Automatic Support for Multi-Module Parallelism from Computational Patterns

_Nithin George_, David Novo, Muhsen Owaida and Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL),
School of Computer and Communication Sciences

_HyoukJoong Lee and Kunle Olukotun_
Stanford University, Pervasive Parallelism Laboratory

_David Andrews_
University of Arkansas, Department of Computer Science and Computer Engineering.
FPGAs for Application Developers

High performance:
Microsoft accelerates Bing search using FPGAs\(^1\)

Improved access:
Intel plans to integrate FPGAs with their Xeon processors\(^2\)

Lack of hardware knowledge:
HLS tools use functional specifications, but need hardware design knowledge to produce good quality designs

Can we produce high-performance FPGA designs without hardware design knowledge?

\(^1\) Burger et al, “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services” in *ISCA 2014*

\(^2\) Clark et al, “Intel Agrees to Buy Altera for $16.7 Billion,” in *WSJ 2015*
Hardware Generation using Patterns

Application $\rightarrow$ High-level, Functional Spec.

Tool Flow

$\rightarrow$ FPGA Bitstream

FPGA

Hardware Generation using Patterns

Application

High-level, Functional Spec.

High-Level Compiler

Delite

Control-Flow Graph

Sequential Patterns

Parallel Patterns

FPGA Bitstream

FPGA

Hardware Generation using Patterns

High-level, Functional Spec.

Application

Delite

High-Level Compiler

Sequential Patterns

Parallel Patterns

Control-Flow Graph

Hardware Generation using Patterns

Application

High-level, Functional Spec.

Delite

High-Level Compiler

Control-Flow Graph

Sequential Patterns

Parallel Patterns

$z = \log(a)$

$z = a + b$

$z = a \times b$

Hardware Generation using Patterns

Application

High-level, Functional Spec.

Delite

High-Level Compiler

Sequential Patterns

Parallel Patterns

Control-Flow Graph

Application’s Inter-pattern Dependencies

Hardware Generation using Patterns

Application → High-Level Compiler
            ^ Delite
              |         |
              v         v
Control-Flow Graph → Sequential Patterns → Hardware Module Synthesis

Standard FPGA Tools → System Design

FPGA Bitstream → FPGA

Hardware Generation using Patterns

Application

High-Level Compiler

Delite

Control-Flow Graph

Sequential Patterns

Processor Program Generation

Software Compiler

Hardware Module Synthesis

System Design

FPGA Bitstream

FPGA

Hardware System

Sequential Operations & Scheduling

Processor (soft-core) → Shared Internal Memory → Shared External Memory

Parallel Operations

Local Buffer

Wide System Bus
Meet the Patterns—Parallel Patterns

- **Map**: e.g., $Z = A + 2$
- **ZipWith**: e.g., $Z = A + B$
- **Reduce**: e.g., $z = \text{sum}(A)$
- **Foreach**: e.g., $A = \log(A)$

Collections (e.g., vectors, matrices, arrays)
Matrix Multiplication in Patterns

A

\[
\begin{bmatrix}
0 \\
i \\
M
\end{bmatrix}
\]

B

\[
\begin{bmatrix}
0 & j & N
\end{bmatrix}
\]

C

\[
\begin{bmatrix}
(i, j)
\end{bmatrix}
\]

ZipWith \( \times \) Reduce

11
Matrix Multiplication in Patterns

\[
\begin{align*}
\text{Reduce} & + \quad \text{Map} \quad (0 \to N) \\
\text{Map} & \quad (0 \to M) \\
\text{ZipWith} & \quad \times
\end{align*}
\]

A

B

C

(\(i,j\))
Single-Module Performance

![Graph showing single-module performance with execution time on the y-axis and logic utilization on the x-axis. The graph includes points labeled U:8, U:32, U:64, and U:96. The graph is titled "Single-Module Design."
Matrix Multiplication Performance

Matrix A

Matrix B

Matrix C

Burst

Map (0 -> N)

Map (0 -> M)

ZipWith x

Reduce +

Individual Reads

(i, j)

⋯
Multi-Module Performance

![Graph showing the comparison between Multi-Module Design and Single-Module Design in terms of execution time and logic utilization. The graph illustrates a significant reduction in execution time as the number of modules increases.](image-url)
Multi-Module Performance

We want to generate multi-module designs automatically from the same specifications!
Outline

• Methodology
  – Synchronization Requirements
  – Work partitioning
  – System configuration

• Results
Synchronization Requirements

Simple Patterns

Reduce

Map, ZipWith or Foreach

Data Race

Bus Width

False Sharing
Synchronization Requirements

Fused Patterns (Special Case)
Synchronization Requirements

Nested Patterns

Reduce

✓

Reduce

✓
Synchronization Requirements

Nested Patterns

Map or ZipWith

✓

Map or ZipWith

✓
Synchronization Requirements

Nested Patterns

Foreach

Foreach

0x4  

\text{A} \quad \text{D}

False Sharing
Reducing Synchronization

Data Race

Private copy

Only once!

Frequent updates

Data Race
Reducing Synchronization
Reducing Synchronization

Map, ZipWith or [Foreach]

False Sharing

Map, ZipWith or [Foreach]

False Sharing

Map, ZipWith or [Foreach]

Map, ZipWith or [Foreach]

Bus Width

A-H

I-P

False Sharing

A
B
C
D
E
F
G
H
I
J
K

0x0
0x4
0x8
0xC
0x10
0x14
Reducing Synchronization

Map, ZipWith or [Foreach]

0x4

A

D

False Sharing

Map, ZipWith or [Foreach]

Map, ZipWith or [Foreach]

Map, ZipWith or [Foreach]

A-H

I-P

Bus Width

0x0

0x4

0x8

0xC

0x10

0x14

A

B

C

D

E

F

G

H

I

J

K

L

✓

✓

False Sharing

Map, ZipWith or [Foreach]

Map, ZipWith or [Foreach]

Map, ZipWith or [Foreach]
Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory

Shared External Memory

Wide System Bus

Parallel Operations

Local Buffer

Matrix Multiply

Local Buffer

Matrix Multiply

Mutex

M0

M1
Outline

• Methodology
  – Synchronization Requirements
  – Work partitioning
  – System configuration

• Results
Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory

Task Pool
Start
End

Shared External Memory

Wide System Bus

Parallel Operations

Local Buffer
Matrix Multiply

M0

Local Buffer
Matrix Multiply

M1

Mutex
Multi-Module Matrix Multiply

Sequential Operations & Scheduling
- Processor (soft-core)

Shared Internal Memory
- Task Pool
  - Start: 0
  - End: 100

Shared External Memory
- Matrices: A, B, C

Parallel Operations
- Local Buffer
- Matrix Multiply
- M0
- Mutex
- M1

Wide System Bus
Multi-Module Matrix Multiply

Sequential Operations & Scheduling
Processor (soft-core)

Shared Internal Memory
Task Pool
Start 20 → End 100

Shared External Memory
Matrices
A B C

Start signal

Local Buffer
Matrix Multiply

M0
0 → 19

Mutex

Wide System Bus

Parallel Operations

Local Buffer
Matrix Multiply

M1

Task Pool

Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory

Task Pool
Start 40 → End 100

Shared External Memory

Matrices
A B C

Parallel Operations

Wide System Bus

Local Buffer
Matrix Multiply
M0 0 → 19

Mutex
M1 20 → 39

Local Buffer
Matrix Multiply
M1

Start 40 → End 100

Shared Internal Memory

Task Pool
Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory
Task Pool
Start 40 → End 100

Shared External Memory
Matrices: A, B, C

Wide System Bus

Parallel Operations

Local Buffer
Matrix Multiply

Mutex

Internal Memory
Shared External Memory

Start
End

M0: 0 → 19
M1: 20 → 39

Task Pool

40
100

Matrices

A
B
C
Multi-Module Matrix Multiply

Sequential Operations & Scheduling
Processor (soft-core)

Shared Internal Memory
Task Pool
Start 40 → End 100

Shared External Memory
Matrices
A B C

Local Buffer
Matrix Multiply
M0 0 → 19

Local Buffer
Matrix Multiply
M1

Mutex

Parallel Operations

Wide System Bus

40 100

End
Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory
Task Pool
Start 60 → End 100

Shared External Memory
Matrices
A B C

Wide System Bus

Parallel Operations

Local Buffer
Matrix Multiply

M0
0 → 19

M1
40 → 59

Mutex

M1

Multi-Module Matrix Multiply

Sequential Operations & Scheduling

Processor (soft-core)

Shared Internal Memory
Task Pool
Start 60 → End 100

Shared External Memory
Matrices
A
B
C

Wide System Bus

Parallel Operations

Local Buffer
Matrix Multiply
M0

Local Buffer
Matrix Multiply
M1

M0
Mutex

Start 40 → End 59
Outline

• Methodology
  – Synchronization Requirements
  – Work partitioning
  – System configuration

• Results
**Multiple modules** per application

**Multiple variants** per module

**Multiple copies** per variant
System Configuration

Objective: \( \text{max}(\text{perf}(\text{System})) \)

Constraints:
- \( \text{resources}(A) + \text{resources}(B) \ldots \leq \text{resources}(\text{FPGA}) \)
- \( \text{bandwidth}(A) \leq \text{System Bus Bandwidth} \)
- \( \text{bandwidth}(B) \leq \text{System Bus Bandwidth} \)

... 

Integer Linear Programming (ILP) to find the solution
Experimental Setup

Virtex 7 VX485T FPGA Device
76K Slices
8Kb Distributed RAM
2.8K DSP Units
37Kb BRAM Units

External Memory: 1GB
On-chip Memory: 64KB
### Benchmark Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Parallel Ops.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication (simple)</td>
<td>3</td>
</tr>
<tr>
<td>Matrix Multiplication (opt)</td>
<td>3</td>
</tr>
<tr>
<td>Autocorrelation</td>
<td>5</td>
</tr>
<tr>
<td>PageRank</td>
<td>6</td>
</tr>
<tr>
<td>Potential Friends</td>
<td>6</td>
</tr>
<tr>
<td>Triangle Count</td>
<td>3</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>9</td>
</tr>
</tbody>
</table>

#### Single-Module Designs

- **Soft-core Processor**
  - Hardware Modules
    - A
    - B
    - C

#### Multi-Module Designs

- **Soft-core Processor**
  - Hardware Modules
    - A
    - B
    - C
Performance Comparison

Lower is better

Applications

Execution Time (s)

Single-Module Design
Multi-Module Design
Conclusions

A methodology to automatically generate multi-module systems from parallel computational patterns

1. Identify synchronization requirements using properties of computational patterns

2. Use task-pools to achieve dynamic load balancing

3. Utilize an ILP solver to find the system configuration

Generates complete hardware design from functional specifications – no hardware design knowledge needed
Hardware Generation using Patterns

Application

High-Level Compiler

Delite

Control-Flow Graph

Sequential Patterns

Soft-Core Program Generation

Parallel Patterns

Hardware Module Synthesis

System Design

Standard FPGA Tools

Software Compiler

FPGA