PUMA Off-Chip interconnect

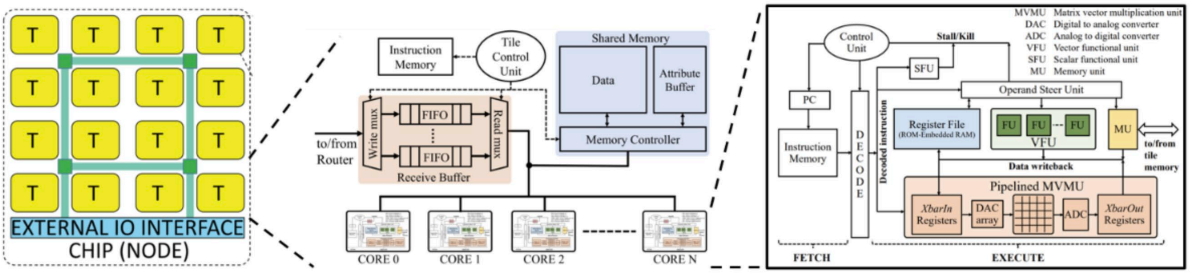


그림 1. PUMA Overview

- PUMA : ISAAC에 microarchitecture들을 추가하여 programmability를 늘리고 ISA와 compiler를 제안.
- PUMA는 Target Neural Network가 하나의 chip에 mapping 되지 않으면 여러 개의 Chip을 연결하여 확장할 수 있음. Chip-to-Chip(off-chip) interconnection interface는 HyperTransport 2.0으로 구성. 따라서 off-chip interconnection 구조를 파악하여 모델링한다면, 추후 여러 개의 chip을 연결하여, 다양한 scale의 Neural Network Model을 acceleration target으로 사용할 수 있을 것이라고 예상됨.
- HyperTransport 구조는 2D mesh를 가정하여 latency 및 Power consumption를 서울대 측에서 부탁 받은 상태.
  - ✓ CACTI로 모델링한 ISAAC의 HyperTransport power consumption이 10.4W라고 논문에 적혀있는데, HyperTransport 상의 packet수에 따라 바뀌는 값인지 확인

ISAAC on-chip interconnect & eDRAM usage

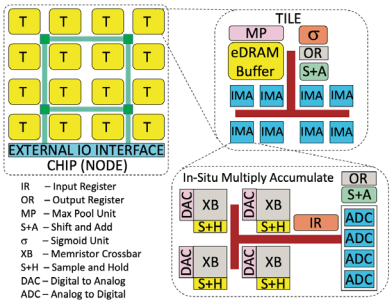


Fig. 2. ISAAC architecture hierarchy.

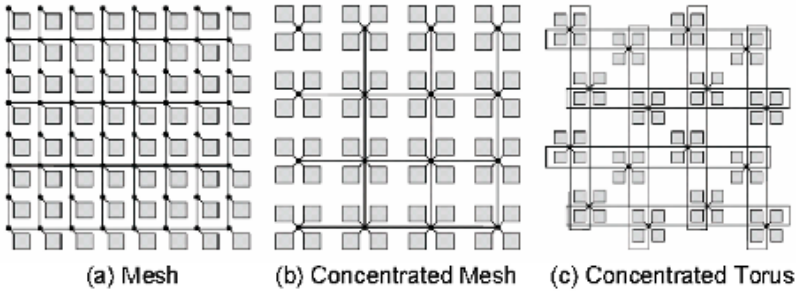


Figure 1. Three examples of NoC topologies [3].

- 하나의 Chip에 12 x 14개의 Tile이 2D c-mesh topology로 연결되어 있음.
- c-mesh는 concentrated mesh의 약자로, 모든 Tile을 서로 연결하는 mesh구조와 달리, Tile 몇 개를 group으로 묶어서 연결하는 구조.
- 각 Tile 내부에는 eDRAM buffers, In-situ Multiply-Accumulate(IMA) units, output registers 가 shared bus로 연결되어 있음.
- On-chip eDRAM에는 synaptic weight matrices that define each neuron layer를 저장.
- ISAAC Tile 내부의 IMA를 구성하는 memristor crossbar array는 synaptic weight를 저장하고, dot-product computation을 수행.
- On-chip Tile-to-Tile NoC의 Bandwidth, Latency, Power consumption이 서울대 측에서 부탁받은 부분임.
- Tiles 0-3은 layer 0을, tiles 4-11은 layer 1을, … 연산하도록 할당하면, tiles 0-3은 layer 0을 위한 모든 weight를 저장하고 layer 0 computation을 수행. 충분한 Layer 0 output이 tiles 4-11의 eDRAM으로 전달되면 tiles 4-11은 layer 1 computation을 수행.
- 따라서 neural network acceleration을 할 때 tile 간 data 이동이 중요한 역할을 할 것이라고 예상되며, Tile-to-Tile interconnection 구조를 modeling하여 parameter들을 알아보려 함.

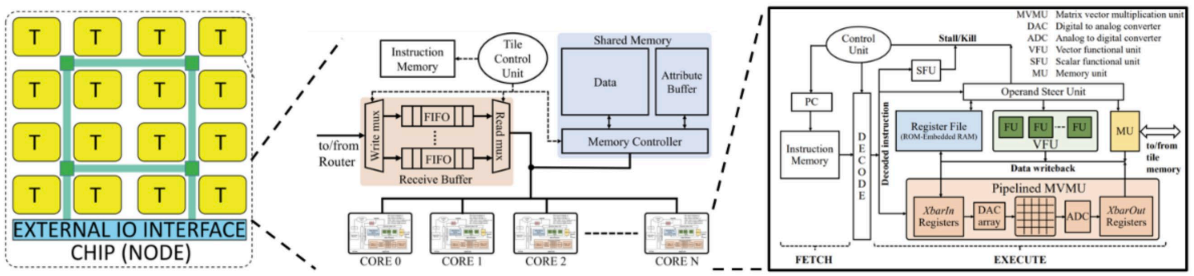


그림 1. PUMA Overview

- PUMA 구조에서 Shared Memory로 표시된 부분은 eDRAM으로 구성되어 있음.
- 각 Tile은 하나의 shared memory, 12개의 Core 등으로 구성되어 있으며, shared memory는 CORE간 data 교환을 담당.
- 각 CORE는 연산에 필요한 데이터를 shared memory에서 load 하거나 연산 결과를 shared memory에 store.

NVSim: nonvolatile memory (NVM) Circuit-Level Performance, Energy, and Area Model Simulation

지원 NVM 목록

- spin-torque-transfer memory (STT-RAM, or MRAM),
- phase-change random-access memory (PCRAM)
- resistive random-access memory (ReRAM)
- legacy NAND Flash

사용 tool

- CACTI: system, network monitoring tool, 일반적인 SRAM, DRAM cache의 performance, energy, area 정보를 estimate 할 때 많이 사용한다.
- Use device data(process parameters) from ITRS report, MASTAR tool

지원 내용

- process node 180 nm, 120 nm, 90 nm, 65 nm, 45 nm, 32 nm to 22 nm
- three transistor types: high performance, low operating power, and low stand-by power
- three transistor sizing choices: optimizing latency, optimizing area, balancing latency&area (기존에 CACTI는 logical effort 사용해서 circuit size, 반면 NVM은 몇몇 transistor의 size가 required driving current에 의해 결정되기 때문에 sizing choices 제공)
- 모든 시뮬레이션 결과는 nominal case, 현재 버전의 NVSim은 process variation은 제공되지 않음
- RC analysis 사용해서 timing, power modeling 진행
    - memory storage element의 equivalent resistance, capacitance를 model하기 위한 look-up table 제작
    - 이를 사용해서 Horowitz's timing model로 delay estimate
    - dynamic energy, leakage power consumption도 계산
    - 전체적인 메모리에 대한 latency, energy는 모든 circuit component의 timing과 power 값을 합쳐서 추정

Memory Hierarchy

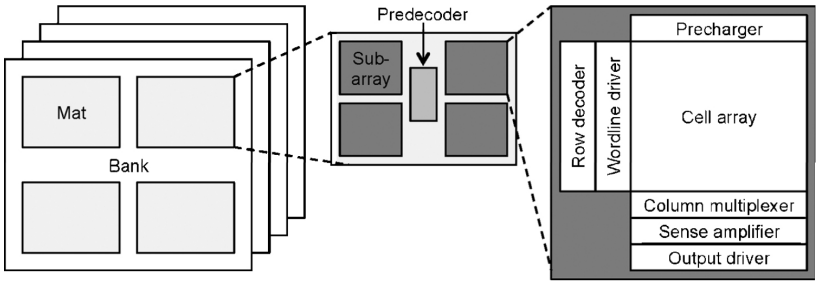


Fig. 6. Memory array organization modeled in NVSim: a hierarchical memory organization includes banks, mats, and subarrays with decoders, multiplexers, sense amplifiers, and output drivers.

NVSim Memory Modeling

- Bank: top-level structure, 하나의 NVM은 여러 개의 bank로 구성되며, 각각의 bank는 독립적으로 동작할 수 있고, 여러 개의 Mat의 H-tree나 bus-like manner로 구성된다.
- Mat: building block of bank, 하나의 bank 속의 Mat들은 memory operation을 동시에 수행한다. 하나의 Mat는 여러 개의 Subarray와 하나의 predecoder block으로 구성된다.
- Subarray: elementary structure, 각각의 Subarray는 row decoders, column multiplexers, and output drivers 같은 peripheral circuitry로 구성된다.

Flexible Activation Mode

1) NMR: number of rows of mat arrays in each bank
2) NMC: number of columns of mat arrays in each bank
3) NAMR: number of active rows of mat arrays during data accessing
4) NAMC: number of active columns of mat arrays during data accessing
5) NSR: number of rows of subarrays in each mat
6) NSC: number of columns of subarrays in each mat
7) NASR: number of active rows of subarrays during data accessing
8) NASC: number of active columns of subarrays during data accessing

eDRAM Simulator DESTINY

링크 : https://code.ornl.gov/3d_cache_modeling_tool/destiny

Required files(destiny/config 디렉토리 내에 존재합니다)

1. **sample_2D_eDRAM.cell** : eDRAM cell의 specification을 수정하여 시뮬레이션의 input으로 사용합니다.

```
-MemCellType: eDRAM

-CellArea (F^2): 33.1
-CellAspectRatio: 2.39

-ReadMode: voltage

-AccessType: CMOS
-AccessCMOSWidth (F): 1.31

-DRAMCellCapacitance (F): 18e-15
-ResetVoltage (V): vdd
-SetVoltage (V): vdd

-MinSenseVoltage (mV): 10
```

2. **sample_2D_eDRAM.cfg** : eDRAM의 capaticy, wordwidth 등의 specification을 수정하여 시뮬레이션의 input으로 사용합니다.

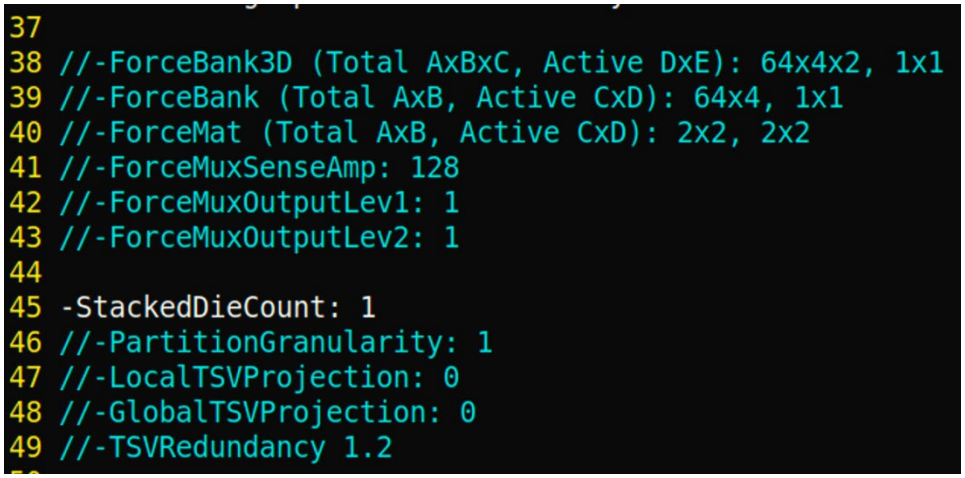
   -PUMA eDRAM data memory capacity = 64KB
   -PUMA는 IBM 45nm SOI technology로 mapping하였으므로 ProcessNode = 45nm
   -eDRAM access bit(wordwidth) = 256 bit

   -Optimization target : Full, ReadLatency, WriteLatency, ReadDynamicEnergy, WriteDynamicEnergy, ReadEDP, WriteEDP, LeakagePower, Area 중 선택할 수 있습니다.

   -Area를 optimization target으로 선택 시
   Total Area = 43332.187um^2
   -Write Latency를 optimization target으로 선택 시 Total Area = 1.179 mm^2

Table 3. PUMA Hardware Characteristics

PUMA Tile at 1GHz on 32nm Technology node				
Component	**Power (mW)**	**Area (mm^2)**	**Parameter**	**Specification**
Control Pipeline	0.25	0.0033	# stages	3
Instruction Memory	1.52	0.0031	capacity	4KB
Register File	0.477	0.00192	capacity	1KB
MVMU	19.09	0.012	# per core dimensions	2 128×128
VFU	1.90	0.004	width	1
SFU	0.055	0.0006	-	-
Core	42.37	0.036	# per tile	8
Tile Control Unit	0.5	0.00145	-	-
Tile Instruction Memory	1.91	0.0054	capacity	8KB
Tile Data Memory	17.66	0.086	capacity technology	64KB eDRAM
Tile Memory Bus	7	0.090	width	384 bits
Tile Attribute Memory	2.77	0.012	# entries technology	32K eDRAM
Tile Receive Buffer	9.14	0.0044	# fifos fifo depth	16 2
Tile	373.8	0.479	# per node	138
On-chip Network	570.63	1.622	flit_size # ports conc	32 4 4
Node	62.5K	90.638	-	-
Off-chip Network (per node)	10.4K	22.88	type link bandwidth	HyperTransport 6.4 GB/sec



3. 시뮬레이션 실행

1) Root directory에서 DESTINY를 compile합니다.
   $ make
2) 시뮬레이션할 .cfg 파일과 함께 시뮬레이터를 실행합니다.
   $ ./destiny sample_2D_eDRAM.cfg

시뮬레이션 결과로는 Total Area, Read Latency, Write Latency, Refresh Latency, Read Bandwidth, Write Bandwidth, Read Dynamic Energy, Write Dynamic Energy, Refresh Dynamic Energy 등의 parameter들과 얻을 수 있습니다.

```
=============
   SUMMARY
=============
Optimized for: Area
Memory Cell: Embedded DRAM
Cell Area (F^2)    : 33.100 (8.894Fx3.721F)
Cell Aspect Ratio  : 2.390

=============
CONFIGURATION
=============
Bank Organization: 32 x 1
 - Row Activation   : 1 / 32
 - Column Activation: 1 / 1
Mat Organization: 2 x 2
 - Row Activation   : 1 / 2
 - Column Activation: 1 / 2
 - Subarray Size    : 4 Rows x 1024 Columns
Mux Level:
 - Senseamp Mux      : 1
 - Output Level-1 Mux: 1
 - Output Level-2 Mux: 4
Local Wire:
 - Wire Type : Local Aggressive
 - Repeater Type: No Repeaters
 - Low Swing : No
Global Wire:
 - Wire Type : Global Aggressive
 - Repeater Type: No Repeaters
 - Low Swing : No
Buffer Design Style: Latency-Optimized
```

```
=============
   RESULT
=============
Area:
 - Total Area = 763.423um x 372.186um = 284135.366um^2
 |--- Mat Area      = 23.857um x 372.186um = 8879.230um^2   (12.368%)
 |--- Subarray Area = 11.928um x 183.841um = 2192.939um^2   (12.519%)
 - Area Efficiency = 12.368%
Timing:
 -  Read Latency = 348.803ps
 |--- Non-H-Tree Latency = 70.152ps
 |--- Mat Latency    = 278.651ps
    |--- Predecoder Latency = 75.981ps
    |--- Subarray Latency   = 202.670ps
       |--- Row Decoder Latency = 174.502ps
       |--- Bitline Latency     = 13.156ps
       |--- Senseamp Latency    = 2.090ps
       |--- Mux Latency         = 3.406ps
       |--- Precharge Latency   = 9.955ps
 - Write Latency = 348.803ps
 |--- Non-H-Tree Latency = 70.152ps
 |--- Mat Latency    = 278.651ps
    |--- Predecoder Latency = 75.981ps
    |--- Subarray Latency   = 202.670ps
       |--- Row Decoder Latency = 174.502ps
       |--- Charge Latency      = 1.118ps
 - Refresh Latency = 1.816ns
 |--- Non-H-Tree Latency = 70.152ps
 |--- Mat Latency    = 1.746ns
    |--- Predecoder Latency = 75.981ps
    |--- Subarray Latency   = 797.056ps
 - Read Bandwidth  = 839.391GB/s
 - Write Bandwidth = 157.892GB/s
```

```
Power:
 -  Read Dynamic Energy = 64.708pJ
 |--- Non-H-Tree Dynamic Energy = 59.256pJ
 |--- Mat Dynamic Energy    = 5.452pJ per mat
    |--- Predecoder Dynamic Energy = 0.080pJ
    |--- Subarray Dynamic Energy   = 5.372pJ per active subarray
       |--- Row Decoder Dynamic Energy = 0.159pJ
       |--- Mux Decoder Dynamic Energy = 0.511pJ
       |--- Senseamp Dynamic Energy    = 2.847pJ
       |--- Mux Dynamic Energy         = 0.152pJ
       |--- Precharge Dynamic Energy   = 1.389pJ
 - Write Dynamic Energy = 60.740pJ
 |--- Non-H-Tree Dynamic Energy = 59.256pJ
 |--- Mat Dynamic Energy    = 1.484pJ per mat
    |--- Predecoder Dynamic Energy = 0.080pJ
    |--- Subarray Dynamic Energy   = 1.404pJ per active subarray
       |--- Row Decoder Dynamic Energy = 0.159pJ
       |--- Mux Decoder Dynamic Energy = 0.511pJ
       |--- Mux Dynamic Energy         = 0.152pJ
 - Refresh Dynamic Energy = 2.491nJ
 |--- Non-H-Tree Dynamic Energy = 2.415nJ
 |--- Mat Dynamic Energy    = 75.988pJ per mat
    |--- Predecoder Dynamic Energy = 57.150pJ
    |--- Subarray Dynamic Energy   = 18.839pJ per active subarray
       |--- Row Decoder Dynamic Energy = 0.000pJ
       |--- Senseamp Dynamic Energy    = 2.847pJ
       |--- Precharge Dynamic Energy   = 1.389pJ
 - Leakage Power = 77.897mW
 |--- Non-H-Tree Leakage Power = 0.000pW
 |--- Mat Leakage Power        = 2.434mW per mat
 - Refresh Power = 62.272uW
```

Refresh Power는 eDRAM의 retention time을 기반으로 계산되는데, config 파일의 retention time 값을 인식하는 데 오류가 있음을 확인하고, 수정한 상태입니다.

Retention time은 40us를 기준으로 시뮬레이션하였으나, Result.cpp의 line 658에서 다른 값으로 수정할 수 있습니다.

```
657     if (cell->memCellType == eDRAM) {
658         cout << string(indent, ' ') << " - Refresh Power = " << TO_WATT(bank->refreshDynamicEnergy / (40/1e6)) << endl;
659     }
```