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!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

RLC entity data structure

```
struct RLCEntity {
    unsigned int rlcId;
    unsigned int cellId; /* Indicates the cell to which the RLC entity belongs.*/
    unsigned pollPdu; -- 32
    unsigned pollByte; -- 25000
    unsigned pduWithoutPoll; -- 0 (initial value) /* Indicates the total number of PDUs that are not polled. */
    unsigned byteWithoutPoll; -- 0 (initial value) /* Indicates the total bytes of PDUs that are not polled. */

    unsigned int sduNum; /* Number of sdus to be sent */ --- 3 -> 3+1
    unsigned int sduBytes; /* Number of sdus bytes to be sent */  --- 3*800 -> 4*800 (Assume that the length of each SDU is 800 bytes.)
    void *sduLinkHdr; /* First SDU to be sent */
    void *sduLinkTail; /* Last SDU to be sent */
    unsigned int Reserve1[64-**]    // Pieced together into a cacheline

    unsigned int vtNextAck; /* First SN to be confirmed */ -- 0 (initial value)
    unsigned int vtNext; /* Next Available RLCSN */ --- 0 (initial value)

    unsigned int sendPduNum; /* Number of pdus to be confirmed */ --- 0 (initial value)
    unsigned int sendPduBytes; /* Number of pdus to be confirmed */ --- 0 (initial value)
    void *waitAckLinkHdr;  /* First SDU to be confirmed */
    void *waitAckLinkTail; /* Last SDU to be confirmed */
    unsigned int Reserve2[64-**] // Pieced together into a cacheline
};
```

- **RLCEntity: 16 * 4 byte variables + 2 * reserved field**
  - Q1: How large should each of the reserved field be? 256 byte - ?
  - Q2: Should it be aligned to cache line?
  - Q3: Should we always load it to cache when the core access the entity?
  - In the following estimation we consider the entity size as 16*4+2*256 = **576 byte**

RLC Packet Handling

- **Test Case 1**
  - **DownLink**
    - RLC receives **one** linked-list of **1157** nodes in a **1/2000** second slot from PDCP
    - Throughput: **20Gbps**
    - Received packet payload size: **1350 Byte**
    - **Active RLC entity data structure size: 576 Byte**
  - **UpLink**
    - RLC receives **one** linked-list of **1953** nodes in a **1/2000** second slot from UE
    - Throughput: **1Gbps**
    - Received packet payload size: **160 Byte**

RLC Packet Handling

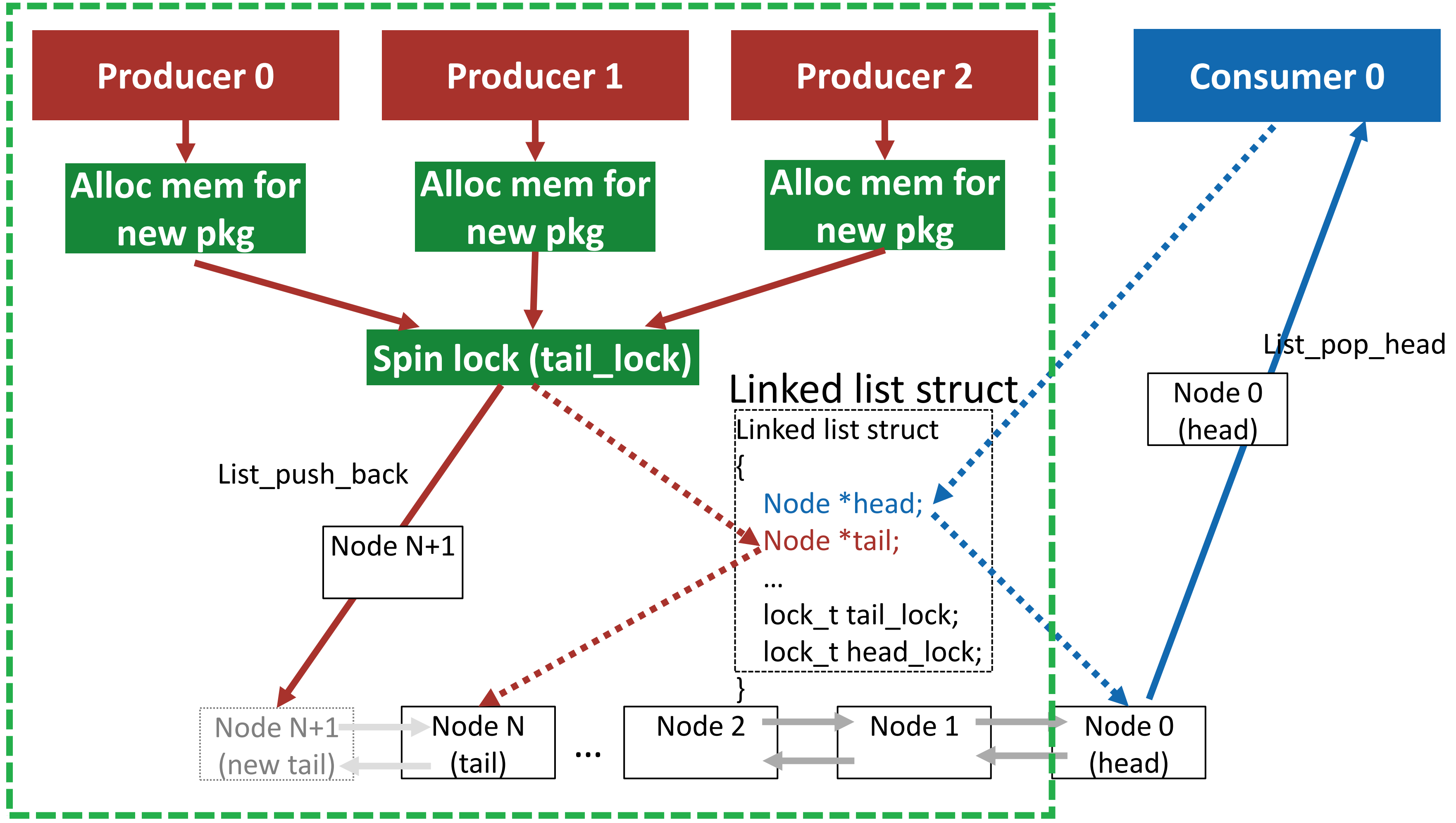
- **Test Case 2**
  - **DownLink**
    - RLC receives **48** linked-lists **x 101** nodes (4882 nodes), in a **1/2000** second slot from PDCP
    - Throughput: **50Gbps**
    - Received packet payload size: **800 Byte**
    - **Active RLC entity data structure size: 27 KB**
  - **UpLink**
    - RLC receives **48** linked-list **x 325** nodes (15625 nodes) in a **1/2000** second slot from UE
    - Input packet payload size: **160 Byte**

RLC Packet Handling

- **Test Case 3**
  - **DownLink**
    - RLC receives **256** linked-lists **x 8** nodes (1963 nodes), in a **1/2000** second slot from PDCP
    - Throughput: **20Gbps**
    - Received packet payload size: **800 Byte**
    - **Active User: 4800**
    - **Active RLC entity data structure size per slot: 144 KB**
  - **UpLink**
    - RLC receives **384** linked-list **x 20** nodes (7813 nodes) in a **1/2000** second slot from UE
    - Input packet payload size: **160 Byte**
    - **Active User: 4800**

Recap: Multi-producers, single consumer linked list

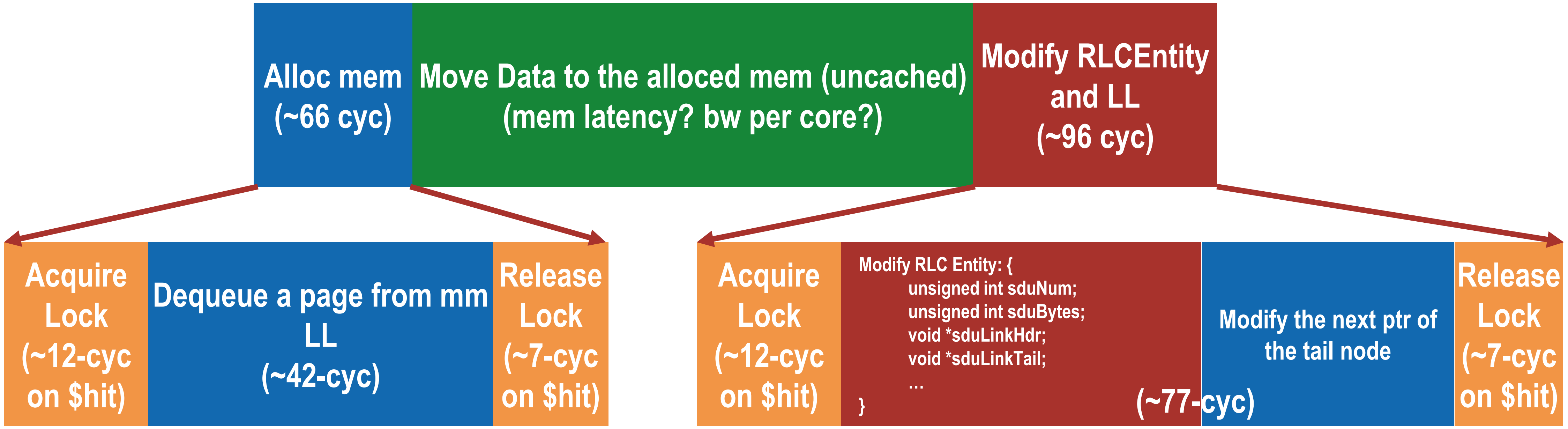
1. Multiple producers, single consumer double linked-list kernel



```
Linked list struct
{
   Node *head;
   Node *tail;
   …
   lock_t tail_lock;
   lock_t head_lock;
}
```

Preliminary Estimation: Timing Breakdown

- **Packet Receiving Processing Task (Producer)**



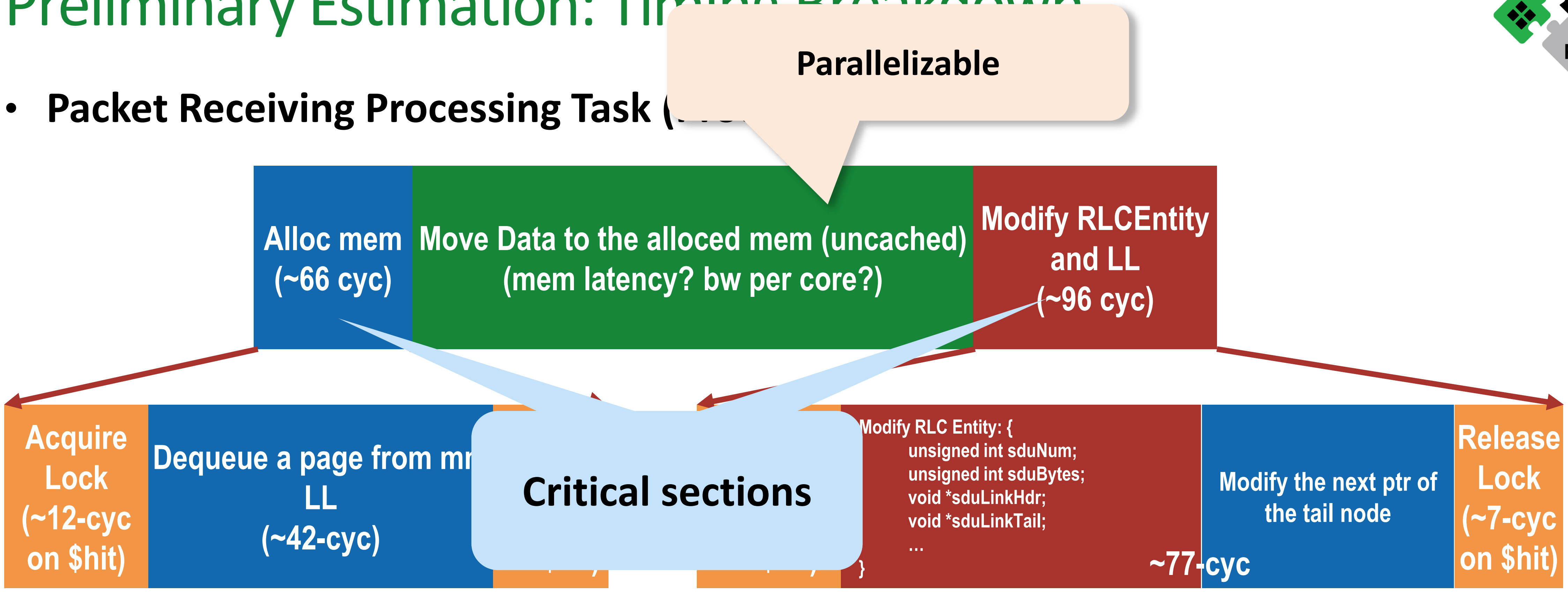
Alloc mem (~66 cyc)
Move Data to the alloced mem (uncached) (mem latency? bw per core?)
Modify RLCEntity and LL (~96 cyc)
Acquire Lock (~12-cyc on $hit)
Dequeue a page from mm LL (~42-cyc)
Release Lock (~7-cyc on $hit)
Acquire Lock (~12-cyc on $hit)
Modify the next ptr of the tail node
(~77-cyc)
Release Lock (~7-cyc on $hit)

```
Modify RLC Entity: {
        unsigned int sduNum;
        unsigned int sduBytes;
        void *sduLinkHdr;
        void *sduLinkTail;
        …
}
```

- To move a 1350 byte data:
  - If use HBM, e.g. latency 150-cyc, BW per core 256-bit/cyc: ~150+1.3KB/(256bit/cyc)=192 cycles, 354 cyc/pkg
  - If use DRAM, e.g. latency 50-cyc, BW per core 64-bit/cyc: ~50+1.3KB/(64bit/cyc)=218 cycles, 380cyc/pkg

Preliminary Estimation: Timing Breakdown

- **Packet Receiving Processing Task (**

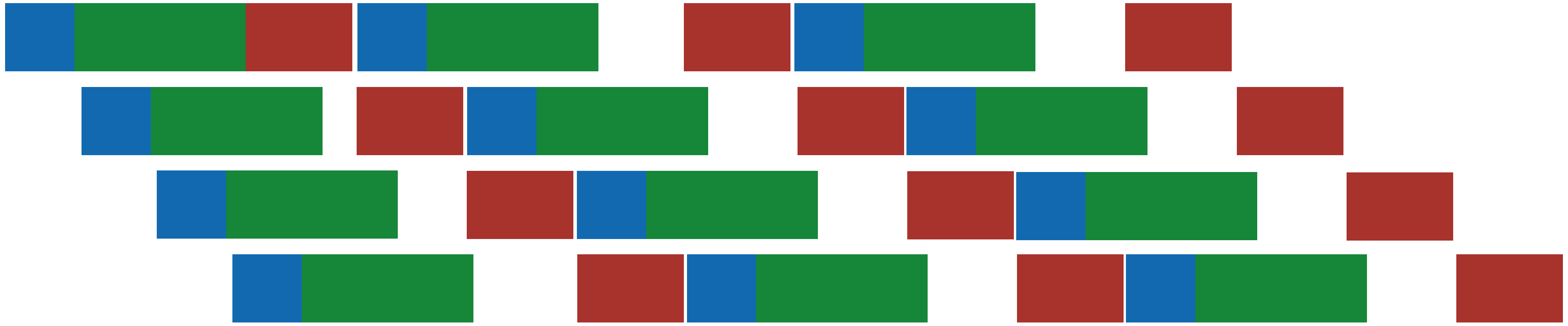

Parallelizable
Alloc mem (~66 cyc)
Move Data to the alloced mem (uncached) (mem latency? bw per core?)
Modify RLCEntity and LL (~96 cyc)
Acquire Lock (~12-cyc on $hit)
Dequeue a page from mr LL (~42-cyc)
Critical sections
Modify the next ptr of the tail node
~77-cyc
Release Lock (~7-cyc on $hit)

```
Modify RLC Entity: {
        unsigned int sduNum;
        unsigned int sduBytes;
        void *sduLinkHdr;
        void *sduLinkTail;
        ...
}
```

- To move a 1350 byte data:
  - If use HBM, e.g. latency 150-cyc, BW per core 256-bit/cyc: ~150+1.3KB/(256bit/cyc)=192 cycles, 354 cyc/pkg
  - If use DRAM, e.g. latency 50-cyc, BW per core 64-bit/cyc: ~50+1.3KB/(64bit/cyc)=218 cycles, 380cyc/pkg

Preliminary Estimation: In multicore scenario

- **Packet Receiving Processing Task**



- If use a tile (4 PEs) to process receiving task, run at 800MHz clk frequency
  - ~480 cycles to receive a pkg per core
    - **TODO**: The “pkg head processing” and the “insertion of sent pkg linked-list” haven’t been analyzed
  - With above preliminary estimation, each tile can receive ~3300 nodes per DL slot
    - Can handle test case 1;
    - Test case 2: may need 2 tiles;
    - Test case 3: the cache size of one tile need to be larger, e.g. 256 KB
    - Note: Other tasks (e.g. consumer, linear algebra …) are not considered here.

Status Update

- **Implement and analyze the “Packet Receiving Processing Task “**
  - Multi-user pkg receiving using a tile
  - Analyze the timing breakdown of the pkg receiving process
  - Fix several bug of the kernel
- **Question**
  - How should we add RLC header to the received pkg?
- **Next**
  - The pkg RLC header processing
  - The sent pkg linked-list insertion and deletion and its data movement
  - The ack message from the UE

Thank you!

Q&A