mNPUsim Explained

Soojin Hwang*
sjhwang@casys.kaist.ac.kr
School of Computing, KAIST

Sunho Lee
myshlee417@casys.kaist.ac.kr
School of Computing, KAIST

December 10th, 2024

Contents

1 Introduction

mNPUsim is a cycle-accurate simulator for modeling multi-NPU systems with off-chip memory resources, first introduced at IISWC 2023 [1]. The public version of mNPUsim is available on GitHub (`https://github.com/casys-kaist/mNPUsim`). Figure 1 visualizes the overview of mNPUsim structure with configurations. This document is a manual for getting started with mNPUsim, providing a detailed explanation of the simulator configurations and code structure.

2 Getting Started

2.1 Dependencies

mNPUsim was implemented and tested in Ubuntu environment using C++ and requires g++ version v7.4.0 or higher for compilation. In addition to the standard C++11 library, mNPUsim requires DRAM-sim3 as a shared library [3], which is included as a submodule in the GitHub repository. Note that mNPUsim uses a modified version of DRAMsim3 provided by the developer of mNPUsim (`https://github.com/terrafin/DRAMsim3`) instead of the original version.

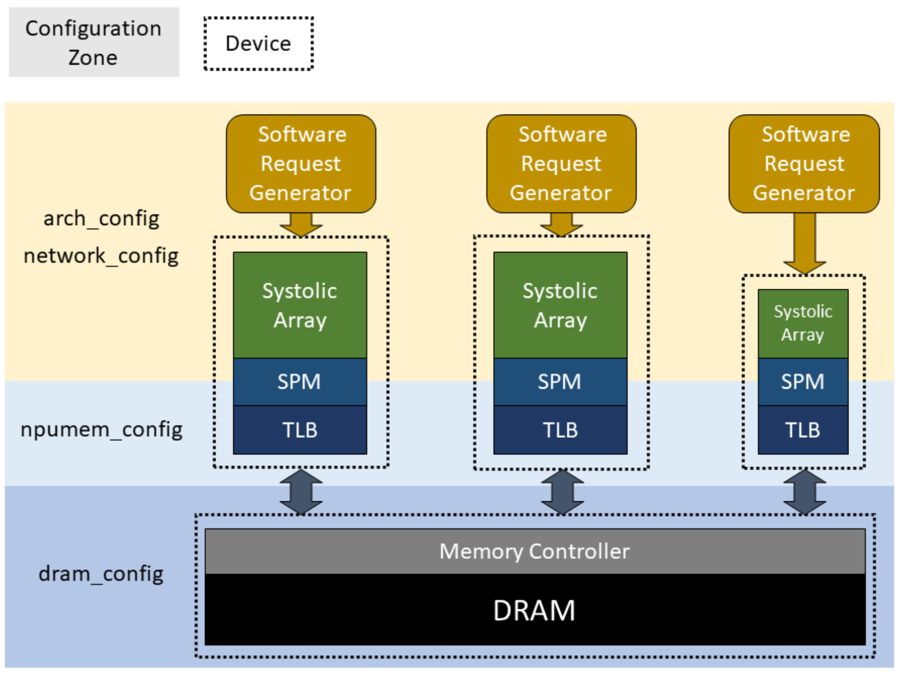

Figure 1: Overall structure of mNPUsim, modeling multi-NPU system.

2.2 Compilation

```
$ cd DRAMsim3
$ make libdramsim3.so
$ cd ..
$ export LD_LIBRARY_PATH=./DRAMsim3:$$LD_LIBRARY_PATH
$ g++ -L./DRAMsim3/src/ -g -O2 *.cpp *.h -o mnpusim -std=c++11 -L./DRAMsim3 -ldramsim3
```

Note that the last two lines can be replaced with `make single` command.

2.3 Running Simulation

```
$ make single
$ ./mnpusim [Architecture config] [Network config] [DRAM config] [NPU-side memory config]
[output directory] [Miscellaneous config (Optional)]
```

Several examples are available in Makefile.

3 Simulator Configurations

As shown in Section 2.3, mNPUsim requires four mandatory configuration files and one optional configuration file for execution. mNPUsim is designed to simulate NPU cores exclusively. However, it also provides partial modeling capabilities for other architectures by enabling custom rearranging of computational resources and adjustment of latency parameters.

3.1 NPU Architectures

mNPUsim receives *architecture configuration list* (.txt) as a first input. mNPUsim represents an architecture composed of multiple NPUs using two types of configuration files.

1. *Architecture configuration list* (arch_config/core_architecture_list): It contains a list of paths for individual *architecture configuration* (.csv), with each file name listed on a separate line.

2. *Architecture configuration* (arch_config/core_architecture): It provides detailed specifications for each NPU and is organized with the multiple components.

This structure enables flexible and scalable representation of architectures composed of multiple NPUs.
Specification of Fields (*architecture configuration*): Each line in this configuration file denotes the field with its name.

- arch_name: Name (or nickname) of the NPU architecture.
- systolic_height, systolic_width: Height and width of systolic array.
- sram_ifmap_size, sram_filter_size, sram_ofmap_size: Size of SRAM assigned for ifmap, filter, ofmap. Must be a multiple of cacheline_size.
- dataflow_type: Systolic array dataflow. Currently, only output-stationary (denoted as *os*) dataflow is supported.
- element_unit: Size of tensor element in Byte.
- cacheline_size: Size of SPM block (word).
- tile_ifmap_size, tile_filter_size, tile_ofmap_size: Tile size. Should be set to half of SRAM size for double buffering. Must be a multiple of cacheline_size.
- unit_compute: Cycle time of MAC operation. 1 for default.

3.2 Machine Learning Workloads

mNPUsim receives *network configuration list* (.txt) as a second input. mNPUsim represents machine learning networks by defining the workloads operating on each NPU and specifying the topology of each workload using two configuration files.

1. *Network configuration list* (network_config/netconfig_list): It contains a list of paths for individual *network configuration* (.csv), with each file name listed on a separate line.
2. *Network configuration* (network_config/network_architecture): It defines the topology of each workload, including its size and connections. This structure allows for flexible modification and simulation of workloads across multiple NPUs.

Specification of Fields (*network configuration*): Each line in this configuration file denotes the configuration of each layer. The network configuration file format mostly follows SCALE-Sim format [5].

- layer_name: Name (or nickname) of the layer.
- IFMAP Height, IFMAP Width, Filter Height, Filter Width, Channels, Num Filter, Strides: Size and stride for the layer.
- Type: Type of the layer. Currently, mNPUsim supports Im2col_conv, Conv, Pool, Gemm, and Gemv. For Gemv, ifmap fields represent matrices and filter fields represent vectors.
- IFMAP Base Addr, Filter Base Addr, Ofmap Base Addr: Connection information. mNPUsim supports the custom management of start address of ifmap/filter/ofmap. There are several options of connection.
  - -1 (Sequential connection): the ifmap of this layer is the ofmap of previous layer.
  - -2 (New address connection): the ifmap of this layer is the new-space (new tensor in this network).
  - n ($\leq$ BOUNDARY_IFMAP_OFMAP_ARRAY): the ifmap/filter/ofmap is the output of n-th layer.
  - m ($>$ BOUNDARY_IFMAP_OFMAP_ARRAY): the ifmap/filter/ofmap is the input of m-th BOUNDARY_IFMAP_OFMAP_ARRAY layer.

  The address is translated from base network file to intermediate network file. Therefore, the ifmap/filter/ofmap base address values in intermediate network files are absolute value while relative values in original network files.

3.3 Off-chip Memory

As mNPUsim assumes shared DRAM among multiple NPUs, only one 'DRAM configuration file' (.cfg) is required as a third input. The dram_config/total_dram_config directory contains several example configurations for 'DRAM configuration' files, while the dram_config/single_dram_config directory stores DRAMsim3 configuration file.

Specification of Fields: Each line in this configuration file defines the value of each field with its name.

- dramconfig_name: Configuration file (.ini) name for DRAMsim3.
- pagebits: Length of page offset (in binary).
- npu_num: Number of NPUs.
- dramoutdir_name: Name of directory for saving DRAM/TLB logs.
- dram_unit: request_size_bytes calculated in DRAMsim3/src/configuration.cc file.
- dram_log: Flag for recording DRAM/TLB logs.
- dram_capacity: DRAM Capacity in Byte.
- dynamic_partition: Dynamic partitioning flag for DRAM. 0 for static partitioning, 1 for dynamic.
- tlb_shared: (single-level) TLB shared among NPUs?
- ptw_shared: PTW shared among NPUs?
- channel_num: Number of channels per Module (Default: same as the definition in .ini file)
- module_num: Number of HBM modules (DRAMsim objects)
- tiled_bw: Tile-level bandwidth interleaving?
- token_bw: Fair bandwidth sharing with token?
- return_ptw

3.4 Memory Management Units

While DRAM is shared among multiple NPUs, TLB and SPM are still private. To handle these efficiently, mNPUsim receives 'NPU-side memory configuration list file' (.txt) name as a fourth input. 'NPU-side memory configuration list file' contains list of 'NPU-side memory configuration file' (.cfg) name, one by one in each line.

Specification of Fields: Each line in the configuration (.cfg) file defines the value of each field with its name.

- template: Architecture configuration file name (to avoid confusion).
- spm_size: Size of efficient scratchpad memory (for double buffering, half of true size).
- cacheline_size: Scratchpad block (word) size.
- tlb_hit_latency, tlb_miss_latency: TLB hit/miss latency in DRAM clock cycle.
- tlb_assoc: TLB associativity (= Number of entries per set).
- tlb_entrynum: Total number of TLB entries. 0 for the system without address translation overhead.
- tlb_portnum: TLB bandwidth (access/cycle). 0 with infinite bandwidth.
- spm_latency: Scratchpad access latency in DRAM clock cycle.
- double_buffer: Flag for double buffering.

- npu_clockspeed, dram_clockspeed: Relative value of clock frequency (NPU frequency : DRAM frequency = npu_clockspeed : dram_clockspeed). Must be coprime.
- tlb_pref_mode: TLB prefetching mode. 0 for no prefetching, 1 for sequential prefetching, ...
- ptw_num: Number of page table walkers. (ptw_num **mod** pt_step_num must be 0)
- pt_step_num: Number of page table walk steps.
- is_tpreg: TPReg [2] activated?
- token_bucket
- token_epoch

3.5 Other Configurations

The sixth input (i.e. the fifth configuration) of mNPUsim - multi-npu is optional: a single 'miscellaneous configuration file' (.cfg) for several optional setups. This configuration file is only required for back-to-back execution, asynchronously-started execution, or non-greedy partitioning of shared resources (PTW, especially). The misc_config directory contains several example configurations for 'miscellaneous configuration' files with the single, dual, and quad core NPU system.
Specification of Fields: Each line in this configuration file denotes the required setup for each NPU for five fields, separated by blanks.

- Start cycle: The time cycle at which each NPU begins execution. mNPUsim provides an option to configure different start times for workloads, enabling the modeling of scenarios where workloads arrive dynamically.
- Number of iterations: The number of iterations for which each NPU operates. A positive value specifies the exact number of iterations, while a negative value indicates the minimum number of iterations across all workloads operating on the NPUs. In other words, after the iteration of the last workload reaches n, each workload individually completes.
- Shared PTW partition (0 for full-dynamic): The sum of this field for each NPU should be equal to the total number of PTWs in 'DRAM configuration file'.
- Upper bound of shared PTW partition: Default is 0.
- Lower bound of shared PTW partition: Default is 0.

Without misc configuration file, default values are always set to "0 1 0 0 [Total number of shared PTW]". Note that, [iteration] = -1 means that repeating specific task until finishing every other workloads. -2 means that repeating all workloads until all co-runners runs at least twice.

4 Code Structure and Layout

Figure 2 shows the structure of mNPUsim source code as a tree.

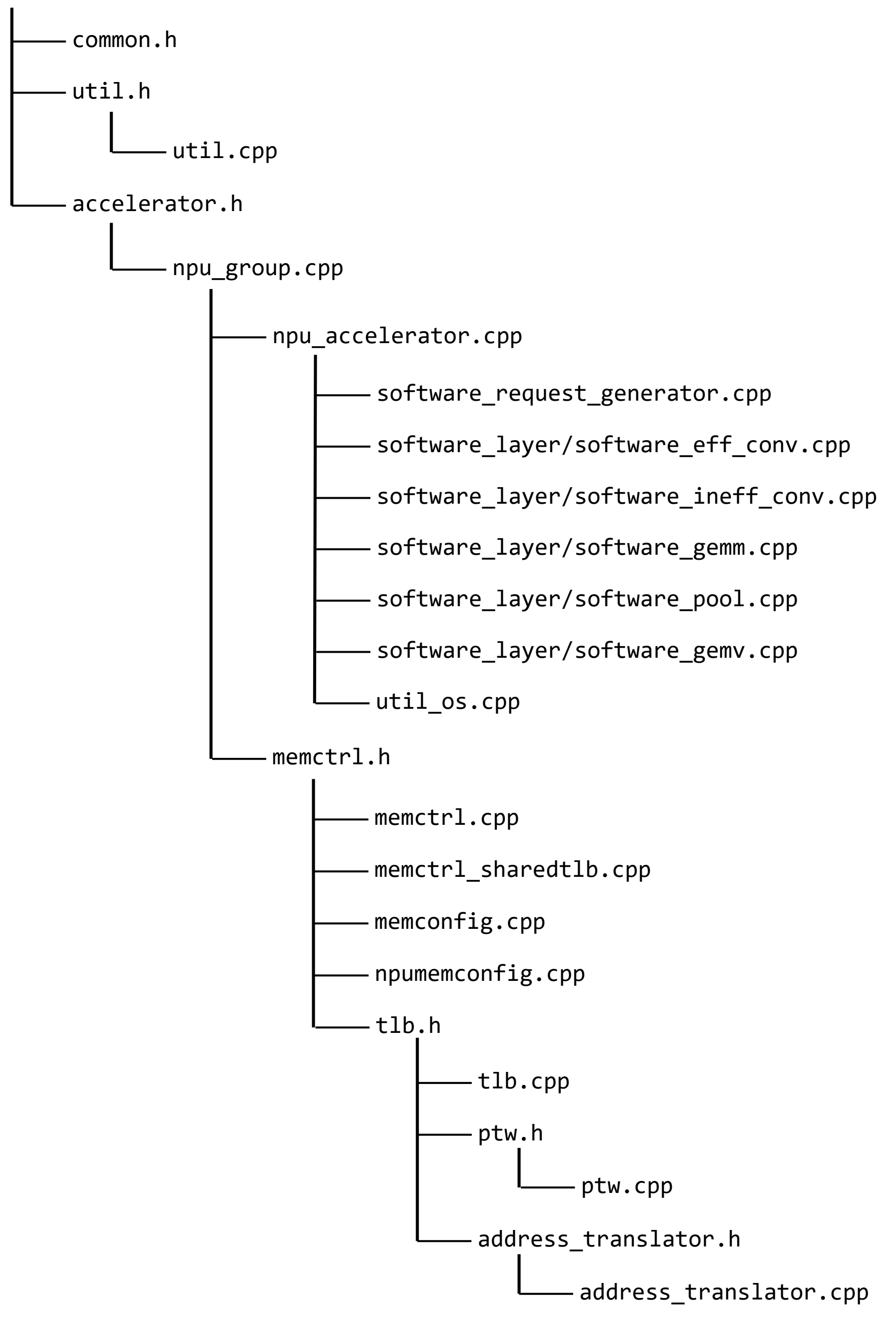

Figure 2: Code structure of mNPUsim.

4.1 NPU Cores

mNPUsim adopts a two-step execution model. In the pre-run step, various transformations are performed to prepare for memory cycle computations. After the pre-run step, in the simulation (run) step, absolute address-based memory requests are generated, computing execution cycle using our DRAMsim3-based memory system.

Pre-run step: There are three transformation to prepare the actual memory requests.

1. Transformation to supported layer operations: Several layer operations are converted into supported layer operations through layer operation transformations. For example, a matrix multiplication of size $(m \times k) \times (k \times n) = (m \times n)$ can be indirectly executed by converting it into a convolution operation with a channel size of k, and input feature map (ifmap) of size $(1 \times m)$, weights of size (1×1), and the number of weight is n. (`software_request_generator::gemm_translation()`)

2. Tiling: Tiling is performed to match the size of the on-chip scratchpad memory (SPM) of the NPU architecture. (`software_request_generator::gemm_translation()`) The tiling algorithm is based on the analytical tile size algorithm proposed in prior research [4]. However, discrepancies may arise between the analytical and actual input/weight/output sizes, particularly at boundaries. This issue is more prevalent in skewed tensors. To address this, mNPUsim supports skewed tiling enlarging the ifmap width of tiles when ifmap height of actual input is smaller than the analytical tile size. Similar enlarging technique is adopted to ifmap width and filter width, and this adjustment prioritizes to ifmap height, ifmap width, and filter width.

3. Absolute address calculation: Initial input addresses are defined using topology-based layer-index. During the pre-run phase, these relative addresses are converted into packed absolute addresses based on the base addresses and sizes of input, weight, and output.

This preparation enables efficient execution during the simulation phase. The pre-run step generates intermediate results, which consist of a new topology that incorporates the aforementioned considerations (layer transformation, tiling, and absolute address calculation) and a list of DRAM requests segmented into DRAM data blocks (cacheline units).
Simulation (run) step: In the simulation (run) step, the memory cycle is calculated based on intermediate results from the pre-run phase. The NPU employs a double buffering mechanism, dividing the simulation into `compute_cycles` and `memory_cycles`. `memory_cycle` is computed by interfacing with our memory system, which is based on DRAMsim. It processes intermediate results in the form of data block request lists to DRAM. The `compute_cycle`, on the other hand, is determined by the local cycle times of computations (`input_local_cycle`) within each tile. Using the calculated `memory_cycle` and `compute_cycle`, memory stalls and compute stalls are managed to ensure parallel execution across multiple NPUs. The simulation records the number of iterations completed for each workload. Once the specified number of iterations (as defined by the misc_input) is reached, the simulation terminates.

4.2 Off-chip Memory

In Figure 2, subtree under `memctrl.h` shows the code structure of off-chip memory system with memory management units.
Memory Controller: The class `MemoryController` is defined in `memctrl.h`. `MemoryController` includes various attributes to model the interface between DRAMsim3 and NPU cores (Section 4.1). `MemoryController` models DRAM access latency and bandwidth with DRAMsim3, and includes MMU subroutines, which will be covered in the following subsection. DRAM read/write requests are sent to DRAMsim3 through `MemoryController::dramRequest()`.
Memory Request Processing:

1. Request generation: The list of absolute addresses (virtual addresses) generated during pre-run steps are transformed into the DRAM request and sent to the memory controller through `MemoryController ::dramRequest()`, packing requests into the TLB queue (`MemoryController::tlbRequest()`).

2. Clock-based execution: `MemoryController::atomic()` is called for every clock tick to process memory requests. When the global clock reached or passed the tick marked in each request (for TLB or DRAM), subroutines for address translation or DRAM transaction are called.

3. Address translation: `NPUMemory::tlbAccess()` performs address translation by calling TLB and page table walker subroutines of Section 4.3 inside `MemoryController::atomic()`.

4. DRAM transaction: `dramsim3::MemorySystem::WillAcceptTransaction()` followed by `dramsim3:: MemorySystem::AddTransaction()` models DRAM access, inside `MemoryController::atomic()`.

5. Callback: The callback function of `dramsim3::MemorySystem` is conjugated with `MemoryController` of mNPUsim, allowing the modeling of DRAM access latency.

4.3 Memory Management Unit

Subtree under `tlb.h` in Figure 2 shows the code structure of the memory management unit, including TLB and the page table walker (PTW).

NPU-side Memory: The class `NPUMemory` is defined in `memctrl.h`. `NPUMemory` includes subroutines for separated TLB modeling, and synchronization of the DRAM clock timing with each NPU core for heterogeneous NPU cores.

TLB: The class `TLB` is defined in `tlb.h`. `TLB` models TLB hit and miss latency, conjugated with page table walker access. As TLB is in charge of address translation in real system, the address translator class is defined in `address_translator.h` (`AddressTranslator`) and referred by `TLB`.

Page Table Walker: The class `PTWManager` manages page table walkers and is defined in `ptw.h`. `PTWManager` models static and dynamic partitioning of page table walkers with page table walk latency.

Page Allocator: The class `DRAMAllocator` is defined in `address_translator.h`. `DRAMAllocator` manages the list of free physical frames, and allocates pages on-demand. `DRAMAllocator` also includes several subroutines for modeling static partitioning of DRAM.

5 Release Note

- 2023.10.02. mNPUsim was released in public (`https://github.com/casys-kaist/mNPUsim`).
- 2023.10.03. "mNPUsim: Evaluating the Effect of Sharing Resources with Multi-Core NPUs" was presented in IISWC 2023.
- 2023.11.30. DRAMsim3 submodule for mNPUsim was updated for the expansion of host memory capacity.
- 2024.12.05. Support for GEMV core (Dot Product Unit) was added.
- 2024.12.10. The whitepaper, "mNPUsim Explained" v1.0.0 was released in public.

6 Acknowledgments

This document was written under the support of the Institute of Information & Communications Technology Planning & Evaluation (IITP) (IITP-2017-0-00466 and RS-2024-00402898).

References

[1] Soojin Hwang, Sunho Lee, Jungwoo Kim, Hongbeen Kim, and Jaehyuk Huh. mNPUsim: Evaluating the Effect of Sharing Resources with Multi-Core NPUs. In *Proceedings of 2023 IEEE International Symposium on Workload Characterization*, 2023.

[2] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, pages 1109–1124, 2020.

[3] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. 19(2):106–109, 2020.

[4] Gordon Euhyun Moon, Hyoukjun Kwon, Geonhwa Jeong, Prasanth Chatarasi, Sivasankaran Rajamanickam, and Tushar Krishna. Evaluating spatial accelerator architectures with tiled matrix-matrix multiplication. *IEEE Transactions on Parallel & Distributed Systems*, 33(04):1002–1014, 2022.

[5] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.