FPGA-based Evaluation Platform for Disaggregated Computing

Dimitris Theodoropoulos, Nikolaos Alachiotis, and Dionisis Pnevmatikatos
Outline

- Introduction
  (motivation, resource disaggregation, dReDBox)

- Evaluation platform
  (architecture overview, software support)

- Hardware prototype
  (multiple interconnected FPGAs)

- Experimental evaluation
  (matrix multiply test case)

- Conclusions
Server trays typically comprise *compute*, *memory*, and *acceleration* resources.

![Server infrastructure diagram]

**Current server infrastructure**
Server trays typically comprise *compute*, *memory*, and *acceleration* resources.

- Efficient SW communication
- Low power consumption
- Resource proportionality fixed at design time
- Reduced granularity when allocating resources to virtual machines
Resource allocation example

- **VM 1**: 3 MEM Units, 1 CPU
- **VM 2**: 2 MEM Units, 1 CPU
- **VM 3**: 3 MEM Units, 1 CPU
Resource allocation example

- VM 1: 3 MEM Units, 1 CPU
- VM 2: 2 MEM Units, 1 CPU
- VM 3: 3 MEM Units, 1 CPU

Memory utilization: 66%
CPU utilization: 50%
Resource allocation example

- **VM 1**: 3 MEM Units, 1 CPU
- **VM 2**: 2 MEM Units, 1 CPU
- **VM 3**: 3 MEM Units, 1 CPU
- **VM 4**: 4 MEM Units, 3 CPUs

(ITALICS: IT DOES NOT FIT)

Memory utilization: 66%
CPU utilization: 50%
Resource disaggregation

Server-centric approach

Intra-tray network
Resource disaggregation

Server-centric approach

Resource-centric approach

Intra-tray network
Resource allocation example

• **VM 1**: 3 MEM Units, 1 CPU
• **VM 2**: 2 MEM Units, 1 CPU
• **VM 3**: 3 MEM Units, 1 CPU
• **VM 4**: 4 MEM Units, 3 CPUs (IT DOES NOT FIT)

Memory utilization: 66%
CPU utilization: 50%
Resource allocation example

- **VM 1**: 3 MEM Units, 1 CPU
- **VM 2**: 2 MEM Units, 1 CPU
- **VM 3**: 3 MEM Units, 1 CPU
- **VM 4**: 4 MEM Units, 3 CPUs (IT FITS)

Server 1

- Memory utilization: 100%
- CPU utilization: 100%

Server 2

- Memory utilization: 66%
- CPU utilization: 50%

Server 3

- Memory utilization: 100%
- CPU utilization: 100%
Paradigm shift: **Mainboard-as-a-unit** → **disaggregated function-block-as-a-unit**

The dReDBox Architecture

- **Tray w/hot-pluggable & interchangeable bricks (CPU, RAM, Accelerators)**
- **Accelerator Brick**
  - Accelerator
  - SoC
  - DIMM
  - DIMM
  - Virtual DIMM
  - Interrupt & Network Logic
  - FPGA
- **Compute Brick**
  - Virtual DIMM
  - Interrupt & Network Logic
- **RAM Brick**
  - 25G, low-latency electrical I/O
  - Integrated optical ICN Switch
  - To/from other Trays
Evaluation Platform
Contribution

- We provide a tool to write and optimize code for disaggregated environments through enabling runs on real hardware.

- This facilitates the exploration of tradeoffs between the overhead for remote data accesses and the effect of caches, either locally or remotely, for real-world execution scenarios.
Three types of resources, referred to as **blocks**

- Master-worker scheme, with a CPU serving as the master and memory blocks and/or accelerators being the workers.
Platform architecture: **cBlock**

- Provides general-purpose processing capacity

- **PU + memory →**
  - instructions/data

- **DMA + PHY pairs →**
  - remote access to multiple workers

![Diagram of cBlock platform architecture](image)
Platform architecture: **mBlock**

- Provides memory resources

- **PU** →
  - manages access to local memories
  - enables computations near memory

- **DMA + PHY pair** →
  - facilitates remote data transfers
Platform architecture: aBlock

- Provides the infrastructure to deploy custom hardware accelerators

- PU ➔
  - controls the accelerators
  - facilitates communication with the master cBlock

- Interconnect ➔
  - Quick accelerator-based datapath construction
Software stack for inter-block communication

- **PHY layer**: block-to-block data transfers
- **cBlock SW API**: remote memory allocation, memcpy from/to remote blocks, accelerator start/poll, local/remote block debug
- **application SW**: user application code
- **cBlock-specific firmware**, **mBlock-specific firmware**, **aBlock-specific firmware**: user task code

- **The Physical Layer**
  - PHY IP module (Aurora IP)
Software stack for inter-block communication

- **PHY layer**: PHY IP module (Aurora IP)
- **Block Layer**
  - Block-specific firmware for initialization, testing, debugging
  - mBlock and aBlock task code execution
- **The Physical Layer**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>application SW</td>
<td>user application code</td>
</tr>
<tr>
<td>cBlock SW API</td>
<td>remote memory allocation, memcpy from / to remote blocks, accelerator start / poll, local / remote block debug</td>
</tr>
<tr>
<td>cBlock-specific firmware</td>
<td>user task code</td>
</tr>
<tr>
<td>mBlock-specific firmware</td>
<td>user task code</td>
</tr>
<tr>
<td>aBlock-specific firmware</td>
<td>user task code</td>
</tr>
<tr>
<td>PHY layer</td>
<td>block-to-block data transfers</td>
</tr>
</tbody>
</table>
Software stack for inter-block communication

- **API Layer**
  - Platform’s API exposed to the user
    (Drives operations on the Block Layer)

- **Block Layer**
  - Block-specific firmware for initialization, testing, debugging
  - mBlock and aBlock task code execution

- **The Physical Layer**
  - PHY IP module (Aurora IP)

---

**application SW**
- user application code

**cBlock SW API**
- remote memory allocation, memcpy from/to remote blocks, accelerator start/poll, local/remote block debug

**cBlock-specific firmware**
- user task code
- mBlock-specific firmware
- aBlock-specific firmware

**PHY layer**
- block-to-block data transfers
Software stack for inter-block communication

- **User Layer**
  - API-based user code

- **API Layer**
  - Platform’s API exposed to the user
    (Drives operations on the Block Layer)

- **Block Layer**
  - Block-specific firmware for initialization, testing, debugging
  - mBlock and aBlock task code execution

- **The Physical Layer**
  - PHY IP module (Aurora IP)
Inter-block communication flow

xBlock = mBlock or aBlock

1. allocateRV
   - cBlock alloc, size, type
   - xBlock ACK, addr, id

2. readRV
   - cBlock read, id, idx
   - xBlock ACK, data

3. writeRV
   - cBlock write, id, idx, data
   - xBlock ACK

4. memcpy ToBlock
   - cBlock toBlock, id
   - xBlock ready
   - data
   - ACK

5. memcpy FromBlock
   - cBlock fromBlock, id
   - xBlock ACK, data

6. nearMemory TaskProcess
   - cBlock taskProc, tId, in ids, out ids
   - xBlock ACK

7. nearMemory TaskAccel
   - cBlock taskAcc, acId, in ids, out ids
   - xBlock ACK
Hardware Prototype
Hardware prototype

- Three-block prototype: one SOC with reconfig. logic per block type
- Xilinx ZC706 boards
  - Zynq 7045 MPSoC:
    - 2 ARM Cortex A9 CPUs
    - Kintex 7 prog. logic
Hardware prototype

- The Processing Unit on the cBlock is the master.
- Connected to local DDR and ...
- … memory interconnect through the ACP (Accelerator Coherency Port)
Hardware prototype

- The Processing Unit on the cBlock is the master.
- Connected to local DDR and ...
- … memory interconnect through the ACP

Cache-coherent remote data transfers
Experimental Evaluation
Experimental evaluation

- Matrix multiplication (HLS-based IP by Xilinx for acceleration, mmult)
- Evaluation of four different execution scenarios (application mappings)
Experimental evaluation

- Matrix multiplication (HLS-based IP by Xilinx for acceleration, \texttt{mmult})
- Evaluation of four different execution scenarios (application mappings)
Experimental evaluation

- Matrix multiplication (HLS-based IP by Xilinx for acceleration, mmult)
- Evaluation of four different execution scenarios (application mappings)
Experimental evaluation

- Matrix multiplication (HLS-based IP by Xilinx for acceleration, mmult)
- Evaluation of four different execution scenarios (application mappings)
Experimental evaluation

- Matrix multiplication (HLS-based IP by Xilinx for acceleration, mmult)
- Evaluation of four different execution scenarios (application mappings)
Experimental evaluation: PS-MEM code excerpt

```c
void main() {
    float **L, **LR; int i, j;
    remoteVariable rvR[MSIZE];
    // malloc, and initialize L, LR matrices
    ...
    // allocate remotely MSIZE 1-d variables, each of MSIZE
    for (i=0; i<MSIZE; i++) {
        strcpy(rvRName, strcat("rvR", sprintf(buff, "%d", i)));
        allocateRemoteVariable(&rvR[i], MEM_BLOCK, MSIZE, FLOAT_T, rvRName);
    }
    // copy R data to MEM_BLOCK
    for (i=0; i<MSIZE; i++) {
        for (j=0; j<MSIZE; j++) {
            tmpBuffer[j] = (float)(i+j+12);
        }
    }
    memcpyToBlock(MEM_BLOCK, (u32 *)tmpBuffer, &rvR[[i]]);
    // do the L * rvR = LR multiplication
    mmult(L, rvR, LR);
```
Experimental evaluation: PS-MEM code excerpt

```c
void mmult(float **L, remoteVariable *rvR, float **LR) {
    int i, j, k; float* leftRow, cell; u32 tmpData[MSIZE];

    for (i = 0; i < MSIZE; i ++ ) {
        for (j = 0; j < MSIZE; j ++ ) {
            cell = 0.0f;
            leftRow = L[i];
            //cache the i-th line to cBlock
            memcpyFromBlock(MEM_BLOCK,tmpData,&rvR[j]),
            for (k = 0; k < MSIZE; k ++ )
                cell += leftRow[k] * ((float*)tmpData)[k];
            LR[i][j] = cell;
        }
    }
}
```

Remote memory access
Performance comparison: speedups

Reference

SoC
L
R
LR
DDR
cBlock
PS

L1: 32KB
L2: 512KB

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x16</td>
<td></td>
</tr>
<tr>
<td>32x32</td>
<td></td>
</tr>
<tr>
<td>64x64</td>
<td></td>
</tr>
<tr>
<td>128x128</td>
<td></td>
</tr>
<tr>
<td>256x256</td>
<td></td>
</tr>
<tr>
<td>512x512</td>
<td></td>
</tr>
<tr>
<td>1024x1024</td>
<td></td>
</tr>
</tbody>
</table>

- PS-ACCEL vs PS
- PS-MEM vs PS
- PS-NEAR-MEM vs PS
PS outperforms PS-MEM in all cases due to sufficient DDR capacity of the cBlock.
Performance comparison: speedups

Reference

- SoC
- DDR
- cBlock
- L
- R
- LR
- PS

- L1: 32KB
- L2: 512KB

- cBlock cache size sufficiently large to fit data

Matrix size

- 16x16
- 32x32
- 64x64
- 128x128
- 256x256
- 512x512
- 1024x1024

- PS-ACCEL vs PS
- PS-MEM vs PS
- PS-NEAR-MEM vs PS

SMA IF
Better cache utilization on the mBlock, hides the transfer overhead due to ACP.

L1: 32KB
L2: 512KB
Performance comparison: speedups

Reference

- SoC
- L
- R
- LR
- DDR
- cBlock
- PS

L1: 32KB
L2: 512KB

Excessive memory requirements + transfer overhead

Matrix size

Speedup
Performance comparison: speedups

Reference

Transfer overhead comparable with computation

L1: 32KB
L2: 512KB
Performance comparison: speedups

Reference:

L1: 32KB
L2: 512KB

Favorable computation to communication ratio
Presented an FPGA-based evaluation platform for code preparation and optimization for disaggregated environments.

- Multiple FPGA boards assume different roles, e.g., compute, memory, and acceleration
- Software support and user-friendly API eliminate deployment/adoptions overhead

Facilitates the exploration of tradeoffs between the overhead for remote data accesses and the effect of caches, either locally or remotely, for real-world execution scenarios.
Future work

- Add functionality and flexibility to the software stack
  - Support for task-based execution to facilitate parallelism
  - Support for library-based legacy codes

- Augment acceleration capabilities and flexibility
  - Partial reconfiguration
  - Task offloading to processors near accelerators for better performance
Thank you!