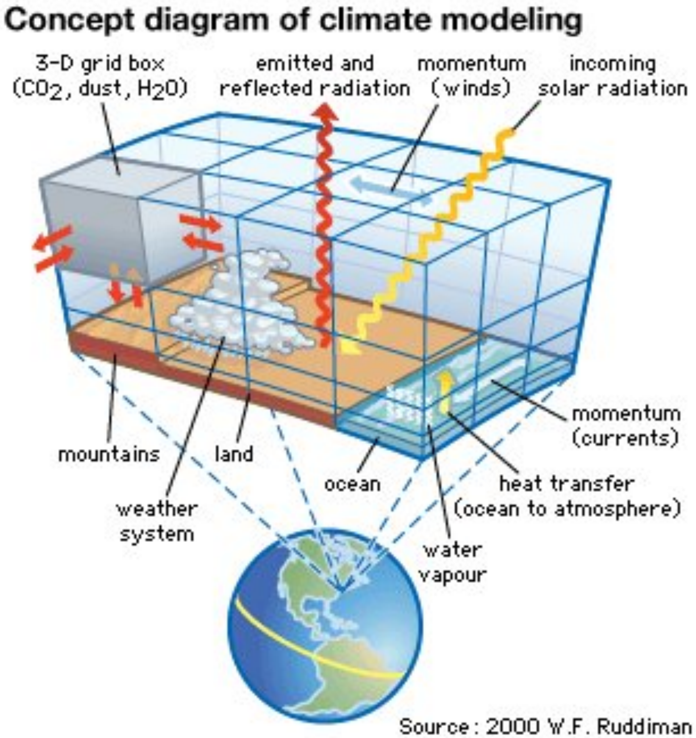
Beyond the hill of Multicores lies the valley of Accelerators

Aviral Shrivastava

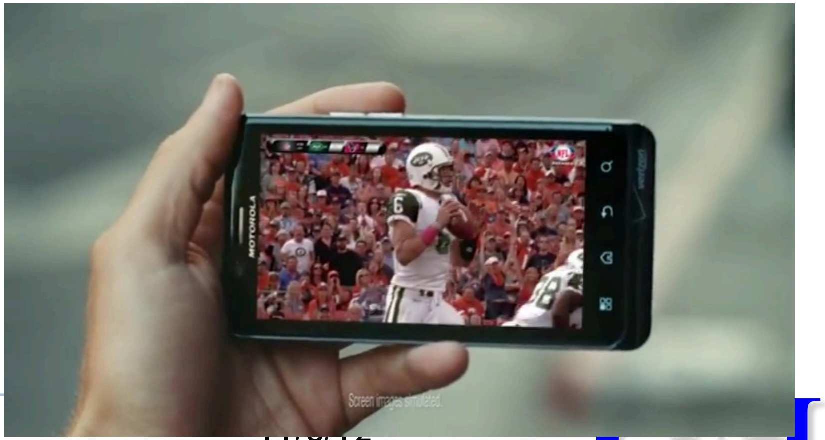
Compiler Microarchitecture Lab
Arizona State University

Need of Higher Performance

- High Performance Computing
  - Weather modeling
  - Modeling carbon sequestration
  - Simulating fusion reactors
- Desktop Computing
  - Applications grow to fill in the computational capabilities
- Embedded Computing
  - Software Defined Radios
  - Baseband processing, 4G, LTE
  - Target Recognition and Collision avoidance at Supersonic speeds

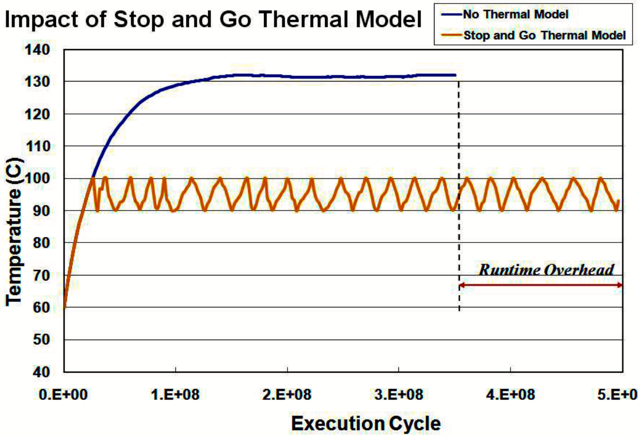

Concept diagram of climate modeling
3-D grid box (CO2, dust, H2O)
emitted and reflected radiation
momentum (winds)
incoming solar radiation
mountains
land
weather system
ocean
momentum (currents)
heat transfer (ocean to atmosphere)
water vapour

Source : 2000 W.F. Ruddiman



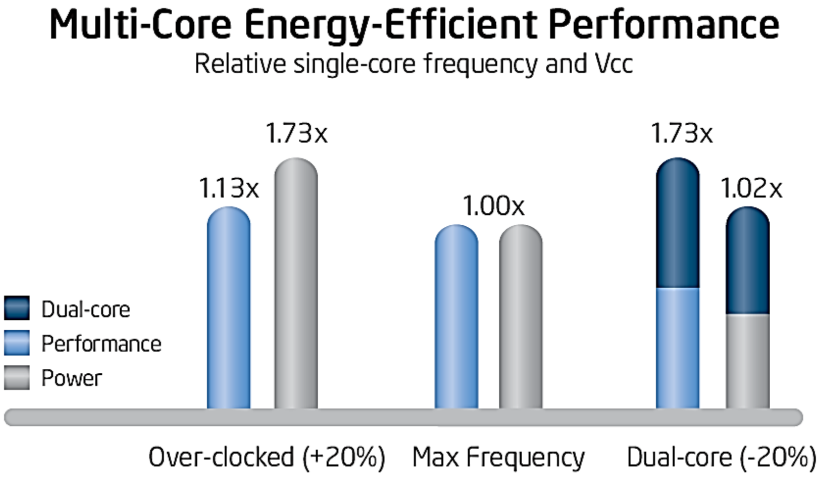
Power-Efficiency is the Bottleneck

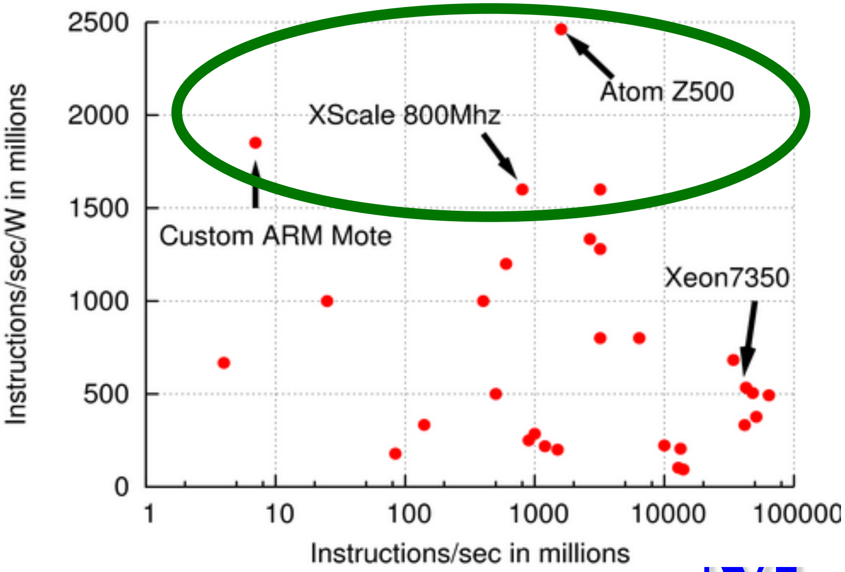
- Data Center Level
  - Total power consumption of data centers is already ~ 20 MW
  - 1/3rd of operating costs are electricity bills
- Chip Level
  - Total power dissipation already more than cooling efficiency of packaging
  - Dynamic Thermal Management
    - a.k.a. turbo boost (Intel)
- Hardware block level
  - Local hotspots, e.g., RF
  - Stop/slow down if any component heats up



Multi-cores pave the road ahead…

- Make cores smaller/simpler
  - Power-efficiency of system is proportional to the power efficiency of a core
- Make many cores
  - For performance
- Run cores at lower frequency
  - Cubic increase in power-efficiency
- New Moore's law





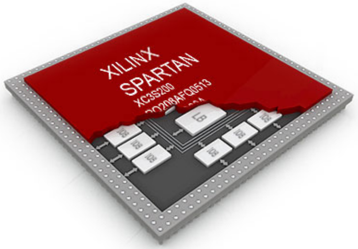
Multi-cores .. How far will they go?

- Cannot scale indefinitely
  - The "U-curve" of power-efficiency
    - Power-efficiency decreases as we make cores simpler
  - Power-efficiency improvements reduce
    - As V_{dd} approaches V_{th}

- Today: Intel core 2 quad
  - 0.3 GOps/W
- HPC target - Exascale computing
  - 10^{18} operations per second in 50MW
  - 20 GOps/W ~100X more power-efficiency
- Embedded target: 4G SDR
  - 100 GOps/W ~1000X more power-efficiency

Accelerators: Beyond Multi-cores

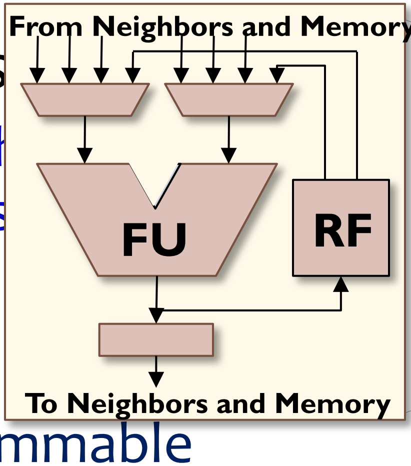
- Power and performance critical computations can be off-loaded to accelerators
  - Perform application specific operations
  - Achieve high throughput without loss of CPU programmability
  - Hardware Accelerator
    - Intel SSE
  - Reconfigurable Accelerator
    - FPGA
  - Graphics Accelerator
    - nVIDIA Tesla, Fermi

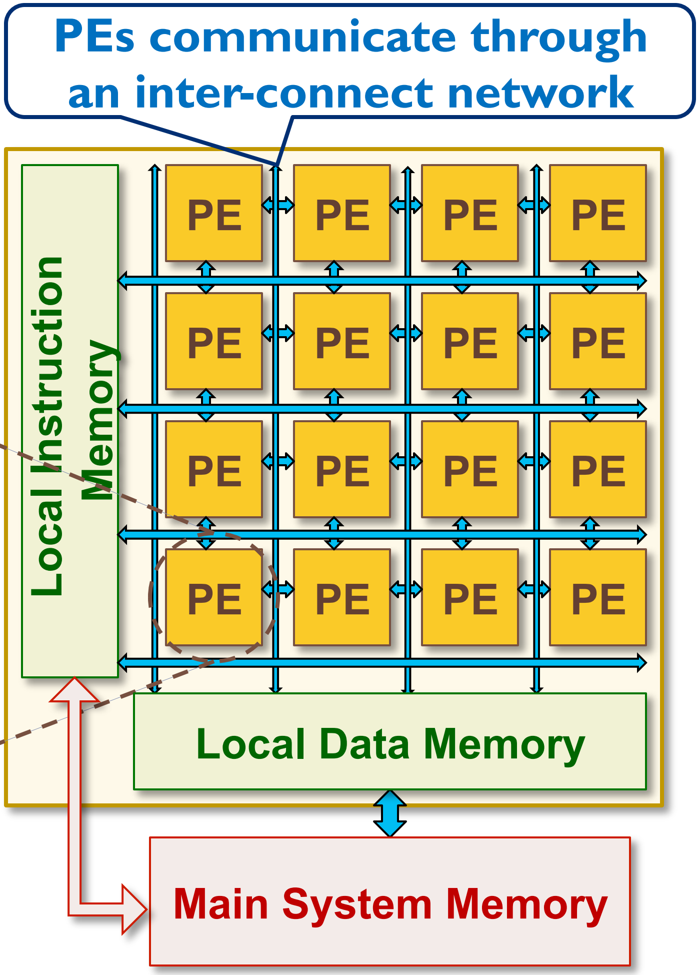


Coarse Grain Reconfigurable Array

- Distinguishing Characteristics
  - Only an order of magnitude less power-efficient than ASICs
    - 50 Gops/W efficiency - ADRES
  - But Programmable
  - High performance
- Several CGRA Des

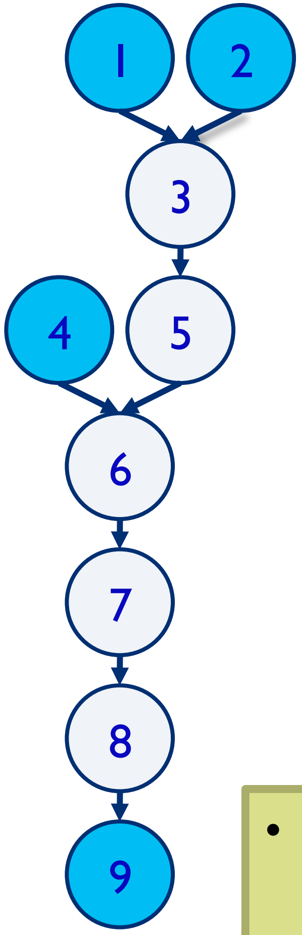
  ADRES[IMEC], Morph
  KressArray[Kaisers
  RSPA[SNU], etc.
- Comparisons
  - vs. ASICS – Programmable
  - vs. FPGA – More power-efficient, and simpler to compile
  - vs. GPUs – More general purpose





How does a CGRA Work?

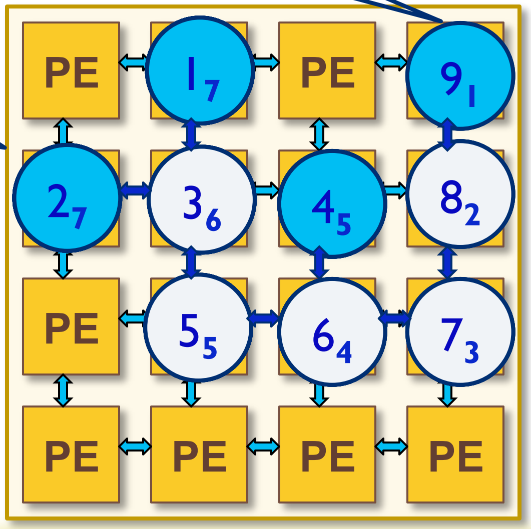
Data-Dependency Graph:



Execution time slot:
(or cycle)

After cycle 6, one iteration of loop completes execution every cycle

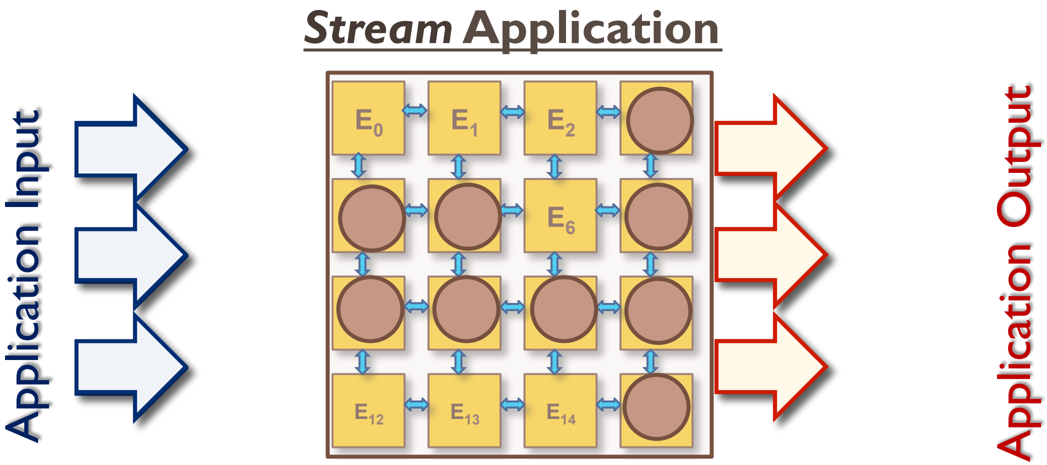
Entire kernel can be mapped onto CGRA by unrolling **6** times



- **Initiation Interval (II) is the quality metric**
- **Very efficient computation**

Initiation Interval = 1

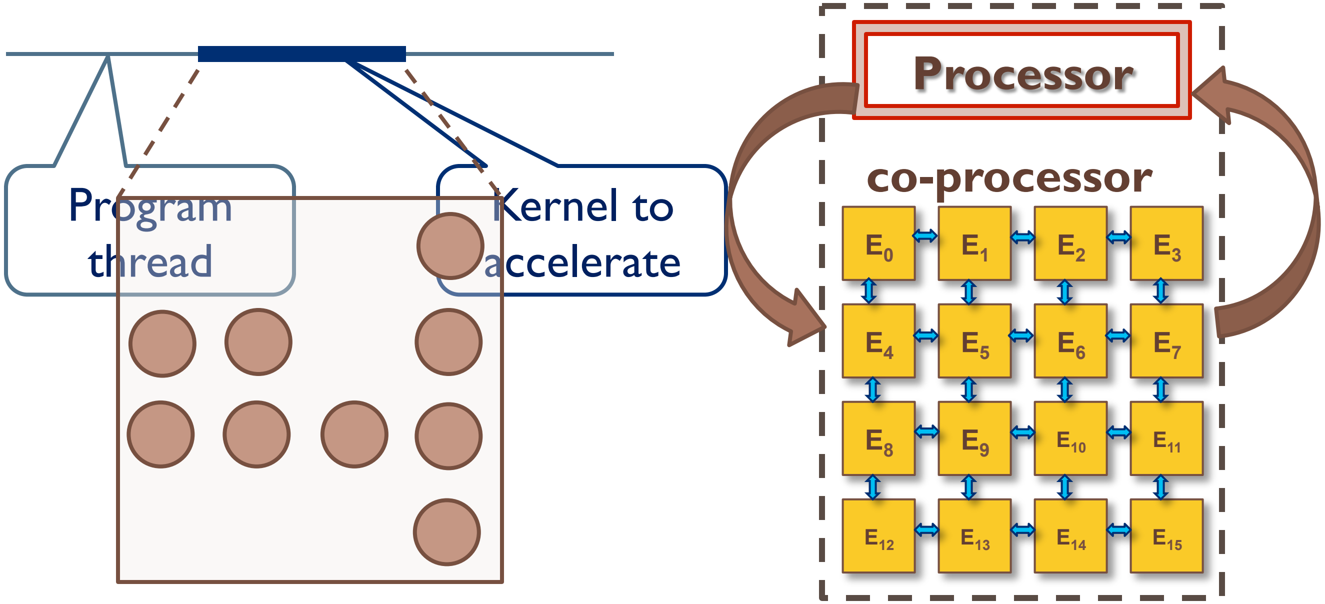
CGRA for Streaming Applications



- Traditionally used for *streaming* applications
- The application kernel is mapped onto the CGRA
- Inputs stream into the CGRA and outputs stream out

- **High performance at high power-efficiency**

CGRA as Accelerator



- Specific kernels in a thread can be power/performance critical
- The kernel can be mapped and scheduled for execution on the CGRA
- **Using the CGRA as a co-processor (accelerator)**
  - Improvement in execution time
  - Reduction in power consumption

Using CGRA as a GP Accelerator

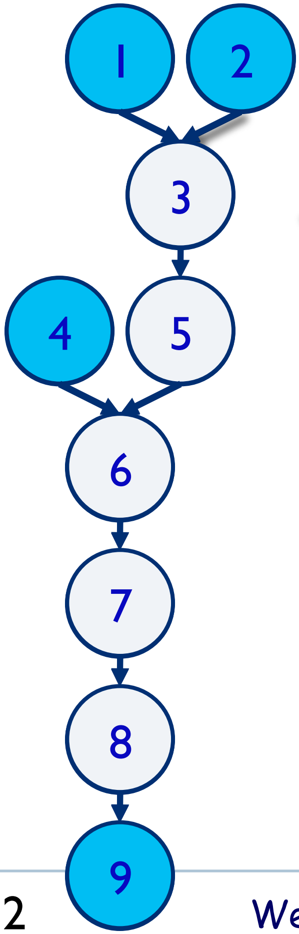
- Compilation is hard
  - CGRA execution is completely statically determined
- Compiler must also perform
  - Explicit scheduling
  - Binding – map operations to PEs
  - Routing – map data dependencies to CGRA edges
- Explicit data management
  - Works using scratch pads
  - Need to know what data the loop will need
- Explicit control flow
  - Reduces CGRA utilization

Holy grail for compiler researchers

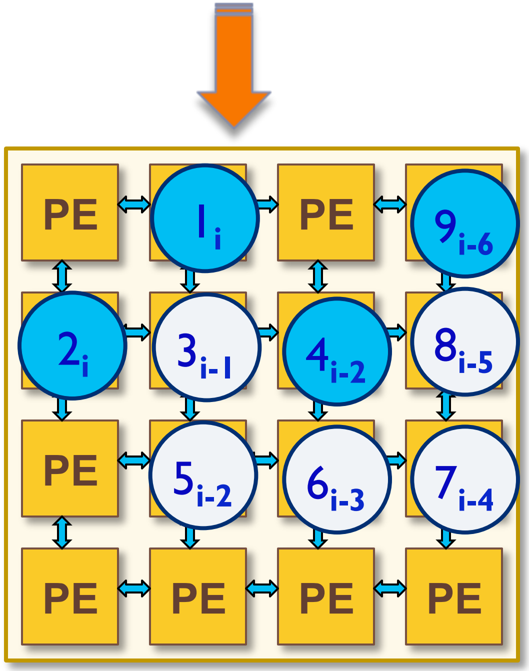
How to Map a Kernel onto CGRA?

Software Pipeline, and **map** the loop on the CGRA, preserving data dependencies, and minimize *II*

Data-Dependency Graph:



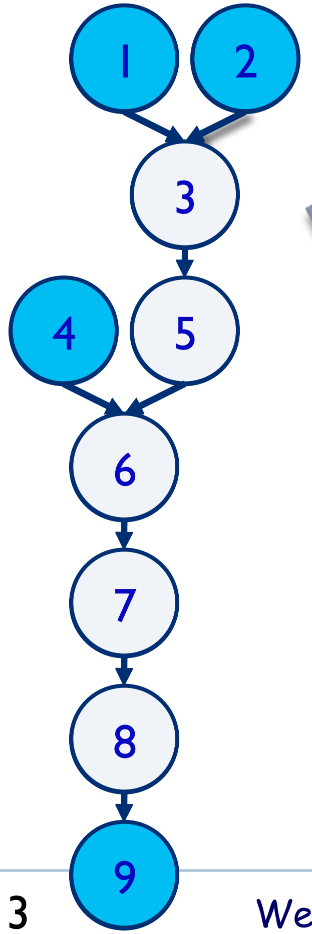
$1^i, 2^i, 3^{i-1}, 4^{i-2}, 5^{i-2}, 6^{i-3}, 7^{i-4}, 8^{i-5}, 9^{i-6}$



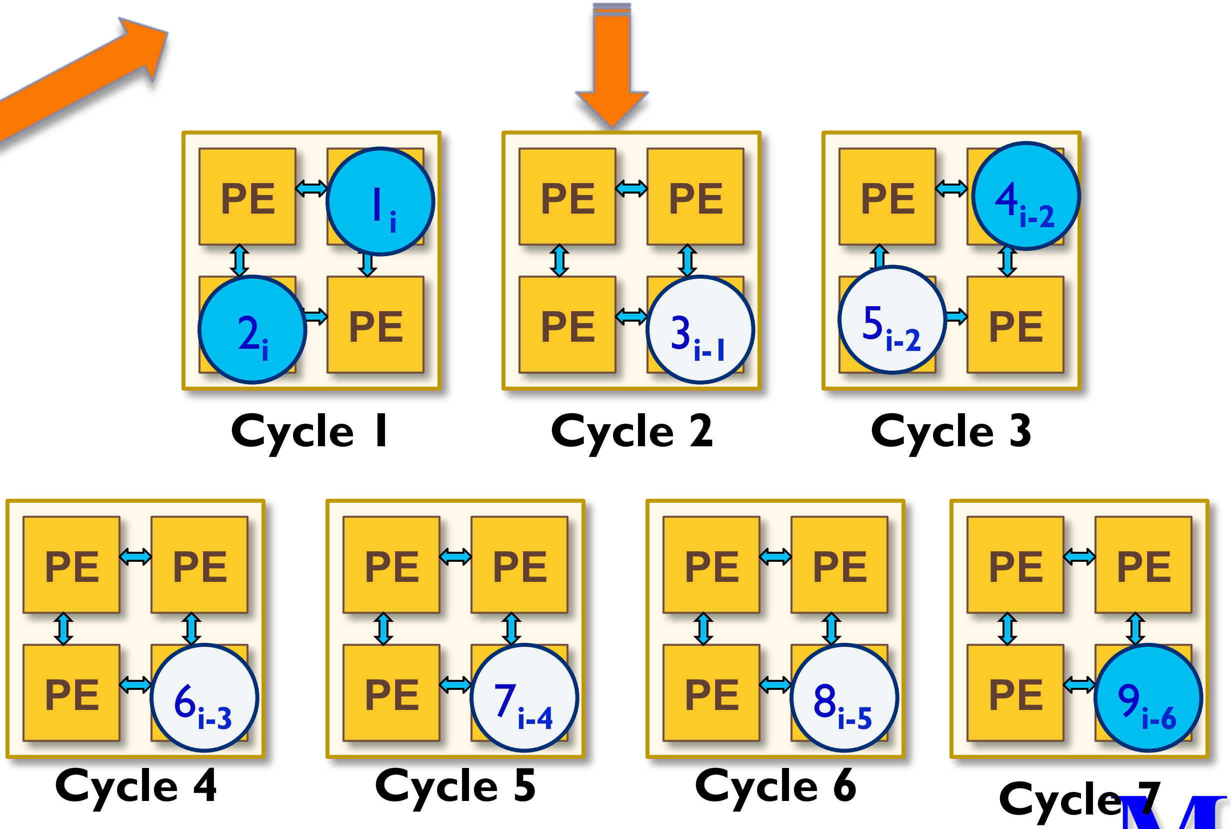
How to Map Kernel onto a CGRA?

Software Pipeline, and **map** the loop on the CGRA, preserving data dependencies, and minimize *II*

Data-Dependency Graph:



$1^i, 2^i, 3^{i-1}, 4^{i-2}, 5^{i-2}, 6^{i-3}, 7^{i-4}, 8^{i-5}, 9^{i-6}$



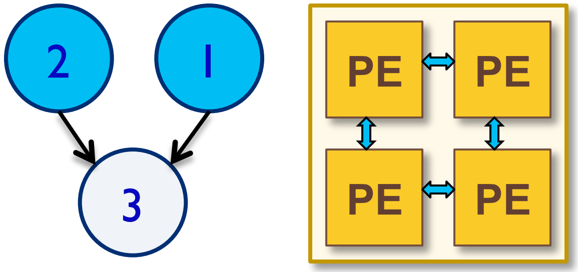
Outline of this Presentation

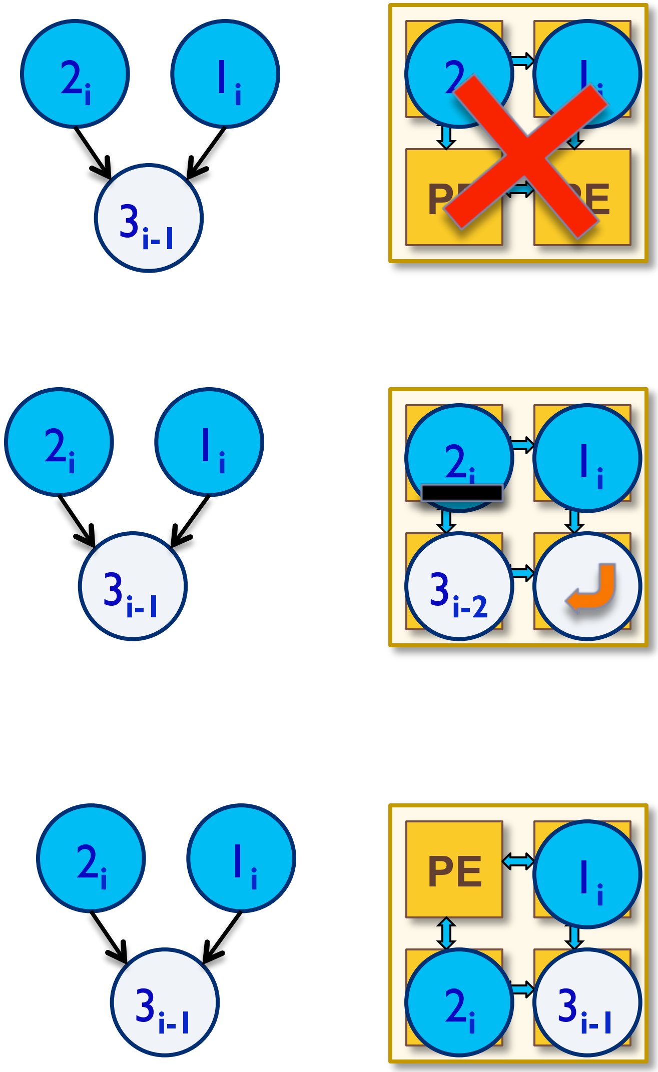
- Power-efficiency is the design metric
  - CGRAs offer a extremely power-efficient and high performance computing platform
  - Several compiler challenges remain

- Presenting
  1. Spatial Mapping on a CGRA
  2. General Problem Definition
     - Models both routing and re-computation
     - EpiMAP: New State-of-the-Art
  3. Enable Multi-threading on CGRAs
     1. Runtime shrinking the schedule

- More Work on
  - Memory-aware mapping on CGRAs
  - Bank conflicts and interleaving

Spatial Mapping

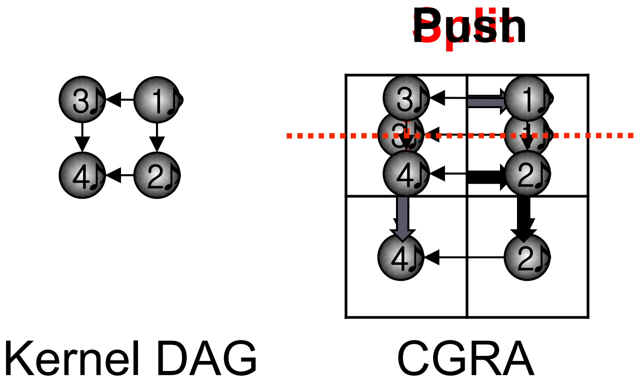




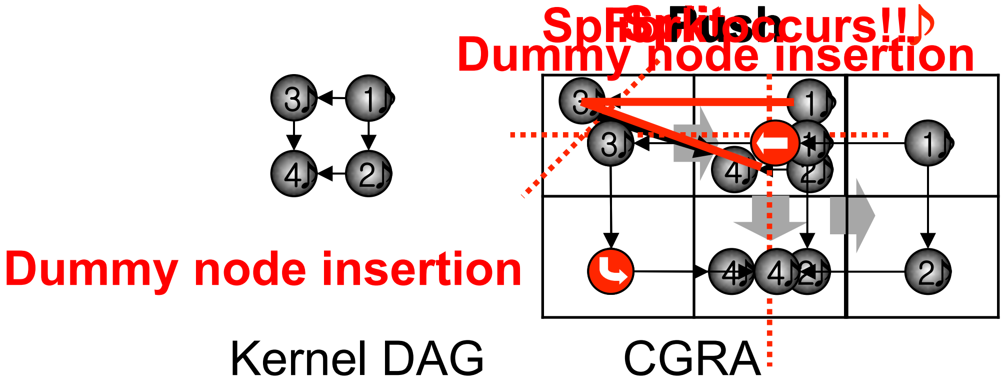
- PEs have to perform the same operation every cycle
  - All vertices must be mapped to PEs, otherwise mapping not possible
  - Implies:
    - All operations must fit in the CGRA
    - Utilization may be low, if the loop is not parallel.
- Main challenge: Minimize Routing
  - Reduces utilization of the CGRA

Graph Drawing Problem

- Split & Push Algorithm[1]



Good Mapping



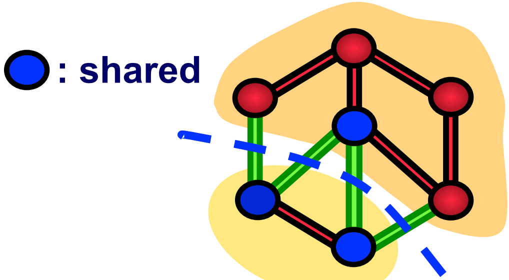
Bad Mapping

- □ Bad split decision incurs more uses of resources
  - □ 2 vs. 3 columns
- □ Forks happen
  - ▣ When adjacent edges are cut by a split
  - ▣ Forks incurs dummy nodes, which are '*unnecessary routing PEs*'
- □ How to reduce forks?

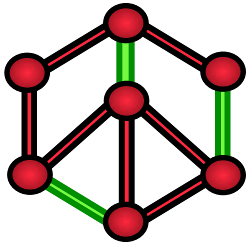
[1] *G. D. Battista et. al. A split & push approach to 3D orthogonal drawing. In Graph Drawing, 1998.*

Graph Drawing Problem

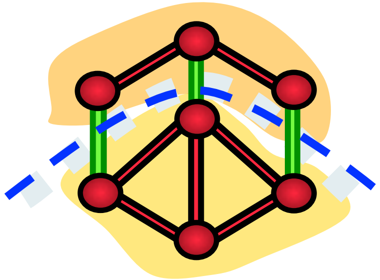
- Matching-Cut[2]
  - Matching: A set of edges which do not share nodes
  - Cut: A set of edges whose removal makes the graph disconnected



A cut, but not a matching

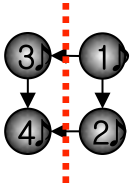


A matching, but not a cut

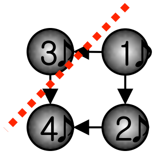


A matching-cut

- Forks can be avoided by finding matching-cut in DAG



A matching-cut, need 4 PEs, no routing PEs

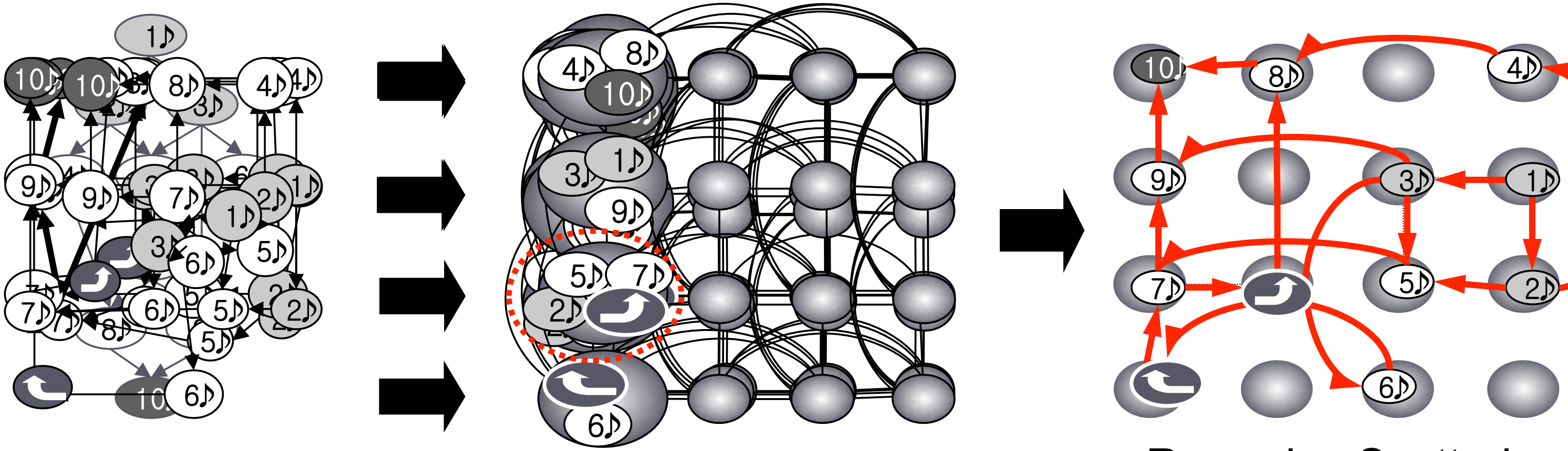


A cut, need 6 PEs, 2 routing PEs

[2] *M. Patrignani and M. Pizzonia. The complexity of the matching-cut problem. In WG '01: Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science, 2001.*

Split & Push Kernel Mapping[3]

- □ PE is connected to at most 6 other PEs.
- □ At most 2 load operations and one store Operation can be scheduled.
  - Load : ◯ Store : ● ALU : ○ RPE : ⤴ Fork : →
- □ # of node : |V| = 10
- □ # of load : L = 3
- □ # of store : S = 1
- □ Initial ROW$_{min}$ = $MAX(\lceil V/N \rceil, \lceil L/L_r \rceil, \lceil S/S_r \rceil)$ = $MAX(\lceil 10/4 \rceil, \lceil 3/2 \rceil, \lceil 1/1 \rceil)$ = 3



[illegible]

Split & Push [illegible] Initial Position

Row-wise Scattering

[3] *Split-Push Kernel Mapping* – ***Best Paper Candidate***, *Design Automation Conference, Asia South Pacific, 2008*

Best Spatial Mapping Algorithm to date

Outline of this Presentation

- Power-efficiency is the design metric
  - CGRAs offer a extremely power-efficient and high performance computing platform
  - Several compiler challenges remain

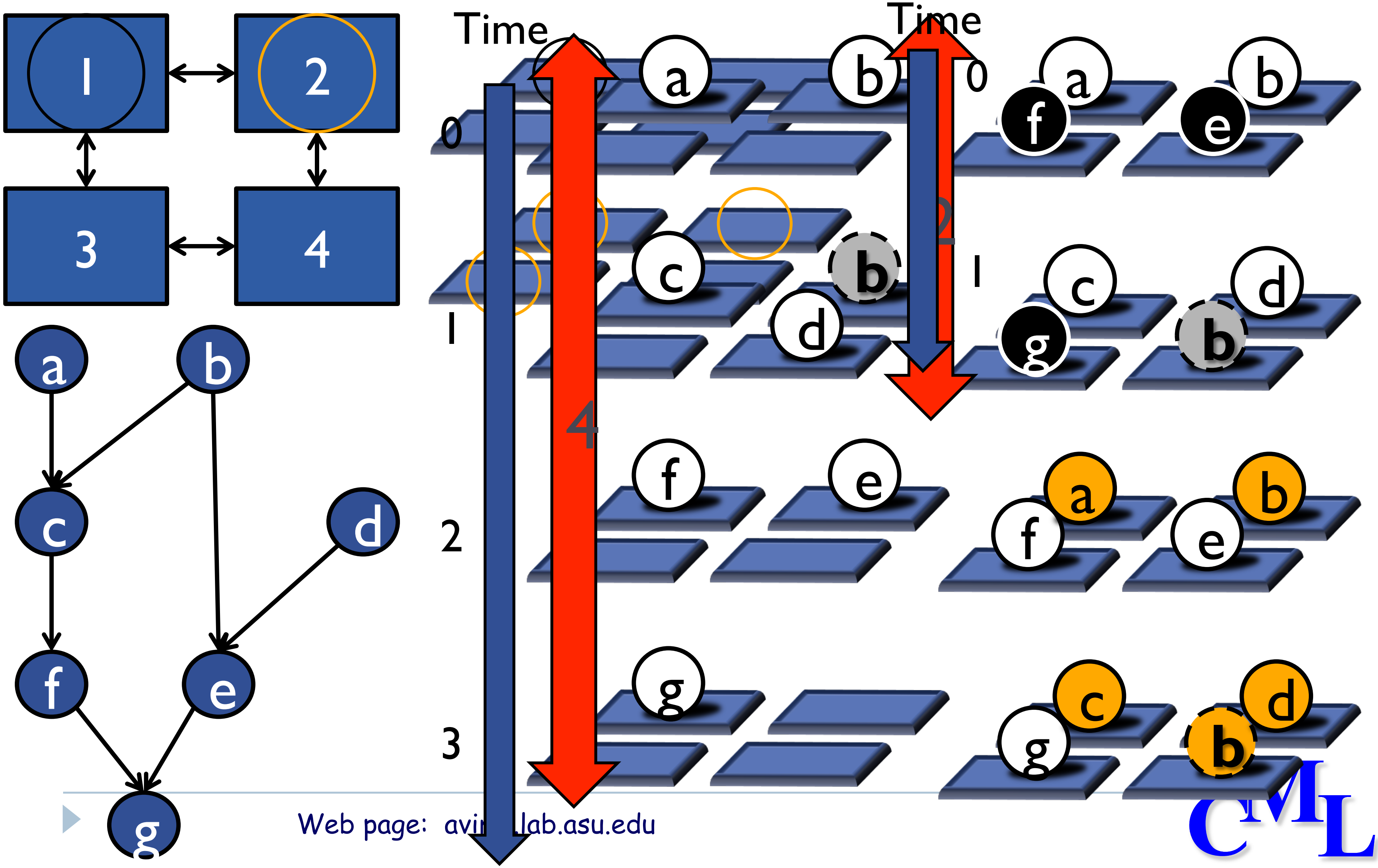
- Presented
  1. Spatial Mapping on a CGRA
  2. General Problem Definition
     - **Models both routing and re-computation**
     - **EpiMAP: New State-of-the-Art**
  3. Enable Multi-threading on CGRAs
     1. **Runtime shrinking the schedule**

- More Work on
  - Memory-aware mapping on CGRAs
  - Bank conflicts and interleaving

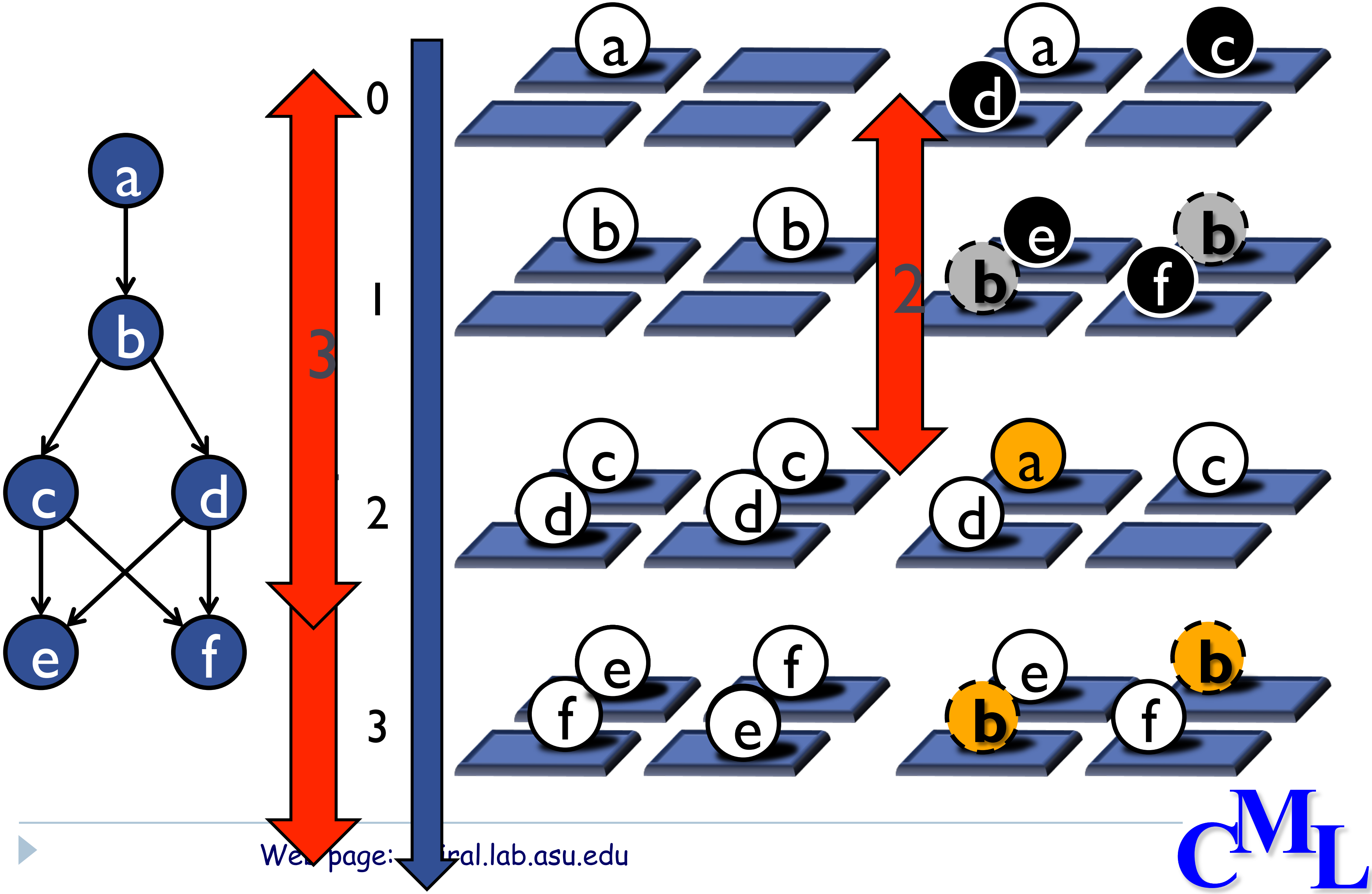
No General Problem Definition

- Existing Problem Definitions are restrictive
  - Inputs: 1. DDG (vertices, and dependencies)
    2. CGRA (PEs and edges)
  - Output: Map1: Vertices to PEs
    Map2: dependencies to paths in CGRA
- Model routing
  - Can map dependencies to paths
- Do not model re-computation

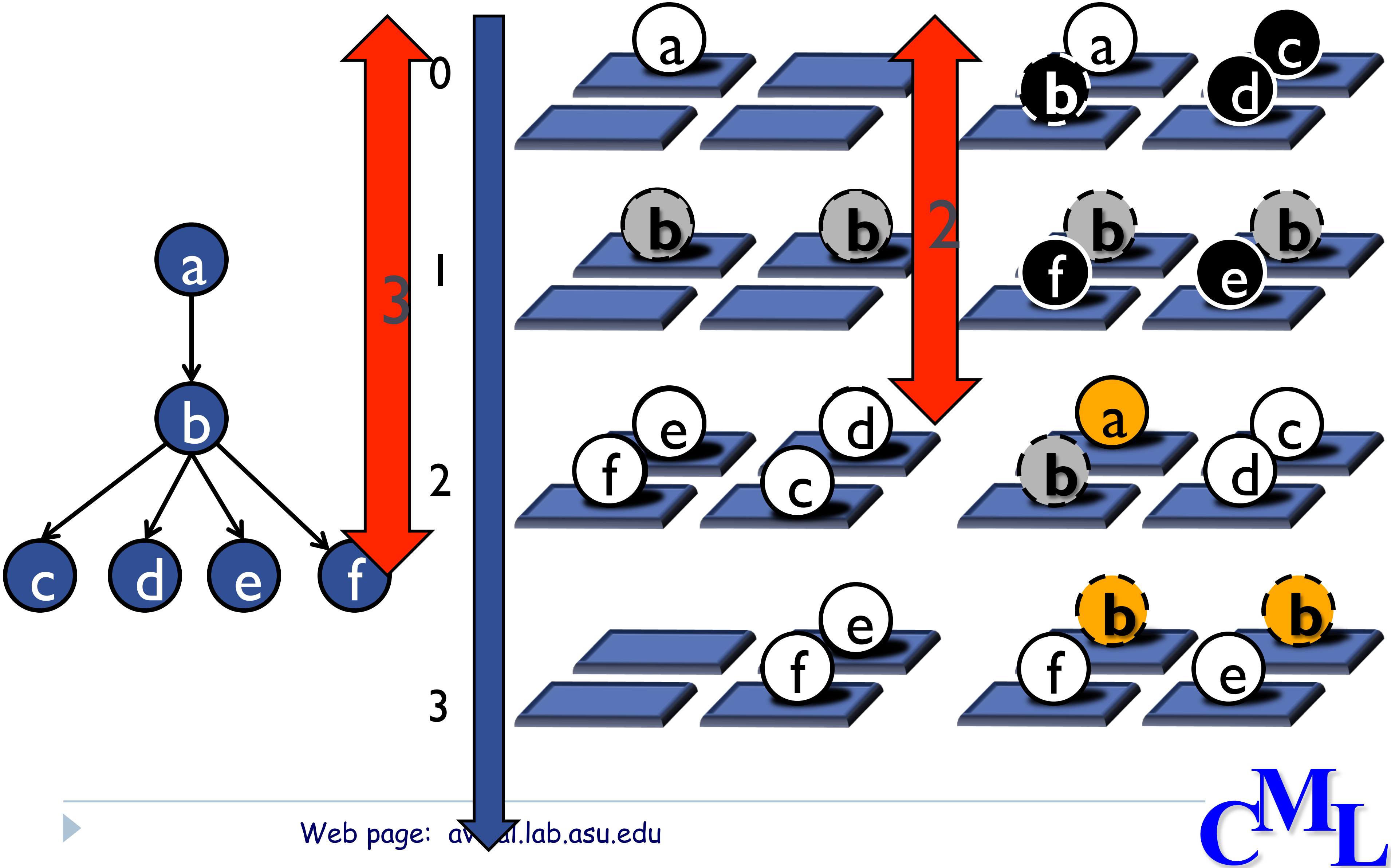
How to Map a Kernel onto CGRA?



What is Re-computation?



Re-computation and Routing



Valid Mapping*

DEFINITION 2. *Valid Mapping: Let n be the number of nodes in V_d, i.e. $n = |V_d|$, and $C = (V_c, E_c)$ be TEC. Let $C^* = (V_{c^*}, E_{c^*})$ be a subset of C, i.e., $C^* \subseteq C$. Let $S = \{s_1, s_2, s_3, ..., s_n\}$ be a set of n disjoint subsets of V_{c^*} such that $1 \leq \forall i \leq n, |s_i| \geq 1$. A mapping function $f : V_d \rightarrow S$ is a valid mapping iff $\forall u, v \in V_d$: $(u, v) \in E_d$,*

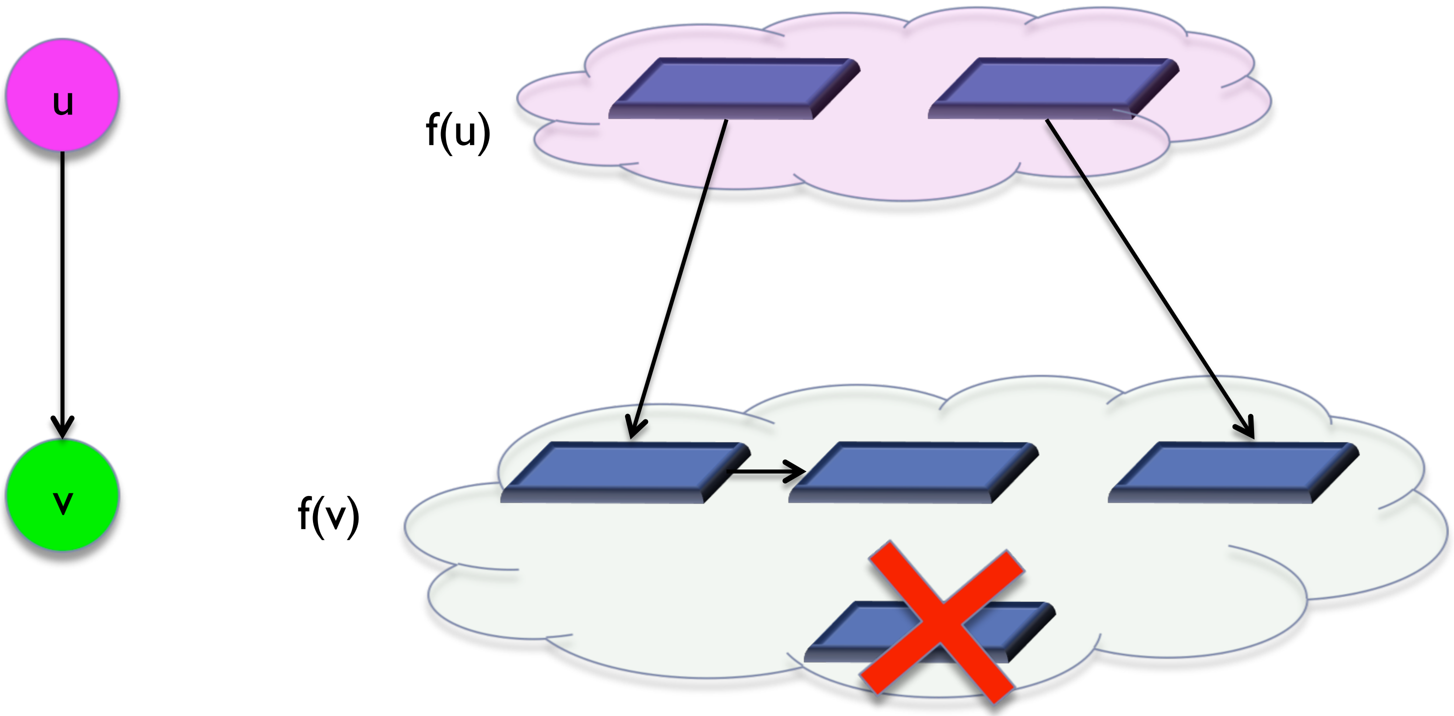
$\forall v' \in f(v)$, there must be a path from a node $u' \in f(u)$ to v'. The nodes in this path must only be nodes in $f(v)$.

Informal Definition

- Inputs: 1. DDG (vertices, and dependencies)
  2. Time Extended CGRA (PEs and edges)
- Output: Function from vertices to disjoint subsets of PEs of CGRA, such that, if there is an edge between two vertices, then there is an edge between corresponding subsets.

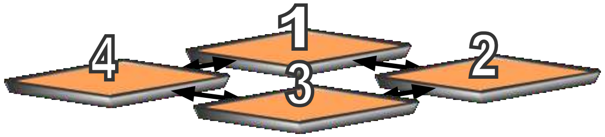
*EPIMAP: Using Epimorphism to map applications onto CGRAs, Design Automation Conference 2012

General Problem Definition

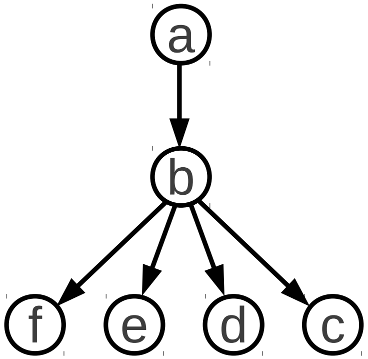


- All nodes in f(v) must have a path from any node in f(u)
  - Otherwise invalid mapping
  - Re-computing: when direct path from some node in f(u)
  - Routing: when path through other nodes in f(v)

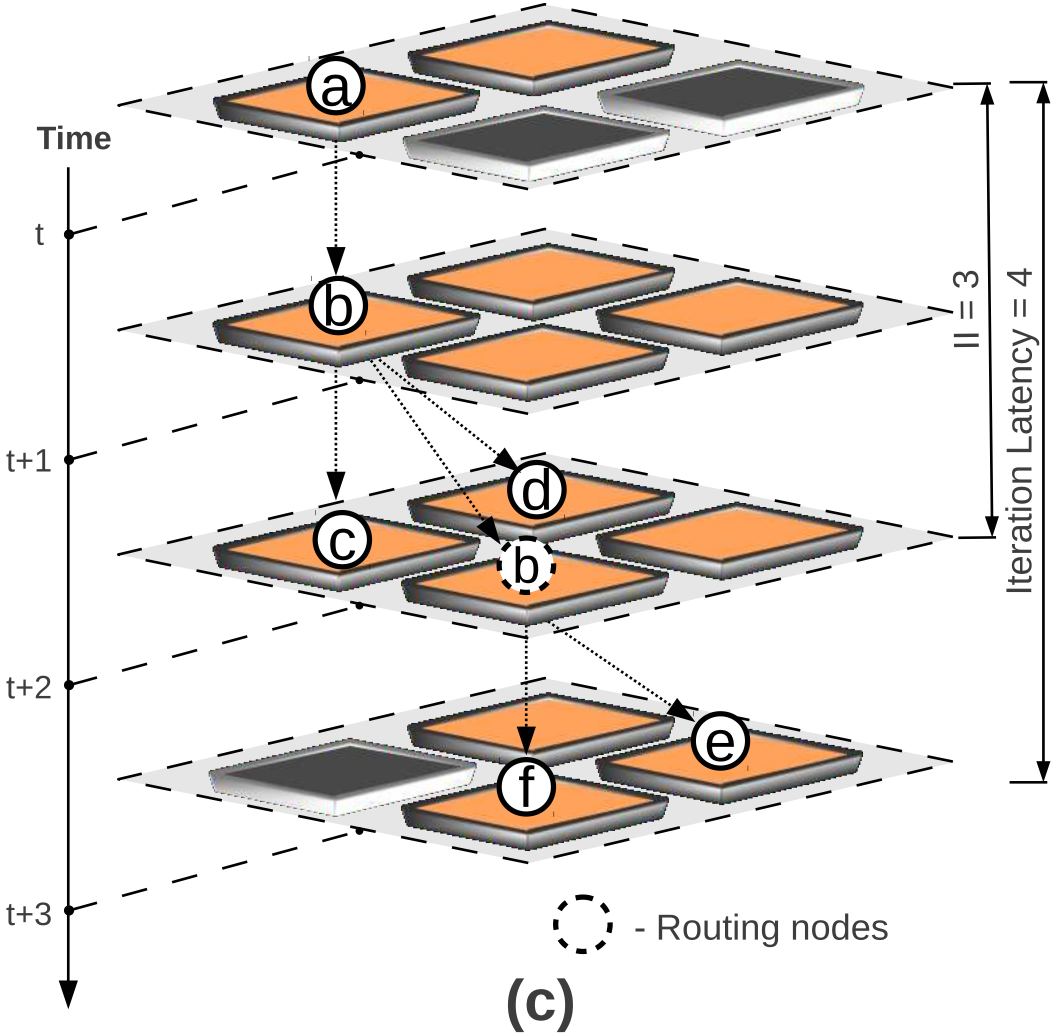
Problem Definition - Routing



(a)

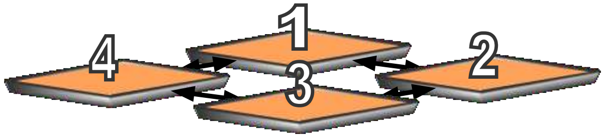


(b)

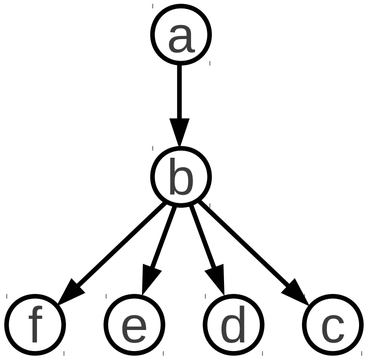


(c)

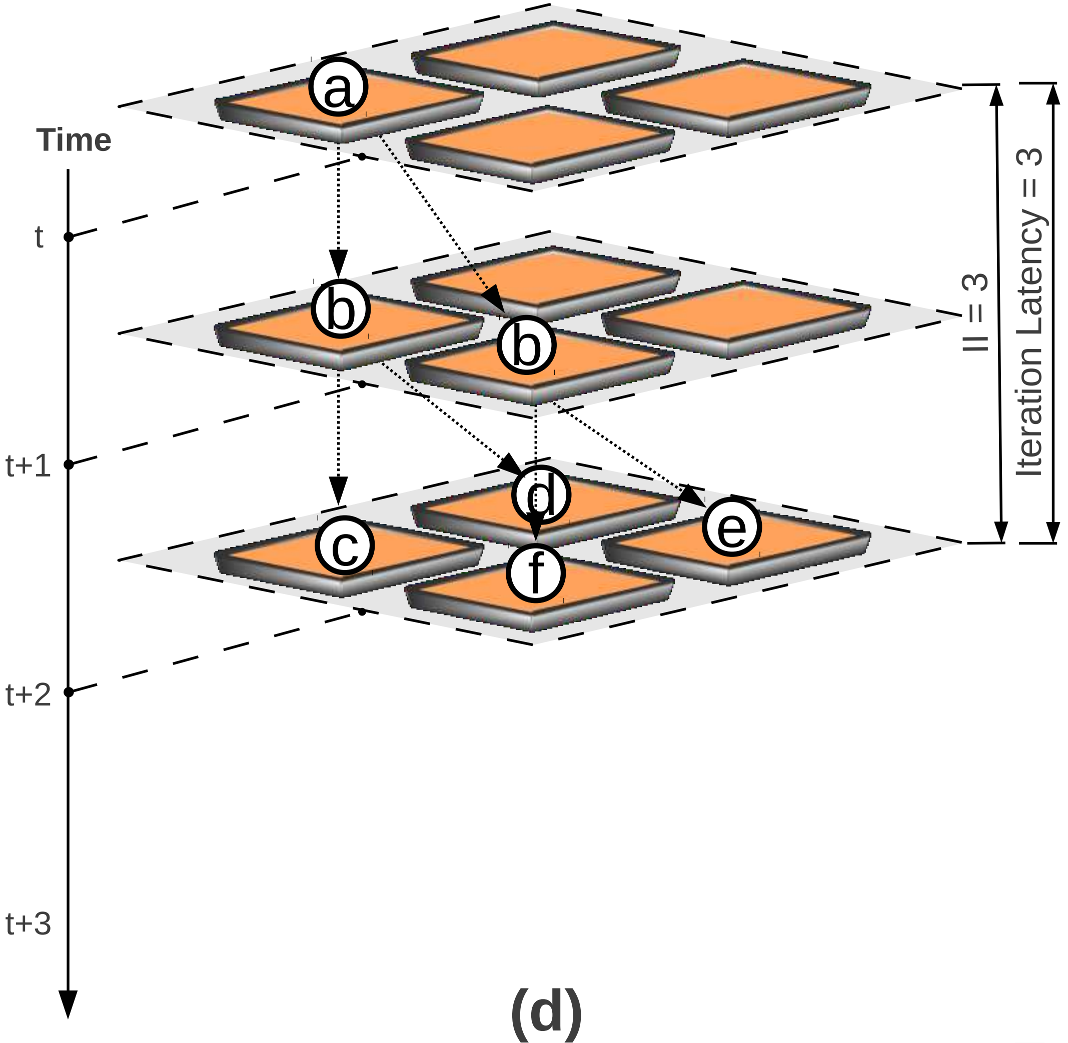
Problem Definition – Re-computation



(a)

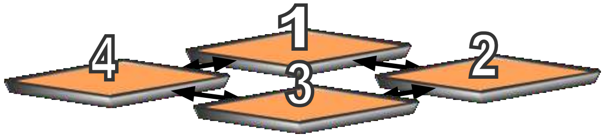


(b)

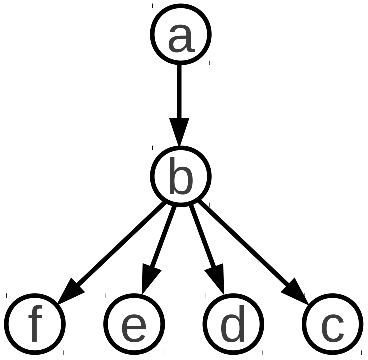


(d)

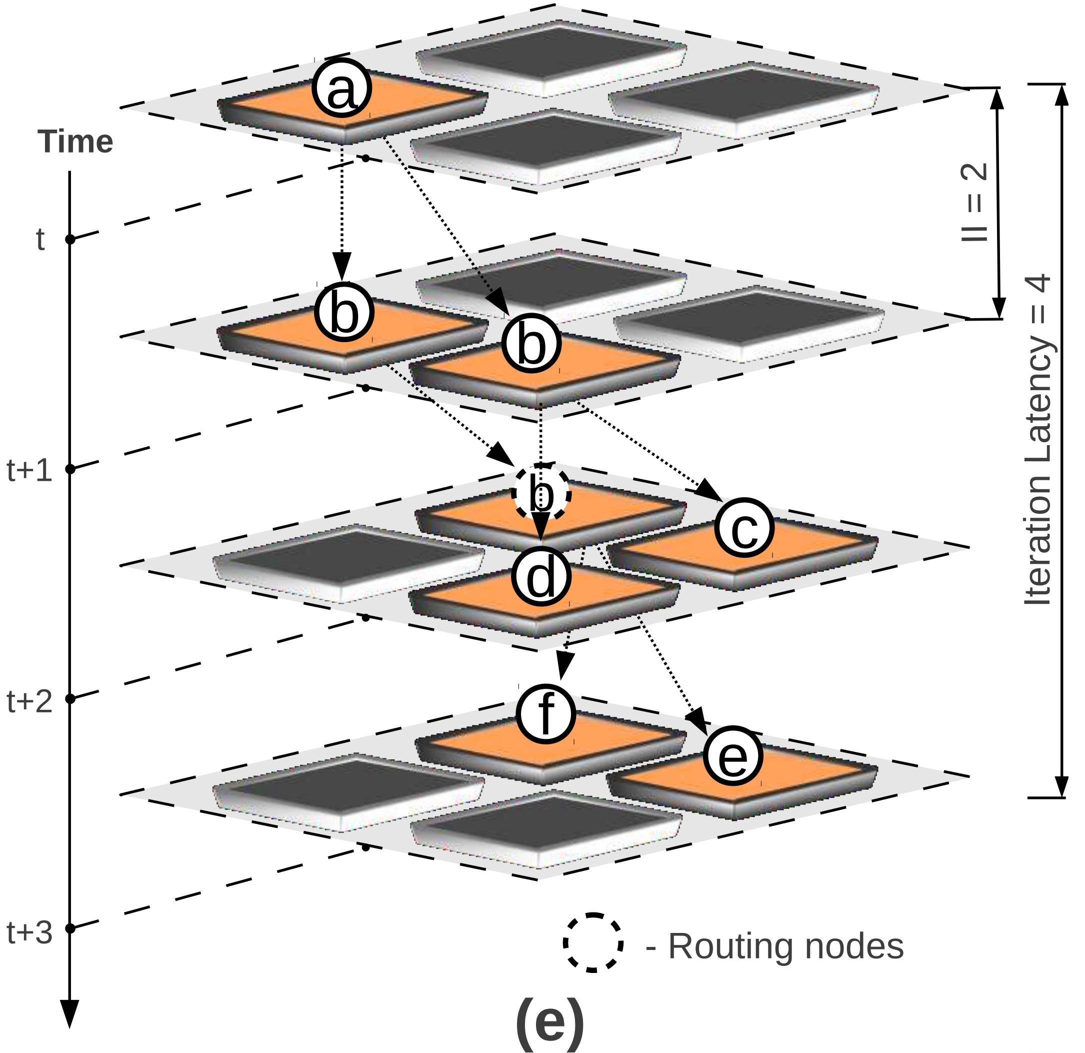
Problem Definition – Routing and Re-computation



(a)

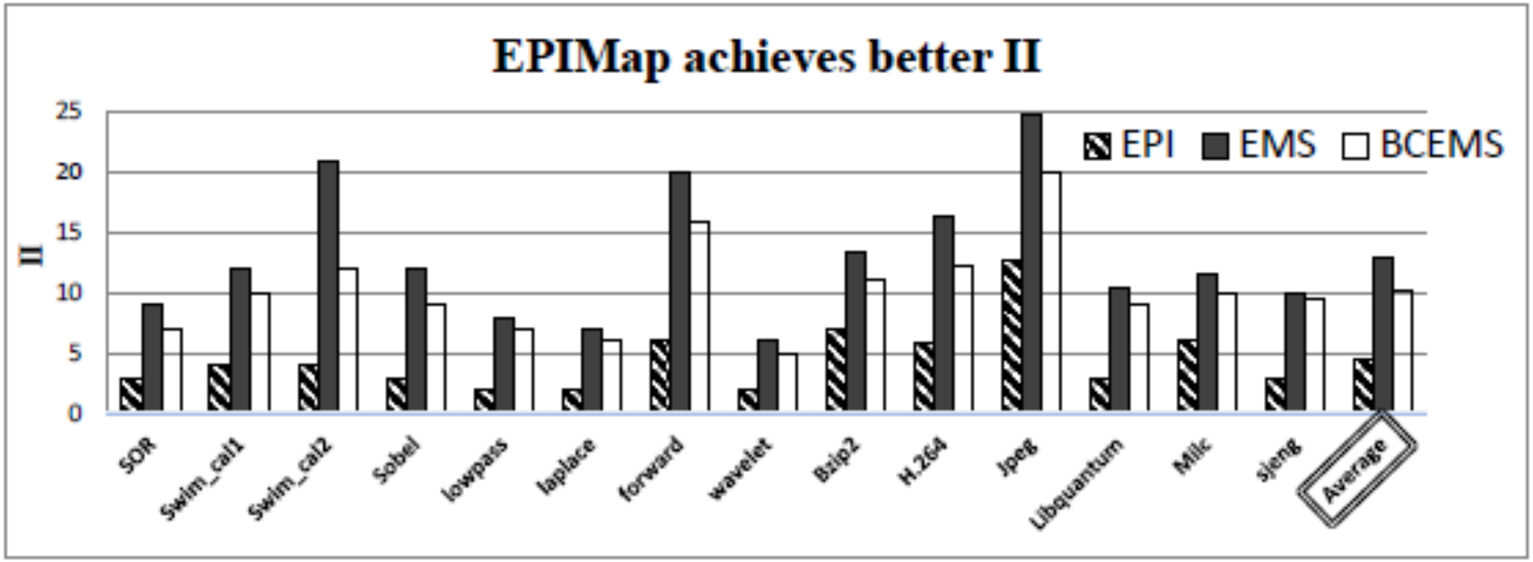


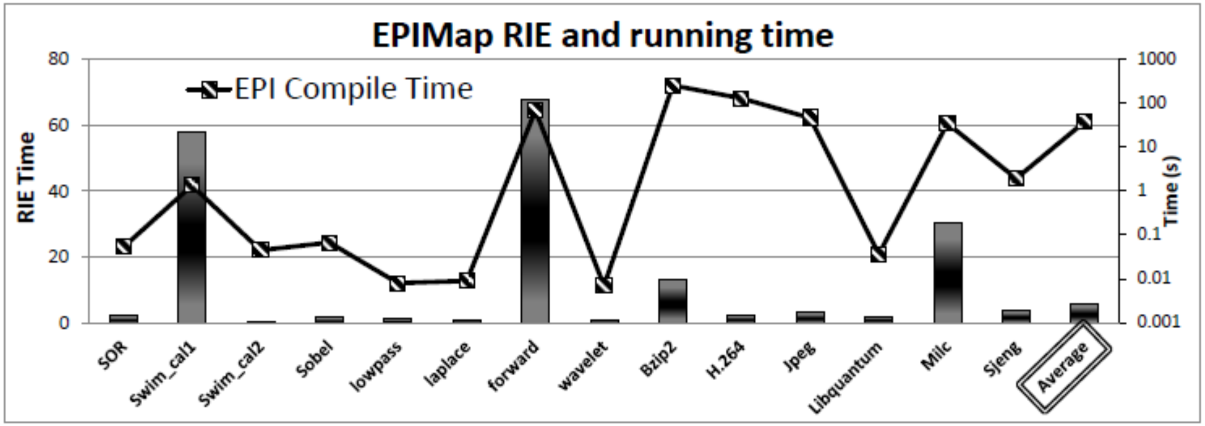
(b)



(e)

EpiMap : 2.5X better mapping





Outline of this Presentation

- Power-efficiency is the design metric
  - CGRAs offer a extremely power-efficient and high performance computing platform
  - Several compiler challenges remain

- Presented
  1. Spatial Mapping on a CGRA
  2. General Problem Definition
     - Models both routing and re-computation
     - EpiMAP: New State-of-the-Art
  3. Enable Multi-threading on CGRAs
     1. Runtime shrinking the schedule

- More Work on
  - Memory-aware mapping on CGRAs
  - Bank conflicts and interleaving

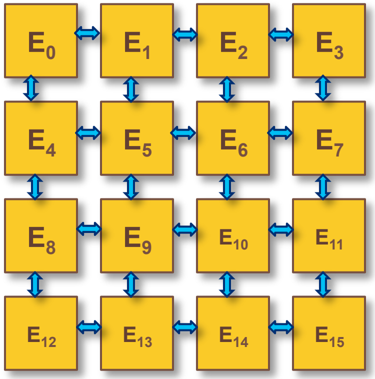
State of the Art

- Single threaded acceleration
  - One kernel of one of the threads can be accelerated at any given moment
  - Entire CGRA used to schedule each kernel of the thread
- While cores are multi-threaded
- If multiple threads require simu
  - threads must be stalled
  - kernels are queued to run on the C

Existing CGRA compilers can be used for efficient single-threaded use

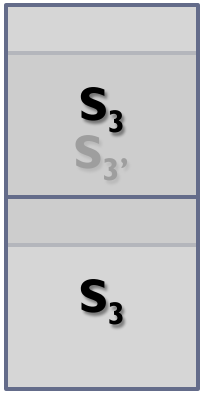
Not all PEs are used in each schedule.

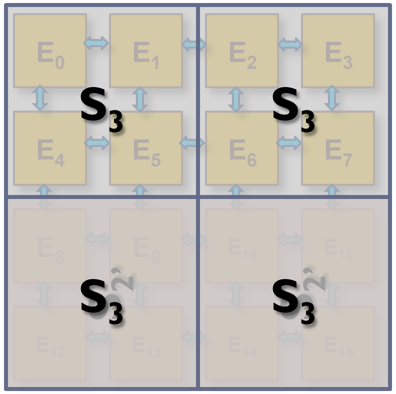
Thread-stalls create a performance bottleneck



Multi-threading on CGRA

- Facilitate multiple kernel to execute on the CGRA simultaneously
- Cannot re-compile
  - Compilation for CGRA is hard
  - Existing algorithms take long time to find a good mapping
  - Use searches for binding and routing





Thread: 3
Schedule Expanded to use whole CGRA and increase performance

Our Multithreading Technique

1. Static compile-time constraints to enable schedule transformations
   - Has minimal effect on overall performance (*II*)
   - May increase compile-time

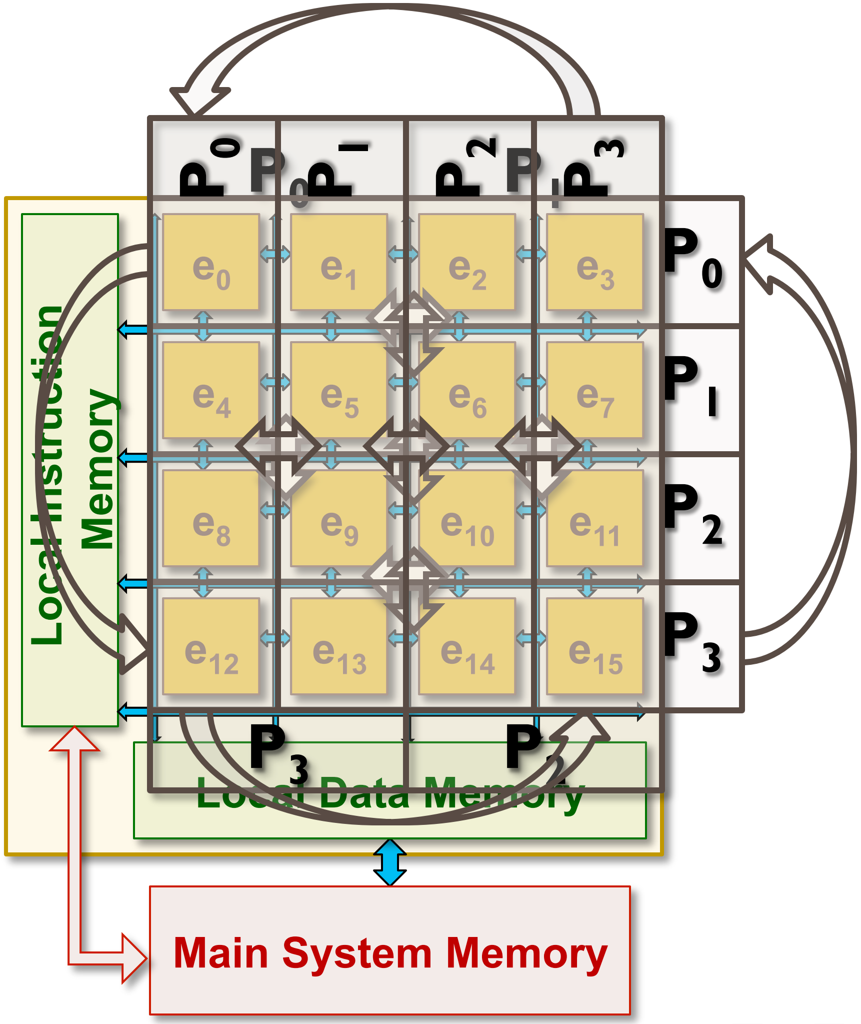
2. Fast runtime transformations
   - Linear time to complete
   - All schedules treated independently

Features:

- Runtime Multithreading enabled in linear runtime
- No additional hardware modifications
- Works with current CGRA mapping algorithms
  - Algorithm must allow for custom PE interconnects
  - Experimentally demonstrated using EMS

Minimal Compiler Constraints

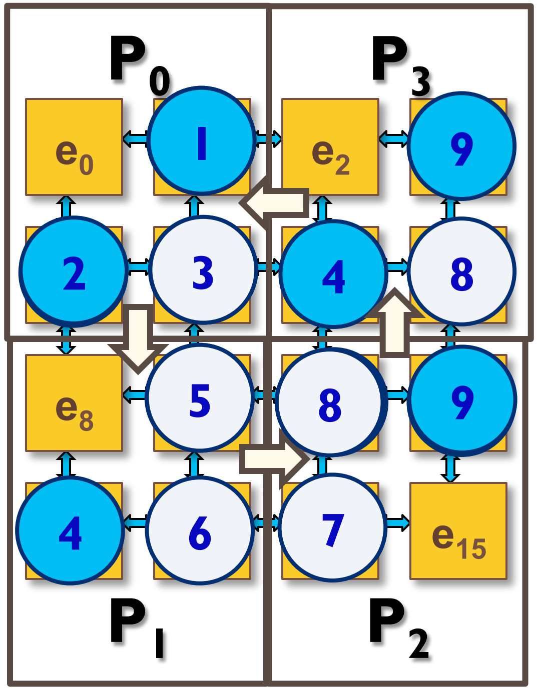
- **Page:** software perspective grouping of PEs
- A page has **symmetrical connections** to each of the neighboring pages
- No additional hardware 'modification' is required.
- Page-level interconnects follow a **ring topology**
  - **Clock-wise**
  - **(or) counter clock-wise**

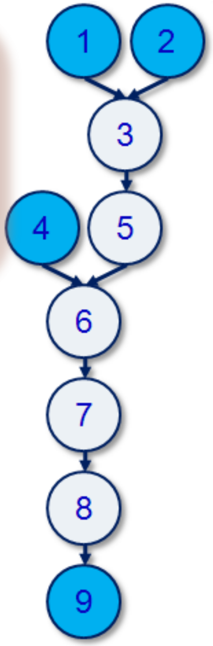


Mapping Kernel onto Pages

- Compile-time Constraints
  - CGRA is collection of pages
  - Each page can interact with only one topologically neighboring page.
  - Inter-PE connections within a page are unmodified
  - Data flow of kernel is maintained across pages through topological assignment of page schedules

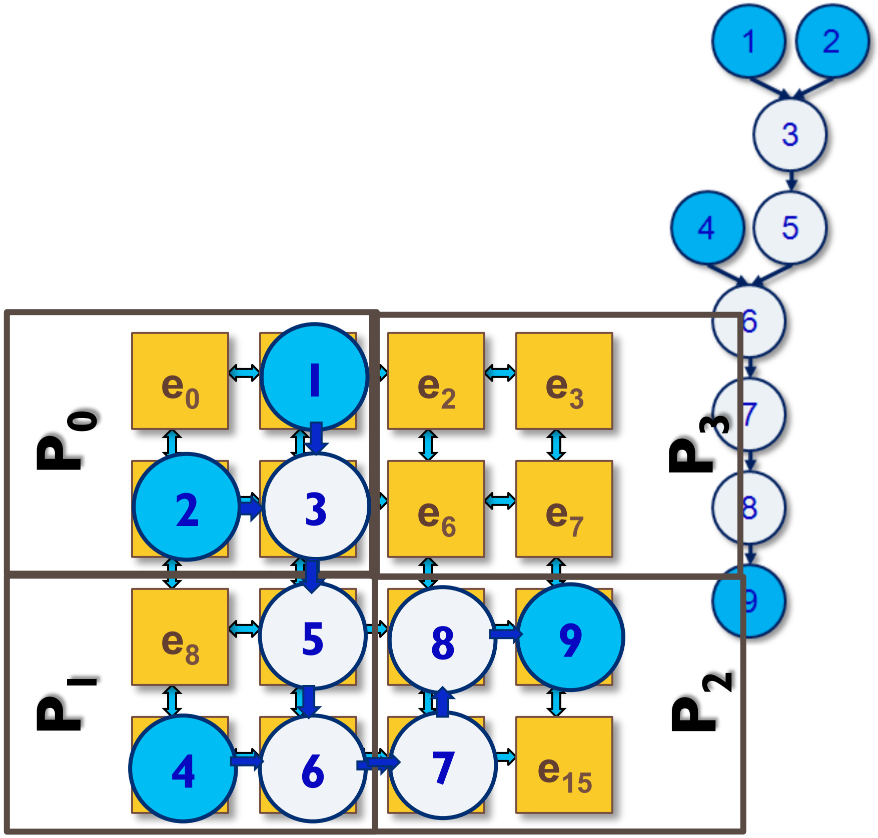
Our paging methodology, helps reduce CGRA resource usage & reduce power consumed





Shrinking schedule at runtime

- Example:
  - application mapped to 3 pages
  - Shrink to execute on 2 pages
- Transformation Procedure:
  1. Isolate the mapped schedule
  2. Split pages in topological order

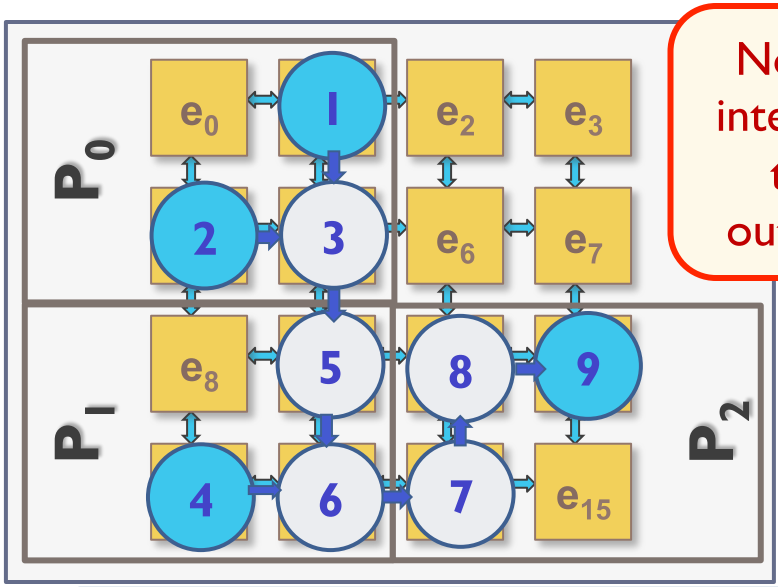


- Constraints
  - inter-page dependencies should be maintained at all instances

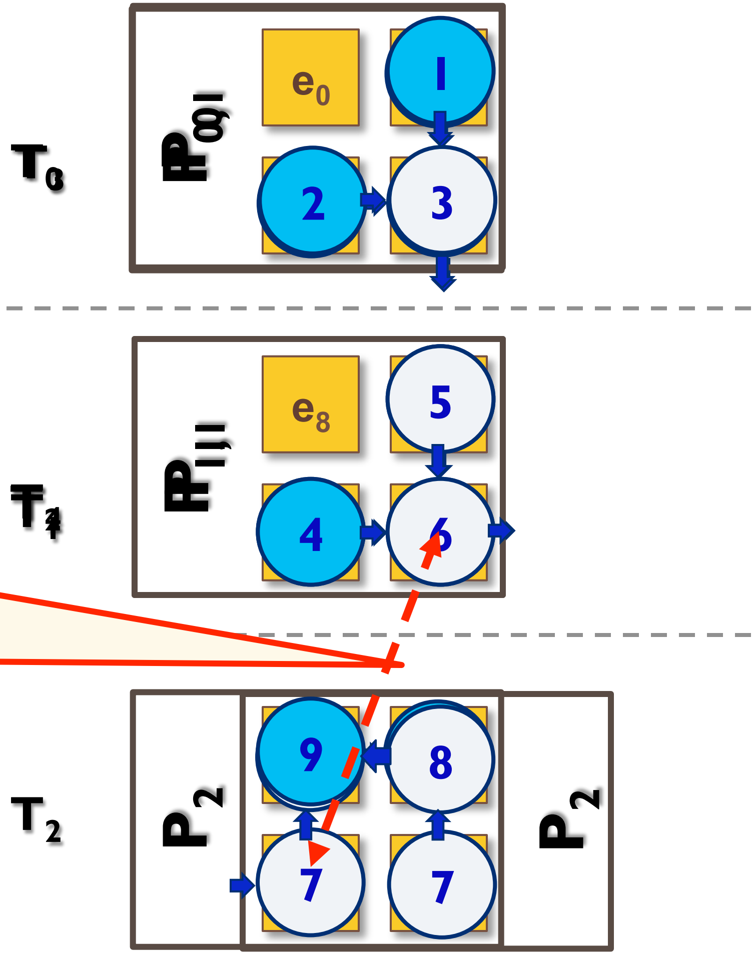
Shrinking schedule at runtime

- Transformation Procedure:
  1. Isolate the mapped schedule
  2. Split pages in topological order
  3. Executed schedule on modified time-schedules (only 2 pages)
  4. Mirror pages to facilitate shrinking

     (To ensure inter-node dependency)



No CGRA interconnect to feed output to **7**



Experiments*

- 1. Compiler constraints are not too restrictive
  - Do not degrade original mapping

- 2. Multithreading can improve performance
  - Loops cannot use entire CGRA
  - Mapping to pages improves utilization

*Enabling Multi-threading on CGRAs," in International Conference on Parallel Processing, ICPP 2011

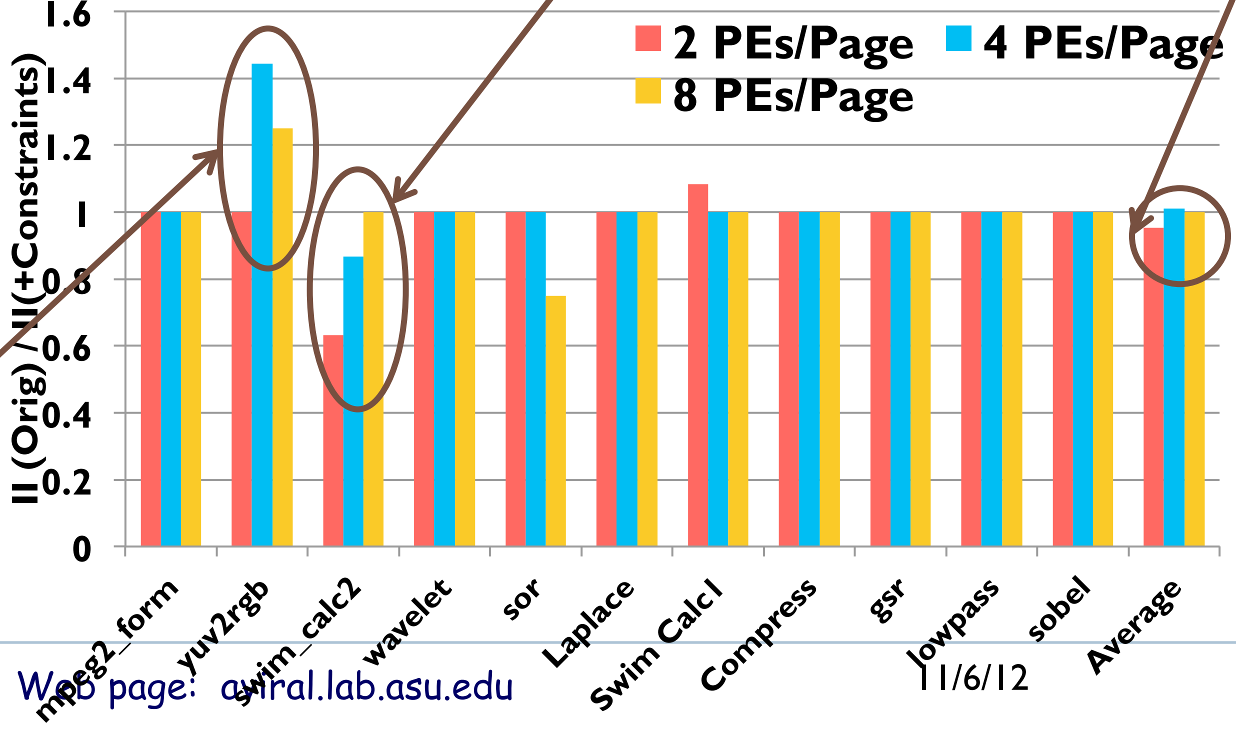
Compiler Constraints are Liberal

- Compile kernels for the CGRA
  - EMS Without constraints
  - EMS + our compiler constraints
- Mapping quality measured in Iteration Intervals
  - smaller II is better

Constraints can degrade individual benchmark performance by limiting compiler search space

On average, performance is minimally impacted

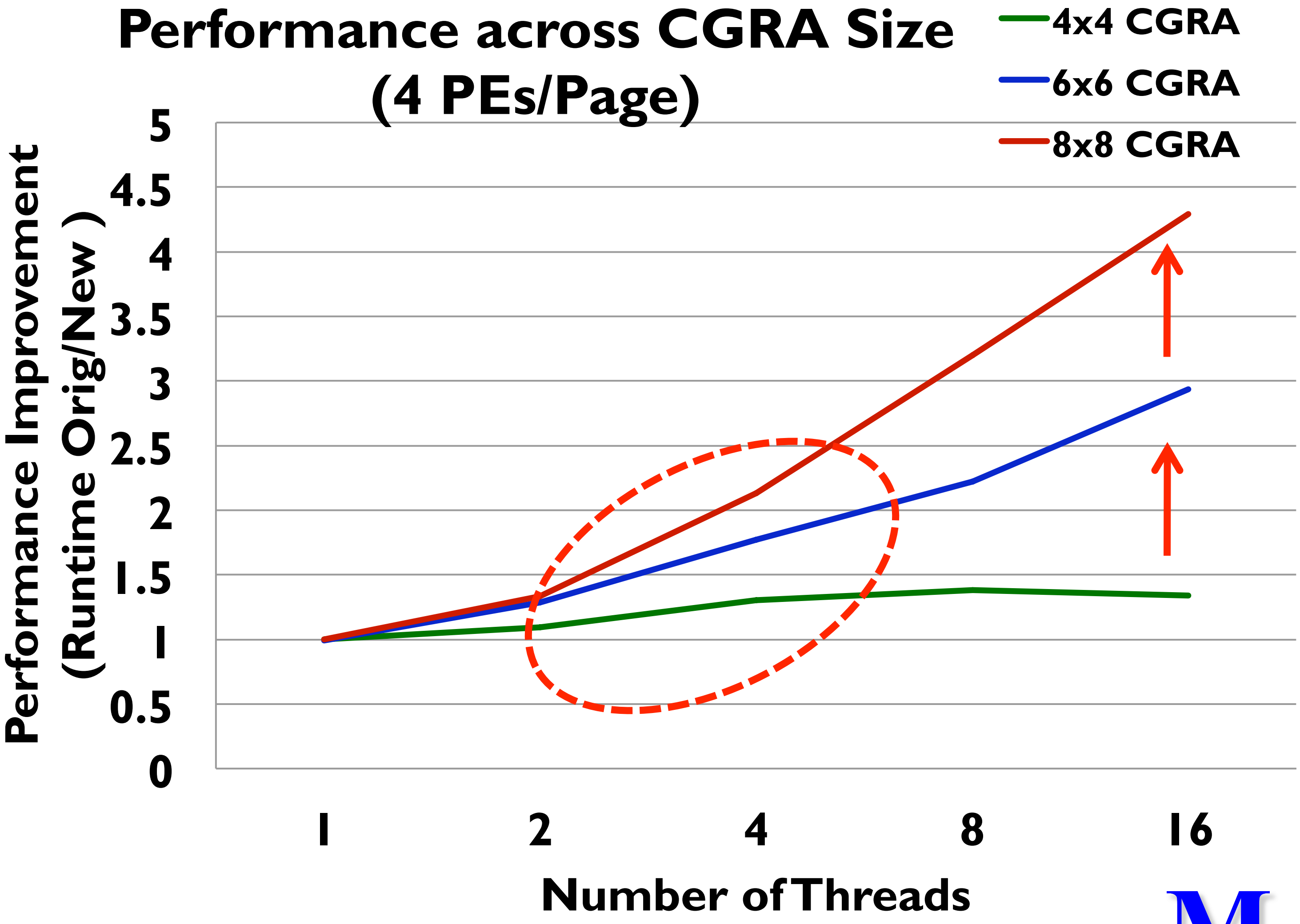
Ironically, the same can also improve individual benchmark performance



Multi-threading → Better Performance

Number of Threads Accessing CGRA: Throughput improves as the number of threads increases

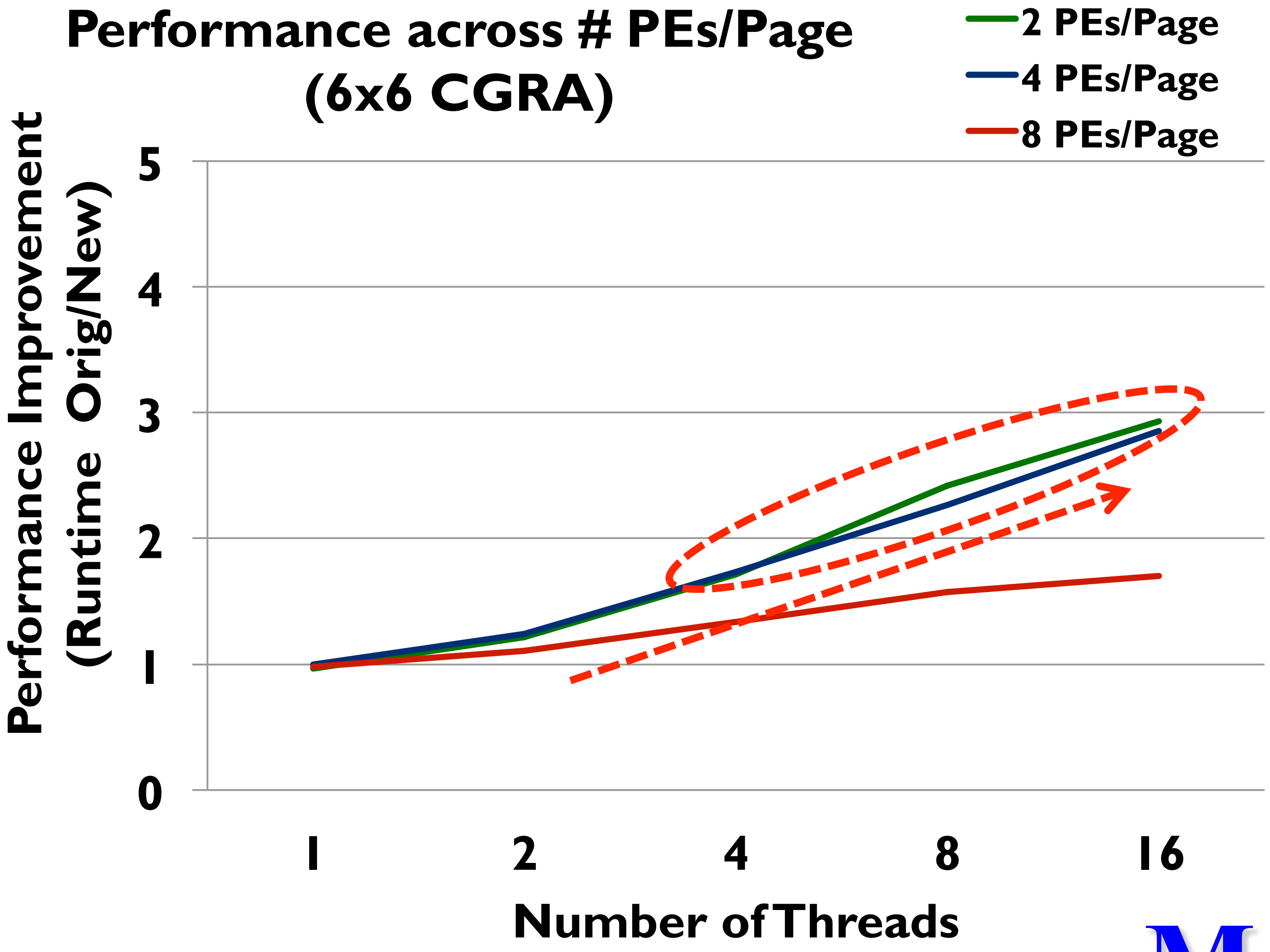
CGRA Size: Utilization and therefore throughput improves as we increase CGRA size



Optimal page size

Increased system performance with more number of threads

Number of PEs per page:
For the set of benchmarks tested, efficient PEs per page is either 2 or 4



Outline of this Presentation

- Power-efficiency is the design metric
  - CGRAs offer a extremely power-efficient and high performance computing platform
  - Several compiler challenges remain

- Presented
  1. Spatial Mapping on a CGRA
  2. General Problem Definition
     - **Models both routing and re-computation**
     - **EpiMAP: New State-of-the-Art**
  3. Enable Multi-threading on CGRAs
     1. **Runtime shrinking the schedule**

- More Work on
  - Memory-aware mapping on CGRAs
  - Bank conflicts and interleaving

Backup

Summary

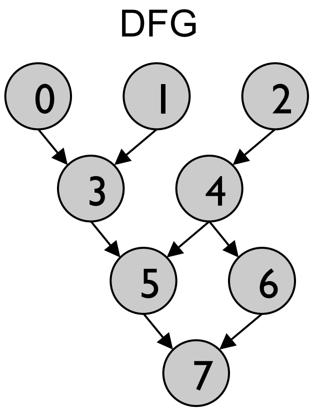
- Power-efficiency is the chief bottleneck for higher performance
- Multi-cores only go so far
  - Time for accelerators has come!
- CGRAs as accelerators
  - State of the art – Single thre
  - Cores are multi-threaded – re to be scheduled on CGRA

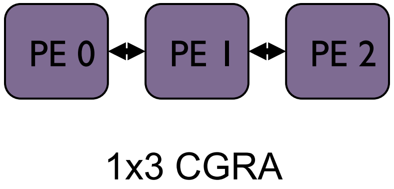
More details in our paper on “

- Propose a two-step methodology
  - Minimally-restrictive compile-time constraints
  - Scheme to quickly shrink schedule at runtime
- Features:
  - No additional hardware required
  - Improved CGRA resource usage
  - Improved system performance

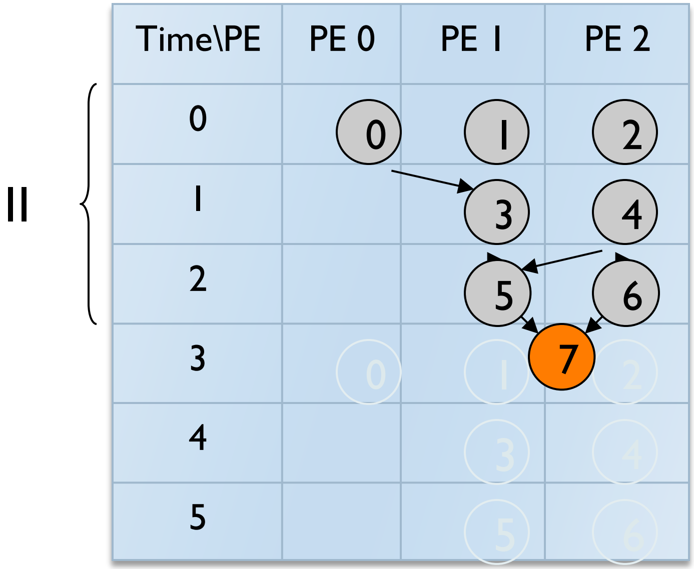
Existing Mapping Approaches

- Most heuristics work like this:
  - Map a node to the CGRA
  - then find out where it's dependents can be mapped
- Heuristics differ in
  - Which node to choose?
    - in recurrence cycle[Oh09], confluence[Park06], in critical path[Lee03], choose edges [Park08]
  - How to search for a good place to map?
    - Simulated annealing [Hatanaka07, Mei04], nearest neighbors[Park08]





1x3 CGRA



Time\PE	PE 0	PE 1	PE 2
0	0	1	2
1		3	4
2		5	6
3	0	1	7 2
4		3	4
5		5	6

Formal Problem Definition

Mapping a Kernel onto a CGRA

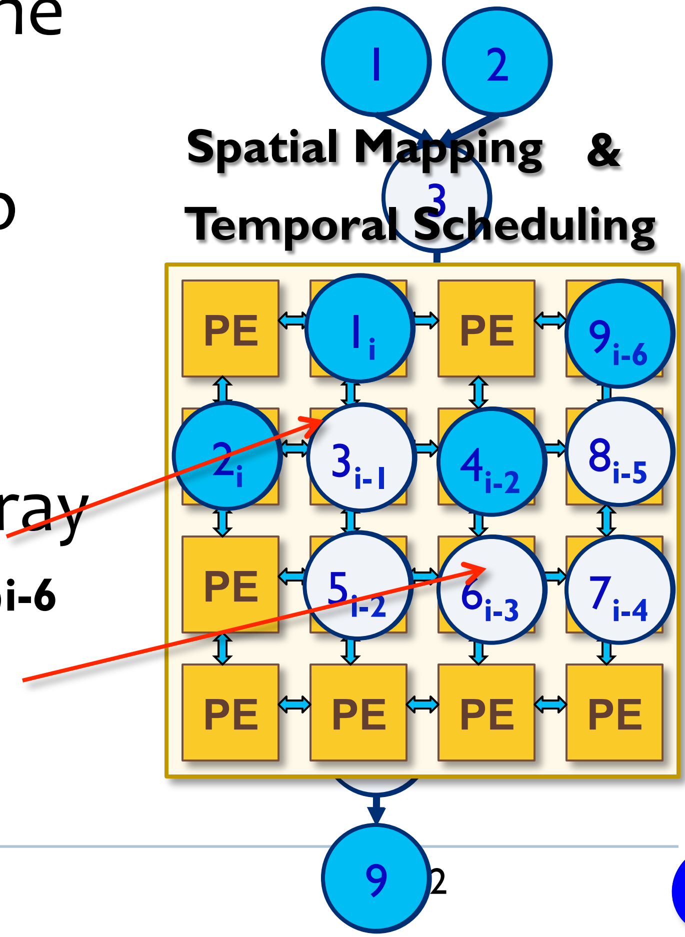
<u>Given the kernel's DDG</u>

1. Unroll and **Software Pipeline** the loop for mapping on the given CGRA
2. **Schedule** the unrolled loop for minimum *II* (=1)
3. Map nodes onto the PE array
   1. Dependent nodes closer to their sources
   2. Ensure dependent nodes have interconnects connecting sources

$1^{i}, 2^{i}, 3^{i-1}, 4^{i-2}, 5^{i-2}, 6^{i-3}, 7^{i-4}, 8^{i-5}, 9^{i-6}$

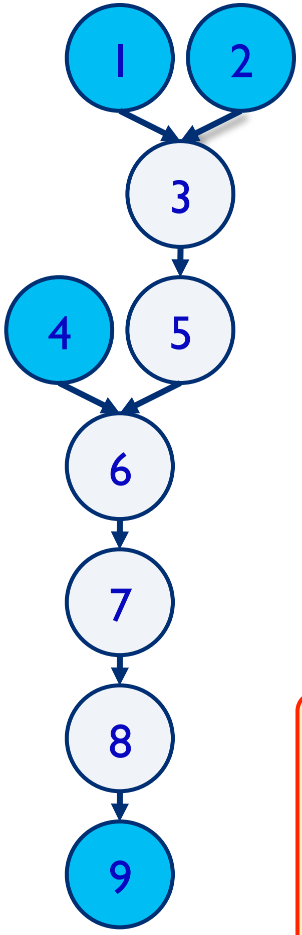
Data-Dependency Graph:

Spatial Mapping & Temporal Scheduling



Mapped Kernel Executed on the CGRA

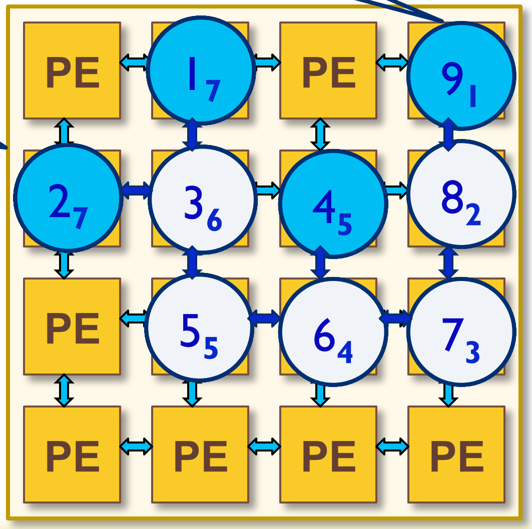
Data-Dependency Graph:



Execution time slot: [illegible]
(or cycle)

After cycle 6, one iteration of loop completes execution every cycle

Entire kernel can be mapped onto CGRA by unrolling **6** times



Iteration Interval (*II*) is a measure of mapping quality

Iteration Interval = 1