Leveraging Parallelism in the Presence of Control Flow on CGRAs

Jihyun Ryoo, Kyuseung Han, and Kiyoung Choi

Design Automation Lab
Department of Electrical and Computer Engineering
Seoul National University

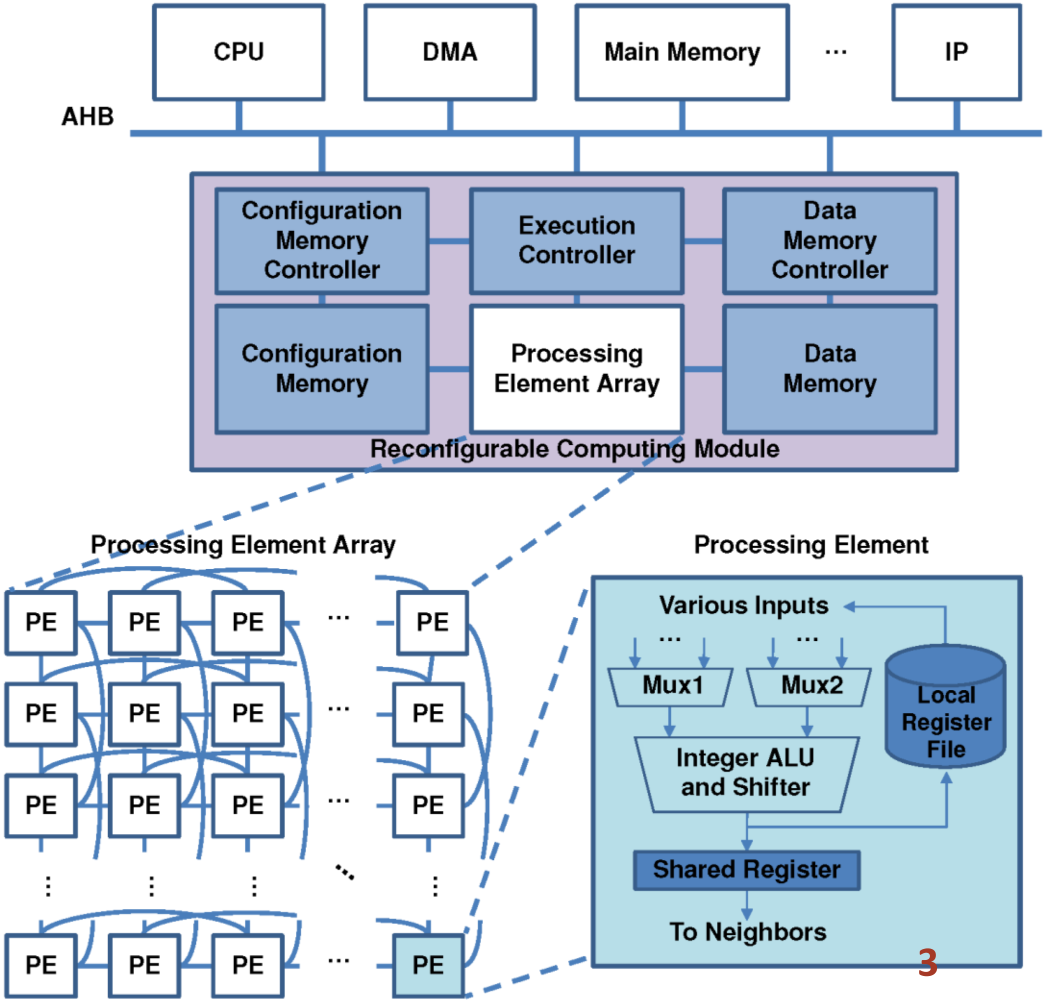
ASP-DAC 2014

Contents

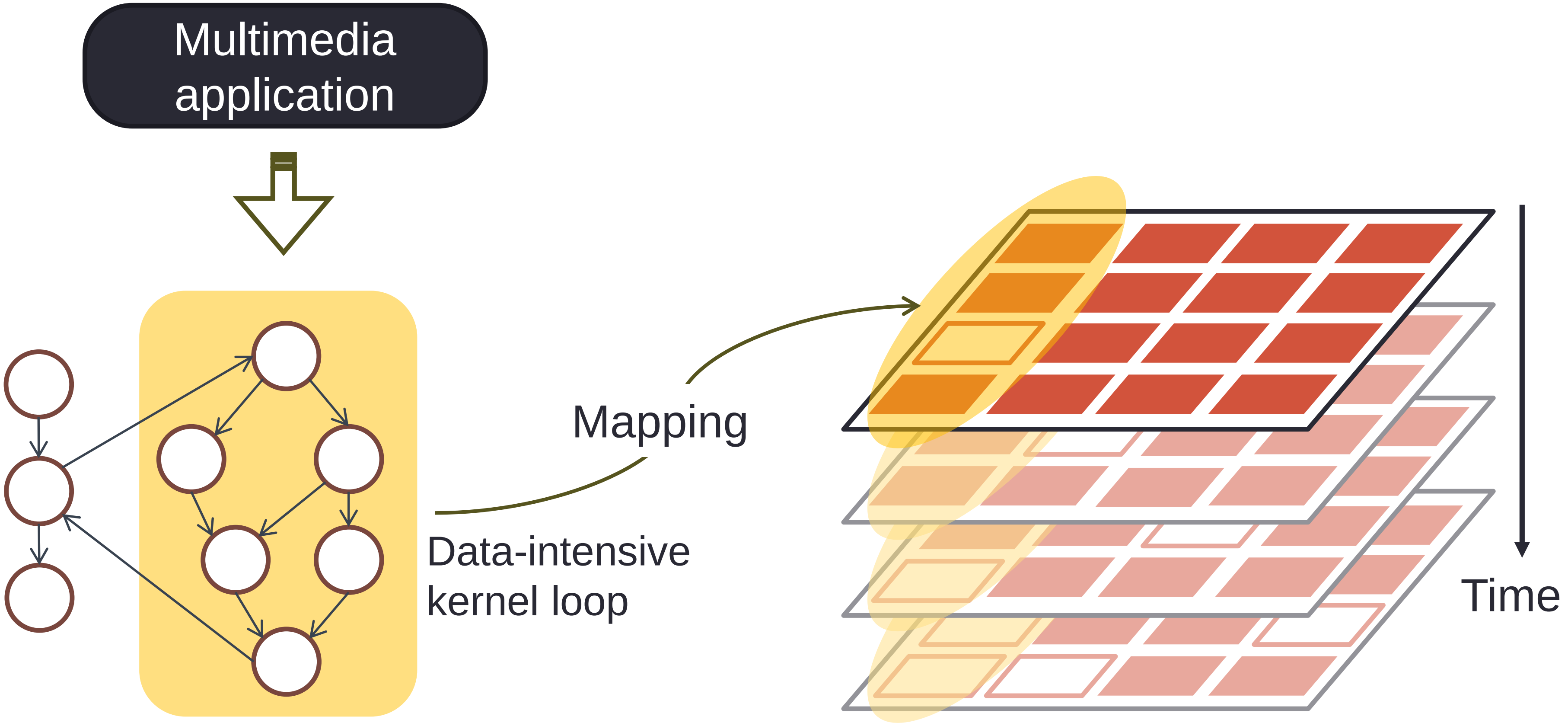
- Introduction
- Control Flows
  - Control flows in SIMD, Handling control flows
- Distributed Register Files
  - Problems of distributed register files, Solutions
- Application Mapping Framework
  - Overall flow, From IR to CDFG, Separation, CDFG Mapping
- Experiments
  - Experimental setup, Results
- Conclusion

CGRA

- Coarse-Grained Reconfigurable Architecture
- FloRA
  - ✓PEs (Processing Elements)
    - ALUs, shifters, register files
    - Word-level granularity
  - ✓Instructions
    - Configuration code
    - Change functionality of PEs and interconnections
    - An instruction may control multiple PEs (SIMD)
- How to compile/map kernel code?



Mapping onto CGRA



Multimedia application

Mapping

Data-intensive kernel loop

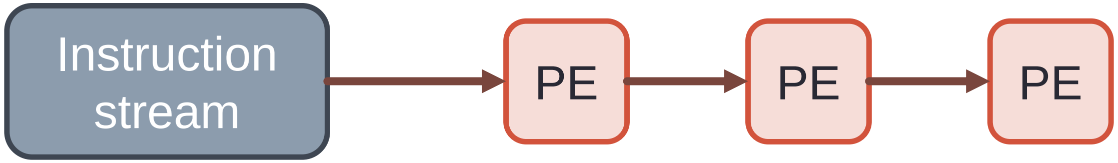
Time

Control Data Flow Graph

CGRA

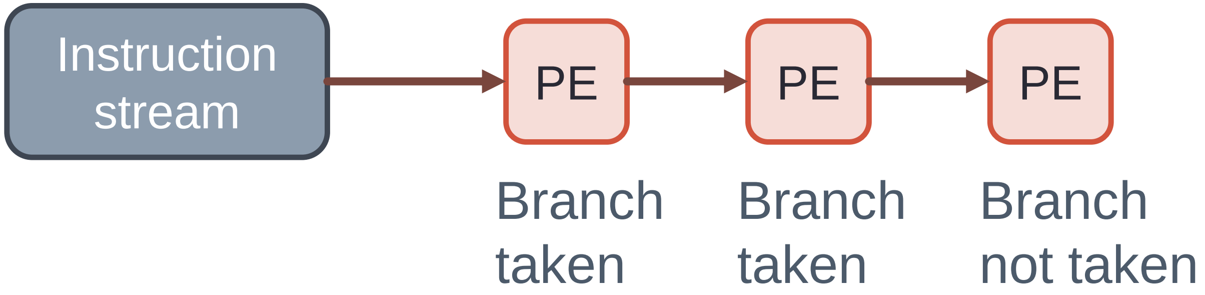
Control Flows in SIMD

- Growing complexity of multimedia application algorithms
- Kernels tend to include more conditional branches
- Limitation of SIMD



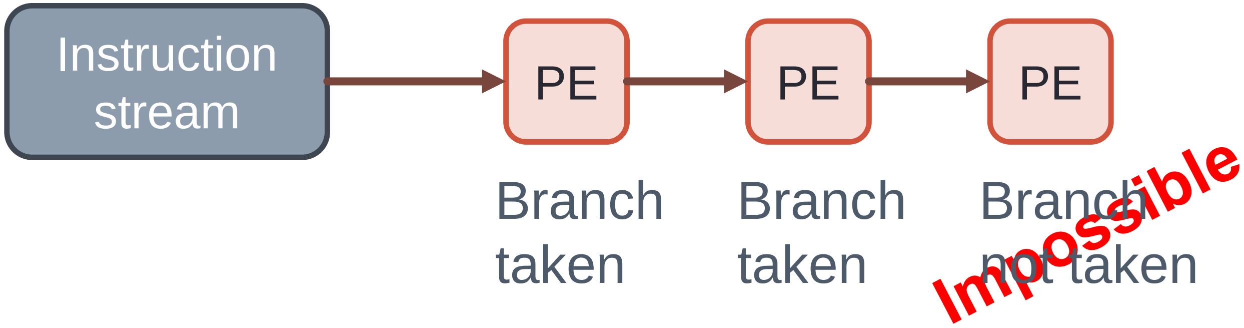
Control Flows in SIMD

- Growing complexity of multimedia application algorithms
- Kernels tend to include more conditional branches
- Limitation of SIMD



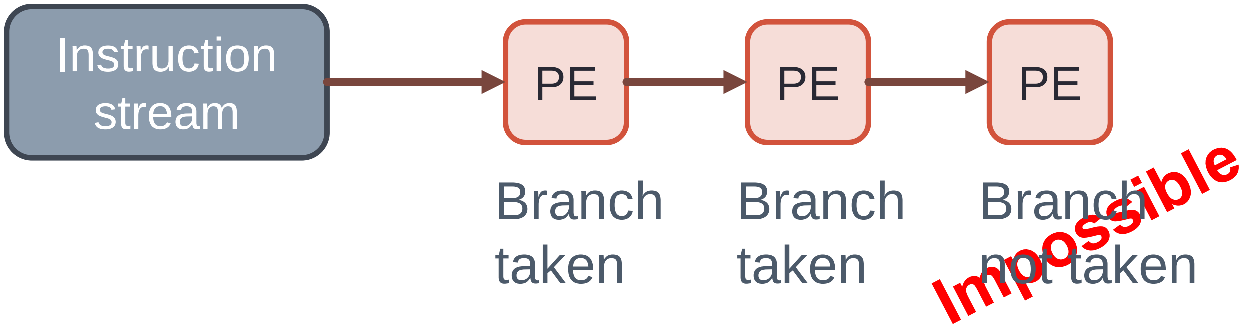
Control Flows in SIMD

- Growing complexity of multimedia application algorithms
- Kernels tend to include more conditional branches
- Limitation of SIMD



Control Flows in SIMD

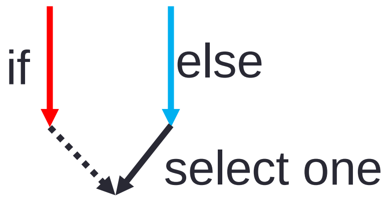
- Growing complexity of multimedia application algorithms
- Kernels tend to include more conditional branches
- Limitation of SIMD



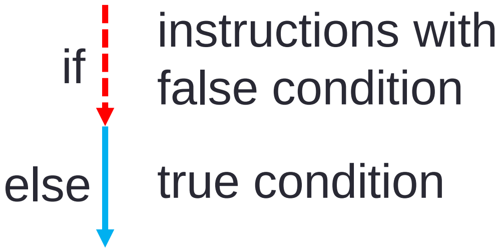
- Solution – Predication
  - ✓ Converts control flows to data flows

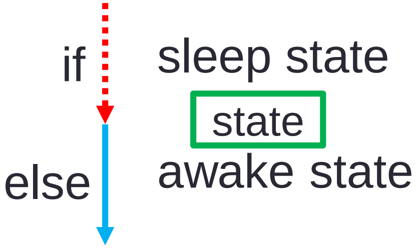
Handling Control Flows

- Partial predication
  - ✓ Execute both (if and else) paths
  - ✓ Choose the correct result later by using a predicated instruction (conditional move)



- Full predication
  - ✓ Two types
    - Condition-based full predication (CONDFULL)
      - Every instruction is predicated
    - State-based full predication (STATEFULL)
      - Execution depends on the state
      - Most instructions are not predicated, but the executions are effectively predicated
      - Nesting is easily implemented





Handling Control Flows

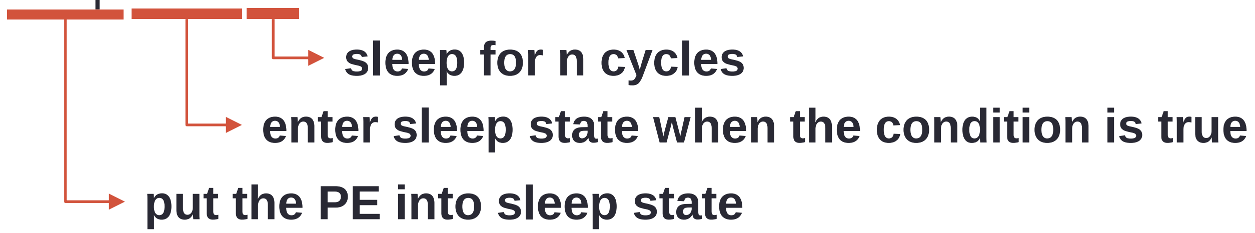
- Condition-based full predication (CONDFULL)
  - ✓Ex) ARM
  - ✓Status register per PE
  - ✓Condition operand per instruction

→ Decide execution of instructions

C code	CONDFULL	
if (c[i] >= 1) {	cmp R0 #1	
x = x+1;	add ge R1 R1 #1	if
y = y+1; }	add ge R2 R2 #1	
else {	sub lt R1 R1 #1	else
x = x-1;	sub lt R2 R2 #1	
y = y-1; }}		

Handling Control Flows

- State-based full predication (STATEFULL)
  - ✓ Each PE has a state register to indicate awake or sleep
  - ✓ New instruction – sleep
    - sleep cond #n



sleep for n cycles

enter sleep state when the condition is true

put the PE into sleep state

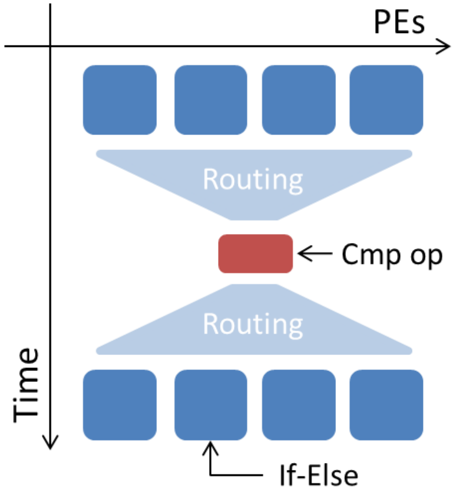
C code	STATEFULL	
if (c[i] >= 1) {	cmp R0 #1	
x = x+1;	sleep lt #3	
y = y+1; }	add R1 R1 #1	if
else {	add R2 R2 #1	
x = x-1;	sleep uc #2	
y = y-1; }}	sub R1 R1 #1	else
	sub R2 R2 #1	

Distributed Register Files

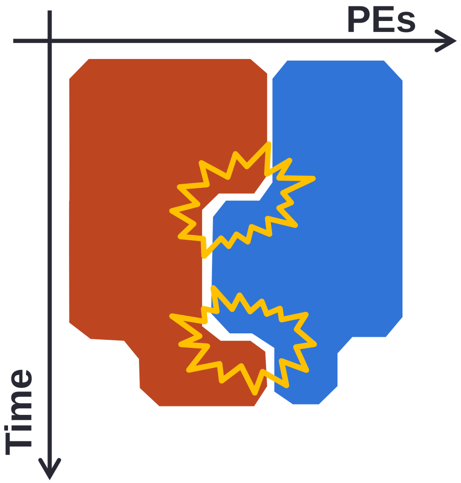
- Each PE has its own local register file
  - → Distributed register files
  - ✓ Good for scalability, wiring, …
- PEs cannot directly access other PEs' local register files
  - ✓ Routing is needed

Problems of Distributed Register Files

- Overhead due to heavy communication
  - ✓ Predicate variables from/to many PEs increase communications

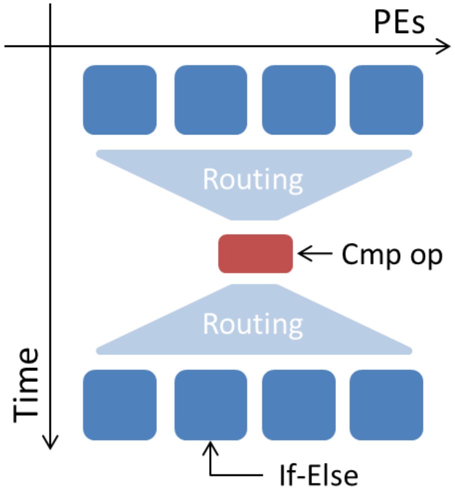


- Overhead due to spilling
  - ✓ Sharing a PE among multiple conditionals executed in parallel may require spilling of the state register

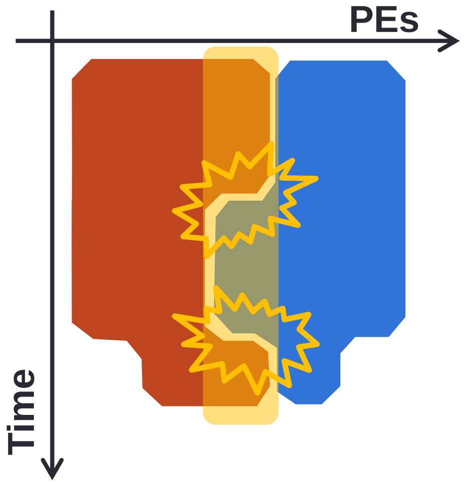


Problems of Distributed Register Files

- Overhead due to heavy communication
  - ✓ Predicate variables from/to many PEs increase communications

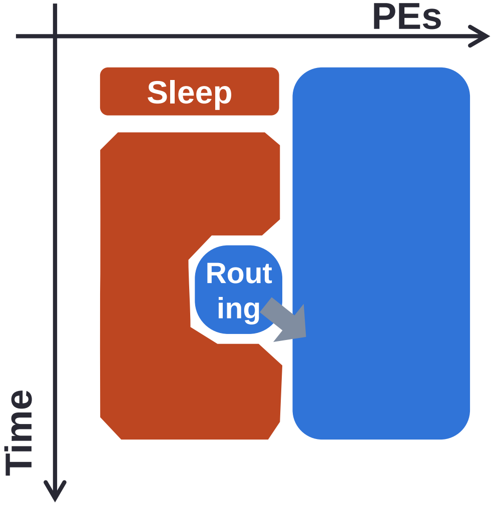


- Overhead due to spilling
  - ✓ Sharing a PE among multiple conditionals executed in parallel may require spilling of the state register



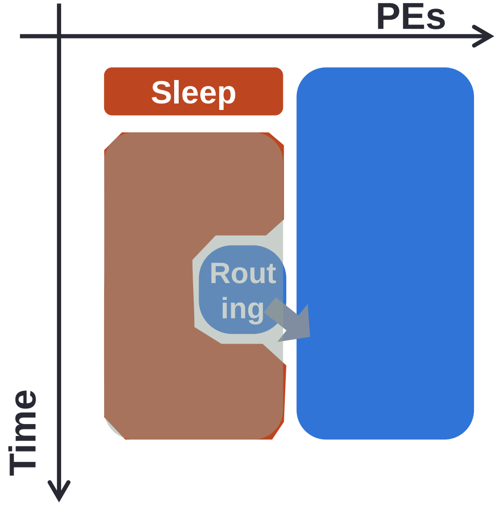
Problems of Distributed Register Files

- Delayed routing in sleep mode
  - ✓On-demand routing is impossible in sleep mode
  - ✓Routing is delay until the end sleep mode
    → performance degradation



Problems of Distributed Register Files

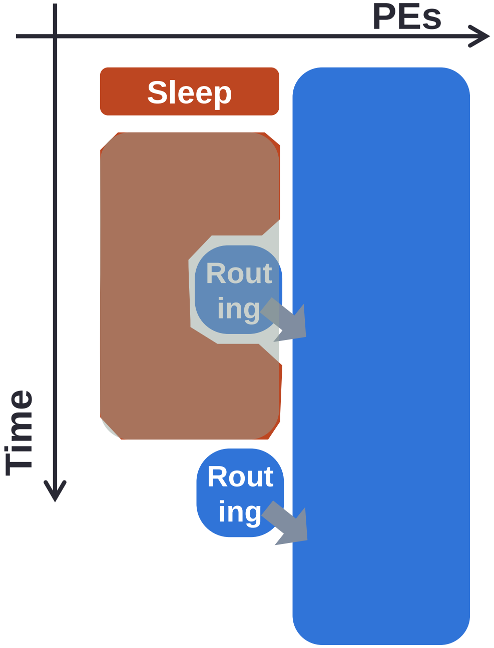
- Delayed routing in sleep mode
  - ✓On-demand routing is impossible in sleep mode
  - ✓Routing is delay until the end sleep mode
    → performance degradation



Problems of Distributed Register Files

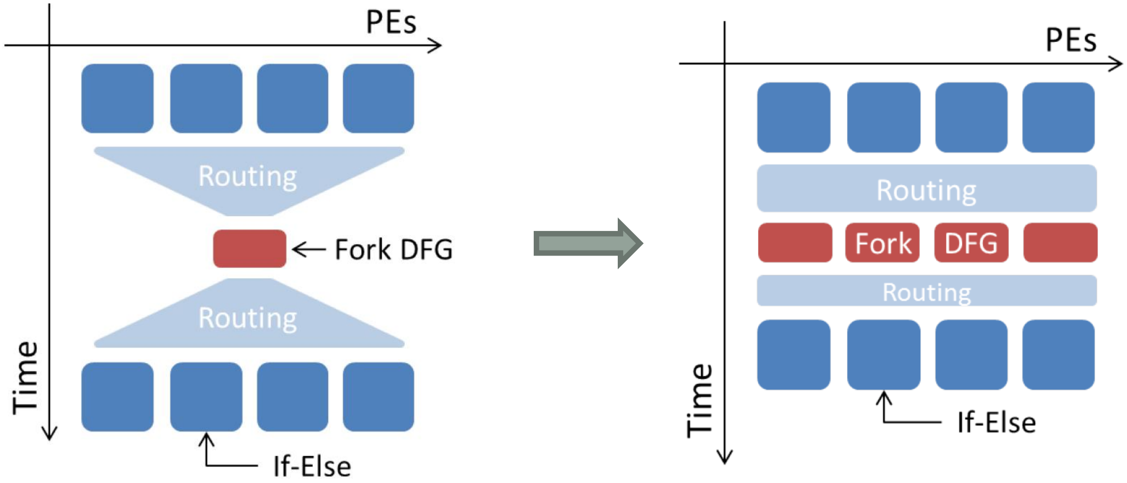
- Delayed routing in sleep mode
  - ✓On-demand routing is impossible in sleep mode
  - ✓Routing is delay until the end sleep mode

    → performance degradation



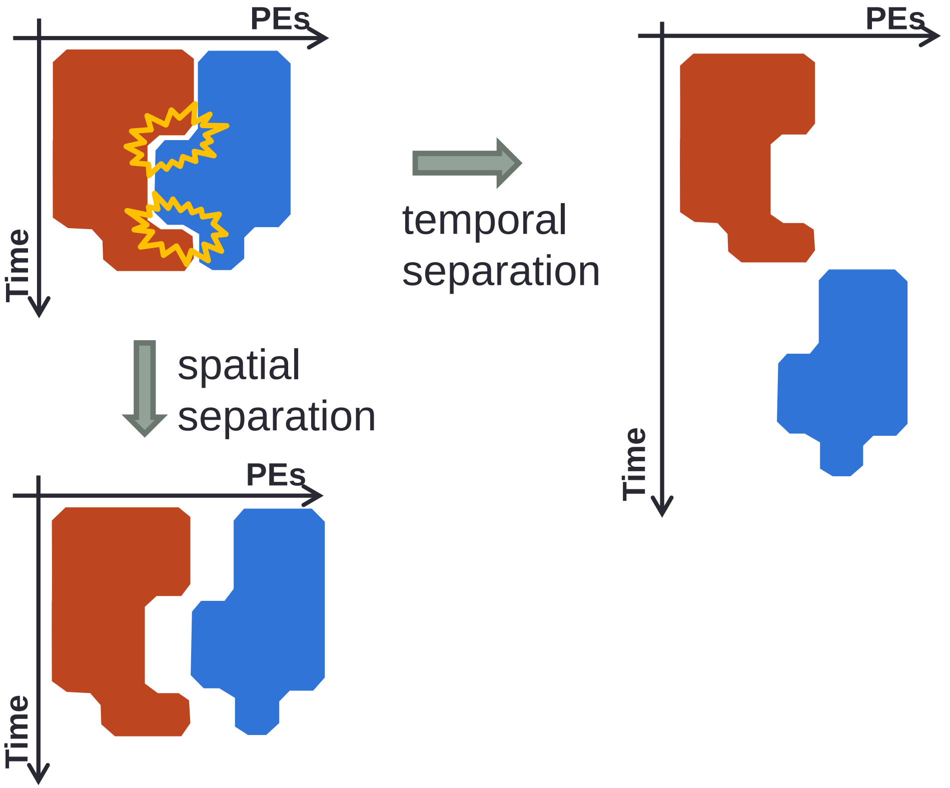
Solutions

- Duplicate operations for a predicate calculation and map them to different PEs
  - ✓Reduces the amount of communication



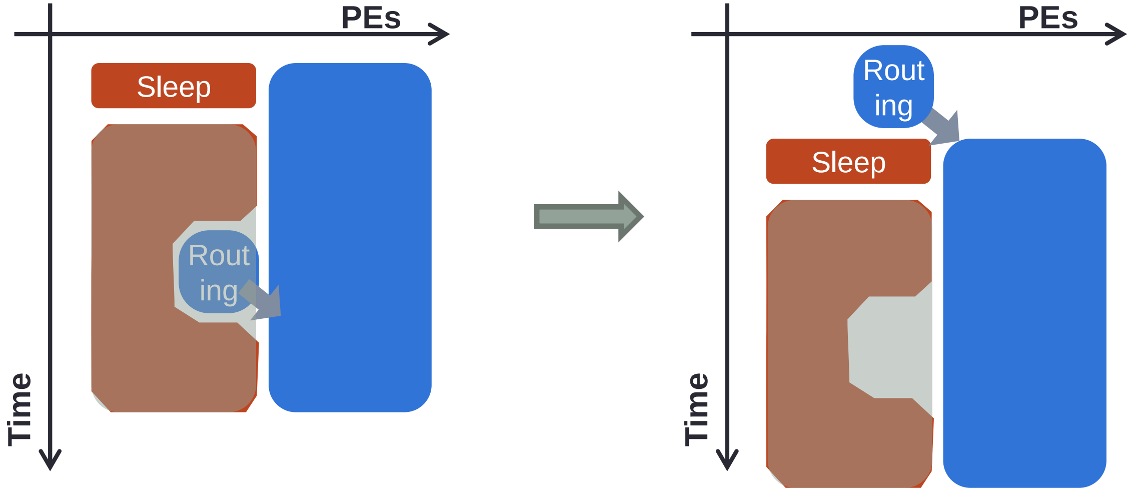
Solutions

- Map operations for different conditionals to different sets of PEs in time or space (separation)
  - ✓ Reduces the amount of spilling

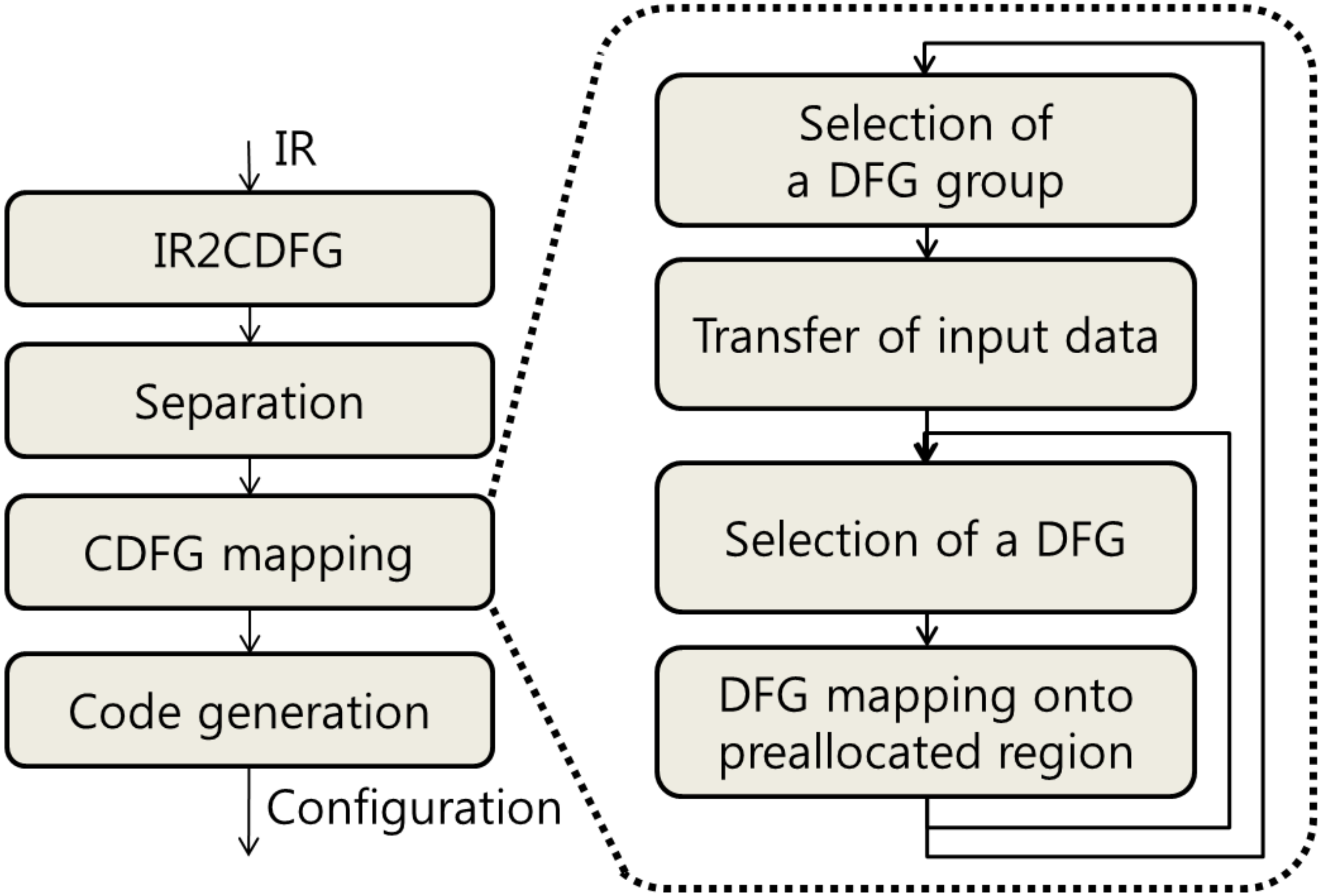


Solutions

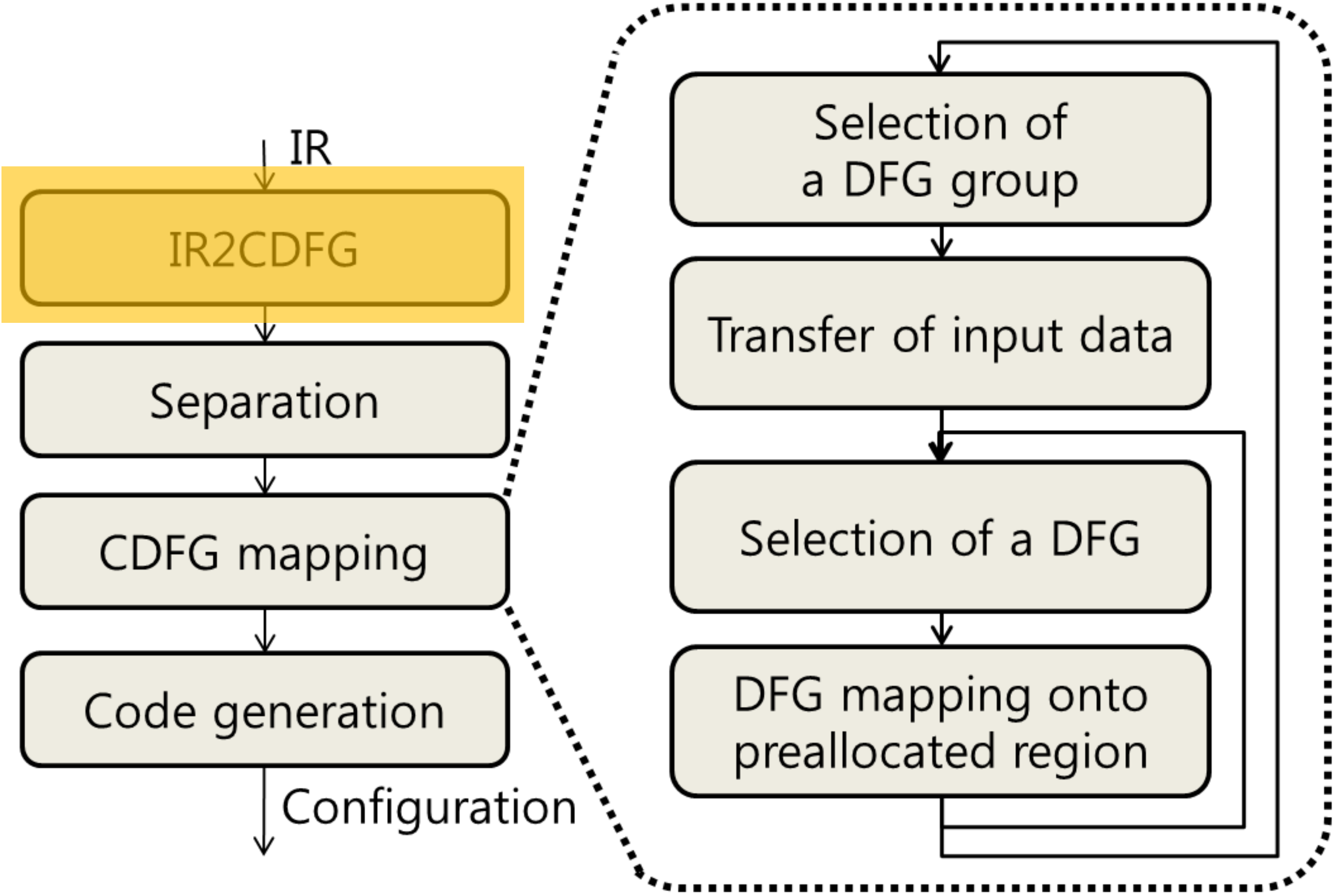
- Route data in advance if it is available
  - ✓ Avoids delayed routing in sleep mode



Overall Flow

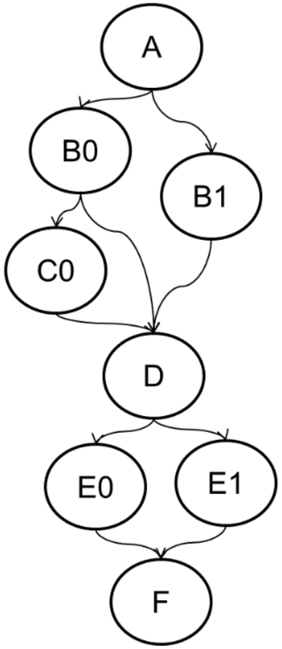


Overall Flow



From IR to CDFG

- Initial (IR)

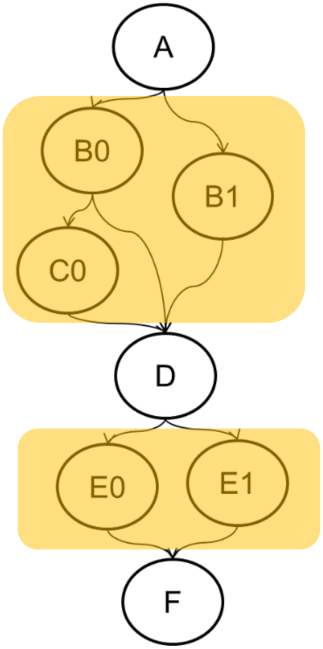


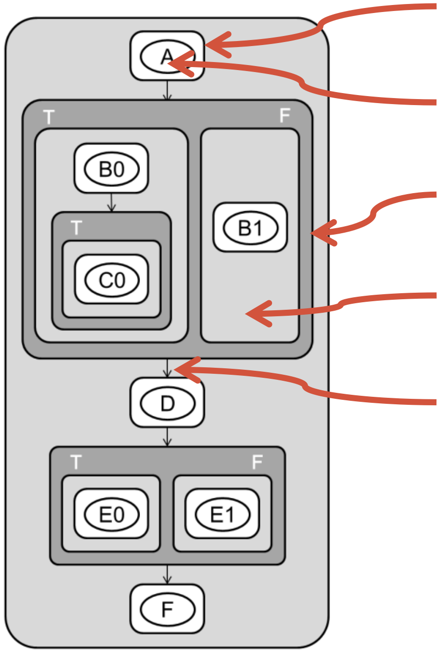
Flat structure

CFG with DFG nodes

From IR to CDFG

- Initial (IR) →
- Hierarchical CDFG representation





Unipath block

DFG

Multipath block

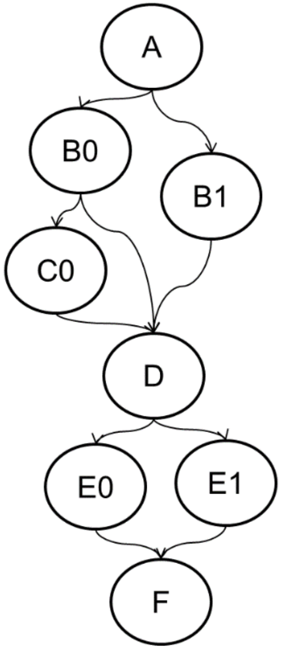
CDFG

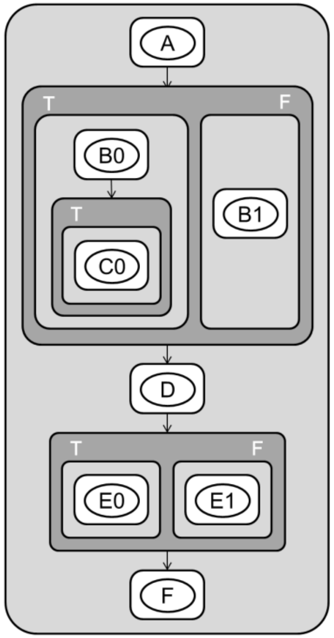
Data dependency

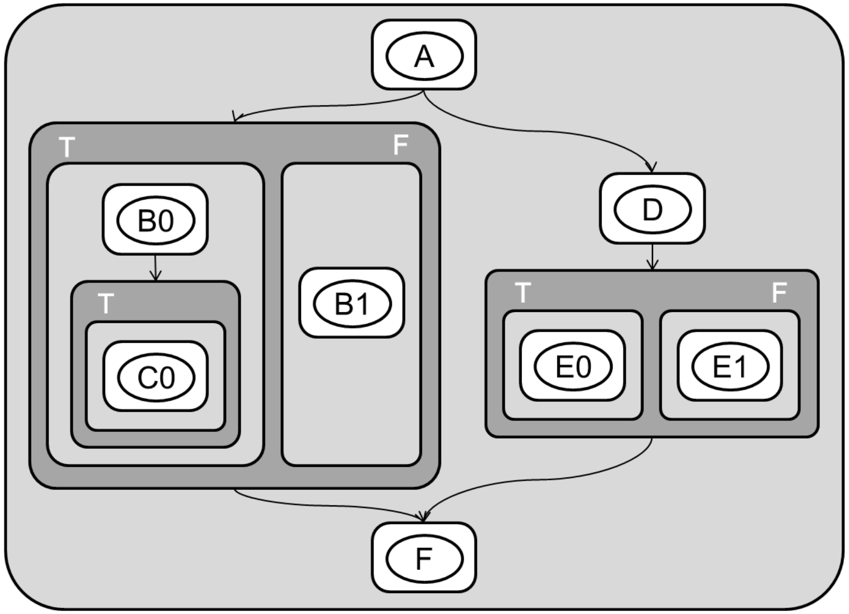
Identify conditionals

From IR to CDFG

- Initial (IR) →
- Hierarchical CDFG representation →
- Extracting parallelism

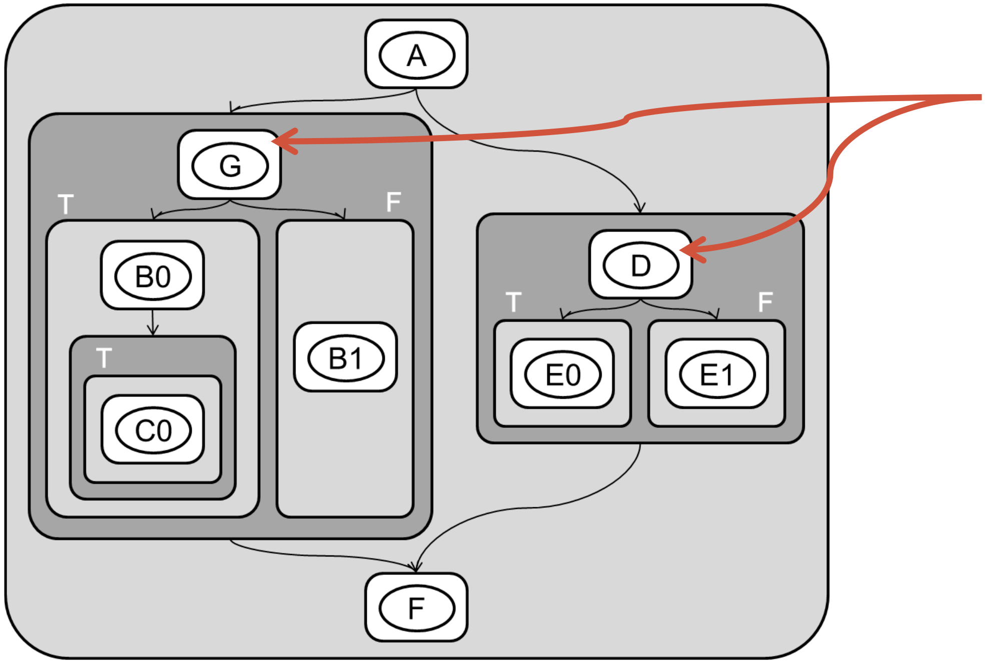






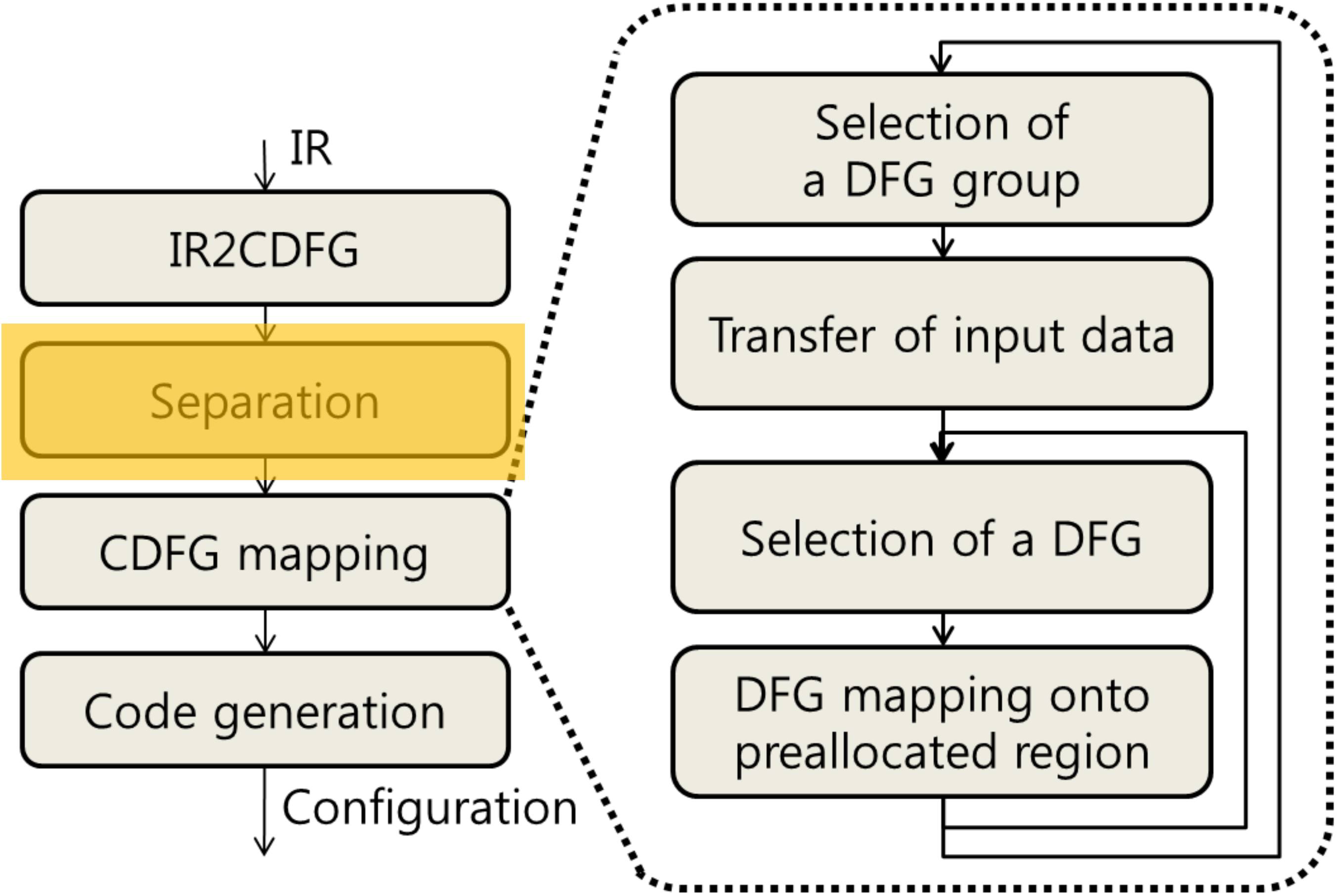
From IR to CDFG

- Final
  - ✓Hierarchical CDFG representation with fork DFGs



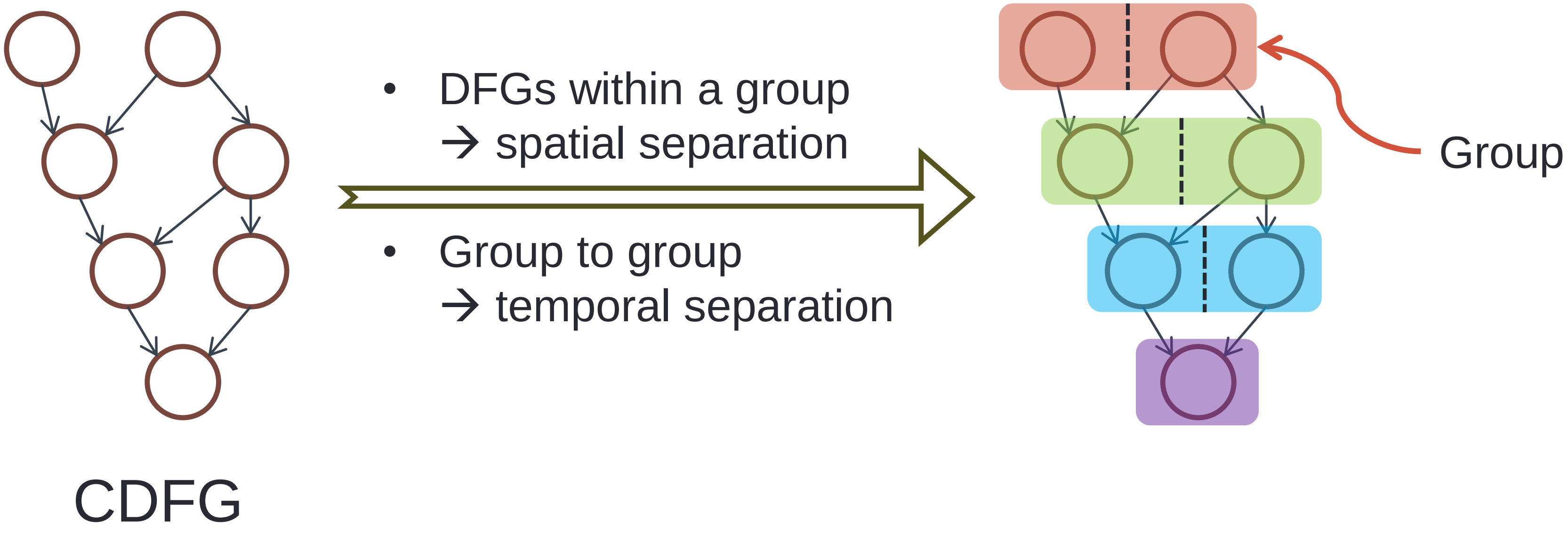
Fork DFGs (compare operations and their predecessors)

Overall Flow



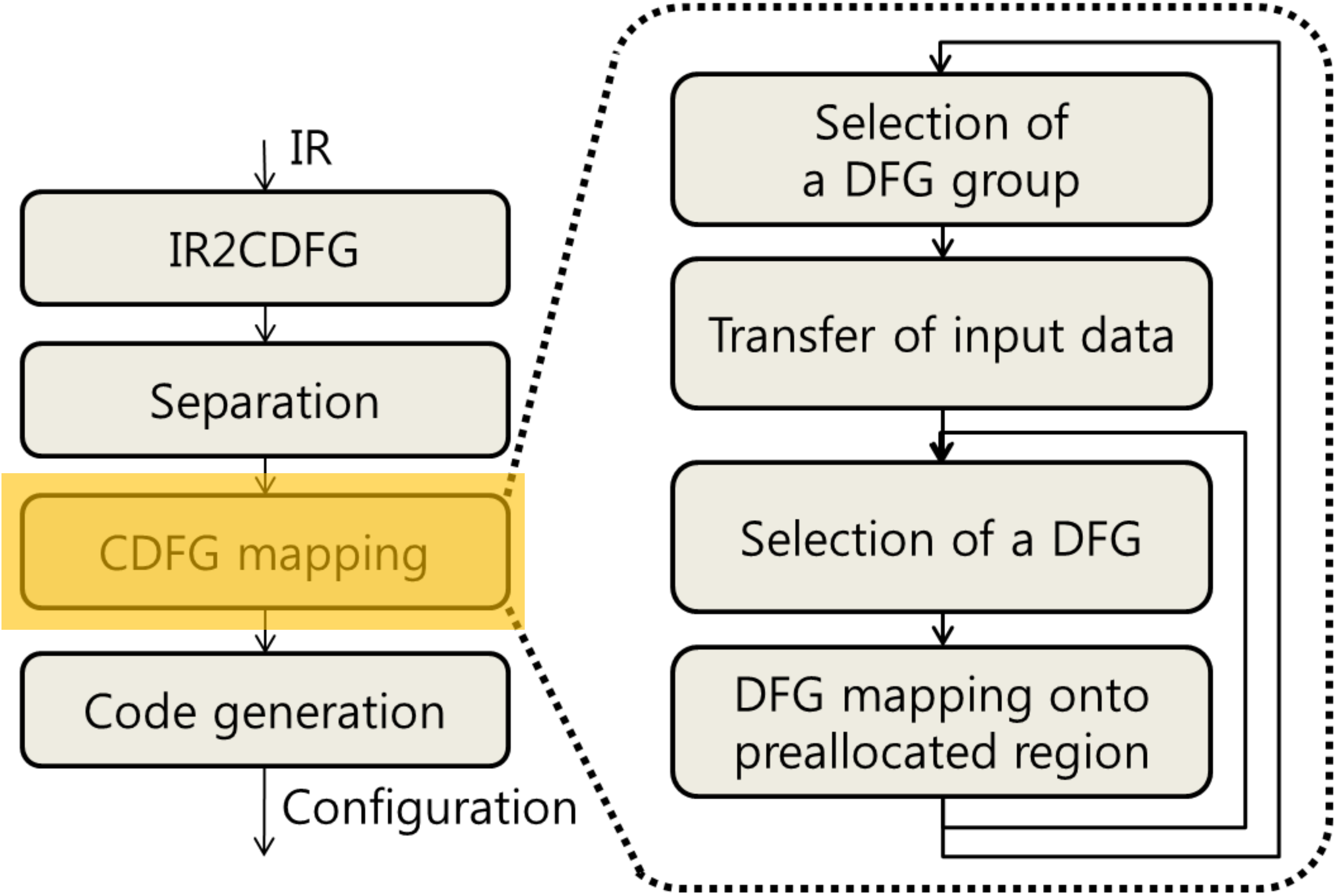
Separation

- Operations in different DFGs need to be separate either in time or space
  - ✓ Achieved by DFG grouping and PE-to-DFG allocation
  - ✓ DFG grouping: Put parallelizable DFGs into a group
  - ✓ PE-to-DFG allocation: Allocate enough number of PEs to avoid spills



- DFGs within a group → spatial separation
- Group to group → temporal separation

Overall Flow



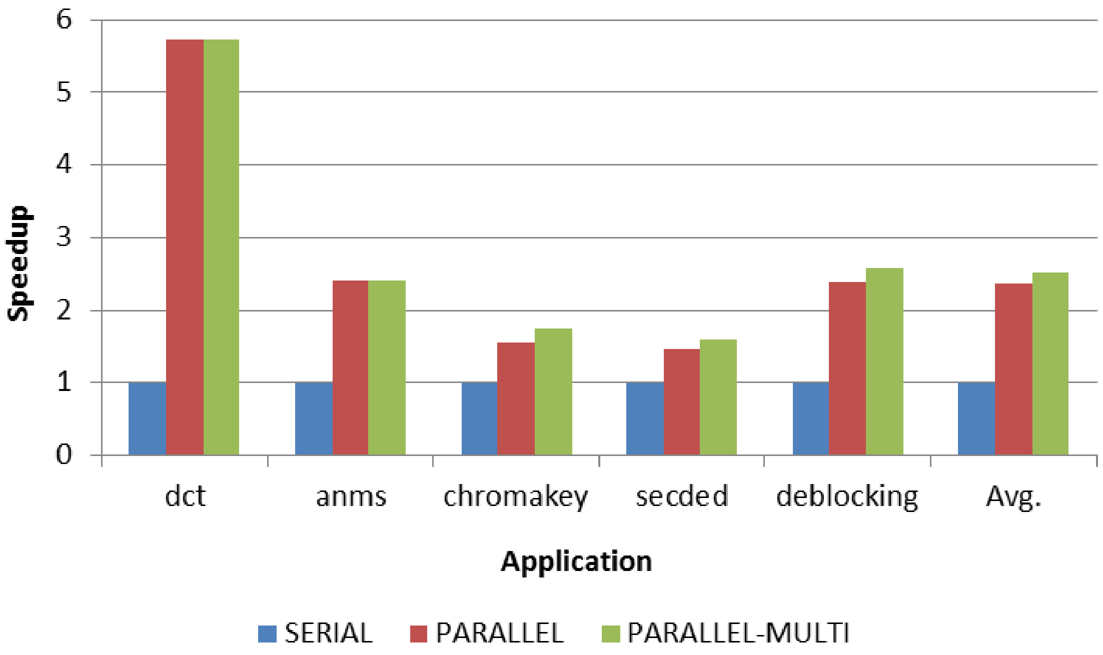
CDFG Mapping

- Selection of a DFG group
  - ✓Select a group to be mapped
- Route input data for each DFG
  - ✓If the input data is not already in the allocated PEs
- Selection of a DFG in the group
- DFG mapping onto the pre-allocated PEs
  - ✓Map the DFG using ILP
  - ✓Operations in fork DFGs are duplicated
    - Only when it improves the performance

Experimental Setup

- Setup
  - ✓ Frontend tool: Clang compiler
  - ✓ ILP solver for mapping DFGs: Gurobi Optimizer 5.0.2
  - ✓ Architecture: FloRA with State-based Full Predication
- Applications
  - ✓ DCT 8x8, getANMS, chromakey, SECDED, deblocking filter

Experimental Results



- SERIAL
  - ✓ Serial mapping of DFGs in a group
- PARALLEL
  - ✓ Parallel mapping of DFGs in a group
- PARALLEL-MULTI
  - ✓ Possible duplication of fork DFGs

- PARALLEL-MULTI
  - ✓ 2.51x speedup compared to SERIAL
  - ✓ 5.7% improvement over PARALLEL

Conclusion

- Kernels of multimedia applications tend to include more conditional branches
- Parallelization of such kernels is important (Amdahl's law)
- A new mapping framework to handle control flows
  - ✓ State-based full predication for SIMD
  - ✓ Extract parallelizable threads
  - ✓ Problems and solutions
    - Reduce communication overhead by duplicating fork DFGs
    - Reduce spill overhead by DFG grouping and PE-to-DFG allocation
    - Avoid delayed routing in sleep mode through pre-routing of input data
  - ✓ 2.51x performance improvement over conventional serial mapping

Thank you!