Design Space Exploration of Function Reusing in Functional IRs for Accelerator Generation

Tzung-Han Juang[1], Christof Schlaak[2], Christophe Dubach[1] [1]McGill University, Canada [2]University of Edinburgh, United Kingdom

Abstract

FPGAs (Field Programmable Gate Arrays) are solid platforms for prototyping hardware accelerators but they are notorious for programmability. Commercial tools provide a high-level abstraction with c-like language. However, the method is still far from perfect since it is difficult to manage resources such as DSPs (Digital Signal Processors). Our work proposes a coarse-grained FPGA resource control based on function reusing in function IRs (Intermediate Representations). The approach provides a friendly way to fine-tune the performance of accelerators.

Motivation

Commercial tools such as Intel OpenCL FPGA SDK poorly support course-grained resource sharing with functions. **FPGA resource usage increases with the number of function calls.**

```
void mm(int* A, int*B, int* C, int sz) {
  for(i=0, j=0; i<sz, j<sz; i++, j++) {
    #pragma unroll
    for(int k=0; k<size; k++)
      C[i][j] = A[i][k] * B[j][k]; }}

kernel mxm(int* A, int*B, int* C) {
  matMul(A, B, C);
  matMul(A, B, C);
  matMul(A, B, C);}
```

Duplicated Matrix Mul. with OpenCL.

Arria 10 FPGA usage

# of calls	Resources Logic(%)	RAM(%)	DSP(%)
1	23	19	34
2	32	36	68
3	Out of DSPs		

We extend SHIR [1, 2], a functional accelerator generation framework, which **embeds type into expression, and therefore is good for design space exploration**.

```
Let matMul = λ X, Y ->
  // X.t = Seq[Seq[Int, size], size]
  // Y.t = Seq[Seq[Int, size], size]
  Map(λ rX => Map(λ cY =>
    Reduce(+, Map(λ m => Mul(m), Zip(rX, cY))), X), Y)
in
  FunCall(matMul, A, B)
  FunCall(matMul, A, B)
  FunCall(matMul, A, B)
```

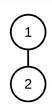
Duplicated Matrix Mul. with SHIR.

Methods

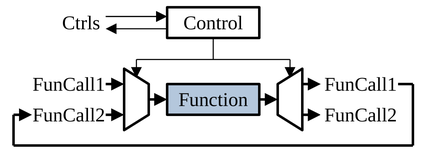
We combine *Let* binding and *Lambda* abstraction to achieve a reusable function. However, it introduces the problem of data dependency with nested function calls at hardware level. We build **interference graphs** and apply the techniques from **register allocation** to solve it.

```
Let f = λ x ->
  /* computation */
in
  FunCall(f,
    FunCall(f, i))
```

(a)



(b)

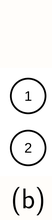


(c)

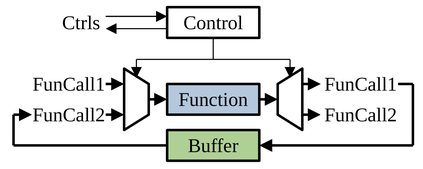
Nested function call in SHIR. (a) Expression. (b) Interference graph. (c) Architecture.

```
Let f = λ x ->
  /* computation */
in
  FunCall(f,
    Buffer(
    FunCall(f, i)))
```

(a)



(b)

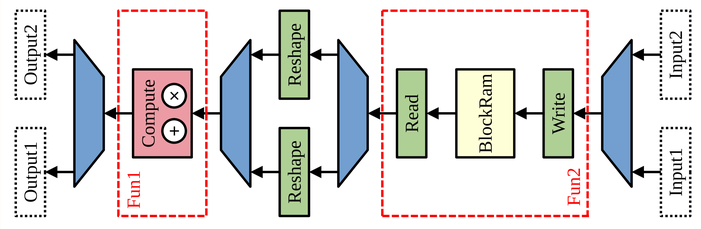


(c)

Nested function call with buffer fix. (a) Expression. (b) Interference graph. (c) Architecture.

Function Merging Optimization. We still have the problem of **redundant interconnects and components** between functions. To solve this, we extend the **rewrite rule** system in SHIR to allow function reconstruction.

```
Let f1 = λ x -> Compute(x) in
  Let f2 = λ x -> BlockRAMBuffer(x) in
    output1 = FunCall(f1, Reshape(FunCall(f2, input1)))
    output2 = FunCall(f1, Reshape(FunCall(f2, input2)))
```



Expression and Architecture with redundant interconnects.

Results

We evaluate our approaches on Intel Arria 10 FPGA with matrix multiplications and convolutional neural networks.

Function Sharing and Optimizations on matrix multiplications.

Opt.	Setup # of MatMul	Perf. DSP eff. (%)	Resources Logic (%)	RAM (%)	DSP (%)	Connect. (%)
No Sharing	1	89	19	22	34	17
No Sharing	2	93	34	39	67	32
Sharing Computation	2	93	29	39	34	28
Function Merging Opt.	2	93	19	22	34	17

Performance of VGG Layers

layer id	Setup input size	in channel	out channel	Perf. DSP eff. (%)	Resources Logic (%)	RAM (%)	DSP (%)	Connect. (%)
0	32	3	64	43	18	6	28	15
1	32	64	64	87	33	20	76	31
2	16	64	128	70	27	20	76	25
3	16	128	128	83	38	21	76	28
4	8	128	256	48	25	21	76	25
layer 0 - 4, w/o opt.	Not synthesizable!							
layer 0 - 4, w/ opt.				75	39	23	76	35

References

[1] Christof Schlaak, Tzung-Han Juang, and Christophe Dubach. Memory-aware functional ir for higher-level synthesis of accelerators. *ACM Transactions on Architecture and Code Optimization*, 19(2), jan 2022.

[2] Christof Schlaak, Tzung-Han Juang, and Christophe Dubach. Optimizing data reshaping operations in functional irs for high-level synthesis. In *23rd ACM International Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES, 2022.