A Universal FPGA-based Floating-point Matrix Processor for Mobile System

Wenqiang Wang[1], Kaiyuan Guo[1], Mengyuan Gu[1], Yuchun Ma[2], Yu Wang[1]
[1]Department of EE, Tsinghua University, Beijing, China
[2]Department of CS, Tsinghua University, Beijing, China

NICS Nano-scale Integrated Circuit and System Lab，
Department of Electronic Engineering, Tsinghua University

Outline

- Introduction
- Implementation
- Experimental result
- Conclusion

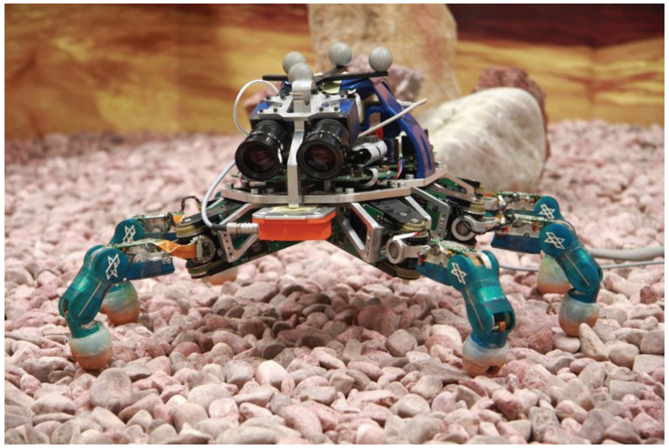
Outline

- Introduction
- Implementation
- Experimental result
- Conclusion

Background

- Mobile system:
  UAV, walking robots …
- Requires low power cost
- Requires high performance
  Computer vision, AI …
  Real-time processing

- Many algorithms involves matrix computation:
  SLAM
  Image Processing
  3D Registration

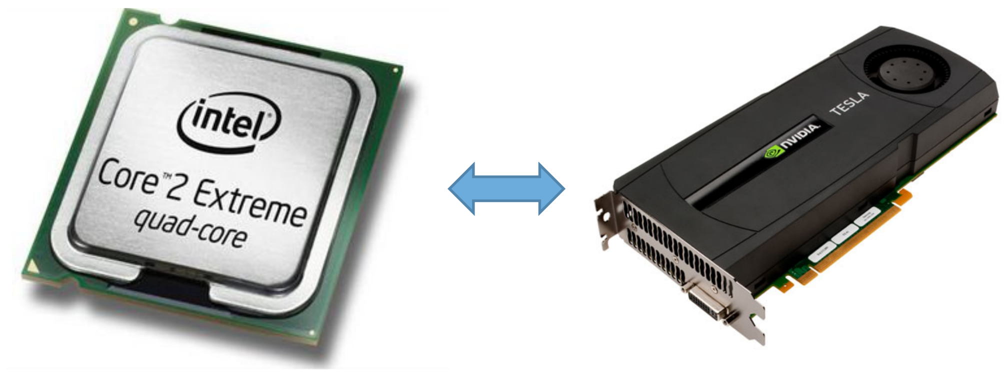


Target

- A matrix processor with:
  a) Low power cost for mobile system
  b) High flexibility for different applications
  c) High performance to support complicated tasks

Related work

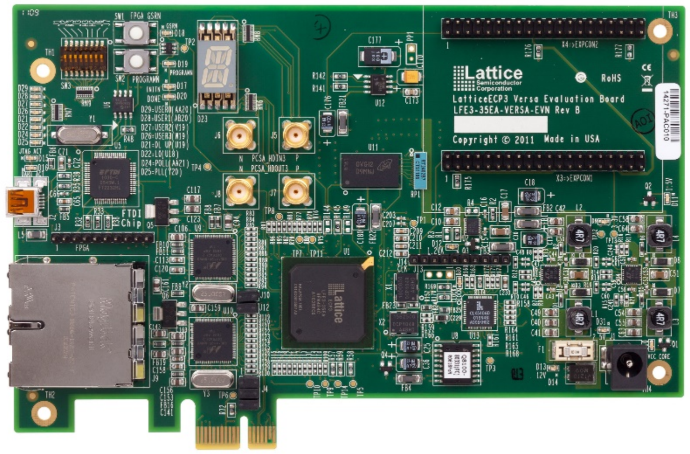
- cuBLAS and MKL
  CPU-GPU platform costs much power

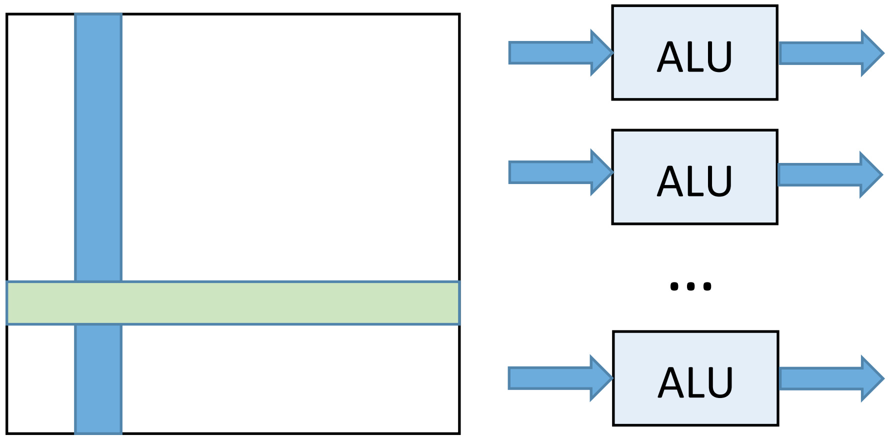


- ASIC chips
  low flexibility

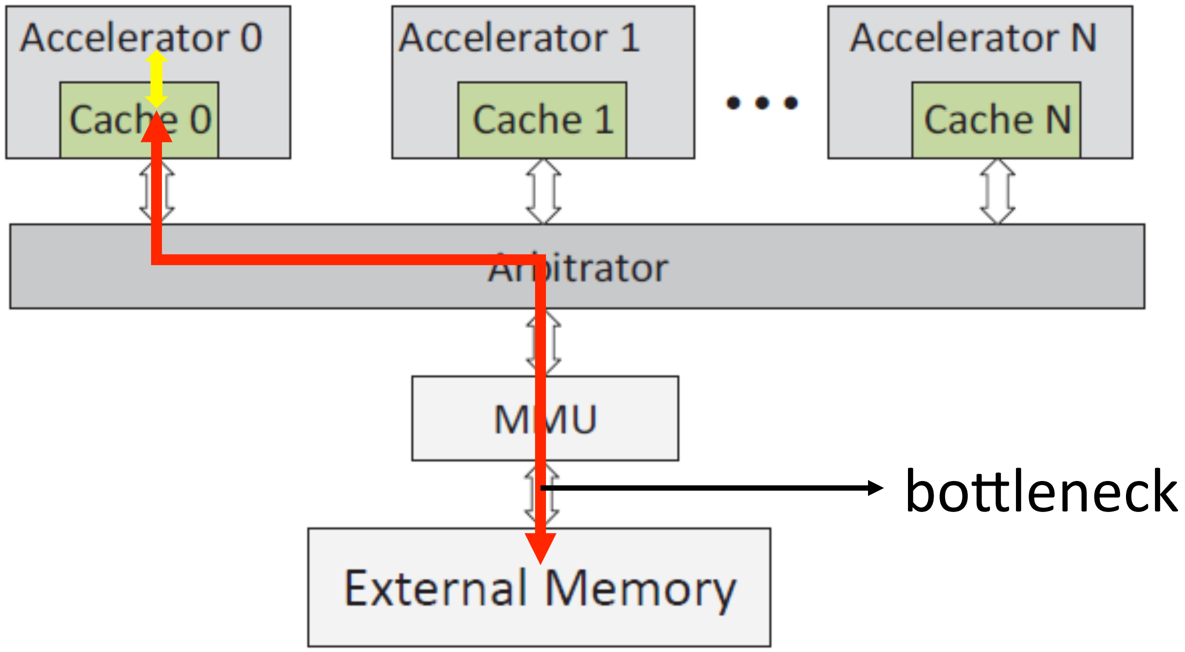
Related work

- FPGA platform
- Vector Processor [C. H. Chou, FPGA'11]
  - Data access problem
  - Workload for master processor





- Accelerator
  Dedicated accelerator for one operation, e.g. matrix multiplication.[T. -C. Lee, WCECS'13]
- A straight forward implementation:
  Accelerators communicate through External Memory



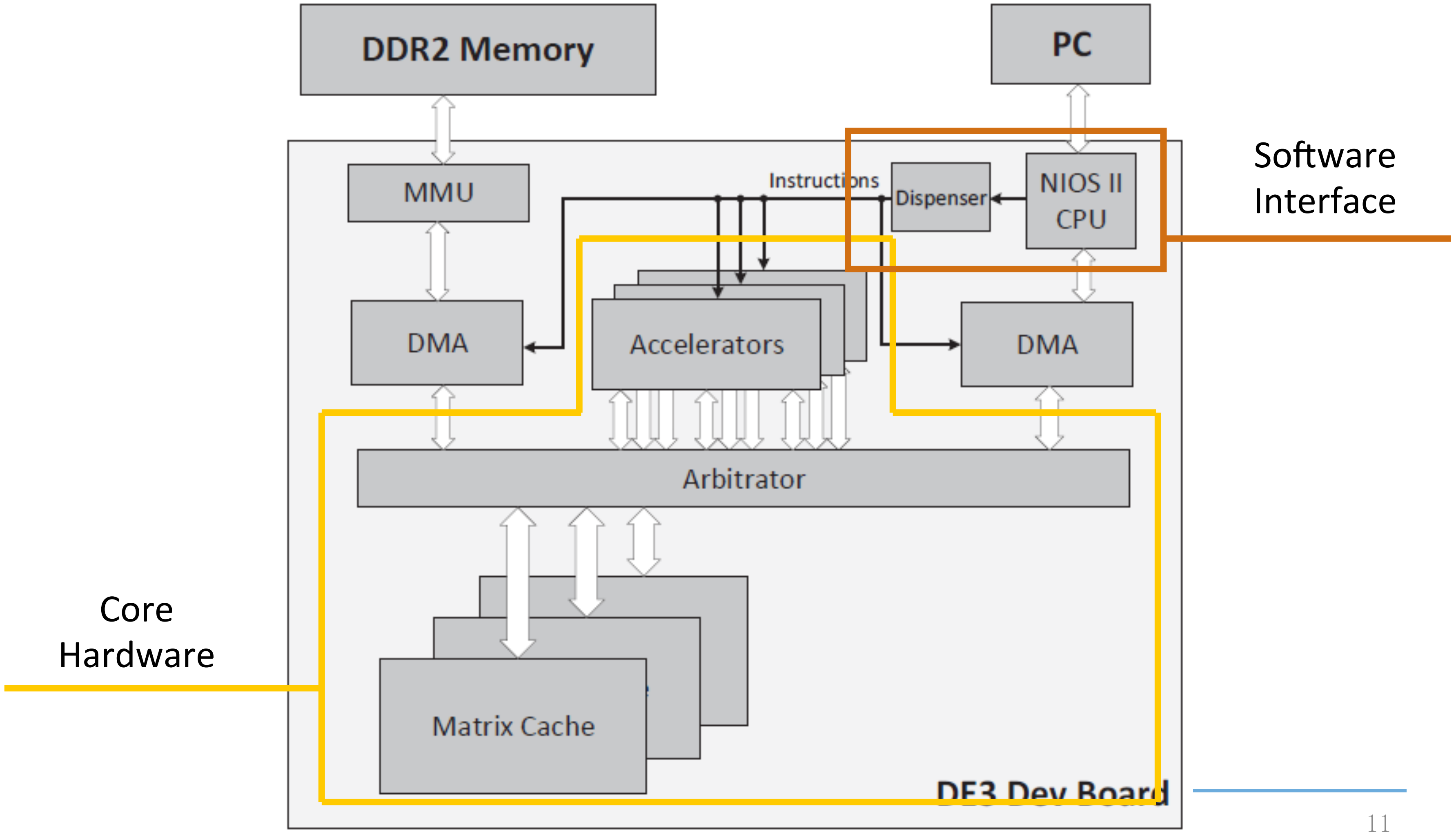
Contributions

- Integration of different accelerators
- A software programming interface with matrix level instructions
- Minimize the workload of master processor

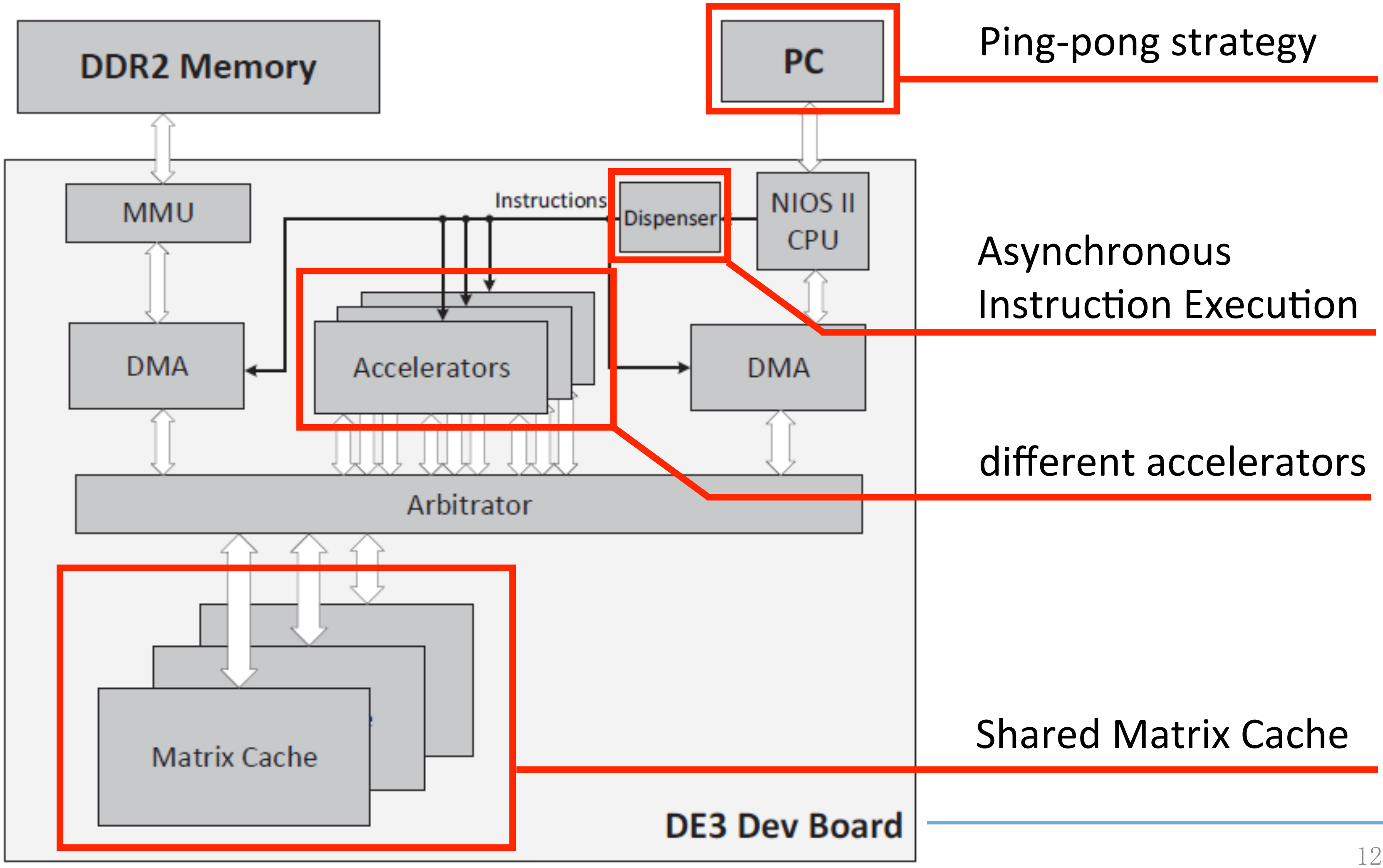
Outline

- Introduction
- Implementation
- Experimental result
- Conclusion

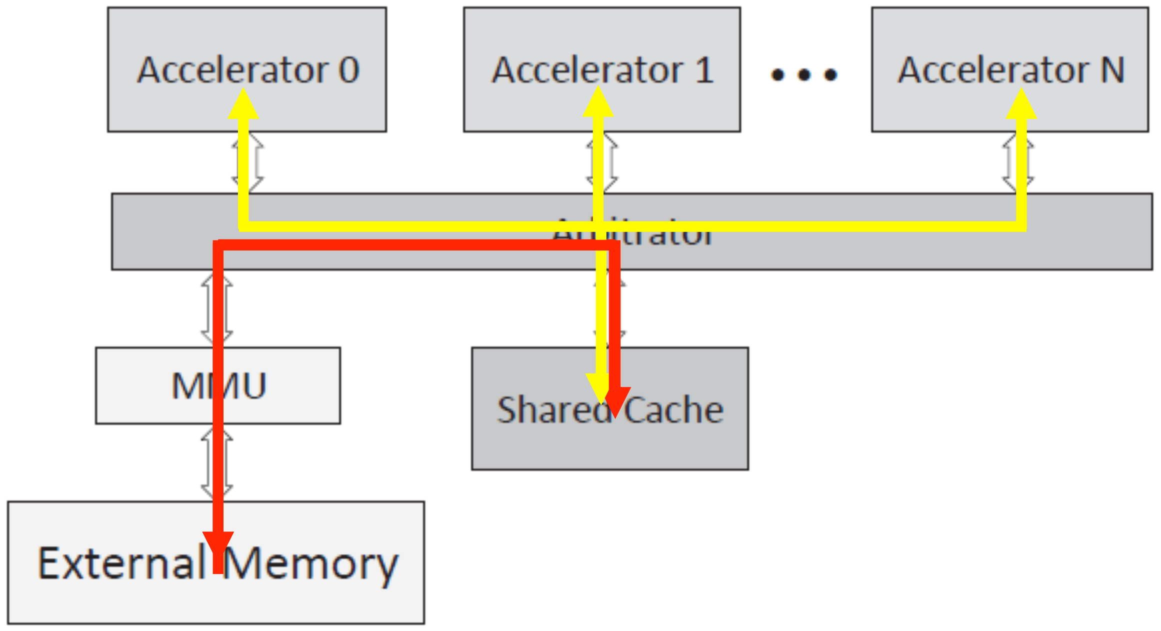
System Description



System Description



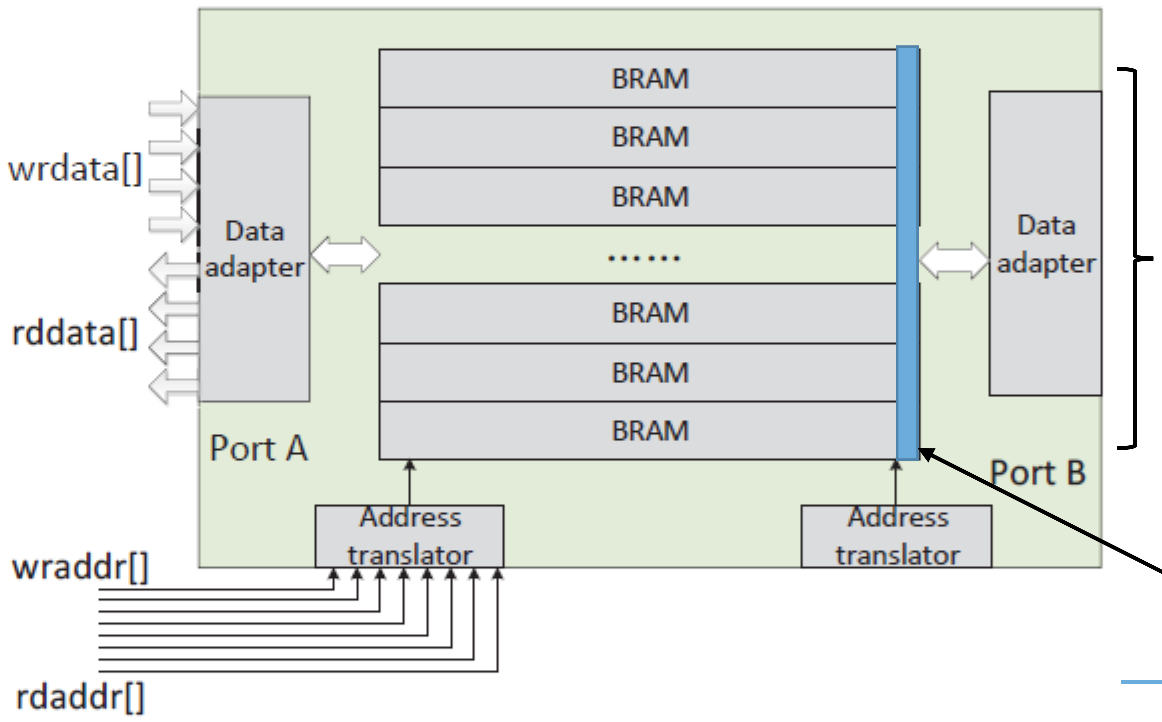
Shared Matrix Cache



- Use a shared cache for the communication
- Reduce the I/O with external memory

Shared Matrix Cache

- Targets:
  a) High bandwidth
  b) Supports 2D parallel data access
  c) Supports different patterns of data access



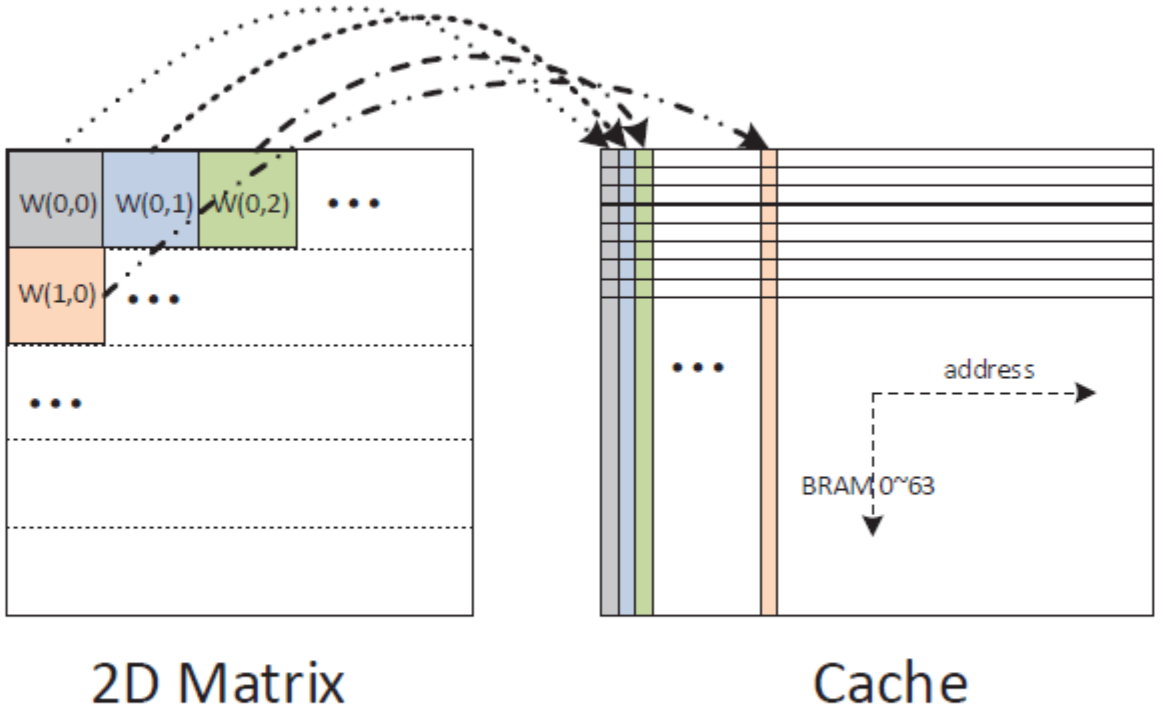
Bandwidth:
Number of BRAM

How to arrange these data?

- The cache is treated as a big 2D matrix
- Divided by 8*8 small windows:

Window ID	⟷	address in BRAM
Relative position	⟷	ID of BRAM

- Four patterns of data access:
  a) Pixel 1*1
  b) Column 8*1
  c) Row 1*8
  d) Window 8*8



Matrix Accelerators

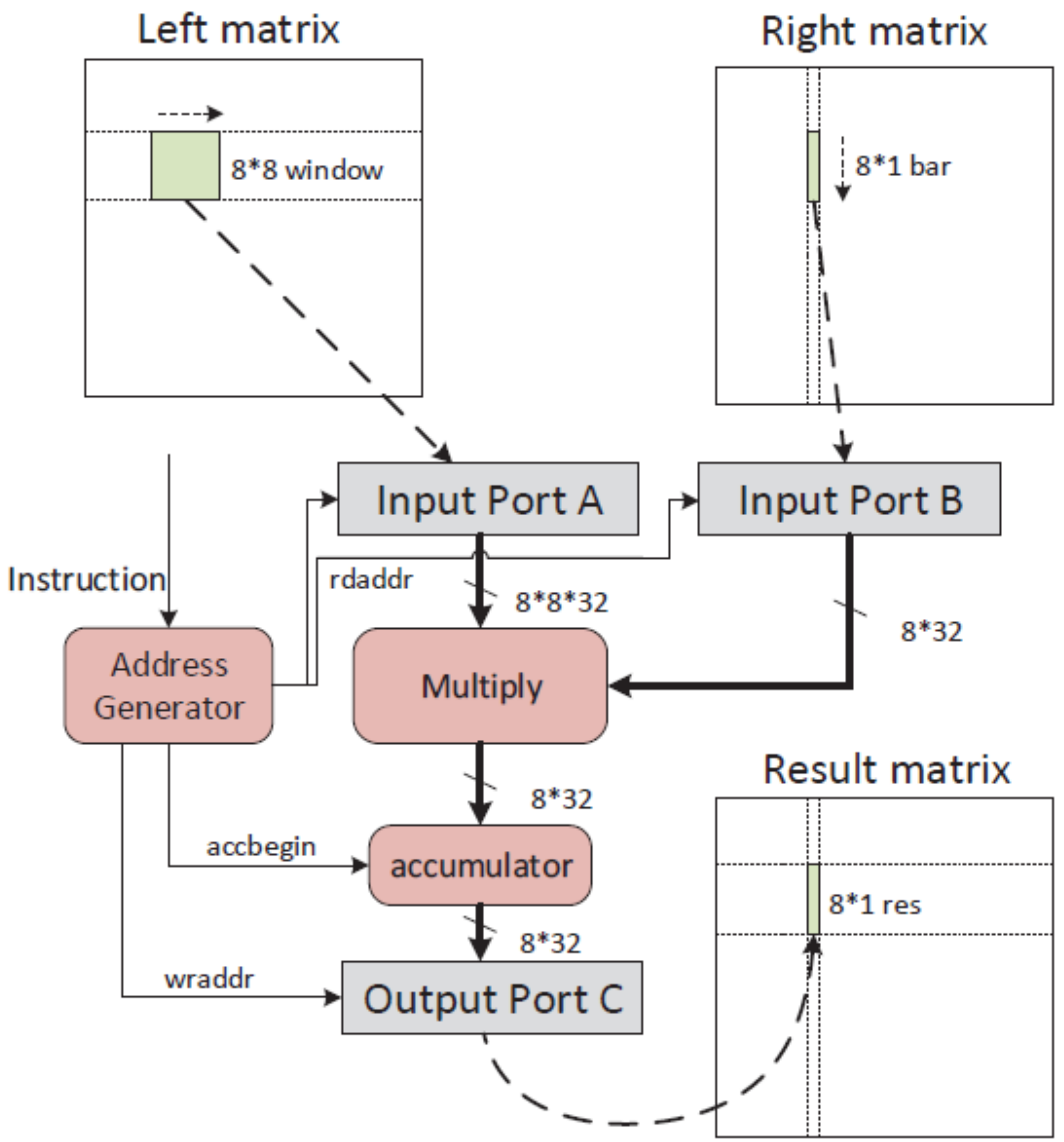
- Currently we have four kinds of accelerators:
  a) Matrix initializer
     8 entries per cycle

  b) Pixel-wise operation: +, -, .*, ./
     8 entries per cycle

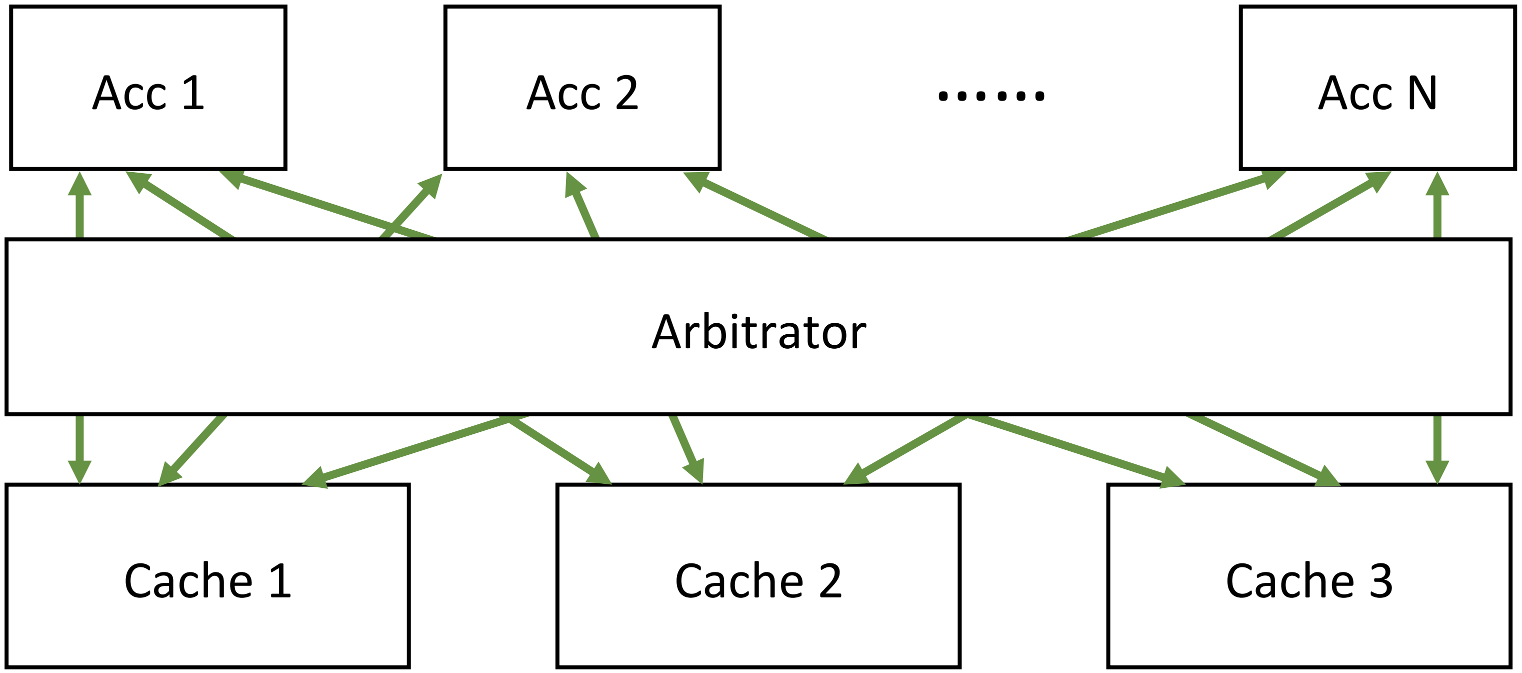
  c) Matrix multiplication
     8*8 window × 8*1 vector per cycle

  d) Maximum and minimum value of each column
     Read in 8 entries in a row per cycle, write back 8 results after N_{row} cycles

- Matrix multiplication:

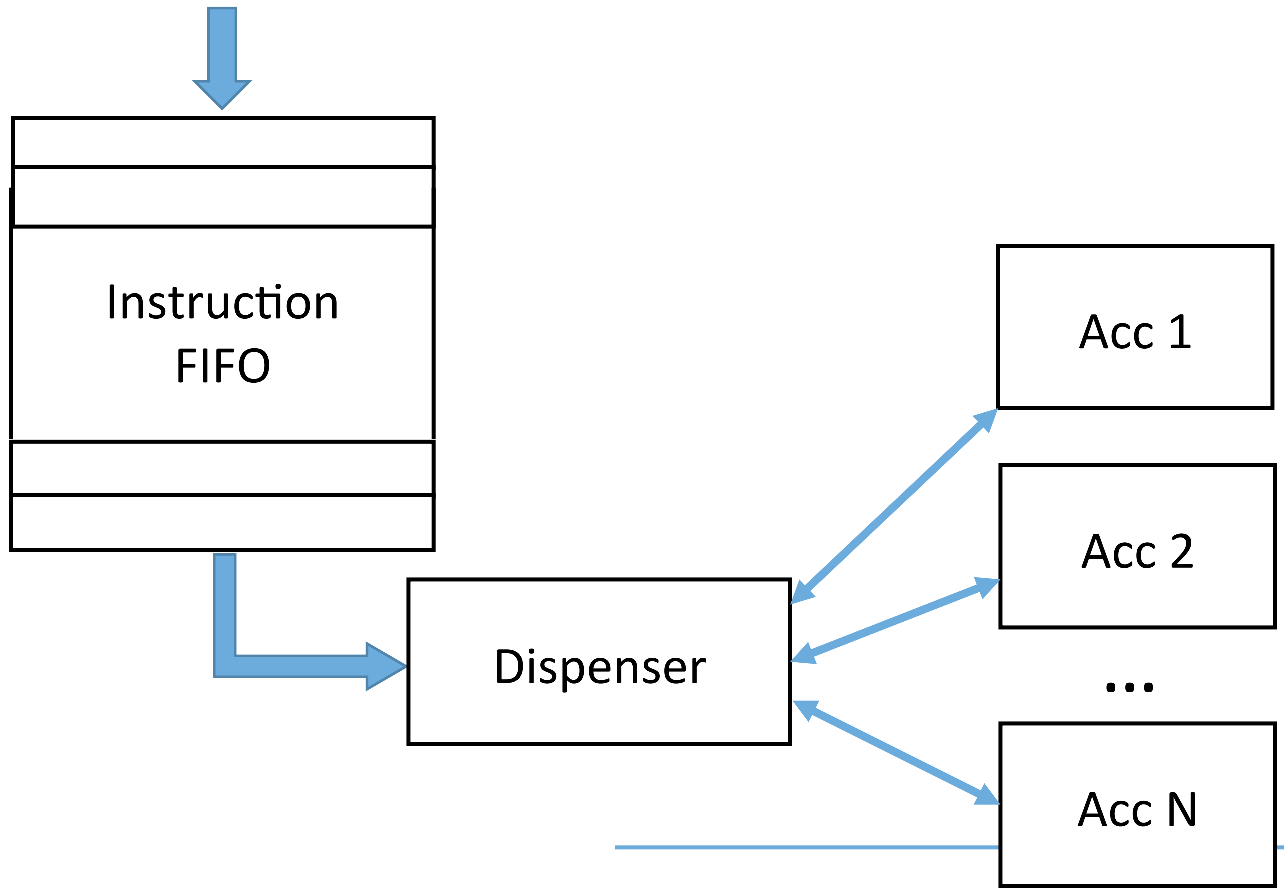


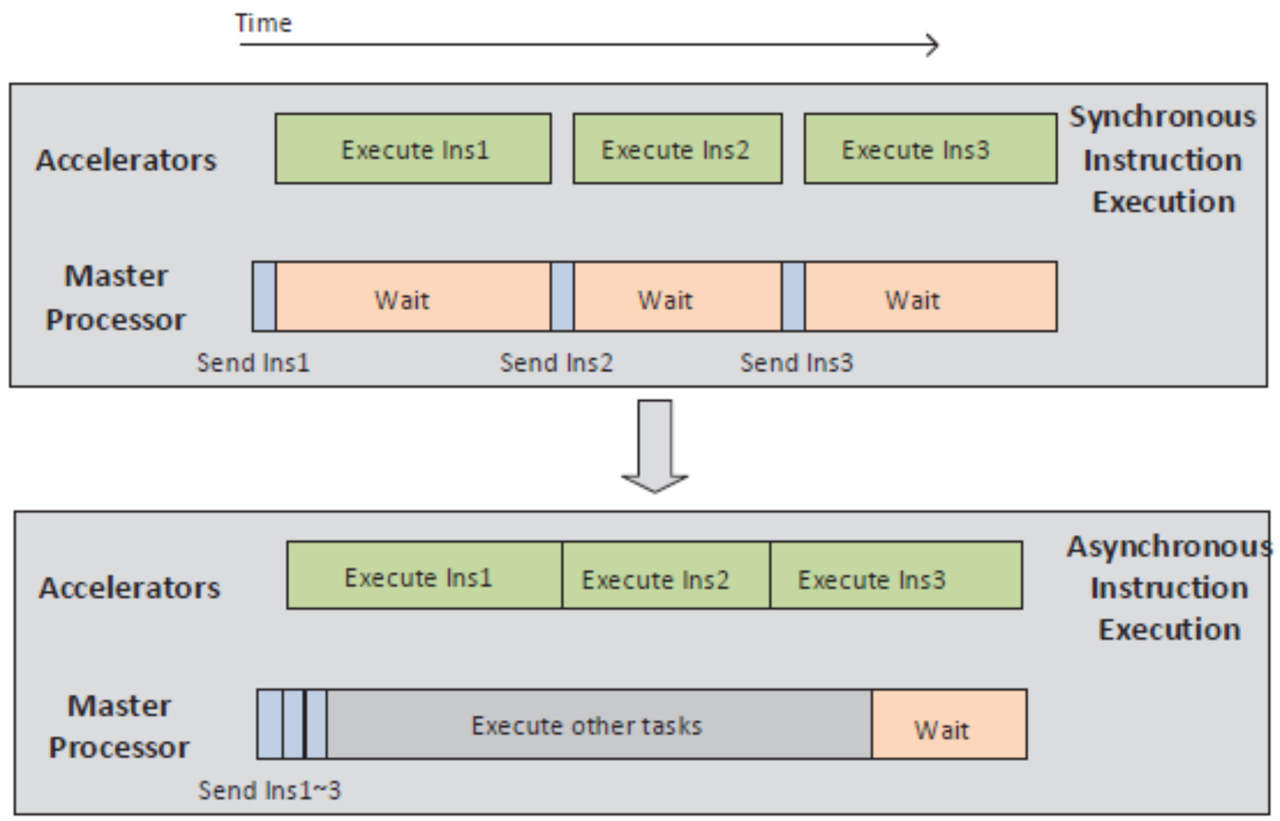
- Arbitrator connects the accelerators with the shared cache.
- Each accelerator can access any of the cache by any way(pixel, row, column or window).



Asynchronous Instruction Execution

- To reduce the workload of NIOS core, we assign the accelerator instructions by hardware.

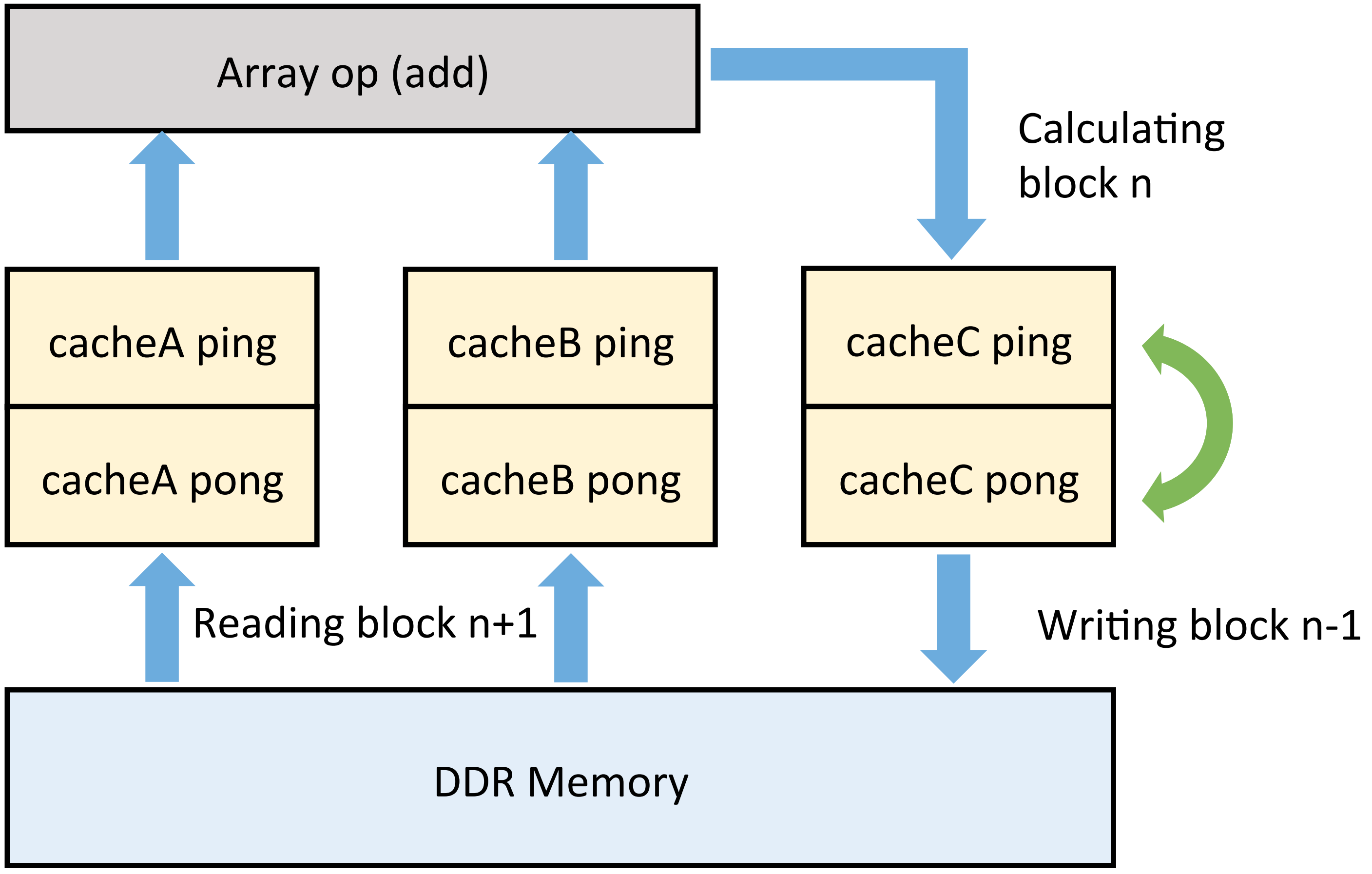




- The master processor spends a lot of time waiting in synchronous execution mode
- But time for waiting can be used for other tasks in asynchronous execution mode.

Ping-pong Strategy

- To deal with big matrix, we use a ping-pong strategy



Outline

Demo System

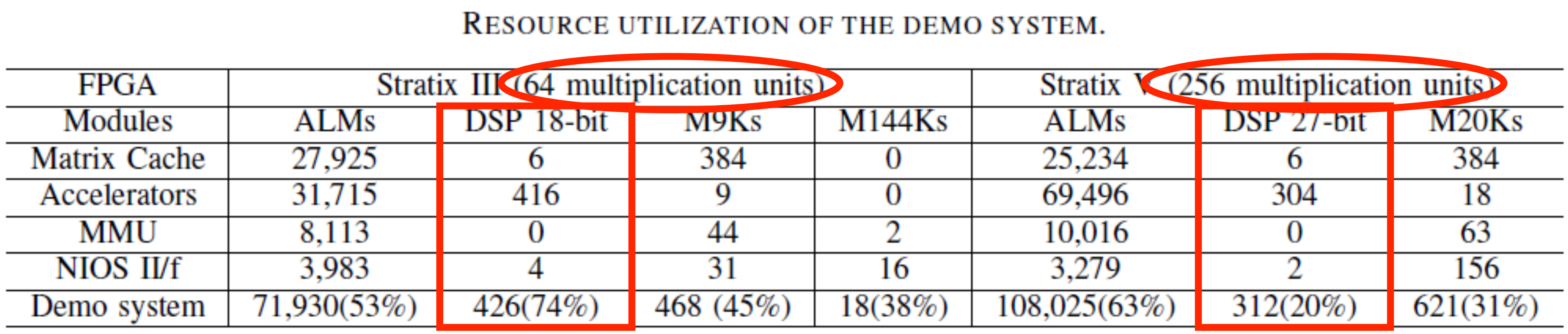
- Altera Stratix III on a DE3 develop board
- About 16Mbits block RAMs
- 1GB DDR2 memory, 533MHz

- System clock: 150MHz
- Cache size: 128*256*3
- Accelerators:

Accelerators:	Initializer	4
	Array operation	1
	Multiplication	1

- Software is C++ code built with *NIOS II Software Build Tools for Eclipse*

Resource Utilization

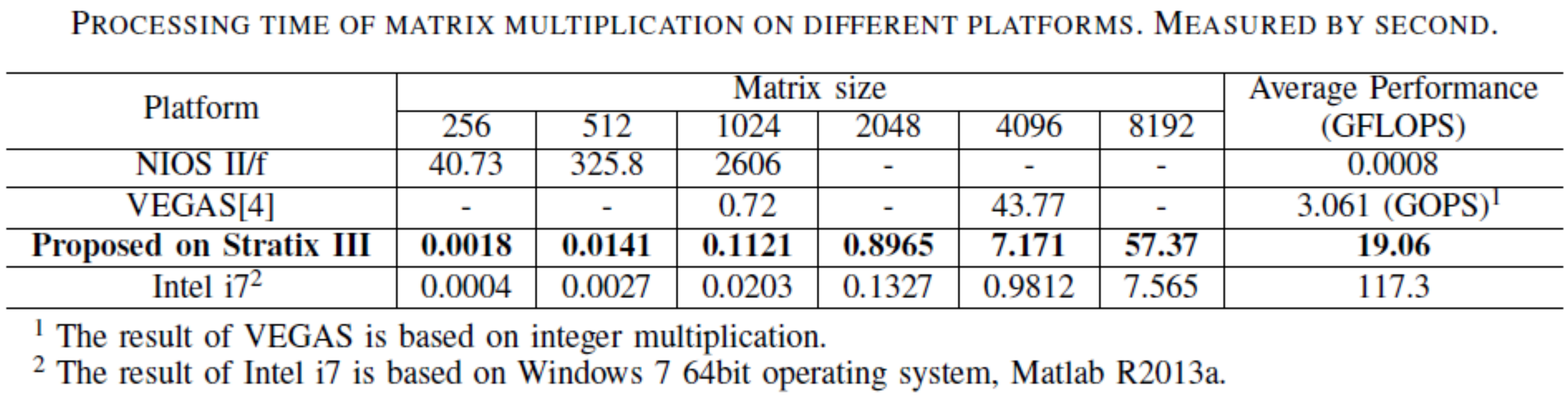


RESOURCE UTILIZATION OF THE DEMO SYSTEM.

FPGA	Stratix III (64 multiplication units)				Stratix V (256 multiplication units)		
Modules	ALMs	DSP 18-bit	M9Ks	M144Ks	ALMs	DSP 27-bit	M20Ks
Matrix Cache	27,925	6	384	0	25,234	6	384
Accelerators	31,715	416	9	0	69,496	304	18
MMU	8,113	0	44	2	10,016	0	63
NIOS II/f	3,983	4	31	16	3,279	2	156
Demo system	71,930(53%)	426(74%)	468 (45%)	18(38%)	108,025(63%)	312(20%)	621(31%)

- 4x multiplication units on Stratix V with more DSP units.
- **The bottleneck of resource is DSP units**

Performance Evaluation



PROCESSING TIME OF MATRIX MULTIPLICATION ON DIFFERENT PLATFORMS. MEASURED BY SECOND.

Platform	Matrix size						Average Performance (GFLOPS)
	256	512	1024	2048	4096	8192	
NIOS II/f	40.73	325.8	2606	-	-	-	0.0008
VEGAS[4]	-	-	0.72	-	43.77	-	3.061 (GOPS)[1]
Proposed on Stratix III	**0.0018**	**0.0141**	**0.1121**	**0.8965**	**7.171**	**57.37**	**19.06**
Intel i7[2]	0.0004	0.0027	0.0203	0.1327	0.9812	7.565	117.3

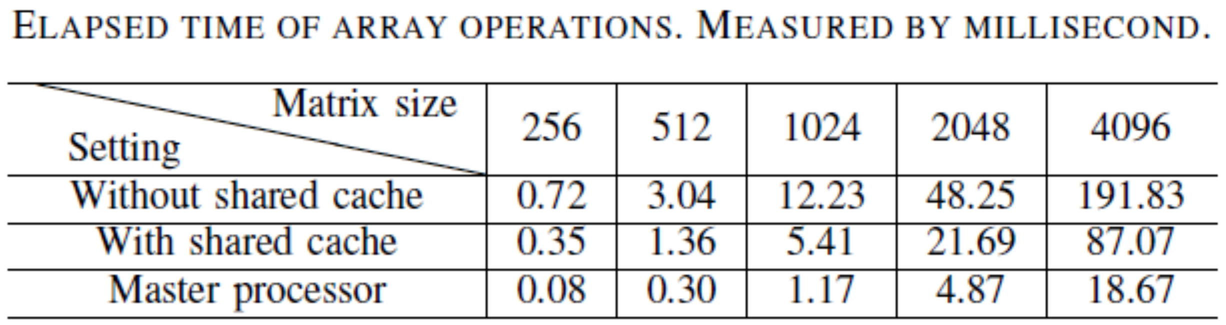
[1] The result of VEGAS is based on integer multiplication.
[2] The result of Intel i7 is based on Windows 7 64bit operating system, Matlab R2013a.

- Theoretical peak performance:

$$(64+64)*F_{clk}=19.2\text{GFLOPs}$$

- **Shows a better performance than VEGAS(vector processor)**

- We also evaluate the performance improvement brought by shared matrix cache and asynchronous instruction execution:

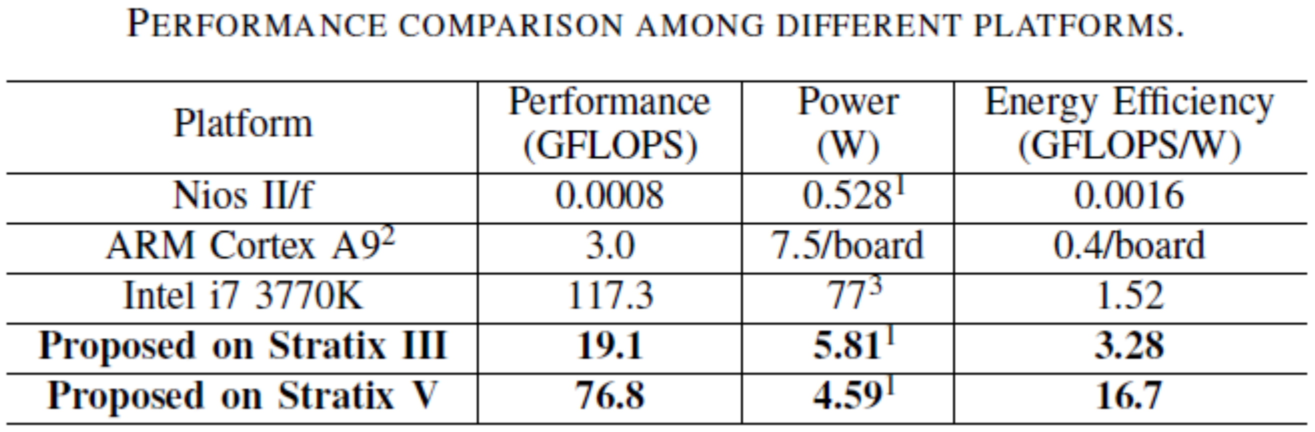

ELAPSED TIME OF ARRAY OPERATIONS. MEASURED BY MILLISECOND.

Setting \ Matrix size	256	512	1024	2048	4096
Without shared cache	0.72	3.04	12.23	48.25	191.83
With shared cache	0.35	1.36	5.41	21.69	87.07
Master processor	0.08	0.30	1.17	4.87	18.67

```
C=(A.*B)./(A+B)
```

- **Shared cache greatly improves the performance of the system.**
- **Workload is greatly reduced by asynchronous execution.**

Energy Efficiency



PERFORMANCE COMPARISON AMONG DIFFERENT PLATFORMS.

Platform	Performance (GFLOPS)	Power (W)	Energy Efficiency (GFLOPS/W)
Nios II/f	0.0008	0.528[1]	0.0016
ARM Cortex A9[2]	3.0	7.5/board	0.4/board
Intel i7 3770K	117.3	77[3]	1.52
Proposed on Stratix III	**19.1**	**5.81**[1]	**3.28**
Proposed on Stratix V	**76.8**	**4.59**[1]	**16.7**

- **FPGA platforms outperforms other platforms in energy efficiency**
- Improves the performance while reducing the power consumption at the same time comparing Stratix III and Stratix V

Outline

Conclusion

- Integrate different accelerators on the processor
  - With shared matrix cache
- We offer a software programming interface with matrix level instructions
- Minimize the workload of master processor
  - Asynchronous instruction execution

- TODO: Implement more accelerators

  Apply the system to mobile applications

Thanks
Q&A

NICS Nano-scale Integrated Circuit and System Lab，
Department of Electronic Engineering, Tsinghua University