Synthèse de communications pour accélérateurs matériels

Jean-Michel Gorius[1]

Univ Rennes[1]

Stage effectué sous la direction de Steven Derrien et Tomofumi Yuki,
Univ Rennes, IRISA, CNRS, Inria

5 septembre 2018

Clouds

- Plateformes de calcul omniprésentes
- Élargissement de plus en plus important de l'offre en terme de calcul dans les *clouds*
- Essor de l'utilisation d'accélérateurs matériels [3]

Clouds

- Plateformes de calcul omniprésentes
- Élargissement de plus en plus important de l'offre en terme de calcul dans les *clouds*
- Essor de l'utilisation d'accélérateurs matériels [3]

Course aux performances

Top500 : Summit, Sunway TaihuLight, Sierra, etc.

Clouds

- Plateformes de calcul omniprésentes
- Élargissement de plus en plus important de l'offre en terme de calcul dans les *clouds*
- Essor de l'utilisation d'accélérateurs matériels [3]

Course aux performances

Top500 : Summit, Sunway TaihuLight, Sierra, etc.

Consommation massive d'énergie

Recherche de solution pour réduire la consommation énergétique des *datacenters*.

Accélérateurs matériels

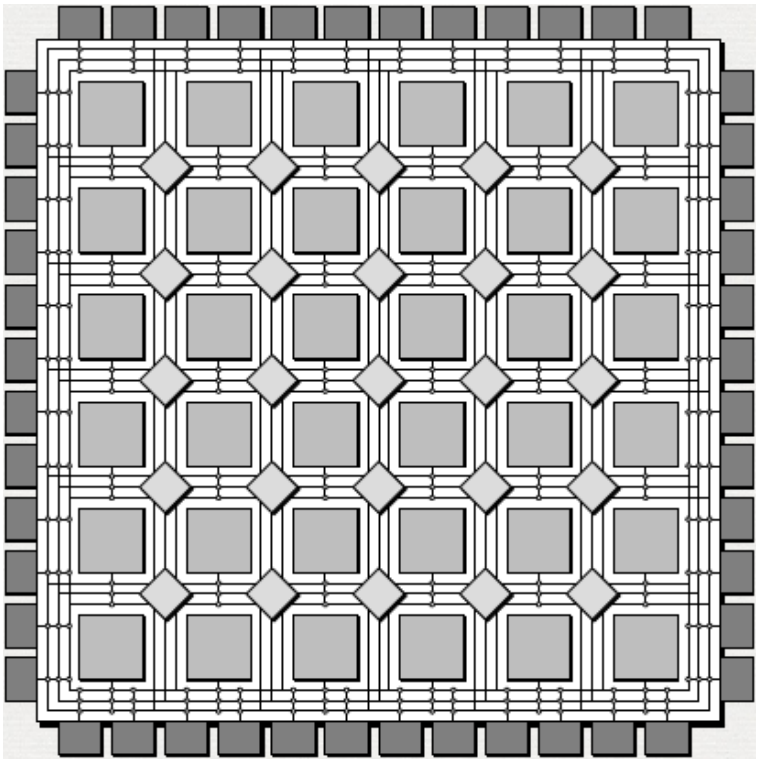
FPGA

Field Programmable Gate Array : cellules de calcul reliées par des routes reconfigurables.

Accélérateurs matériels

FPGA

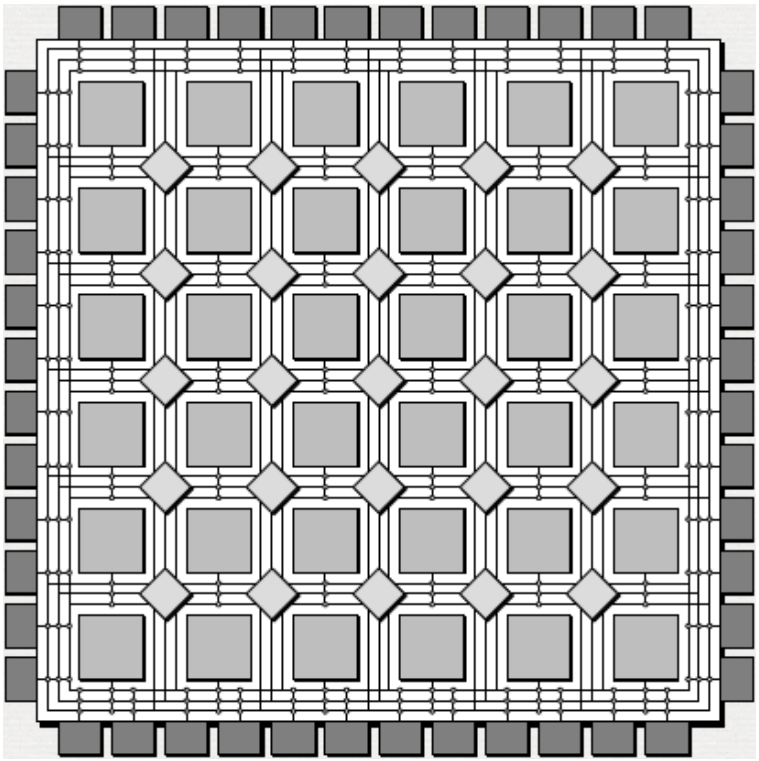
Field Programmable Gate Array : cellules de calcul reliées par des routes reconfigurables.



Accélérateurs matériels

FPGA

Field Programmable Gate Array : cellules de calcul reliées par des routes reconfigurables.



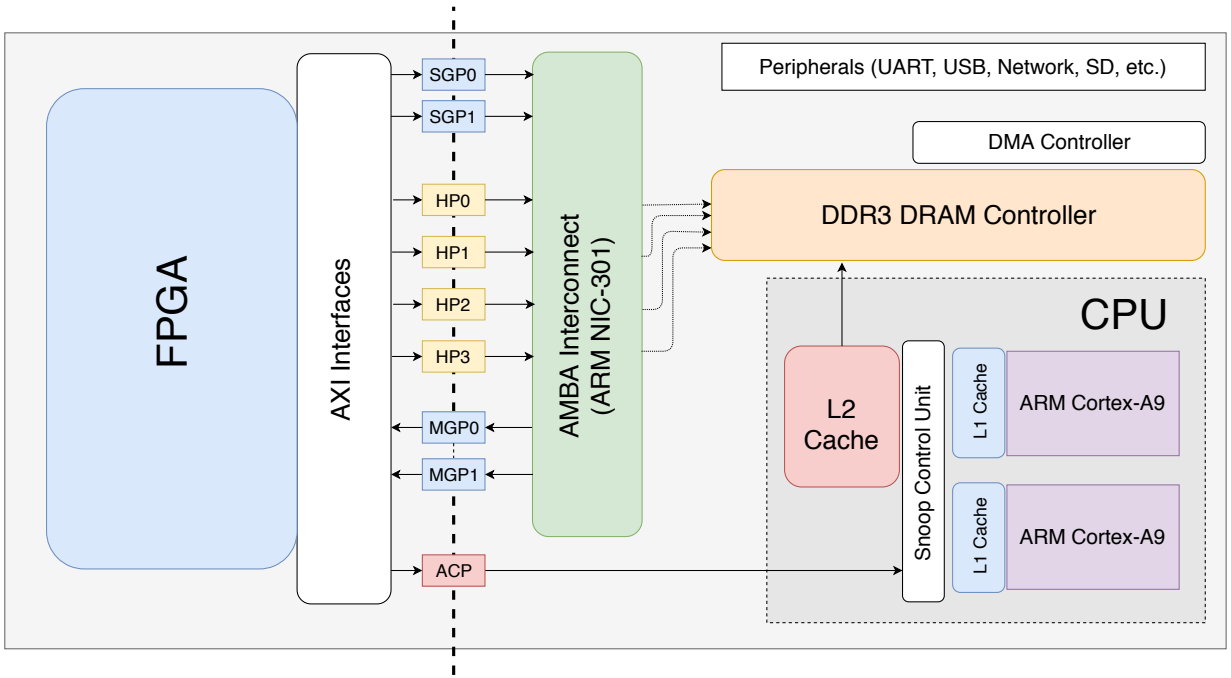
Avantages

- Faible consommation énergétique
- Spécialisation du matériel
- Utilisation optimale des ressources de calcul

System on Chip (SoC)

SoCs et MPSoCs

- Composants hétérogènes sur la même puce
- Partage des ressources
- Un ou plusieurs cœurs de processeur associés à un FPGA



VHDL

VHSIC Hardware Description Language

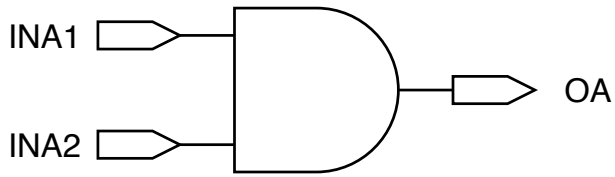
Langage de description matérielle au niveau des portes logiques.

VHDL

VHSIC Hardware Description Language

Langage de description matérielle au niveau des portes logiques.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_or_top is
    Port ( INA1 : in  STD_LOGIC;
           INA2 : in  STD_LOGIC;
           OA   : out STD_LOGIC; )
end and_or_top;
architecture Behavioral of and_or_top is
begin
    OA <= INA1 and INA2;
end Behavioral;
```

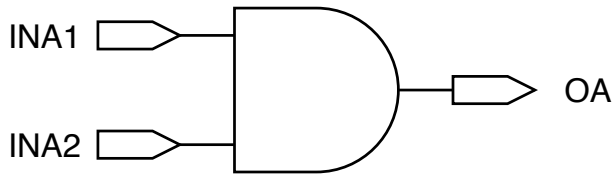


VHDL

VHSIC Hardware Description Language

Langage de description matérielle au niveau des portes logiques.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_or_top is
    Port ( INA1 : in  STD_LOGIC;
           INA2 : in  STD_LOGIC;
           OA   : out STD_LOGIC; )
end and_or_top;
architecture Behavioral of and_or_top is
begin
    OA <= INA1 and INA2;
end Behavioral;
```



Conception complexe

Peut conduire à de nombreuses erreurs. Parfois trop bas niveau.

High Level Synthesis (HLS) (1/3)

Synthèse de haut niveau

- Point de vue haut niveau de la conception de circuit
- Génération d'une description matérielle à partir de spécifications algorithmiques

High Level Synthesis (HLS) (1/3)

Synthèse de haut niveau

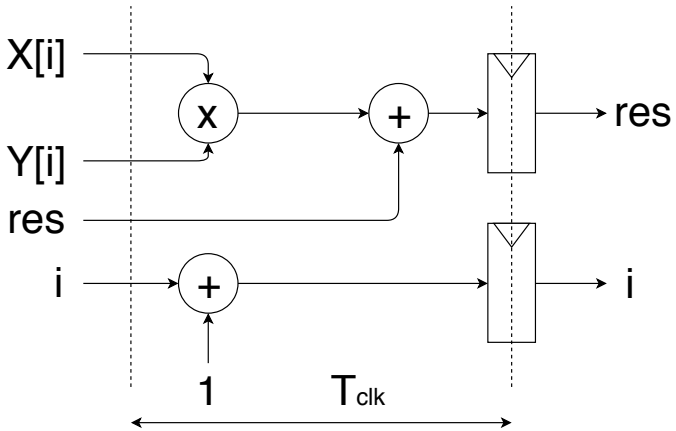
- Point de vue haut niveau de la conception de circuit
- Génération d'une description matérielle à partir de spécifications algorithmiques

Exemple : Produit scalaire de deux vecteurs

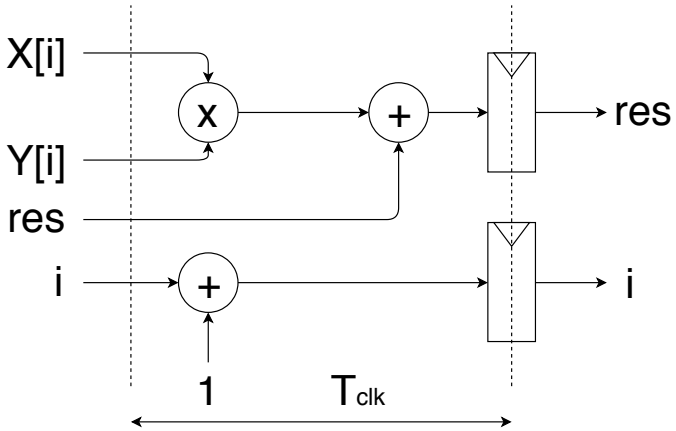
Plusieurs sorties de synthèse possibles.

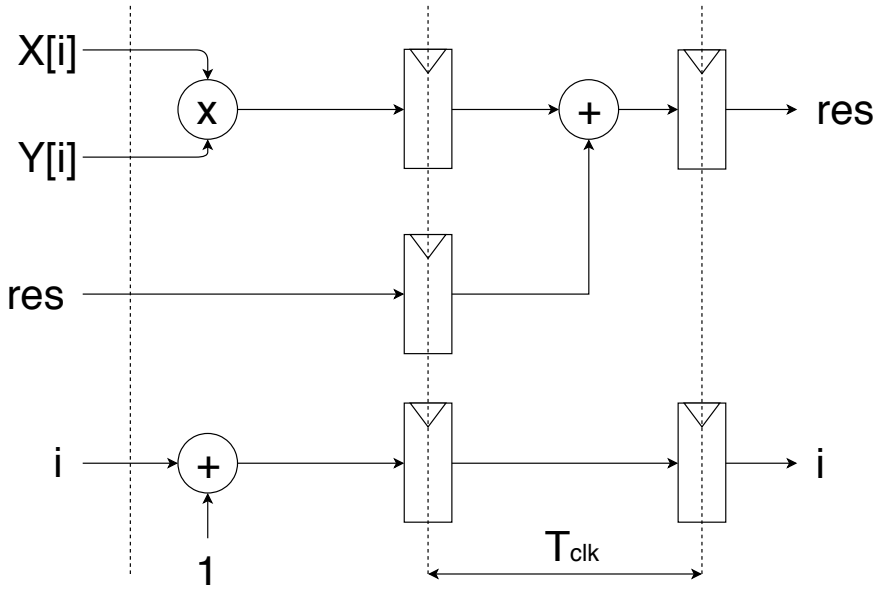
```
1  int X[512], Y[512];
2  int tmp, res = 0;
3  for(int i = 0; i < 512; ++i) {
4      tmp = X[i]*Y[i];
5      res += tmp;
6  }
```

High Level Synthesis (HLS) (2/3)



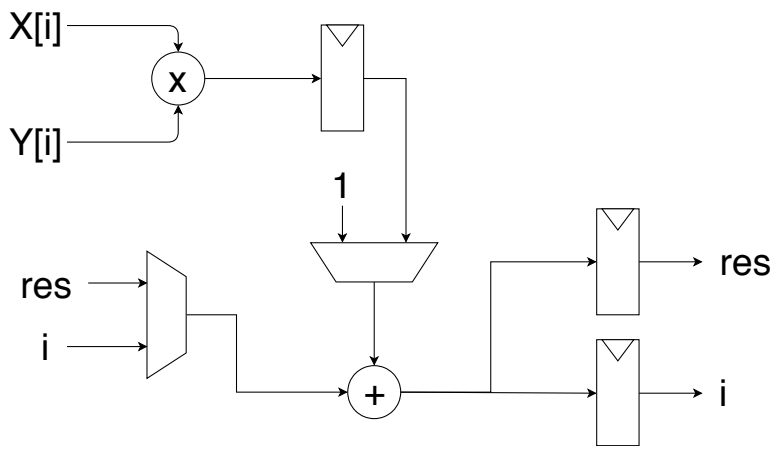
High Level Synthesis (HLS) (2/3)





High Level Synthesis (HLS) (3/3)

```
1  int X[512], Y[512];
2  int tmp, res = 0;
3  #pragma HLS mult=1, adder=1
4  for(int i = 0; i < 512; ++i) {
5      tmp = X[i] * Y[i];
6      res += tmp;
7  }
```



Programmation de SoCs

Compilation d'un programme ciblant un SoC

- Génération de la partie matérielle à implanter sur FPGA
- Compilation de la partie *software* à exécuter sur le processeur généraliste
- Mise en place des interfaces de communications FPGA/CPU et FPGA/périphériques

Programmation de SoCs

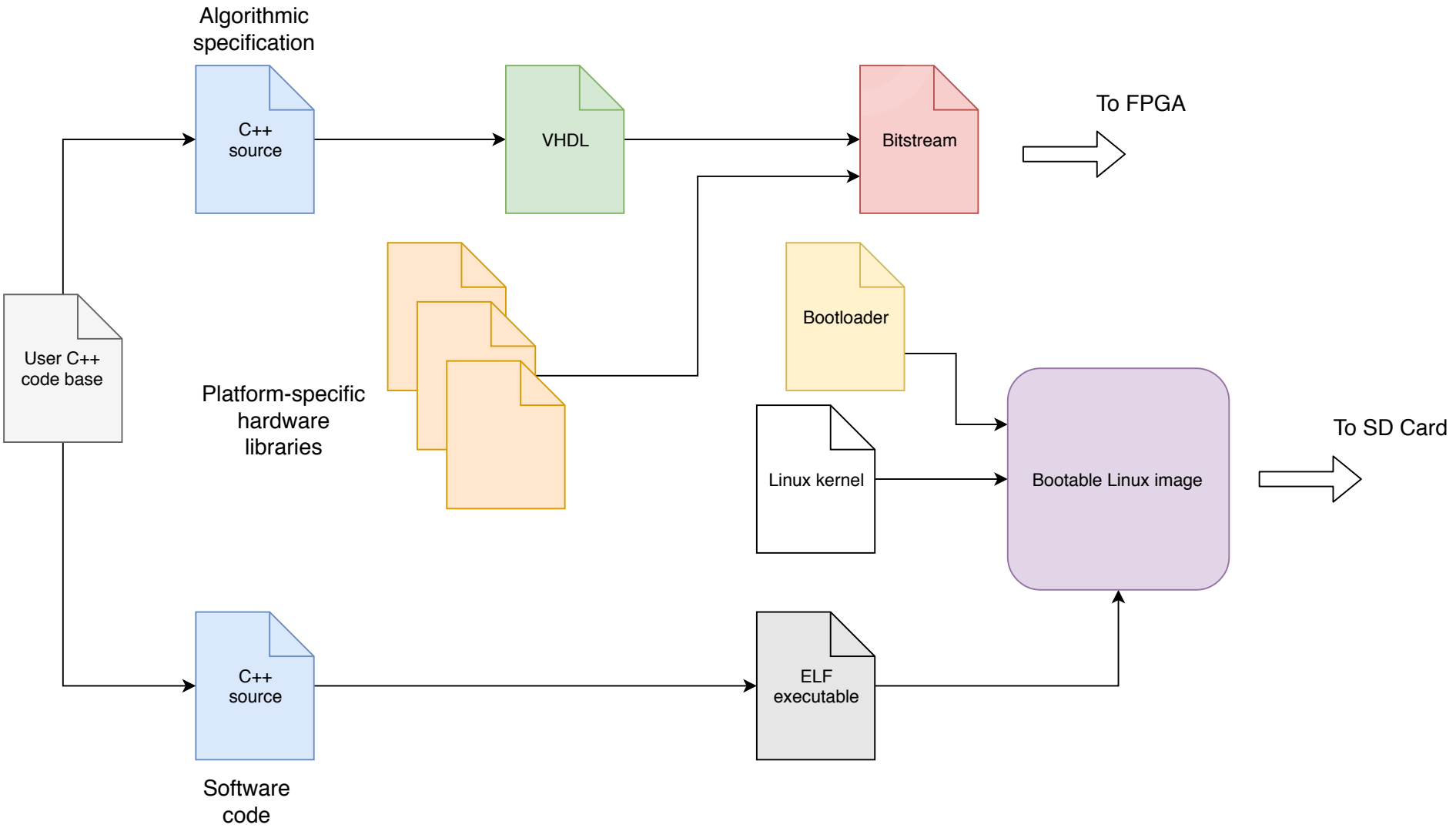
Compilation d'un programme ciblant un SoC

- Génération de la partie matérielle à implanter sur FPGA
- Compilation de la partie *software* à exécuter sur le processeur généraliste
- Mise en place des interfaces de communications FPGA/CPU et FPGA/périphériques

SDSoC

Outil développé par Xilinx, intègre des outils de HLS et génère les interfaces de communication au niveau *hardware*.

SDSoC


Algorithmic specification
C++ source
VHDL
Bitstream
To FPGA
User C++ code base
Platform-specific hardware libraries
Bootloader
Linux kernel
Bootable Linux image
To SD Card
C++ source
ELF executable
Software code

Cohérence des données

Partage des calculs

L'utilisation de SoCs/MPSoCs permet de partager les calculs entre le CPU et l'accélérateur matériel implanté sur FPGA.

Cohérence des données

Partage des calculs

L'utilisation de SoCs/MPSoCs permet de partager les calculs entre le CPU et l'accélérateur matériel implanté sur FPGA.

Problème de la cohérence

La distribution des calculs entre CPU et FPGA peut conduire à des conflits lors des accès à la mémoire ! Comment résoudre ce problème ?

Cohérence des données

Partage des calculs

L'utilisation de SoCs/MPSoCs permet de partager les calculs entre le CPU et l'accélérateur matériel implanté sur FPGA.

Problème de la cohérence

La distribution des calculs entre CPU et FPGA peut conduire à des conflits lors des accès à la mémoire ! Comment résoudre ce problème ?

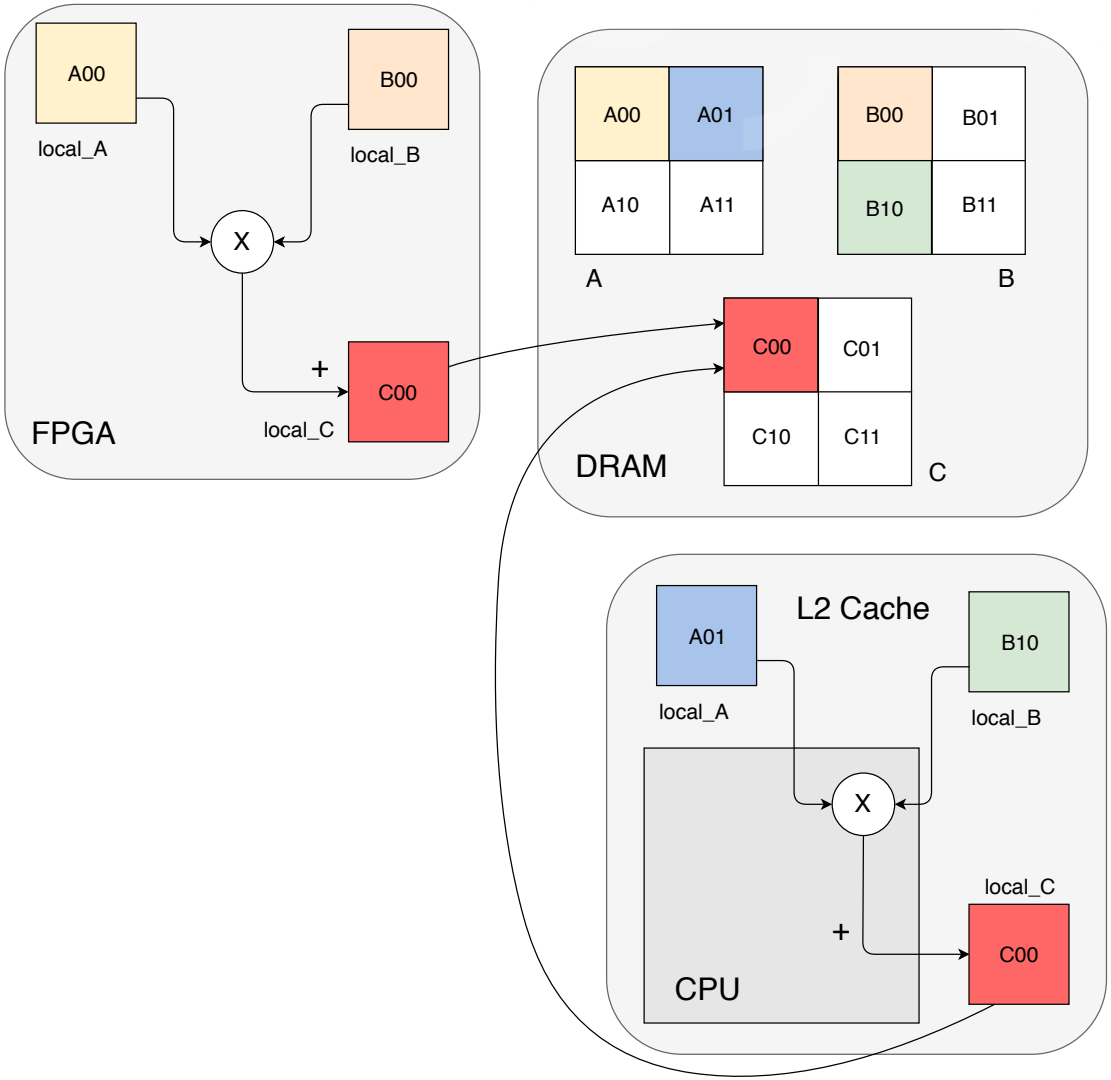
Une solution ?

Désactiver le cache du processeur évite les problèmes de cohérence mais a un impact extrême sur les performances.

Exemple : Multiplication de matrices (1/2)

```
1  void matmul_block(/* ... */) {
2      const int M = BLOCK_SIZE;
3      static int block0[M * M] = { 0 };
4      static int block1[M * M] = { 0 };
5      static int blockOut[M * M] = { 0 };
6      /* Sélection des ports d'entrée et copie des données dans la mémoire locale. */
7      for(int i = 0; i < M; ++i) {
8          for(int j = 0; j < M; ++j) {
9              int sum = 0;
10             for(int k = 0; k < M; ++k)
11 #pragma HLS pipeline
12                 sum += block0[i * M + k] * block1[k * M + j];
13             blockOut[i * M + j] += sum;
14         }
15     }
16     /* Sélection du port de sortie et copie des données en mémoire externe. */
17 }
```

Exemple : Multiplication de matrices (2/2)



Résoudre le problème de la cohérence

Détermination des zones mémoires partagées

Par une analyse des dépendances *Read-after-Write* (RaW) entre les données manipulées par le CPU et le FPGA.

Résoudre le problème de la cohérence

Détermination des zones mémoires partagées

Par une analyse des dépendances *Read-after-Write* (RaW) entre les données manipulées par le CPU et le FPGA.

Réduction de la classe de programmes considérés

On se limite aux programmes manipulant les données dans des boucles imbriquées.

Résoudre le problème de la cohérence

Détermination des zones mémoires partagées

Par une analyse des dépendances *Read-after-Write* (RaW) entre les données manipulées par le CPU et le FPGA.

Réduction de la classe de programmes considérés

On se limite aux programmes manipulant les données dans des boucles imbriquées.

Compilation polyédrique

Représentation mathématique des boucles, permet de déterminer les dépendances RaW entre les itérations successives et de paralléliser automatiquement les programmes [1].

Compilation polyédrique : un exemple [2]

```
1  for(int t1 = 0; t1 < tsteps - 1; ++t1)
2      for(int t2 = 0; t2 <= n - 1; ++t2)
3          for(int t3 = 0; t3 <= n - 1; ++t3)
4          S:  A[t2][t3] = 1 + A[t2+1][t3];
```

Compilation polyédrique : un exemple [2]

```
1  for(int t1 = 0; t1 < tsteps - 1; ++t1)
2      for(int t2 = 0; t2 <= n - 1; ++t2)
3          for(int t3 = 0; t3 <= n - 1; ++t3)
4          S:  A[t2][t3] = 1 + A[t2+1][t3];
```

Dépendances RaW

$$\mathcal{D}_{\text{flow}} = \{S[t_1, t_2, t_3] \mapsto S[t'_1 = t_1 + 1, t'_2 = t_2 - 1, t'_3 = t_3] : \\ 0 \leqslant t1 \leqslant \text{tsteps} - 2 \wedge 1 \leqslant t2 \leqslant n - 1 \wedge 1 \leqslant t3 \leqslant n - 1\}$$

Compilation polyédrique : un exemple [2]

```
1  for(int t1 = 0; t1 < tsteps - 1; ++t1)
2      for(int t2 = 0; t2 <= n - 1; ++t2)
3          for(int t3 = 0; t3 <= n - 1; ++t3)
4          S:  A[t2][t3] = 1 + A[t2+1][t3];
```

Dépendances RaW

$$\mathcal{D}_{\text{flow}} = \{S[t_1, t_2, t_3] \mapsto S[t'_1 = t_1 + 1, t'_2 = t_2 - 1, t'_3 = t_3] : \\ 0 \leqslant t1 \leqslant \text{tsteps} - 2 \wedge 1 \leqslant t2 \leqslant n - 1 \wedge 1 \leqslant t3 \leqslant n - 1\}$$

Gestion fine du cache

La connaissance des dépendances permet d'insérer automatiquement des instructions d'éviction des lignes de cache concernées par un conflit RaW.

Génération d'instructions de gestion du cache

Contribution

- Exploration de la base de code Xilinx
- Développement d'utilitaires de *micro-benchmarking*
- Mise au point d'un algorithme de gestion fine du cache pour une exécution CPU/FPGA

Génération d'instructions de gestion du cache

Contribution

- Exploration de la base de code Xilinx
- Développement d'utilitaires de *micro-benchmarking*
- Mise au point d'un algorithme de gestion fine du cache pour une exécution CPU/FPGA

Expérimentations

- *Micro-benchmarking* pour déterminer la bande passante des interfaces de communication processeur/accélérateur
- Tentative d'implantation de notre algorithme

Limite des outils

Allocation mémoire SDSoC

Politique de gestion de la cohérence des données extrêmement conservative : insertion d'une éviction complète du cache avant chaque accès à une zone potentiellement partagée.

Limite des outils

Allocation mémoire SDSoC

Politique de gestion de la cohérence des données extrêmement conservative : insertion d'une éviction complète du cache avant chaque accès à une zone potentiellement partagée.

Implantation de notre algorithme

Nécessite de contourner les mécanismes de génération de SDSoC pour gérer finement le cache.

Conclusion

Synthèse

- Partager les calculs entre CPU et FPGA amène à des problèmes de cohérence
- La compilation polyédrique fournit des outils permettant de gérer finement le cache CPU pour éviter ces problèmes
- Les outils actuels ont des limites importantes qu'il faut contourner

Future work

Intégration de notre algorithme dans un flot de conception de circuit, mise en place de compression pour améliorer le débit des échanges entre CPU et FPGA.

Références


Uday Bondhugula et al. « A Practical Automatic Polyhedral Parallelizer and Locality Optimizer ». In : *SIGPLAN Not.* 43.6 (juin 2008), p. 101–113. ISSN : 0362-1340.


Sanket Tavarageri et al. *Automatic Generation of Coherence Instructions for Software-Managed Multiprocessor Caches*. Rapp. tech. OSU-CISRC-1/14-TR03. 2014.

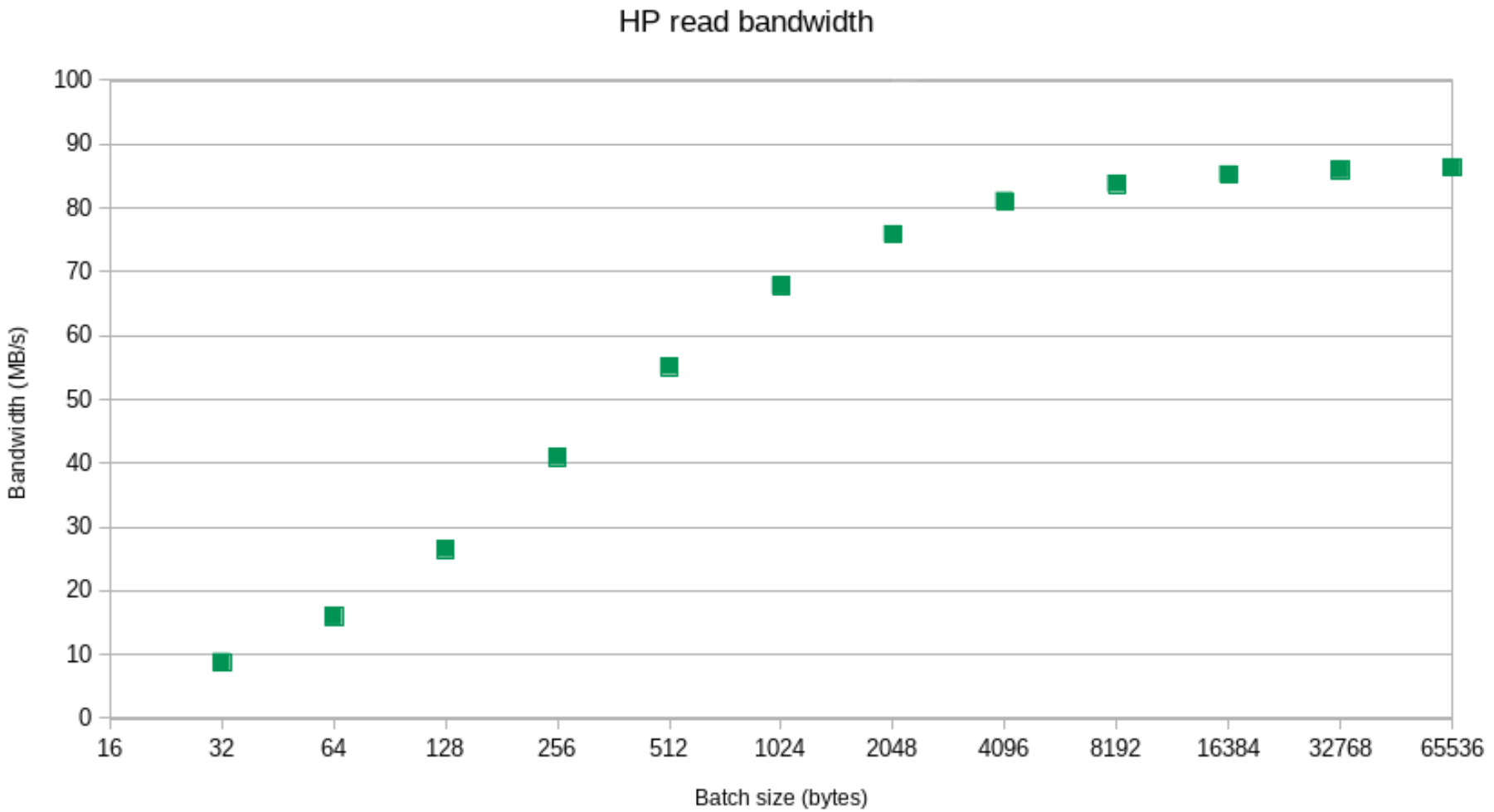

Chen Zhang et al. « Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks ». In : *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA : ACM, 2015, p. 161–170. ISBN : 978-1-4503-3315-3.

Notre algorithme

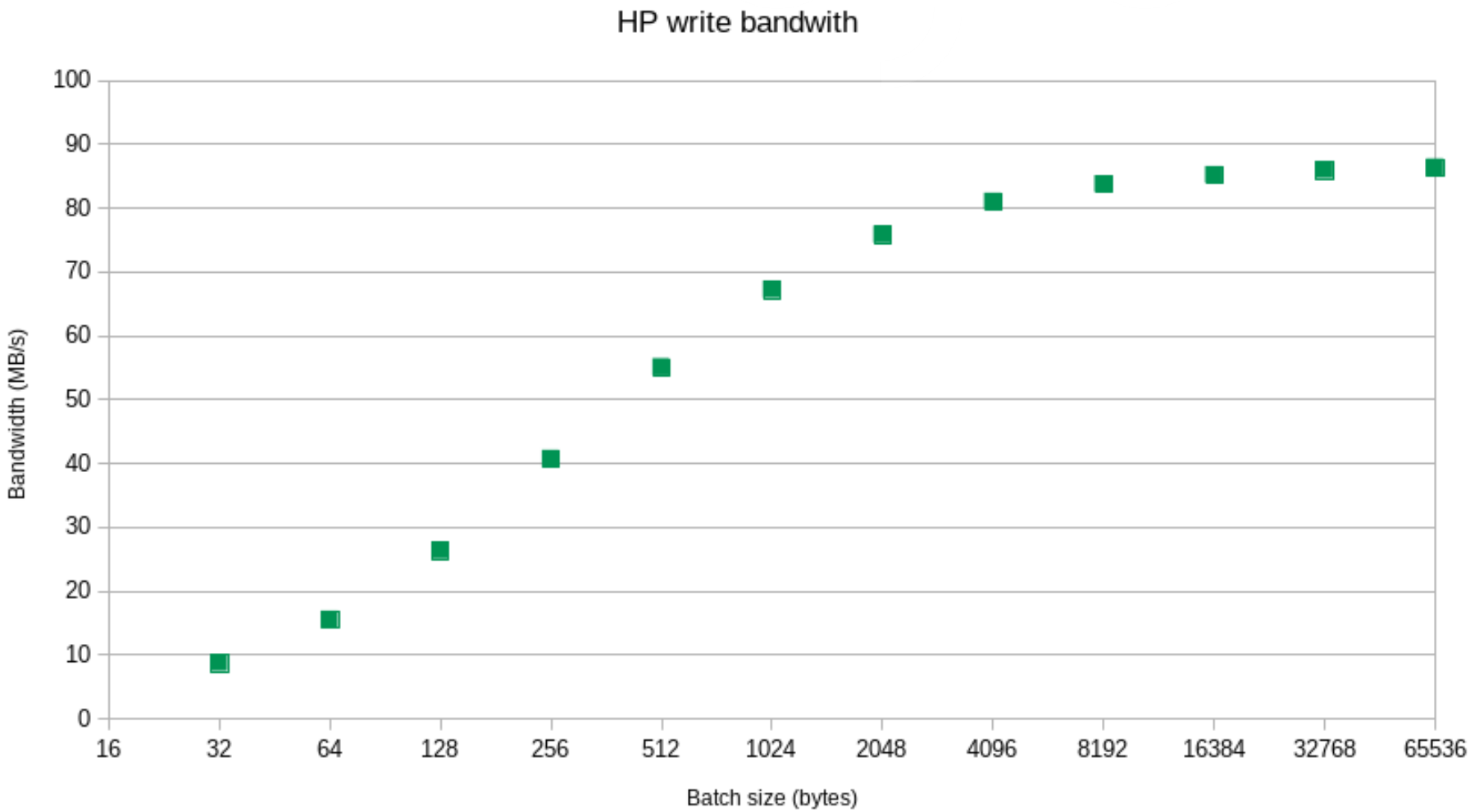
Entrée: Calcul $\mathcal{C}$
Sortie : Répartition des calculs entre CPU et FPGA, ensemble des blocs à invalider après calcul sur FPGA

```
1  Décomposer C en p sous-calculs (C_i)_{1≤i≤p};
2  Soit computed[1..p] tableau donnant l'état de chacun des C_i;
3  Soit dep[1..p] graphe de dépendances calculé à l'aide de D_flow;
4  nToDo ← p;
5  while nToDo > 0 do
6      for i ← 1 to p do
7          if depscomputed(i) then
8              Schedule ← i :: Schedule;
9              nToDo ← nToDo − 1;
10         end
11     end
12 end
13 toComputeCPU, toComputeFPGA ← dispatch(Schedule);
14 for c in toComputeFPGA do
15     Voir [2];
16     invalidateSet ← computeInvalidateSet(c);
17     writebackSet ← computeWritebackSet(c);
18 end
19 return Schedule, invalidateSet, writebackSet;
```

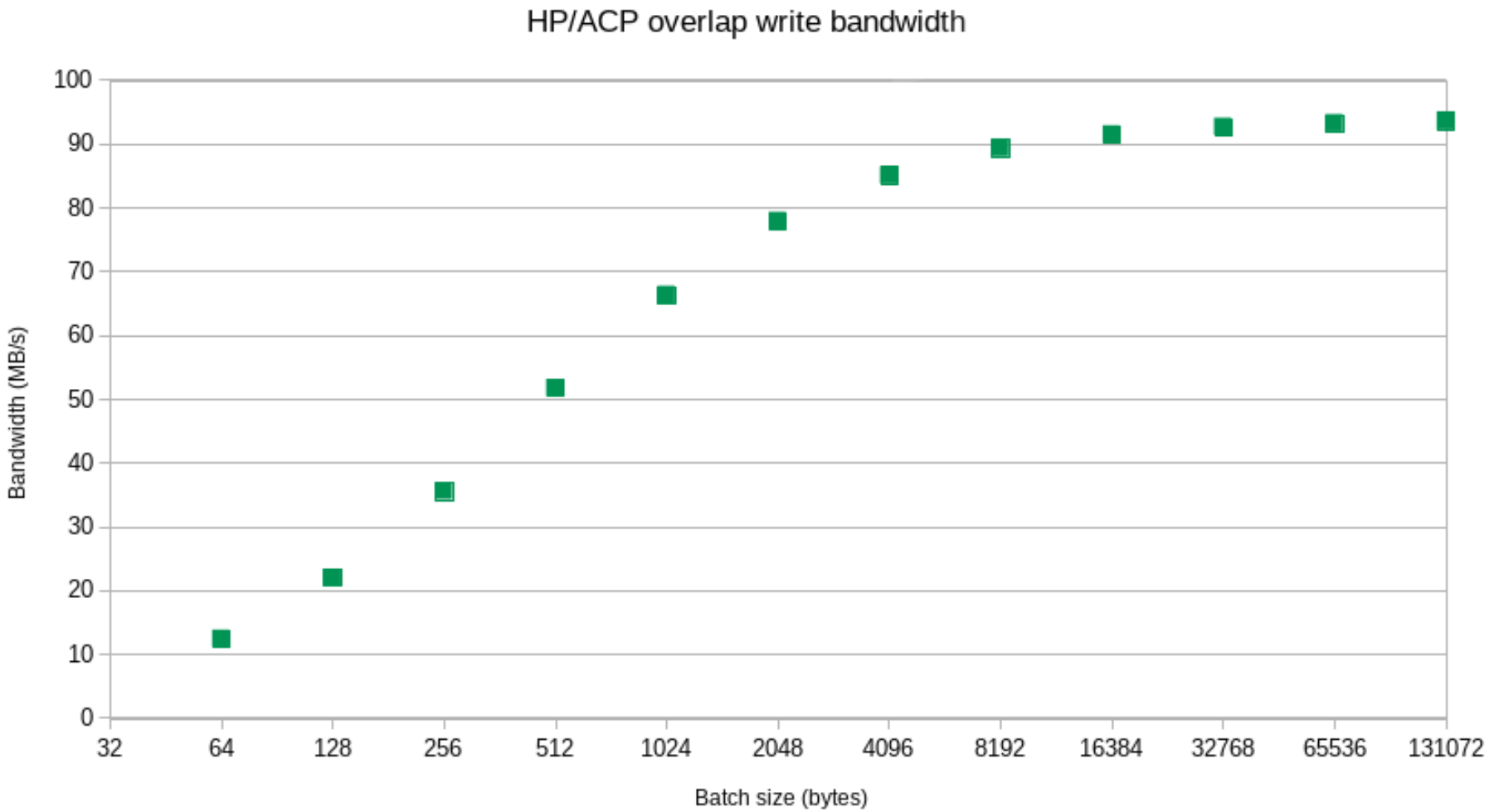
Micro-benchmarking (1/4)



Micro-benchmarking (2/4)



Micro-benchmarking (3/4)



Micro-benchmarking (4/4)