ARCHITECTURE SUPPORT FOR DOMAIN-SPECIFIC ACCELERATOR-RICH CMPS

Jason Cong, Mohammad Ali Ghodrat, Michael Gill,
Beayna Grigorian, and Glenn Reinman. 2014.

Presenter – David Young

Background:

- 2 types of On-chip accelerators
  - *Tightly coupled (functional unit attached to a core)*
  - *Loosely coupled (distant, attached to NoC, shared among multiple cores)*

- Sharing LCAs among all cores on-chip requires dispute resolution and scheduling

- Paper focuses on efficient use of Loosely Coupled Accelerators (LCA)

Overview:

- Related Work
- Hardware resource **management** scheme for…
  - *sharing of loosely coupled accelerators*
  - *arbitration of multiple requesting cores*
- Mechanism for accelerator **virtualization**
  - *compose larger virtual accelerator out of multiple smaller accelerators*
  - *collaborate multiple copies of a simple accelerator*
- Simulation methods
- Results from benchmarks representing 4 different application domains
  - *Demonstrate performance + energy improvements over software*
  - *Additional improvements from load balancing + simplified communication*

Current use of accelerators is limited

- ASIC designs lack reusability across different applications
- On-chip accelerators are mostly limited to specific applications
  - *Via specific functionality (be it simple or complex)*
- Factors limited the use of accelerators
  - *Overhead involved in their use (mostly from interacting with OS)*
  - *No efficient management for sharing LCAs*
    - among different cores
    - across different applications

Related work does not focus on On-Chip, Hardware Based management of LCAs

- Implementing application specific accelerators with ASIC or FPGA
  - *Mostly consider single accelerator dedicated to single application*
- Reconfigurable computing
  - *Mostly deal with customized accelerators which are OFF-CHIP*
- On-Chip Integration of accelerators
  - *Generally deal with tightly coupled accelerators*
- Accelerator management
  - *Mostly OS support/software implementations*

New architecture framework could increase use of Accelerators

- Propose ARC – accelerator rich CMP architecture framework
  - *Resource management scheme with low overhead*
  - *Minimally invasive to core designs*
  - *Easy for application authors to take advantage of*
- Paper provides
  - *Accelerator allocation protocol*
  - *Approach to accelerator virtualization*
  - *Fully automated simulation tool-chain*

ARC Idea 1: Reduce overhead by moving mechanisms from OS driver to hardware.

- Typical system using accelerators
  - *Arbitration and communication mechanisms provided by an OS driver*
  - *To access an accelerator, a core uses an accelerator driver (OS call).*
    - Overhead! Significant if accelerators are numerous, and used frequently.
- Moving mechanisms from OS driver to hardware
  - *Reduce cost of interacting with accelerators*
  - *Allow for efficient sharing of accelerator resources*

ARC Idea 2: Breaking accelerators into smaller pieces will increase utilization.

- Different functional accelerators may share internal sub components
- Break down accelerators into smaller pieces and permit communication
- Share common elements
  - *Increase utilization*
  - *Increase Range of accelerator functionality in given chip area*
- Requires more complex management mechanism

Two different micro-architectures of ARC intended to address arbitration of accelerator resources

- GAM (Global Accelerator Manager)
  - *First implementation*
- GAM+
  - *Improved form*
  - *Address shortcomings of first implementation*

ARC with GAM

- Components
  - *Cores, accelerators, GAM, shared L2 caches, shared NoC routers*
  - *NoC routers everywhere!*
- Expanded ISA
  - *4 new instructions used for interacting with accelerators*
    - Request accelerator availability (lcacc-req)
    - Request use of specific accelerator (lcacc-rsv)
    - Interact directly with accelerator (lcacc-cmd)
    - Release accelerator (lcacc-free)
  - *No OS interaction required*
  - *Execution of instructions results in message being send to device on NoC (GAM or accelerator)*

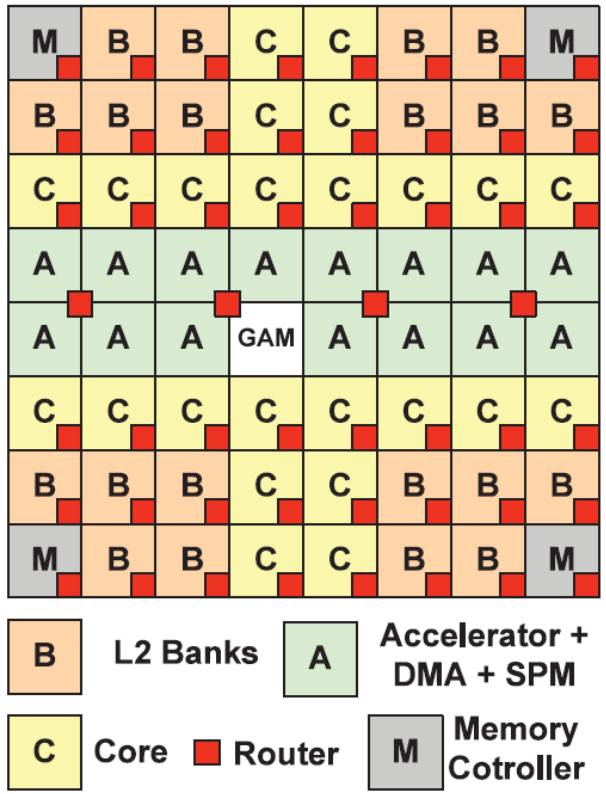


Fig. 1. Overall architecture of ARC.

GAM: Cores communicate with accelerators directly.

- Steps taken when a core uses a single accelerator
  1. *Core sends request to GAM with cores it may need, GAM responds list of accelerator IDs & estimated wait times*
  2. *Core sends sequence of reservations for specific accelerators to GAM and waits for permission to use while GAM configures requested accelerators.*
  3. *Core write task description into shared memory and sends command msg to accelerator identifying the memory address of the task description. Accelerator loads and begins task.*
  4. *Accelerator finishes and notifies core. Core sends free msg to GAM to free accelerator.*
- Significant Shortcomings
  - *Cannot divide work among multiple accelerators*
  - *Potentially long wait times when thread reserving accelerator*

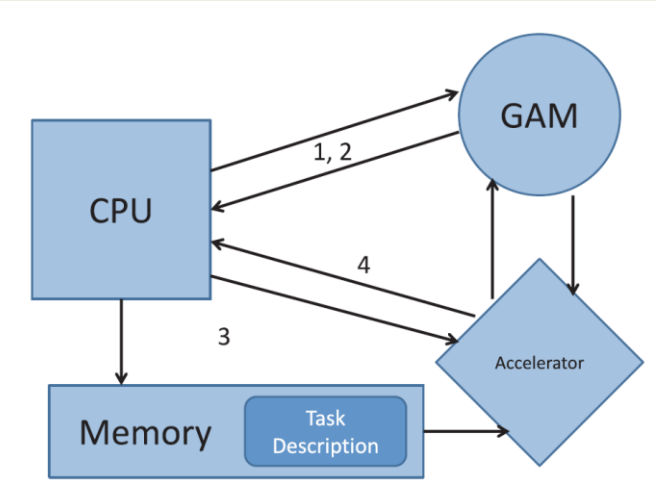


Fig. 2. Communication between core, GAM, and accelerator.

Revisions for GAM+ address shortcomings of GAM

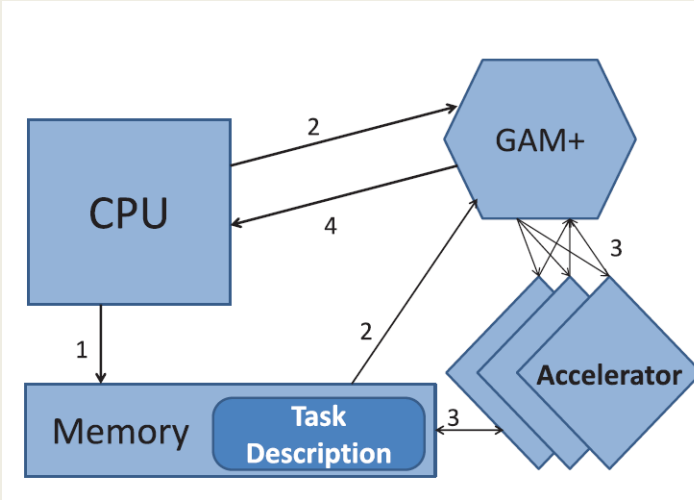
- Revised method by which threads communicate work to accelerators
- Created task groups without dependencies (can be scheduled in parallel)
- GAM becomes interface when communicating with accelerators
  - *Schedules task groups to accelerators*
  - *Cores don't have to interact directly with accelerators*

GAM+ revisions simplify communication

- Revisions
  - *Simple scheduling algorithm*
  - *Small local TLB*
  - *Small table that maps “requesting threads” to “accelerator progress info”*
  - *Threads send task description to GAM+ which forwards it to accelerators as resources become available.*
  - *No longer need request, reserve or free instructions. Only 1 instruction (command) is required.*
- GAM+ vs GAM
  - *Simplified communication*
  - *Simplified ISA*
  - *Reduction in overhead of setting up accelerator*
  - *Increase accelerator utilization, by allowing single requesting thread to utilize all accelerators of given type in the system.*

GAM+, not core, communicates with accelerators directly

- Steps taken when a core uses a single accelerator (GAM+)
  1. *Core writes (to the shared memory) the task description to be performed.*
  2. *Core sends mem. address of task description to GAM+, which reads header to find out how many task groups and what kind of accelerators needed.*
  3. *As accelerators become available, GAM+ forwards task description to each accelerator, specifying which task group is assigned to that accelerator. Accelerators notify GAM+ when finished.*
  4. *When all tasks completed, GAM+ notifies CPU that computation is done.*
- GAM+ interleaves task groups of multiple requesting threads
  - *Round-robin scheduling policy*
  - *Priority based scheduling possible*
  - *Helps eliminate wait times*



Lightweight Interrupt Support

- CPU needs to be notified of accelerator progress
- ARC lightweight interrupts do not involve OS interaction
- Sources of Interrupts
  - *3 for GAM (Gam responses, TLB misses, accelerator finished)*
  - *2 for GAM+ (GAM notifying core that work is done, TLB misses)*
- Cores in ARC system modified to support lightweight interrupts
  - *Interrupts sent to core via interrupt packet through NoC*
    - Packet includes recipient thread ID, and interrupt specific info
  - *Interrupt controller in core's network interface receives packets*
    - Uses queue to buffer packets
  - *Lightweight interrupt interface in the core*
    - Receives interrupt from interrupt controller
    - Software interface to help service interrupt

Invoking Accelerators still costly, so amortize overhead over large amount of work

- Task descriptions should be extremely detailed
  - *Arg locations, data layout, accelerators involved, computations, operation order*
  - *Removes need for communication between accelerator and core*
  - *Allows accelerators to be general and coordinate in groups*
- Accelerator evaluates the detailed description
  - *Series of steps performed in order*
    - Each has mem transfers and computations that can be executed concurrently
  - *Naming convention*
    - Task : each step
    - Task group: collection of tasks performing same computation on different data

Sharing accelerators with GAM requires time estimates

- Accomplished jointly between software and the GAM
  - *GAM provides thread with pre-allocation info and arbitrates among threads*
  - *Software thread responsible for making use of accelerator post allocation*
- Thread can opt to run task itself if estimated wait time is too high
  - *GAM will predict these situations for the thread, but needs an estimated duration of tasks to be performed*
  - *Time estimates are data dependent, so requesting core submits estimate of time accelerator would be used*
    - Estimates based on pre-constructed profiles of different execution on given accelerators + polynomial fitting

Sharing accelerators with GAM+ is less complicated

- Simplified arbitration mechanism (hardware scheduler)
  - *Threads simply send a task (core not responsible for acquiring, scheduling or freeing accelerators)*
  - *Eliminates some software time*
  - *No need for wait-time estimation*
  - *Under very staged circumstances GAM+ slower than GAM.*
  - *In most cases GAM+ has a net gain*

Accelerator virtualization with two techniques

- Composing different types to create new types, or same types to create larger accelerator
- ***Accelerator chaining*** *(Output of accelerator feeds input of another)*
  - *Traditionally accelerators communicate through system memory*
    - Thread uses shared memory to read from acc#1's SPM and write to acc#2's SPM.
  - *Instead, use two DMA-controllers*
    - Source DMA-cntrl sends content of src SPM to dst DMA-cntrl who writes to dst SPM

- **Accelerator composition** (implemented as a series of calls to other physical accelerators)
  - *Library of virtual accelerators provided to app author as if they existed in hardware*
  - *Just decomposition rules breaking down large problem into smaller sub-problems*
    - Rules applied recursively to express virtual accelerators as calls to physical accelerators
  - *In GAM, software selects best decomposition strategy*
  - *In GAM+, core allows GAM+ to schedule the strategy*

Accelerator Extraction Methodology:

- Software module called “virtualizer” outputs a DLL
- Given an application, the module uses static analysis and profiling to…
  - *Create list of candidates for accelerators to be extracted*
  - *Weigh candidates by selection criteria to create database of accelerators*
    - Criteria: area, performance, energy, criticality etc.
  - *Create new accelerators from available accelerators in the platform hardware using the database and transformation rules.*
- Restrictions:
  - *Systems with GAM/GAM+ restrict the type of LCAs that can be included*
    - GAM requires LCAs to have finite execution time that can be estimated
    - LCAs cannot be permanently associated with certain cores

Evaluation Introduction:

- Varied benchmarks
  - *Different levels of parallel friendly, computation bound, communication bound.*
  - *Represent 4 application domains (MedImg, Com, CompVis, Nav)*
- Automated Simulation Tool-Chain
  - *Systems too complex to author by hand (lots of accelerators + communication)*
  - *Complex tools used to create cycle-accurate simulator modules for accelerators*
  - *GAM/+ modeled as state machine and then automatic tools synthesize timing info*
- Simulator infrastructure
  - *Modified version of GEMS cycle accurate sim (runs on top of Simics)*
  - *Accelerators, GAM/+ and CPU all attached to network interface stubs*
    - Allows flexible communication among components via virtual channel in NoC
- Specific Simulation
  - *Multi-core based on mix of Ultra-SPARC-III-i processors and accelerators*
  - *Maintain fixed cache and network configs, per processor L1 cache + distributed L2*

Results focus on GAM+:

- Cong et al. [2012] showed
  - *ARC reducing OS overhead for LCA arbitration*
  - *GAM estimation accuracy*
  - *Benefits of using lightweight interrupt protocol*
- Cong et al. [2014] focuses on other aspects
  - *Performance improvements*
  - *Energy savings*
  - *OS overhead reduction for new domains*
  - *Benefit of GAM+ over GAM*
- Cong et al. [2014] included benchmark results
  - *Baseline… native (non-simulated) execution of benchmarks on 2GHz intel xeon E5405 core*
  - *Accelerators + HW (GAM)*
  - *Accelerators + HW (GAM+)*

GAM+ demonstrates speedup + energy efficiency gains over native software implementation

- Increase going from 1->4 LCAs shows benefit of dynamic load balancing
- Energy efficiency trends closely with performance
  - *accelerators have minimal impact on total power consumption, which is mostly attributed to NoCs and caches*
- Some results don't scale as well w/ # of accelerators
  - *SURF, Texture Synthesis, Stream Clusters*
  - *feature large # of small accelerated regions resulting in uneven distribution and poor scaling*
- Takeaway:
  - *Benefit of adding accelerators correlates heavily with data parallelism inherent in given application*

Fig. 10. All domains - performance improvements of GAM+ over SW-only versions.

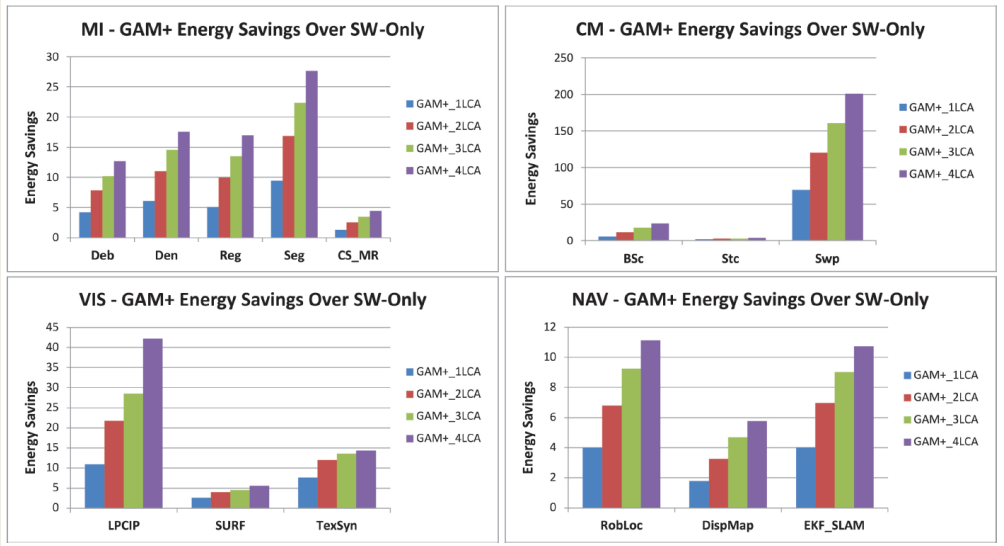


Fig. 11. All domains - energy improvements of GAM+ over SW-only versions.

Overall, GAM+ improves on GAM

- GAM restricted to single accelerator while GAM+ can use arbitrary number
- Again energy benefit trends with performance
- In most cases, GAM & GAM+ perform similarly provided 1 LCA
- The overhead of initializing an accelerator with GAM is greater than with GAM+
  - *Generally this is insignificant*
  - *But if accelerator use time >> prep time, it becomes significant.*

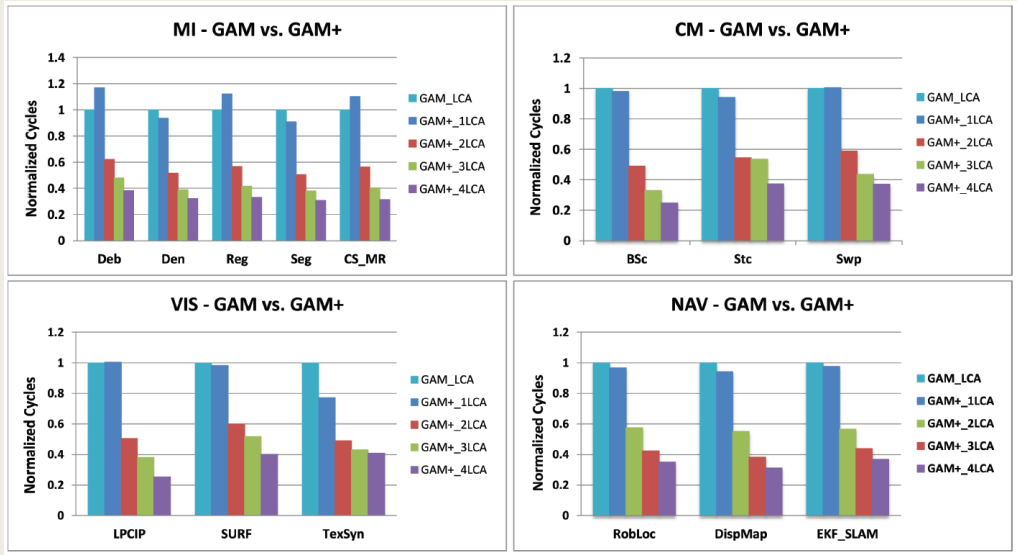


Fig. 12. All domains - performance of GAM vs. GAM+ (normalized to GAM).

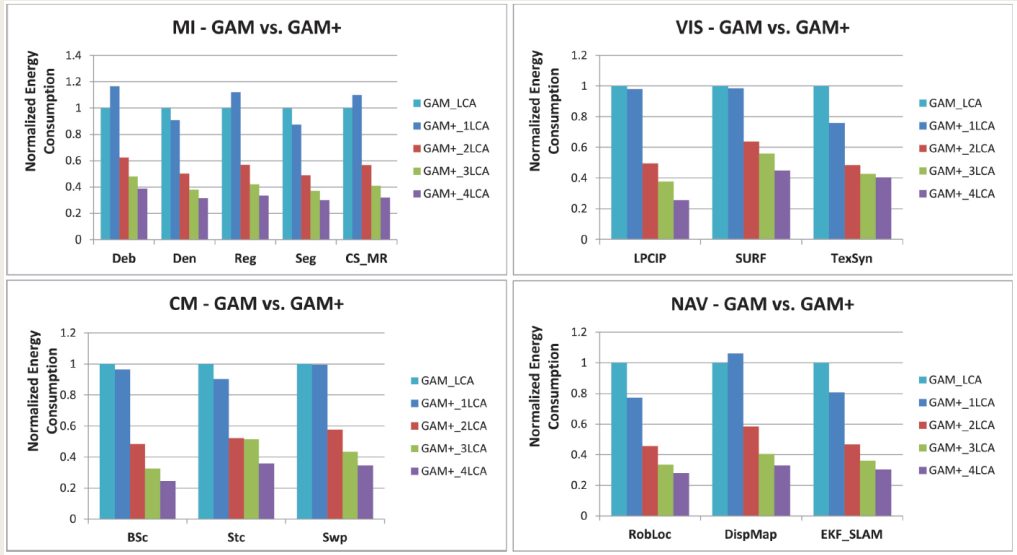


Fig. 13. All domains - energy consumption of GAM vs. GAM+ (normalized to GAM).

GAM+ has situational benefits and drawbacks

- Benefits of GAM+
  - *Texture Synthesis Benchmark*
    - Accelerator is used repeatedly for small jobs
    - For GAM, ~20% of total execution time is initialization.
    - Compare 1LCA GAM to GAM+
- Drawbacks of GAM+
  - *Medical imaging benchmarks using 1LCA*
  - *GAM+ won't assign more work to the accelerator until previous work is done.*
  - *If the assigned task group is small relative to total amount of tasks, there will be an overhead with GAM+*
- Balancing act
  - *smaller task groups increase the overhead associated with allocation*
  - *larger task groups make it hard for GAM+ to distribute work evenly amongst many accelerators*

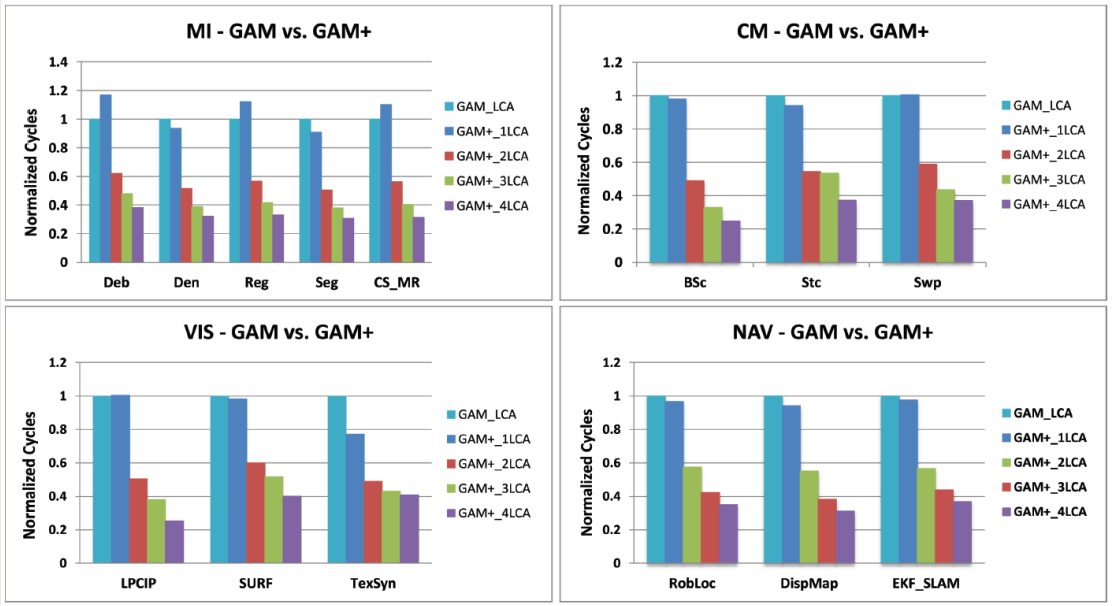


Fig. 12. All domains - performance of GAM vs. GAM+ (normalized to GAM).

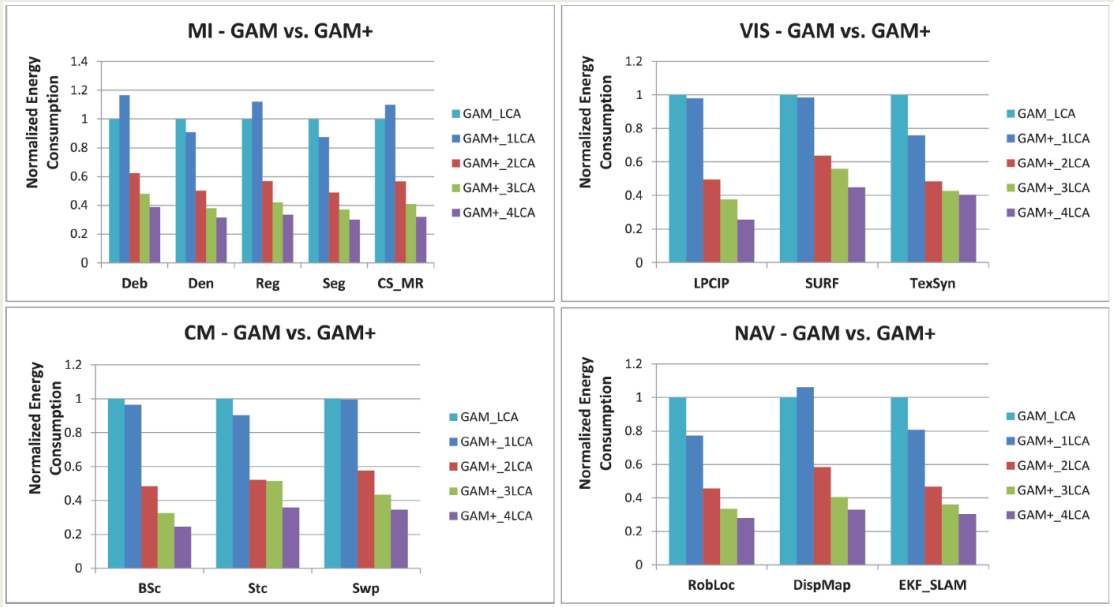


Fig. 13. All domains - energy consumption of GAM vs. GAM+ (normalized to GAM).

Virtualized Accelerators can match performance of Physical Accelerators

- Virtualized a 2D and a 3D FFT computation using a 1D FFT engine
- Graphs compare virtualized approach with the physical/monolithic implementation of 2D and 3D FFT engines
- results show the ratio of computation time
  - *(virtual LCA computation time)/(physical LCA computation time)*
- Can match performance of physical 2D FFT using 4x 1D FFT
- Can match performance of physical 3D FFT using 5x 1D FFT
- No data on energy efficiency.

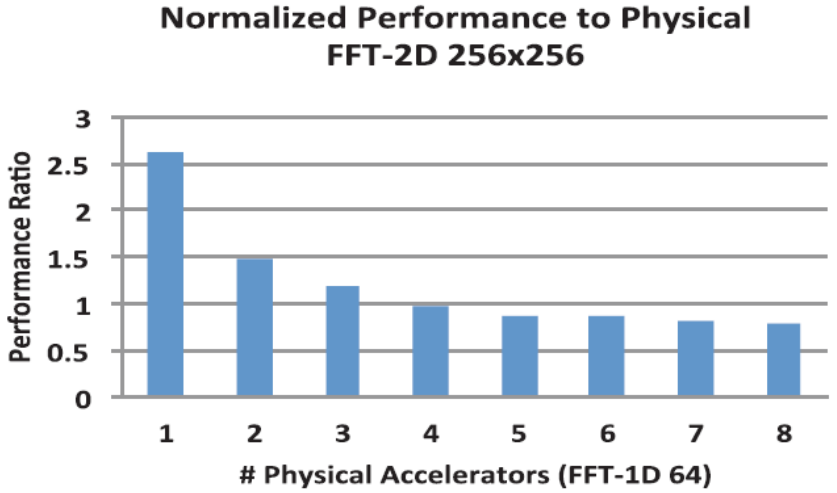


Fig. 18. FFT-2D virtualization.

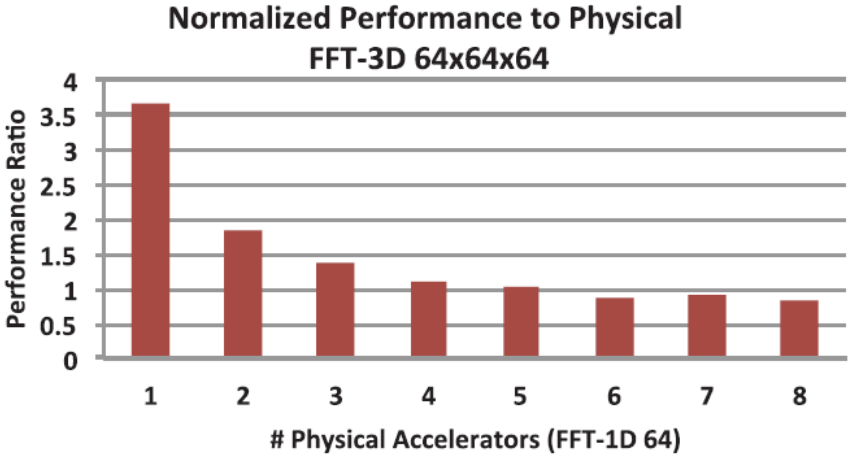


Fig. 19. FFT-3D virtualization.

Advocating use of efficient hardware-based techniques for managing and interfacing with LCAs.

- Presented hardware resource management scheme for sharing LCA and arbitrating multiple requesting cores
- Presented mechanism of virtualizing accelerators
  - *Compose larger accelerator from smaller accelerators*
  - *Collaborating with multiple copies of single accelerators*
- Results show significant improvements over software implementation
- Results showed additional benefits using GAM+
  - *Enhanced load balancing*
  - *Simplified communication*