Affine Multibanking for High-Level Synthesis

Ilham Lasfar Christophe Alias Matthieu Moy
Rémy Neveu **Alexis Carré**

The 12th International Workshop on Polyhedral Compilation Techniques
(IMPACT'22)

High-Level Synthesis at a glance

High-Level Synthesis (HLS): Program → Hardware

- Typically: compute-intensive kernel → hardware accelerator IP
- Target: ASIC or FPGA

Typical flow: $C \xrightarrow{HLS} RTL \xrightarrow{synthesis} Hardware$

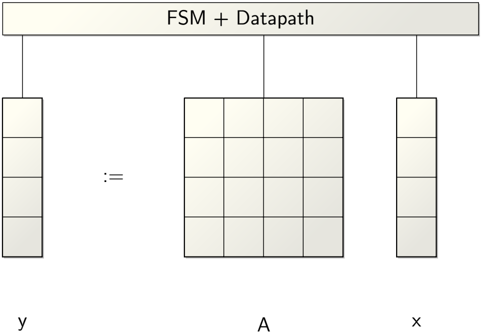
Architecture model:

- Von-Neumann (VivadoHLS)
- Synchronous dataflow (e.g. systolic networks) (AlphaZ)
- Asynchronous dataflow (e.g. KPN, RPN partitioning) (Dcc)

Focus: Efficient data mapping for Von-Neumann model

The trouble with arrays

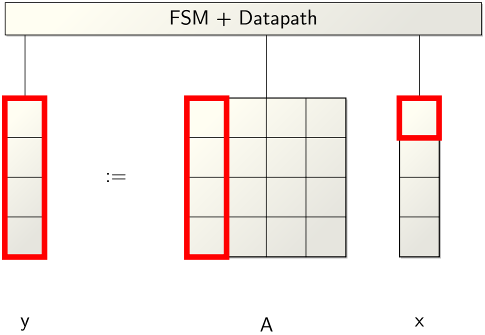
```
for (i = 0; i < N; i++) //parallel
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```



Arrays are implemented as block RAM, two data ports max.

The trouble with arrays

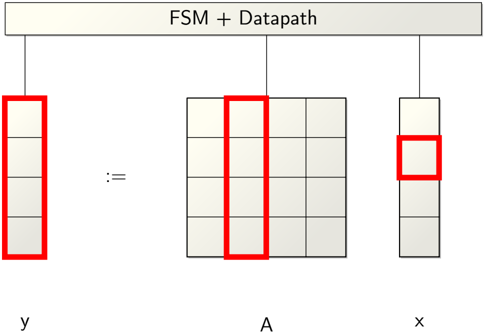
```
for (i = 0; i < N; i++) //parallel
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```



Arrays are implemented as block RAM, two data ports max.

The trouble with arrays

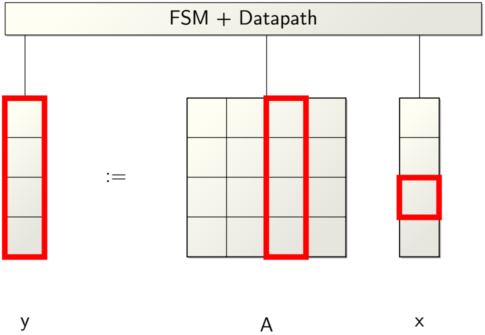
```
for (i = 0; i < N; i++) //parallel
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```



Arrays are implemented as block RAM, two data ports max.

The trouble with arrays

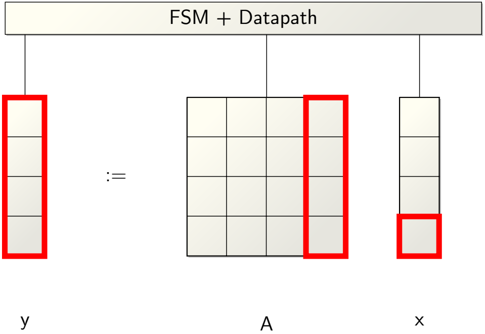
```
for (i = 0; i < N; i++) //parallel
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```



Arrays are implemented as block RAM, two data ports max.

The trouble with arrays

```
for (i = 0; i < N; i++) //parallel
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```



Arrays are implemented as block RAM, two data ports max.

Solution: multibanking

Multibanking: Partition data across memory banks readable in parallel

Vivado HLS: language-level array partitioning

- Array dimension(s) to be partitioned
- Array partitioning:
  - (cyclic or block) + factor
  - complete

0	1	...	$N-1$

$\longrightarrow$

block(2)

0	1	...	$N/2-1$
$N/2$	...	$N-2$	$N-1$

cyclic(2)

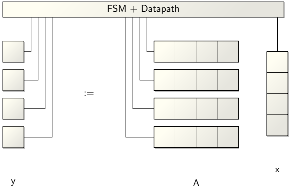
0	2	...	$N-2$
1	...	$N-3$	$N-1$

complete

0	1	...	$N-1$

Use case 1: matrix-vector product

```
#pragma HLS ARRAY_PARTITION \
    variable=y complete dim=1
#pragma HLS ARRAY_PARTITION \
    variable=A complete dim=1
for (i = 0; i < N; i++)
#pragma HLS PIPELINE
    for (j = 0; j < N; j++)
        y[i] += A[i][j] * x[j];
```

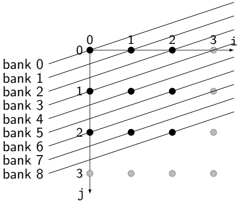


Synthesis results: (VivadoHLS 2019.1, Kintex 7 FPGA)

Kernel	Version	Latency	Interval	**Speed-up**	BRAM18K	DSP	FF	LUT	URAM
matvec	Baseline	532	533	**10.2**	0	320	4799	4423	0
	With banking	52	53		0	320	67618	13518	0

Use case 2: 2D convolution

```
for(i=1; i<N-1; i++)
    for(j=1; j<N-1; j++)
        out[i,j] =
            in[i-1,j-1]+in[i-1,j]+in[i-1,j+1]+
            in[i,j-1]  +in[i,j]  +in[i,j+1]  +
            in[i+1,j-1]+in[i+1,j]+in[i+1,j+1];
```



$$\text{bank}_{in}(i,j) = i + 3j \bmod 9 \quad \text{offset}_{in}(i,j) = j \bmod N$$

Methodology

- $in[u(\vec{i})] \mapsto \hat{in}[\text{bank}_{in}(u(\vec{i}))][\text{offset}_{in}(u(\vec{i}))]$
- Add pragmas to partition the bank dimensions: option cyclic, factor=9, dim=1

Multibanking problem

Input: Program + schedule
Output: allocation mappings:

- $\text{bank}_a(\vec{i})$: bank number of $a[\vec{i}]$ (can be a vector)
- $\text{offset}_a(\vec{i})$: offset of $a[\vec{i}]$ into his bank (can be a vector)

Multibanking problem

Input: Program + schedule

Output: allocation mappings:

- $\text{bank}_a(\vec{i})$: bank number of $a[\vec{i}]$ (can be a vector)
- $\text{offset}_a(\vec{i})$: offset of $a[\vec{i}]$ into his bank (can be a vector)

Focus: affine transformations (easier to derive)

- $\text{bank}_a(\vec{i}) = \phi_a(\vec{i}) \mod \sigma(\vec{N})$
- $\text{offset}_a(\vec{i}) = \psi_a(\vec{i}) \mod \tau(\vec{N})$

Multibanking problem

Input: Program + schedule
Output: allocation mappings:

- $\text{bank}_a(\vec{i})$: bank number of $a[\vec{i}]$ (can be a vector)
- $\text{offset}_a(\vec{i})$: offset of $a[\vec{i}]$ into his bank (can be a vector)

Focus: affine transformations (easier to derive)

- $\text{bank}_a(\vec{i}) = \phi_a(\vec{i}) \mod \sigma(\vec{N})$
- $\text{offset}_a(\vec{i}) = \psi_a(\vec{i}) \mod \tau(\vec{N})$

Source-to-source transformation:

- $a[u(\vec{i})] \mapsto \hat{a}[\text{bank}_a(u(\vec{i}))][\text{offset}_a(u(\vec{i}))]$
- Add pragmas to partition the bank dimensions

Contributions

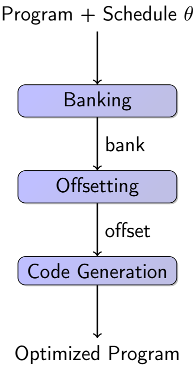
General formalization of the multibanking problem, which subsumes the previous approaches.

A general algorithm to compute our multibanking transformation.

Our approach **reduces the number of banks and the maximal bank size**, without hindering parallel accesses.

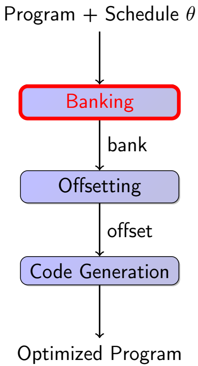
Outline

Overview



- **Solve** a system of affine constraints (parametric ILP)
- **Code generation:** $a[u(\vec{i})] \mapsto \hat{a}[\text{bank}_a(u(\vec{i}))][\text{offset}_a(u(\vec{i}))]$
  Then, add pragmas to partition the bank dimensions

Overview



- **Solve** a system of affine constraints (parametric ILP)
- **Code generation:** $a[u(\vec{i})] \mapsto \hat{a}[\text{bank}_a(u(\vec{i}))][\text{offset}_a(u(\vec{i}))]$
  Then, add pragmas to partition the bank dimensions

Banking constraints

Correctness: enforce distinct banks for concurrent access

$$a(\vec{i}) \parallel_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \mathrm{bank}_a(\vec{i}) \neq \mathrm{bank}_b(\vec{j})$$

Relaxed as:

$$a(\vec{i}) \parallel_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \phi_a(\vec{i}) \ll \phi_b(\vec{j})$$

Efficiency: reduce the bank number for each dimension

$$\phi_b(\vec{j}) - \phi_a(\vec{i}) \leq \sigma(\vec{N})$$

Banking constraints

Correctness: enforce distinct banks for concurrent access

$$a(\vec{i}) \parallel_\theta b(\vec{j}) \land (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \mathrm{bank}_a(\vec{i}) \neq \mathrm{bank}_b(\vec{j})$$

Relaxed as:

$$a(\vec{i}) \parallel_\theta b(\vec{j}) \land (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \phi_a(\vec{i}) \ll \phi_b(\vec{j})$$

Efficiency: reduce the bank number for each dimension

$$\phi_b(\vec{j}) - \phi_a(\vec{i}) \leq \sigma(\vec{N})$$

Analogous to **affine scheduling:**

operation	array cell
dependence	concurrent access
latency	number of banks

Outline

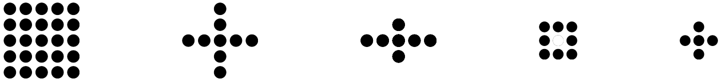
Experimental results (1/2)

Setup:

- VivadoHLS 2019.1
- Target: Kintex 7 FPGA (xc6k70t-fbv676-1)

Benchmarks:

- *Linear algebra:* matvec, matmul
- *Stencils:* jacobi2d, seidel2d
- *Convolutions:* conv2d, canny, gaussian, median, prewitt, se



Preliminary prototyping, using fkcc

Experimental results

Kernel	Version	Latency	Interval	**Speed-up**	BRAM18K	DSP	FF	LUT	URAM
matvec	Baseline	532	533	**10.2**	0	320	4799	4423	0
	With banking	52	53		0	320	67618	13518	0
matmul	Baseline	1555	1556	**29.9**	0	10240	135581	123129	0
	With banking	52	53		0	10240	196648	152161	0
conv2d	Baseline	1442	1443	**29.4**	0	0	923	4290	0
	With banking	49	50		0	0	65562	33043	0
jacobi2d	Baseline	11011	11012	**1.6**	0	0	117140	96019	0
	With banking	6851	6852		0	0	192295	137499	0
seidel2d	Baseline	6914	6915	**2.0**	0	0	452	1280	0
	With banking	3458	3459		0	0	574	2903	0
canny	Baseline	10194	10195	**4.3**	0	0	669	1837	0
	With banking	2355	2356		0	0	6616	6085	0
gaussian	Baseline	3922	3923	**1.7**	0	0	449	1012	0
	With banking	2354	2355		0	0	2367	2811	0
median	Baseline	3362	3363	**1.3**	0	0	373	846	0
	With banking	2522	2523		0	0	2367	2501	0
prewitt	Baseline	3846	3847	**2.0**	0	0	371	906	0
	With banking	1924	1925		0	0	2249	2142	0

- **Trade-off** surface $\leftrightarrow$ performance still to be explored

Outline

Conclusion

Contributions:

- A general formalization & algorithm for affine multibanking
- Our approach reduces the number of banks and the maximal bank size, without hindering parallel accesses.
- Promising (but still preliminary) experimental validation

Perspectives:

- Common bank size, minimize each bank size
- Investigate the trade-off circuit size/latency (through tiling?)

Questions?

Banking algorithm

Input: Program (P, θ)
Output: Bank mapping $\text{bank}_a : (\vec{i}, \vec{N}) \mapsto \phi_a(\vec{i}) \mod \sigma(\vec{N})$, for each array a

1. $\mathcal{C} \leftarrow \{(a(\vec{i}), b(\vec{j})) \mid a(\vec{i}) \parallel_\theta b(\vec{j}) \wedge \vec{i} \ll \vec{j} \wedge \vec{i} \in \mathcal{D}_a \wedge \vec{j} \in \mathcal{D}_b\}$

Banking algorithm

Input: Program (P, θ)
Output: Bank mapping $\text{bank}_a : (\vec{i}, \vec{N}) \mapsto \phi_a(\vec{i}) \mod \sigma(\vec{N})$, for each array a

1. $\mathcal{C} \leftarrow \{(a(\vec{i}), b(\vec{j})) \mid a(\vec{i}) \parallel_\theta b(\vec{j}) \wedge \vec{i} \ll \vec{j} \wedge \vec{i} \in \mathcal{D}_a \wedge \vec{j} \in \mathcal{D}_b\}$
2. $d \leftarrow 0$
3. **while** $\mathcal{C} \neq \emptyset$
    1. $\min_{\ll} \sigma^d$ coefficients s.t. $\text{correct}(\mathcal{C}, \phi^d) \wedge \text{efficient}(\mathcal{C}, \phi^d, \sigma^d) \wedge \phi^d$ non-constant
    2. $\mathcal{C} \leftarrow \mathcal{C} \cap \{(a(\vec{i}), b(\vec{j})) \mid \phi_a^d(\vec{i}) = \phi_b^d(\vec{j})\}$
    3. $d \leftarrow d + 1$
4. **return** bank

$\text{correct}(\mathcal{C}, \phi) : (a(\vec{i}), b(\vec{j})) \in \mathcal{C} \wedge \vec{i} \ll \vec{j} \Rightarrow \phi_a(\vec{i}) \leq \phi_b(\vec{j})$
$\text{efficient}(\mathcal{C}, \phi, \sigma) : (a(\vec{i}), b(\vec{j})) \in \mathcal{C} \wedge \vec{i} \ll \vec{j} \Rightarrow \phi_b(\vec{j}) - \phi_a(\vec{i}) \leq \sigma(\vec{N})$

Offset constraints

Correctness: enforce distinct offsets for conflicting array cells

$$\mathrm{bank}_a(\vec{i}) = \mathrm{bank}_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \mathrm{offset}_a(\vec{i}) \neq \mathrm{offset}_b(\vec{j})$$

Relaxed as:

$$\phi_a(\vec{i}) = \phi_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \psi_a(\vec{i}) \ll \psi_b(\vec{j})$$

Efficiency: minimize the number of offsets (into a same bank)

$$\phi_a(\vec{i}) = \phi_b(\vec{j}) \Rightarrow \psi_b(\vec{j}) - \psi_a(\vec{i}) \leq \tau(\vec{N})$$

Offset constraints

Correctness: enforce distinct offsets for conflicting array cells

$$\mathrm{bank}_a(\vec{i}) = \mathrm{bank}_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \mathrm{offset}_a(\vec{i}) \neq \mathrm{offset}_b(\vec{j})$$

Relaxed as:

$$\phi_a(\vec{i}) = \phi_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge (a, \vec{i}) \neq (b, \vec{j}) \Rightarrow \psi_a(\vec{i}) \ll \psi_b(\vec{j})$$

Efficiency: minimize the number of offsets (into a same bank)

$$\phi_a(\vec{i}) = \phi_b(\vec{j}) \Rightarrow \psi_b(\vec{j}) - \psi_a(\vec{i}) \leq \tau(\vec{N})$$

Again, analogous to affine scheduling:

operation	array cell
dependence	liveness conflict
latency	number of offsets

Offsetting algorithm (almost the same)

Input: Program (P, θ), $\text{bank}_a : (\vec{i}, \vec{N}) \mapsto \phi_a(\vec{i}) \bmod \sigma(\vec{N})$ for each array a

Output: Offset mapping $\text{offset}_a : (\vec{i}, \vec{N}) \mapsto \psi_a(\vec{i}) \bmod \tau(\vec{N})$, for each array a

1. $\mathcal{C} \leftarrow \{(a(\vec{i}), b(\vec{j})) \mid \phi_a(\vec{i}) = \phi_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge \vec{i} \ll \vec{j} \wedge \vec{i} \in \mathcal{D}_a \wedge \vec{j} \in \mathcal{D}_b\}$

Offsetting algorithm (almost the same)

Input: Program (P, θ), $\text{bank}_a : (\vec{i}, \vec{N}) \mapsto \phi_a(\vec{i}) \bmod \sigma(\vec{N})$ for each array a

Output: Offset mapping $\text{offset}_a : (\vec{i}, \vec{N}) \mapsto \psi_a(\vec{i}) \bmod \tau(\vec{N})$, for each array a

1. $\mathcal{C} \leftarrow \{(a(\vec{i}), b(\vec{j})) \mid \phi_a(\vec{i}) = \phi_b(\vec{j}) \wedge a(\vec{i}) \bowtie_\theta b(\vec{j}) \wedge \vec{i} \ll \vec{j} \wedge \vec{i} \in \mathcal{D}_a \wedge \vec{j} \in \mathcal{D}_b\}$
2. $d \leftarrow 0$
3. **while** $\mathcal{C} \neq \emptyset$
    1. $\min_{\ll} \tau^d$ coefficients s.t. $\text{correct}(\mathcal{C}, \psi^d) \wedge \text{efficient}(\mathcal{C}, \psi^d, \tau^d) \wedge \psi^d$non-constant
    2. $\mathcal{C} \leftarrow \mathcal{C} \cap \{(a(\vec{i}), b(\vec{j})) \mid \psi_a^d(\vec{i}) = \psi_b^d(\vec{j})\}$
    3. $d \leftarrow d + 1$
4. **return** offset

$\text{correct}(\mathcal{C}, \psi) : (a(\vec{i}), b(\vec{j})) \in \mathcal{C} \wedge \vec{i} \ll \vec{j} \Rightarrow \psi_a(\vec{i}) \leq \psi_b(\vec{j})$
$\text{efficient}(\mathcal{C}, \psi, \tau) : (a(\vec{i}), b(\vec{j})) \in \mathcal{C} \wedge \vec{i} \ll \vec{j} \Rightarrow \psi_b(\vec{j}) - \psi_a(\vec{i}) \leq \tau(\vec{N})$