Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures

Sitao Huang¹, Li-Wen Chang², Izzat El Hajj³, Simon Garcia De Gonzalo¹, Juan Gómez-Luna⁴, Sai Rahul Chalamalasetti⁵, Mohamed El Hadedy⁶, Dejan Milojicic⁵, Onur Mutlu⁴, Deming Chen¹, Wen-mei Hwu¹

¹University of Illinois at Urbana-Champaign, USA
²Microsoft, USA
³American University of Beirut, Lebanon
⁴ETH Zürich, Switzerland
⁵Hewlett Packard Labs, USA
⁶California State Polytechnic University, Pomona, USA
Outline

• Introduction to Collaborative Computing
• Analytical Models of Collaborative Computing
• Evaluation
  • Evaluation platform
  • Data partitioning vs. task partitioning
  • Impact of kernel replication
• Key Insights
• Chai Benchmarks for CPU-FPGA Systems
Collaborative Computing

• Traditionally, accelerators (GPUs, FPGAs, etc.) have been used as offload engines

Host processor *offloads* computation tasks to accelerators
Collaborative Computing

- Traditionally, accelerators (GPUs, FPGAs, etc.) have been used as offload engines.
- Heterogeneous architectures moving towards tighter integration:
  - Unified memory
  - System-wide atomics

Xilinx Zynq UltraScale+ MPSoC
Collaborative Computing

- Traditionally, accelerators (GPUs, FPGAs, etc.) have been used as offload engines
- Heterogeneous architectures moving towards tighter integration
  - Unified memory
  - System-wide atomics
- Tighter integration allows fine-grained collaboration

**Key challenge:** identify the best CPU-FPGA collaboration strategy
Integrated Heterogeneous Systems

Our vision of an integrated heterogeneous system:
Collaborative Patterns

- data-parallel tasks
- sequential sub-tasks
- coarse-grained synchronization

Program Structure

Device 1  Device 2

Data Partitioning
Collaborative Patterns

Program Structure

- Data-parallel tasks
- Sequential sub-tasks
- Coarse-grained synchronization

Fine-grained Task Partitioning

Device 1

Device 2
Collaborative Patterns

Program Structure

Coarse-grained Task Partitioning

data-parallel tasks

sequential sub-tasks

coarse-grained synchronization

Device 1

Device 2
Data Partitioning

Using Canny Edge Detection (CED) as an example

Input images

Gaussian Filter → Sobel Filter → Non-maximum Suppression → Hysteresis Thresholding → Output images

CPU Implementation
Data Partitioning

Using Canny Edge Detection (CED) as an example

Input images

CPU

FPGA

- Gaussian Filter
- Sobel Filter
- Non-maximum Suppression
- Hysteresis Thresholding

FPGA Acceleration

Output images
Data Partitioning

Using Canny Edge Detection (CED) as an example

Input images

Gaussian Filter
Sobel Filter
Non-maximum Suppression
Hysteresis Thresholding

CPU

Output images (CPU part)

Gaussian Filter
Sobel Filter
Non-maximum Suppression
Hysteresis Thresholding

FPGA

Output images (FPGA part)

CPU-FPGA Collaboration
Data Partitioning vs. Task Partitioning

Using Canny Edge Detection (CED) as an example

Input images

\[ \alpha \]

\[ 1-\alpha \]

CPU

Gaussian Filter

Sobel Filter

Non-maximum Suppression

Hysteresis Thresholding

Output images (CPU part)

FPGA

Gaussian Filter

Sobel Filter

Non-maximum Suppression

Hysteresis Thresholding

Output images (FPGA part)

FPGA

Gaussian Filter

Sobel Filter

Non-maximum Suppression

Hysteresis Thresholding

CPU

Non-maximum Suppression

Hysteresis Thresholding

Output images
Another Data Partitioning Example: Image Histogram

- Input pixels distributed across devices
- Output bins distributed across devices
Analytical Models

- $N$: Number of data parallel tasks in the application
- $t_{↓i}, C$: Execution time of sub-task $i$ by a CPU worker
- $t_{↓i}, F$: Execution time of sub-task $i$ by an FPGA worker
- $w_{↓C}$: Number of available CPU workers
- $w_{↓F}$: Number of available FPGA workers
- $\beta$: Distribution and aggregation overhead factor
- $\alpha$: Fraction of data parallel tasks assigned to CPU
Analytical Models

- \( N \): Number of data parallel tasks in the application
- \( t_{↓i}, C \): Execution time of sub-task \( i \) by a CPU worker
- \( t_{↓i}, F \): Execution time of sub-task \( i \) by an FPGA worker
- \( w_{↓C} \): Number of available CPU workers
- \( w_{↓F} \): Number of available FPGA workers
- \( \beta \): Distribution and aggregation overhead factor
- \( \alpha \): Fraction of data parallel tasks assigned to CPU

\[
\text{Total execution time is: } t_{↓\text{data, total}} = \beta_{↓\text{data}} \cdot \max(\alpha N \sum i \uparrow \beta t_{↓i}, C / w_{↓C}, (1 - \alpha) N \sum i \uparrow \beta t_{↓i}, F / w_{↓F})
\]
Analytical Models

- \( N \): Number of data parallel tasks in the application
- \( t_{\downarrow i}, C \): Execution time of sub-task \( i \) by a CPU worker
- \( t_{\downarrow i}, F \): Execution time of sub-task \( i \) by an FPGA worker
- \( w_{\downarrow C} \): Number of available CPU workers
- \( w_{\downarrow F} \): Number of available FPGA workers
- \( \beta \): Distribution and aggregation overhead factor
- \( \alpha \): Fraction of data parallel tasks assigned to CPU

Data partitioning

The total execution time is

\[
\hat{t}_{\downarrow \text{data, total}} = \beta \hat{t}_{\downarrow \text{data}} \cdot \max \left( \frac{\alpha N \sum i \uparrow \parallel t_{\downarrow i}, C}{w_{\downarrow C}}, \frac{(1-\alpha) N \sum i \uparrow \parallel t_{\downarrow i}, F}{w_{\downarrow F}} \right)
\]

Fixing all the variables except \( \alpha \), the optimal \( \alpha \) (global minimum point) is

\[
\alpha^* = \frac{\sum i \uparrow \parallel t_{\downarrow i}, F}{w_{\downarrow F}} / \left( \frac{\sum i \uparrow \parallel t_{\downarrow i}, C}{w_{\downarrow C}} + \frac{\sum i \uparrow \parallel t_{\downarrow i}, F}{w_{\downarrow F}} \right)
\]
Analytical Models

- \( N \): Number of data parallel tasks in the application
- \( t_{\downarrow i, C} \): Execution time of sub-task \( i \) by a CPU worker
- \( t_{\downarrow i, F} \): Execution time of sub-task \( i \) by an FPGA worker
- \( w_{\downarrow C} \): Number of available CPU workers
- \( w_{\downarrow F} \): Number of available FPGA workers
- \( \beta \): Distribution and aggregation overhead factor

Fine-grained task partitioning

The total execution time is

\[
\begin{align*}
\downarrow t_{\text{task, total}} &= \beta_{\downarrow \text{task}} N \cdot \max \left( \sum_{i \in S_{\downarrow C}} t_{\downarrow i, C} / w_{\downarrow C}, \sum_{i \in S_{\downarrow F}} t_{\downarrow i, F} / w_{\downarrow F} \right) \\
\end{align*}
\]

(Assume sub-tasks are very fine-grained)

Coarse-grained task partitioning

The total execution time is

\[
\begin{align*}
\downarrow t_{\text{task, total}} &= \beta_{\downarrow \text{task}} N \cdot \left( \sum_{i \in S_{\downarrow C}} t_{\downarrow i, C} / w_{\downarrow C} + \sum_{i \in S_{\downarrow F}} t_{\downarrow i, F} / w_{\downarrow F} \right) \\
\end{align*}
\]
Chai Benchmark Suite

**Chai:** Collaborative **H**eterogeneous **A**pplications for **I**ntegrated-architectures

• Chai benchmark suite:

  chai-benchmarks.github.io

  • 14 benchmarks covers data partitioning, fine-grain task partitioning, and coarse-grain task partitioning patterns
  • OpenCL, C++ AMP, and CUDA versions
  • Unified memory and system-wide atomic versions and traditional discrete architecture versions
# Evaluated Chai Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CED-D</td>
<td>Canny Edge Detection</td>
<td>Data Partitioning</td>
</tr>
<tr>
<td>CED-T</td>
<td>Canny Edge Detection</td>
<td>Task Partitioning</td>
</tr>
<tr>
<td>RSC-D</td>
<td>Random Sample Consensus</td>
<td>Data Partitioning</td>
</tr>
<tr>
<td>RSC-T</td>
<td>Random Sample Consensus</td>
<td>Task Partitioning</td>
</tr>
<tr>
<td>BS</td>
<td>Bézier Surface</td>
<td>Data Partitioning</td>
</tr>
<tr>
<td>HSTO</td>
<td>Image Histogram</td>
<td>Data Partitioning</td>
</tr>
<tr>
<td>SSSP</td>
<td>Single-Source Shortest Path</td>
<td>Task Partitioning</td>
</tr>
<tr>
<td>TQ</td>
<td>Task Queue System (Synthetic)</td>
<td>Task Partitioning</td>
</tr>
<tr>
<td>TQH</td>
<td>Task Queue System (Histogram)</td>
<td>Task Partitioning</td>
</tr>
</tbody>
</table>

OpenCL-D (OpenCL discrete architecture) versions of these benchmarks are used.
## Evaluation Platforms

<table>
<thead>
<tr>
<th></th>
<th>Platform A</th>
<th>Platform B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA Board</td>
<td>Terasic DE5-Net</td>
<td>Nallatech 510T</td>
</tr>
<tr>
<td>FPGA Chip</td>
<td>Intel Stratix V GX</td>
<td>Intel Arria 10 GX</td>
</tr>
<tr>
<td>On-Board Memory</td>
<td>4 GB (DDR3)</td>
<td>8 GB (DDR4)</td>
</tr>
<tr>
<td>Host CPU</td>
<td>Intel Xeon E3-1240 v3</td>
<td>Intel Xeon E5-2650 v3</td>
</tr>
<tr>
<td>Host Memory</td>
<td>8 GB (DDR3)</td>
<td>96 GB (DDR4)</td>
</tr>
<tr>
<td>Interface</td>
<td>PCIe gen3.0 x8</td>
<td>PCIe gen3.0 x8</td>
</tr>
</tbody>
</table>
Intel OpenCL SDK for FPGA

- Intel OpenCL SDK for FPGA is used to compile and synthesize host executable and FPGA design.
Compute Unit Replication

- OpenCL kernels are synthesized to compute units on FPGA
- The compute units on FPGA can be replicated by adding `num_compute_units` attribute in the OpenCL kernel code
- `num_compute_units` attribute modifies the number of compute units to which work-groups can be scheduled, which also modifies the number of times a kernel accesses global memory
- We evaluate the impact of compute unit replication
Evaluation: Canny Edge Detection (Data Partitioning)

- C: CPU; F: FPGA
- $\alpha$: Fraction of data parallel tasks assigned to CPU

Dynamic data partitioning: assigning batch of data parallel tasks to idle devices, achieving dynamic workload balance.
Evaluation: Canny Edge Detection (Data Partitioning and Task Partitioning)

```
+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
|                | CPU            | FPGA           | Data           | Task           | CPU            | FPGA           | Data           | Task           |
|                | Stratix V      | Arria 10       |                |                | Stratix V      | Arria 10       |                |                |
+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
| Idle           | 0.1            | 0.2            | 0.3            | 0.4            | 0.1            | 0.2            | 0.3            | 0.4            |
| Copy           | 0.03           | 0.02           | 0.01           | 0.01           | 0.03           | 0.02           | 0.01           | 0.01           |
| Compute        | 0.07           | 0.05           | 0.09           | 0.09           | 0.07           | 0.05           | 0.09           | 0.09           |
+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
Evaluation: Random Sample Consensus (Data Partitioning and Task Partitioning)
Bézier Surface (BS, Data Partitioning)
Histogram (HSTO, Output Data Partitioning)

Execution Time (ms)

- Deallocation
- Copy Back and Merge
- Kernel
- Copy To Device
- Initialization
- Allocation

Execution Time (ms) vs. \(\alpha\): 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
We evaluated the performance under different kernel replication factors.
Kernel Replication – Task Partitioning

We evaluated the performance under different kernel replication factors.
Impact of Replication

Canny Edge Detection

- Replication factor for this application has little impact on performance
- Further profiling reveals the reason of performance saturation is the saturation of the memory bandwidth
- Task partitioning can afford a larger replication factor
Impact of Replication

Random Sample Consensus

- Replication improves performance of this application
- Bounding resource: DSP blocks
- Task partitioning releases the pressure on DSP block and thus can afford a larger replication factor
Key Insights

- Collaborative execution is beneficial
- Data partitioning requires careful choice of partitions to provide the highest performance
- Task partitioning generally enables more kernel replication on the FPGA than data partitioning
- Data partitioning inflicts less burden on programmers and has less communication overhead than task partitioning
- OpenCL stack provides a convenient programming model while there is still room for better programmability and higher performance
Chai Project

• Papers:
  • Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures. *ICPE’19*. (this work)
  • Collaborative Computing for Heterogeneous Integrated Systems. *ICPE’17 Vision Track*.
  • Chai: Collaborative Heterogeneous Applications for Integrated-architectures. *ISPASS’17*.

• Chai Benchmark Suite:
  • Website: chai-benchmarks.github.io
  • Code: github.com/chai-benchmarks/chai
  • Online Forum: groups.google.com/d/forum/chai-dev
Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures

Sitao Huang¹, Li-Wen Chang², Izzat El Haji³, Simon Garcia De Gonzalo¹, Juan Gómez-Luna⁴, Sai Rahul Chalamalasetti⁵, Mohamed El Hadedy⁶, Dejan Milojicic⁵, Onur Mutlu⁴, Deming Chen¹, Wen-mei Hwu¹

¹University of Illinois at Urbana-Champaign, USA
²Microsoft, USA
³American University of Beirut, Lebanon
⁴ETH Zürich, Switzerland
⁵Hewlett Packard Labs, USA
⁶California State Polytechnic University, Pomona, USA

Thanks!