# Ramulator 2.0 사용법

*Intelligent System*

*Laboratory*

# Ramulator 2.0 setup

## ❑ Using the revised version of Ramulator2.0 (load/store enabled)

- Clone the repository
  ```
  $ git clone https://github.com/RayKim3103/my_ramulator2
  ```

- Configure the project and build the executable
  ```
  $ mkdir build
  $ cd build
  $ cmake ..
  $ make -j
  $ cp ./ramulator2 ../ramulator2
  $ cd ..
  ```

- Executing Simulation
  ```
  $ ./ramulator2 -f ./example_config.yaml
  $ ./ramulator2 -f ./example_config_LDST.yaml
  ```

# Ramulator 2.0 setup

## ❑ 기존 Ramulator2.0에서의 2가지 simulation 설정

- simplistic out-of-order core model
  - 입력은 (bubble 개수) (virtual load address) (virtual store address) 형식으로 줌
  - 간단한 out of order cpu model 및 LLC(Last Level Cache)를 모델링
  - Cache가 Miss 시 DRAM memory system에 Request(Load/Store)를 send
  - 이 send된 Request(Load/Store)를 memory system에서 처리하고 완료 시 cpu system에게 callback

example_inst.trace

```
1    3 20734016
2    1 20846400
3    6 20734208
4    1 20846400
5    8 20841280 20841280
6    0 20734144
7    2 20918976 20734016
8    1 20846400
```

*오른쪽 trace 파일 입력 시 scenario – CPU입장 instruction fetch기준 cycle*
Cycle 0 : 1cycle간 stall (IPC = 4이기에 1cycle에 4개 처리하도록 modeling됨)
Cycle 0: load request issue (virtual address: 20734016)
Cycle 1: 1cycle 간 stall
Cycle 1: load request issue (virtual address: 20846400)
Cycle 2~3: 2cycle 간 stall
Cycle 3: load request issue (virtual address: 20734208)
Cycle 4: 1cycle간 stall
Cycle 4: load request issue (virtual address: 20846400)
Cycle 5~6: 2cycle간 stall
Cycle 7: load request issue (virtual address: 20841280)
Cycle 8: store request issue (virtual address: 20841280)
…

# Ramulator 2.0 setup

## ❑ 기존 Ramulator2.0에서의 2가지 simulation 설정

- simplistic out-of-order core model

example_inst.trace

```
3 20734016
1 20846400
6 20734208
1 20846400
8 20841280 20841280
0 20734144
2 20918976 20734016
1 20846400
```

- Cycle by Cycle 분석

```
root@762430b8c6ee:/workspace# ./ramulator2 -f ./example_config.yaml
cycle: 1 Frontend-llc->send(), non-memory instruction
cycle: 1 Frontend-llc->send(), non-memory instruction
cycle: 1 Frontend-llc->send(), non-memory instruction
cycle: 1 Frontend-llc->send(), load_addr: 20734016
cycle: 2 Frontend-llc->send(), non-memory instruction
cycle: 2 Frontend-llc->send(), load_addr: 20846400
cycle: 3 Frontend-llc->send(), non-memory instruction
cycle: 3 Frontend-llc->send(), non-memory instruction
cycle: 3 Frontend-llc->send(), non-memory instruction
cycle: 3 Frontend-llc->send(), non-memory instruction
cycle: 4 Frontend-llc->send(), non-memory instruction
cycle: 4 Frontend-llc->send(), non-memory instruction
cycle: 4 Frontend-llc->send(), load_addr: 20734208
cycle: 5 Frontend-llc->send(), non-memory instruction
cycle: 5 Frontend-llc->send(), load_addr: 20846400
cycle: 6 Frontend-llc->send(), non-memory instruction
cycle: 6 Frontend-llc->send(), non-memory instruction
cycle: 6 Frontend-llc->send(), non-memory instruction
cycle: 6 Frontend-llc->send(), non-memory instruction
cycle: 7 Frontend-llc->send(), non-memory instruction
cycle: 7 Frontend-llc->send(), non-memory instruction
cycle: 7 Frontend-llc->send(), non-memory instruction
cycle: 7 Frontend-llc->send(), non-memory instruction
cycle: 8 Frontend-llc->send(), load_addr: 20841280
cycle: 9 Frontend-llc->send(), writeback_addr: 20841280
```

# Ramulator 2.0 setup

## ❑ 기존 Ramulator2.0에서의 2가지 simulation 설정

- memory-trace parser model (Load/Store trace의 경우, Read/Write trace는 배제)
  - 입력은 (LD/ST) (physical load/store address) 형식으로 줌
  - 입력에서 LD/ST 중간에 n Cycle동안 Bubble을 주고 싶다면, (BU) (stalled_cycles) 형식으로 줌
  - Simulation은 Frontend(CPU part)가 instruction을 매 cycle마다 한 줄씩 memory system으로 send
  - 이 send된 Request(Load/Store)를 memory system에서 처리하고 완료 시 Frontend에게 callback

  - 여기서 Frontend(CPU part)는 Memory Request를 매 cycle memory system으로 넘기는 작업만을 수행합니다.

관련된 내용은 아래 링크의 2page 끝에서 3page 중간까지 적혀 있습니다. (Onur Mutlu 강의)

fetch.php

# Ramulator 2.0 setup

## ❑ 기존 Ramulator2.0에서의 2가지 simulation 설정

- memory-trace parser model (Load/Store trace의 경우, Read/Write trace는 배제)
  - Physical Address에서 DRAM vector로의 변환은 아래 folder에 정의되어 있음
    - ➢ src > addr_mapper > impl > linear_mappers.cpp

example_LDST.trace

```
1   LD 0x00000000
2   ST 0x00000400
3   LD 0x00000800
4   ST 0x00000C00
5   LD 0x00001000
6   ST 0x00001400
7   LD 0x00001800
8   ST 0x00001C00
9
```

- Cycle by Cycle 분석

```
cycle: 0 System - send() Read Request Address: 0 / Request Address Vector: 0 0 0 0 0 0
cycle: 0 Controller - send() Read - Request Address Vector: 0 0 0 0 0 0
cycle: 1 System - send() Write Request Address: 1024 / Request Address Vector: 0 0 0 0 0 16
cycle: 1 Controller - send() Write - Request Address Vector: 0 0 0 0 0 16
cycle: 2 System - send() Read Request Address: 2048 / Request Address Vector: 0 0 0 0 0 32
cycle: 2 Controller - send() Read - Request Address Vector: 0 0 0 0 0 32
cycle: 3 System - send() Write Request Address: 3072 / Request Address Vector: 0 0 0 0 0 48
cycle: 3 Controller - send() Write - Request Address Vector: 0 0 0 0 0 48
cycle: 4 System - send() Read Request Address: 4096 / Request Address Vector: 0 0 0 0 0 64
cycle: 4 Controller - send() Read - Request Address Vector: 0 0 0 0 0 64
cycle: 5 System - send() Write Request Address: 5120 / Request Address Vector: 0 0 0 0 0 80
cycle: 5 Controller - send() Write - Request Address Vector: 0 0 0 0 0 80
cycle: 6 System - send() Read Request Address: 6144 / Request Address Vector: 0 0 0 0 0 96
cycle: 6 Controller - send() Read - Request Address Vector: 0 0 0 0 0 96
cycle: 7 System - send() Write Request Address: 7168 / Request Address Vector: 0 0 0 0 0 112
cycle: 7 Controller - send() Write - Request Address Vector: 0 0 0 0 0 112
```

# Ramulator 2.0 setup

## ❑ 기존 Ramulator2.0에서의 2가지 simulation 설정

- memory-trace parser model (Load/Store trace의 경우, Read/Write trace는 배제)

example_LDST.trace

```
1    LD 0x00000000
2    BU 5
3    ST 0x00000400
4    LD 0x00000800
5    ST 0x00000C00
6    BU 5
7    LD 0x00001000
8    ST 0x00001400
9    LD 0x00001800
10   ST 0x00001C00
11
```

- Cycle by Cycle 분석

```
cycle: 0 System - send() Read Request Address: 0 / Request Address Vector: 0 0 0 0 0 0
cycle: 0 Controller - send() Read - Request Address Vector: 0 0 0 0 0 0
cycle: 5 Bubble finished
cycle: 6 System - send() Write Request Address: 1024 / Request Address Vector: 0 0 0 0 0 16
cycle: 6 Controller - send() Write - Request Address Vector: 0 0 0 0 0 16
cycle: 7 System - send() Read Request Address: 2048 / Request Address Vector: 0 0 0 0 0 32
cycle: 7 Controller - send() Read - Request Address Vector: 0 0 0 0 0 32
cycle: 8 System - send() Write Request Address: 3072 / Request Address Vector: 0 0 0 0 0 48
cycle: 8 Controller - send() Write - Request Address Vector: 0 0 0 0 0 48
cycle: 13 Bubble finished
cycle: 14 System - send() Read Request Address: 4096 / Request Address Vector: 0 0 0 0 0 64
cycle: 14 Controller - send() Read - Request Address Vector: 0 0 0 0 0 64
cycle: 15 System - send() Write Request Address: 5120 / Request Address Vector: 0 0 0 0 0 80
cycle: 15 Controller - send() Write - Request Address Vector: 0 0 0 0 0 80
cycle: 16 System - send() Read Request Address: 6144 / Request Address Vector: 0 0 0 0 0 96
cycle: 16 Controller - send() Read - Request Address Vector: 0 0 0 0 0 96
```

# Ramulator 2.0 setup

## ❑ 전체 Report

example_LDST.trace

```
LD 0x00000000
BU 5
ST 0x00000400
LD 0x00000800
ST 0x00000C00
BU 5
LD 0x00001000
ST 0x00001400
LD 0x00001800
ST 0x00001C00
```

example_config_LDST.yaml

```yaml
Frontend:
  impl: LoadStoreTrace
  path: example_LDST.trace  # trace 파일 경로 (상대/절
  clock_ratio: 1

  Translation:
    impl: RandomTranslation
    max_addr: 2147483648


MemorySystem:
  impl: GenericDRAM
  clock_ratio: 1

  DRAM:
    impl: DDR4
    org:
      preset: DDR4_8Gb_x8
      channel: 1
      rank: 2
    timing:
      preset: DDR4_2400R
    drampower_enable: true
    # power_debug: true  # option: debug log
    voltage:
      preset: Default  # option: voltage preset
    current:
      preset: Default  # option: current preset

  Controller:
    impl: Generic
    Scheduler:
      impl: FRFCFS
    RefreshManager:
      impl: AllBank
    RowPolicy:
      impl: ClosedRowPolicy
      cap: 4
    plugins:

  AddrMapper:
    impl: RoBaRaCoCh
```

```
Frontend:
  impl: LoadStoreTrace

MemorySystem:
  impl: GenericDRAM
  total_num_other_requests: 0
  total_num_write_requests: 4
  total_num_read_requests: 4
  memory_system_cycles: 144

  DRAM:
    impl: DDR4
    active_cycles_rank1: 0
    pre_background_energy_rank1: 8.0967599999999997
    total_background_energy_rank0: 8.6515379999999986
    idle_cycles_rank1: 144
    total_cmd_energy: 3.5885639999999999
    total_cmd_energy_rank0: 3.5885639999999999
    total_energy_rank0: 12.240101999999998
    act_background_energy_rank0: 6.7960304999999996
    pre_background_energy_rank0: 1.8555074999999999
    active_cycles_rank0: 111
    act_background_energy_rank1: 0
    total_energy: 20.336861999999996
    idle_cycles_rank0: 33
    total_energy_rank1: 8.0967599999999997
    total_background_energy_rank1: 8.0967599999999997
    total_background_energy: 16.748297999999998
    total_cmd_energy_rank1: 0
  AddrMapper:
    impl: RoBaRaCoCh

Controller:
  impl: Generic
  id: Channel 0
  avg_read_latency_0: 36.75
  read_queue_len_avg_0: 0.909722209
  write_queue_len_0: 278
  queue_len_0: 452
  num_other_reqs_0: 0
  num_write_reqs_0: 4
  read_latency_0: 147
  priority_queue_len_avg_0: 0.298611104
  row_hits_0: 6
  priority_queue_len_0: 43
  row_misses_0: 2
  row_conflicts_0: 0
  read_row_misses_0: 1
  queue_len_avg_0: 3.13888884
  read_row_conflicts_core_0: 0
  read_row_hits_0: 3
  write_queue_len_avg_0: 1.93055558
  read_row_conflicts_0: 0
  write_row_misses_0: 1
  write_row_conflicts_0: 0
  read_queue_len_0: 131
  write_row_hits_0: 3
  read_row_hits_core_0: 0
  read_row_misses_core_0: 0
  num_read_reqs_0: 4
  Scheduler:
    impl: FRFCFS
  RefreshManager:
    impl: AllBank


  RowPolicy:
    impl: ClosedRowPolicy
    num_close_reqs: 2
```