TeraPool: Boosting Wireless Communications by Pooling 1000s cores with PULP

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!
6G: A tight mesh of network tranceivers

- Underwater Network
  - Autonomous Underwater Vehicles
- Terrestrial Network
  - CRAN (CU + DU + RU)
  - Vehicular, M2M, IoT
- Aerial Network
  - Drones (Unmanned Aerial Vehicles)
  - LEO, MEO, GEO Satellites

Hakeem, 2022
This huge complexity is a computing problem!

6G puts constraints on the computing load of the Base-stations, in terms of throughput and latency...

More BW, lower latency, more devices!

- < 10 GHz → 3 THz
- 20 Gbps → 1 Tbps
- 1 ms → 0.1/0.01 ms
- 10^6 dev/km^2 → 10^7 dev/km^2

Khan, 2022

Rappaport, 2019
6G drives needs of today's hardware

Mobile Network Traffic is Driving Compute Needs at a Higher Rate Than Moore’s Law

Trends driving higher processing:
- # Antenna branches growing with Massive MIMO
- More carrier bandwidth at higher frequencies
- Wider spectrum allocations in new bands
- Shorter transmission time interval

Ericsson Silicon
A range of purpose built ASICs in advanced technology augmented by partner innovation.

© 2023 IEEE
International Solid-State Circuits Conference
Can we meet the 6G tight requirements with PULP?

Parallel architectures

- Shared-Memory programming
- Low-Latency Tightly-Coupled Data Memory (TCDM)

ISA specialization

- General-Purpose vs. ASICs
- Open ISA (RISC-V) for architecture specialization

Huge amount of data flowing in 5G/6G workloads!
Use more cores and more memory!

MemPool → 256 cores

TeraPool → 1024 cores

Any core can access any bank, some interconnection resources are shared → NUMA

Snitch Cores (rv32ima) are grouped in Tiles with TCDM
How do you program 1000s cores?
We target fork-join parallelism

Fork-join programming model

- Serial execution forks to parallel execution
- Cores access memory concurrently
- Cores are synchronized and parallel execution joins to serial
Synchronization: a log-tree approach

Synchronization barriers

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

**ISSUE**: cores arriving all together will contend for the same memory resource!
Overhead depends on cores’ arrival time

- Small delays → small radices are better,
- Large delays → central-counter wins
Choose the barrier depending on the kernel

Different kernels have different cores’ arrival times: choose barrier accordingly!

<table>
<thead>
<tr>
<th>Radix</th>
<th>Len</th>
<th>AXPY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2^10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>65536</td>
<td></td>
</tr>
<tr>
<td>2^2</td>
<td>131072</td>
<td></td>
</tr>
<tr>
<td>2^3</td>
<td>262144</td>
<td></td>
</tr>
<tr>
<td>2^4</td>
<td>524288</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Radix</th>
<th>Len</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2^10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2x4096</td>
<td></td>
</tr>
<tr>
<td>2^2</td>
<td>4x4096</td>
<td></td>
</tr>
<tr>
<td>2^3</td>
<td>8x4096</td>
<td></td>
</tr>
<tr>
<td>2^4</td>
<td>16x4096</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Radix</th>
<th>Len</th>
<th>MATMUL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2^10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>128x32x128</td>
<td></td>
</tr>
<tr>
<td>2^2</td>
<td>128x128x128</td>
<td></td>
</tr>
<tr>
<td>2^3</td>
<td>256x128x256</td>
<td></td>
</tr>
</tbody>
</table>
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)
$$x = (\hat{H}^H \hat{H} + \sigma^2 I)^{-1} \hat{H}^H y$$

Antennas → OFDM demodulation FFT → Beamforming Matrix-Matrix Multiplication → MIMO Linear System Solver

- **CHE**: Element-wise division
  - $\hat{H} = y / x_p$
  - $y_p = \hat{H} x_p$

- **NE**: Autocorrelation
  - $\varepsilon = y - y_p$
  - $\sigma^2 I = R_{\varepsilon \varepsilon}$
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.

Impact of MIMO stage depends on the number of UEs transmitting on the same sub-carrier.
Low access latency programming

The kernels compute and store locally → reduce load latency of cores that are using data in the next phase of the computation.
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

4096 subcarriers, 64 antennas, 32 beams, 4 UEs on the same subcarrier

Speedups $\rightarrow$ 762, 781, 722
Runtime $\rightarrow$ 0.785ms @1GHz

Further improvement from architecture specialization!

People will ask questions...

Is this physically feasible?

Does it place & route?

Was TeraPool ever taped-out?
Let’s get Physical!

Hierarchical low-latency interconnect + Many Latency-Tolerant Cores (Snitch)

**MinPool: first tape-out**
- 16 cores, 64 KiB, 3 cycles
- TSMC 65

**MemPool: main driver**
- 256 cores, 1 MiB, 5 cycles
- GF 22FDX
  - 500 MHz (WC)
- MemPool-3D

**TeraPool:**
- 1024 cores, 4 MiB, ? cycles
- GF 12 LP+
- How connecting?
- How go Physically?
Scale Up From MemPool? - TeraPool 1-3-5 Design

- Direct Scale-up Version from MemPool
  - 4 Cores -> 8 Cores per Tile
  - 16 Tiles per Group
  - 4 Groups -> 8 Groups
TeraPool Implementation Challenges

- **1.6ns+ critical path**
  - The timing could not close at 500MHz, TT

- **Design runtime is unmanageable**
  - 1.5 Weeks for Group Level Design

- **Not routable**
  - ~33K DRCs

- Diagonal groups timing;
- Large routing channel required;

Design evaluates on GF12nm LP+, TT/0.8V/25C, 13MGE, 2100x2100um, 58% Utilization.
Brainstorming: what do we need?

• Short diagonal timing paths
  • Let’s Keep 4 Groups per Cluster

• 256 Cores per Group, that’s too large
  • New hierarchy: Sub-Group level

• Iteration runtime control
  • 8 Tiles per Subgroup.

• Frequency? or Latency?
  • Flexibility to add spill registers
TeraPool\textsubscript{1-3-5-X}: Hierarchical View

- **Hierarchical Architecture View:**
  - 4 Groups per Cluster
  - 4 Sub-Groups per Group
  - 8 Tiles per Sub-Group
  - 8 Cores per Tile

- **Flexible spill registers adding:**
  - Break long-distance remote accessing paths
  - Different latency/frequency targets
  - Hardware-configurable X = 7/9/11
TeraPool 1-3-5-X: Physically FEASIBLE!

- Methodology:
  - GlobalFoundries’ 12P+ FinFET
  - Synopsys’ FusionCompiler 2022.03
  - Synopsys’ PrimeTime 2022.03
  - WC: SS/0.72V/125C ; TT: TT/0.80V/25C

- Implement Area:
  - Subgroup: 1.52 x 1.52 mm² (58% utilization)
  - Group: 3.8 x 3.8 mm²
  - Cluster: 8.3 x 8.3 mm²
Latency vs. Operating Frequency

- Registers spill critical paths
- Latency-Frequency trade-off
- Near-GigaHertz Design
Energy Breakdown

- Energy-Efficiency:
  - Only 9-13.5pJ for whole L1 access
  - Even p.mac, only 12.5pJ

- With target frequency increases:
  - More design optimization cells add on
  - With larger driveability, SLVT \(\rightarrow\) Power suffers

Design evaluates on GF12nm LP+, TT/0.8V/25C

Avg. +20%
Beat MemPool, go go go

TeraPool is NOT MemPoolx4
- Larger Scale Cluster
  - Reduce system overhead
- Less data movement
  - Memory-Cluster
  - Cluster-Cluster
- Less workload distribution
  - Data allocation
  - Data Splitting

<table>
<thead>
<tr>
<th></th>
<th>MemPool</th>
<th>TeraPool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Number</td>
<td>256</td>
<td>1024</td>
</tr>
<tr>
<td>Tech Node</td>
<td>22nm</td>
<td>12nm</td>
</tr>
<tr>
<td>Die Area (mm²)</td>
<td>12.9mm²</td>
<td>68.9mm²</td>
</tr>
<tr>
<td>Hier-Latency (cycles)</td>
<td>1-3-5</td>
<td>1-3-5-7</td>
</tr>
<tr>
<td>Avg-Latency (zero load)(cycles)</td>
<td>4.7</td>
<td>6.4</td>
</tr>
<tr>
<td>Throughput (req/core/cycle)</td>
<td>0.33</td>
<td>0.23</td>
</tr>
<tr>
<td>Frequency (Typ. Condition)</td>
<td>587MHz</td>
<td>730MHz</td>
</tr>
<tr>
<td>Peak Performance (TOPS)</td>
<td>0.3</td>
<td>1.50</td>
</tr>
<tr>
<td>Area Efficiency (GPOS/MM²)</td>
<td>23.3</td>
<td>21.8</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Example: MatMul Kernel¹</td>
<td></td>
</tr>
<tr>
<td>Kernel Performance (TOPS)</td>
<td>0.17</td>
<td>0.60</td>
</tr>
<tr>
<td>Power Consumption (W)</td>
<td>2.25</td>
<td>4.16</td>
</tr>
<tr>
<td>Energy Efficiency (W)</td>
<td>136</td>
<td>144</td>
</tr>
</tbody>
</table>

1: Matrix-Matrix Multiplication, 256x128x256, 4x4 Tile-based Computing.
What’s next?
Much more will come...

More Configurations:
• 4 Cores/Tile -> more throughput
• 16 Cores/Tile -> less average latency

More Recipes:
• FPU implementation: 8bits, 16bits, even 32bits
• Spatz-TeraPool: Vector unit powered, more ILP

More Clusters:
• Multi-Cluster + Memory hierarchy
• 3D Implementation
• Full MIMO MMSE Receiving Chain

github.com/pulp-platform/mempool