Polymorphic Pipeline Arrays: Expanding CGRAs Beyond Innermost Loops

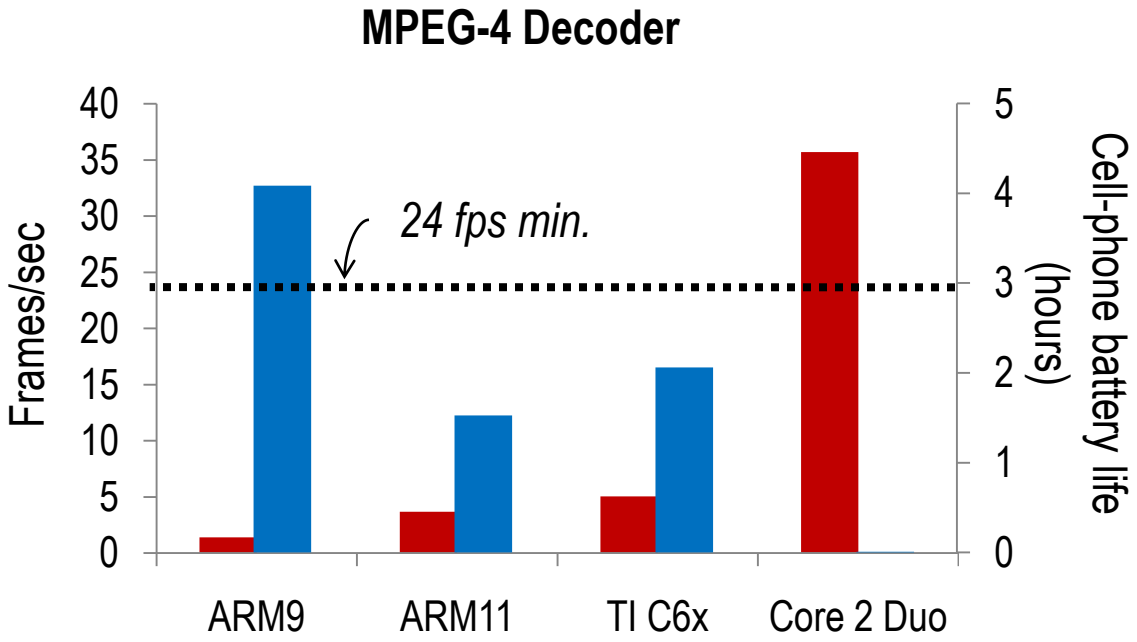
Scott Mahlke

EECS Dept., University of Michigan

With lots of help from: Yongjun Park and Hyunchul Park (now at TI)
in collaboration with Samsung Advanced Institute of Technology

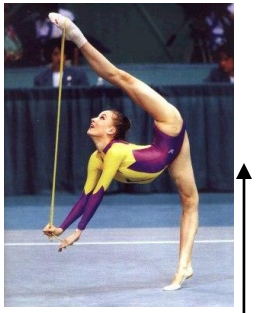
October 2010

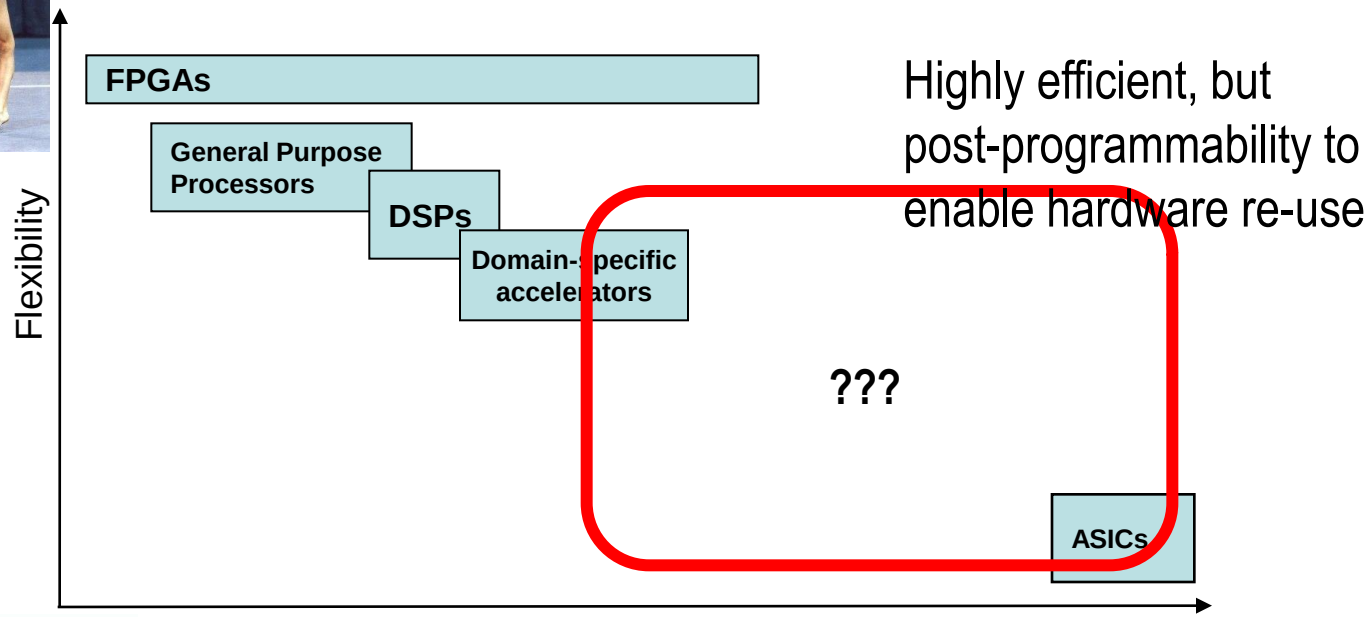
Introduction



- Emerging applications have high performance, cost, energy demands
  - High-quality video
  - Flash animation
- Clear need for application and domain-specific hardware

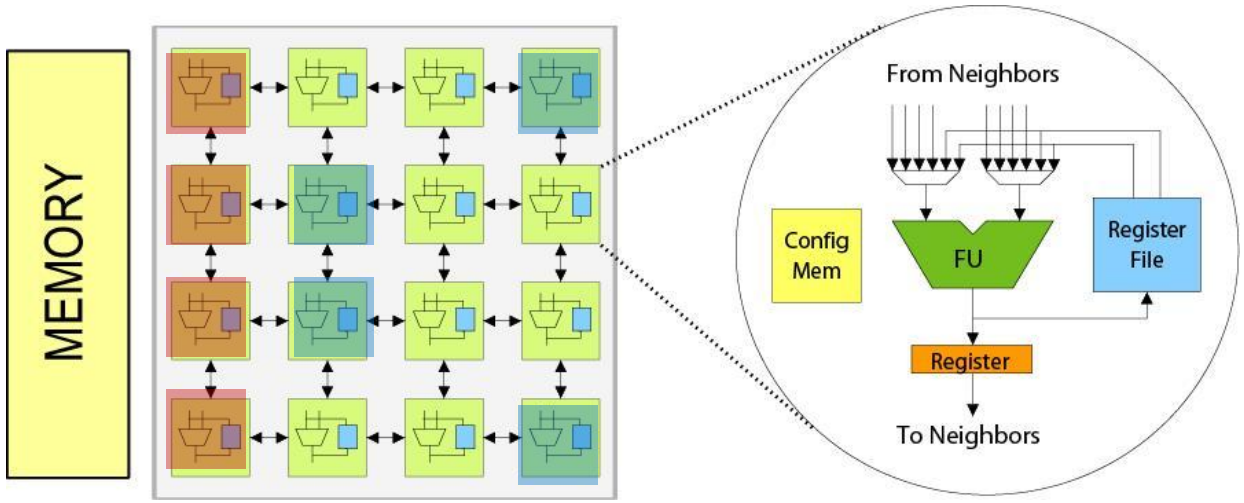
Hardware Alternatives





Highly efficient, but post-programmability to enable hardware re-use

Coarse-Grained Reconfigurable Architecture (CGRA)

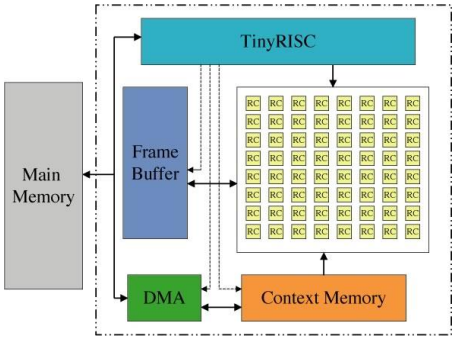


- Array of PEs connected in a mesh-like interconnect
- High throughput with a large number of resources
- Distributed hardware offers low cost/power consumption
- High flexibility with dynamic reconfiguration

CGRA : Attractive Alternative to ASICs

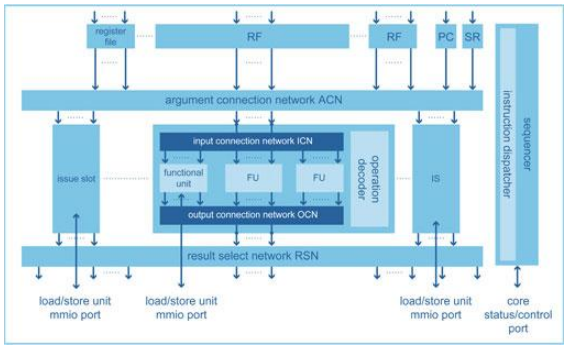
- Suitable for running multimedia applications for future embedded systems
  - High throughput, low power consumption, high flexibility
- Morphosys : 8x8 array with RISC processor
- SiliconHive : hierarchical systolic array
- ADRES : 4x4 array with tightly coupled VLIW

Morphosys



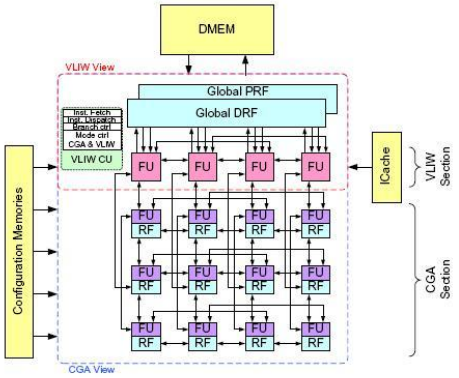
viterbi at 80Mbps

SiliconHive



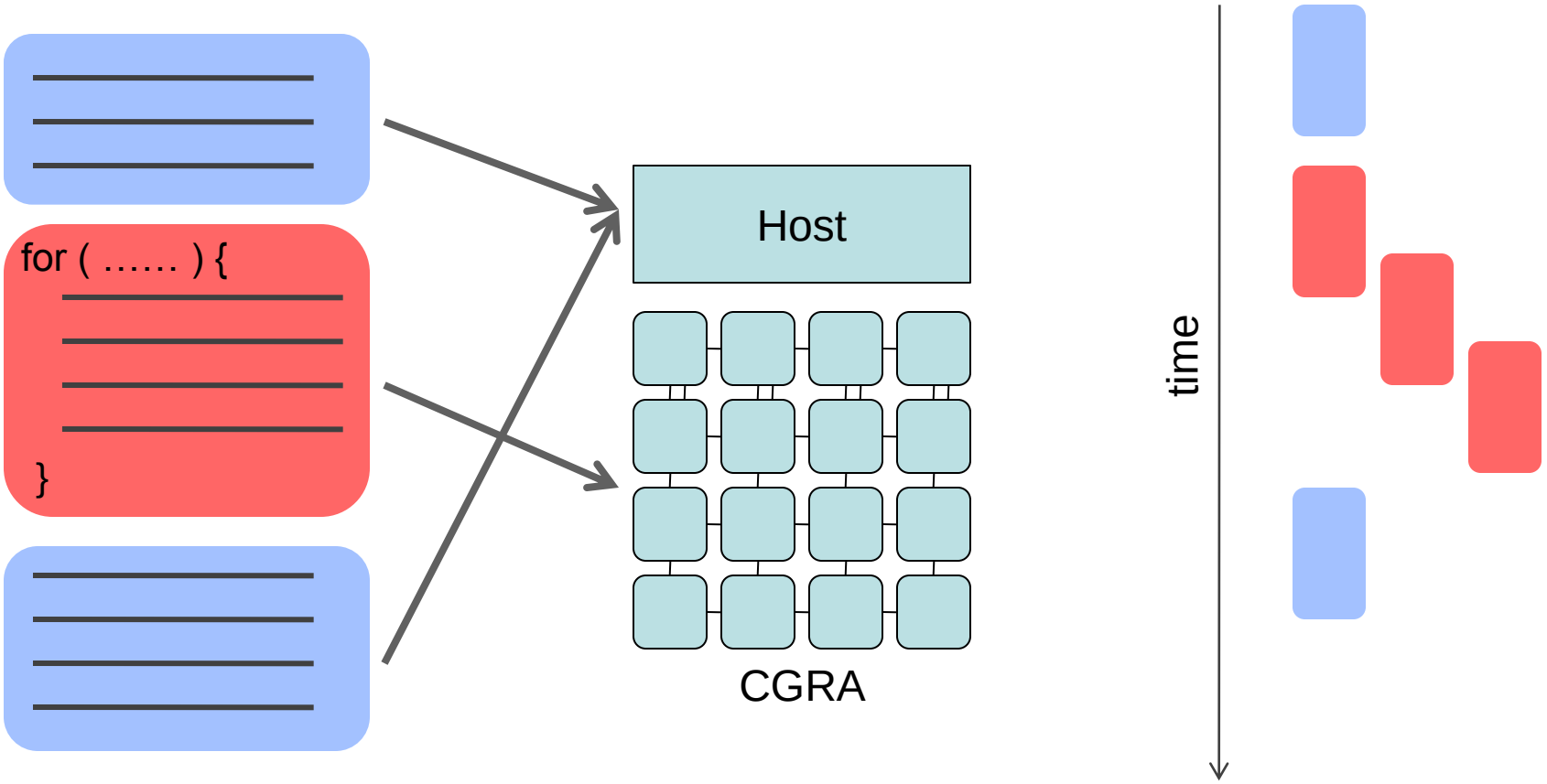
h.264 at 30fps

ADRES



50-60 MOps /mW

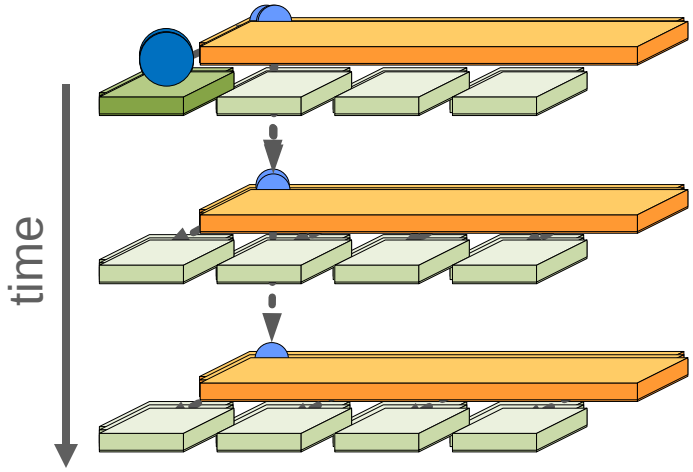
Execution Model of CGRAs



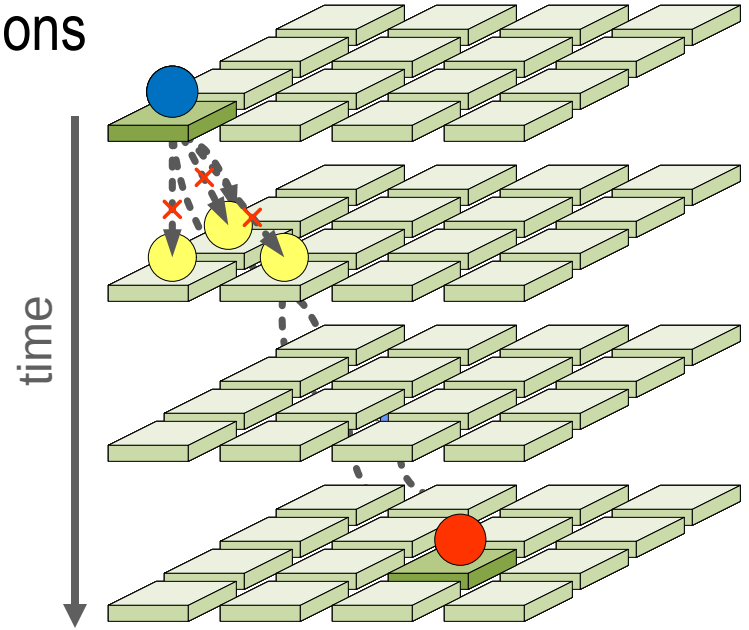
- Exploit instruction-level parallelism through modulo scheduling

CGRA Scheduling Challenges

- VLIW : routing is guaranteed by central RF
- CGRA : Multiple possible routes
- Compiler is responsible for finding routes
- Routing can easily fail by other operations

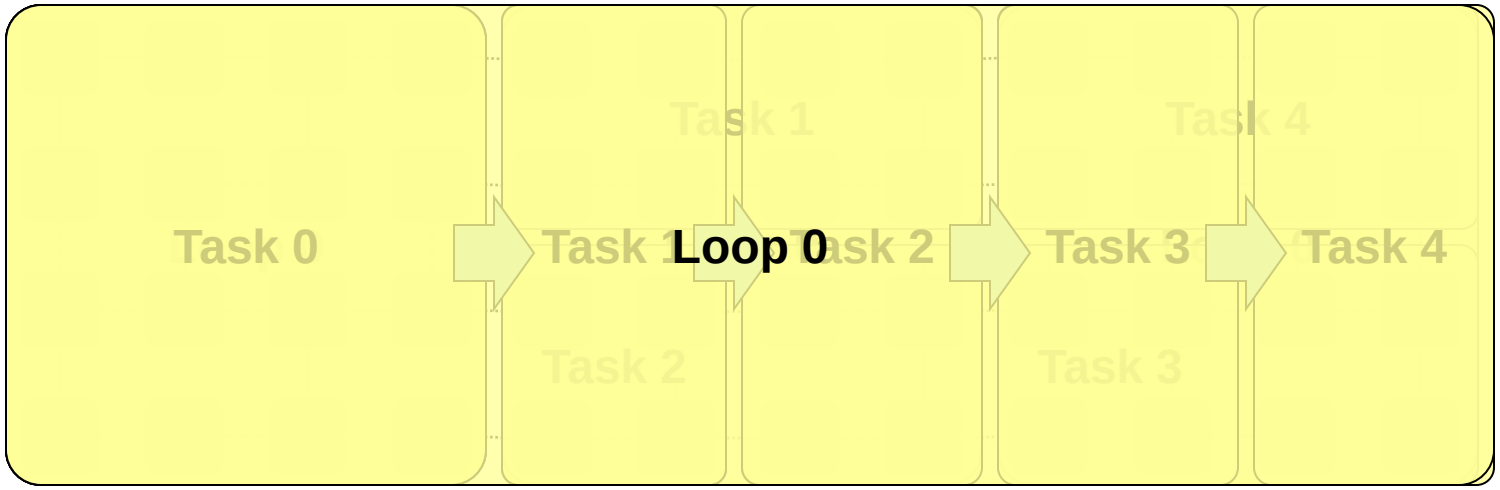


VLIW



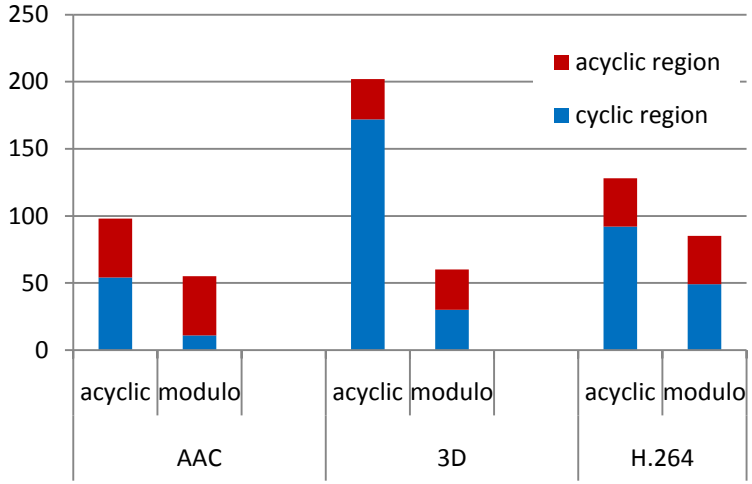
CGRA

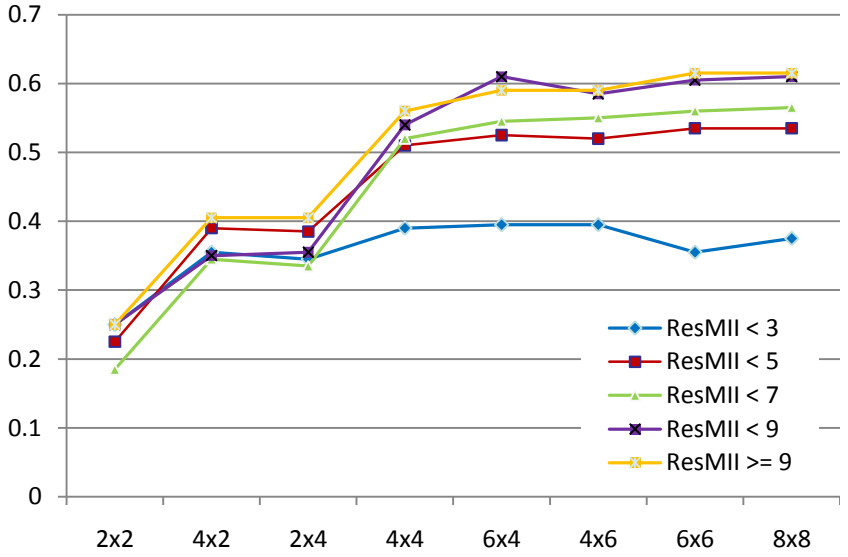
Large Scale CGRA



- Increasing technology allows more resources available
- Various ways to exploit the extra resources

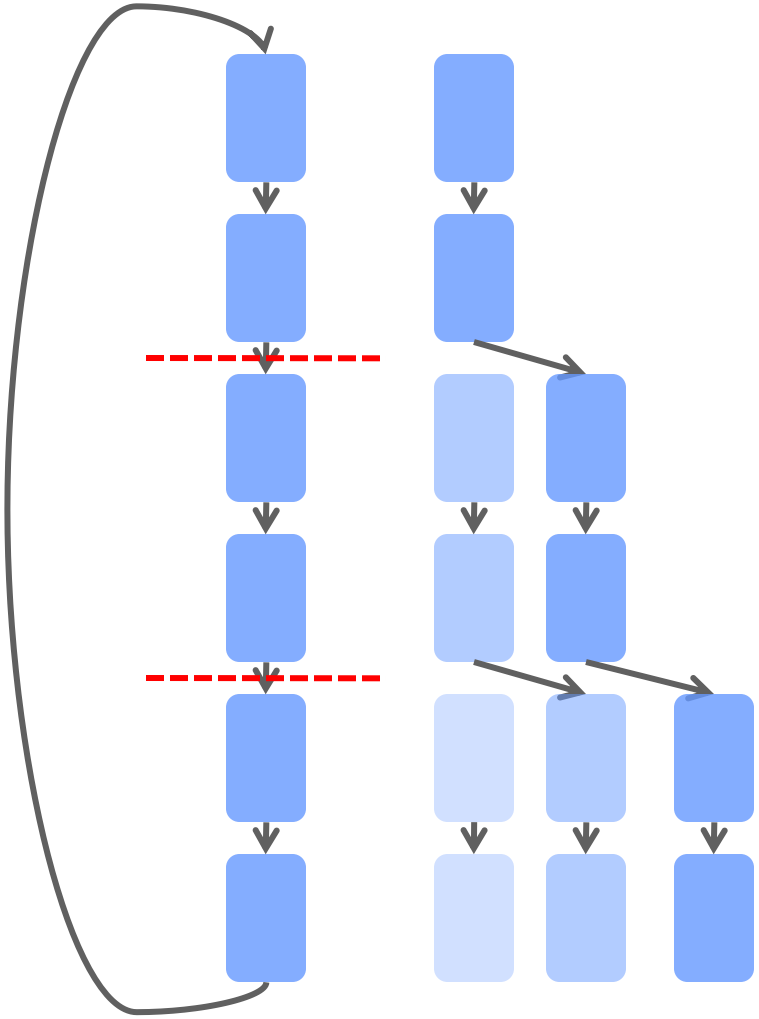
Limitations of CGRAs





- Acyclic code region becomes bottleneck after software
- Performance gain diminishes beyond 4x4

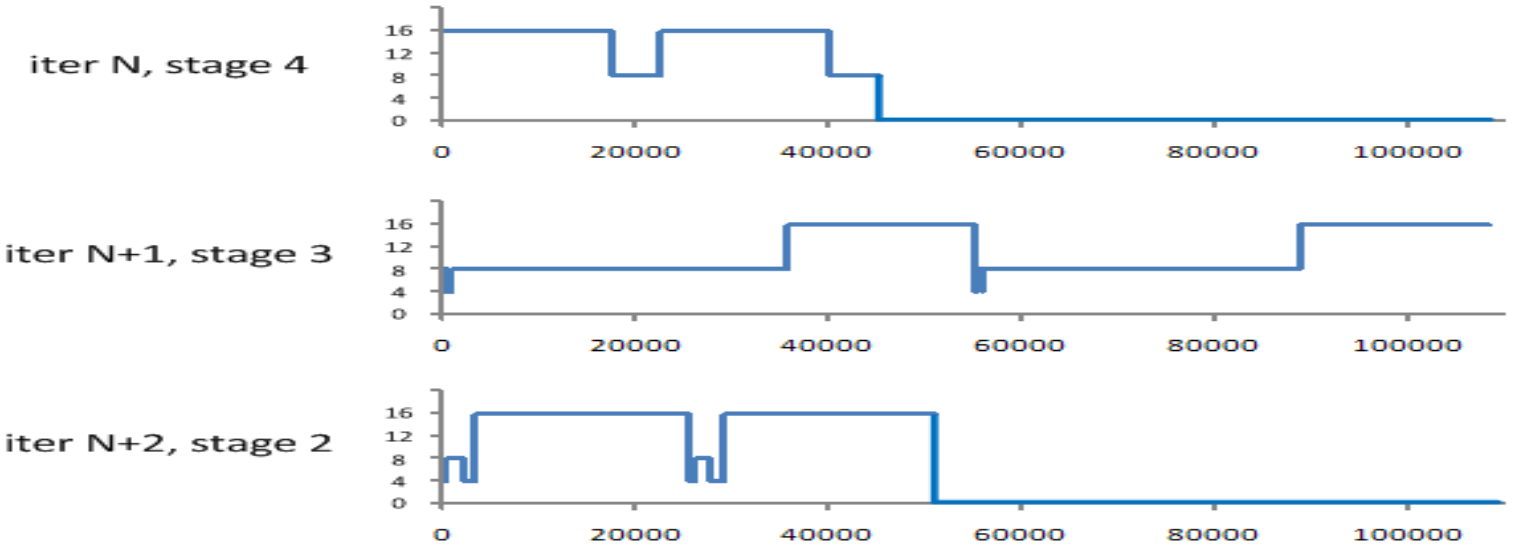
Streaming Execution Model



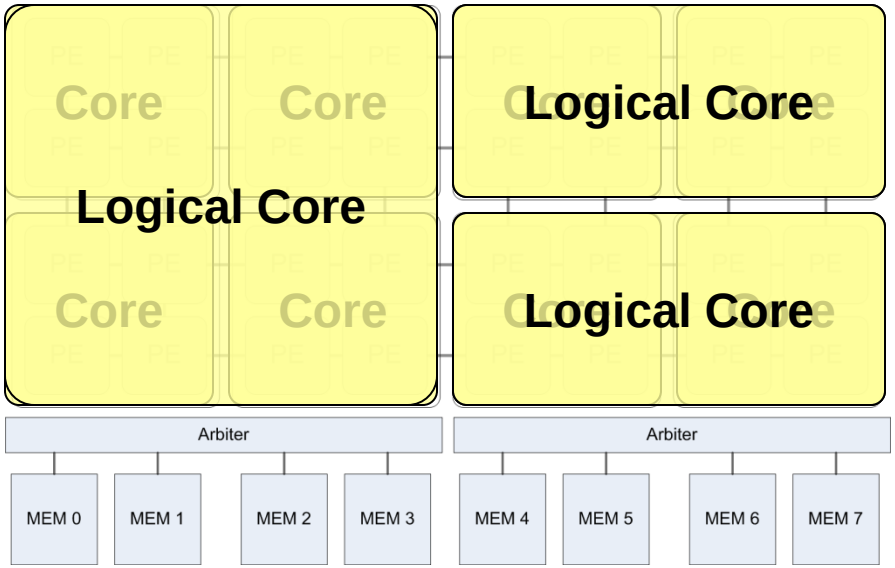
- Streaming property
  - Packet of data goes through independent tasks
- Partition tasks into stages
  - Map each stage onto different hardware
- Pipeline parallelism
  - Pipeline the outermost loop

Media Application Characteristics

- Multimedia applications rich both in ILP/pipeline parallelism
  - Not mutually exclusive, cooperatively enhance performance
- But, resource requirements vary statically/dynamically

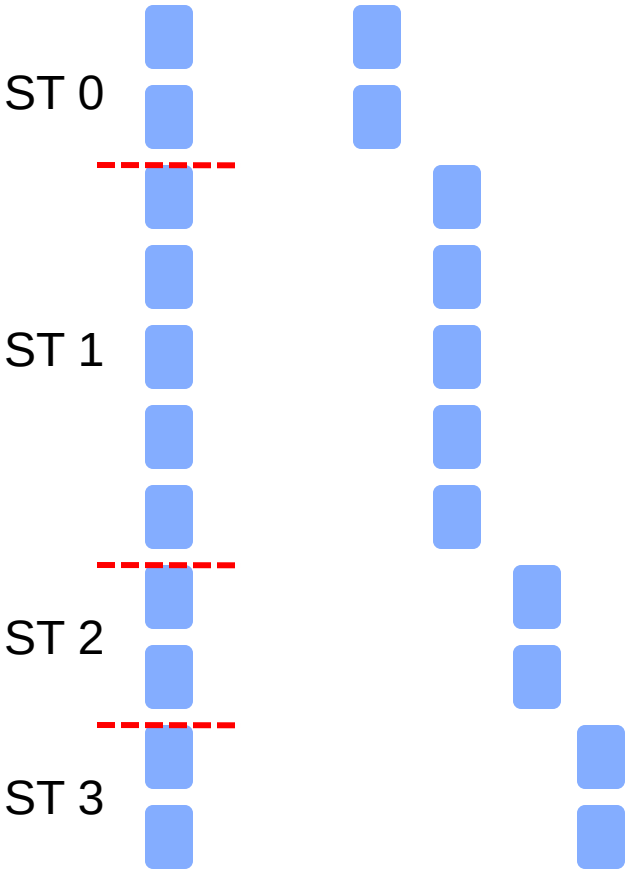


Polymorphic Pipeline Array

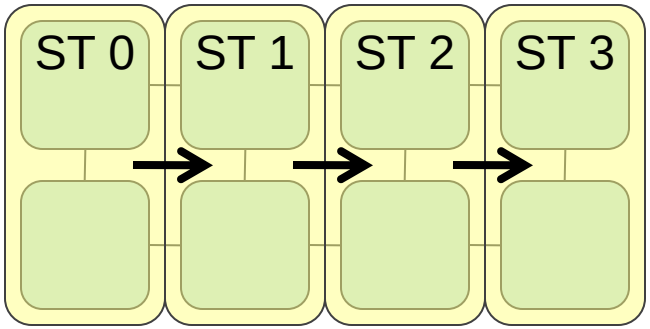


- Multi-core accelerator : each 2x2 array becomes a processor
- Cores can be combined to form a larger logical core
- Exploit both coarse-grain and fine-grain pipeline parallelism
- No dynamic routing logic: all communications statically generated

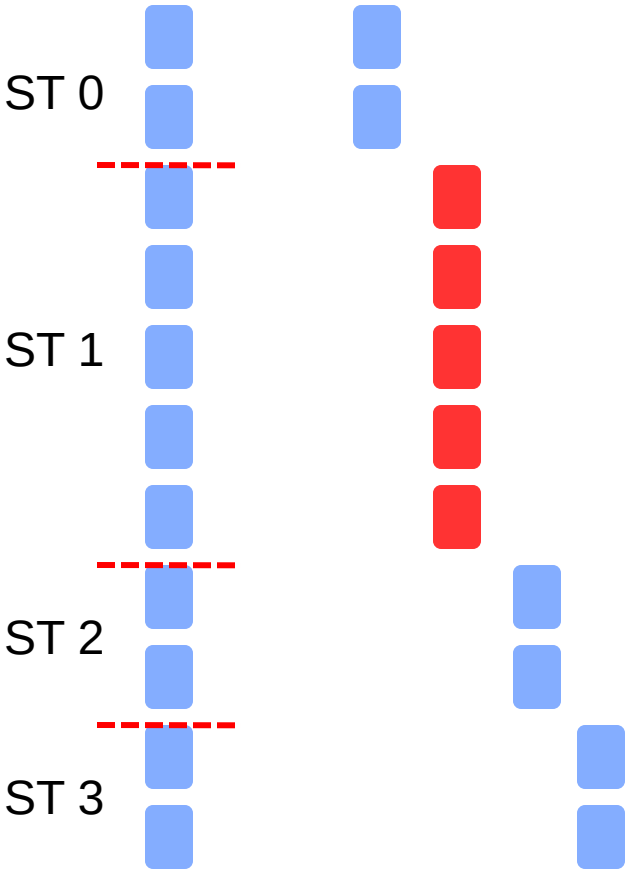
PPA Execution Model



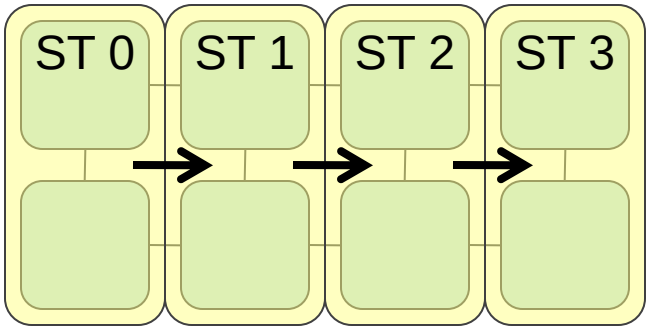
- Pipeline outermost loop



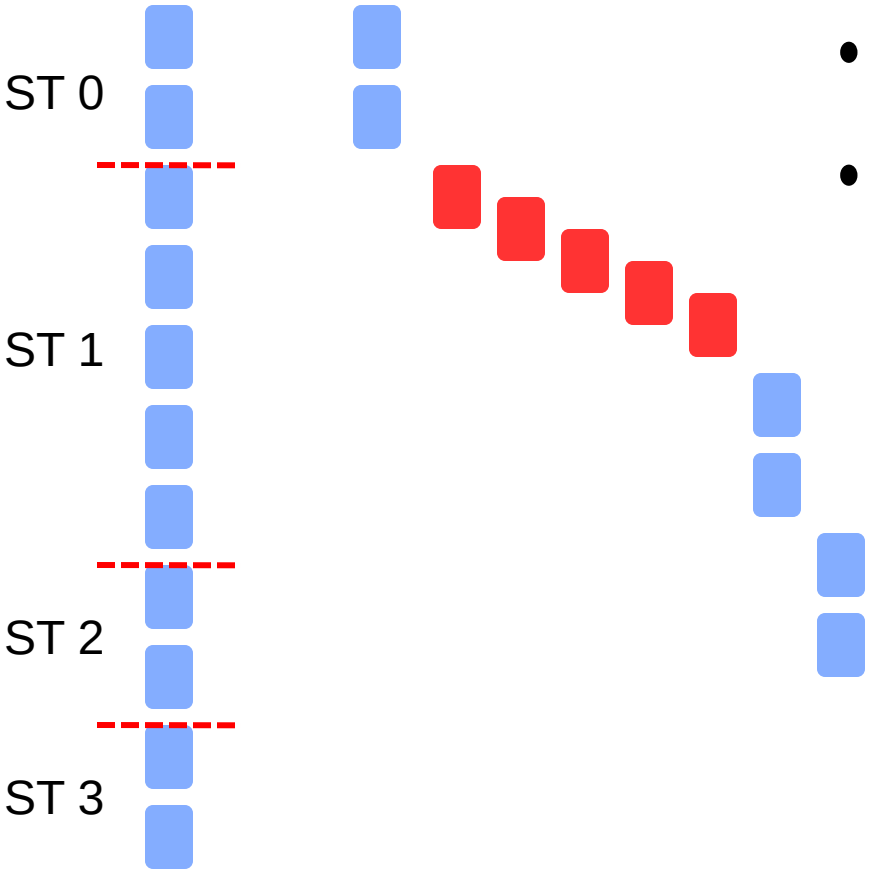
PPA Execution Model



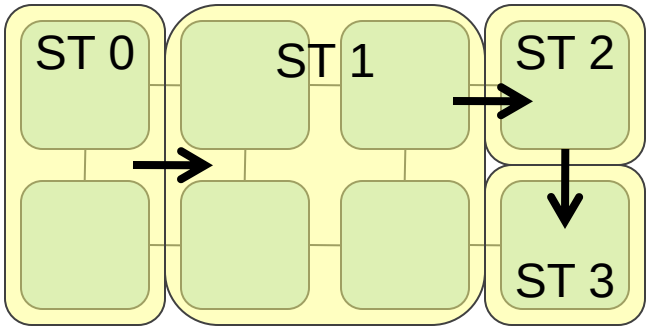
- Pipeline outermost loop
- Compute intensive stage
  - Assign more resources
  - Modulo scheduling



PPA Execution Model

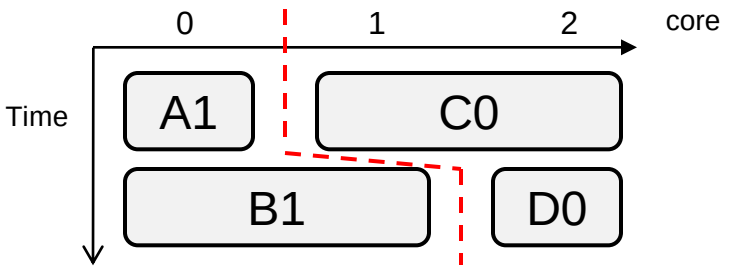


- Pipeline outermost loop
- Compute intensive stage
  - Assign more resources
  - Modulo scheduling

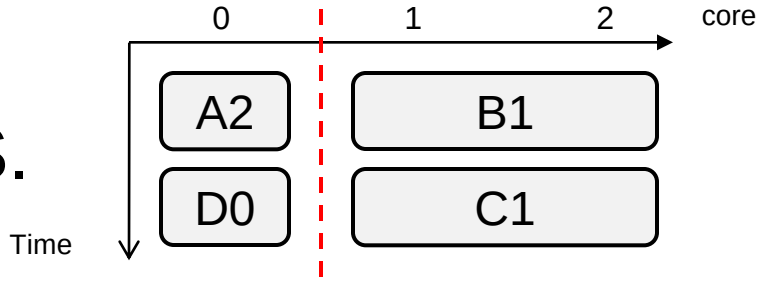


PPA Partitioning: Static

- Grouping tasks with same resource assignment
  - Goal (avoid resource conflict!)
    - Keep core-merging configuration at runtime
    - Give more resource to slowest task
  - Minimize stall time due to the resource conflict or reconfiguration



VS.

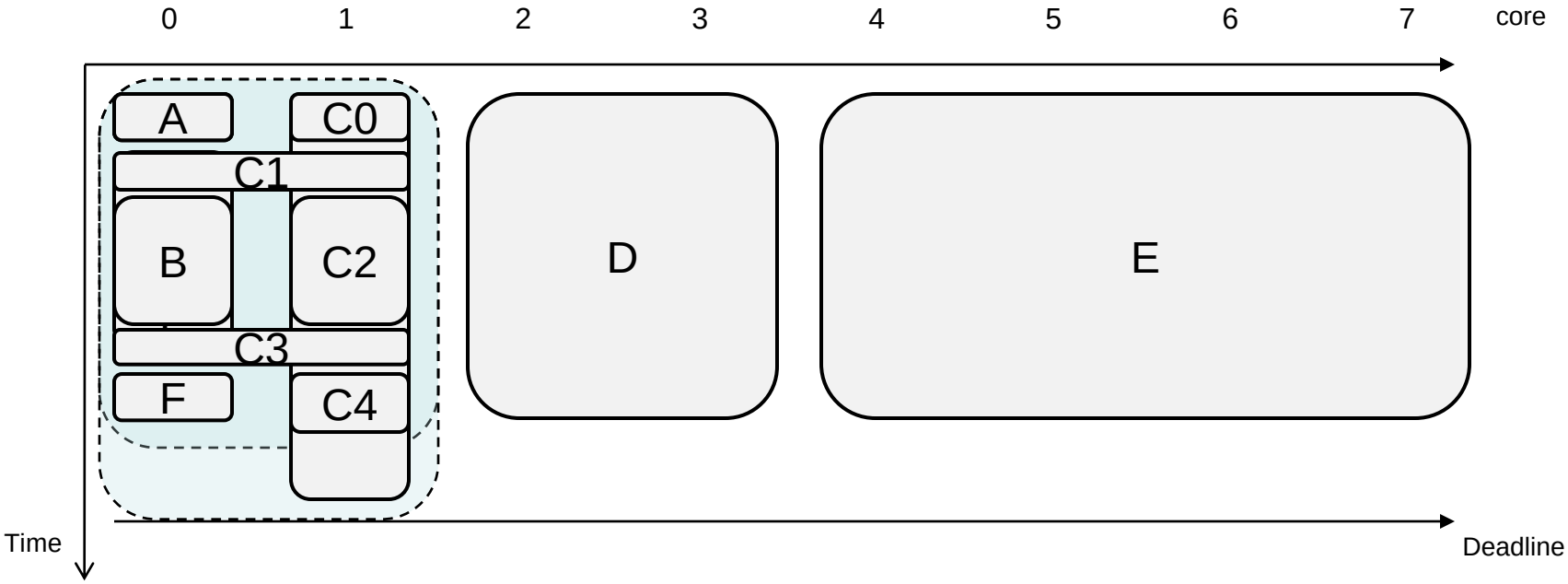


Resource conflict
2 Stage(AB->CD)

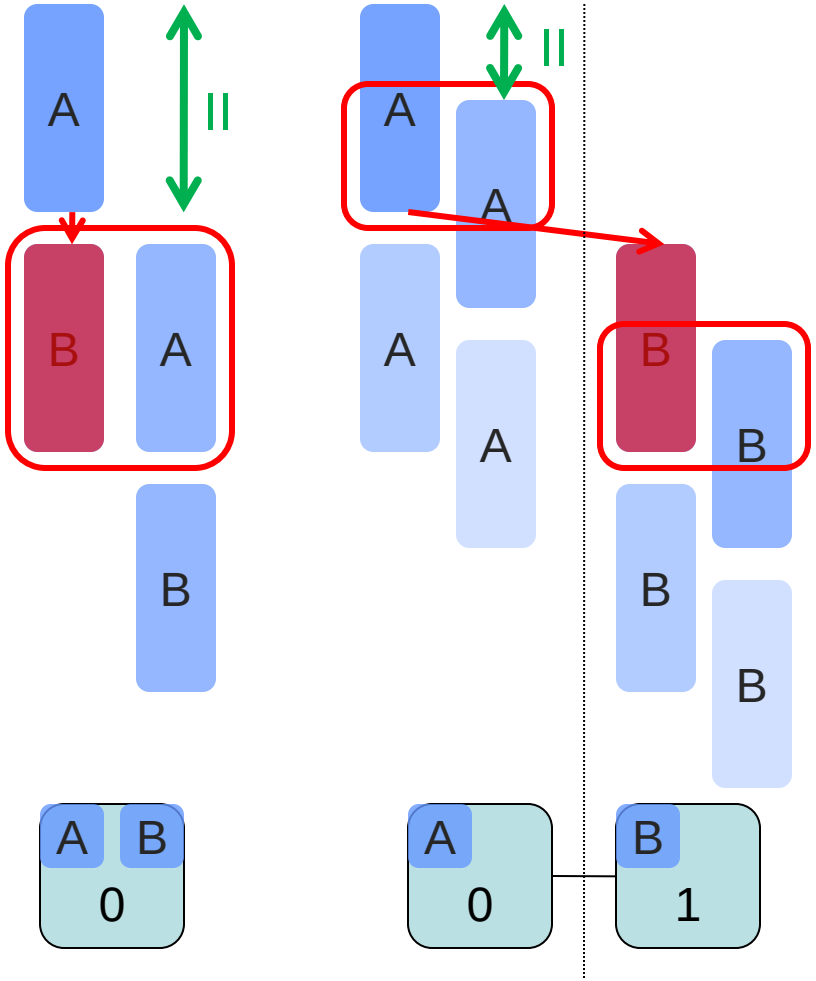
No resource conflict
3 Stage(A>BC->D)

PPA Partitioning: Dynamic

- Final performance tuning
  - Goal: minimized the highest execution time reusing neighbor core's resource.
  - Software pipelining for special fraction of the task using both own and neighbor core's resources.

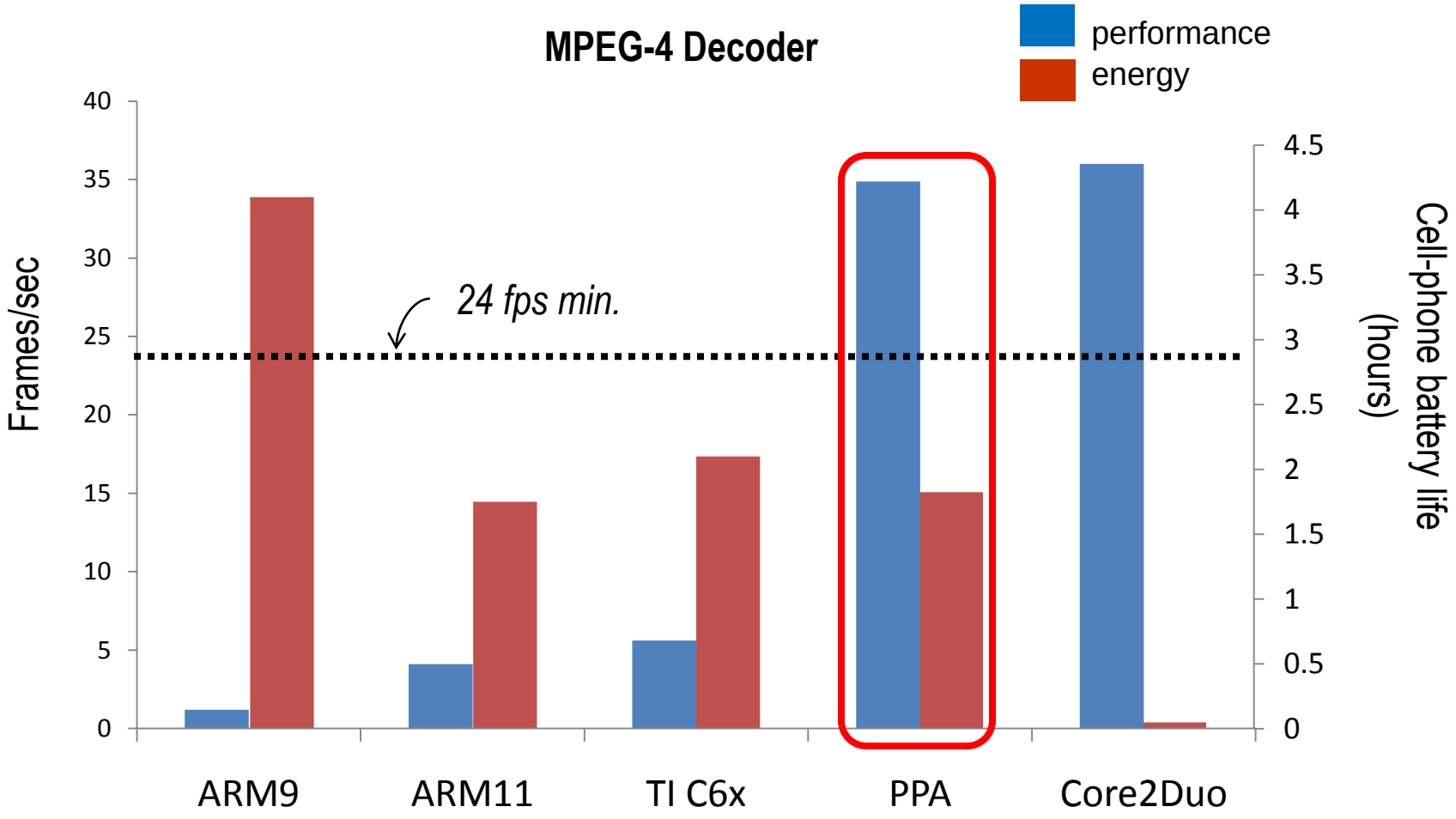


Realizing Dynamic Partitioning: Virtualized Modulo Scheduling



- One binary that can run in multiple targets
  - Part of code migrate to neighboring core
  - No rescheduling
- Challenges
  - Avoid resource conflict
  - Enforce multiple modulo constraints
  - Inter-core communication

Where PPA stands



Concluding Remarks

- Polymorphic pipeline array
  - Efficient scaling of CGRA performance
  - Shift focus from ILP to coarse-grain pipeline parallelism
- Resource usage variability
  - Static: Match resources to work
  - Dynamic: Repartition to recycle idle resources
- Ongoing work
  - Memory system design
  - Integration of SIMD processing