Efficient Parallelization of 5G-PUSCH on a Scalable RISC-V Many-Core Processor

Integrated Systems Laboratory (ETH Zürich)

Marco Bertuletti  mbertuletti@iis.ee.ethz.ch
Yichao Zhang  yiczhang@iis.ee.ethz.ch
Alessandro Vanelli-Coralli  avanelli@iis.ee.ethz.ch
Luca Benini  lbenini@iis.ee.ethz.ch

PULP Platform
Open Source Hardware, the way it should be!
Introduction

• 5G processing requires high throughput on large dimensional signals
• From ASIC design to software-defined network processing (time-to-market ↓)
• Research on RISCV open platforms: ensures long-term scalability, speeds-up community-developed solutions, reduces vendor captivity

- **Complexity** evaluation of 5G-PUSCH processing chain
- Implementation of key kernels on a RISCV many-core cluster with low access latency
- Barriers for **partial synchronization** in the cluster
- Evaluation of **speed-up** and **utilization**
PUSCH processing

We receive frequency-multiplexed transmissions = symbols

- Orthogonal subcarriers
- From multiple antennas
- 14 symbols in Transmission Time-Interval (0.5ms)

(Pilot symbols, are known at the RX + TX, and allow the reconstruction of the channel)
PUSCH processing

\[
x = \left( \hat{H}^H \hat{H} + \sigma^2 I \right)^{-1} \hat{H}^H y
\]

- Antennas
- OFDM demodulation
- FFT
- Beamforming Matrix-Matrix Multiplication
- MIMO Linear System Solver
- CHE Element-wise division
- NE Autocorrelation

\[
\begin{align*}
\hat{H} &= y / x_p \\
y_p &= \hat{H} x_p \\
\varepsilon &= y - y_p \\
\sigma^2 I &= R_{\varepsilon \varepsilon}
\end{align*}
\]
PUSCH processing: Computational complexity

- A computational complexity analysis shows that most of the MACs are in the FFT, the BF and the MIMO stages
- We therefore focus on the optimization of these steps

MACs per stage in PUSCH chain

Impact of MIMO stage depends on the number of UEs transmitting on the same sub-carrier.
MemPool/TeraPool: our target many-core

Snitch processing core

- RV32IMA instruction set architecture + Xpulpimg
- Single-stage single-issue core + LSU & IPU (pipelined)
MemPool/TeraPool: NUMA architecture

Tiles are grouped in hierarchical levels

<table>
<thead>
<tr>
<th></th>
<th>MemPool</th>
<th>TeraPool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per Tile</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Tiles per Group</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Groups per cluster</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

=256 cores  =1024 cores

<table>
<thead>
<tr>
<th>Memory request</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank in the same Tile</td>
<td>1 cycle</td>
</tr>
<tr>
<td>Bank in a different Tile of the same Group</td>
<td>3 cycles</td>
</tr>
<tr>
<td>Bank in a Tile of another Group</td>
<td>5 cycles</td>
</tr>
</tbody>
</table>

Programming model

Fork-join programming model

- Serial execution forks to parallel execution
- Cores access memory concurrently
- Cores are synchronized and parallel execution joins to serial
Synchronization barriers

Synchronization barriers

- Arrival = atomic writes to a synch variable
- Hardwired **wake-up triggers** for departure

8 CSRs for Tiles
1 CSR for Groups
1 CSR for Cores
Implemented kernels

To implement the most computationally complex PUSCH kernels

• We enforced **local access** to the banks in a Tile, to avoid long latency
• We limited the **contentions** for memory shared interconnection resources
• We kept **synchronization** to the bare minimum
Implemented kernels: FFT

The radix-4 butterfly gets inputs at distance N/4

Data is folded to keep these accesses local

Load access pattern

Store access pattern

Data stored in the local memory of cores using it in the subsequent stage
Implemented kernels: FFT

4 cores are working on a 64-points FFT → we partially synchronize these cores

Independent FFTs can be run in sequence by the same cores before synchronization
Implemented kernels: Matrix-Matrix Multiplication

• 4x4 output window maximizes the use of the RF in Snitch
• Parallel version is optimized to avoid contentions

Each core is assigned 4 rows of A

Cores from the same tiles shift to avoid accessing the same group

Cores are assigned columns of B to compute the output windows
Implemented kernels: Cholesky Decomposition

• Output matrix is computed column by column

• At each iteration cores access in parallel different rows → fold rows in the local memory

• Two mirrored matrices are computed at a time by the same core, to increase utilization
High IPC is obtained on all benchmarks

**FFT**
4096-points (16 independent FFTs run between barriers)

**MMM**
(Input 1 4096x64 Input 2 64x32)

**Cholesky**
4x4 matrix (16 independent dec. Run between barriers)

- TeraPool scales well compared to MemPool (overhead = synchronization)
- **LSU stalls** are reduced to less than 10% of the total execution time
Quasi-ideal speed-up and low latency

Use case: 4096 subcarriers, 64 antennas, 32 beams and 4 UEs on the same subcarrier

- The three benchmarks sum up to **0.785ms** @**1GHz**
- Further improvement from architecture specialization
Conclusions

• Identified most **computationaly complex** kernels in PUSCH lower PHY

• **Partial synchronization** between cores of the cluster

• Reduced the **LSU stalls** to less than **10%** of the execution time

• Achieved high speed-up and utilization → **0.785ms** execution time @1GHz

---

github.com/pulp-platform/mempool

Marco Bertuletti mbertuletti@iis.ee.ethz.ch
ETZ, Gloriastrasse 35, 8092 Zürich
@pulp_platform  
@MarcoBertuletti  

17-19 April 2023