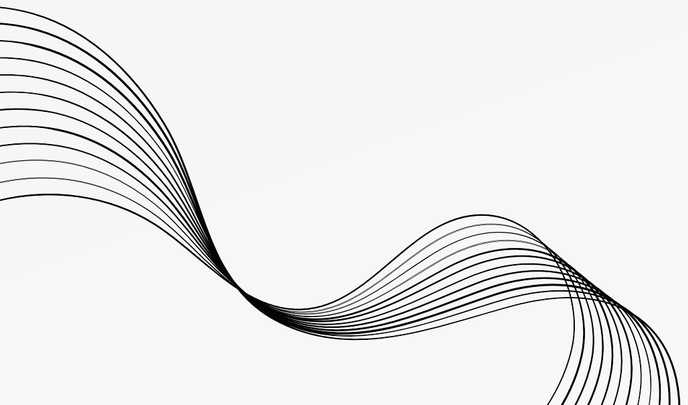
AuRORA: Virtualized Accelerator Orchestration for Multi-Tenant Workloads

MOTIVATION AND BACKGROUND

- The cumbersome physical accelerator integration does not scale to many-accelerator systems running multi-tenant workloads, especially when resources need to be frequently reallocated to meet the distinct demands of applications during execution.
- The existing methods are classified into two main categories:
- physical integration, where workloads are explicitly mapped onto physical accelerators, and
- virtual integration, where programmers interact solely with virtualized accelerators, with the workload-to-accelerator binding managed by a separate integration layer.
- In particular, dynamic accelerator orchestration through preemptive allocation is required for multi-tenant execution where multiple tasks share the system resource with different target requirements.



AURORA: VIRTUALIZED ACCELERATOR

- Low-overhead shim microarchitecture to interface between cores and accelerators,
- A hardware messaging protocol between CPU and accelerators to enable scalable and virtualized accelerator deployment on SoCs,
- ISA extensions to allow user threads to interact with the AuRORA hardware in a programmable fashion, and
- A lightweight software runtime to dynamically reallocate available resources for multi-tenant workloads.

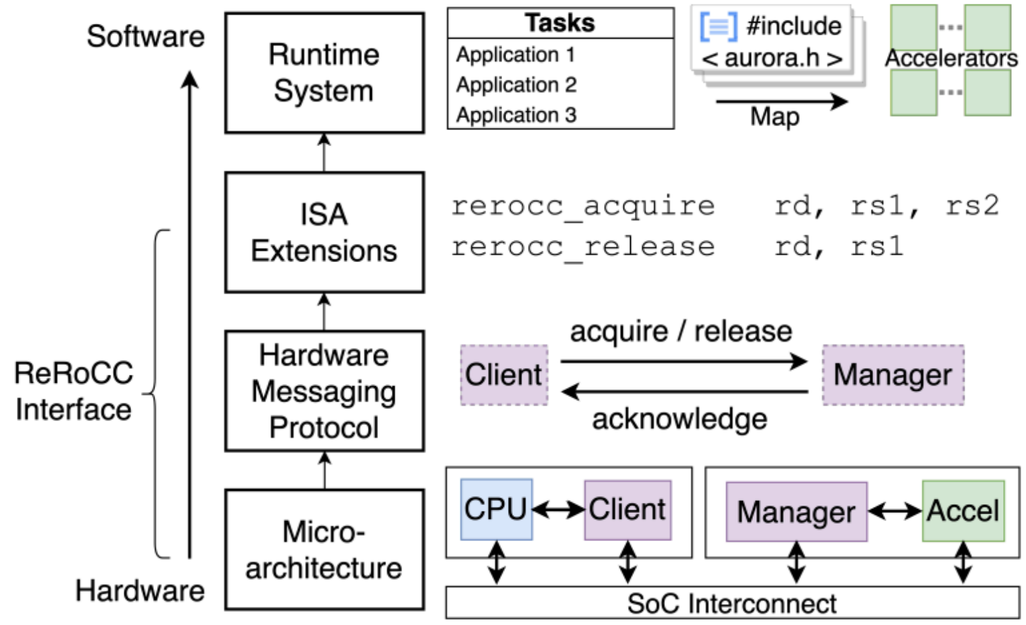
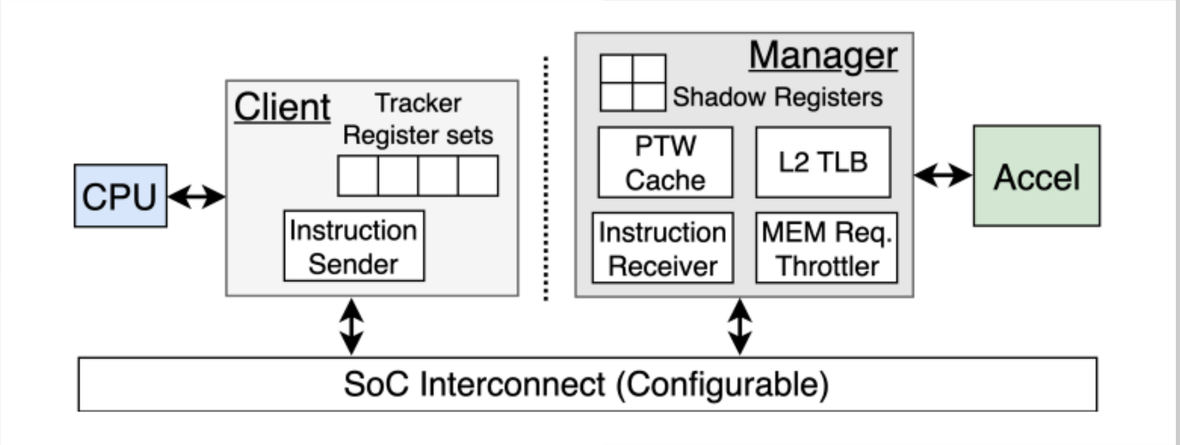


Figure 1: AuRORA is a full-stack accelerator integration methodology for scalable accelerator deployment.

CLIENT

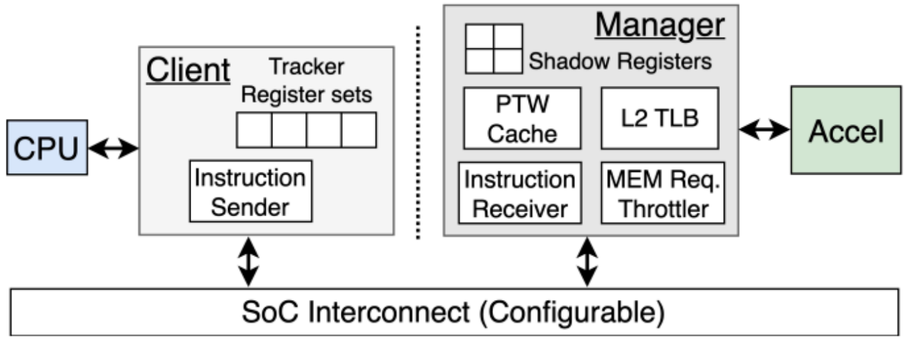
- **Integration with Host Cores:**
- The Client shim integrates with general-purpose cores (like Rocket and BOOM cores) to enable communication with disaggregated accelerators.
- It provides an architectural illusion that makes the accelerators appear tightly coupled with the host cores, despite being physically disaggregated across the SoC (System-on-Chip) interconnect.
- **Reservation Tracking:**
- Each core managed by the Client keeps track of which accelerators it has currently reserved.
- This tracking is facilitated through a hardware table within the Client, which maintains the state of accelerator reservations.
- **RoCC Accelerator Implementation:**
- Implemented as a RoCC (Rocket Custom Coprocessor) accelerator, the Client can seamlessly integrate with existing RoCC-compatible cores.
- This compatibility allows AuRORA to leverage the flexibility and extensibility of the RoCC interface for accelerator management and control



AURORA MICROARCHITECTURE

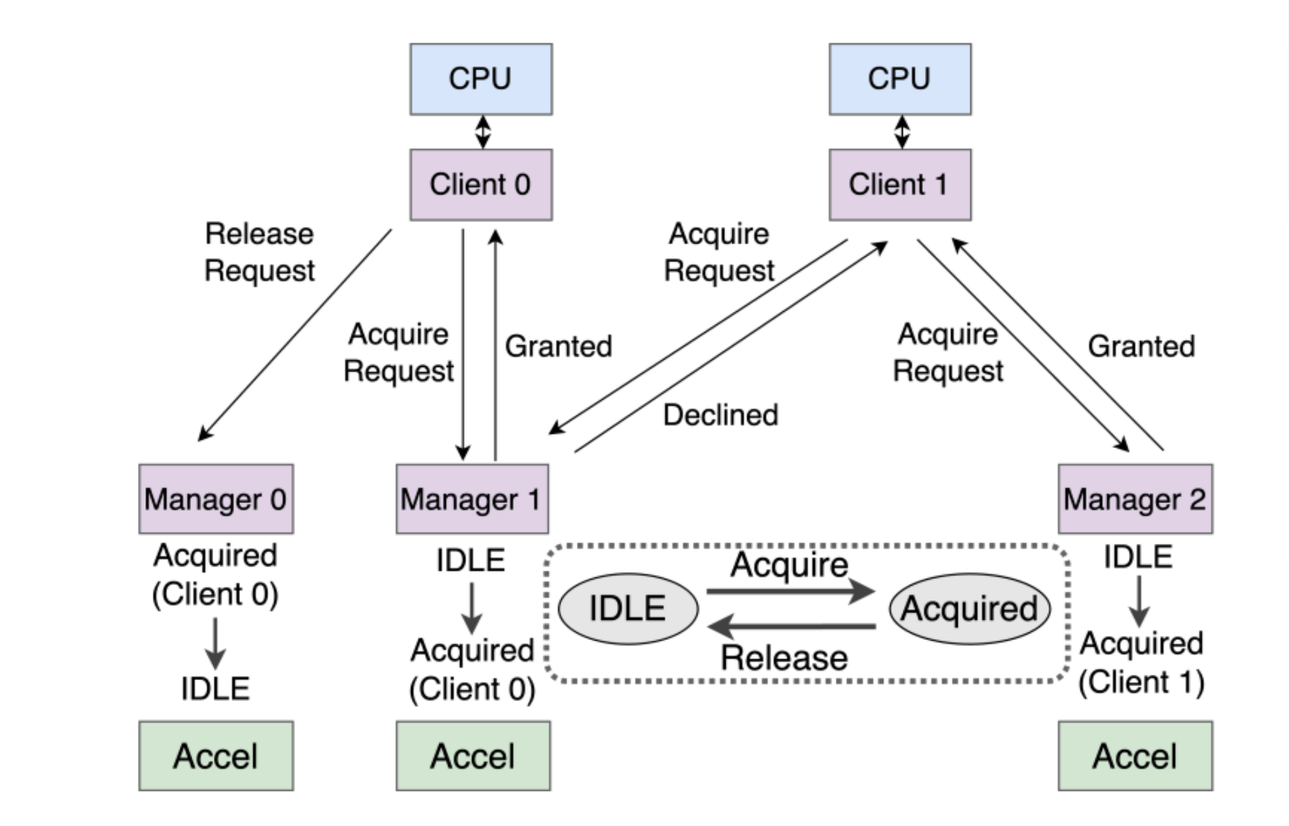
MANAGER

- **Wrapper for Accelerators:**
- The Manager shim wraps around individual accelerators, and acts as an intermediary between the Client and the accelerators, receiving commands from the Client and forwarding them to the appropriate accelerators.
- **Management of Architectural CSRs:**
- The Manager maintains a shadow copy of critical architectural Control and Status Registers (CSRs) used by the accelerator MMU (Memory Management Unit) which hold information related to thread privilege levels, memory translation modes, and page table addresses, ensuring that the accelerators operate with correct memory access permissions and virtual addressing.
- **Memory Management Unit (MMU):**
- Implements an architecturally compliant MMU for the accelerators, including components such as a page table walker (PTW), optional PTW cache, and an L2 TLB (Translation Lookaside Buffer).
- **Configurable Traffic Throttlers:**
- These throttlers enable setting bandwidth limits on memory traffic generated by accelerators, ensuring fair resource allocation and preventing performance degradation due to excessive memory bandwidth usage.



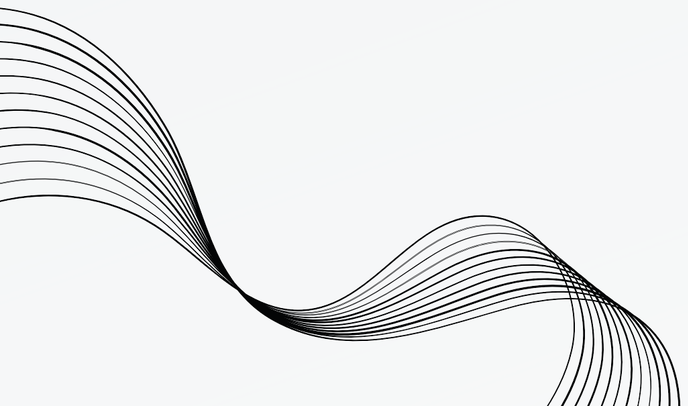
AURORA MICROARCHITECTURE

AURORA HARDWARE PROTOCOL



AURORA ISA EXTENSIONS

AuRORA Pseudoinst.	Operands	Purpose
`rerocc_acquire`	`success, acc_id, acq_id`	Acquires accelerator and maps it to the local client, returns success status.
`rerocc_release`	`acq_id`	Releases an accelerator currently acquired by the local client.
`rerocc_assign`	`acq_id, opcode`	Maps a currently acquired accelerator to an available instruction opcode.
`rerocc_fence`	`acq_id`	Memory fence between core memory and an acquired accelerator.
`rerocc_memrate`	`acq_id, rate`	Sets the maximum memory request rate the accelerator can make.



AURORA RUNTIME

- This runtime operates within userspace software, utilizing custom AuRORA instructions accessible in userspace, to adaptively partition available resources for multi-tenant execution
- The AuRORA runtime provides support for two key contention-aware partitioning: compute-resource allocation and memory-resource allocation
- The LatencyEst module estimates the latency of each task based on its current acquired accelerators (ACQω). Together with the remaining Slack to its target deadline, this latency is fed into the calc_score module to calculate its dynamic deadline score (ddl_score), which indicates the likelihood of meeting the target deadline (i.e., a higher score indicates it is more likely to meet its deadline)
- The Analyze function in the AuRORA runtime compares the score of task ω with the scores of other tasks to decide whether ω needs to release its acquired accelerator or acquire other idle ones, based on the relative confidence in meeting the deadline target. If release is necessary, the runtime releases acquired ac□celerators, so that tasks with tighter deadlines can acquire them.

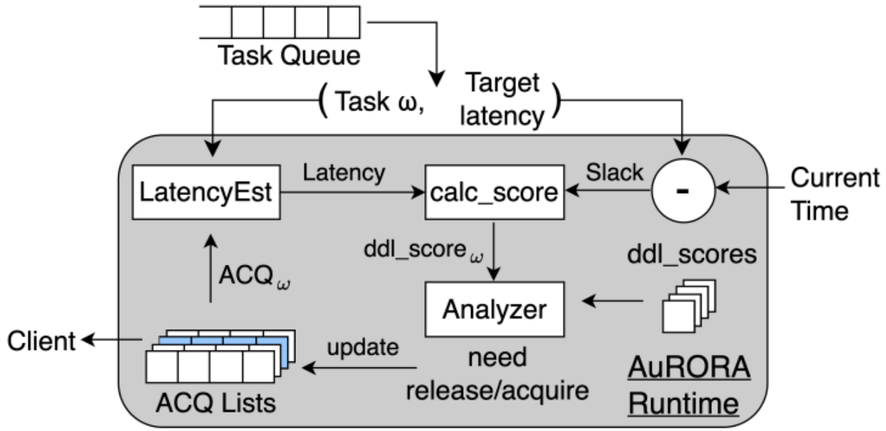
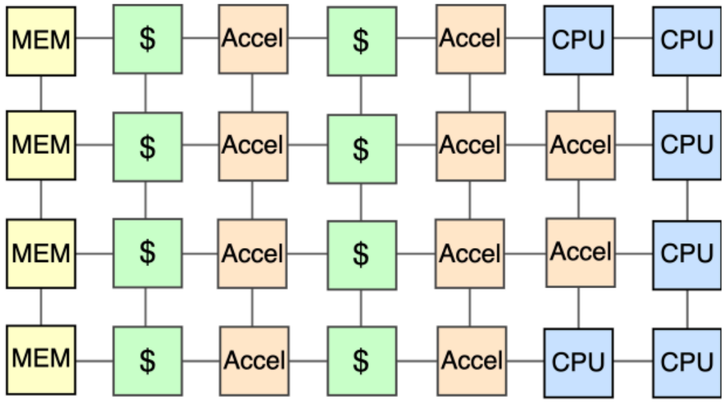


Figure 4: AuRORA runtime takes Task ω and its target latency and reconfigures the acquired accelerators for each `Client`.

AURORA RUNTIME

- **NUMA-aware compute partitioning**
- **D**istributing accelerators and memory across an SoC's network-on-chip (NoC) interconnect inevitably causes non-uniform memory accesses (NUMA)
- The AuRORA runtime quantifies each task's slowdown caused by the NUMA effect based on its assigned accelerators.
- When deciding on new accelerators to acquire, the AuRORA runtime compares the relative NUMA slowdown to co-running tasks across different accelerator assignments.
- It assigns the set of accelerators that causes the lowest relative slowdown for each task.
- In addition, AuRORA runtime performs accelerator swapping optimization before running the layer if there are idle accelerators in the system with a lower relative NUMA slowdown for the task implemented as an atomic series of acquire and release.

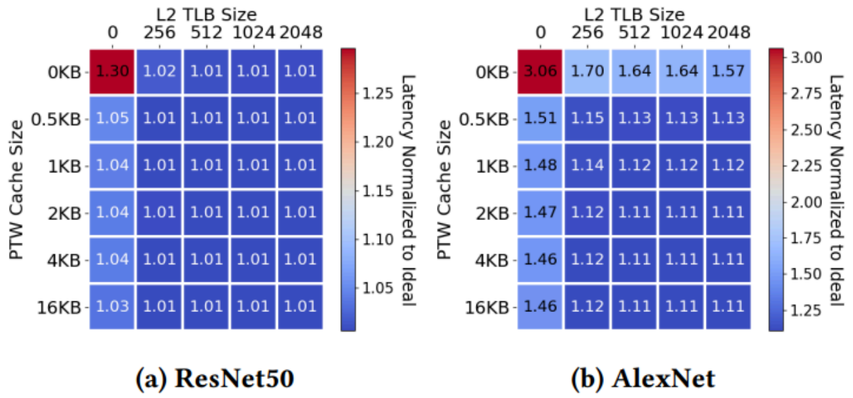


Symbol	Description
$	L2 cache bank
Accel	AuRORA manager + accelerator
CPU	AuRORA client + CPU
MEM	DRAM channel

AURORA IMPLEMENTATION

- We use **Gemmini**, a systolic-array-based DNN accelerator without multi-tenancy support, as a representative DNN accelerator in our evaluation.
- we evaluate AuRORA in two different SoC configurations

1. Crossbar: All components (AuRORA Client, Manager, memory system) are connected to a crossbar. This configuration provides a uniform memory system.
2. NoC: All components are integrated in a 7x4 2D mesh. This configuration provides a realistic and scalable NUMA memory system for a many-core/many-accelerator



(a) ResNet50 (b) AlexNet

Figure 6: Normalized latency sweeping Manager's L2 TLB and PTW cache size. Latency is normalized to the ideal case.

Table 5: AuRORA end-to-end latency overhead across SoC configurations and ResNet sizes.

# accel		ResNet50			ResNet18		
		xbar	xbar+ NoC	Shared NoC	xbar	xbar+ NoC	Shared NoC
1	Total cycle (Normalized)	1	1.07	1.07	1	1.044	1.044
	AuRORA overhead (%)	0.38	0.36	0.38	0.49	0.48	0.48
2	Total cycle (Normalized)	1	1.112	1.132	1	1.05	1.05
	AuRORA overhead (%)	0.46	0.43	0.45	0.59	0.57	0.61
4	Total cycle (Normalized)	1	1.14	1.165	1	1.076	1.079
	AuRORA overhead (%)	0.63	0.57	0.62	0.83	0.79	0.85

AURORA IMPLEMENTATION

- **Workloads:**
- DNN Models: Seven different state-of-the-art DNN inference models are used for evaluation. These are categorized into workload sets based on model size (Workload set-A: light, Workload set-B: heavy, Workload set-C: mixed).
- **Metrics:**
- SLA Satisfaction Rate: This measures the percentage of workloads meeting the Service Level Agreement (SLA) targets. Higher SLA satisfaction rates indicate better performance in meeting latency targets.
- System Throughput (STP): This measures the total system throughput of co-located applications, indicating overall hardware utilization.
- Fairness: This metric assesses the equal progress of workloads under multi-tenant execution compared to isolated execution.
- **Results:**
- SLA Satisfaction Rate:
- Crossbar Configuration: AuRORA's virtual compute resource partitioning significantly improves SLA satisfaction rates across various scenarios, achieving up to 2.68× improvement over baseline methods.
- NoC Configuration: Similar improvements are observed, with dynamic resource re-partitioning contributing to better performance under varied workloads.