Programming Reconfigurable Heterogeneous Computing Clusters Using MPI With Transpilation

Burkhard Ringlein\textsuperscript{1,2}, Francois Abel\textsuperscript{1}, Alexander Ditter\textsuperscript{2}, Beat Weiss\textsuperscript{1}, Christoph Hagleitner\textsuperscript{1}, and Dietmar Fey\textsuperscript{2}

\textsuperscript{1}IBM Research Europe
Zurich, Switzerland

\textsuperscript{2}Department of Computer Science
Computer Architecture
Friedrich-Alexander University Erlangen-Nürnberg

H2RC20 – 2020-11-13
“Power efficiency → compute power”

- end of Dennard scaling
- slowdown of Moore’s law
- → HPC needs more and more accelerators

→ actually, *Heterogeneous HPC (H²PC)*
Reconfigurable Heterogeneous HPC (ReH²PC)

Today’s HPC systems can be classified into three classes:
1) HPC: only CPU servers
2) ReHPC: only FPGAs
3) ReH²PC: CPUs + FPGAs
Programming Standards for Heterogeneous HPC

- MPI is the de-facto standard for HPC
- the presence of a standard interface accelerates both: platform and app development
- there is no similar established standard for ReHPC or ReH$^2$PC
- since FPGAs barely come without CPUs, we focus on ReH$^2$PC

→ Goal: bring MPI to ReH$^2$PC
→ Goal: bring MPI to ReH²PC = *one source of code, no modifications*

```c
int msg[1];
int next_node = (rank + 1) % size;
int previous_node = rank - 1;
if(rank == 0) {
    msg[0] = 0xcaffee;
    MPI_Send(&msg[0], 1, MPI_INTEGER, 1, 0, MPI_COMM_WORLD);
    MPI_Recv(&msg[0], 1, MPI_INTEGER, size-1, 0, MPI_COMM_WORLD, &status);
} else {
    MPI_Recv(&msg[0], 1, MPI_INTEGER, previous_node, 0, MPI_COMM_WORLD, &status);
    MPI_Send(&msg[0], 1, MPI_INTEGER, next_node, 0, MPI_COMM_WORLD);
}
```

**Heterogeneous MPI message ring example**

→ It’s about offloading both computing and communication to FPGAs
Why MPI for ReH\(^2\)PC

- we can learn from proven concepts
- widely established, thousands of applications
- using HLS design flow:

\[
\text{MPI C} \quad \rightarrow \quad \text{HLS} \quad \rightarrow \quad \text{RTL} \quad \rightarrow \quad \text{partial bitstream}
\]

- communication model is similar to how HDL or HLS processes communicate
Why MPI for ReH²PC (2)

• MPI is based on the BSP model:
  ▶ each node is single threaded (pure MPI, no OpenMP etc.)
  ▶ computation and communication is clearly separated

```c
void MPI_Send(void* data, int count, MPI_Datatype datatype,
              int destination, int tag, MPI_Comm communicator);

void MPI_Recv(void* data, int count, MPI_Datatype datatype,
               int source, int tag, MPI_Comm communicator, MPI_Status* status);
```

• the explicit MPI semantic can be easily turned to stream-like read and writes
Difficulties using MPI for ReH²PC

1) “compute first, send later”
   - does not fit parallel processing of FPGAs
   - → “compute while communicating”

2) Single Program, Multiple Data
   - wastes FPGA logic
   - → Multiple Program, Multiple Data

→ MPI for ReH²PC requires:
   1) we need a transpiler: C/MPI → HLS
   2) we need the *cluster run-time description* at compile-time
Transpilation process

- cluster description and MPI C as input
- analyze and replace collective routines
- split SPMD into MPMD
  - insert rank and size
  - constant folding
  - evaluate partial AST (Abstract Syntax Tree) for each rank
- optimize collective routines
- generate HLS code

```
{  
  "nodes": {  
    "cpu" : [0, 1],  
    "fpga" : "2 – 33"  
  }
}
```

```
if condition == x
    true stmts
else
    false stmts
```

```
printf 1
printf "not 1"
```
Transpilation of SPMD to MPMD

```c
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
...
int msg[1];
int next_node = (rank + 1) % size;
int previous_node = rank - 1;
if (rank == 0) {
    msg[0] = 0xcaffee;
    MPI_Send(&msg[0], 1, MPI_INTEGER, 1, 0, MPI_COMM_WORLD);
    MPI_Recv(&msg[0], 1, MPI_INTEGER, size-1, 0, MPI_COMM_WORLD, &status);
} else {
    MPI_Recv(&msg[0], 1, MPI_INTEGER, previous_node, 0, MPI_COMM_WORLD, &status);
    MPI_Send(&msg[0], 1, MPI_INTEGER, next_node, 0, MPI_COMM_WORLD);
}
...
```

clearly defined by MPI syntax & semantic → “obvious” to transpile
Tree Optimizations

- in general: speed up computation by parallelization of communication
- ReH²PC: leverage lower network latency of FPGAs
The ZRLMPI Framework

- **Transpiler**

![Diagram](image)

- **HLS library + MPE:**
  The FPGA run-time environment

- **ZRLMPIlib.so:**
  The CPU run-time environment
$ ZRLMPIrun new udp 10.0.47.11 0ddb12b2-8459-4843-b339-236b2b92b59f 6 ./msg_ring_SW

using udp
setting up cluster...
verify network...
start MPI...
....

Heterogeneous MPI
message ring example
Evaluation: 2D stencil

- CPU: i7 CPU 960 @ 3.20GHz
- simple 2D Jacobi iteration
  - mostly communication bound
- resource usage for 1024x1024 on 8 nodes
  - 1-5% for MPI run-time env. (MPE)

<table>
<thead>
<tr>
<th>Resource</th>
<th>Available</th>
<th>Total</th>
<th>Used</th>
<th>MPE</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT</td>
<td>331680</td>
<td>113832</td>
<td>1940</td>
<td>1715</td>
<td></td>
</tr>
<tr>
<td>LUTRAM</td>
<td>146880</td>
<td>11103</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FF</td>
<td>663360</td>
<td>132553</td>
<td>1744</td>
<td>1058</td>
<td>0</td>
</tr>
<tr>
<td>BRAM</td>
<td>1080</td>
<td>609</td>
<td>3</td>
<td>320</td>
<td>0</td>
</tr>
</tbody>
</table>

cloudFPGA platform:

- Kintex KU60 @ 6.4ns
- 64 FPGAs / 2U chassis
- 10GbE
Evaluation: 2D stencil

- Power on average (incl. memory & NIC):
  - CPU node 127 W
  - FPGA node 8.5 W
- FPGA speedup: 2.0 – 5.6
- mostly communication bound:
  - halo regions synchronization
  - “unoptimized” protocol
  - for FPGAs earlier than for CPUs
- tree optimization (still communication bound):
  - CPU only: speedup 1.27 due to parallelization
  - ReH²PC: speedup 1.32 due to parallelization and FPGAs are doing most of the communication

<table>
<thead>
<tr>
<th>Data size</th>
<th>cluster size</th>
<th>CPU only ms / iteration</th>
<th>ReH²PC ms / iteration</th>
<th>avg. Watt</th>
<th>avg. Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU only</td>
<td>ReH²PC (1 CPU, size=1 FPGAs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16x16</td>
<td>2</td>
<td>23.33</td>
<td>254.49</td>
<td>7.59</td>
<td>136.24</td>
</tr>
<tr>
<td>256x256</td>
<td>4</td>
<td>1670.90</td>
<td>508.48</td>
<td>444.77</td>
<td>154.21</td>
</tr>
<tr>
<td>256x256</td>
<td>8</td>
<td>2136.72</td>
<td>1018.08</td>
<td>842.58</td>
<td>187.68</td>
</tr>
<tr>
<td>1024x1024</td>
<td>8</td>
<td>43120.16</td>
<td>1017.99</td>
<td>7763.87</td>
<td>187.09</td>
</tr>
<tr>
<td>1024x1024</td>
<td>16</td>
<td>36261.91</td>
<td>2036.16</td>
<td>9311.71</td>
<td>253.23</td>
</tr>
<tr>
<td>1024x1024</td>
<td>32</td>
<td>41021.51</td>
<td>4072.33</td>
<td>20939.65</td>
<td>387.99</td>
</tr>
<tr>
<td>256x256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Future Work

- initial philosophy: “functionality first, optimized performance later”
  - common programming model for ReH²PC requires holistic approach
  - learn how to optimize on the way
- example: latest execution time based on improved protocol

<table>
<thead>
<tr>
<th>256x256 8 nodes w/ tree-optimization</th>
<th>old ms/iteration</th>
<th>new ms/iteration</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU only</td>
<td>1686.55</td>
<td>681.29</td>
<td>2.5</td>
</tr>
<tr>
<td>ReH²PC</td>
<td>637.87</td>
<td>34.41</td>
<td>18.6</td>
</tr>
</tbody>
</table>
Summary: MPI for ReH²PC using Transpilation

- ZRLMPI as *proof of concept*
  - exhibit the potential of MPI as standard for ReH²PC
  - full stack solution
  - ...but also full stack to optimize
- Roughly: (early state)
  ZRLMPI on a large ReH²PC cluster:
  - \( \frac{1}{4} \) time (speedup 2.0 – 5.6)
  - 10% of the power (9% – 53%)
- Future Work:
  - optimize tree structure
  - more collectives, memory operations
  - handle non-synthesizable code
cloudFPGA: Further Reading

- F. Abel, “How do you squeeze 1000 FPGAs into a DC rack?” online at LinkedIn
- The cloudFPGA project page at ZRL: https://www.zurich.ibm.com/cci/cloudFPGA/
References

Backup
Transpilation of MPI_Scatter

**Original**

```c
MPI_Comm_size( MPI_COMM_WORLD, &size )
....
int ldim = DIM,size;
int start = rank * ldim;
....
MPI_Scatter &grid[0][0], ldim*DIM, MPI_INTEGER,
original start    datatype
....
&grid[start][0], ldim*DIM, MPI_INTEGER,
root rank
0, MPI_COMM_WORLD);
```

**Transpilation**

```c
for(int i = 0; i < SIZE; i++) {
    int chunk_size = ldim*DIM;
    int *new_start = &grid[0][0] + i*chunk_size;
    if(i == 0){
        memcpy(&grid[start][0], new_start,
              chunk_size * sizeof(int));
        continue;
    }
    MPI_Send(new_start, chunk_size, MPI_INTEGER,
             i, SCATTER_TAG, MPI_COMM_WORLD);
}
```

**to send**

**MPI_Scatter**

![Diagram of MPI_Scatter transpilation with nodes 0, 1, 2, 3, 4, and edges connecting them]