OSCAR Vector Multicore System Platinum Vector Accelerator on FPGA

Kazuki Fujita・Kazuki Yamamoto・Honoka Koike・
Toshiaki Kitamura・Keiji Kimura・Hironori Kasahara

Waseda University

Background

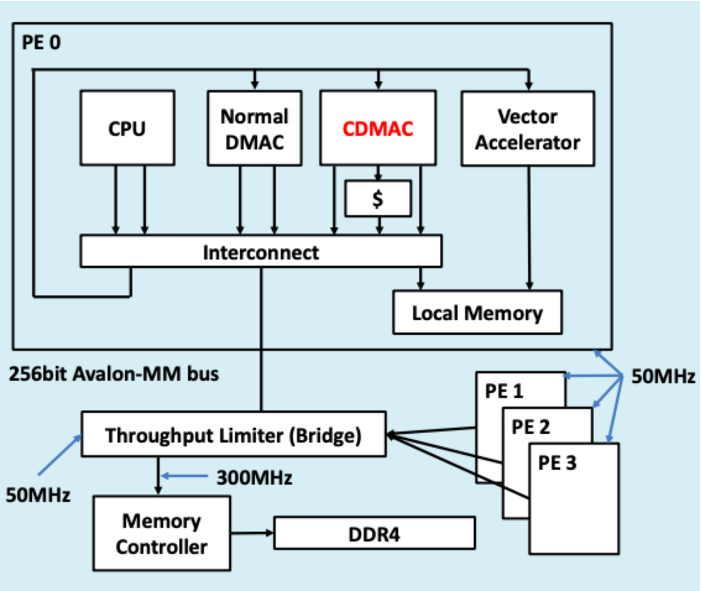
- The demand for accelerating image processing and machine learning application
  - these application has high data-parallelism
- Accelerators are used to speed-up these applications
  - Ex) GPGPU, FPGA…
  - special codes are required to use these accelerators

Platinum Multicore Architecture

- Vector Accelerator can speed-up data-intensive application.
- There is no need for writing code to use vector Accelerator (Automatically produce code by using OSCAR Compiler)

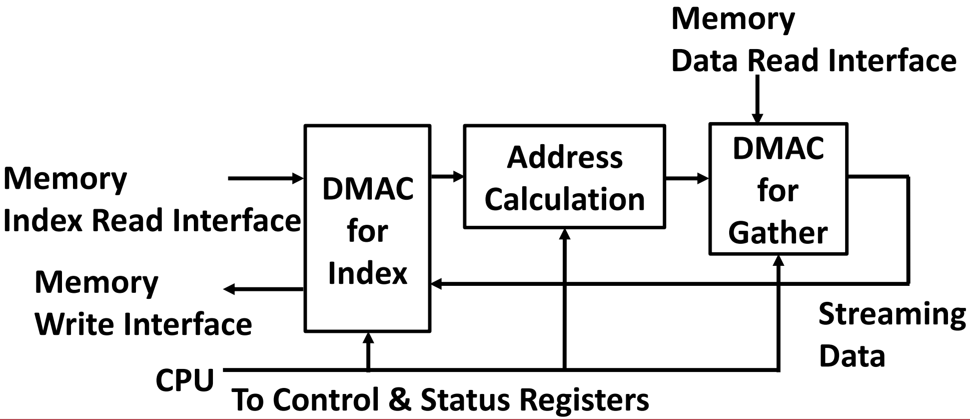
Platinum Multicore Architecture

- **Each PE has CPU and Vector Accelerator**
  - reduce CPU ⇔ VA communication time
- Vector Accelerator
  - accelerate **data-intensive application**
    - Machine Learning, Deep Learning, Scientific and Technological Execution…
- DMA Controller(Normal DMAC)
  - **directly access** DDR4⇔Local Memory
  - computation and data-transfer exec **simultaneously**
- **Cascaded DMA Controller(CDMAC)**
  - accelerate **indirect access memory transfer**
    - Ex) out_arr[i] = data_arr[indices[i]]



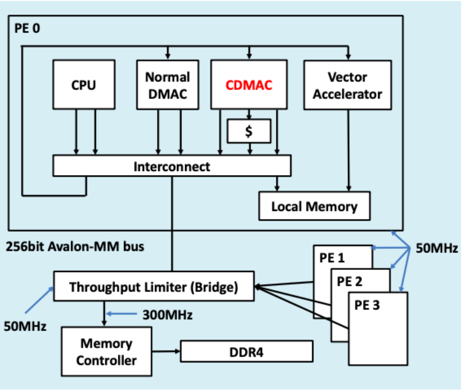
Cascaded DMA Controller

- Normal DMAC
  - transfer data for accelerator from/to main memory
- Difficult to speed-up application include indirect access
  - indirect memory access: out_arr[i] = data_arr[indices[i]]
  - application handling sparce matrices
- Using CDMAC to speed-up indirect access
  - combining streaming address calculation and DMA components
  - with cache memory
    - exploit memory locality of scattered data

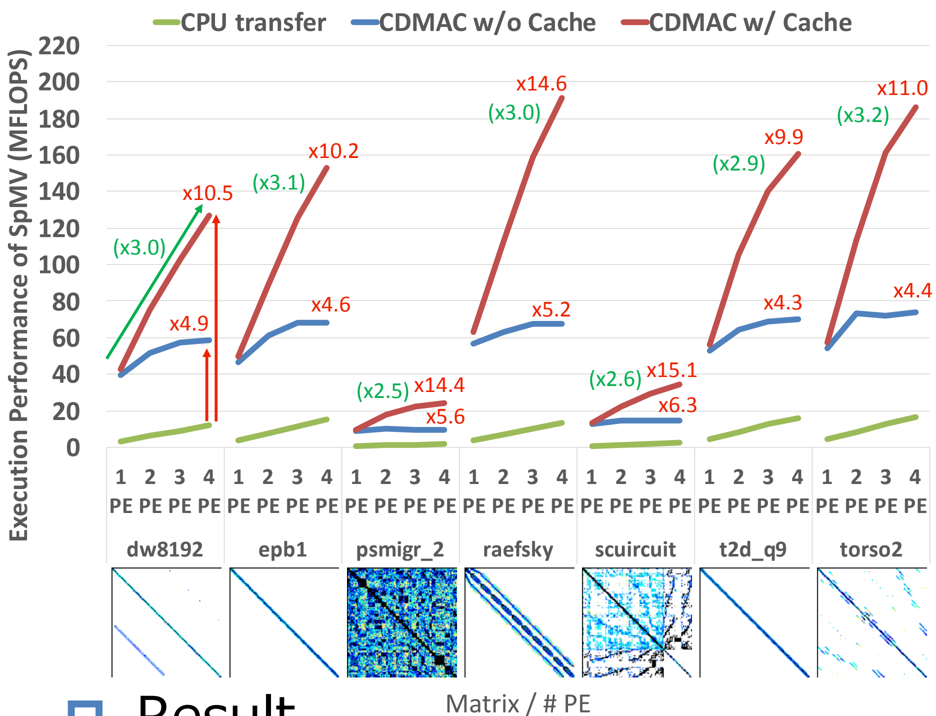


Evaluation Environment

- Platinum Multicore implemented on FPGA
  - Board: De5a-Net DDR4
    (Intel FPGA Arria10)
  - Vector Accelerator Specification
    - 16 single precision ops/cycle
    - Local Memory Bandwidth 32byte/clock
    - All data located on Local Data Memory
    - Local Data Memory size: 32KB
  - CPU (NIOS II/f)
    - Compiler: nios2-elf-gcc
    - FPU: Floating Point Hardware 2
    - Cache: 32KB

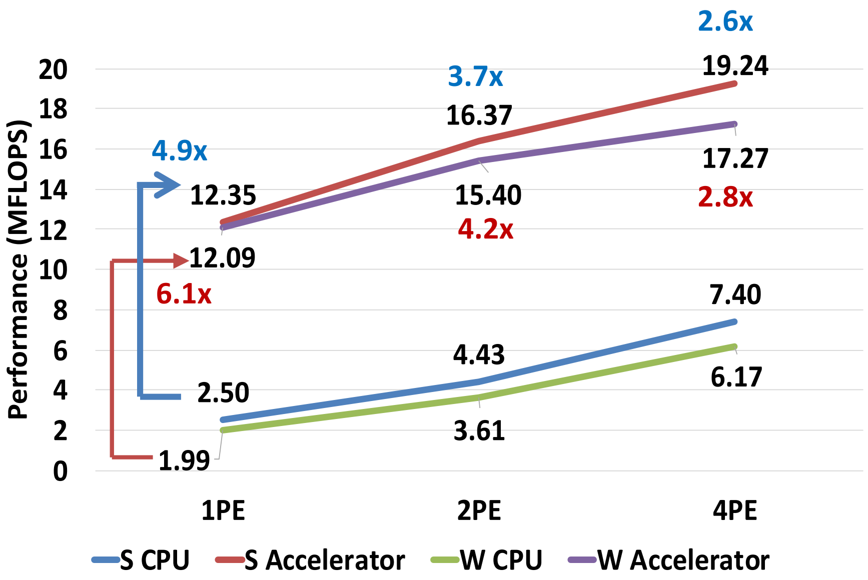


Experimental Evaluation: Sparse Matrix vector Multiplication



- Application
  - sparse matrices from SuiteSparse Matrix Collection
  - sparse matrices are stored in SELL format
  - using floating point and 32-bit integer
- Result
  - **CDMAC shows speedups up to 14.6x compared to CPU execution**
  - band matrices show better performance
    - suitable for structural calculation

Experimental Evaluation: NAS Parallel Benchmark CG



- Application
  - NAS PARALLEL Benchmark CG
    - SIZE S, W
- Code Modification
  - convert Fortran to C
  - CSR format → SELL format
  - parallelized using software coherent cache
- Result
  - maximum speedups using CDMAC compared to only using CPU
    - SIZE S: **6.1x**(1PE), SIZE W: **4.9x**(1PE)

Speed-up solving linear equation A**x = b**
by using Vector Accelerator and CDMAC

Conclusion

- Platinum Multicore Architecture
  - speed-up data-intensive application by using Vector Accelerator
    - aim for Machine Learning, Deep Learning, Scientific and Technological Execution…
  - using Cascaded-DMA Controller(CDMAC) to speed-up application include Indirect Access
    - indirect Access: out_arr[i] = data_arr[indices[i]]
    - sparse matrix vector multiplication
    - NAS Parallel Benchmark CG (solving linear equation A**x**=**b**)

- The maximum speed-ups of using CDMAC and Vector Accelerator compared to only using CPU
  - sparse matrix vector multiplication: **14.6x**
  - NAS Parallel Benchmark CG: (SIZE S) **6.1x** ・ (SIZE W) **4.9x**