Stitch: Fusible Heterogeneous Accelerators Enmeshed with Many-Core Architecture for Wearables

Cheng Tan, Manupa Karunaratne, Tulika Mitra, Li-Shiuan Peh
Emerging Wearables

- Software programmable to support diverse applications

- Health care apps on smart watches
- Here Maps on Samsung gear s2
- Pokemon go on Apple Watch
- Bus stop detection app (user defined) on LG Watch Urban
- Navigation on smart glass
Emerging Wearables

- Software programmable to support diverse applications

- Health care apps on smart watches
- Here Maps on Samsung gear s2
- Pokemon go on Apple Watch
- Performance Requirement (10000 MIPS)
- Navigation on smart glass
- Bus stop detection app (user defined) on LG Watch Urban
Emerging Wearables

- Software programmable to support diverse applications

- Pokemon go on Apple Watch
- Here Maps on Samsung gear s2
- Health care apps on smart watches
- Bus stop detection app (user defined) on LG Watch Urban
- Navigation on smart glass
- Power constraint (500 mW)
- Performance Requirement (10000 MIPS)
Wearable SoC Architecture Trend

Sony Smartwatch1
ARM Cortex-M3

Qualcomm toq,
ARM Cortex-M3

Samsung Gear S
ARM Cortex-A7

Samsung Gear S2
ARM Cortex-A7

Samsung Gear S3
ARM Cortex-A7

Huawei Watch 2
ARM Cortex-A7

Sony Smartwatch2
ARM Cortex-M4

Motorola Moto 360 1st
ARM Cortex-A8

LG G Watch R
ARM Cortex-A7

Motorola Moto 360 2nd
ARM Cortex-A7

Asus Zenwatch 3
ARM Cortex-A7

chronology
Wearable SoC Architecture Trend

Core Count Trend
DMIPS/watt Trend
Power Trend
DMIPS Trend

Core Count
DMIPS/watt
Power
DMIPS

Jan-2013
Nov-2013
Aug-2014
May-2015
Feb-2016
Nov-2016
Sep-2017

Sony Smartwatch1
ARM Cortex-M3
Qualcomm toq, ARM Cortex-M3
Samsung Gear S
ARM Cortex-A7
Samsung Gear S2
ARM Cortex-A7
Samsung Gear S3
ARM Cortex-A7
Huawei Watch 2
ARM Cortex-A7

Sony Smartwatch2
ARM Cortex-M4
Motorola Moto 360 1st
ARM Cortex-A8
LG G Watch R
ARM Cortex-A7
Motorola Moto 360 2nd
ARM Cortex-A7
Asus Zenwatch 3
ARM Cortex-A7

chronology
Motivating Case Study

- Finger gesture recognition application
Motivating Case Study

• Finger gesture recognition application

• State-of-the-art smartwatch
  ➢ Odroid board emulating the state-of-the-art smartwatch
  ➢ Time per gesture: 13 ms $\times > 10$ ms
  ➢ Cannot meet the target throughput

<table>
<thead>
<tr>
<th></th>
<th>4-core ARM Cortex-A7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting throughput</td>
<td>no</td>
</tr>
<tr>
<td>Time per Gesture (ms)</td>
<td>13</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>469</td>
</tr>
<tr>
<td>Frequency</td>
<td>1200</td>
</tr>
<tr>
<td>Technology</td>
<td>28nm</td>
</tr>
</tbody>
</table>
Wearable Application Characteristics

*Finger gesture application* (6-stage pipeline, 16 kernels)

Acc/Gyro (X, Y, Z)

Window moving

FFT1
FFT2
FFT3
FFT4
FFT5
FFT6

Update feature

Filter

IFFT1
IFFT2
IFFT3
IFFT4
IFFT5
IFFT6

Classify

Abundant parallelism -> many-core architecture

Memory Controller

page 8
Wearable Application Characteristics

Finger gesture application
(6-stage pipeline, 16 kernels)

Power budget -> simple in-order core

Each tile: 8.75 mW

Memory Controller

In-order CPU
Wearable Application Characteristics

Finger gesture application
(6-stage pipeline, 16 kernels)

Accelerators
(e.g., ASIC, FPGA, CGRA, and
Reconfigurable Functional Unit)

Improve performance/power -> accelerators

Each tile: 8.75 mW

Memory Controller

Acc/Gyro (X, Y, Z)

Window moving
Wearable Application Characteristics

**Finger gesture application**
(6-stage pipeline, 16 kernels)

Different kernels -> heterogeneous accelerators
Wearable Application Characteristics

Finger gesture application
(6-stage pipeline, 16 kernels)

Different kernels -> heterogeneous accelerators

Heterogeneous Accelerator
In-order CPU
Wearable Application Characteristics

*Finger gesture application* (6-stage pipeline, 16 kernels)

**Imbalanced workload -> fusible accelerators**

Compiler decides the fusion of accelerators offline

Actual fusion happens at runtime

**Stitch compiler tool chain**

**Heterogeneous Accelerator**

**In-order CPU**
Stitch Architecture - Overview

- Many-core architecture with simple in-order CPU and accelerator
- Heterogeneous customizable accelerators – *polymorphic patches*
- Patches are able to fuse together to alleviate the bottleneck kernels
- The fusion of patches is directed offline by our compiler tool chain
Patch Architecture

- Heterogeneous customizable accelerators – *polymorphic patches*
- Patch architecture motivated by representative wearable kernels
Patch Architecture

- Heterogeneous customizable accelerators – *polymorphic patches*
- Patch architecture motivated by representative wearable kernels

---

**‘Hot’ patterns**

Multiple rounds of *Longest Common Substring (LCS)* identification

---

Simple computation fragment

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>{AT}</td>
<td>arithmetic + memory access</td>
<td>95.7%</td>
</tr>
<tr>
<td>{MA}</td>
<td>Multiply + arithmetic</td>
<td>47.8%</td>
</tr>
<tr>
<td>{AA}</td>
<td>arithmetic + arithmetic</td>
<td>34.8%</td>
</tr>
<tr>
<td>{AS}</td>
<td>arithmetic + shift</td>
<td>21.7%</td>
</tr>
<tr>
<td>{SA}</td>
<td>shift + arithmetic</td>
<td>21.7%</td>
</tr>
</tbody>
</table>
Patch Architecture

- Heterogeneous customizable accelerators – *polymorphic patches*
- Patch architecture motivated by representative wearable kernels
  - 8 x Acc1 -> {AT-MA}
  - 4 x Acc2 -> {AT-AS}
  - 4 x Acc3 -> {AT-SA}
Patch Architecture

- **AT-MA**
  - ALU, SPM access; Multiplier, ALU

- **AT-SA**
  - ALU, SPM access; Shifter, ALU

- **AT-AS**
  - ALU, SPM access; ALU, Shifter

(a) patch {AT-MA}

(b) patch {AT-SA}

(c) patch {AT-AS}
Patch Architecture

- **AT-MA**
  - ALU, SPM access; Multiplier, ALU

- **AT-SA**
  - ALU, SPM access; Shifter, ALU

- **AT-AS**
  - ALU, SPM access; ALU, Shifter

- T indicates the memory access operation
- A scratchpad memory is attached beside the CPU
- Both CPU and accelerator can access the SPM
Patch Architecture

- **AT-MA**
  - ALU, SPM access; Multiplier, ALU

- **AT-SA**
  - ALU, SPM access; Shifter, ALU

- **AT-AS**
  - ALU, SPM access; ALU, Shifter
• Single *patch* accelerates DFG

(a) Data flow graph
Mapping Computation to Patches

- Single patch accelerates DFG
  - By \{AT-MA\}: 4 cycles

(a) Data flow graph

(b) Accelerated by patch \{AT-MA\}
Mapping Computation to Patches

- Single *patch* accelerates DFG
  - By \{AT-MA\}: 4 cycles
  - By \{AT-AS\}: 2 cycles

(a) Data flow graph

(b) Accelerated by patch \{AT-MA\}

(c) Accelerated by patch \{AT-AS\}
Mapping Computation to Patches

• Single patch accelerates DFG
  - By {AT-MA}: 4 cycles
  - By {AT-AS}: 2 cycles

• Fused patch accelerates DFG
  - {AT-AS} ∪ {AT-AS}: 1 cycle

(e) Accelerated by a fused patch {AT-AS, AT-AS}
Inter-patch NoC

- **Single patch** accelerates DFG
  - By \{AT-MA\}: 4 cycles
  - By \{AT-AS\}: 2 cycles

- **Fused patch** accelerates DFG
  - \{AT-AS\} ∪ \{AT-AS\}: 1 cycle

- **Fusion** is achieved by a lightweight NoC
  - Crossbar-based; Bufferless; Compiler-scheduled

(a) Data flow graph

![Data flow graph diagram]

![Memory Controller diagram]

![Chip diagram]
Inter-patch NoC

- Multiple-hop stitching per cycle
- Bufferless
- Configure before running the application

![Diagram of inter-patch NoC with patch2, patch6, and patch10 configurations, Crossbar Config Reg, Crossbar Switch, Asynchronous Repeaters, and SPM port.]
Inter-patch NoC

- Multiple-hop stitching per cycle
- Bufferless
- Configure before running the application

4 word-size register values
38-bit size patch configuration

North\textsubscript{in} = South\textsubscript{out}
Finger gesture application
(6-stage pipeline, 16 kernels)

Window moving

Acc/Gyro (X, Y, Z)

1. Bottlenecks identified
2. Kernel mapping
3. Patch fusion

Stitch architecture

cores: core1 core2 • • • core16

patches: patch1 patch2 • • • patch5

Stitch compiler tool chain
Compiler Support Illustrated

- State-of-the-art smartwatch: 13 ms/gesture > 10 ms/gesture

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting throughput</td>
<td>no</td>
</tr>
<tr>
<td>Time per Gesture (ms)</td>
<td>13</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>469</td>
</tr>
<tr>
<td>Frequency</td>
<td>1200</td>
</tr>
<tr>
<td>Technology</td>
<td>28nm</td>
</tr>
</tbody>
</table>

4-core ARM Cortex-A7
**Compiler Support Illustrated**

- State-of-the-art smartwatch: 13 ms/gesture > 10 ms/gesture
- Stitch without fusion: 11.49 ms/gesture

<table>
<thead>
<tr>
<th>Feature</th>
<th>4-core ARM Cortex-A7</th>
<th>Stitch w/o fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting throughput</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Time per Gesture (ms)</td>
<td>13</td>
<td>11.49</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>469</td>
<td>108</td>
</tr>
<tr>
<td>Frequency</td>
<td>1200</td>
<td>200</td>
</tr>
<tr>
<td>Technology</td>
<td>28nm</td>
<td>40nm</td>
</tr>
</tbody>
</table>
Compiler Support Illustrated

- State-of-the-art smartwatch: 13 ms/gesture > 10 ms/gesture
- Stitch without fusion: 11.49 ms/gesture
- Stitch: 7.62 ms/gesture

<table>
<thead>
<tr>
<th>4-core ARM Cortex-A7</th>
<th>Stitch w/o fusion</th>
<th>Stitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting throughput</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Time per Gesture (ms)</td>
<td>13</td>
<td>11.49</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>469</td>
<td>108</td>
</tr>
<tr>
<td>Frequency</td>
<td>1200</td>
<td>200</td>
</tr>
<tr>
<td>Technology</td>
<td>28nm</td>
<td>40nm</td>
</tr>
</tbody>
</table>
Evaluation – Wearable Applications

**APP2 – Image recognition**

```
CONV1,1
CONV1,2
CONV1,3
CONV1,4

POOL1

CONV2,1
CONV2,2
CONV2,3
CONV2,4
CONV2,5
CONV2,6

POOL2

CONV3,1
CONV3,2
CONV3,3

POOL&FC
```

**APP3 – SVM-based machine learning**

```
SVM1
SVM2
SVM3
SVM4
SVM5
SVM6
SVM7
SVM8

scatter

AES 1
AES 2
AES 3
AES 4
AES 5
AES 6

gather
```

**APP4 – Transportation context-detection**

```
DAES 1
DAES 2
DAES 3
DAES 4
DAES 5
DAES 6

DTW1
DTW2
DTW3
DTW4

AES 1
AES 2
AES 3
AES 4
AES 5
AES 6
```
Wearable Applications on Stitch

APP1

APP2

APP3

APP4
Performance

- Gem5 simulation for performance evaluation
- Message passing-based many-core architecture
- Comparing with baseline architecture without acceleration

- 2.3X improvement in terms of throughput
RTL Synthesis of Stitch – Power

- 16-core chip at 40nm technology (Synopsis tools)
- 140 mW at 200 MHz

![Diagram showing normalized power and area efficiency]
Area-Efficiency

- 16-core chip at 40nm technology (Synopsis tools)
- 140 mW at 200 MHz
- Accelerator (patch) area overhead: 0.169 mm$^2$
  - 0.5% of the entire chip

<table>
<thead>
<tr>
<th></th>
<th>LOCUS</th>
<th>Stitch w/o fusion</th>
<th>Stitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual area (μm$^2$)</td>
<td>1,288,044</td>
<td>49,872</td>
<td>168,568</td>
</tr>
<tr>
<td>area (%)</td>
<td>3.68%</td>
<td>0.15%</td>
<td>0.50%</td>
</tr>
</tbody>
</table>

![Tile power breakdown](image1)
![Tile area breakdown](image2)
Comparing with the state-of-the-art wearable SoC Snapdragon W2100 (quad-core ARM Cortex-A7)

- Average 1.65× improvement in terms of throughput
- With only average 27% power consumption
- Average 6.04× improvement in terms of performance/watt
Stitch Conclusion

• We propose Stitch, a many-core architecture where tiny heterogeneous, configurable and fusible accelerators (polymorphic patches) are effectively enmeshed with the cores

  ➢ Each patch can handle very simple custom instructions

  ➢ Multiple polymorphic patches are able to be fused together across the chip to create large, virtual accelerators for complex custom instructions

  ➢ Fusion is achieved by using an ultra-light weight compiler-scheduled network-on-chip without any buffers or control logic

• Improvement:

  ➢ 6.04x vs. quad-core ARM-A7 in terms of performance/watt;

  ➢ 1.77x vs. 16-core baseline architecture in terms of performance/watt;

  ➢ 2.28x vs. 16-core baseline architecture in terms of area-efficiency.
Thanks
Backup Slides
Accelerators with different couplings

- **Loosely coupled accelerators**
  - require local register files, control and data memories, and high data transfer bandwidth
  - High design complexity and area overhead

- **Tightly coupled accelerators**
  - Sharing processor resources (instruction fetch, decode, register file, and even on-chip memory)
  - Consideration of stringent area and power budget
### Related Works

<table>
<thead>
<tr>
<th>Related Works</th>
<th>Different Architectures Incorporated with Reconfigurable Fabrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td><strong>Integration</strong></td>
</tr>
<tr>
<td>RISPP</td>
<td>loose</td>
</tr>
<tr>
<td>Plasticine</td>
<td>loose</td>
</tr>
<tr>
<td>MorphoSys</td>
<td>loose</td>
</tr>
<tr>
<td>EGRA</td>
<td>loose</td>
</tr>
<tr>
<td>BERET</td>
<td>tight</td>
</tr>
<tr>
<td>CCA</td>
<td>tight</td>
</tr>
<tr>
<td>C-Cores</td>
<td>tight</td>
</tr>
<tr>
<td>QsCores</td>
<td>tight</td>
</tr>
<tr>
<td>DySer</td>
<td>tight</td>
</tr>
<tr>
<td>LOCUS</td>
<td>tight</td>
</tr>
<tr>
<td><strong>Stitch</strong></td>
<td>tight</td>
</tr>
</tbody>
</table>
Lightweight Message Passing

- Replace complex power-hungry shared memory cache coherence with explicit message passing.

```
Source Core 1

mutex.lock();
value = (value + 1) * 3;
mutex.unlock();

Destination Core 11

mutex.lock();
value = (value - 2) / 4;
mutex.unlock();
```
Lightweight Message Passing

- Replace complex power-hungry shared memory cache coherence with explicit message passing.

![Diagram showing the process of register-to-register data transfer between Source Core 1 and Destination Core 11. The diagram includes code snippets demonstrating the operations and data flow.]
Lightweight Message Passing

- Larger block size cache-to-cache message passing is supported.

\[ \text{Source Core 1} \]

\[
\text{for (i = 1 to 100) } \quad \text{value}[i] = (\text{value}[i] + 1) \times 3; \\
\text{mutex.lock();} \\
\text{mutex.unlock();}
\]

\[ \text{Destination Core 11} \]

\[
\text{for (i = 1 to 100) } \quad \text{value}[i] = (\text{value}[i] - 2) / 4; \\
\text{mutex.unlock();} \\
\text{mutex.lock();}
\]

\[ \text{Our method} \]

\[ \text{cache-to-cache data transfer} \]

- Less/faster on-chip communication.

\[ \text{NIC} \quad \text{D-cache} \quad \text{In-order core} \quad \text{SFU} \quad \text{LMP} \quad \text{SMART Router} \]

\[ \text{NIC} \quad \text{D-cache} \quad \text{In-order core} \quad \text{SFU} \quad \text{LMP} \quad \text{SMART Router} \]

\[ \text{Shared memory} \]
Compiler Support Illustrated

1) Multi-kernel application -> assembly by the GNU GCC front-end.
2) Profiling each kernel -> bottlenecked kernels and ‘hot’ basic blocks.
3) ‘Hot’ computational patterns (4/2 in/out) -> DFGs.
4) All custom instruction candidates -> mapped onto each patch -> potential speedup accelerated by any patch.
5) Modified GNU Assembler -> new assembly/executable with the patch control signals.
6) Stitching algorithm targeting maximizing overall throughput -> appropriate kernel mapping, version selection, patch stitching, and inter-patch NoC configuration.