TAPAS: Generating Parallel Accelerators from Parallel Programs

Steven Margerm¹, Amirali Sharifian¹, Apala Guha¹
Gilles Pokam², Arrvindh Shriraman¹

https://github.com/sfu-arch/tapas

Simon Fraser University¹, Intel Corp.²
Motivation

FPGAs are everywhere

• Lots parallelism
  – 150$ Cyclone V SoC - 60 stencil tasks

• 10s of cycles for invoking a hardware “task”

• Fine-grain parallelism
  – Cyclone V. 512 arithmetic ops
High Level Synthesis

- Mixes schedule and algorithm
  - #pragma
- Static schedule
  - limited concurrency control
- Domain specific templates
  - generalizable?
TAPAS: Auto generating Parallel Dataflow Accelerator

- MIT’s parallel compiler (TAPIR)

- Hardware component library:
  - like UCB Rocket, but for accelerators

- Generator
  - synthesizing RTL from compiler IR

Cilk/Go/OpenMP → TAPAS → Parallel Accelerator
Overview

• HLS Challenge: Static Parallelism

• TAPAS: modular high level synthesis

• TAPAS: generating task units
HLS Challenge: Static Parallelism

for(i = 0 until n){
  if(node[i].valid){
    compute(&node[i]);
  }
}

Loop bound is unknown
Conditional Body
Non-deterministic latency
HLS Challenge: Static Parallelism

Unrolled Program

```c
#pragma UNROLL 2
for(i = 0 until n){
    if(node[i].valid){
        compute(&node[i]);
    }
}
```

Worst case schedule —> Low utilization
Our Approach: Dynamic Parallelism

Task Program

```c
for(i = 0 until n){
    if(node[i].valid){
        Spawn compute(&node[i]);
    }
} Sync;
```

Task Hardware

```
for (; ; if (valid)
    Spawn compute()
Sync
```

Run time schedule → High utilization
Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator
Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

for(i = 0 until n){
    if(node[i].valid){
        compute(&node[i])
    }
}


Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

Static Task Graph

Root

Child
Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

Task Execution Unit

Root

Child

for (; ; if (valid)

Spawn

Sync

compute
cilk_for(i = 0 until n){
cilk_for(j = 0 until n){
    c[i][j] = a[i][j]+b[i][j];
}
}

- Parallel Compiler
  - Captures Spawn and Sync from IR
- Task Extractor:
  - Wraps each task in a first class entity
Task-Level Architecture

Heterogeneous!  Nested Parallel!  Asynchronous!
Task Level Execution

T0
   for_i

T1
   for_j

T2
   body

T0

T1

Parent ID | Child
----------|-------
Root     | 0

Parent ID | Child
----------|-------
R         |
Task Level Execution

T0 \(\text{for}_i\)

T1 \(\text{for}_j\)

T2 \text{body}

T0

T1

T0

Spawn

Parent ID

Child

E

Root

0

Parent ID

Child

R


Task Level Execution

T0: for_i

T1: for_j

T2: body
Task-Level Execution

T0: for_i

T1: for_j

T2: body

<table>
<thead>
<tr>
<th>Parent ID</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0:0</td>
<td>0</td>
</tr>
<tr>
<td>T0:0</td>
<td>1</td>
</tr>
<tr>
<td>Root</td>
<td>N</td>
</tr>
<tr>
<td>T0:0</td>
<td>M-1</td>
</tr>
</tbody>
</table>

Spawn

<table>
<thead>
<tr>
<th>Parent ID</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0:0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task Level Execution

T0

for_i

T1

for_j

T2

body

<table>
<thead>
<tr>
<th>Parent ID</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>N</td>
</tr>
<tr>
<td>T0:0</td>
<td>M</td>
</tr>
<tr>
<td>T0:0</td>
<td>0</td>
</tr>
<tr>
<td>T1:0</td>
<td>0</td>
</tr>
</tbody>
</table>
Task Level Execution

T0
  for_i
T1
  for_j
T2
  body

Sync

<table>
<thead>
<tr>
<th>Parent ID</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>N</td>
</tr>
<tr>
<td>T1:0</td>
<td>M</td>
</tr>
<tr>
<td>T1:0</td>
<td>0</td>
</tr>
</tbody>
</table>
Task Level Execution

\[
\begin{align*}
&T_0: \text{for}_i \\
&T_1: \text{for}_j \\
&T_2: \text{body}
\end{align*}
\]
Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?

**Dedup**

![Task Graph Diagram]
Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?

**Dedup**

Regulare task pipeline
Supporting Dynamic Task Graphs

• What does task pipelining look like in TAPAS?

Dedup

Irregular task pipeline
Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?

Dedup

TAPAS supports recursive tasks as well (Paper:⌘5)
Compilation Flow

Cilk/Go

- Task extraction

Task Graph Representation

- Top RTL Generation

Task-Level RTL

- Generate TXUs

TAPAS Accelerator

Task Execution Unit

Root

for (; ; if (valid)

Child

compute

Spawn

Sync
Task Execution Unit (TXU)

• What are the elements inside each TXU?

Dynamic issue

Task Queue

Asynchronous!

Available task!

Multi-core

compute

compute

compute

Ready
\[ c[i][j] = a[i][j] + b[i][j]; \]
Task Execution Unit (TXU)

c[i][j] = a[i][j]+b[i][j];
Task Execution Unit (TXU)

c[i][j] = a[i][j] + b[i][j];
Task Execution Unit (TXU)

c[i][j] = a[i][j] + b[i][j];

i = 0, j = 3

i = 0, j = 2

i = 0, j = 1

i = 0, j = 0
Experiment

• Board:
  – Arria 10 SOC
  – Intel core i7

• Execution time reported
  – Number of Cycles

• Goal:
  – Performance/watt improvement
  – Reducing overhead of spawning tasks with few instructions
How does performance scale with workload size?

• Unlike a CPU, FPGA performance scales with #TXU even for fine grained parallelism.

![Graph showing performance (Million Adds/s) vs. number of tiles for different workloads (10, 20, 40 Ops) in software and hardware (1, 2, 3, 4 Tile).](image)
Does performance scale with recursion?

- Performance scales with recursive algorithms

<table>
<thead>
<tr>
<th>Normalized Performance</th>
<th>1 Tile</th>
<th>2 Tiles</th>
<th>4 Tiles</th>
<th>8 Tiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stencil</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(n=15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mergesort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= 1
How does performance compare to CPU?

- Performance gain compare to a Intel core i7

(> 1) = FPGA Faster

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Add</td>
<td>1.2</td>
</tr>
<tr>
<td>Stencil</td>
<td>0.81</td>
</tr>
<tr>
<td>Saxpy</td>
<td>0.59</td>
</tr>
<tr>
<td>Dedup</td>
<td>3.21</td>
</tr>
<tr>
<td>fib(n=15)</td>
<td>0.61</td>
</tr>
<tr>
<td>Average</td>
<td>0.9</td>
</tr>
</tbody>
</table>
How does Performance/Watt compare to CPU?

- Performance/Watt has significant improvement
What is the overhead of task controller?

• ALM Utilization by Sub-block

![Pie chart showing ALM utilization with labeled percentages for different sub-blocks: Tiles, Parallel for, Task Ctrl, Mem Arb, Misc.](image)

1 Tile / 1 Ins:
- Tiles: 37%
- Parallel for: 40%
- Task Ctrl: 9%
- Mem Arb: 2%
- Misc: 11%

10 Tile / 1 Ins:
- Tiles: 76%
- Parallel for: 7%
- Task Ctrl: 13%
- Mem Arb: 2%
- Misc: 1%
Available now

https://github.com/sfu-arch/tapas

Thanks Chisel and Tapir folks

Shout out to related…

• An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware (MICRO51)
• Dynamically scheduled high-level synthesis (FPGA18)
Parametrization and Configuration

- TAPAS generated accelerator is *Parametrizable and Configurable*.
  - The number of TXUs can be set specifically for each task base on different criteria.
  - Datapath width can be set at this phase, supporting *mixed precision* as well.
  - Memory modules within each Task Unit are configurable like scratchpads, network and cache.