Exploiting Run-time Reconfiguration In Stencil Computation

Xinyu Niu, Qiwei Jin, Wayne Luk, Qiang Liu and Oliver Pell

Dept. of Computing, School of Engineering, Imperial College London, UK
School of Electronic Information Engineering, Tianjin University, China
Maxeler Technologies, UK
Overview

• Background
• Partitioning algorithm
• Analytical model
• Reconfiguration scheduling
• Results
• Future work
• Conclusion
Background: run-time reconfiguration

• Purna and Bhatia 1999
  - temporal partitioning and data flow graph scheduling
• Singhal and Bozorgzadeh 2006
  - floorplanning for reconfiguration
• Bruneel, Abouelella and Stroobandt 2009
  - mapping applications to self-reconfiguring platform
• Iskander 2010
  - accelerate design validation
• Koch and Torresen 2011
  - run-time reconfigurable sorter for large-scale problems
Background: run-time reconfiguration

- Purna and Bhatia 1999
  - temporal partitioning and data flow graph scheduling
- Singhal and Bozorgzadeh 2006
  - floorplanning for reconfiguration
- Bruneel, Abouelella and Stroobandt 2009
  - mapping applications to self-reconfiguring platform
- Iskander 2010
  - accelerate design validation
- Koch and Torresen 2011
  - run-time reconfigurable sorter for large-scale problems
- Our focus: automate reconfigurable stencil computation
Contributions

1. identify reconfiguration opportunities
   - partitioning algorithm: generate configurations based on variations in time dimension

2. analyse runtime benefits for stencil computation
   - analytical model: optimise generated partitions, to fully utilise available resources

3. evaluate run-time solutions
   - Scheduling algorithm: evaluate run-time benefits and overhead, to provide optimal run-time solution
Stencil Computation: Reverse Time Migration (RTM)

\[ \frac{\partial^2 p}{\partial t^2} = v^2 \nabla^2 p + s(t) \]

for \( t = 1 \) to \( t_{\text{max}} \):
  for \( i = 0 \) to \( X \times Y \times Z - 1 \):
    \( l = \text{convolve}(\text{curr}[i], \text{stencil}) \)
    \( \text{next} := 2 \times \text{curr}[i] - \text{prev}[i] + \text{vv}[i] \times l \)
    \( \text{next}[i] := \text{next} + \text{source}[i] \)
    apply\_boundary\_condition(\text{curr}, \text{next})
    swap\_buffers(\text{prev}, \text{curr}, \text{next})

Example: Oil & Gas Application

Forward propagation

Backward propagation

correlation
1. Partitioning algorithm: functions to segments

- segments
  - functions in same data dependency level combined as a segment
  - segment variations express algorithm variations in time dimension
Partitioning algorithm: segments to configurations

- **segments**
  - functions in same data dependency level combined as a segment
  - segment variations express algorithm variations in time dimension

- **configurations**
  - one or more segments are combined as a valid configuration to be executed
  - optimised configurations utilise run-time benefits
Partitioning algorithm: partitions

• segments
  – functions in same data dependency level combined as a segment
  – segment variations express algorithm dimension

• configurations
  – one or more segments are combined as a configuration to be executed
  – optimised configurations utilise run-time benefits

• partitions
  – one or more configurations are coordinated to accomplish application tasks
  – optimal partition balances run-time benefits and overhead
Dynamic partitioning: algorithm
2. Analytical model: data-paths

[Diagram of operations on X-, Y-, and Z-dimensions with nodes labeled Add, Mult, and with edges connecting them.]
Analytical model: error and resource consumption

- fully pipelined data-paths
- bit-width optimisation, arithmetic operation transform

![Graph showing error and resource consumption versus number of mantissa bits.](image)
Analytical model: FPGA resources

- fully pipelined data-paths
- bit-width optimisation, arithmetic operation transform
Analytical model: FPGA resource consumption

- fully pipelined data-paths
- bit-width optimisation and arithmetic operation transformation
- accumulating resource consumption

\[
D_s = \frac{\sum N_{op,i} \cdot B_D \cdot T_{D,i}}{A_D}
\]

\[
L_s = \frac{\sum (N_{op,i} \cdot B_L \cdot T_{L,i}) + I_L}{A_L}
\]

\[
F_s = \frac{\sum (N_{op,i} \cdot B_F \cdot T_{F,i}) + I_F}{A_F}
\]

DSP consumption

LUT consumption

FF consumption
• customised memory architecture
Analytical Model: Memory Systems

• customised memory architecture
• data access blocking
Analytical model: memory systems

- customised memory architecture
- analysing resource / bandwidth requirements

memory resource consumption

\[ B_S = \sum_{1}^{P_{knl}} P_t \cdot P_{dp} \cdot m_{dp} \cdot \left( S \cdot (2 + N_c) + 1 \right) \]
\[ \frac{A_B}{(W_{dp} \cdot B_w)} \]

\[ m_{dp} = \frac{nx + (P_t - 1) \cdot 2S}{P_{dp}} \cdot (ny + (P_t - 1) \cdot 2S) \]
**Analytical model: memory systems**

- customised memory architecture
- analysing resource / bandwidth requirements

\[
B_S = \sum_{1} P_{knl} \cdot P_t \cdot P_{dp} \cdot m_{dp} \cdot \left( S \cdot \left( 2 + N_c \right) + 1 \right) \left/ (A_B \cdot W_{dp} \cdot B_w) \right.
\]

\[
m_{dp} = \frac{nx + \left( P_t - 1 \right) \cdot 2S}{P_{dp}} \cdot \left( ny + \left( P_t - 1 \right) \cdot 2S \right)
\]

off-chip bandwidth requirement

\[
BW_m \geq (W_{dp} \cdot B_w \cdot P_{dp}) \cdot P_{knl} \cdot f_{knl}
\]

\[
W_m = N \cdot (W_{dp} \cdot B_w \cdot P_{dp}) \quad N \in \{1, 2, 3, ... \}
\]
Analytical model: design scalability

- parallel data-paths: multiple memory access
- serial: multiple time steps
Analytical model: optimisation

• Objective

Minimise:

$$C_t = R_{c/c} \cdot \frac{D \cdot O_b \cdot O_t}{f_{knl} \cdot P_{knl} \cdot P_{dp} \cdot P_t}$$

$$O_b = \frac{x - 2 \cdot S}{nx - 2 \cdot S} \cdot \frac{y - 2 \cdot S}{ny - 2 \cdot S} \cdot nx \cdot ny$$

$$nx = \frac{x - 2 \cdot S}{\alpha} + 2 \cdot S$$
$$ny = \frac{y - 2 \cdot S}{\beta} + 2 \cdot S$$

$$O_t = \left\{ \begin{array}{ll} \frac{nx + (P_t - 1) \cdot 2 \cdot S}{nx} \cdot \frac{ny + (P_t - 1) \cdot 2 \cdot S}{ny} & \alpha \cdot \beta = 1 \\ 1 & \alpha \cdot \beta \neq 1 \end{array} \right.$$
Analytical model: optimisation

- **Objective**
- **Constraints**

Minimise:

\[
C_t = R_{c/c} \cdot \frac{D \cdot O_b \cdot O_t}{f_{knl} \cdot P_{knl} \cdot P_{dp} \cdot P_t}
\]

Subject to:

1. \[1 \geq D_s = \frac{\sum N_{op,i} \cdot B_D \cdot T_{D,i}}{A_D}
\]

2. \[1 \geq L_s = \frac{\sum (N_{op,i} \cdot B_L \cdot T_{L,i}) + I_L}{A_L}
\]

3. \[1 \geq F_s = \frac{\sum (N_{op,i} \cdot B_F \cdot T_{F,i}) + I_F}{A_F}
\]

\[
O_b = \frac{x-2 \cdot S}{nx-2 \cdot S} \cdot \frac{y-2 \cdot S}{ny-2 \cdot S} \cdot nx \cdot ny
\]

\[
x = \frac{x - 2 \cdot S}{\alpha} + 2 \cdot S
\]

\[
y = \frac{y - 2 \cdot S}{\beta} + 2 \cdot S
\]

\[
O_t = \left\{\begin{array}{l}
\frac{nx+(P_t-1) \cdot 2 \cdot S}{nx} \cdot \frac{ny+(P_t-1) \cdot 2 \cdot S}{ny} \quad \text{if } \alpha \cdot \beta = 1 \\
\frac{nx+(P_t-1) \cdot 2 \cdot S}{nx} \cdot \frac{ny+(P_t-1) \cdot 2 \cdot S}{ny} \quad \text{if } \alpha \cdot \beta \neq 1
\end{array}\right.
\]

\[
1 \geq B_s = \frac{\sum P_{knl} \cdot P_t \cdot P_{dp} \cdot m_{dp} \cdot (S \cdot (2 + N_c) + 1)}{A_B / (W_{dp} \cdot B_w)}
\]

\[
BW_m \geq (W_{dp} \cdot B_w \cdot P_{dp}) \cdot P_{knl} \cdot f_{knl}
\]
Analytical model: optimisation

- Objective
- Constraints
- Overhead
  - reconfiguration time
  - data transfer time

\[ C_{re} = \frac{\gamma \cdot \text{Max}(B_s, D_s, L_s, F_s)}{\theta} \]
\[ C_{m} = \frac{2 \cdot D \cdot W_{dp} \cdot (1 + N_c) \cdot B_w}{\phi} \]
3. Reconfiguration scheduling

Algorithm 3 Partition Scheduling Algorithm.

Labels: \( v_i \): nodes, \( p_i \): partitions, Cur: current configuration

Functions Conf\((v_i, p_i)\): find the configuration in partition \( p_i \) that \( v_i \in c_i \)

1: for \( p_i \in \) Partitions do
2: \hspace{1em} for \( v_i \in \) Source Nodes do
3: \hspace{2em} Cur \leftarrow \text{Conf}(v_i, p_i)
4: \hspace{2em} while \( v_i \).NextNode \neq \emptyset \) do
5: \hspace{3em} if \( v_i \not\in \text{Cur} \) then
6: \hspace{4em} Cur \leftarrow \text{Conf}(v_i, p_i)
7: \hspace{4em} \( T_{exe} \leftarrow T_{exe} + \text{Cur.Cre} + \text{Cur.Cm} \)
8: \hspace{3em} end if
9: \hspace{2em} \( T_{exe} \leftarrow T_{exe} + \text{Cur.Ct} \)
10: \hspace{2em} \( v_i \leftarrow v_i \).NextNode
11: \hspace{2em} end while
12: \( p_i \).T \leftarrow \max(T_{exe})
13: \hspace{1em} end for
14: Partition \leftarrow \min(p_i.T)
15: end for
Results: model performance

- fully utilise available resources
Results: model performance

- fully utilise available resources
- model accuracy
Results: model performance

- fully utilise available resources
- model accuracy
- approximating peak performance
Results: optimal reconfiguration solution

- improved system performance
  - 1.59 times faster than the best published GPU and FPGA results
### Results: optimal reconfiguration solution

#### Improved system performance

- 1.59 times faster than the best published GPU and FPGA results
- 1.45 times faster than optimized static implementation

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FPGA (static)</th>
<th>FPGA (optimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data size</strong></td>
<td>s</td>
<td>m</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>execution time (t)</td>
<td>181.7</td>
<td>1458.2</td>
<td>5574.8</td>
<td></td>
</tr>
<tr>
<td>overhead time (t)</td>
<td>0.03</td>
<td>0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>throughput (GFlop/s)</td>
<td>1.8</td>
<td>0.9</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>speed-up</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>n/a</td>
</tr>
<tr>
<td>power (W)</td>
<td>182</td>
<td>185</td>
<td>183</td>
<td>461</td>
</tr>
<tr>
<td>energy ($10^3$J)</td>
<td>33</td>
<td>269</td>
<td>1020</td>
<td>n/a</td>
</tr>
<tr>
<td>efficiency ((MFlop/s)/W)</td>
<td>9.8</td>
<td>4.9</td>
<td>9.8</td>
<td>76.5</td>
</tr>
<tr>
<td>efficiency gains</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>n/a</td>
</tr>
</tbody>
</table>

- Improved system performance
- 1.59 times faster than the best published GPU and FPGA results
- 1.45 times faster than optimized static implementation
### Results: optimal reconfiguration solution

- Improved system performance:
  - 1.59 times faster than the best published GPU and FPGA results
  - 1.45 times faster than optimized static implementation
  - 1.42 times more energy efficient than the static designs

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FPGA (static)</th>
<th>FPGA (optimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution time (s)</td>
<td>181.7</td>
<td>1458.2</td>
<td>5574.8</td>
<td></td>
</tr>
<tr>
<td>overhead time (s)</td>
<td>0.03</td>
<td>0.01</td>
<td>0</td>
<td>0.14 0.58 3.88</td>
</tr>
<tr>
<td>throughput (GFlop/s)</td>
<td>1.8</td>
<td>0.9</td>
<td>1.8</td>
<td>35.8 70.6 68.0</td>
</tr>
<tr>
<td>speed-up</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>n/a 39.2x 76.4x</td>
</tr>
<tr>
<td>power (W)</td>
<td>182</td>
<td>185</td>
<td>183</td>
<td>0.5 2.3 18.3</td>
</tr>
<tr>
<td>energy (10^3J)</td>
<td>33</td>
<td>269</td>
<td>1020</td>
<td>0.4 1.7 14.6</td>
</tr>
<tr>
<td>efficiency ((MFlop/s)/W)</td>
<td>9.8</td>
<td>4.9</td>
<td>9.8</td>
<td>5.51 52.7 538.9</td>
</tr>
<tr>
<td>efficiency gains</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>n/a 56.7x 108.4x</td>
</tr>
</tbody>
</table>

---
Future work

• generalise design methods
• partial reconfiguration
• run-time scheduling
• further applications
Conclusion

• design method to exploit run-time reconfiguration
  – partitioner
  – analytical model
  – scheduler

• improved system performance
  – 1.59 times faster than the best published GPU and FPGA results
  – 1.45 times faster than optimized static implementation
  – 1.42 times more energy efficient than the static designs