System Optimizations and Performance Tuning for New Generation FPGAs

Bingsheng He

NUS Computing

http://www.comp.nus.edu.sg/~hebs/

Learning Outcomes

- After this lecture, you should be able to understand
  - FPGA is an emerging hardware accelerator, which can be challenging for programming.
  - System optimization and performance tuning is very important for the system performance on FPGA.
  - More R&D efforts have to be put in systems, applications and tools for FPGAs.

Outline

- Background
  - FPGA
  - OpenCL SDK for FPGAs
- A Performance Analysis Framework
- Case Study: Database Systems
- Conclusion

What is FPGA?

- Wiki: A **field-programmable gate array** (**FPGA**) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

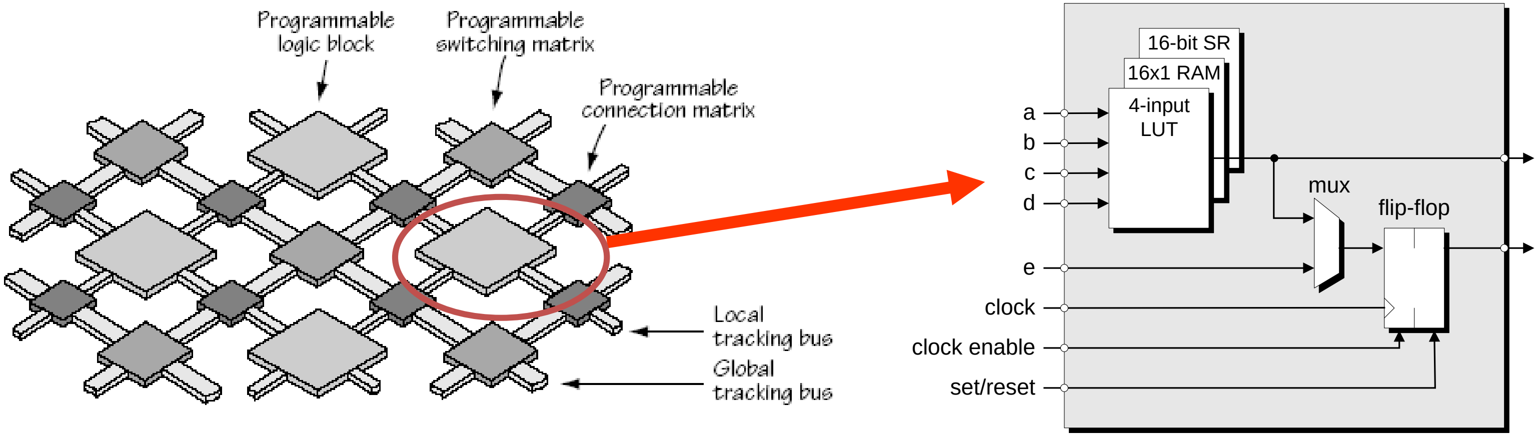
Advantages of FPGA

- Fine-grained hardware parallelism
  - Simultaneously process data
  - Bit-level computing
- The power of dedicated circuits
  - Custom for each application (without the limitation of Von Neumann architecture)
  - Higher performance/watt ratio→ energy efficient.
- Real time processing
  - Latency is usually predictable

Many Applications

- An FPGA can be used to solve any problem which is computable.
    - This is trivially proven by the fact FPGA can be used to implement a soft microprocessor.
- Specific applications of FPGAs include software-defined radio, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, and a growing range of other areas.
- Another trend on the usage of FPGAs is **hardware acceleration**, where one can use the FPGA to accelerate certain parts of an algorithm.

Basic Hardware Resources in FPGA



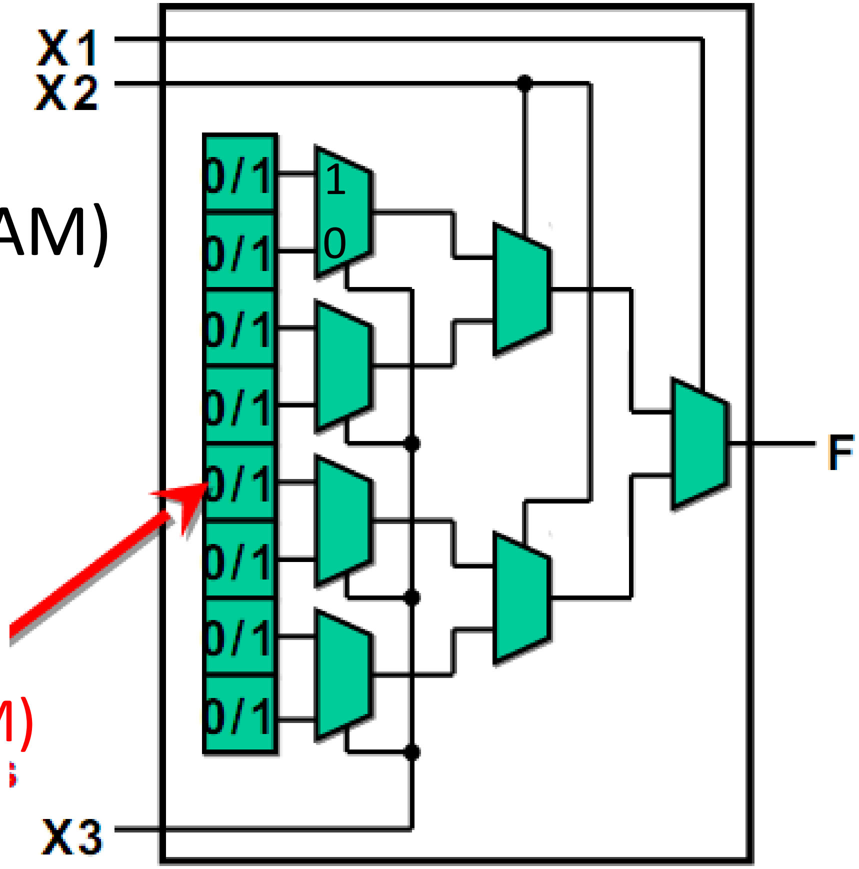
FPGA Fabric

Logic Block

- LUT (Look Up Table): implement logic functions
- REG (Flip-Flops): clocked storage elements.
- Interconnection & switch: switch matrix/switch box

LUT in FPGA

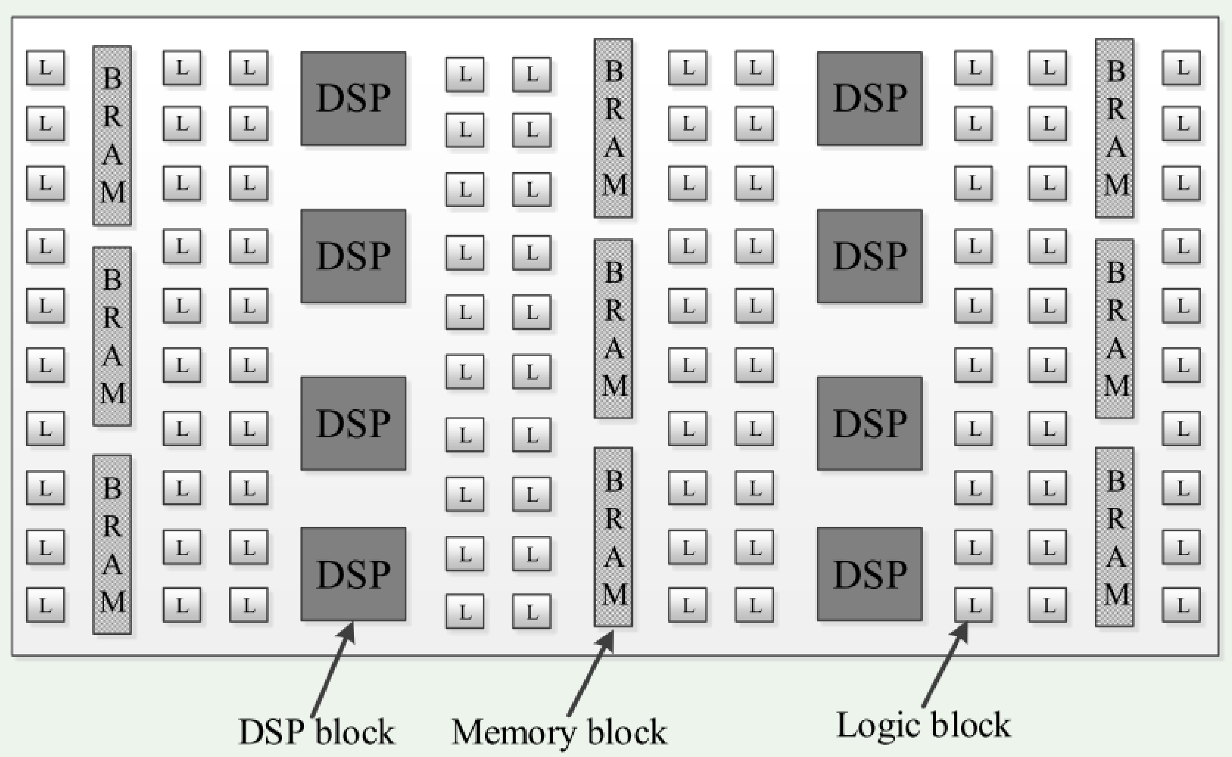
- Example: 3-input LUT
  - Multiplexer
  - Configuration memory (SRAM)
  - Filling content in SRAM (programming)



Configuration memory (SRAM)

$$F = x1.x2.x3 + \overline{x1}.x2.x3 + \cdots + \overline{x1}.\overline{x2}.\overline{x3}$$

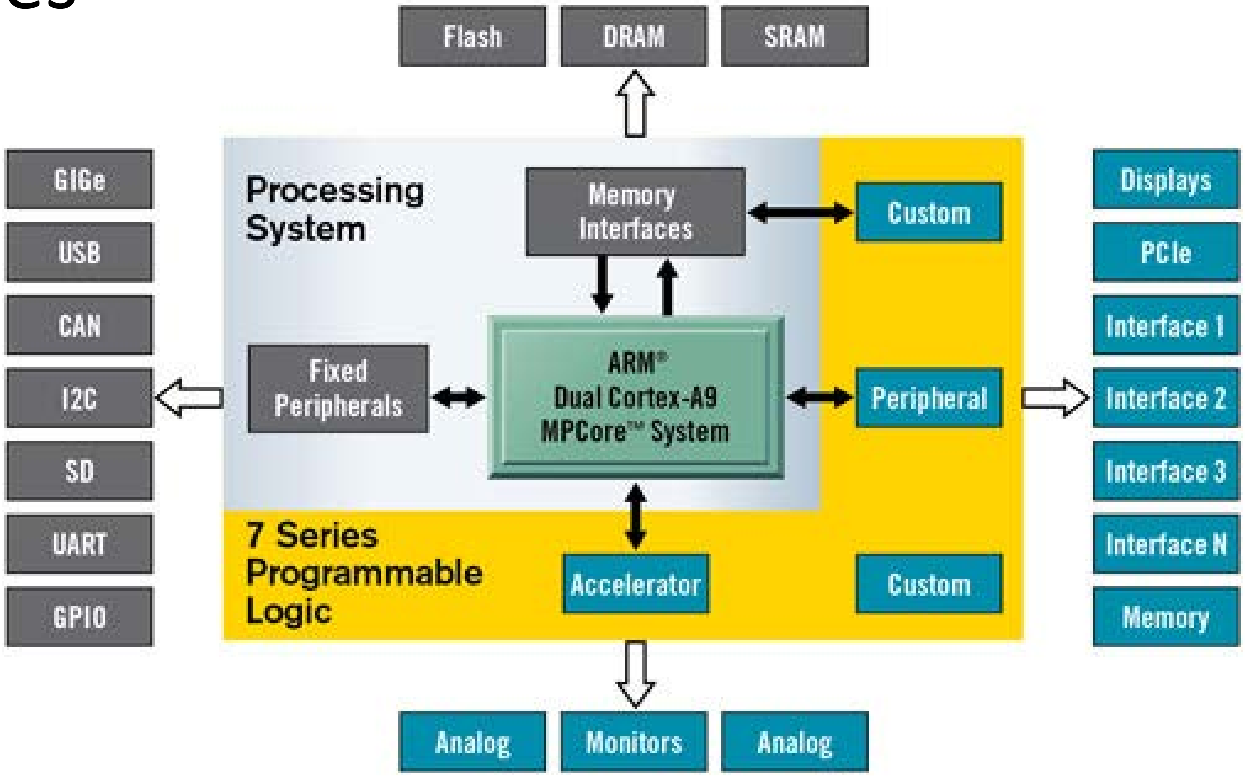
"Architectural Evolution" of FPGA (Field Programmable Gate Arrays)



- Hardware-centric → fine-grained parallelism
- Users need to program with hardware description languages ☹

Towards More Powerful FPGAs

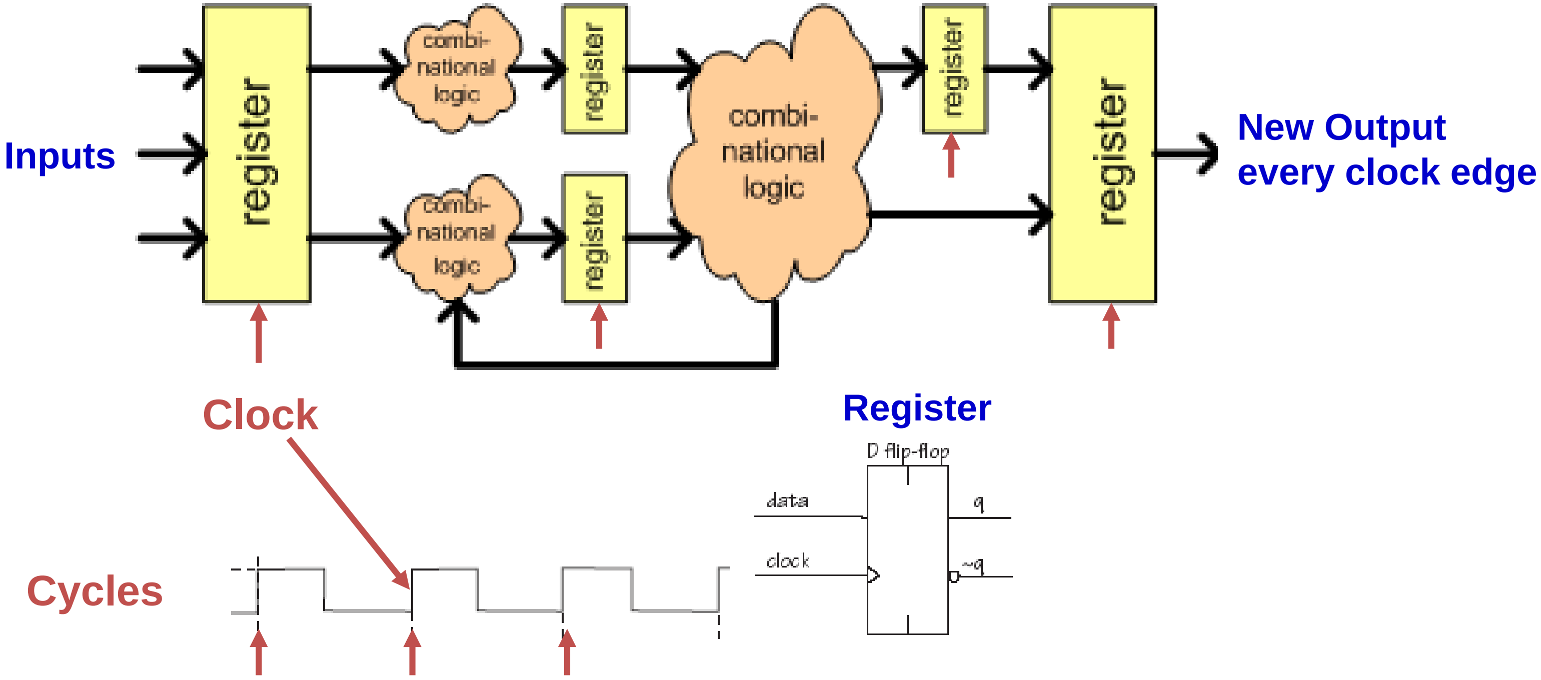
- General-purpose processors
- More interfaces



A Xilinx Zynq-7000 All Programmable System on a Chip.

Programming FPGA with Verilog

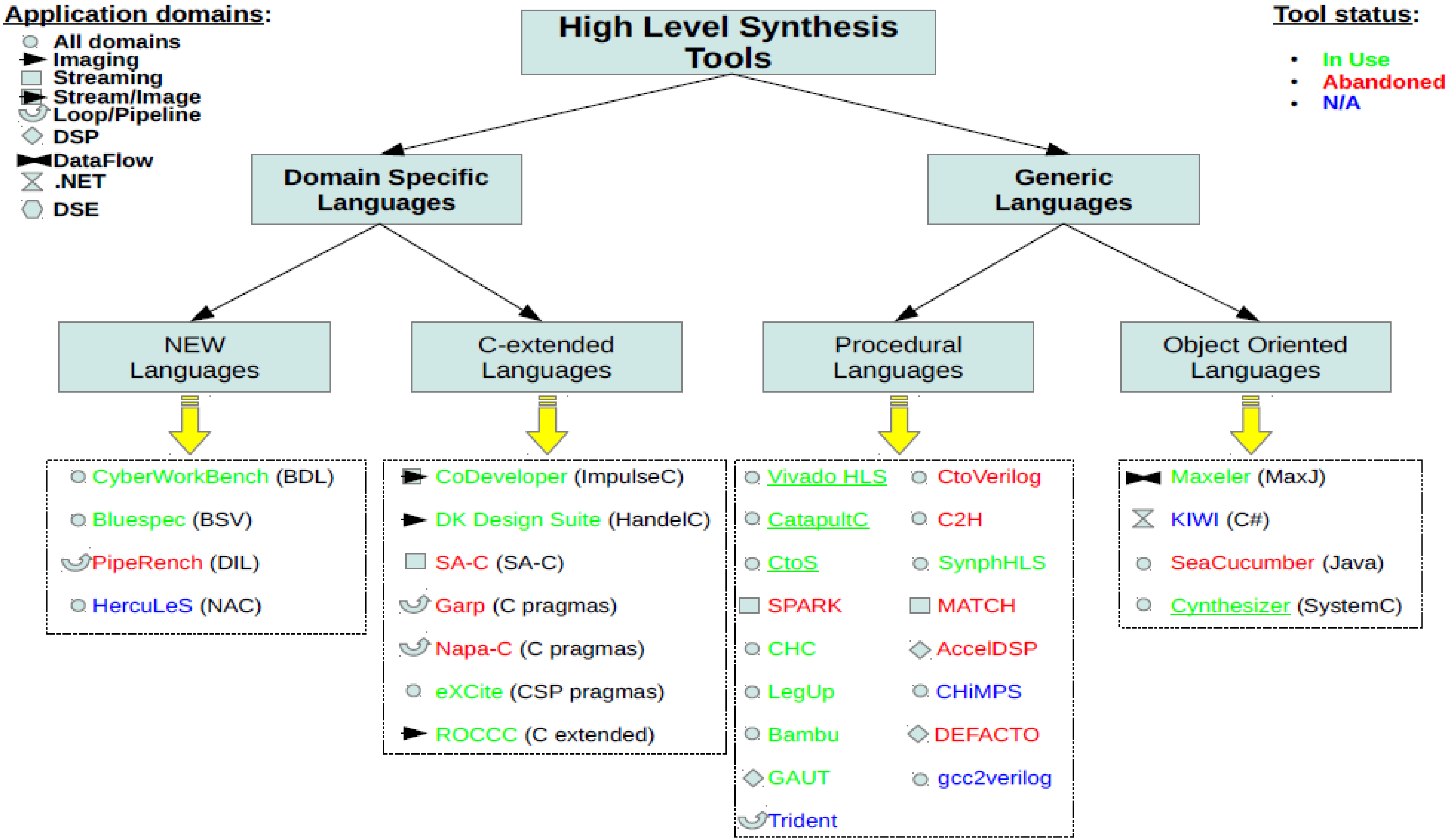
- Design for each register at each cycle.
- Combinational logic must meet timing (in one cycle)



High Level Synthesis (HLS)

- Generates register-transfer level (RTL) description from behavioral specification (e.g., C, OpenCL), in an automatic or semi-automatic way.
- The user can just focus on the high-level program.

HLS Taxonomy



Application domains:

- ○ All domains
- ► Imaging
- □ Streaming
- ⏵ Stream/Image
- ↻ Loop/Pipeline
- ◇ DSP
- ⋈ DataFlow
- ⧗ .NET
- ⬡ DSE

Tool status:

- In Use
- Abandoned
- N/A

High Level Synthesis Tools

Domain Specific Languages

NEW Languages

- ○ CyberWorkBench (BDL)
- ○ Bluespec (BSV)
- ↻ PipeRench (DIL)
- ○ HercuLeS (NAC)

C-extended Languages

- ⏵ CoDeveloper (ImpulseC)
- ► DK Design Suite (HandelC)
- □ SA-C (SA-C)
- ↻ Garp (C pragmas)
- ↻ Napa-C (C pragmas)
- ○ eXCite (CSP pragmas)
- ► ROCCC (C extended)

Generic Languages

Procedural Languages

- ○ Vivado HLS
- ○ CatapultC
- ○ CtoS
- □ SPARK
- ○ CHC
- ○ LegUp
- ○ Bambu
- ◇ GAUT
- ↻ Trident
- ○ CtoVerilog
- ○ C2H
- ○ SynphHLS
- □ MATCH
- ◇ AccelDSP
- ○ CHiMPS
- ◇ DEFACTO
- ○ gcc2verilog

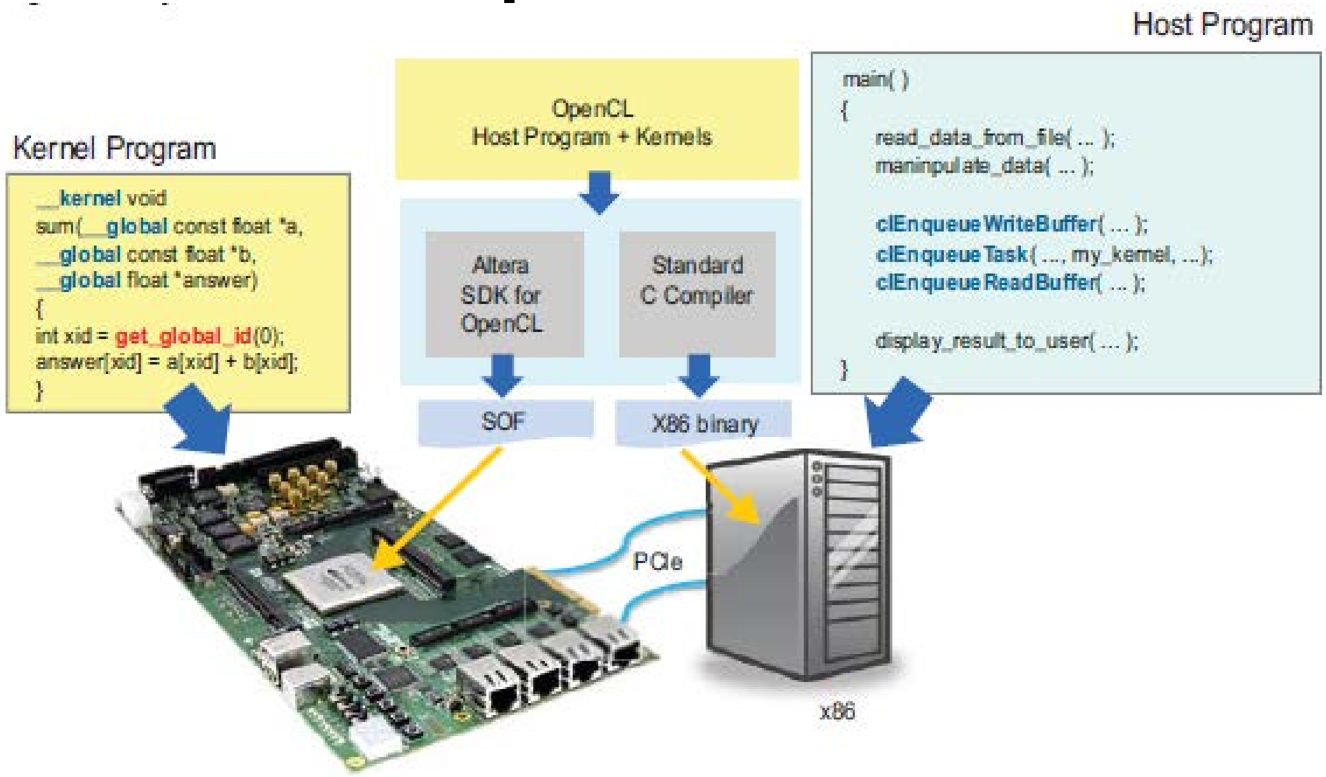
Object Oriented Languages

- ⋈ Maxeler (MaxJ)
- ⧗ KIWI (C#)
- ○ SeaCucumber (Java)
- ○ Cynthesizer (SystemC)

What is OpenCL?

- OpenCL stands for *Open Computing Language*
- OpenCL has been developed for heterogeneous system with a host-accelerator model of program execution.
- Intel plans to use OpenCL to program its CPU-FPGA heterogeneous system.

OpenCL

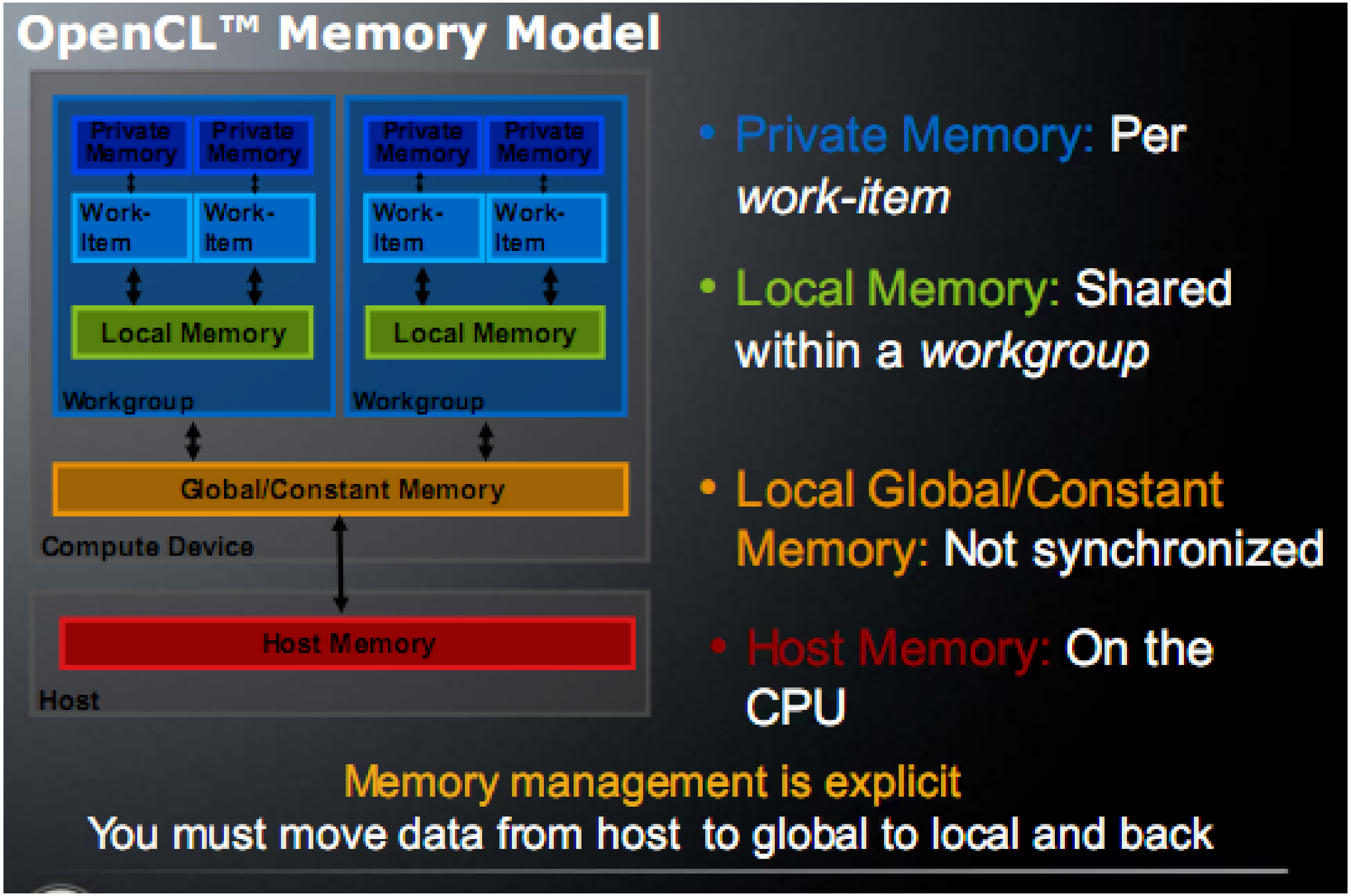


```
__kernel void
sum(__global const float *a,
__global const float *b,
__global float *answer)
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```

```
main( )
{
    read_data_from_file( ... );
    maninpulate_data( ... );

    clEnqueueWriteBuffer( ... );
    clEnqueueTask( ..., my_kernel, ...);
    clEnqueueReadBuffer( ... );

    display_result_to_user( ... );
}
```

- The user only needs to program with OpenCL, leaving the tedious details to the OpenCL SDK.

OpenCL



OpenCL™ Memory Model

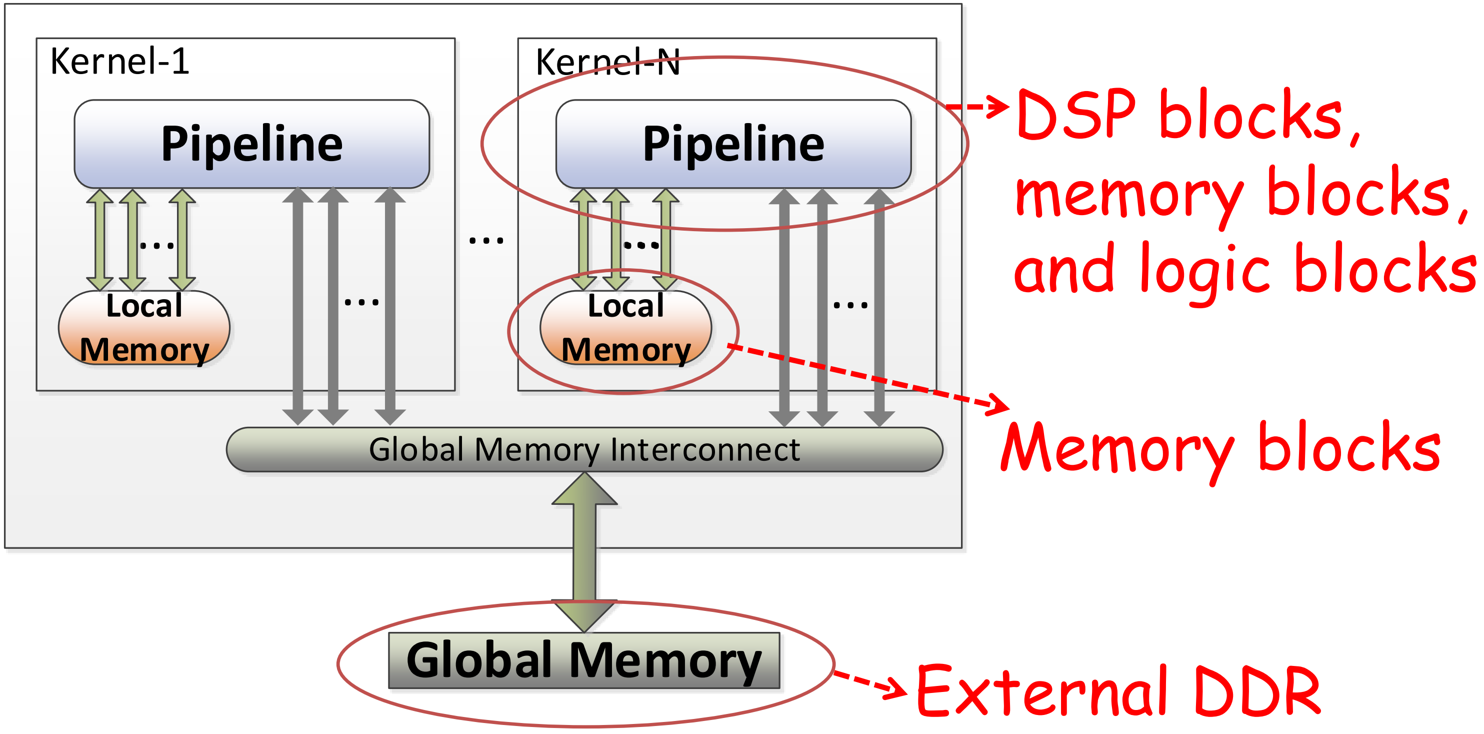
- Private Memory: Per *work-item*
- Local Memory: Shared within a *workgroup*
- Local Global/Constant Memory: Not synchronized
- Host Memory: On the CPU

Memory management is explicit

You must move data from host to global to local and back

From: http://dna.cs.byu.edu/CS484/lectures/opencl-20training.pdf

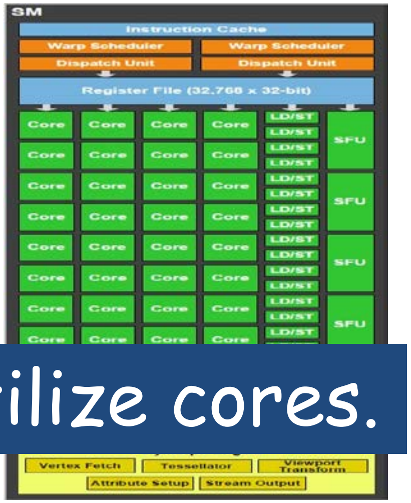
"Architectural Evolution" of FPGAs: From OpenCL's Perspective



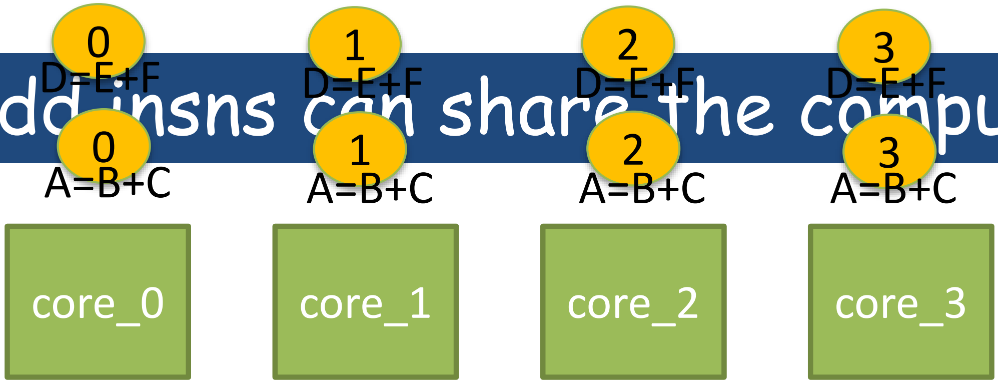
- Software-centric → FPGA as a parallel architecture.
- Users can program with OpenCL. ☺

OpenCL on GPU

```
__kernel void vectorAdd (global int *A, *B, *C, *D, *E, *F)
{
   int gid  = get_global_id(0);
   A[gid]  = B[gid] + C[gid];
   D[gid]  = E[gid] + F[gid];
}
```



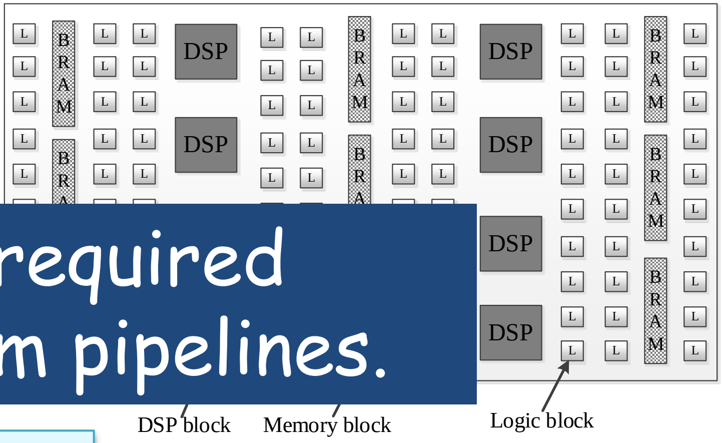
Sufficient threads are required to utilize cores.

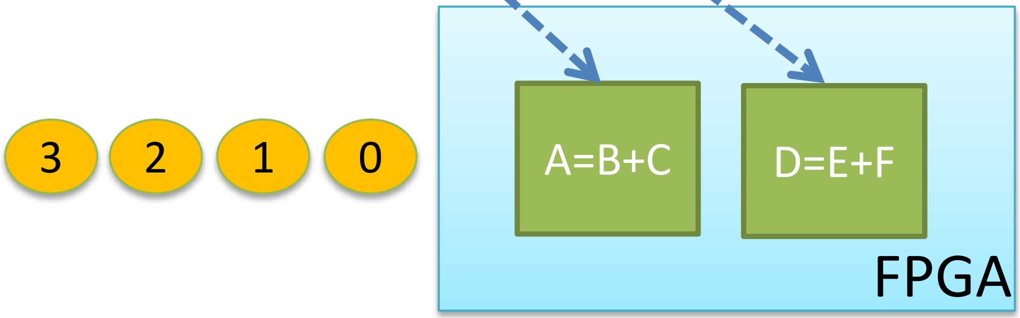

Both add insns can share the computing cores.

OpenCL on FPGA

```
__kernel void vectorAdd (global int *A, *B, *C, *D, *E, *F)
{
    int gid  = get_global_id(0);
    A[gid]  = B[gid] + C[gid];
    D[gid]  = E[gid] + F[gid];
}
```

Enough threads are required to fully utilize custom pipelines.





Instructions mean real circuits on FPGA

→ Pipeline parallelism.

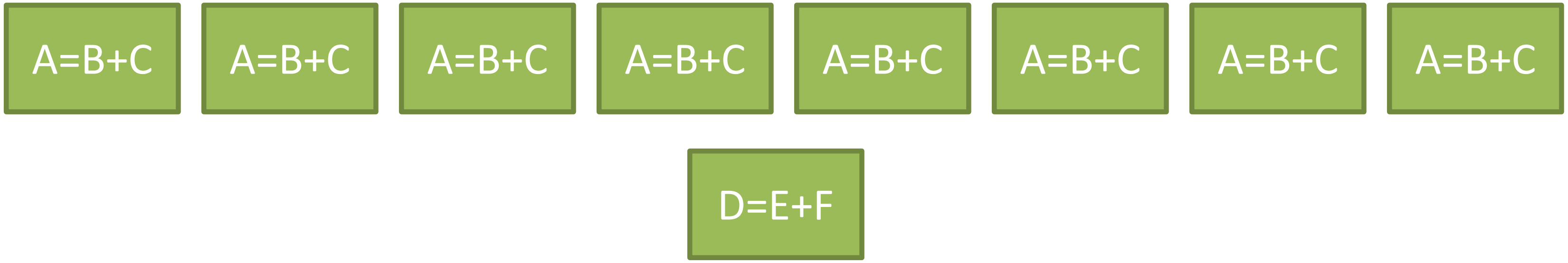
Impact of attributes

```
__kernel void vectorAdd_int(__global int *A, *B, *C, *D, *E, *F)
{
   int gid = get_global_id(0);

   #pragma unroll 8
   for (i
      A[(

   D[gid] = E[gid] + F[gid];
}
```

For GPU: larger code size, less dependency between instructions.

Instructions (with attributes) mean real circuits on FPGA.

For FPGA: 8x real circuits for addition instruction, 8 additions per cycle.



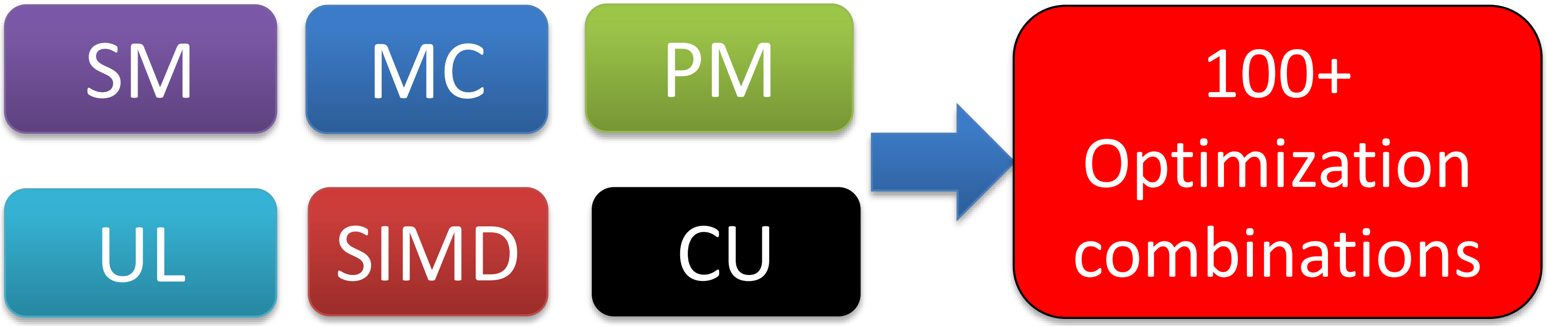
Outline

- Background
  - FPGA
  - OpenCL SDK for FPGAs
- A Performance Analysis Framework
- Case Study: Database Systems
- Conclusion

Motivating Questions

- OpenCL has improved the programmability of new generation FPGAs.
- Good programmability does NOT necessarily mean good performance.
- How can we tune the performance of an OpenCL program on FPGAs?

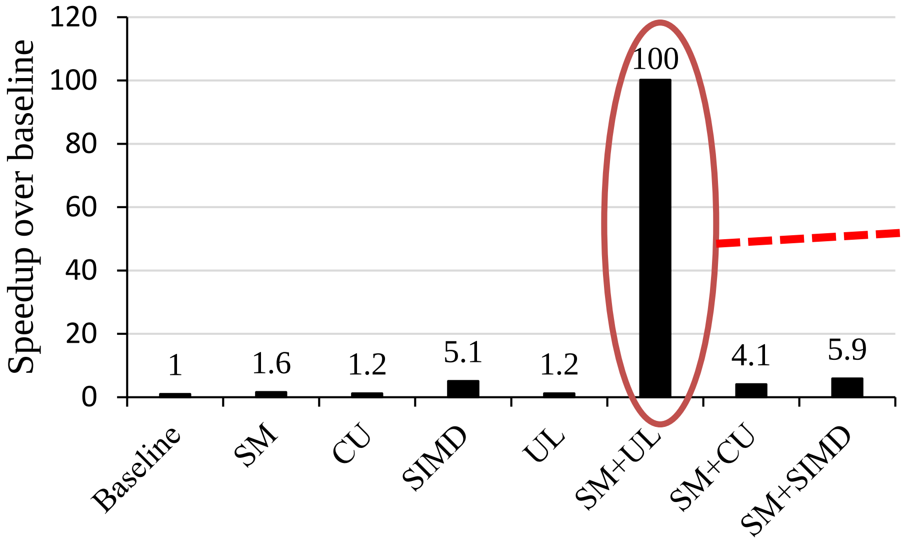
Impact of Optimization Combinations



SM MC PM

UL SIMD CU

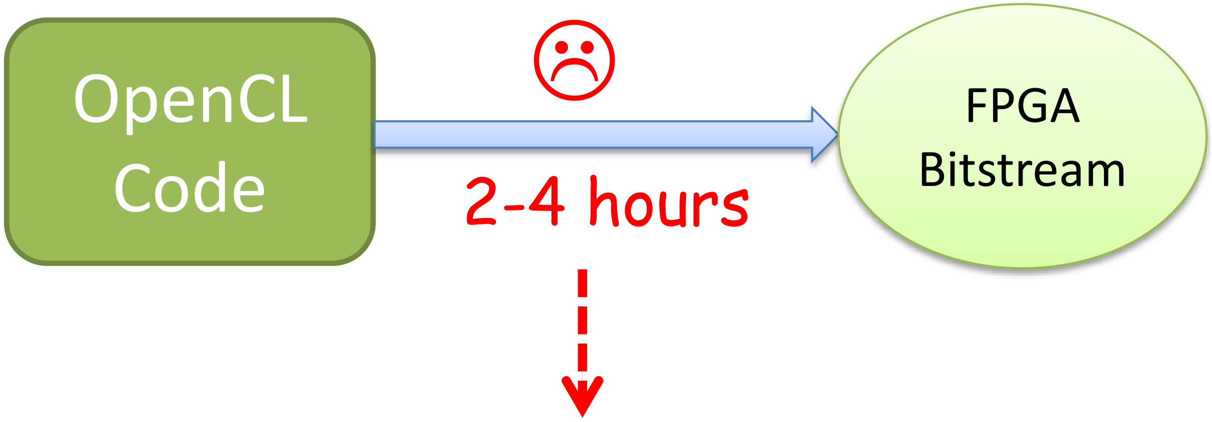
100+ Optimization combinations



Different optimization combinations can significantly affect performance.

- Local Memory (SM), Memory Coalescing (MC), Private Memory (PM),
- Loop Unrolling (UL), Kernel Vectorization (SIMD), Kernel Pipeline Replication (CU)

Challenge from Long Synthesis Time



OpenCL Code

☹

2-4 hours

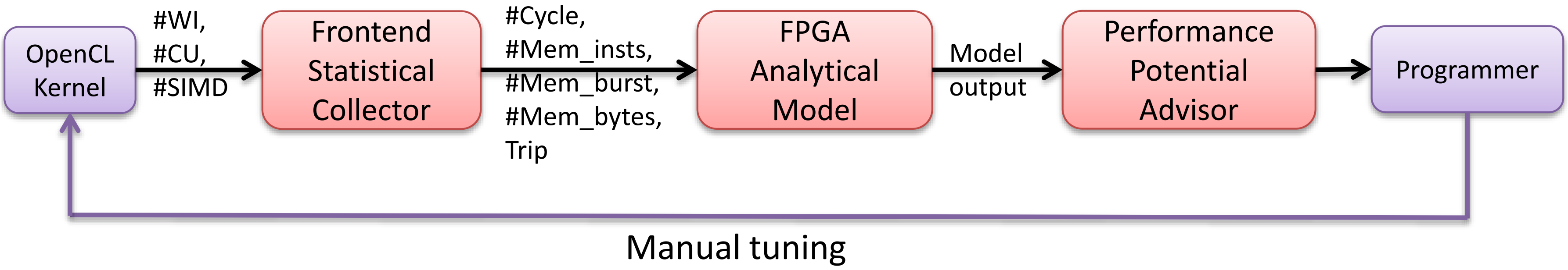
FPGA Bitstream

It takes too much time to evaluate all the optimization combinations on real FPGAs.

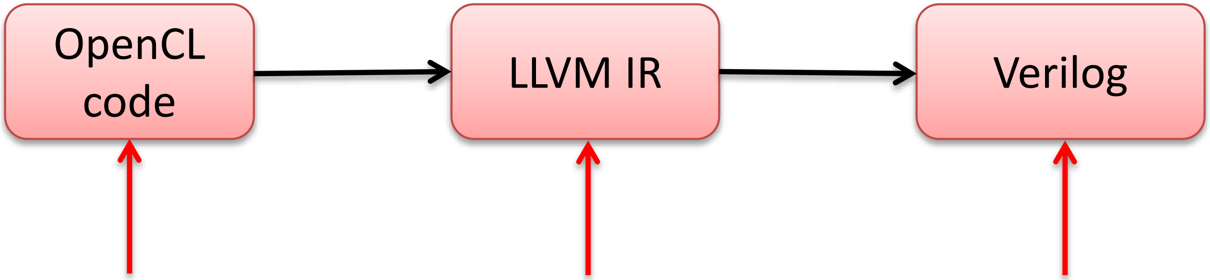
A tool for estimating optimization optimizations for OpenCL kernel is needed.

Our Solution: A Performance Analysis Framework

- We propose a performance analysis framework to provide performance guidance on FPGA
  - Frontend Collector: Obtain necessary OpenCL information (LLVM IR) about each basic block
  - Analytical model: Estimate the performance of the OpenCL kernel and generate model output
  - Performance Potential Advisor: Convert program analysis information into user-friendly metrics



OpenCL Compiler on FPGA



OpenCL code

LLVM IR

Verilog

Too high-level, hard to capture all the cases

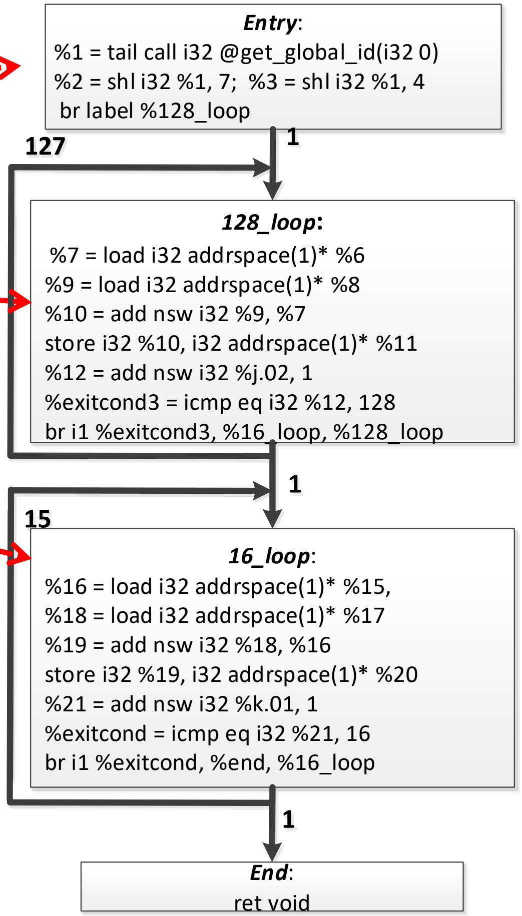
Fixed intermediate instructions, Plenty of static analysis tools

Too low-level, large HDL code size

Our performance analysis framework is based on LLVM IR.

LLVM IR View of OpenCL Kernel

```
__kernel void vectorAdd (__global int *A, *B, *C, *D, *E, *F)
{
   int gid = get_global_id(0);

   for (int j = 0; j < 128; j++)
      A[(gid<<7)+j]  = B[(gid<<7)+j] + C[(gid<<7)+j];

   for (int k = 0; k < 16; k++)
      D[(gid<<4)+k] = E[(gid<<4)+k] + F[(gid<<4)+k];
}
```



```
Entry:
%1 = tail call i32 @get_global_id(i32 0)
%2 = shl i32 %1, 7;  %3 = shl i32 %1, 4
br label %128_loop
```

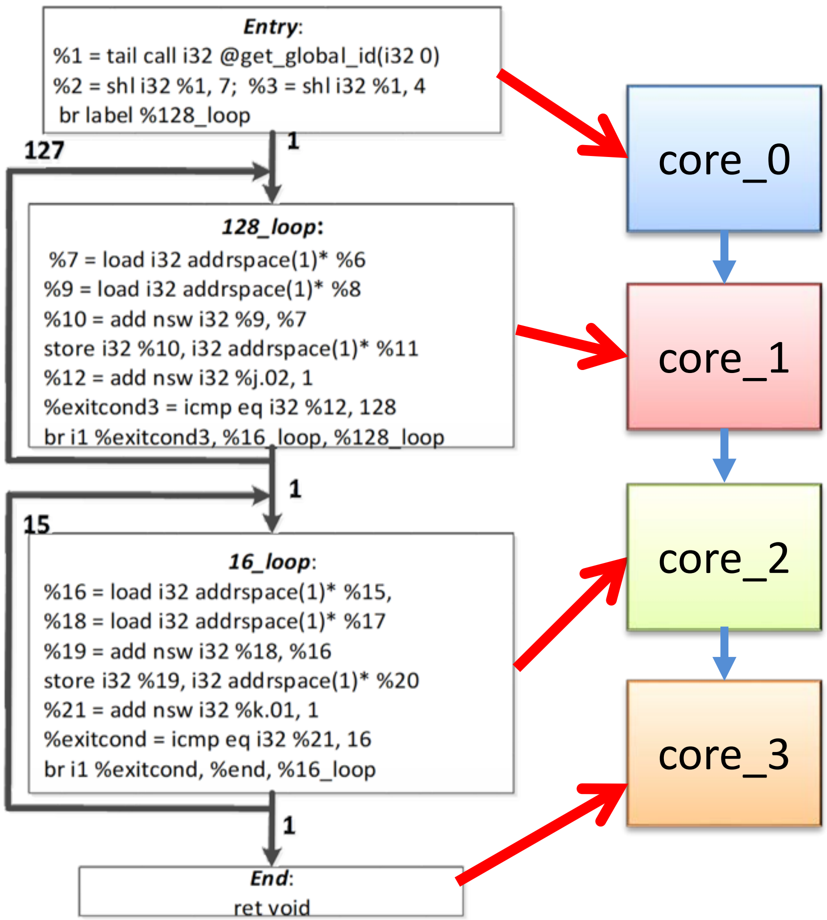
```
128_loop:
%7 = load i32 addrspace(1)* %6
%9 = load i32 addrspace(1)* %8
%10 = add nsw i32 %9, %7
store i32 %10, i32 addrspace(1)* %11
%12 = add nsw i32 %j.02, 1
%exitcond3 = icmp eq i32 %12, 128
br i1 %exitcond3, %16_loop, %128_loop
```

```
16_loop:
%16 = load i32 addrspace(1)* %15,
%18 = load i32 addrspace(1)* %17
%19 = add nsw i32 %18, %16
store i32 %19, i32 addrspace(1)* %20
%21 = add nsw i32 %k.01, 1
%exitcond = icmp eq i32 %21, 16
br i1 %exitcond, %end, %16_loop
```

```
End:
ret void
```

Basic block: a group of instructions in a contiguous sequence.

Basic Block → Heterogeneous core



```
Entry:
%1 = tail call i32 @get_global_id(i32 0)
%2 = shl i32 %1, 7;  %3 = shl i32 %1, 4
 br label %128_loop
```

```
128_loop:
 %7 = load i32 addrspace(1)* %6
%9 = load i32 addrspace(1)* %8
%10 = add nsw i32 %9, %7
store i32 %10, i32 addrspace(1)* %11
%12 = add nsw i32 %j.02, 1
%exitcond3 = icmp eq i32 %12, 128
br i1 %exitcond3, %16_loop, %128_loop
```

```
16_loop:
%16 = load i32 addrspace(1)* %15,
%18 = load i32 addrspace(1)* %17
%19 = add nsw i32 %18, %16
store i32 %19, i32 addrspace(1)* %20
%21 = add nsw i32 %k.01, 1
%exitcond = icmp eq i32 %21, 16
br i1 %exitcond, %end, %16_loop
```

```
End:
ret void
```

A basic block == a core

Cores run concurrently

Keeping the load balance among cores is critical.

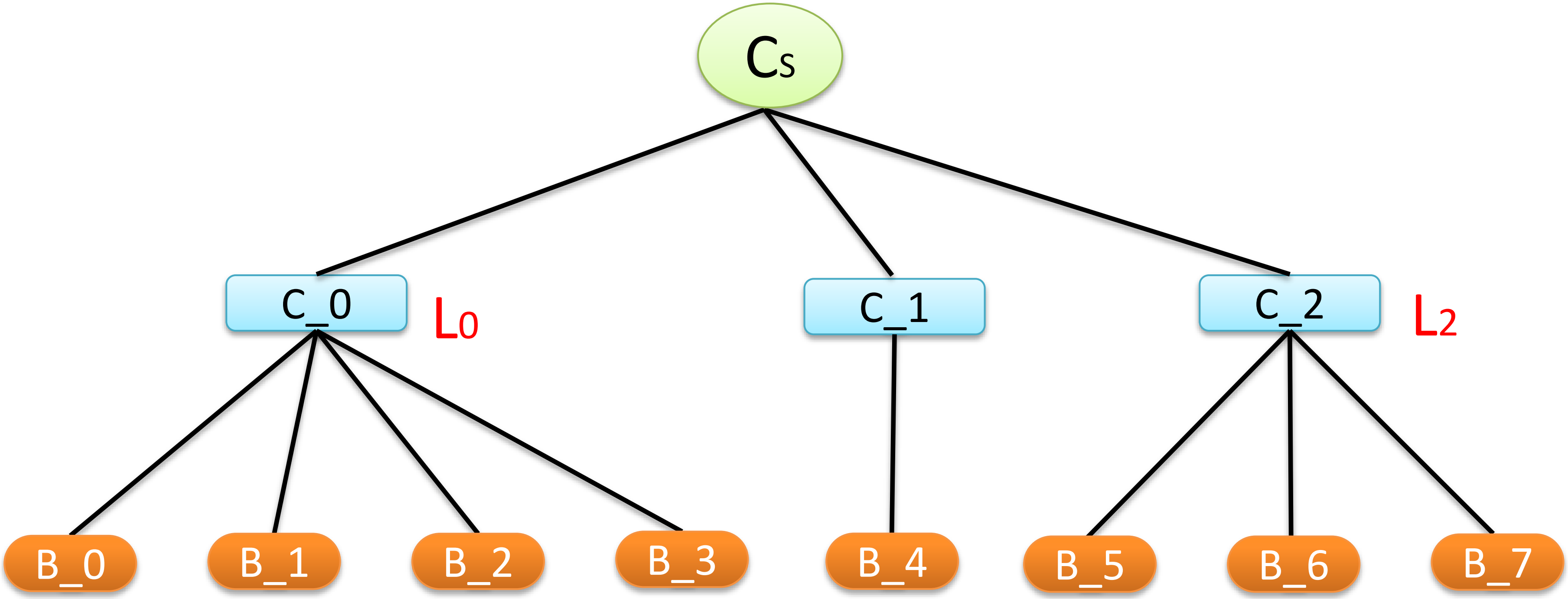
Frontend Collector

- Standard inputs:
  - Frequency and resource utilizations of logic block, memory block, DSP block (from compilation report).
  - Memory width and memory channels (from vendor).
  - Number of input work items (from host code).

- Static inputs of basic block with **UL** and **SIMD** (LLVM::BasicBlock):
  - Memory instructions, average memory bytes and burst length.
  - Critical path for computing instructions.

- Run-time Inputs (LLVM:: LoopInfo):
  - Loop hierarchy.
  - Trip counter for each loop (Run same OpenCL code on GPU).

Performance Analytical Model



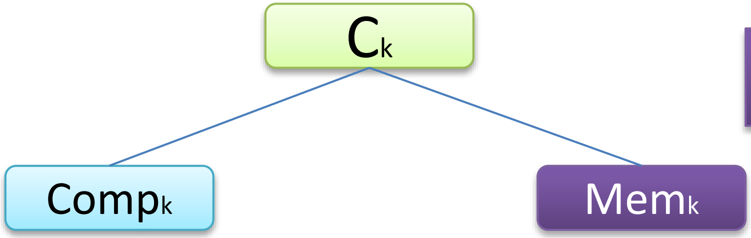
Our model follows a down-up approach.

B_0--B_7: Basic blocks of the kernel (leaf nodes)

C_0--C_2: LLVM non-leaf nodes

C_s : Cycles to execute the input kernel on FPGA

Evaluation of Basic Block



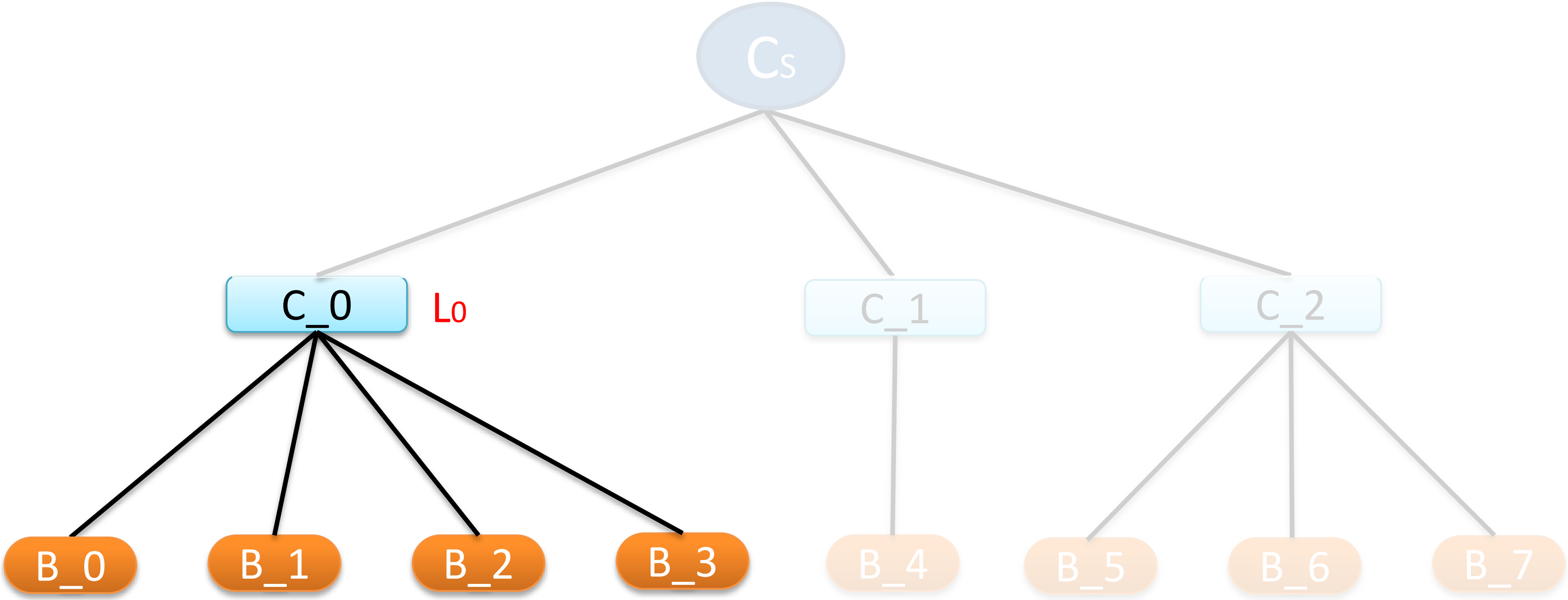
$$C_k = Max(Comp_k, Mem_k)$$

- $Comp_k$: Computation cycles of the basic block k
- Mem_k : Memory cycles of the basic block k

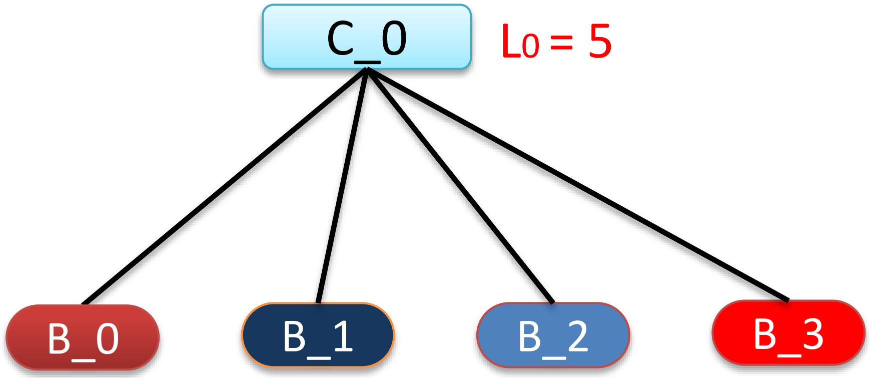
Each basic block has its own computing pipeline.

All the basic blocks share the memory bandwidth.

Evaluation of C_0

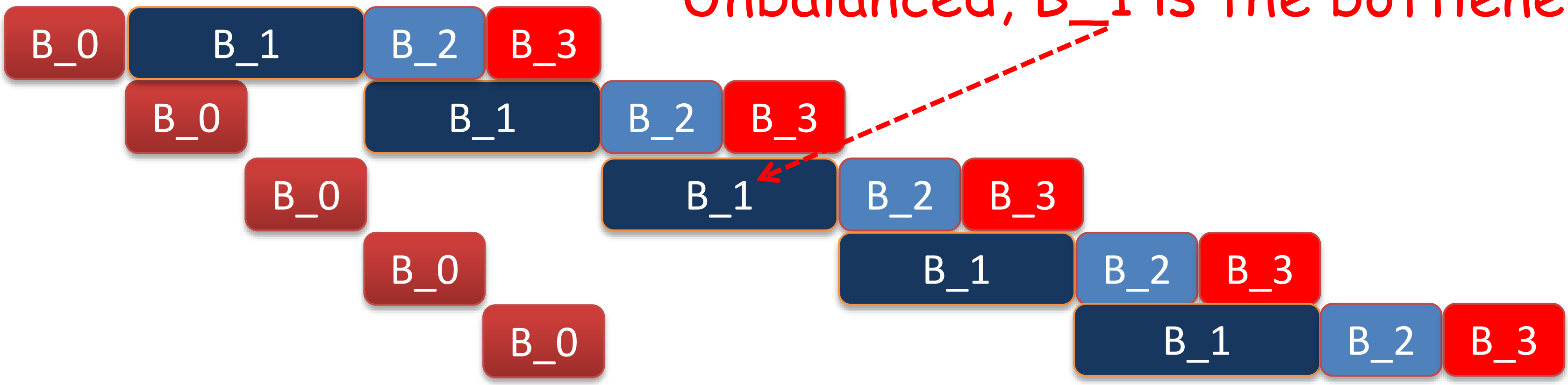


Evaluation of C_0



$L_0 = 5$

Unbalanced, B_1 is the bottleneck!



C_0 = B_0 + 5*B_1 + B_2 + B_3

Performance Potential Advisor

- B_{comp}: high → FPGA resource still available.

- B_{mem}: high → low global memory bandwidth utilization.

- $B_{balance}$: high → unbalancing among basic blocks.

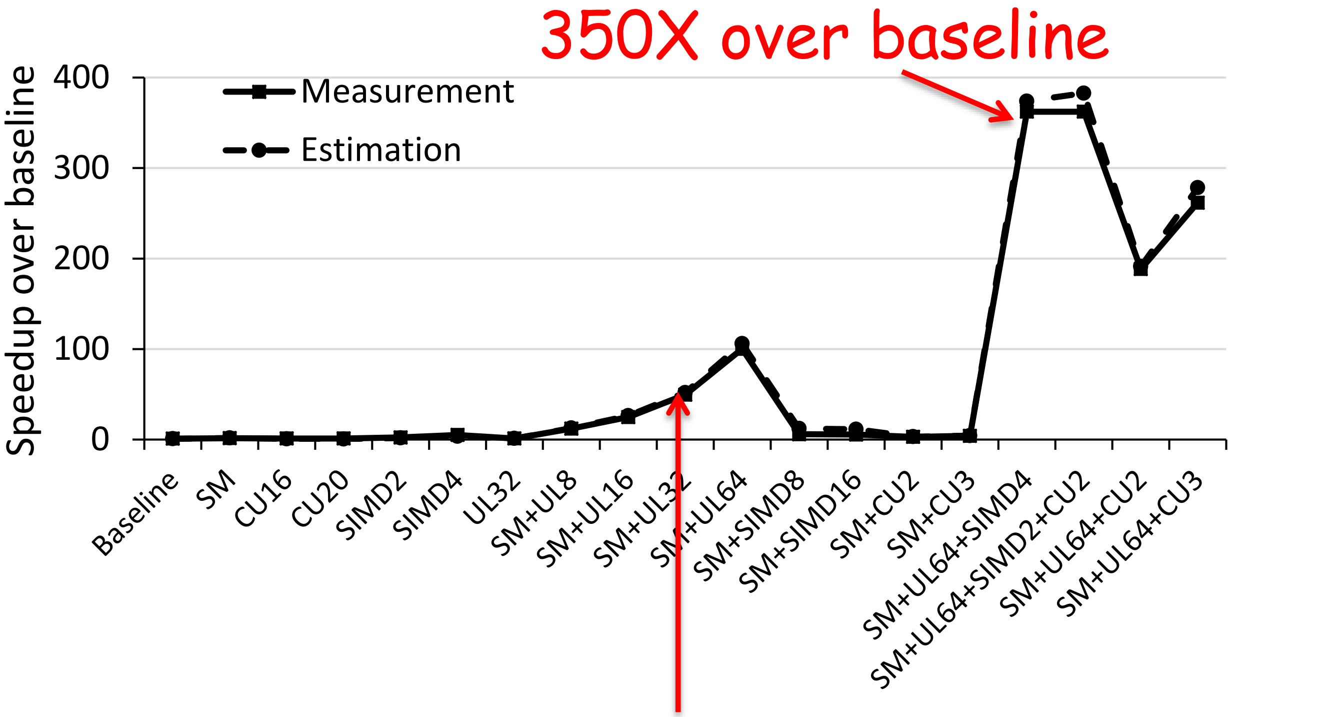
- B_{itpp}: high → more threads required to fully utilize pipeline.

Experimental Setup

- **Platform:**
  - Terasic's DE5-Net board: 4GB 2-bank DDR3 and Altera Stratix V A7, with Altera OpenCL SDK version 14.0 (now Intel).
  - Connected to the host with PCI-e bus

Application	Data Size
Matrix Multiplication (MM)	2048*2048 matrices
K-Means, K=128, 8 features (KM)	32M points
Word Count (WC)	1500MB text file
Similarity Scope (SS)	8000 files each with 8000 features

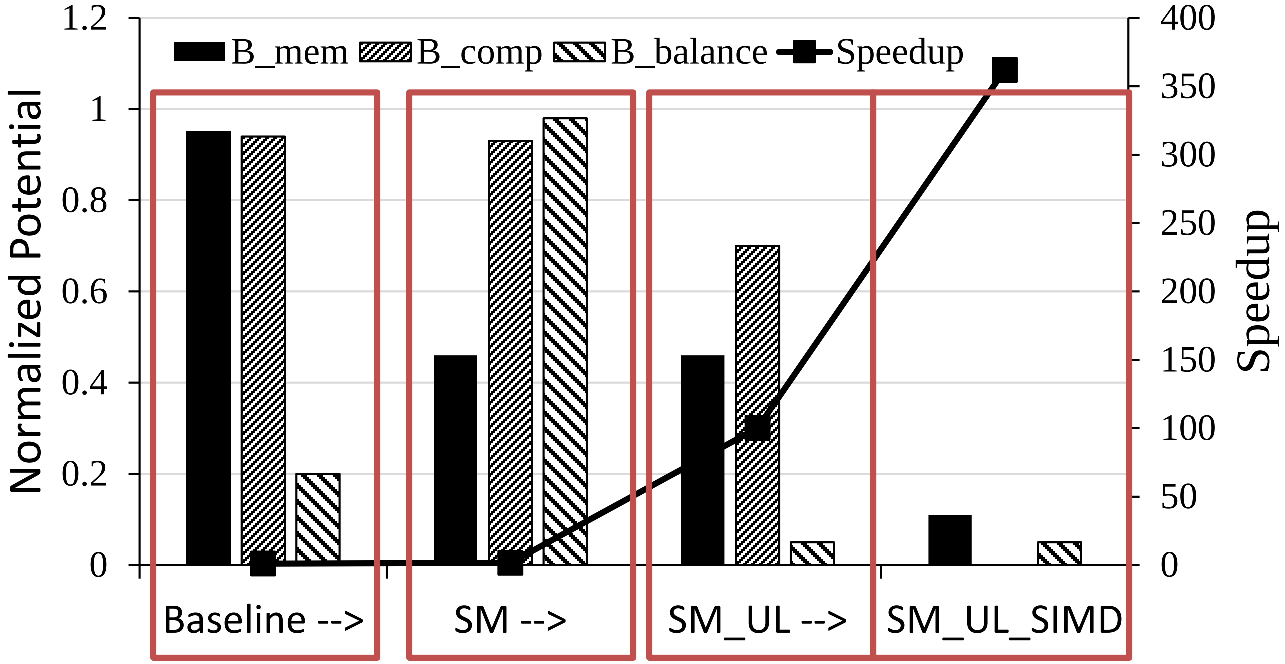
Analytical Model Evaluation (MM)



350X over baseline

Our estimation can capture the performance trend of 18 optimization combinations.

Code Tuning Steps (MM)



- The tuning rule: Each step aims to decrease the metric with largest value.

Comparison with Altera OpenCL SDK

Speedup over baseline	MM	KM	WC	SS
Altera -O3 approach	0.7	4.4	0.93	1
Our framework	362.4	293.4	11.4	9.4

Summary

- Tuning OpenCL code on FPGAs is still an open problem, since FPGA is significantly different from GPU.
- We use static and dynamic analysis to develop an performance analytical framework.
- Our framework can guide the programmer to effectively tune the code on FPGA.

Outline

- Background
  - FPGA
  - OpenCL SDK for FPGAs
- A Performance Analysis Framework
- Case Study: Database Systems
- Conclusion

Problem

- *OmniDB* [1]: State-of-the-art OpenCL-based query processor on CPU/GPU.
  - Kernel-based execution
  - Common optimization methods
  - Cost-based approach to schedule
- How *OmniDB* performs on OpenCL-based FPGAs?

[1] *Shuhao Zhang and et al. OmniDB: Towards Portable and Efficient Query Processing on Parallel CPU/GPU Architectures, VLDB'13.*

Challenge (Large Exploration Space)

- A single SQL query can have many possible query execution plans on FPGAs.
  – Each query has multiple operators, and each operator consists of multiple OpenCL kernels.
  – Each OpenCL kernel can have different FPGA-specific optimization combinations.

- We also consider another dimension of using multiple FPGA images.

Observation

- There is an FPGA-specific trade-off between the following two factors.
  - Optimization combination for each kernel
  - Reconfiguration overhead

More aggregative optimizations for each kernel ☺ →

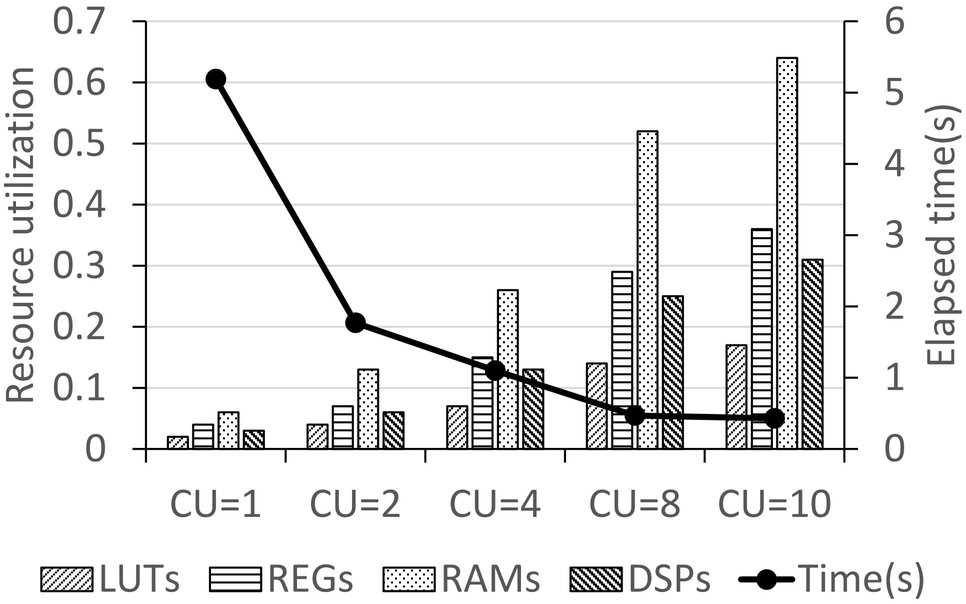
More resources for each kernel ☺ →

More resources for the entire query →

More FPGA images ☹ →

Higher FPGA reconfiguration overhead ☹

Impact of Optimization Combination



Time and resource utilization of *scanLargeArrays* kernel (@prefix scan) with 128M tuples

More aggregative optimizations →

More resource utilization →

Higher performance

FPGA Reconfiguration Overhead

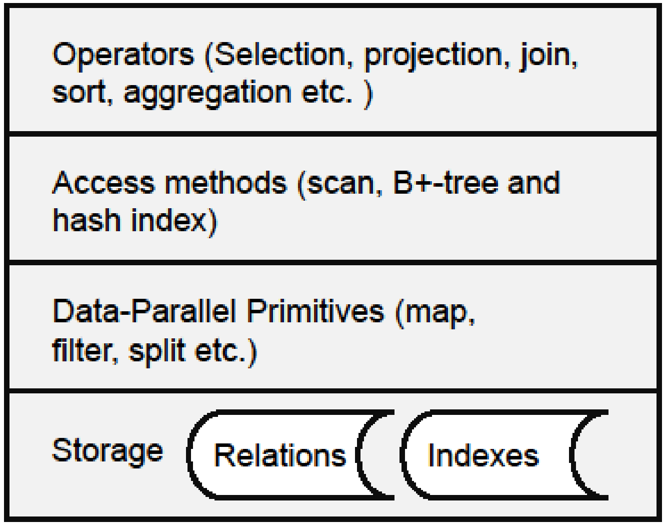
- According to Altera, FPGA reconfiguration overhead contains three sources.
  - Transfer the active contents (memory footprint) from FPGA memory to host memory via PCIe (roughly 2GB/s)
  - Fully reconfigure the FPGA (roughly 1914.6ms).
  - Transfer the active contents from host memory to FPGA memory via PCIe (roughly 2GB/s)

FPGA reconfiguration overhead is significant in the current FPGA board.

Our Approach

- Query processor: accelerated with FPGA-specific optimizations
- FPGA-specific cost model: to determine the optimal query plan for the input query

Query Processor (Operator Kernel Level)



Operators (Selection, projection, join, sort, aggregation etc.)

Access methods (scan, B+-tree and hash index)

Data-Parallel Primitives (map, filter, split etc.)

Storage

- The layered design of query processor contains four operators (constituting the SQL query).
  - Selection (5 operator kernels)
  - Order-by (2 operator kernels)
  - Grouping and Aggregation (7 operator kernels)
  - Join (2 operator kernels)

Experimental Setup

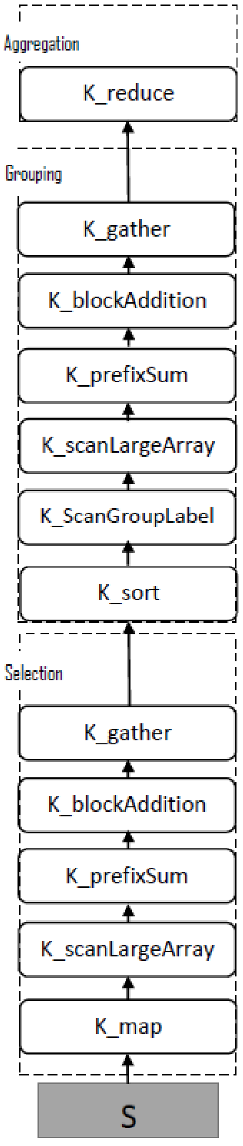
- Platform:
  - Terasic's DE5-Net board: Altera Stratix V A7 and 4GB 2-bank DDR3
  - PCI-e 2.0 (X8)
  - Altera OpenCL SDK version 14.0
- Workloads:
  - Four queries (Q1, Q2, Q3 and Q4)
  - Tuple format: <key, payload>. Both keys and payloads are 4-bytes.

We use Q3 for example.

Details of Q3

- SQL query:
  - **SELECT** *S.key*, SUM(*S.payload*)
FROM *S*
WHERE Lo ≤ *S.paylaod* ≤ Hi
GROUP BY *S.key*

Q3: 12 operator kernels



Q3

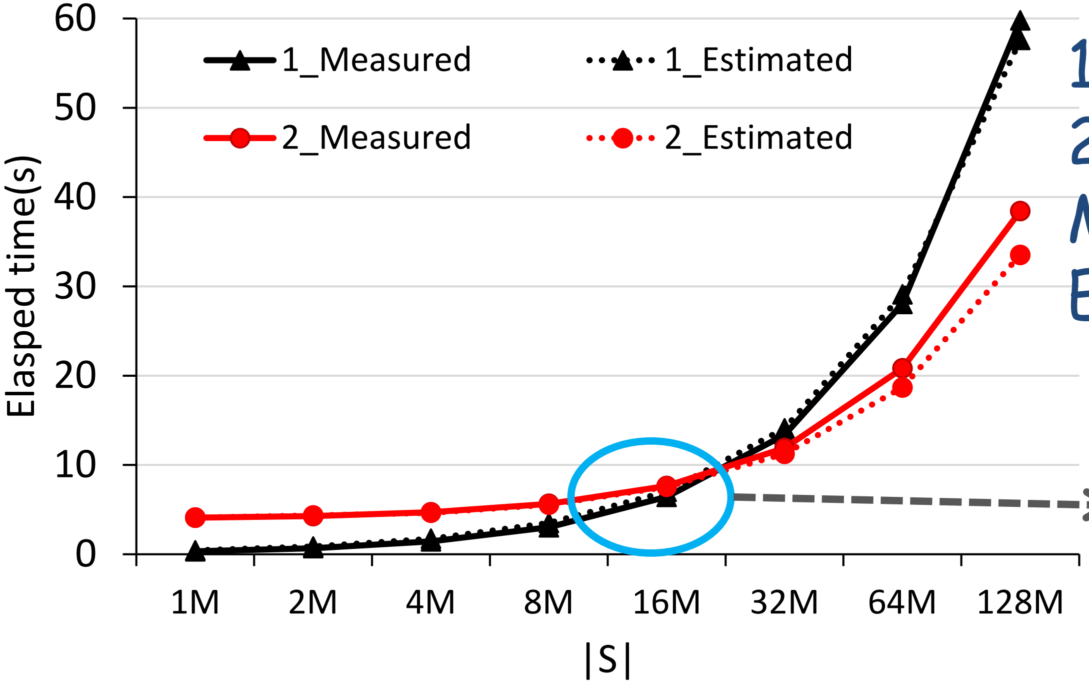
Generation of Execution Plans

Execution plan 1					
FPGA image	LUTs	REGs	RAMs	DSPs	Freq.
Estimated	151460	339134	2175	42	198
Measured	154509	283131	1973	34	233

Execution Plan 2					
FPGA image 1	LUTs	REGs	RAMs	DSPs	Freq.
Estimated 1	187738	349051	2416	72	182M
Measured 1	184082	334093	2342	72	192.5M
FPGA image 3	LUTs	REGs	RAMs	DSPs	Freq.
Estimated 3	155187	294559	1950	90	223M
Measured 3	171434	348651	2112	90	203M

Our cost model can roughly predict the resource utilization and frequency of each FPGA image.

Break-even Point for Execution Plans



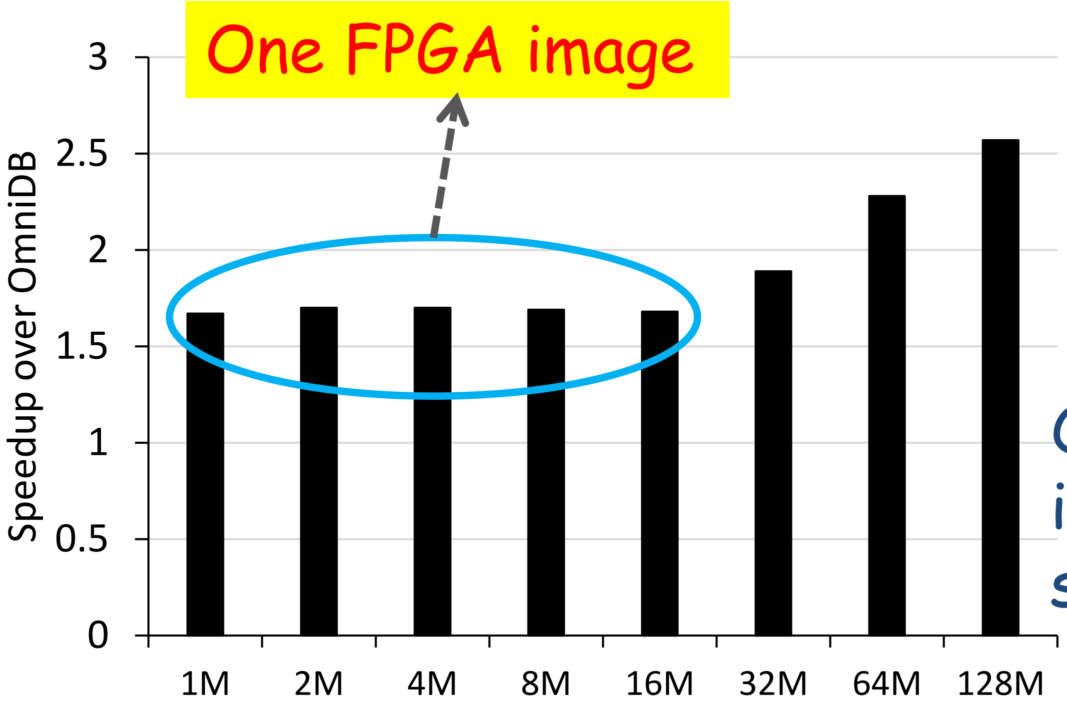
1: execution plan 1
2: execution plan 2
Measured: real FPGA
Estimated: cost model

Break-even point

Our cost model can roughly predict the performance for each execution plan.

Our cost model can recommend the optimal execution plan for different table sizes.

Comparison with OmniDB on FPGA

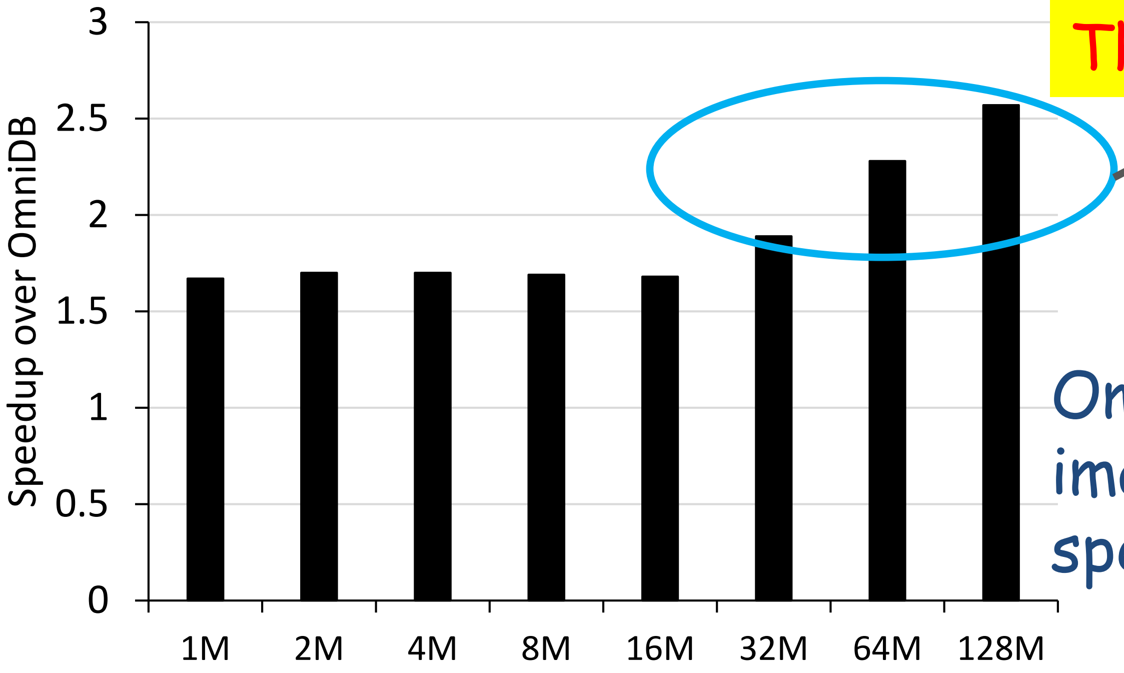


One FPGA image

OmniDB: one FPGA image without FPGA-specific optimizations

FPGA reconfiguration overhead > Benefit from the reduced execution time (more aggregative optimizations for each involved kernel.

Comparison with OmniDB on FPGA



Three FPGA images

OmniDB: one FPGA image without FPGA-specific optimizations

FPGA reconfiguration overhead <
Benefit from the reduced execution time

Outline

- Background
  - FPGA
  - OpenCL SDK for FPGAs
- A Performance Analysis Framework
- Case Study: Database Systems
- Conclusion

Conclusions

- The OpenCL support in new generation FPGAs has brought significant technical challenges and opportunities.
- Our performance analysis tool and case study demonstrate the importance of system optimizations and performance tunings.
- Hardware and software co-design can enable many interesting systems and applications.

References

- Zeke Wangˆ, Johns Paul*, Hui Yan Cheahˆ, **Bingsheng He** and Wei Zhang. Accelerating Database Query Processing on OpenCL-based FPGAs. *FPL 2016: International Conference on Field Programmable Logic and Applications.*
- Zeke Wang^, **Bingsheng He**, Wei Zhang, Shunning Jiang. A Performance Analysis Framework for Optimizing OpenCL Applications on FPGAs. ***HPCA 2016***: *IEEE International Symposium on High Performance Computer Architecture* [53/240=22%]
- Zeke Wang^, **Bingsheng He**, Wei Zhang. A Study of Data Partitioning on OpenCL-based FPGA. *FPL 2015: International Conference on Field Programmable Logic and Applications.* **[Top-quality papers of FPL 2015]**
- Zeke Wang^, Shuhao Zhang*, **Bingsheng He**, Wei Zhang. *Melia: A MapReduce Framework on OpenCL-based FPGAs. IEEE TPDS: IEEE Transactions on Parallel and Distributed System (TPDS) Volume ??, Number ?, April 2015, pp. ???-???.*