instructions spécifiques), il n'y a plus de branchement, et le processeur va donc récupérer les instructions qui correspondent à A et à B à chaque appel de *foo*. Cependant, l'exécution "réelle" des instructions dépend d'un prédicat, ici la valeur de la condition *cond*, de telle sorte qu'à chaque appel de *foo*, soit les instructions de *A.cond* soit les instructions **B.!cond** seront rendues sans effets en fonction de *cond*.

```
foo:
  A.cond;
  B.!cond;
```

La prédiction est avantageuse quand le branchement conditionnel aurait été difficile à prédire par le matériel, car elle n'implique jamais de pénalité de mauvaise prédiction. Cependant, elle implique de récupérer plus d'instructions depuis la mémoire et si le nombre d'instructions dans *A* et *B* est grand, utiliser la prédiction devient contre-productif. De manière générale, si *A* et *B* ont le même nombre d'instructions, alors on aura toujours 50% d'instructions "inutiles" à récupérer depuis la mémoire.

Dans ce stage, on se propose de ne pas faire de prédiction à la compilation, mais d'introduire une nouvelle instruction de branchement qui, dans certains cas d'exécution, indiquera au processeur qu'il n'est pas nécéssaire d'enlever des instructions sur le mauvais chemin même si le prédicteur de branchement a mal prédit, ce qui revient à faire de la prédication dynamique au

sein du processeur, quand l'instruction de branchement indique que c'est possible. Un exemple de cas ou cette technique peut-être bénéfique est le suivant :

```
foo:
  if(cond1 || cond2)
    A;
  B;
```

D'après la sémantique C/C++, si *cond1* est vraie, alors *cond2* n'est pas évaluée (court-circuit du OU logique). Après compilation naïve, cela correspond à ajouter un branchement conditionnel **C1** qui saute "par dessus" l'évaluation de *cond2* si *cond1* est vraie (saute vers A). Si *cond2* est fausse, un autre branchement conditionnel **C2** sautera "par dessus" A pour aller directement vers B.

Lors d'une exécution de *foo*, il est possible que le prédicteur de branchement prédise **C1** comme "non pris", ce qui revient à prédire que *cond1* est fausse. Les instructions évaluant *cond2* vont donc rentrer dans le processeur. Le prédicteur de branchement peut ensuite prédire **C2** comme "non pris", indiquant que *cond2* est vraie et qu'il faut donc rentrer dans le corps du *if* (continuer en séquence dans A).

Imaginons maintenant que *cond1* soit au finale vraie, cest à dire que **C1** a été mal prédit puisqu'il est devrait sauter par dessus l'évaluation de *cond2* vers A). En théorie, le processeur devrait jeter toutes les instructions qui sont entrées le pipeline après **C1**, soit les instructions qui correspondent à l'évaluation de *cond2*, à A et à B. Cependant, comme *cond1* est vraie, A et B sont les instructions correctes, et le souci est uniquement les instructions correspondant à *cond2*. Il serait dommage d'enlever toutes les instructions suivant **C1** car les instructions correspondant à *A* et *B* sont correctes (sous certaines hypothèses).

L'idée est donc de remplacer **C1** par une nouvelle instruction de branchement conditionnel qui indique que si elle est mal prédite ET que le branchement conditionnel suivant (ici, **C2**) est prédit "non pris", alors il suffit d'annuler les instructions correspondant à *cond2*, plutôt que d'enlever *cond2*, A et B du pipeline. Cela revient à dynamiquement prédiquer les instructions de *cond2* uniquement lors de certains exécutions ou le prédicteur de branchement à mal prédit **C1**.

Objectifs : Le stage s'articule en trois temps. Premièrement, il conviendra de se familiariser avec l'instructure de compilation LLVM. A noter que l'équipe n'a pas d'expertise particulière avec LLVM, et il faut donc s'attendre à devoir progresser par soi-même. Deuxièmement, il conviendra de spécifier les cas où introduire la nouvelle instruction de branchement conditionnel sera correcte, suivant certaines hypothèses sur la capacité du matériel à "annuler" les instructions prédiquées dynamiquement, ainsi que d'implémenter l'émission de l'instruction dans ces cas par LLVM. Dans un troisième temps, et si le temps le permet, l'idée serait d'introduire le support de cette instruction soit dans un simulateur fonctionnel (e.g., Spike [1]) ou un simulateur de performance (e.g., gem5 [2]) pour vérifier son comportement et confirmer qu'un gain de performance est obtenu.

Compétences attendues/mises en œuvre :

— Bases solides en C++ (LLVM et gem5 sont écrits en C++)
— Maîtrise d' (au moins) un langage de scripting (*e.g.*, Bash, Python)
— Notion de jeu d'instruction (des notions de RISC-V sont bienvenues)
— Notion de ce qu'est un compilateur et des abstractions utilisées (*e.g.*, ASA)
— Forte capacité à apprendre par soi-même et autonomie

— Notion de base en architecture des processeurs (*e.g.*, pipeline, prédiction de branchement est un plus)

English

Keywords : *RISC-V, LLVM compiler, predication, processor simulation*

Context : General purpose processors use advanced hardware algorithm to predict the direction of conditionnal branches before they are executed. This allows the processor to "runahead" in the program, increasing performance.

However, some branches are hard to predict, which is problematic because it requires all wrong path instructions that have entered the processor to be thrown away. This costs performance. A solution is software predication, who replaces a conditional branch by predicated instructions, that is instructions that will produce their result only if the associated predicate is true during execution.

For instance, consider the below example. Using conditional branches, we will test *cond* then jump to B if *cond* is false, or sequence towards A if *cond* is true. In other terms, depending on the value of *cond*, the processor will execute either A or B, but never both for a single call to function *foo*.

```
foo:
  if(cond)
    A;
  else
    B;
```

Si the *if-else* construct is implemented with predicted instructions (specific instructions), then there are no conditional branches anymore, and the processor will fetch instructions corresponding to A and B every time *foo* is called. However, the actual execution of the instructions depend on the predicted, which is the value of *cond* in this case. As a result, each time *foo* is called, either the instructions of *A.cond* or *B. !cond* will be "annuled" or rendered "transparent", even if both *A.cond* and *B. !cond* will have entered the processor.

```
foo:
  A.cond;
  B.!cond;
```

Predication is advantageous if the conditional branch being replaced would have been hard to predict by the hardware branch predictor, because it never suffers from branch mispredictions. However, it implies fetching more instructions every time (in the example, both *if* and *else* are always fetched), and if the number of instructions in the predicated paths is large, it becomes counter productive. Generally speaking, if *A* and **B** have the same number of instructions, then the processor will always fetch 50% of "useless" instructions with predication.

In this internship, we propose to perform prediction at runtime, within the processor, with a little help from the compiler. The idea is to introduce a new conditional instruction who, in some cases, will indicate to the processor that if it is mispredicted, and under some circumstances, it is not necessary to remove wrong path instructions from the pipeline, but it is necessary to "annul" them. This is equivalent to dynamically predicating those wrong path instructions. An example where this would be beneficial is :

```
foo:
  if(cond1 || cond2)
    A;
  B;
```

According to C/C++ semantics, if *cond1* is true, then *cond2* is not evaluated (logical OR short-circuit). Assuming naive compilation, this corresponds to adding a conditiona branch **C1** that jumps overs *cond2* to A if *cond1* is true. If *cond2* is false, another conditional branch **C2** will jump over A to go to B (the if body is not entered).

During one execution of *foo*, it is possible that the branch predictor predicts **C1** as "not taken", meaning *cond1* is false. The instructions corresponding to *cond2* will therefore enter the processor. The branch predictor can then predict **C2** as "not taken", meaning *cond2* is true, and the if body must be entered, hence A and B will enter the processor.

Let us now imagine that *cond1* is ultimately true, meaning **C1** has been mispredicted since it should jump over *cond2* to A. In theory, the processor should remove all instructions younger than the mispredicted **C1** from the pipeline, meaning instructions of *cond2*, but also instructions of A and B. However, since *cond1* is true, A and B are correct path instructions, and the problem is only with the instructions corresponding to *cond2*. It would be wasteful to also remove instructions of A and B just to refetch them again, if we can just keep them.

The idea is then to replace **C1** by a new conditional branch instruction that indicates that if it is mispredicted AND the following conditional branch (here, **C2**) is predicted "not taken", then *cond2* instructions can just be annulled, rather than removing all of *cond2*, A and B. This is equivalent to dynamically predicting the instructions of *cond2*, but only during some executions of function *foo* during which **C1** was mispredicted.

Objectifs : The internship has three steps. First, the candidate will familiarize themselves with the LLVM framework and the concepts it uses. Note that the group does not have expertise with LLVM and the candidate should expect to learn on their own. Second, the candidate will specify in which cases this transformation could be applied, under some hypothesis of what the hardware is capable of annulling vs. not. The candidate will then implement the introduction of the instruction in those cases within LLVM. Finally, it time permits, the candidate will add support for this instruction in a functional (e.g., Spike [1]) or performance (e.g., gem5 [2]) simulator to validate its behavior.

Expected Skills :

— Solid foundations in C++ (LLVM and gem5 are written in C++)
— Proficiency of (at least) one scripting language (*e.g.*, Bash, Python)
— Understanding of concept of instruction set (knowledge of RISC-V is welcome)
— Understanding of what a compiler is and the abstractions in use (e.g., AST)
— Strong capabilities in learning by oneself and be autonomous
— Basics in processor architecture (*e.g.*, pipeline, branch prediction is a plus)

References

[1] `https://github.com/riscv-software-src/riscv-isa-sim`
[2] `https://www.gem5.org/`

Location and Contact :

The internship takes place at the TIMA Lab, in the center of Grenoble in the historical premises of Grenoble INP, 46, avenue Félix Viallet. **Contacts :** Arthur Perais (Arthur.Perais@univ-grenoble-alpes.fr)

4 High-Level Modelling of a High-Performance L1 Data Cache

Français

Mots-Clés : *Mémoire Cache, Processeur Haute-Performance, Modélisation haut-niveau*

Contexte : Les mémoires cache sont des composants critiques dans les processeurs afin de réduire en moyenne la latence d'accès à la mémoire vive. Ces caches exploitent la localité temporelle et spatiale des programmes afin de garder les données et instructions proches de cœurs des processeurs. Lorsque le processeur demande une donnée, le cache vérifie s'il en garde une copie. Si c'est le cas, il répond au processeur immédiatement. Si ce n'est pas le cas, le cache demande la donnée au niveau suivant de la hiérarchie mémoire, qui peut être soit un autre cache soit directement la mémoire vive. Le premier cas est appelé « cache hit », le deuxième « cache miss ». L'objectif est donc d'avoir un « taux de miss » faible (rapport entre le nombre de cache miss et le nombre total d'accès mémoire).

Cependant, le besoin toujours croissant de données par les applications et des motifs d'accès irréguliers diminuent l'efficacité des caches (taux de miss élevé). Afin de pallier à ce problème, la micro-architecture des caches est de plus en plus complexe. En concert avec le processeur, les caches doivent permettre le recouvrement de plusieurs requêtes afin de réduire la latence globale des accès. Les caches de ce type sont nommés caches non-bloquants.

Le HPDcache[1] est un cache de données non-bloquant pour processeurs RISC-V. Il est actuellement utilisé par plusieurs industriels au sein de leurs Systèmes sur Puce (SoC) à base de RISC-V tels que Thales ou Bosch et des instituts de rechercher tels que le CEA, Inria, Barcelona Supercomputing (BSC), ETH-Zurich ou l'University of California Santa Barbara (UCSB). Son code source RTL (écrit en SystemVerilog) est ouvert et libre d'accès dans le Github OpenHW. Ce cache possède une micro-architecture relativement complexe avec l'objectif de permettre le plus possible le recouvrement des requêtes (même en cas de cache miss) pour améliorer les performances des applications.

Objectifs : Ce stage aura comme objectif de réaliser un modèle haut-niveau du HPDcache. Ce modèle permettra de réaliser l'exploration architecturale afin d'évaluer différentes optimisations dans sa micro-architecture de manière plus simple et rapide qu'en modifiant le code SystemVerilog.

Le modèle haut-niveau devra être développé sur la plateforme Gem5[2]. Ceci permettra d'évaluer le HPDcache et ses possibles optimisations avec différents types de cœurs (scalaires, superscalaire et/ou avec exécution dans le désordre « out-of-order ») et sur des systèmes complets (cœurs, hiérarchie mémoire et périphériques) déjà existants dans la plateforme Gem5.

Le stage se déroulera donc en trois étapes :

1. Etude de la micro-architecture du HPDcache. Pour cela, la/le stagiaire bénéficiera de la documentation disponible (assez complète) du HPDcache ainsi que de son code SystemVerilog.
2. Modélisation du HPDcache sur la plateforme Gem5 en C/C++. Au préalable, la/le stagiaire pourra étudier le fonctionnement de cette plateforme de simulation.
3. Évaluation du modèle et comparaison des performances avec le modèle RTL existant (modèle de référence en SystemVerilog). Cela permettra de vérifier que le modèle est fidèle à l'implémentation de référence. Pour cela, la/le stagiaire exécutera différents benchmarks sur la plateforme de simulation.

Compétences attendues/mises en œuvre :

— Notion d'architecture des processeurs (processeurs, pipeline, caches)
— Notion de jeu d'instruction (des notions de RISC-V sont bienvenues)
— Des bases en C++
— Des bases en un langage de description matérielle : SystemVerilog ou VHDL
— Maîtrise d'au moins un langage de scripting (e.g. Bash, Python)

English

Keywords : *Cache Memory, High-Performance, High-Level Modeling*

Context : Cache memories are essential components in processors. They reduce the average access latency to the memory. Caches exploit locality (temporal and spatial) of programs to keep the useful data and instructions near to the processor. When a processor requests a data, the cache checks if it has a copy. If it is the case, it responds immediately to the processor. Otherwise it asks for the requested data to the next level of the memory hierarchy, which is either a cache either the main memory. The first case is called « cache hit », and the second case is called « cache miss ». The objective then is to have a low miss rate (ratio between the number of cache misses and the total number of memory accesses).

However, the continuous increase of data manipulated by applications and irregular memory access patterns limit the efficiency of caches (high miss rate). To deal with this problem, the micro-architecture of high-performance caches is complex. In cooperation with the processor, the caches shall allow the overlapping of multiple requests to reduce the global memory access latency. This kind of caches is called non-blocking caches.

The HPDcache[1] is a non-blocking L1 data cache for RISC-V processors. It is currently used by different companies in their RISC-V based Systems-on-Chip (SoC) sush as Thales or Bosch and research instituts such as CEA, Inria, Barcelona Supercomputing (BSC), ETH-Zurich or l'University of California Santa Barbara (UCSB). Its RTL code (written in SystemVerilog) is open-source and freely accessible through a dedicated OpenHW Github repository. This cache has a relatively complex micro-architecture that aims to enable the overlapping of multiple requests (up to hundreds) even in the case of cache misses to improve the performance.

Objective : This internship aims to develop a high-level model of the HPDcache. This model will enable the architecture exploration to evaluate different optimisations in its micro-architecture in a more efficient way than modifying directly the SystemVerilog code.

The high-level model will be developed in the Gem5[2] simulation platform. This platform will allow to evaluate the HPDcache integrated with different types of processor cores (scalars, superscalar, and/or with Out-of-Order execution). It will also allow to simulate full systems, with processor cores, memory hierarchy and peripherals which are already part of the Gem5 platform.

The internship consists in three stages :

1. Study of the HPDcache's micro-architecture. To do this, the intern will benefit from the available documentation of the HPDcache as well as its SystemVerilog RTL code.
2. High-level modelling of the HPDcache in C/C++ for the Gem5 platform.
3. Evaluation of the model and comparison of its performance with the reference RTL model to check its fidelity. For this, the intern will run available benchmarks on the full simulation system.

Expected skills :

- Understanding of computer architecture (processors, pipeline, caches)
- Understanding of concept of instruction set (knowledge of RISC-V is welcome)
- Basics in C/C++
- Basics in at least one HDL (hardware description language) : SystemVerilog or VHDL
- Proficiency of (at least) one scripting language (*e.g.*, Bash, Python)

References

[1] `https://github.com/openhwgroup/cv-hpdcache`
[2] `https://www.gem5.org/`

Location and Contact :

The internship takes place at the TIMA Lab, in the center of Grenoble in the historical premises of Grenoble INP, 46, avenue Félix Viallet. **Contact :** César Fuguet (Cesar.Fuguet@univ-grenoble-alpes.fr)

5 Integration of a High-Performance L1 Data Cache with a GPU

Français

Mots-Clés : *Mémoire Cache, Processeur Haute-Performance, GPU*

Contexte : Les systèmes sur puce (SoC) sont de plus en plus hétérogènes. Au lieu d'intégrer des cœurs de processeur généralistes identiques, ces systèmes intègrent de tels cœurs avec des accélérateurs (ou processeurs spécialisés). Le but étant d'améliorer l'efficacité énergétique et les performances des systèmes.

Dans le Top500[1] des supercalculateurs, la plupart sont maintenant hétérogènes. Ils contiennent des processeurs généralistes (GPCPUs - General Purpuse CPU) intégrés avec des GPUs (Graphical Processing Unit). Les GPUs initialement réservés pour des tâches graphiques comme leur nom l'indique, sont maintenant amplement utilisés pour du calcul scientifique haute-performance (HPC) et plus récemment pour des applications d'apprentissage machine et d'intelligence artificielle (ML/AI - Machine Learning/Artificial Inteligence).

La hiérarchie mémoire dans les systèmes sur puce joue un rôle très important dans leur efficacité. Cette hiérarchie mémoire commence par les registres des processeurs/accélérateurs, puis un ou plusieurs niveaux de cache, la mémoire vive puis enfin le stockage permanent. L'augmentation en continue des données demandées par les applications ainsi que l'écart des performances entre la mémoire vive et les processeurs/accélérateurs (appelé mur mémoire) requière des mécanismes complexes pour permettre aux processeurs/accélérateurs de pouvoir travailler sans devoir attendre en permanence les données.

Il existe actuellement en libre accès le code source (open-source) des processeurs, caches, GPUs et autres périphériques. Ceci permet de faire la recherche sur ce type des systèmes tout en permettant d'aller jusqu'à la démonstration sur des prototypes FPGA ou silicium (ASIC). Concernant les cœurs de processeur, caches et autres périphériques il y a le catalogue des composants proposé par l'OpenHW Foundation. On y retrouve, notamment le processeur RISC-V CVA6 et son cache de données haute-performance (HPDcache).

Le HPDcache[2] est un cache de données non-bloquant pour processeurs RISC-V. Il est actuellement utilisé par plusieurs industriels au sein de leurs Systèmes sur Puce (SoC) à base de RISC-V tels que Thales ou Bosch et des instituts de rechercher tels que le CEA, Inria, Barcelona Supercomputing (BSC), ETH-Zurich ou l'University of California Santa Barbara (UCSB). Son code source RTL (écrit en SystemVerilog) est ouvert et libre d'accès dans le Github OpenHW. Ce cache possède une micro-architecture relativement complexe avec l'objectif de permettre le plus possible le recouvrement des requêtes (même en cas de cache miss) pour améliorer les performances des applications.

Concernant le GPU, Georgia-Tech a ouvert le code source de Vortex[3]. C'est un GPU s'appuyant sur le jeu d'instructions RISC-V. Il est maintenant également utilisé par plusieurs instituts de recherche nord-américains et européens. Le code source de Vortex est en SystemVerilog mais il existe également un modèle haut-niveau (source ouverte). Par rapport à d'autres alternatives source ouverte existantes, ce GPU supporte le flot de programmation standard OpenCL, permettant ainsi de pouvoir exécuter une panoplie d'applications existantes.

Objectifs : Ce stage aura comme objectif d'intégrer le HPDcache avec Vortex. Le HPDcache est un cache conçu pour être polyvalent. Il possède plusieurs paramètres lui permettant de s'adapter à différents types de processeurs/accélérateurs.

L'hypothèse de ce stage est que le HPDcache améliorera les performances de Vortex et sa hiérarchie de cache actuelle.

Le stage se déroulera donc en plusieurs étapes :

1. Étude de la micro-architecture du HPDcache, Vortex et leur interfaces. Pour cela, la/le stagiaire bénéficiera de la documentation disponible et du code SystemVerilog.
2. Intégration du HPDcache dans Vortex
3. Mise en place d'une plateforme de simulation en SystemVerilog pour valider l'intégration. La/le stagiaire pourra partir des plateformes existantes.
4. Évaluation du système et analyse des résultats.
5. De manière optionnelle, si le temps le permet, la/le stagiaire pourra proposer et réaliser des modifications dans les micro-architectures afin de améliorer les performances si elle/il a identifié des goulots d'étranglement.

Compétences attendues/mises en œuvre :

- Notion d'architecture des processeurs (processeurs, pipeline, caches)
- Notion de jeu d'instruction (des notions de RISC-V sont bienvenues)
- Des bases en un langage de description matérielle : SystemVerilog ou VHDL
- Des bases en langage C/C++ pour la réalisation de tests unitaires
- Maîtrise d'au moins un langage de scripting (e.g. Bash, Python)

English

Keywords : *Cache Memory, High-Performance, GPU*

Context : Modern Systems-on-Chip (SoCs) are heterogeneous. Instead of integrating identical general-purpose processor cores, they integrate such cores with accelerators (application-specific processors). The objective is to improve the energy efficiency and performance of such systems.

In the Top500[1] of supercomputers most of them are heterogeneous. They integrate General-Purpose CPUs (GPCPUs) with Graphical Processing Units (GPUs). GPUs, initially reserved for graphical tasks as implied by their name, are now widely used in scientific high-performance computing (HPC) and machine learning / artificial intelligence (ML/AI) applications.

The memory hierarchy in SoCs plays a major role in their efficiency. This hierarchy starts with the registers inside the cores, then one or more levels of cache, main memory and finally permanent storage. The continuous increase of the data requested by the applications and the gap between the performance of the memory and the one of processors (also known as the memory wall) requires complex mechanismes to allow the processors/accelerators to work without continuously waiting for the data.

There exist open-source solutions for processor cores, caches, GPUs and other peripherals. This enables research on those topics while allowing to build demonstrators on FPGA or silicon (ASIC). Regarding the cores, caches, and other peripherals, there is the OpenHW Foundation open-source components' catalog. This calatog offers in particular the RISC-V CVA6 core and its high-performance L1 data cache (HPDCache).

The HPDcache[2] is a non-blocking L1 data cache for RISC-V processors. It is currently used by different companies in their RISC-V based Systems-on-Chip (SoC) sush as Thales or Bosch and research instituts such as CEA, Inria, Barcelona Supercomputing (BSC), ETH-Zurich or l'University of California Santa Barbara (UCSB). Its RTL code (written in SystemVerilog) is open-source and freely accessible through a dedicated OpenHW Github repository. This cache has

a relatively complex micro-architecture that aims to enable the overlapping of multiple requests (up to hundreds) even in the case of cache misses to improve the performance.

Regarding the GPU, Georgia-Tech open-sourced the Vortex GPU[3], which is a RISC-V based GPU. It is also widely used by different research institutes in North America and Europe. The source code of Vortex is written in SystemVerilog but there is also a high-level model available in open-source. In comparison with other open-source alternatives, this GPU supports the standard OpenCL programming flow, which allows to execute a wide range of existing applications.

Objective : Integrate the HPDcache with Vortex. The HPDcache its designed to be versatile. It has multiple parameters that enable it to adapt to different types of processors/accelerators.

The hypothesis of this internship is that the HPDcache will improve the performance of Vortex and its current cache hierarchy.

The internship consists in multiple stages :

1. Study of the micro-architecture and interfaces of both the HPDcache and Vortex. To do this, the intern will benefit from the available documentation as well as the SystemVerilog RTL code.
2. Integration of the HPDcache with Vortex.
3. Development of a SystemVerilog simulation platform to validate the integration. There exists different simulation platforms from which the intern can build up its work.
4. Evaluation of the system and analysis of the results.
5. Optionally, if the time allows it, the intern can propose and make some modification in the micro-architectures to improve the performance if some bottlenecks are identified.

Expected skills :

— Understanding of computer architecture (processors, pipeline, caches)
— Understanding of concept of instruction set (knowledge of RISC-V is welcome)
— Basics in at least one HDL (hardware description language) : SystemVerilog or VHDL
— Basics in C/C++ to develop unitary tests.
— Proficiency of (at least) one scripting language (*e.g.*, Bash, Python)

References

[1] `https://top500.org/`
[2] `https://github.com/openhwgroup/cv-hpdcache`
[3] `https://vortex.cc.gatech.edu`

Location and Contact :

The internship takes place at the TIMA Lab, in the center of Grenoble in the historical premises of Grenoble INP, 46, avenue Félix Viallet. **Contact :** César Fuguet (Cesar.Fuguet@univ-grenoble-alpes.fr)

6 Hardware implementation of quantized LLM networks on FPGA

Mots-Clés : *Réseaux de neurones, LLM, FPGA*

Contexte : Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix.

Afin de réduire cet impact, on s'intéresse aux réseaux fortement quantifiés, c'est-à-dire aux réseaux où les valeurs en jeu (poids et/ou activations) peuvent être représentées sur un faible nombre de bits. Cela permet de réduire les besoins en quantité de mémoire, en bande passante mémoire, en surface et consommation des opérateurs de calcul. La contrepartie est que la qualité des résultats est souvent moindre qu'avec des valeurs et opérateurs flottants habituels, mais ce point n'est pas étudié directement dans ces travaux.

Les réseaux de neurones à convolution traditionnels ont déjà été profondément étudiés sous l'angle de la forte quantification (jusqu'au ternaire voire binaire), mais l'application aux grandes modèles de lagage (LLM) est toujours assez confidentielle et il n'existe pas de plateforme d'expérimentation.

Objectifs : Le but est de créer une plateforme matérielle s'exécutant notamment sur FPGA, permettant d'expérimenter et de réaliser des preuves de concept dans le domaine de la forte quantification appliquée aux LLMs.

On dispose déjà d'une plateforme permettant de cibler des FPGA Xilinx, appelée NNawaq [1]. Cette plateforme permet de générer et d'exécuter sur carte FPGA des réseaux de neurones à convolution. Il s'agit d'étendre cette plateforme avec le support d'un petit LLM, à titre de démonstration.

Les LLMs apportent des défis supplémentataires par rapport aux réseaux de neurones à convolution "traditionnels". Il y a d'une part des opérations arithmétiques complexes dans leur définition formelle, pour lesquelles une implémentation efficace en matériel doit être créée, voire pour lesquelles une approximation serait souhaitable. D'autre part, la taille des LLM est généralement particulièrement massive, il y aura probalement besoin d'apporter un support pour des cartes FPGA plus grandes que celles aujourd'hui utilisées.

Le stage se déroulera donc en plusieurs étapes :

1. Etude de l'architecture de réseau de neurones dite "transformer", et des grands réseaux basés sur cette architecture (réseaux LLM - Large Language Model).
2. Proposition d'une décomposition en couches de calcul élémentaires permettant une implémentation matérielle efficace. La/le stagiaire utilisera autant que possible les composants déjà existant dans la plateforme NNawaq [1].
3. Implémentation matérielle des couches qui manquent, avec des choix micro-architecturaux et arithmétiques pertinents pour l'efficacité des circuits résultants.
4. Validation par simulation des nouveaux composants, et exécution du réseau entier sur carte FPGA.

Compétences attendues/mises en oeuvre :

— Notions en architecture des réseaux de neurones

— Notions en implémentations matérielles d'opérations arithmétiques

— Notions en architecture des circuits FPGA et de leur flot de conception associé
— Maîtrise d'un langage de description matérielle : VHDL (préférence), Verilog
— Des bases en C / C++ / Python

References :
[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`

Lieu et Contacts :
Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :
Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Adrien Prost-Boucle (adrien.prost-boucle@univ-grenoble-alpes.fr)

7 Support of library ONNX to implement pre-trained neural networks on FPGA

Mots-Clés : *Réseaux de neurones, ONNX, FPGA*

Contexte : Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix.

Afin de réduire cet impact, on étudie les architectures de réseaux qui ont un impact réduit sur les besoins en matériel et en consommation. Pour cela, on travaille avec une plateforme matérielle développée en interne, NNawaq [1], qui permet d'expérimenter des solutions matérielles efficaces en conditions réelles sur cible FPGA. Cela permet d'étudier notamment les problématiques de stockage/compression en mémoire, la quantification des nombres, les compromis en parallélisme et temps de calcul, etc.

Des scripts (en langages TCL) permettent de reconstruire l'architecture de réseaux connus (Lenet5, les réseaux MobileNet et Resnet, etc), mais cela demande un effort spécial pour chaque nouveau réseau, puis une maintenance sur le long terme. On aimerait donc pouvoir importer directement des réseaux pré-entrainés dans le format standard ONNX[2].

Objectifs : Le but est d'ajouter le support du format ONNX[2] au logiciel de la plateforme NNawaq [1]. Ce format aujourd'hui très répandu permet de décrire l'architecture du réseau ainsi que les poids et paramètres pour exécuter des opérations d'inférence.

Le format ONNX ne permet cependant pas de représenter des réseaux fortement quantifiés, notamment en-dessous de 8 bits par poids. Or ce sont précisément les cas fortement quantifiés qui nous intéressent. Il existe des travaux dédiés à cela, dans le contexte du projet QONNX[3] (le Q désignant *quantifié*), qu'il faudra évaluer.

Le stage se déroulera donc en plusieurs étapes :

1. Etude du format ONNX, des bibliothèques logicielles associées, et de la variante QONNX.
2. Intégration de la bibliothèque ONNX / QONNX dans le logiciel NNawaq.
3. Tests en conditions réelles sur carte FPGA avec des réseaux traditionnels connus (Lenet, MobileNet, ResNet, ...).
4. Evaluer le support d'un réseau plus évolué tel que Yolo et étendre le logiciel NNawaq si nécessaire pour le supporter.

Compétences attendues/mises en oeuvre :

— Notions en architecture des réseaux de neurones
— Notions en architectures de circuits numériques et de FPGA
— Maîtrise du langage de script Python
— Maîtrise des langages de programmation C / C++

Références :

[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`
[2] `https://onnx.ai/`
[3] `https://qonnx.readthedocs.io/en/latest/`

Lieu :
Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :
Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Adrien Prost-Boucle (adrien.prost-boucle@univ-grenoble-alpes.fr)

8 Evaluation de solutions PCI Express et intégration dans un générateur de réseaux de neurones

Mots-Clés : *Réseaux de neurones, FPGA, PCIe*

Contexte : Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix. Afin de réduire cet impact, l'équipe SLS s'intéresse aux réseaux fortement quantifiés. Il a été montré qu'en représentant les données manipulées (poids et/ou activations) sur un faible nombre de bits il est possible de conserver une qualité de résultats proche des versions en représentation flottante. Grâce à une forte quantification, on peut réduire drastiquement la quantité de mémoire, la bande passante mémoire, la surface du circuit et sa consommation, tout en augmentant la vitesse des calcul. Le laboratoire TIMA dispose déjà d'un générateur de réseau de neurones à convolution capable de cibler des FPGA Xilinx, appelée NNawaq [1]. Pour révéler son plein potentiel, ce générateur a besoin d'être étendu de manière à exploiter mieux les ressources des FPGA Xilinx et à supporter plus de cartes.

La norme PCI Express est couramment utilisée pour relier un ordinateur hôte et une carte FPGA. Elle permet de fournir la bande passante élevée nécessaire pour évaluer les accélérateurs implantés sur le FPGA sans les ralentir. Notre générateur actuel s'appuie sur le projet open-source RIFFA qui n'est plus maintenu. Cette dépendance limite la portabilité vers de nouvelles cartes.

Objectifs : Le but du stage est d'évaluer des implémentations fournies sur étagère (IP PCIe fournie dans Vitis, LitePCIe fourni dans le projet LiteX [2]), d'adapter les interfaces par rapport aux besoins de notre générateur de circuits et de les tester sur différentes cartes FPGA.

Ces trois étapes se dérouleront séquentiellement durant le stage. Pour chacune, il est attendu un travail de spécification, développement, validation et documentation adéquat.

Pour les validations sur cartes, des cartes VCU128 et VC709 sont disponibles au sein du laboratoire et des cartes Alvéo U200 sont accessible dans un cloud local. Au besoin, le stage peut être étendu en réalisant un support des instances AWS EC2 F1 de Amazon ou en travaillant sur d'autres bus de communication manquant à notre générateur comme une interface Ethernet.

Compétences attendues/mises en oeuvre :

- Notions en architecture des circuits FPGA et de leur flot de conception associé
- Maîtrise d'un langage de description matérielle : VHDL (préférence), Verilog
- Des bases en C

References :

[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`
[2] `https://github.com/enjoy-digital/litex`

Lieu et Contacts :

Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :
Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Adrien Prost-Boucle (adrien.prost-boucle@univ-grenoble-alpes.fr)

9 Evaluation des mémoires HBM et intégration dans un générateur de réseaux de neurones

Mots-Clés : *Réseaux de neurones, FPGA, HBM*

Contexte : Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix. Afin de réduire cet impact, l'équipe SLS s'intéresse aux réseaux fortement quantifiés. Il a été montré qu'en représentant les données manipulées (poids et/ou activations) sur un faible nombre de bits il est possible de conserver une qualité de résultats proche des versions en représentation flottante. Grâce à une forte quantification, on peut réduire drastiquement la quantité de mémoire, la bande passante mémoire, la surface du circuit et sa consommation, tout en augmentant la vitesse des calcul. Le laboratoire TIMA dispose déjà d'un générateur de réseau de neurones à convolution capable de cibler des FPGA Xilinx, appelée NNawaq [1]. Pour révéler son plein potentiel, ce générateur a besoin d'être étendu de manière à exploiter mieux les ressources des FPGA Xilinx et à supporter plus de cartes.

Les mémoires HBM ont été introduites dans les FPGA Xilinx depuis l'apparition des architectures Ultrascale+. Ces mémoires de 8Go sont intégrées au plus près du FPGA en offrant à la fois un large débit et une latence réduite. Un tel volume de mémoire permet de changer radicalement les circuits implémentables sur les FPGA les possédant. Vu la taille des mémoires dans les circuits de réseaux de neurones, c'est un ajout primordial pour améliorer l'efficacité de ces circuits.

Objectifs : Le but du stage est de caractériser au mieux les mémoires HBM (latence, débit, fragmentation), d'identifier les mémoires d'un réseau de neurones bénéficiant le plus d'une implantation en HBM et d'intégrer le support automatique des mémoires HBM dans le générateur NNawaq.

Ces trois étapes se dérouleront séquentiellement durant le stage. Pour chacune, il est attendu un travail de spécification, développement, validation et documentation adéquat. Les validations sur cartes pourront être réalisées sur des cartes telles que Xilinx VCU128 et VC709 qui sont disponibles dans le laboratoire.

Compétences attendues/mises en oeuvre :
- Notions en architecture des circuits FPGA et de leur flot de conception associé
- Maîtrise d'un langage de description matérielle : VHDL (préférence), Verilog
- Des bases en C

References :
[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`

Lieu et Contacts :
Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :
Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Adrien Prost-Boucle (adrien.prost-boucle@univ-grenoble-alpes.fr)

10 Evaluation de solutions Ethernet et intégration dans un générateur de réseaux de neurones

Mots-Clés : *Réseaux de neurones, FPGA, Ethernet*

Contexte : Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix. Afin de réduire cet impact, l'équipe SLS s'intéresse aux réseaux fortement quantifiés. Il a été montré qu'en représentant les données manipulées (poids et/ou activations) sur un faible nombre de bits il est possible de conserver une qualité de résultats proche des versions en représentation flottante. Grâce à une forte quantification, on peut réduire drastiquement la quantité de mémoire, la bande passante mémoire, la surface du circuit et sa consommation, tout en augmentant la vitesse des calcul. Le laboratoire TIMA dispose déjà d'un générateur de réseau de neurones à convolution capable de cibler des FPGA Xilinx, appelée NNawaq [1]. Pour révéler son plein potentiel, ce générateur a besoin d'être étendu de manière à exploiter mieux les ressources des FPGA Xilinx et à supporter plus de cartes.

L'Ethernet est couramment utilisée pour relier un ordinateur hôte et une carte FPGA. Elle permet de fournir la bande passante élevée nécessaire pour évaluer les accélérateurs implantés sur le FPGA sans les ralentir. Notre générateur actuel n'a qu'une connexion PCI Express, ce qui limite sa portabilité sur d'autres plateformes.

Objectifs : Le but du stage est d'évaluer des implémentations fournies sur étagère (IP Ethernet fournie dans Vitis, LiteEth fourni dans le projet LiteX [2]), d'adapter les interfaces par rapport aux besoins de notre générateur de circuits et de les tester sur différentes cartes FPGA.

Ces trois étapes se dérouleront séquentiellement durant le stage. Pour chacune, il est attendu un travail de spécification, développement, validation et documentation adéquat.

Pour les validations sur cartes, des cartes VCU128 et VC709 sont disponibles au sein du laboratoire et des cartes Alvéo U200 sont accessible dans un cloud local.

Compétences attendues/mises en oeuvre :

- Notions en architecture des circuits FPGA et de leur flot de conception associé
- Maîtrise d'un langage de description matérielle : VHDL (préférence), Verilog
- Des bases en C

References :

[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`
[2] `https://github.com/enjoy-digital/litex`

Lieu et Contacts :

Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :

Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Adrien Prost-Boucle (adrien.prost-boucle@univ-grenoble-alpes.fr)

11 Analyse de l'impact de la réorganisation des kernels sur la compressibilité des Réseaux Neuronaux Convolutifs (CNN)

À mesure que l'IA continue à évoluer et être de plus en plus utilisée, la nécessité de systèmes de calcul plus efficaces et durables devient de plus en plus pressante. Un domaine qui présente un potentiel de réduction de la consommation d'énergie est la compression des données, en particulier dans les réseaux neuronaux. Ce projet de stage propose de mener une analyse sur l'impact de la réorganisation des kernels sur la compressibilité des poids des réseaux neuronaux convolutifs.

11.1 Contexte

Les réseaux neuronaux convolutifs sont une classe de modèles d'apprentissage profond largement utilisés dans la vision par ordinateur et dans d'autres applications. Cependant, ces réseaux nécessitent des ressources de calcul et une énergie importantes pour être entraînés et déployés. Les accès à la mémoire dominent les coûts de fonctionnement des inférences sur les CNNs, ce qui en fait une étape critique vers une réduction de la consommation d'énergie.

11.2 Objectifs

Les objectifs de ce projet de stage sont triples :

- Comprendre l'architecture des CNNs.
- Conception et mise en œuvre d'un framework capable de réorganiser les noyaux d'un CNN pour introduire de la redondance dans l'organisation des données.
- Mener une analyse sur l'impact de la réorganisation des noyaux sur la compressibilité des poids des CNNs.

11.3 Compétences requises

Pour atteindre ces objectifs, le stagiaire doit disposer des compétences suivantes :

- Maîtrise de la programmation Python
- Connaissance de base des réseaux neuronaux, y compris les architectures des CNNs et les concepts d'apprentissage profond
- Familiarité avec la théorie de l'information et ses applications dans la compression des données est un plus
- Connaissance de PyTorch ou d'autres frameworks d'apprentissage profond est un plus

Lieu et Contacts :
Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :
Olivier Romane (olivier.romane@univ-grenoble-alpes.fr)
Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)

12 Analyse non-linéaire des poids des Réseaux Neuronaux Convolutifs (CNN) dans le but d'améliorer leur compressivité

12.1 Contexte

Les applications qui utilisent des technologies à base d'intelligence artificielle sont en augmentation extrêmement forte. C'est la source de problématiques multiples telles que la fabrication des composants de calcul performants en grande quantité, leur consommation énergétique et leur prix. Afin de réduire cet impact, l'équipe SLS s'intéresse aux réseaux fortement quantifiés. Il a été montré qu'en représentant les données manipulées (poids et/ou activations) sur un faible nombre de bits il est possible de conserver une qualité de résultats proche des versions en représentation flottante. Grâce à une forte quantification, on peut réduire drastiquement la quantité de mémoire, la bande passante mémoire, la surface du circuit et sa consommation, tout en augmentant la vitesse des calcul. Le laboratoire TIMA dispose déjà d'un générateur de réseau de neurones à convolution capable de cibler des FPGA Xilinx, appelée NNawaq [1]. Dans nos expérimentations actuelles, les mémoires contenant les poids du réseau restent un goulot d'étranglement.

Pour aller plus loin, il est impératif de compresser cette information le plus possible. Au delà de la compression entropique, l'état de l'art nous indique qu'il existe des redondances et des relations cachées dans ce jeu de données. Des analyses linéaire de type PCA ont déjà montré leur utilité pour réduire les dimensions sur ce problème avec beaucoup de dimension. Cependant un certain nombre de structures complexes non-linéaires ne sont pas captées par ces analyses.

12.2 Objectifs

Les objectifs de ce projet de stage sont :

— Analyser les poids de réseaux de CNN pré-entrainés (type Resnet) avec techniques d'analyse linéaire (PCA) et non-linéaire (UMAP)
— Identifier les réductions de dimensionnalité applicables (linéaire ou non) et analyser leur exploitabilité dans l'optique d'une compression ah-hoc des données
— Evaluer l'impact algorithmique et matérielle des compressions retenues

12.3 Compétences requises

Pour atteindre ces objectifs, le stagiaire doit disposer des compétences suivantes :

— Maîtrise de la programmation Python
— Connaissance de base des réseaux neuronaux, y compris les architectures des CNNs et les concepts d'apprentissage profond
— Familiarité avec la théorie de l'information et ses applications dans la compression des données est un plus
— Connaissance de PyTorch ou d'autres frameworks d'apprentissage profond est un plus

References :

[1] `https://gricad-gitlab.univ-grenoble-alpes.fr/prostboa/nnawaq`

Lieu et Contacts :

Le stage se déroulera au Laboratoire TIMA, dans les locaux historiques de Grenoble-INP, au 46 avenue Félix Viallet, Grenoble.

Contacts :

Olivier Muller (olivier.muller@univ-grenoble-alpes.fr)
Olivier Romane (olivier.romane@univ-grenoble-alpes.fr)