

CachePool: Many-core cluster of customizable, lightweight scalar-vector PEs for irregular L2 data-plane workloads

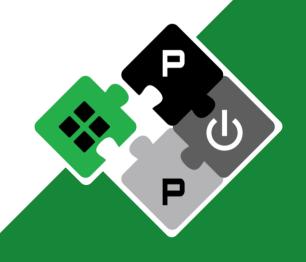
Integrated Systems Laboratory (ETH Zürich)

Zexin Fu, Diyou Shen zexifu, dishen@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!





Outline



- Recall: RLC packet handling and control
- Proposed possible micro-kernels
 - RLC packet handling
 - RLC control
- Requested additional information
- Open discussions



Recall: RLC Packet Handling and Control



RLC Packet Handling

- Massive unstructured sparse data handling in double linked-list format
- No data dependency between different users
- Minimum calculation overhead
- Need buffer for ACK retransmission handling
- Hard or impossible to utilize vector instruction

RLC Control

- Mixed scalar/vector instruction (40%-60%)
- Need to support different types of INT (8, 16, 32, 64)
- FP support is not needed



Reasons for Micro-Kernel Extraction



What is a (micro-)kernel?

Fundamental building blocks or operations for a specific workload or algorithm, e.g. matmul

Why we need them?

- Decoupling the workload chain into several key tasks can reflect the application performance.
- We need some standard results to evaluate our architecture design.

How do we use them?

- Ideally, a large algorithm/workload can be decomposed into a chain of kernels
 - E.g. PUSCH OFDM-BF-MIMO-CHE-NE can be transferred into FFT-MatMul-Cholesky Decomposition-Division-Autocorrelation
- If not possible, we can still use them to evaluate the most critical parts of the workload.
 - Extremely helpful in early development phase of the architecture design.
 - Helpful to tune some architecture design choices/parameters.



Micro-Kernel Extraction



Based on our readings and understandings in the last month, we propose several possible related kernels following these criteria:

RLC Packet Handling

- This part can be abstractly treated as operating on several large linked-lists corresponding to different users, where each node is a packet sent.
- We treated the workload as traversing through the linked-list, dissecting one node into several nodes depending on the output payload size, and retransmitting if needed.

RLC Control Algorithm

- The control part is an optimal value finding problem limited by two constraints and is divided into two algorithm of iterative power allocation and resource allocation.
- We plan to separate the control part from the packet handling for now, treating them as two
 independent problems to solve by one architecture.
- We extract several key operations from the algorithm as the kernels.



Possible Micro Kernels – RLC Packet Handling



Linked List

- Description
 - Randomly generate a large linked list with random payload size.
 - There are various variants for this kernel, such as sorting/inserting/reassembling using several cores.
 - Can integrate some operations on the payload into it, e.g. carry out a dotp/axpy on the payload data.
 - More complex linked list structure is possible, e.g. a list of the lists (double linked list)
- Usage
 - Cache performance and coherence test.
 - A good reference kernel for RLC packet handling.



Possible Micro Kernels – RLC Packet Handling



Pointer Chasing

- Description
 - An important variant of Linked List kernel. Traverse the list and sum the value.
 - Standard test used for DRAM performance, can also be adapted for cache test.
 - Linked list can be placed in different memory hierarchy for different testing purpose.
- Usage
 - Cache performance and coherence test.
 - Cache miss handling policy.



Possible Micro Kernels – RLC Control



Sparse Matrix-Vector Multiplication (SpMV)

- Description
 - A standard test for sparse data handling.
 - Matrix-vector multiplication between a sparse matrix and a dense vector.
 - Can be vectorized.
- Usage
 - (Many-core) vector PE performance analysis
 - Cache performance evaluation on (un)structured sparse data

Maximum Value Sorting

- Description
 - Built upon the linked-list kernels, finding the maximum value from an array/linked-list.
- Usage
 - argmax function in the control algorithm



Possible Micro Kernels – RLC Control



Logarithm Calculation Using Taylor Expansion

- Description
 - Taylor expansion: $\log(1+x) \approx x \frac{x^2}{2} + \dots + \frac{(-1)^{n+1}x^n}{n} + \dots = \sum_{n=1}^{N} (-1)^{n+1} \frac{x^n}{n}$
- Usage
 - $\log(1 + p \cdot g)$ in the control algorithm.

Sum Reduction

- Description
 - Summation across a large number of cores.
 - Can be integrated into some other performance testing kernels, like dotp.
- Usage
 - Widely used in the control algorithm.



Requested Additional Information



	Payload Size (# and type of element)	List Depth / Length	Sparsity Level & Addr. Range	Parallel Handling? Vectorized ops?	Data Transfer Bandwidth	TTI
Linked List	Size range of each node's data (packet size range).* The operation load needs to be done on each payload.	# of packets each user sent	Addr. range of all users; addr allocation policy for new packet.	# of users handled in parallel	e.g. packet arrival rate and size from upper chain, and the output rate to the lower chain	TBD
Pointer Chasing						TBD
(Opt) SpMV	Matrix/Vec size and data format (CSR, CSC, COO)	N/A	Structured types / Unstructured	How's the data dependency across the inputs of the control part? Is it efficient to calculate it using many-core? Which parts are expected to be calculated using vector instructions?	e.g. packet arrival rate, number of users, user's request resources	TBD
Max Value Sorting	Data type and precision (int8/16/32/64)	Array size of control	We do not fully understand the inoutputs of the control part. Are they the same sparse linked-lists as the packet handling?			TBD
Log Calc.	Precision (order of Taylor) Data type and precision (int8/16/32/64)	input				TBD
Sum Reduction						TBD



Open Discussions



Understandings on the RLC

- Packet handling
- Control

Kernel extractions

- Proposed kernels
- Kernel parameters
- Kernel suggestions

Next meeting schedule





Thank you!





