ARAPrototyper: Enabling Rapid Prototyping and Evaluation for the Accelerator-Rich Architecture

Center for Future Architectures Research

CFAR

Yu-Ting Chen, Jason Cong, Zhenman Fang, and Peipei Zhou

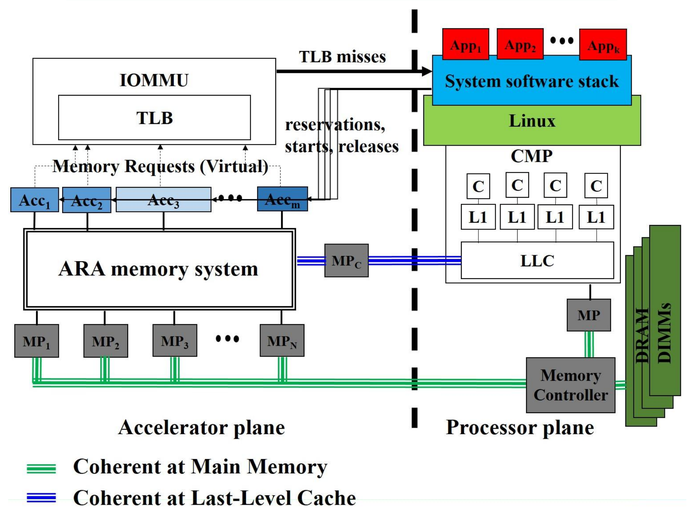
UCLA

Center for Domain Specific Computing

Motivation, Contributions, and Prototype

- **Motivation**
  - Accelerator-rich architectures (ARAs) can provide 10-100X energy efficiency over current chip multi-processors
  - Need methodology to efficiently and accurately evaluate an ARA
  - Limitations of full-system simulation:
    - Modeling: difficult to model the customized interconnects of an ARA
    - Speed: 300KIPS ~ 3MIPS; > 1000x slower than native execution
- **Contributions**
  - Show 4000 to 10,000 evaluation time saving over full-system simulation.
  - Provide a highly-automated prototyping flow
  - Provide an efficient evaluation framework and APIs for users
- **Prototype platform:**
  - Xilinx Zynq ZC706
    - SoC
    - Dual-core Cortex-A9 ARM
    - On-board BRAMs
    - FPGA (accelerators and interconnects
  - 1GB on-board DRAM
  - Linux can be ported

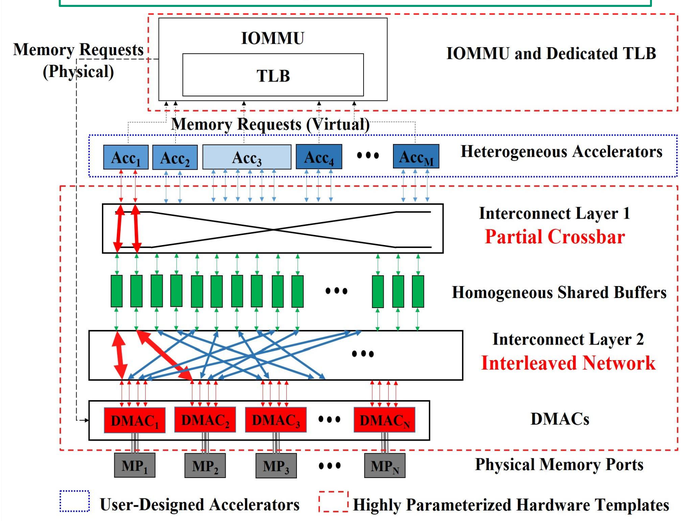
Baseline Accelerator-Rich Architecture (ARA) Prototype



Accelerator-Rich Architecture (ARA)

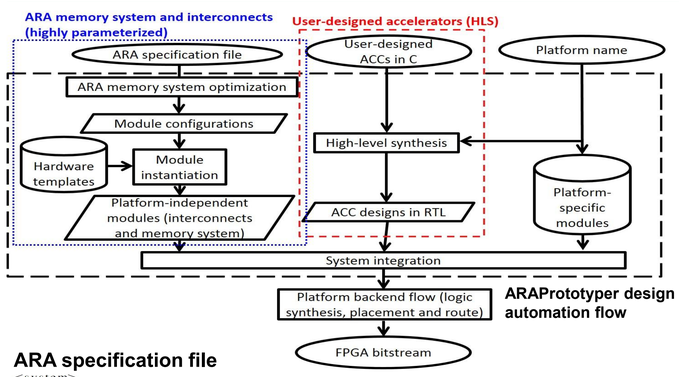
- **Accelerator plane**
  - Heterogeneous accelerators + ARA memory system
- **Processor plane**
  - Cores + a shared last-level cache

ARA Memory System Template



- **Shared homogeneous buffers**
- **Partial crossbar: accelerators ⇔ shared buffers**
  - Provide enough connectivity and guaranteed fixed latency
- **Interleaved network: shared buffers ⇔ DMACs**
  - Improve off-chip prefetching efficiency
- **IOMMU + IOTLB => improve page translation efficiency**

Hardware Design Automation



ARA specification file

```
<system>
<ACCs>
    <acc type="gradient" num="2" num_params="5">
        <port size="16384" num="6"/>
    </acc>
    <acc type="segmentation" num="1" num_params="13">
        <port size="16384" num="8"/>
    </acc>
    <acc type="rician" num="1" num_params="7">
        <port size="16384" num="12"/>
    </acc>
    <acc type="gaussian" num="1" num_params="7">
        <port size="16384" num="5"/>
    </acc>
</ACCs>
<SharedBuffers size="16384" num="32" numDMACs="4"/>
<Interconnects>
    <ACCs_to_Buffers type="crossbar" connectivity="3" auto="1"/>
    <Buffers_to_DMACs type="interleaved" use="1" auto="1"/>
</Interconnects>
<IOMMU>
    <TLB size="8192" evict="LRU"/>
</IOMMU/>
<CoherentCache use="0" />
<AccFrequency hz="75000000" />
</system>
```

Accelerator kernels

Shared buffer banks # and sizes

Interconnects:
(1) Partial crossbar configuration
(2) Interleaved network

TLB

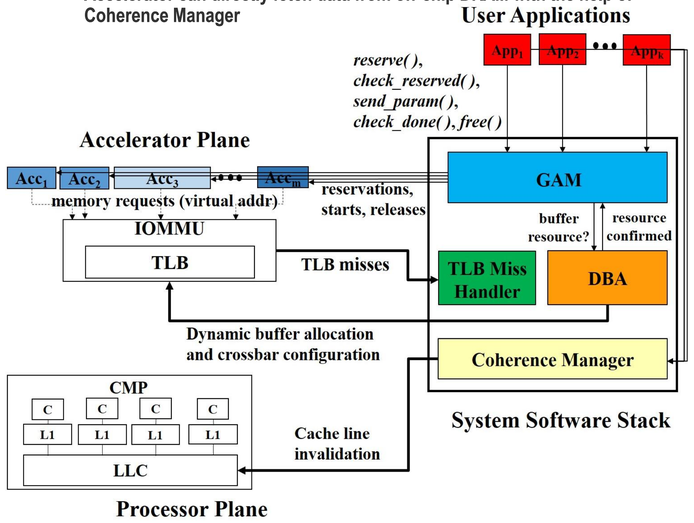
Coherent at memory or L2 cache

Frequency

Listing 1. An example ARA specification file created for design space exploration with four types of accelerators via ARACompiler.

System Software Stack

- **Major Components**
  - GAM – global accelerator manager
  - DBA – dynamic buffer allocator
  - TLB miss handler
  - Coherence Manager
    - Accelerator can directly fetch data from off-chip DRAM with the help of Coherence Manager



Program the Accelerators

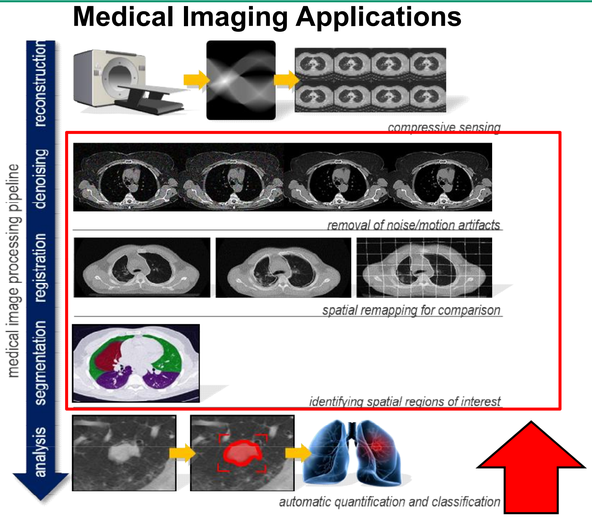
- **API supports for utilizing accelerators in an ARA**
  - C/C++ based APIs for manipulating accelerators
    - *reserve*(), *check_reserve*() (make reservations)
    - *send_param*() (send parameters and start the Acc.)
    - *check_done*() (poll the done signal)
    - *free*() (release the accelerator)
  - Easy compilation – only gcc is required
  - Executable can be run on board with Linux directly!

```
#include "accelerator_type.h"
void main()
{
  class Acc_gaussian acc;
  Image a;

  acc.reserve();
  while(acc.check_reserved() == 0);
  acc.send_param(7, Image::get_M(),
    Image::get_N(), Image::get_P(),
    a.get_ptr(), 1, 1, 1);
  while(acc.check_done() == 0) wait(1000);
  acc.free();
}
```

Listing 4. An example code of a user application

Case Study: Medical Imaging & MachSuite

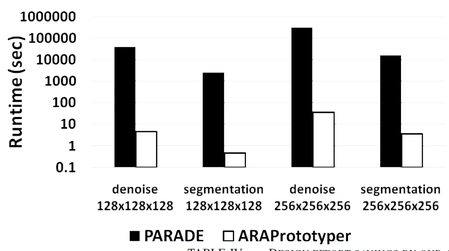


Acc_0: Gradient | Acc_2: Segmentation | Acc_1: Gaussian | Acc_3: Rician

Heterogeneous Accelerators

Code Size, Evaluation Time (Our Flow vs. Full-System Simulations)

- Show 2.9x to 42.6x evaluation time saving over the full-system simulations
  - More than 97% of flow time: C/C++ -> RTL -> FPGA bitstream
  - Native execution on the prototype: 10^3 ~ 10^4 faster than the full-sys simulations



- **Code size reduction**
  - ARAPrototyper and HLS tools reduce the design efforts

TABLE IV. DESIGN EFFORT SAVINGS BY OUR AUTOMATION FLOW IN GENERATION OF AN ARA WITH FIVE ACCELERATORS DESCRIBED IN LISTING 1.

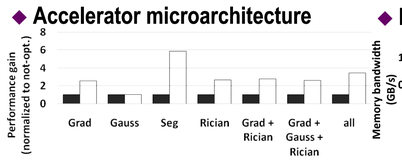
		# line of XML codes
input	ARA description file	33
	components	# line of RTL codes
automatically generated	IOMMU	21407
	crossbar	1526
	top module	14253

Table 3: Lines of code (LOCs) to integrate medical imaging and third-party MachSuite kernels into PARC/ARACompiler/ARAPrototyper including total generated RTL code, total HLS C/C++ code, kernel only HLS code, and integration only code.

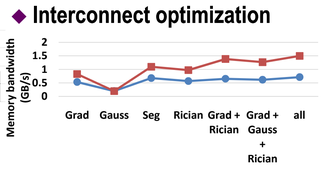
Domain	Accelerator	Total RTL	Total HLS	Kernel	PARC/ARACompiler Integration	ARAPrototyper Integration
Medical Imaging	gaussian	15107	513	363	150	5
	gradient	32538	778	616	162	6
	segmentation	63857	1304	1070	234	8
	rician	42291	1140	850	290	12
MachSuite [37] (third-party)	FFT/TRANSPOSE	17072	530	412	118	4
	GEMM/NCUBED	3201	121	23	98	3
	GEMM/BLOCKED	3226	158	20	138	5
	KMP/KMP	3593	167	45	122	4
	MD/KNN	7023	243	53	190	7
	SORT/MERGE	2996	128	54	74	2
	SPMV/CRS	4080	160	18	142	5
	VITERBI/VITERBI	4212	177	35	142	5

Results & Prototype

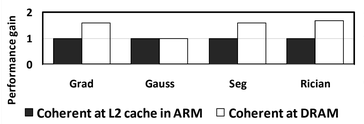
- **Accelerator microarchitecture**

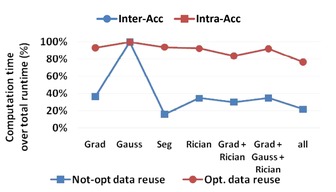


- **Interconnect optimization**



- **Coherence choices**





- **ARA FPGA prototype (100MHz) vs. state-of-the-art processors**

	Cortex-A9	Xeon (24 threads)	ARA
Freq.	667MHz	1.9GHz	Acc@100MHz CPU@667MHz
Runtime(s)	28.34	0.55	4.53
Power	1.1W	190W(TDP)	3.1W
Energy Eff.	1x	3.35x	7.44x

Energy-Efficiency Summary:

Prototype: 7.44x over Intel Xeon

ASIC projection: 84x over Intel Xeon (Haswell @1.9GHz), according to Kuon, TCAD'07