Architectural Synthesis of Computational Pipelines with Decoupled Memory Access

Shaoyi Cheng & John Wawrzynek

FPT 2014, Dec 10, Shanghai

Berkeley
UNIVERSITY OF CALIFORNIA

Background & Motivation

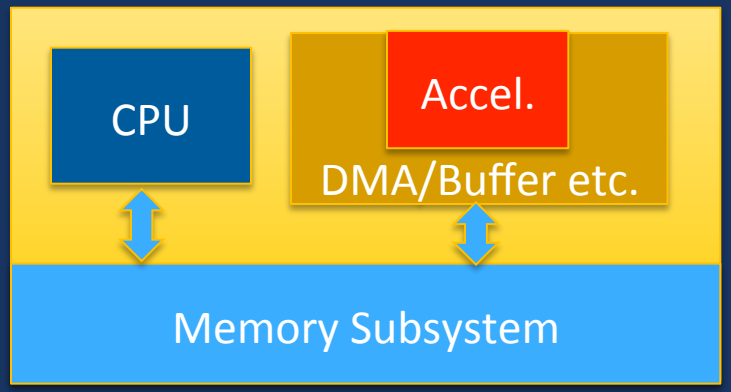
How to create high performance accelerators?

- Systems with CPUs+accelerators
- Fast mapping from high level description to HDL
- Of course, good QoR

Isn't this what high level synthesis is for?

You need to be a good HW engineer to use HLS properly

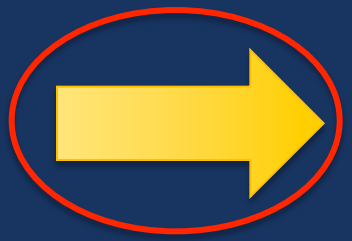
- How to ensure the generated hardware is good?
- All the stuff surrounding your accelerators



You need to be a good HW engineer to use HLS properly

- How to ensure the generated hardware is good?
- All the stuff surrounding your accelerators

Software Kernel written for CPUs



More than just a language change

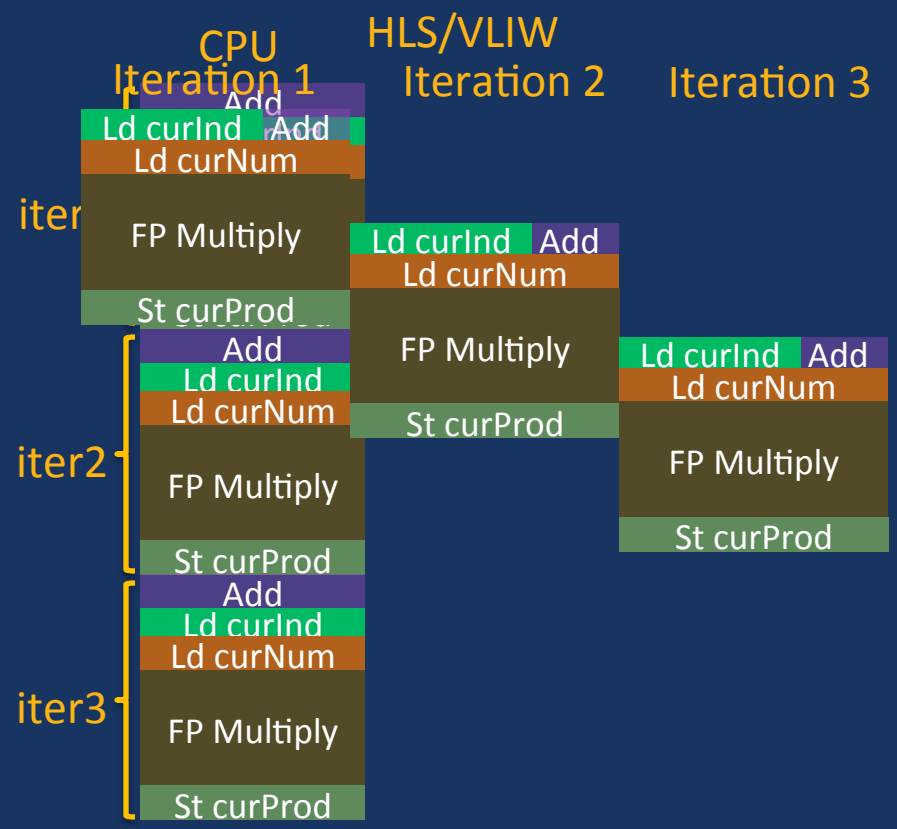
Hardware Accelerator on FPGA

Overview of the Proposed Methodology

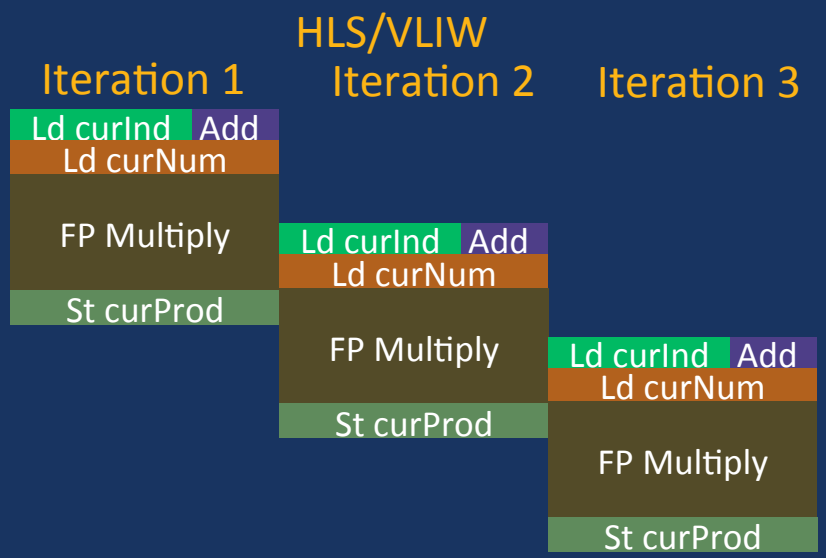
Berkeley
UNIVERSITY OF CALIFORNIA

Overview of the Proposed Methodology | An Example Kernel

```
float foo (float* x, float* product, int* ind)
{
  float curProd = 1.0;
  for(int i=0; i<N; i++)
  {
    int curInd = ind[i];
    float curNum = x[curInd];
    curProd = curProd * curNum;
    product[i] = curProd;
  }
  return curProd;
}
```

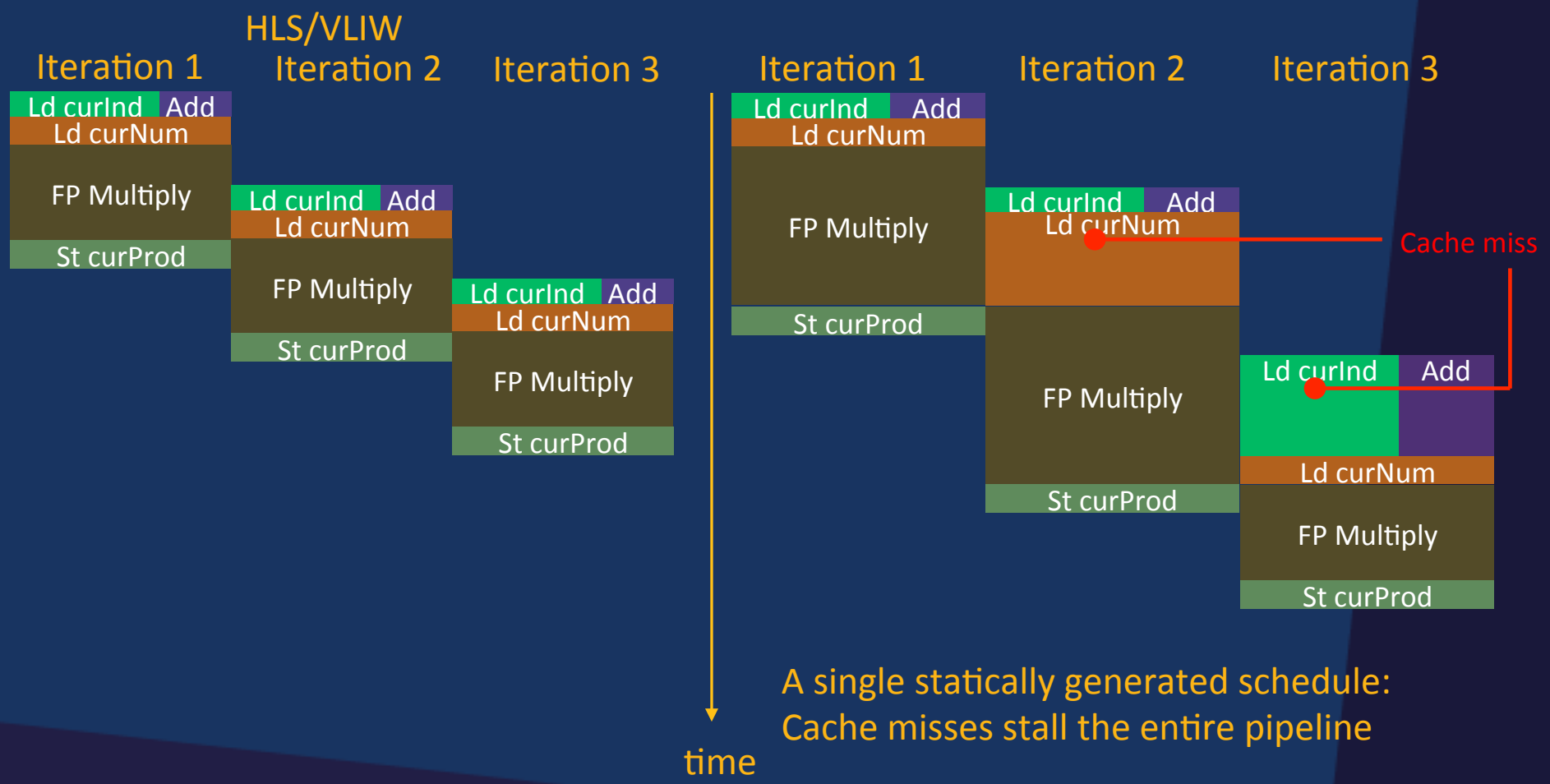


Overview of the Proposed Methodology | An Example Kernel



Berkeley
UNIVERSITY OF CALIFORNIA

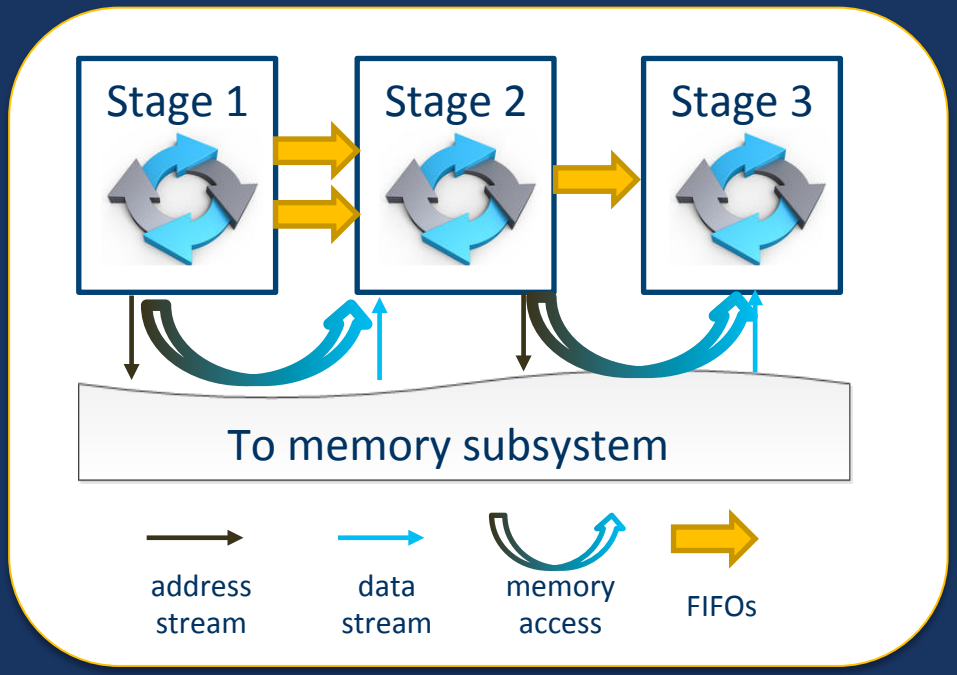
Overview of the Proposed Methodology | An Example Kernel



A single statically generated schedule:
Cache misses stall the entire pipeline

Berkeley
UNIVERSITY OF CALIFORNIA

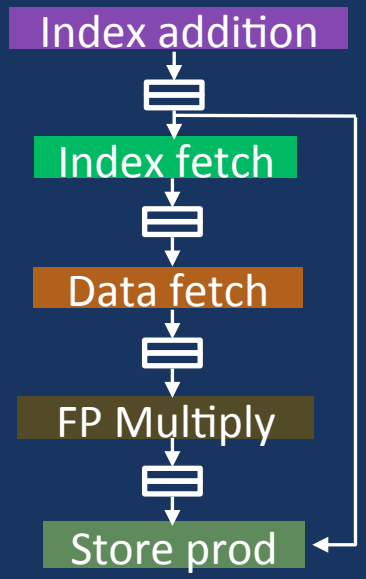
Pipeline Parallelism



- Break the nested loop into pipeline of processing stages
- Each stage runs independently
  - Can be generated using HLS
- Amenable for FPGA acceleration

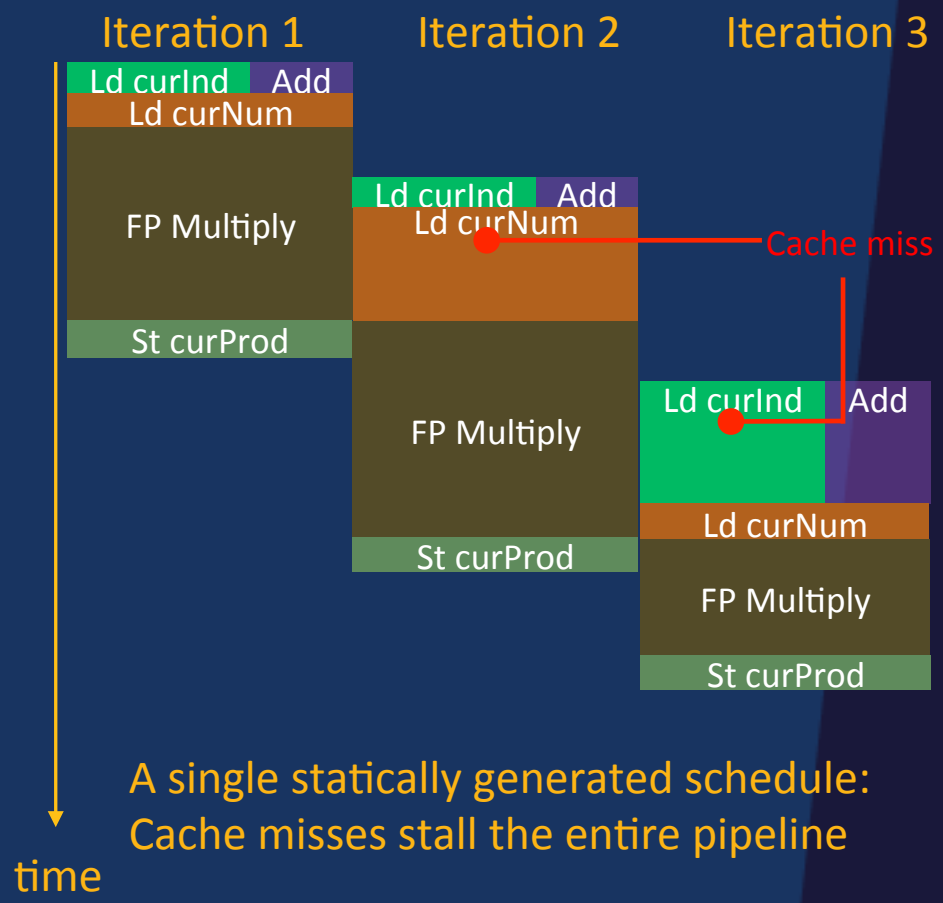
Overview of the Proposed Methodology | An Example Kernel

```
float foo (float* x, float* product, int* ind)
{
  float curProd = 1.0;
  for(int i=0; i<N; i++)
  {
    int curInd = ind[i];
    float curNum = x[curInd];
    curProd = curProd * curNum;
    product[i] = curProd;
  }
  return curProd;
}
```



Berkeley
UNIVERSITY OF CALIFORNIA

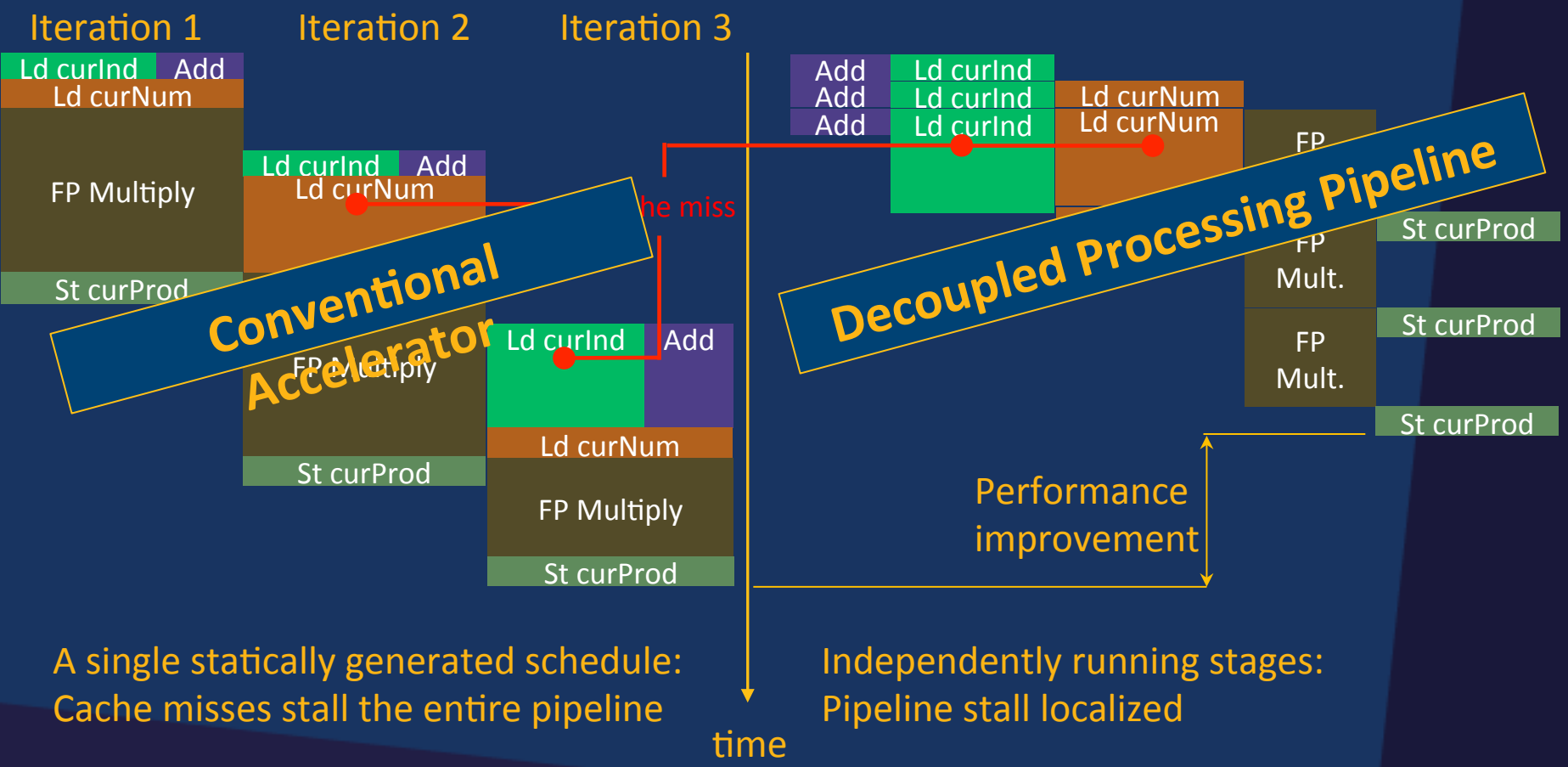
Overview of the Proposed Methodology | An Example Kernel



A single statically generated schedule:
Cache misses stall the entire pipeline

Berkeley
UNIVERSITY OF CALIFORNIA

Overview of the Proposed Methodology | An Example Kernel



A single statically generated schedule:
Cache misses stall the entire pipeline

Independently running stages:
Pipeline stall localized

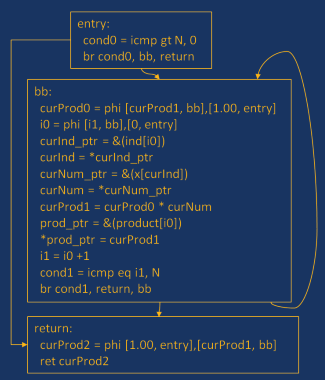
Berkeley
UNIVERSITY OF CALIFORNIA

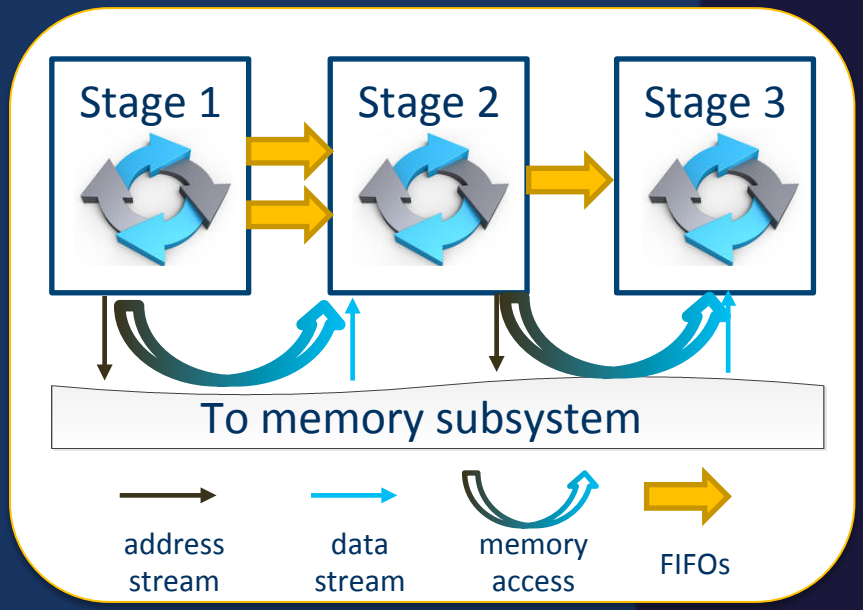
Generating Decoupled Processing Pipelines

Berkeley
UNIVERSITY OF CALIFORNIA

Generating Decoupled Processing Pipelines

Control Data Flow Graph (CDFG)





- Acyclic between stages – all cycles contained in individual stages
- Decouple memory operations from dependency cycle with long latency computation
- Min. number of memory operation in each stage

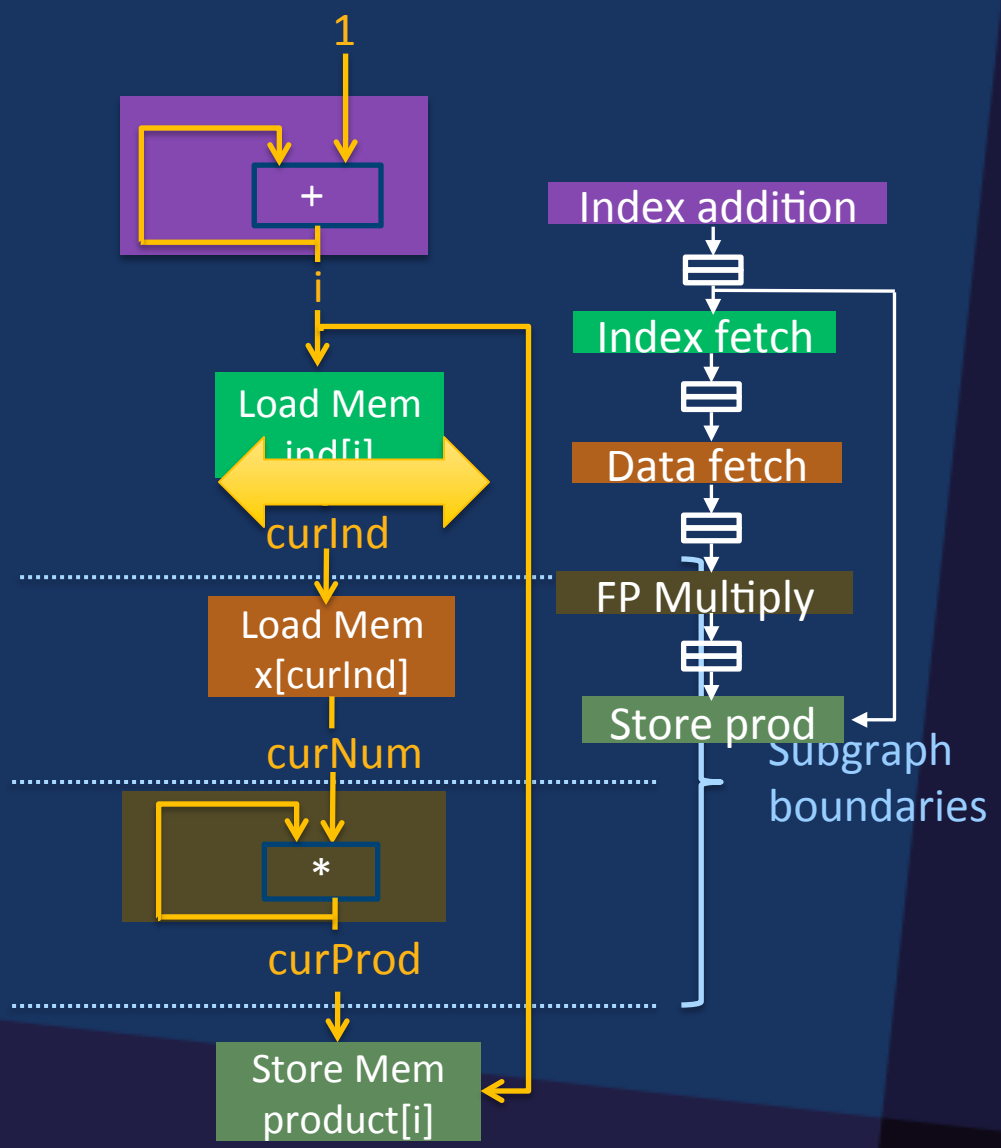
Berkeley
UNIVERSITY OF CALIFORNIA

From one CDFG to multiple subgraphs

- Collapse strongly connected components in CDFG
  - Obtain a directed acyclic graph (DAG)
- Perform topological sort on DAG
- Add a subgraph "boundary" when appropriate
  - After long latency SCCs. E.g. SCCs with multiply/floating point arith.
  - After long memory operation.

Generating Decoupled Processing Pipelines | Algorithm

```
float foo (float* x, float* product, int* ind)
{
  float curProd = 1.0;
  for(int i=0; i<N; i++)
  {
    int curInd = ind[i];
    float curNum = x[curInd];
    curProd = curProd * curNum;
    product[i] = curProd;
  }
  return curProd;
}
```

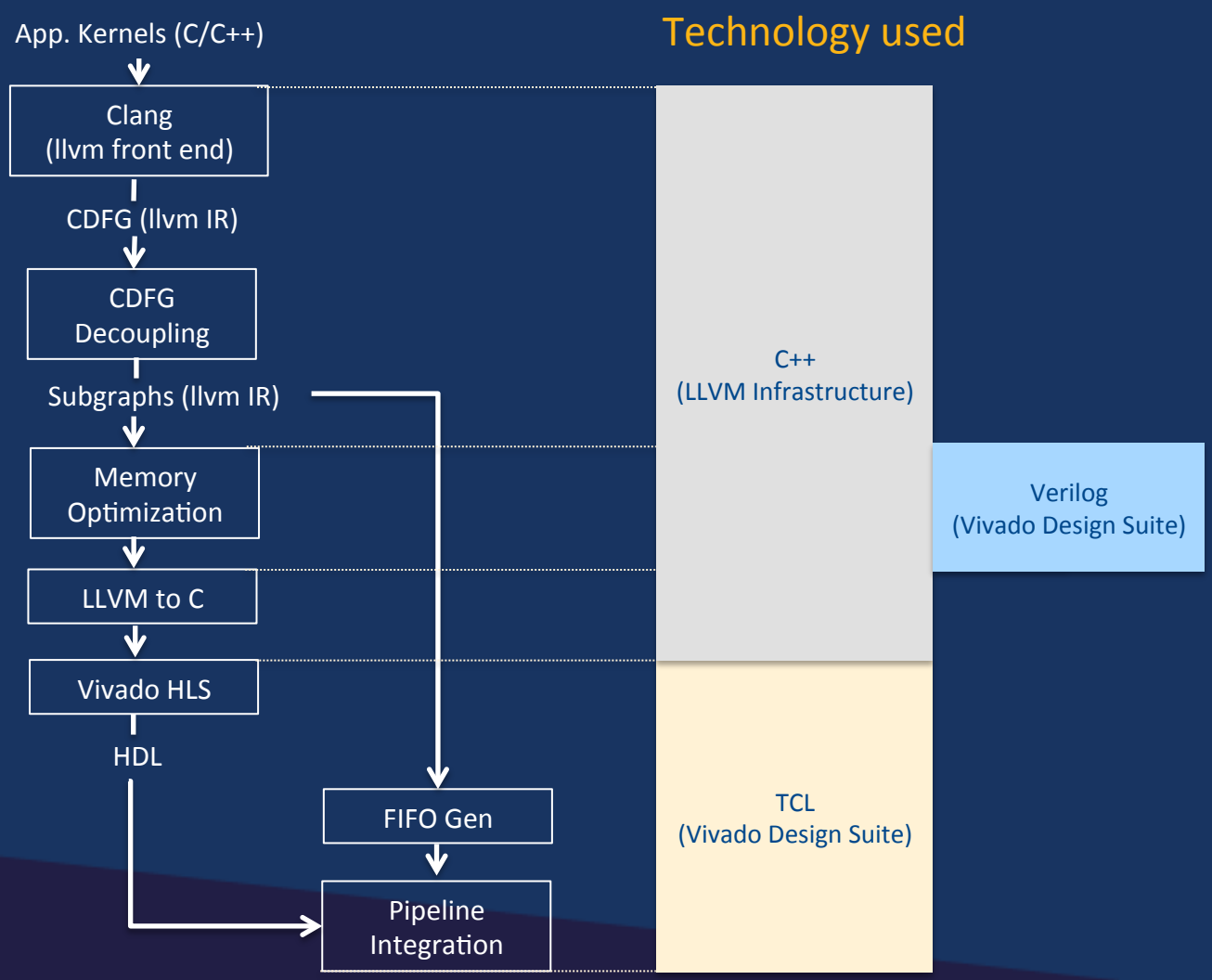


Berkeley
UNIVERSITY OF CALIFORNIA

How to use memory BW efficiently

- Memory BW improves faster than memory latency
- Pipelined memory accesses
  - Each stage can send in more mem. req without waiting for previous ones to come back
- Burst memory accesses
  - Each memory request can carry multiple words

Summary of Flow



Berkeley
UNIVERSITY OF CALIFORNIA

Experimental Evaluation

Berkeley
UNIVERSITY OF CALIFORNIA

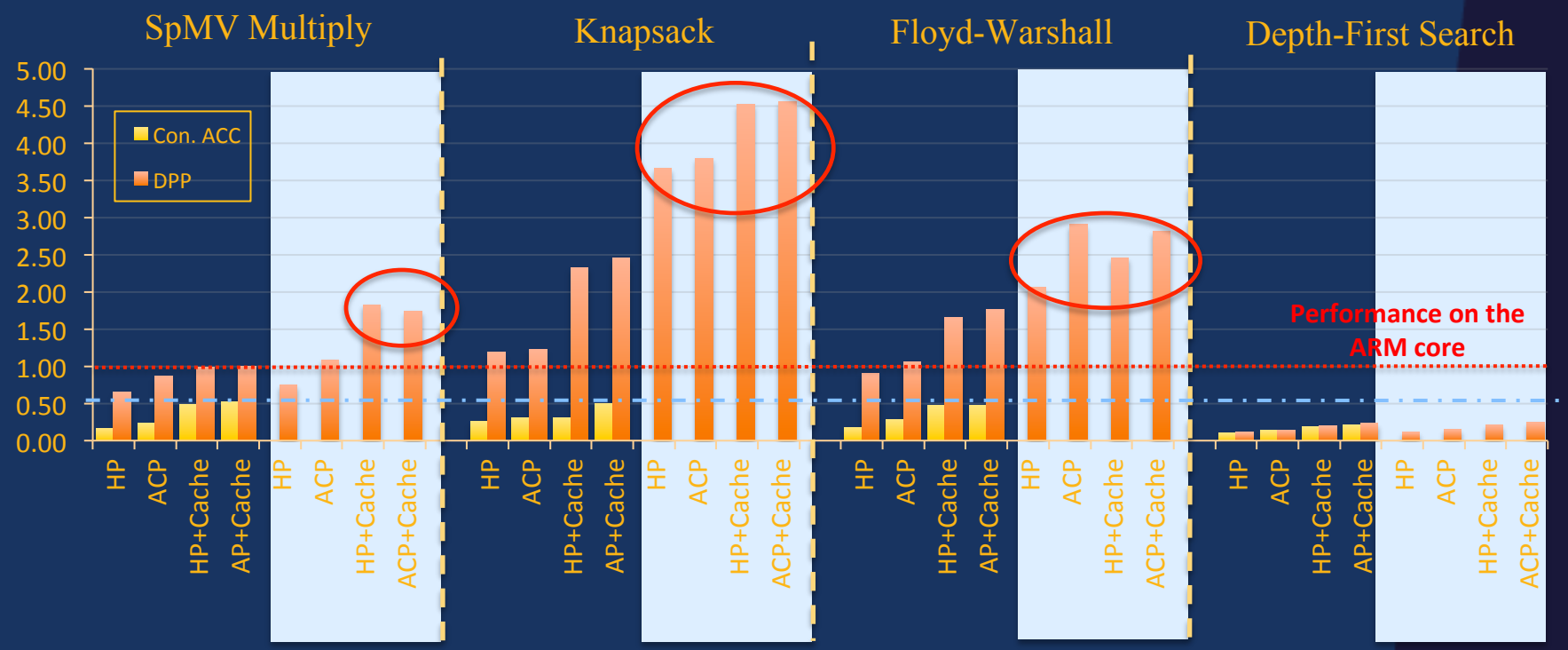
Experiment Platform

- Xilinx Zynq
  - Software running on ARM, generated accelerators on FPGA
  - Not using separate DMA engines etc.
- Different memory subsystem configuration
  - ACP & HP bus to memory
  - With & without on FPGA cache

Benchmarks

- Kernels with irregular mem. access pattern
  - SpMV Multiply, Knapsack Problem, Floyd-Warshall and Depth first search
  - Memory locations accessed depend on computation results
- Large input data set
  - Does not fit on chip

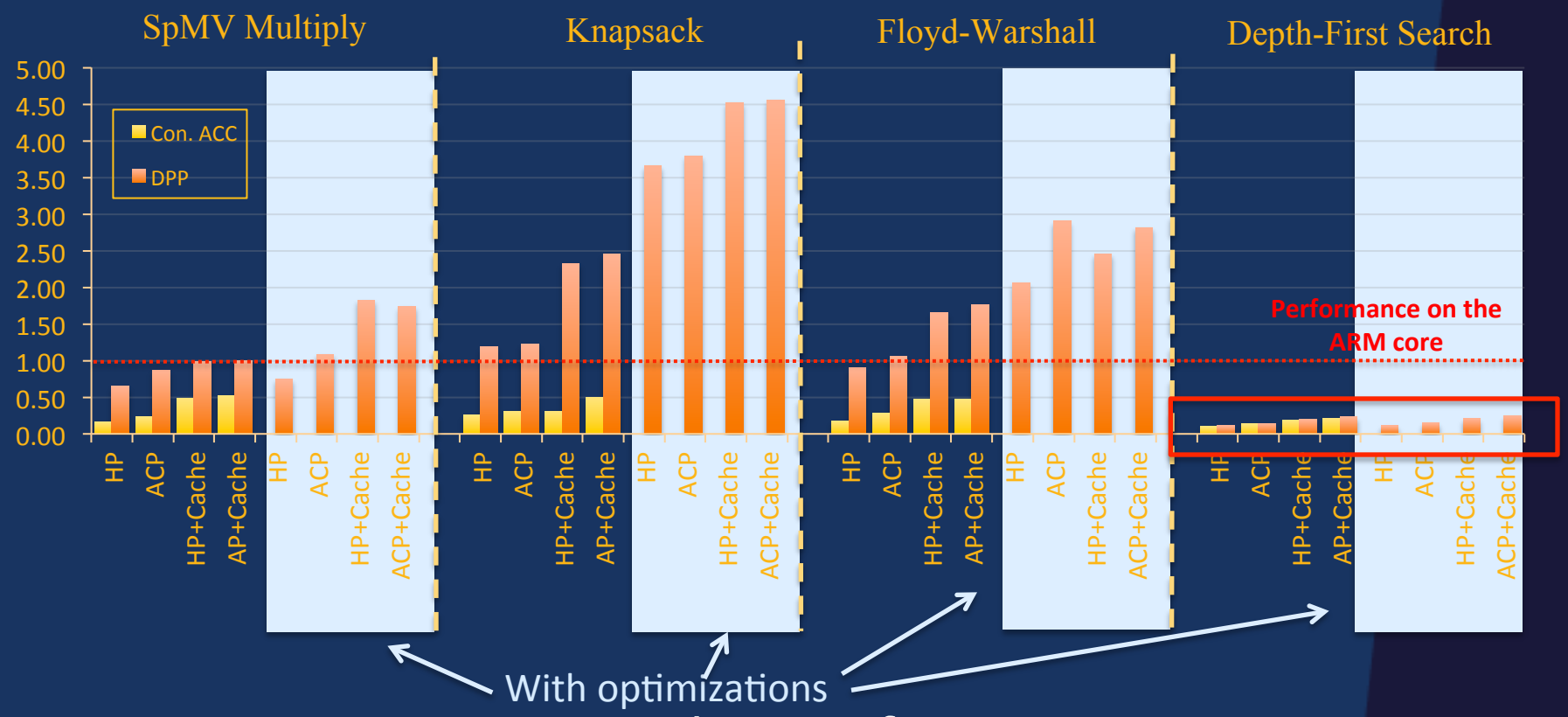
Experimental Evaluation | Performance Comparison



- Conventional Accelerators (Con. ACC) are actually slower than the baseline (superscalar ARM core)
- Decoupled processing pipelines (DPP) performs much better in 3 out of 4 case studies – 3.3x to 9.1 x over Con.ACC

Berkeley
UNIVERSITY OF CALIFORNIA

Experimental Evaluation | Performance Comparison



- Optimization in DPP provides significant improvement
  - Burst access makes good use of memory BW
- Limitation: no improvement when the critical cycle of dependence go through memory

Berkeley
UNIVERSITY OF CALIFORNIA

Experimental Evaluation | Area Comparison

Benchmark		ACP			ACP + 64KB Cache		
		LUT	FFs	BRAM	LUT	FFs	BRAM
SpMV Multiply	Con.ACC	9873	9116	10	7918	6792	21
	DPP	8577	8837	10	6718	6788	21
	% Change	*-13.1*	*-3.1*	0.0	*-15.2*	*-0.1*	0.0
Knapsack	Con.ACC	7672	7490	8	6573	5885	21
	DPP	8089	8787	8	6970	7256	21
	% Change	*+5.4*	*+17.3*	0.0	*+6.0*	*+23.3*	0.0
Floyd-Warshall	Con.ACC	2491	3528	0	3806	4629	19
	DPP	7659	7210	0	8995	8309	19
	% Change	*+207.5*	*+104.3*	0.0	*+104.4*	*+79.5*	0.0
DFS	Con.ACC	4810	4929	4	4931	4594	21
	DPP	8509	7813	4	7436	6298	21
	% Change	*+76.9*	*+58.5*	0.0	*+50.8*	*+37.1*	0.0

- Most benchmarks have higher area cost when DPPs are generated
- Design space exploration or formalized optimization may help

Berkeley
UNIVERSITY OF CALIFORNIA

Conclusion

- A simple transformation technique creating a better architectural paradigm for accelerators
- Use current HLS tool as “backend” for hardware implementation
- Average performance improvement of 5.6x over conventional HLS flow

Q & A