Enable Resource-aware Scheduling for Reconfigurable Hardware Accelerators

Qiaoyi (Joey) Liu
Apr 12 2023
Benefit of Accelerator

Task(Pipeline) Parallelism + locality
Review of the App Mapping Flow

Halide Schedule:
- Define how to compute (loop tiling, ordering, mem-hierarchy)
Review of the App Mapping Flow

- **Buffer Extraction**:  
  - Extract access pattern  
  - Polyhedral Scheduling (cycle accurate) to create pipeline parallelism

- **Halide Schedule**:  
  - Define how to compute (loop tiling, ordering, mem-hierarchy)
Review of the App Mapping Flow

**Halide Schedule:**
- Define how to compute (loop tiling, ordering, mem-hierarchy)

**Buffer Extraction:**
- Extract access pattern
- Polyhedral Scheduling (cycle accurate) to create pipeline parallelism

**Buffer Mapping:**
- Mapping to resource constrained **physical hardware**
Bring Resource Awareness to the Scheduler

- **Halide Schedule**: Define how to compute (loop tiling, ordering, mem-hierarchy)

- **Buffer Extraction**: Extract access pattern, Polyhedral Scheduling (cycle accurate) to create pipeline parallelism

- **Buffer Mapping**: Mapping to resource-constrained physical hardware
Related Work: Clockwork Loop Fusion [1]

- Analyze the data dependency between operations

```plaintext
for (y, 0, 64)
    for (x, 0, 64)
        brighten(x, y) = input(x, y) * 2;

for (y, 0, 63)
    for (x, 0, 63)
        blur(x, y) = (brighten(x, y) + brighten(x+1, y) +
                      brighten(x, y+1) + brighten(x+1, y+1))/4;
```

[1] Clockwork: Resource-Efficient Static Scheduling for Multi-Rate Image Processing Applications on FPGAs
Related Work: Clockwork Loop Fusion\cite{1}

- Analyze the data dependency between operations
- Loop fusion (Generate optimized code)
  - Bring consumer closer to producer, finer grained interleaving

```
for (y, 0, 64)
  for (x, 0, 64)
    brighten(x, y) = input(x, y) * 2;

for (y, 0, 63)
  for (x, 0, 63)
    blur(x, y) = (brighten(x, y) + brighten(x+1, y) +
                  brighten(x, y+1) + brighten(x+1, y+1))/4;
```

```
for (y, 0, 64)
  for (x, 0, 64)
    brighten(x, y) = input(x, y) * 2;

  if (y >= 1 && x>= 1)
    blur(x, y) = (brighten(x-1, y-1) + brighten(x, y-1) +
                  brighten(x-1, y) + brighten(x, y))/4;
```

[1] Clockwork: Resource-Efficient Static Scheduling for Multi-Rate Image Processing Applications on FPGAs
Related Work: Clockwork Loop Fusion [1]

- Analyze the data dependency between operations
- Loop fusion (Generate optimized code)
  - Bring consumer closer to producer, finer grained interleaving
  - Storage folding
- Hand to HLS (Assume each operation will get dedicated HW)
  - Loop pipeline, resource binding -> reduce latency
  - Logic synthesize the memory implementation on FPGA

---

[1] Clockwork: Resource-Efficient Static Scheduling for Multi-Rate Image Processing Applications on FPGAs
Motivations

• We are mapping broader set of applications
  • Loop fusion may not always legal to apply

• We are targeting a CGRA
  • Does not have HLS
    • Loop pipeline
    • Memory synthesize
  • Create dedicated hardware for each operation may not be optimal
Example 1: Gaussian Pyramid

Stage 1: Load

```c
For (y, 0, 64)
    For (x, 0, 64)
        cpy (input_GLB(x,y), input(x,y));
```

Stage 2: p1

```c
For (y, 0, 32)
    For (x, 0, 32)
        p1(x,y) = gp(input(2x,2y)…);
```

Stage 3: p2

```c
For (y, 0, 16)
    For (x, 0, 16)
        p2(x,y) = gp(p1(2x,2y)…);
```

Stage 4: p3

```c
For (y, 0, 8)
    For (x, 0, 8)
        out(x,y) = gp(p2(2x,2y)…);
```

Low Temporal Utilization for compute(PE) ...
Example 2: Multi-Layer DNN: Compute

Simply map 3 CNN layer will exceed the number of PEs on our CGRA
Example 2: Multi-Layer DNN: Memory

Simply map 3 CNN layer onto our CGRA will exceed the number of PE we have.

Output partial sum has four different operations But memory may not have enough port
Scheduling Questions: Resource Binding

• How can I create pipeline with resource constraints, including
  ▪ Sharing(time-multiplex) the **compute** hardware between operations
  ▪ Sharing(time-multiplex) the **memory** port between operations
Scheduler with resource awareness

• Assign a timestamp (cycle accurate) for every iteration
• Minimize: latency
• Without violating:
  • Data dependency
  • Resource constraints
Scheduler Flow

- Loopnest
  - Loop Fusion
  - Fused Loopnest
  - Loop Pipeline
  - Cycle Accurate Schedule

- User Guided information from Halide
- Target Fuse Level
- Hardware Binding & Constraints

- Post-processing
  - Loop Flatten
  - Loop Perfection
When should we fuse the loop?

```c
for (yo, 0, 2)
    for (xo, 0, 2)
        for (y, 0, 64)
            for (x, 0, 64)
                brighten(x, y) = input(x, y) * 2;
        for (x, 0, 63)
            for (y, 0, 63)
                blur(x, y) = (brighten(x, y) + brighten(x+1, y) +
                               brighten(x, y+1) + brighten(x+1, y+1))/4;
```
When should we fuse the loop?

```plaintext
for (yo, 0, 2)
    for (xo, 0, 2)
        for (y, 0, 64)
            for (x, 0, 64)
                brighten(x, y) = input(x, y) * 2;

for (x, 0, 63)
    for (y, 0, 63)
        blur(x, y) = (brighten(x, y) + brighten(x+1, y) +
                      brighten(x, y+1) + brighten(x+1, y+1))/4;
```

- **Fusion level**
  - By default: fuse till the innermost loop
  - User could manually adjust the fusion level from Halide

- **Whether to fuse or not**
  - If producer and consumer access the data in the same order
Loop Pipeline [1] [2]

Sequentially schedule

Overlap the Blur(yo, xo) with Brighten(yo, xo+1), compute restart ASAP

• Increasing compute utilization, reduce latency
Create a double buffer in the middle

```
for (yo, 0, 2)
    for (xo, 0, 2) ← pipeline target loop

for (y, 0, 64)
    for (x, 0, 64)
        brighten(x, y) = input(x, y) * 2;

for (x, 0, 63)
    for (y, 0, 63)
        blur(x, y) = (brighten(x, y) + brighten(x+1, y) +
                      brighten(x, y+1) + brighten(x+1, y+1))/4;
```

[1] HLS provide innermost loop pipeline
[2] Spatial also provide coarse-grained pipeline
So how to model the resource constraints?
Resource Reservation Table: model the resource constraints

- Resource Reservation table (RRT): single iteration

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>LD</th>
<th>Alu</th>
<th>ST</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Static Schedule

<table>
<thead>
<tr>
<th>Operation</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 LD</td>
<td>S(0) = 0</td>
</tr>
<tr>
<td>1 LD</td>
<td>S(1) = 1</td>
</tr>
<tr>
<td>2 Mul</td>
<td>S(2) = 2</td>
</tr>
<tr>
<td>3 Add</td>
<td>S(3) = 5</td>
</tr>
<tr>
<td>4 ST</td>
<td>S(4) = 7</td>
</tr>
</tbody>
</table>
Iterative Modulo Schedule

- Find a static schedule $S(t)$, minimize the initiation interval ($II$)
- Without resource conflicts and dependency violation

Start
0 LD
1 LD
2 Mul
3 Add
4 ST

$II=2$

[1] CS243 Software Pipelining Course Slides
Iterative Modulo Schedule

- Find a static schedule $S(t)$, minimize the initiation interval (II)
- Without resource conflicts and dependency violation
Iterative Modulo Schedule

• Find a static schedule $S(t)$, minimize the initiation interval (II)
• Without resource conflicts and dependency violation
Modulo Reservation Table: Steady State

- Resource Reservation table (RRT): single iteration
- Modulo Reservation table (MRT): steady state
Extend to Hardware Pipeline

1. Single operation/instruction
   • \( \Rightarrow \) A pipeline stage: a set of operation wrapped by loop nest
   **Assuming:** Resource reserved until the pipeline stage is finished

2. Resource Reservation Table:
   • \( \text{ALU} + \text{LD} + \text{ST} \Rightarrow \text{compute(PE)} + \text{memory Rd} + \text{Memory Wr} \)
DNN kernel and memory port constraints

For (p, 0, 16) //pipeline for
  init (out(x,y,k), 0)

Copy (input_gb(x,y,c,p), input(x,y,c))

Copy (weight_gb(kx,ky,c,k,p), weight(kx.ky,c,k))

out <- MACC(out, input, weight)

Copy (out(x,y,k), out_gb(x,y,k))
Output Buffer has 2 wr 2 rd
Model the Memory Port
Build the RRT for memory

Input
Rd.  Wr

Weight
Rd.  Wr

Output
Rd.  Wr

init O
load I
load w
Compute
Drain O
Modulo Iterative Schedule

init O → load I → load w → Compute → Drain O
Modulo Iterative Schedule

init O -> load I -> load w -> Compute -> Drain O

I
Rd. Wr
W
Rd. Wr
O
Rd. Wr

32
Modulo Iterative Schedule

init O -> load I -> load w

Compute

Drain O

I
Rd. Wr

W
Rd. Wr

O
Rd. Wr

init O

load I

load w

Compute

Drain O
Modulo Iterative Schedule

- init O
- load I
- load w

Diagram:

I
Rd. Wr

W
Rd. Wr

O
Rd. Wr

init O
load I
load w
Compute
Drain O

34
Modulo Iterative Schedule
Minimum Coarse grained II = 2 with dual port
II = max(compute.latency, 
load_I.latency, 
load_w.latency, 
drain_o.latency + compute.latency 
init.latency + compute.latency)
What happened if we have two input and two output port
Coarse grained $\Pi=1$

\[ \Pi = \max(\text{compute.latency}, \text{load}_I\text{.latency}, \text{load}_W\text{.latency}, \text{drain}_O\text{.latency}, \text{compute.latency}) \]
Resource Aware Scheduling: Pipeline Diagram
Resource Aware Scheduling: memory port constraints for resnet layer 3x

![Graph showing compute latency comparison between dual and quad port configurations, with 35% less latency for quad port configuration.](image-url)
Gaussian pyramid

Stage 1: Load from GLB

```
For (y, 0, 64)
  For (x, 0, 64)
    cpy (input_GLB(x,y), mem(x,y));
```

Stage 2: p1

```
For (y, 0, 32)
  For (x, 0, 32)
    p1(x,y) = gp(input(2x,2y)...);
```

Stage 3: p2

```
For (y, 0, 16)
  For (x, 0, 16)
    p2(x,y) = gp(p1(2x,2y)...);
```

Stage 4: p3

```
For (y, 0, 8)
  For (x, 0, 8)
    out(x,y) = gp(p2(2x,2y)...);
```
First Fuse the Loops

For (y, 0, 64)
  For (x, 0, 64)
    cpy (input_GLB(x,y), mem(x,y));

For (y, 0, 32)
  For (x, 0, 32)
    p1(x,y) = gp(input(2x,2y)…);

For (y, 0, 16)
  For (x, 0, 16)
    p2(x,y) = gp(p1(2x,2y)…);

For (y, 0, 8)
  For (x, 0, 8)
    out(x,y) = gp(p2(2x,2y)…);

For (y, 0, 64)
  For (x, 0, 64)
    cpy (input_GLB(x,y), mem(x,y));

  if (x%2 == 0 && y%2 == 0)
    p1(x,y) = gp(input(2x,2y)…);

  if (x%4 == 0 && y%4 == 0)
    p2(x,y) = gp(p1(2x,2y)…);

  if (x%8 == 0 && y%8 == 0)
    out(x,y) = gp(p2(2x,2y)…);
Under-Utilization if we assign dedicated compute

For \((y, 0, 64)\)
For \((x, 0, 64)\)
cpy (input\_GLB\((x, y)\), mem\((x, y)\));

For \((y, 0, 32)\)
For \((x, 0, 32)\)
p1\((x, y)\) = gp(input\((2x, 2y)\)...);

For \((y, 0, 16)\)
For \((x, 0, 16)\)
p2\((x, y)\) = gp(p1\((2x, 2y)\)...);

For \((y, 0, 8)\)
For \((x, 0, 8)\)
out\((x, y)\) = gp(p2\((2x, 2y)\)...);

For \((y, 0, 64)\)
For \((x, 0, 64)\)
cpy (input\_GLB\((x, y)\), mem\((x, y)\));

\[
\begin{align*}
\text{if } (x \% 2 == 0 \&\& y \% 2 == 0) \\
p1(x, y) &= \text{gp(input}(2x, 2y)\text{...});
\end{align*}
\]

\[
\begin{align*}
\text{if } (x \% 4 == 0 \&\& y \% 4 == 0) \\
p2(x, y) &= \text{gp(p1}(2x, 2y)\text{...});
\end{align*}
\]

\[
\begin{align*}
\text{if } (x \% 8 == 0 \&\& y \% 8 == 0) \\
out(x, y) &= \text{gp(p2}(2x, 2y)\text{...});
\end{align*}
\]

1/4
1/16
1/64
Time Multiplex the Compute Hardware

\[
\begin{align*}
\text{For } (y, 0, 64) \\
\text{For } (x, 0, 64) \\
\text{cpy } \text{(input\_GLB}(x,y), \text{mem}(x,y)); \\
\text{if } (x%2 == 0 \text{ && } y%2 == 0) \\
p1(x,y) = \text{gp(input}(2x,2y)\text{...}); \\
\text{if } (x%4 == 0 \text{ && } y%4 == 0) \\
p2(x,y) = \text{gp(p1}(2x,2y)\text{...}); \\
\text{if } (x%8 == 0 \text{ && } y%8 == 0) \\
\text{out}(x,y) = \text{gp(p2}(2x,2y)\text{...});
\end{align*}
\]
Change the interleaving granularity:

### After Loop Fusion

```plaintext
For (yo, 0, 8) <- Fusing_tag()
   For (y, 0, 8)
      For (x, 0, 64)
         cpy (input,GLB(x,y), mem(x,y));
   
   For (y, 0, 4)
      For (x, 0, 32)
         p1(x,y) = gp(input(2x,2y)...);

   For (y, 0, 2)
      For (x, 0, 16)
         p2(x,y) = gp(p1(2x,2y)...);

   For (x, 0, 8)
      out(x,y) = gp(p2(2x,2y)...);
```
## Resource Reservation Table

<table>
<thead>
<tr>
<th>Latency</th>
<th>Input</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>GP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$64 \times 8 = 512$</td>
<td>Rd.</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$32 \times 4 = 128$</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$16 \times 2 = 32$</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8 \times 1 = 8$</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Latency calculations:

- Latency = 64x8 = 512
- Latency = 32x4 = 128
- Latency = 16x2 = 32
- Latency = 8x1 = 8
Resource Aware Scheduling: Compute Share

**Spatial Usage**
- No-share: Bar heights range from 0 to 18
- Share: Bar heights range from 0 to 4

**Temporal Usage**
- No-share: Bar heights range from 0 to 1
- Share: Bar heights range from 0 to 0.5

**Gaussian Pyramid**
- PE Temporal Occupancy
  - No-share: 0.1
  - Share: 0.4

**Cycles**
- No-share: Approximately 4500 cycles
- Share: Approximately 4000 cycles
Increase Compute Temporal Utilization

**Gaussian Pyramid**

**PE Count**
- Spatial Usage
  - No-share: 18
  - Share: 2

**PE Temporal Occupancy**
- Temporal Usage
  - No-share: 0.1
  - Share: 0.9
  - Share w/t BW-opt: 1

**Compute Latency**
- Cycles
  - No-share: 50
  - Share: 4500
  - Share w/t BW-opt: 1500
Resource Aware Scheduling: Compute Share

**SRCNN (Multi Layer CNN for super resolution)**

![Graphs showing PE Count, PE Temporal Occupancy, and Latency for Spatial Usage and Temporal Usage with No-share and Share]

- **PE Count**
  - Spatial Usage:
    - No-share: 700
    - Share: 200
  - Temporal Usage:
    - No-share: 0.9
    - Share: 0.8

- **PE Temporal Occupancy**
  - Spatial Usage:
    - No-share: 0.4
    - Share: 0.5
  - Temporal Usage:
    - No-share: 0.1
    - Share: 0.9

- **Latency**
  - Spatial Usage:
    - No-share: 100
    - Share: 300
  - Temporal Usage:
    - No-share: 25000
    - Share: 20000

---

51
Hardware Generation

- Adding an input mux and output broadcast
Future works: memory sharing

• If there is memory time-multiplex, we need to update a controller to support more complex access pattern
  • Build a piecewise-affine controller
  • No extra hardware, reconfigure the controller!
Thanks! Questions?