Edge-AI Accelerator in 2-nm Technology

June 20th 2025

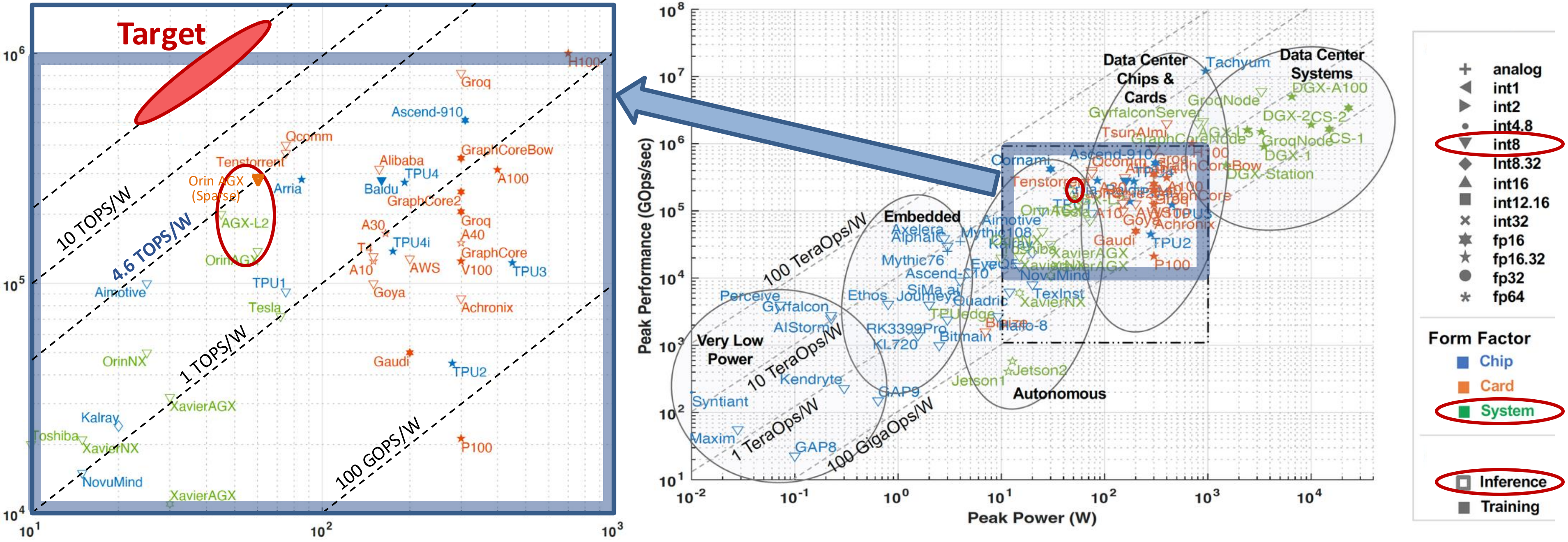
Fumio Arakawa
d.lab, The University of Tokyo

Outline

- Introduction
  - Various AI Accelerators and Our Target
- Project Overview
  - Development items: 1) AI-accelerator architecture, **2) Accelerator Chip,**
    3) CPU Chip, 4) Software development kit
- Background
  - Large Language Model (LLM) – Large Matrix Processing
- Our Approach
  - **Data-dependent division** & **broadcast of load data**
- Our AI Accelerator Overview
- Summary

Various AI Accelerators and Our Target

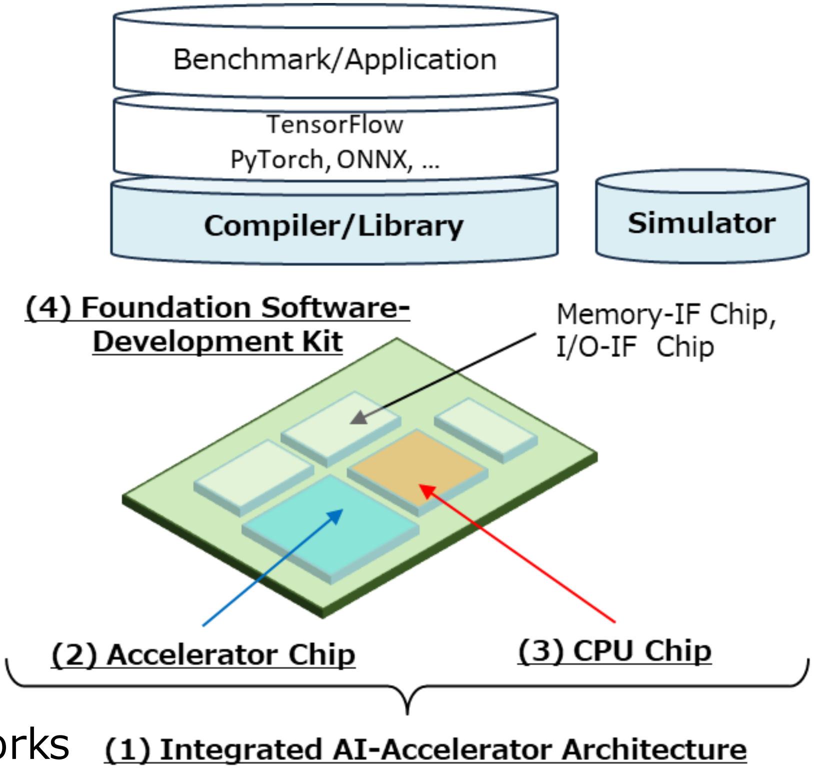
- Many Accelerators have been announced by NVIDIA, Google, Tenstorrent, etc.
- NVIDIA's **Jetson AGX Orin:** 138 (275) TOPS, 60 W, **2.3 (4.6) TOPS/W**, Int8 (w/ sparse accel.)
  DRIVE AGX L2: 200 TOPS, 45 W, **4.4 TOPS/W**, Int8
- Our PJ target is to achieve 5-times higher efficiency.



source: Albert Reuther, et al., "AI and ML Accelerator Survey and Trends," 2022 IEEE High Performance Extreme Computing (HPEC) Conference (Oct 2022)

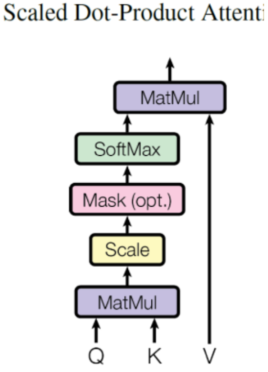
Project Overview

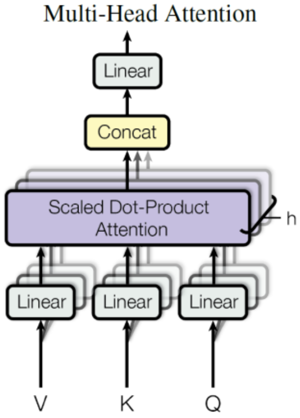
- Leading-edge Semiconductor Technology Center (LSTC)
  - member: AIST, Rapidus, The Univ. of Tokyo
- International collaboration with Tenstorrent inc.
- Development items
  - ① **AI-accelerator architecture**
    - integrating accelerator and CPU chips
    - HW-SW codesign
  - ② **Accelerator chip**
    - New highly-efficient Architecture
  - ③ **CPU chip**
    - conforming to RISC-V ISA
  - ④ **Software development kit**
    - conforming to de-fact-standard AI frameworks

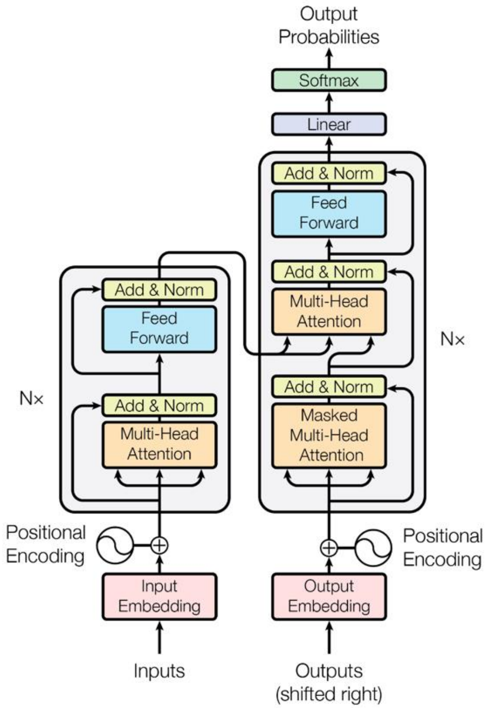


Background

- Large Language Model (**LLM**), an important **AI** application
  - multiplication of **large** matrices (ex. 4,096-by-4,096) with **many** arithmetic units
- Transfer the data **timely to registers**
  - Unable to keep all large-matrix data near the arithmetic units
- Cost and Power in the advanced process (We use **2nm**)
  - calculation : decreased
  - transfer: **relatively increased**



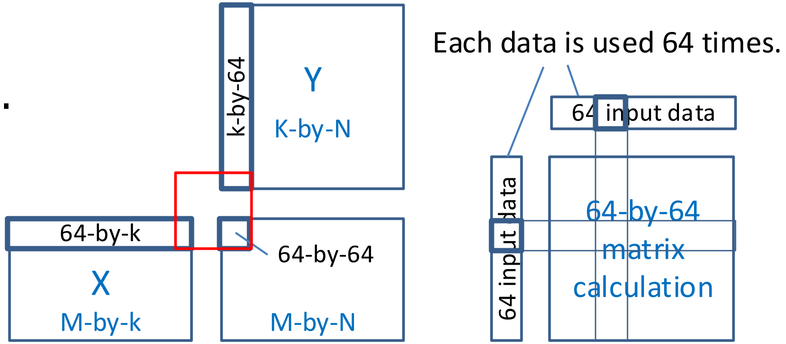




Efficient data transfer: more and more important !!

Matrix processing

- **Less data transfer** per calculation ➡ Many AI accelerators have matrix units
  - vs. Scalar/Vector Processing
  - Difference is particularly noticeable in **large matrices**.
- Data transfer (vs. scalar processing)
  - 65/128 [1)] ≅ **1/2** for 64-parallel **vector** processing
  - **1/64** [2)] for 64-by-64 **matrix** processing

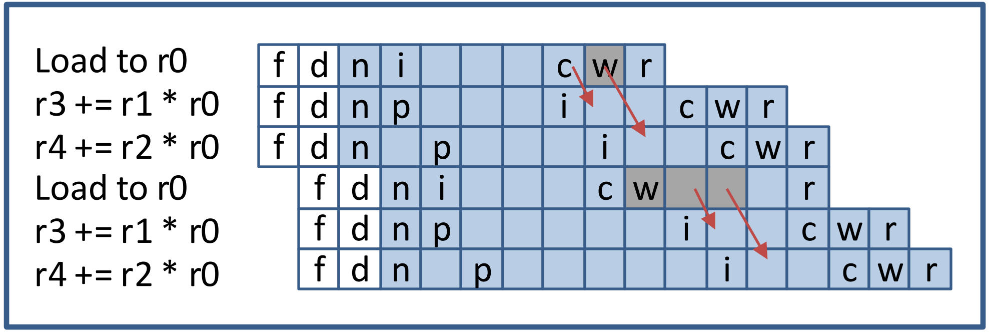


1) X*Y calculates MN sums of k products, ie. **MNk** calculations.
   Scalar: 2-operand transfer/calculation, ie. **2MNk** transfers
   n-parallel **vector**: (1+n)-operand transfer/(n calculations), ie. **(1+1/n) MNk** transfers
   For n=64: (1+1/64)/2 = 65/128
2) m-by-n **matrix**: (m+n)-operand transfer/(mn calculations), ie. **(1/n+1/m) MNk** transfers
   For n=m=64: (1/64+1/64)/2 = 1/64

Conventional Approach

- **Hierarchical** Memory Structure
  - from **high-speed small (L1)** to low-speed large (Main Memory)
  - achieving both high speed and large capacity at system level.
- For power efficiency
  - lowering supply voltage for **logic parts**
  - cannot lowering it for **SRAM**
  - Should not use L1 memory/cache
- **out-of-order** processing
  - memory latency can be hidden

    ➔ L1 memory is no longer essential.
  - Instead, **large and high-speed register files** and complex control are required.



Long register lifetime for out-of-order processing

f: fetch, d: decode, n: rename, p: dispatch,
i: issue, c: complete, w: write, r: retire

Our Approach

- **Data-dependent division** & **multicore** assignment of processing
  - Divide & Assign processing to **transfer** (memory load & store) **& execution cores.**
  - Conventionally, only data-**in**dependent division is allowed. ⬅ Independent processing & Sync.
- A transfer core **broadcasts** memory load data to **multiple execution** cores.
  - Further **enlarge** the matrix processing
  - Broadcasting to **4 cores** (Each has a 64-by-64 matrix unit) ➡ 256-by-64 matrix processing
  - Transfers: **1/100** $\cong (1/64 + 1/256)/2 = 5/512$ (vs. Single core with a Scalar unit)

cf.) w/o Broadcast: No Transfer Decrease

- **Copy/Cache**: to local memories, and loaded to registers ➡ **Memory Footprint Increase**
- Each core fetch data directly form **Shared memory** ➡ **Long Latency, Access Conflict**

Data-dependent division

- Significant effect of **reducing data loads** and **memory footprint**

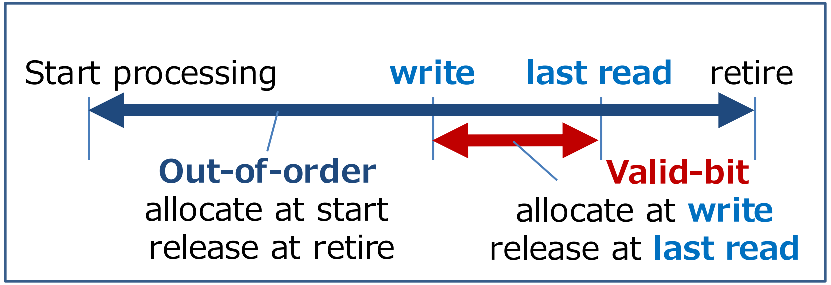
- Order of data definition and use
  - Single core w/ Single flow: Naturally defined ➡ collapsed by Data-dependent division

- Order definition by **Valid bit** of each register
  - **Setting**/**Clearing** valid bit when data is **written**/**lastly read**, respectively
  - Wait **write**/**read** until register becomes **invalid**/**valid**, respectively

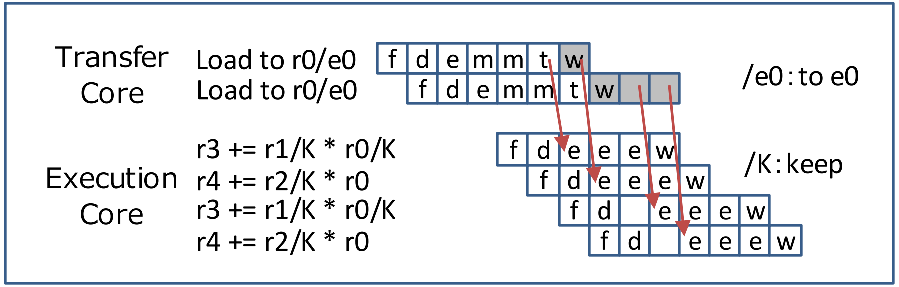
- **This ensures correct operation of data-dependent multicore processing.**

Short Register Lifetime

- Allocate a register only while register value is necessary (**write** to **last read**).
- **Overrun buffer** ensure correct operation even if register allocation is failed.
- Perfect load timing control by software with hardware assistance is also ensure it.
- Hardware can identify "**last read**" by "**valid**" bit and "**keep**" option.



Short register lifetime of valid & keep method



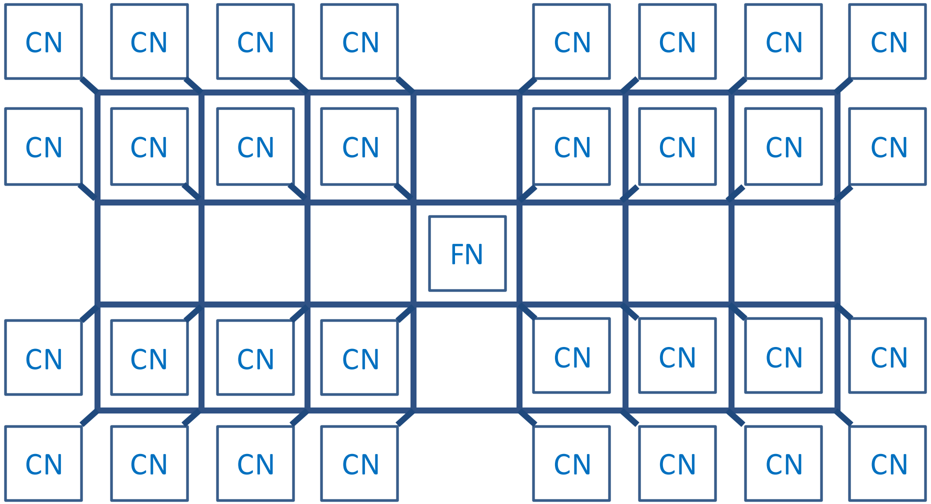
Example pipeline flow of short register lifetime

- **Drastically reduce # of registers required, compared to out-of-order method.**

Compute Node Array Overview

- Compute node (CN): EC + TC + SM (Shared memory)
  - Execution Core (EC): Max 64x4x64 matrix contraction per 4 cycles
  - Transfer Core (TC): EC-sustainable data load & Broadcast
- Flow control node (FN): FCC + IM (Instruction memory)
  - Flow control core (FCC):

    Load and dispatch instructions to all CNs.
- Connect 32 CNs by NoC.



Summary

- Large Language Model (LLM) Support
- **Efficient data transfer is more and more important.**
- 64-by-64 **Matrix processing** per 4 cycles
  - **Less data transfer** per calculation ➡ **1/64**
- **Data-dependent division** & **multicore** assignment of processing
  - Order definition by **Valid bit** of each register
- A transfer core **broadcasts** memory load data to **multiple execution** cores.
  - Significant effects of **reducing data loads** and **memory footprint**
- **Short Register Lifetime** ➡ **Drastically reduce # of registers required**
- Connect 32 Compute nodes (CNs) and a Flow control node (FN) by NoC
  - CN: Execution Core (EC) + Transfer Core (TC) + Shared memory (SM)

Acknowledgement

- This work is based on results obtained from a project, JPNP20017*), commissioned by the New Energy and Industrial Technology Development Organization (NEDO)

Thank you !!

*) Research and Development Project of the Enhanced infrastructures for Post-5G Information and Communication Systems