Overlays: a solution paradigm for FPGA high-level design?

Tarek S. Abdelrahman

The Edward S. Rogers Department of
Electrical and Computer Engineering
University of Toronto

tsa@ece.utoronto.ca

Reconfigurable Systems on the Rise

- FPGAs are increasingly integrated in computing systems
  - Massive parallelism can lead to high performance
  - Lower power
  - Customizability

- Newer generation of high-performance systems integrate FPGAs with multicores, targeting data centers
  - Example systems from Intel, IBM and Xilinx
  - Used mainly by software developers

Reconfigurable Systems on the Rise

- FPGAs are increasingly integrated in computing systems
  - Massive parallelism can lead to high performance
  - Lower power
  - Customizability

- Newer generation of high-performance systems integrate FPGAs with multicores, targeting data centers
  - Example systems from Intel, IBM and Xilinx
  - Used mainly by software developers

FPGA Programmability Burdens

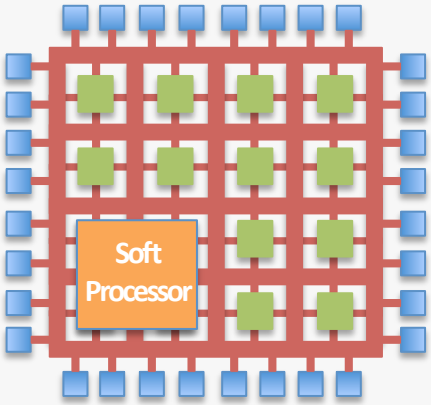
- FPGAs are programmed using a hardware design abstraction, which is foreign to the bulk of software developers
  - HDL, Timing, fitting, seed sweeps, etc.
- FPGA development tools lead to extremely long development cycles compared to their software counterparts
  - A large circuit can take days to compile (synthesis, place, route, time, etc.) and may need several compiles
- There is a pressing need to alleviate these burdens and make FPGA design accessible to software developers

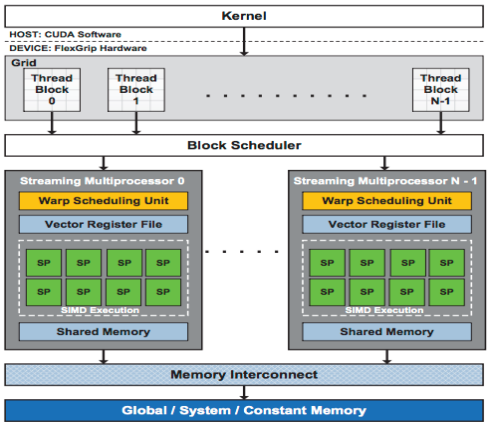
Tackling the Burden

- High-Level Synthesis (HLS)
  - Generated hardware increasingly competitive with HDL design
- High-level programming models
  - Dataflow model from Maxeler
- Nonetheless:
  - Developer remains exposed to various aspects of hardware design
  - Use of FPGA design tools is still required! ⇒ long development cycles

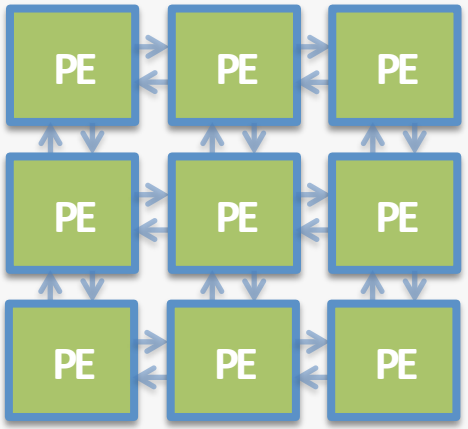
Overlays

- Pre-compiled FPGA circuits that are in themselves configurable/programmable, i.e., run-time configurable
  - Examples: soft processors, GPU-on-FPGA, mesh-of-FUs, etc.

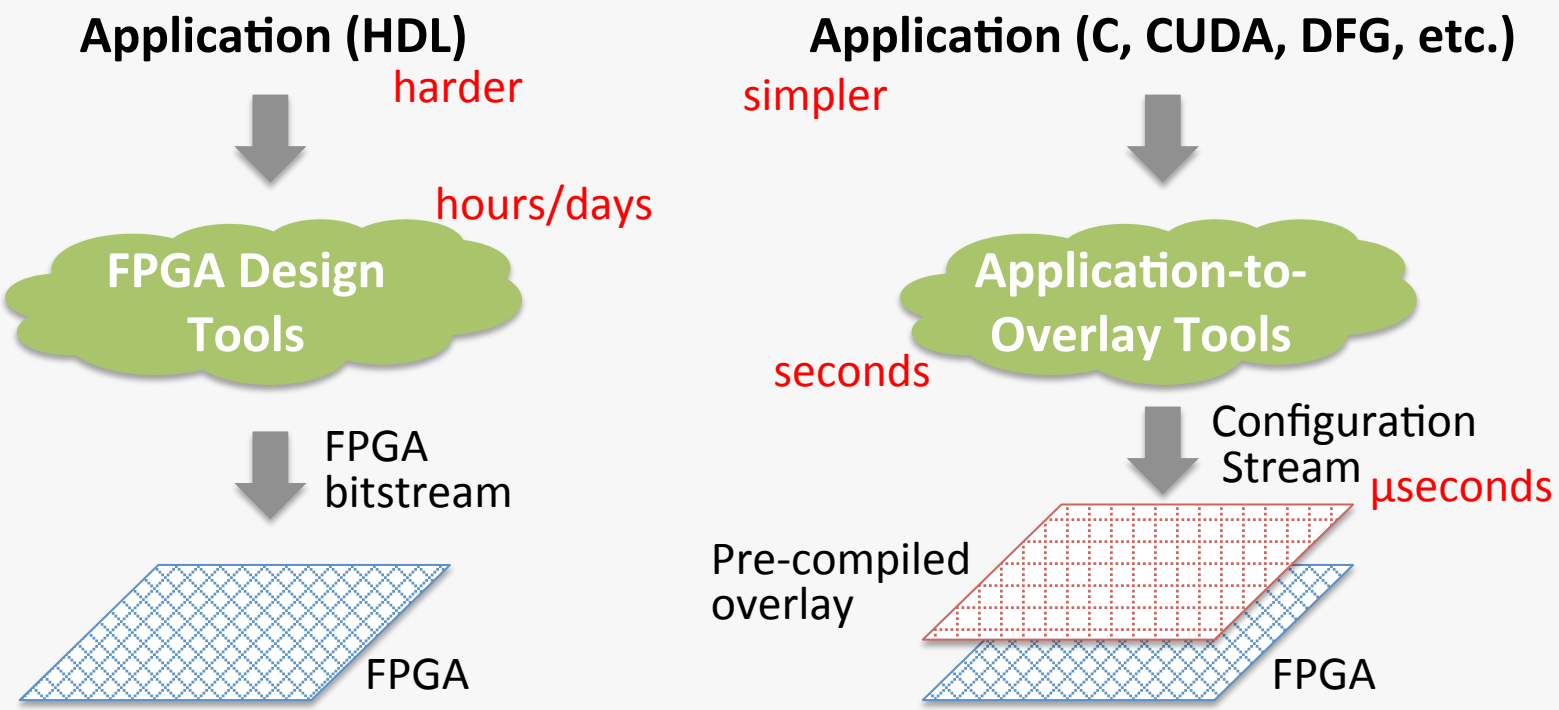




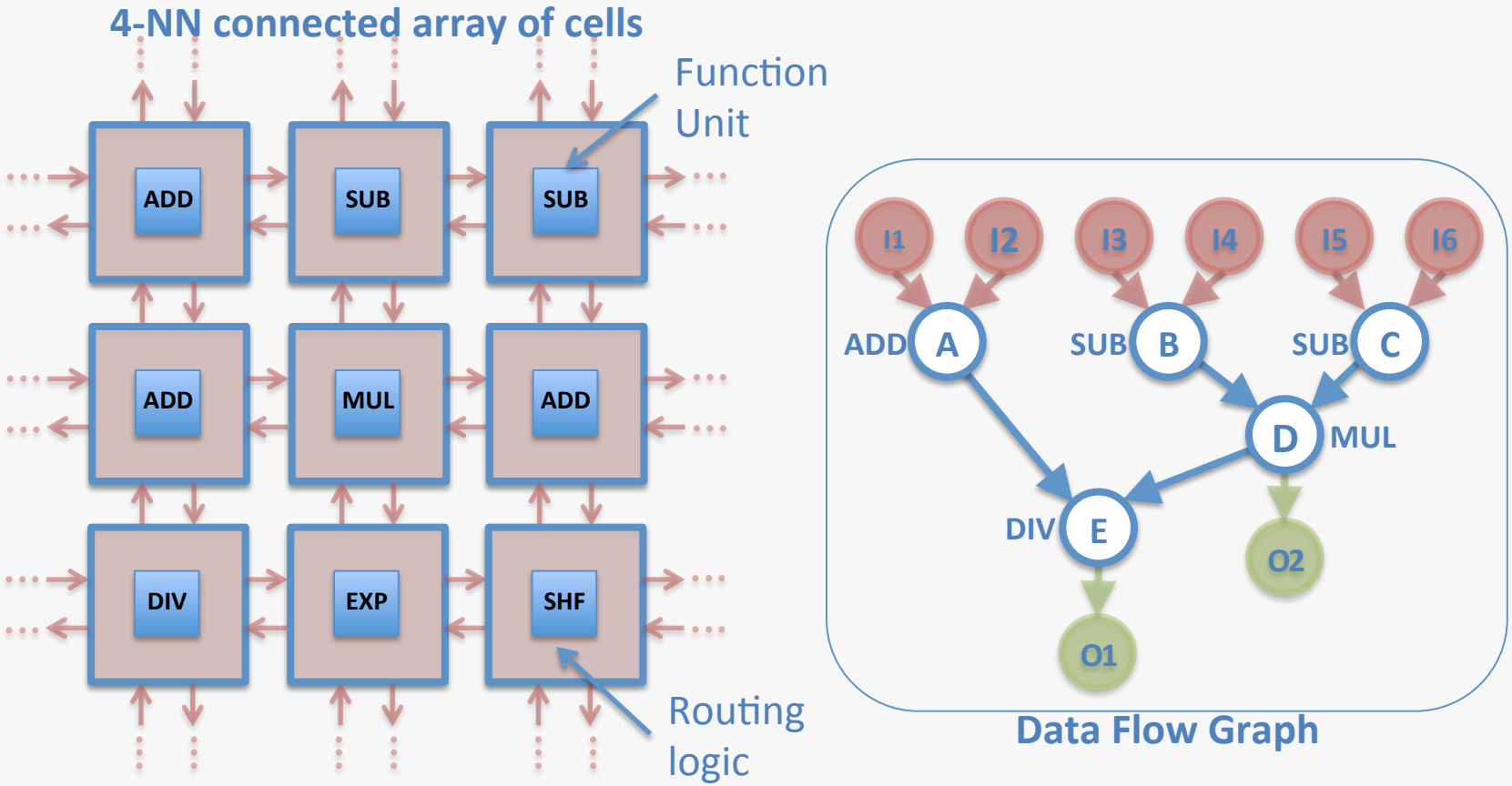
Source: Andryc et al: FlexGrip: A Soft GPGPU for FPGAs, FPT 13



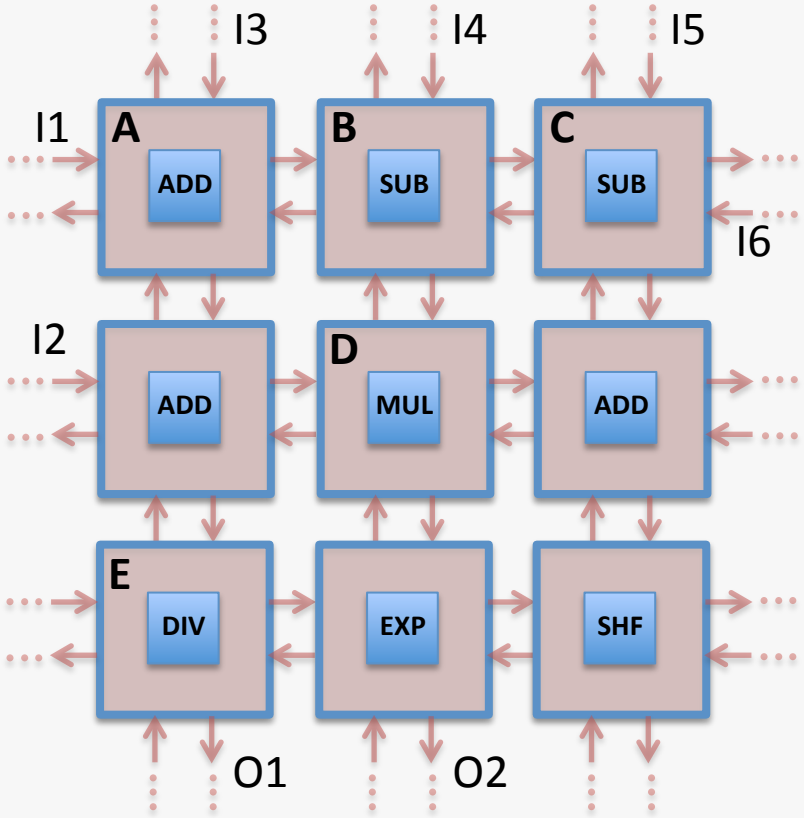
FPGA vs. Overlay Design Flows

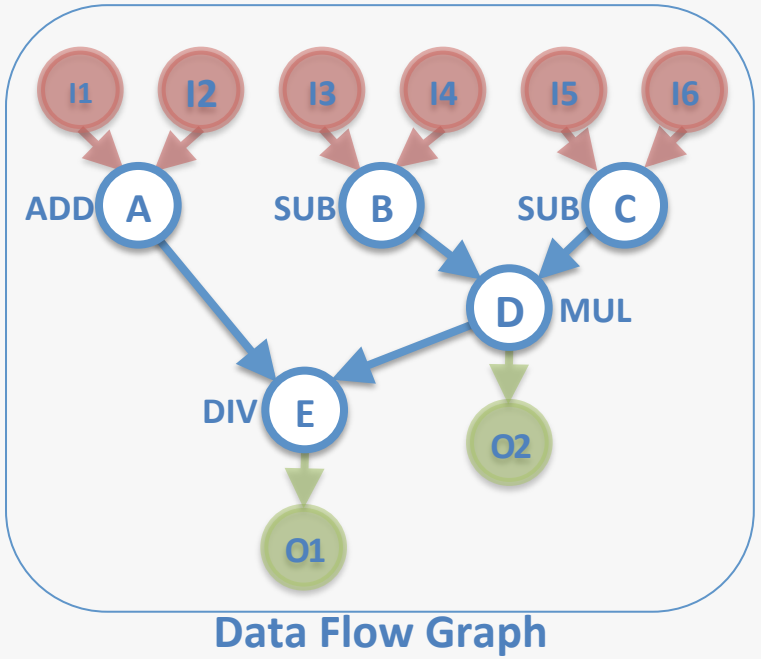


Mesh-of-FUs Overlays [FPL 2013]



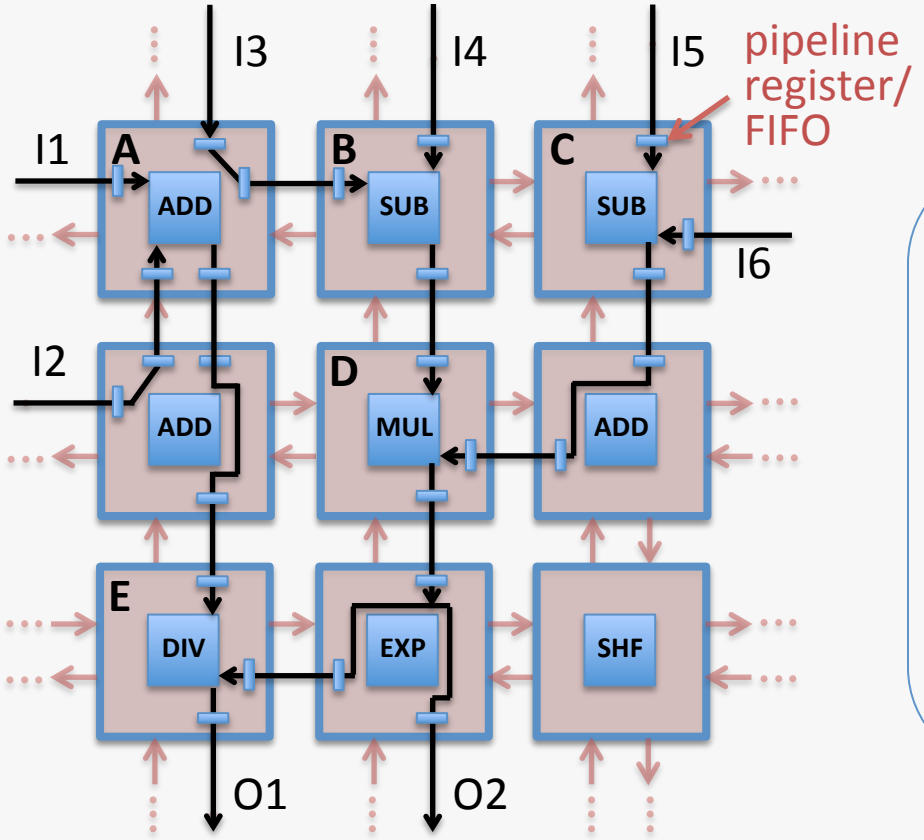
Mapping DFGs to Overlay – Place

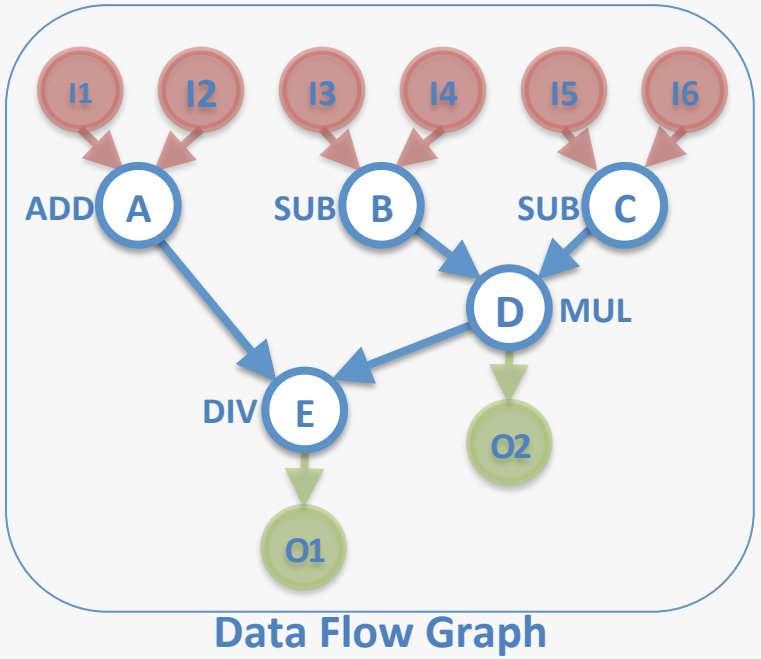




Data Flow Graph

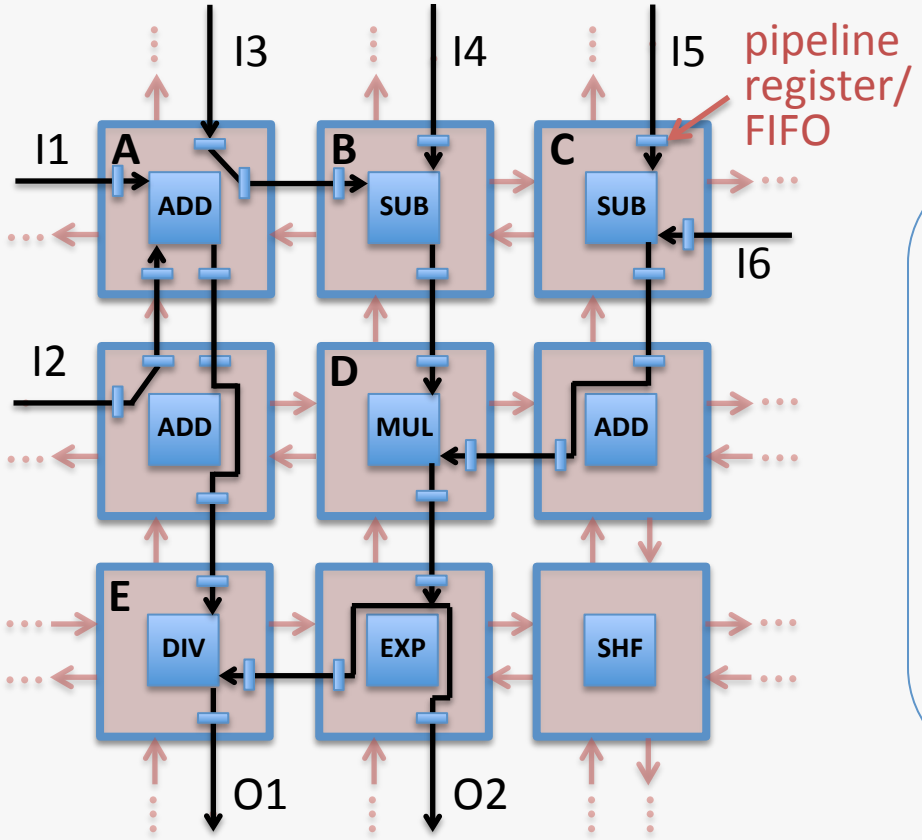
Mapping DFGs to Overlay – Route

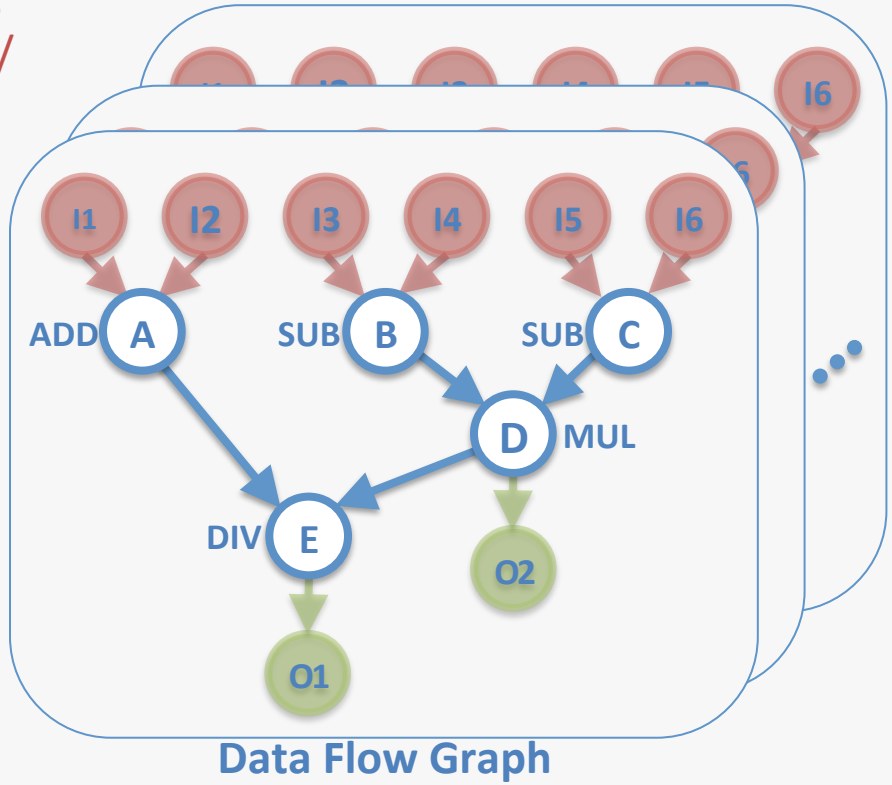




Data Flow Graph

Pipelined Execution





Data Flow Graph

Mesh-of-FUs Tools

- Application-to-overlay tool chain that:
  - Extracts DFG of bodies of parallel loops in C code
  - Places and routes the DFG nodes onto the overlay
    - Configures the switches to establish DFG connectivity
    - Generates the configuration stream of the overlay

High Performance with no Hardware Design

- Example mesh-of-FUs overlay on a Stratix IV [FPL 2013]
  - Single precision floating point operations
  - 288 cells implemented as an 18x16 mesh
  - f_{MAX} of 312 MHz and 32.4 GFLOPS peak (integer at 415 MHz)

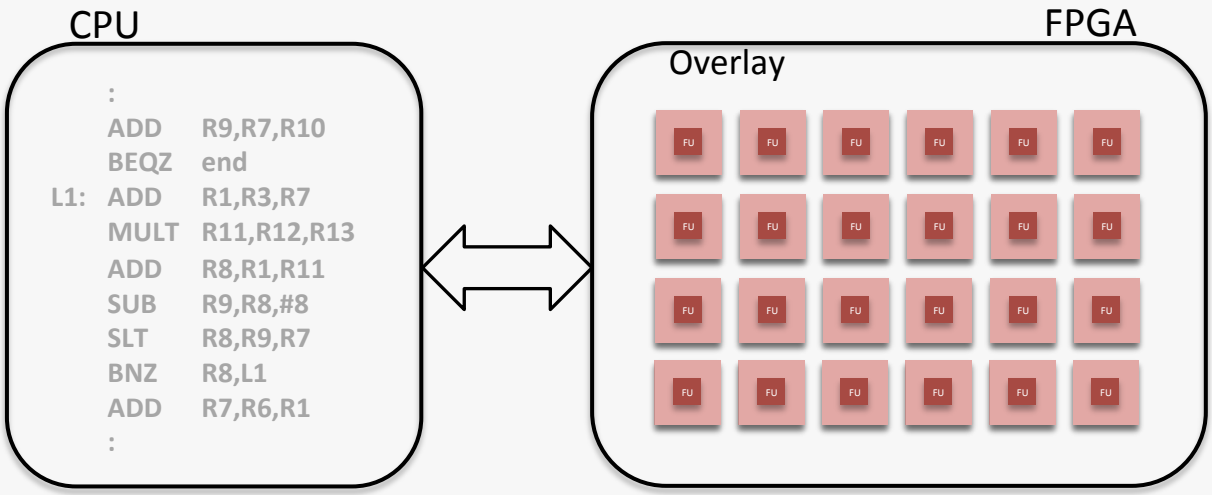
	Overlay			HDL	
DFG	Size (nodes)	GFLOPS	Compile Time (sec)	GFLOPS	Compile Time (sec)
n-Body	125	18.72	0.44	21.52	2724
BlackSholes	131	21.22	1.33	22.10	2508
MatMul	96	19.66	1.05	25.21	2045
MatMulAdd	114	22.46	3.80	28.79	919

- Others also report high performance results

Software-Friendly Target

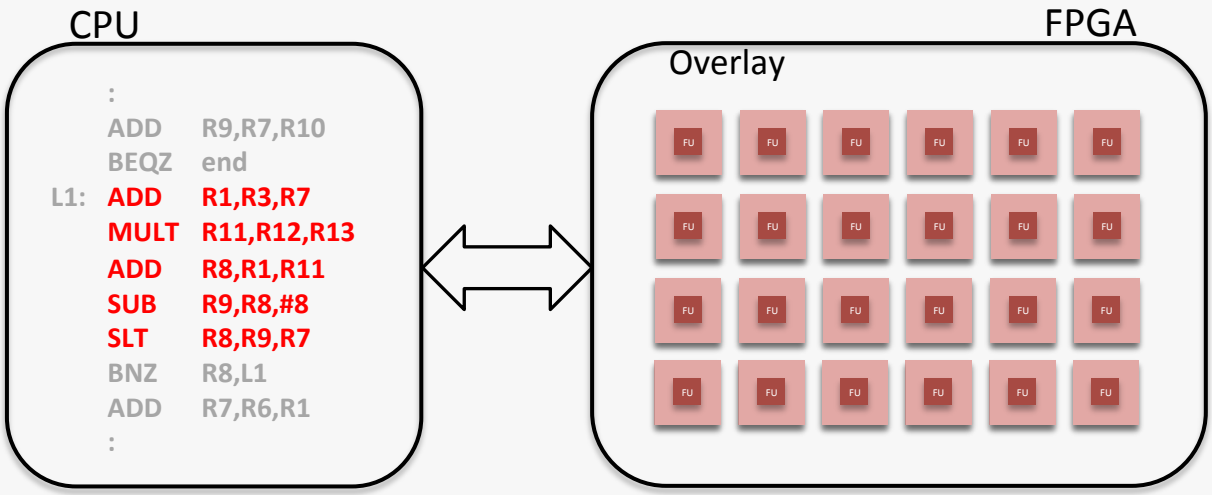
- Overlays raise the level of abstraction of using FPGAs to one that is more familiar to software designers
  - C programming for a soft processor
  - CUDA/OpenCL for GPU overlays
  - Data flow graphs for mesh-of-FUs
- This opens up opportunities for “standard” software tools to target FPGAs

JIT Compilation to Hardware



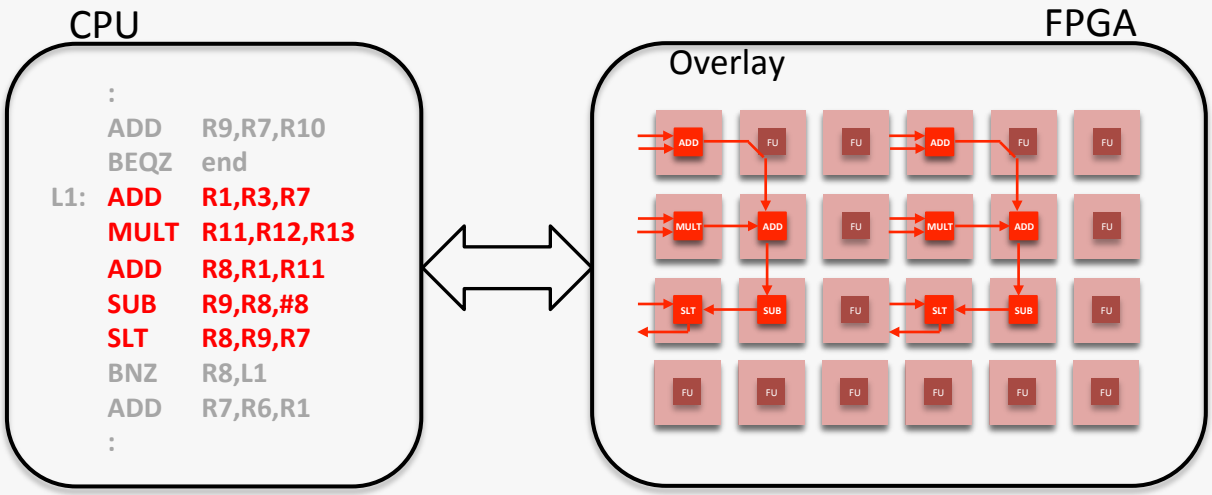
- Profile code

JIT Compilation to Hardware



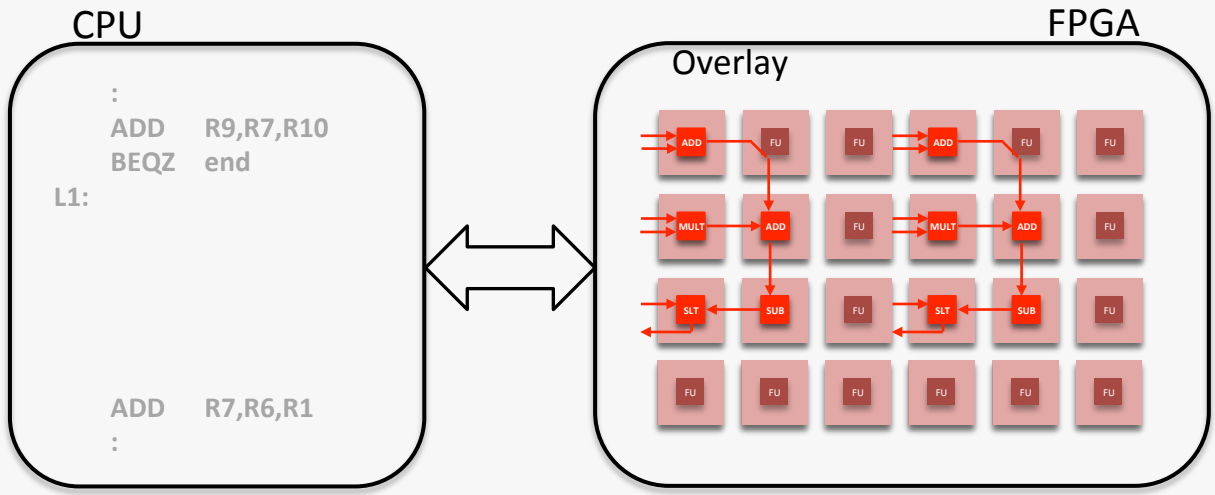
- Identify hot segments of code

JIT Compilation to Hardware



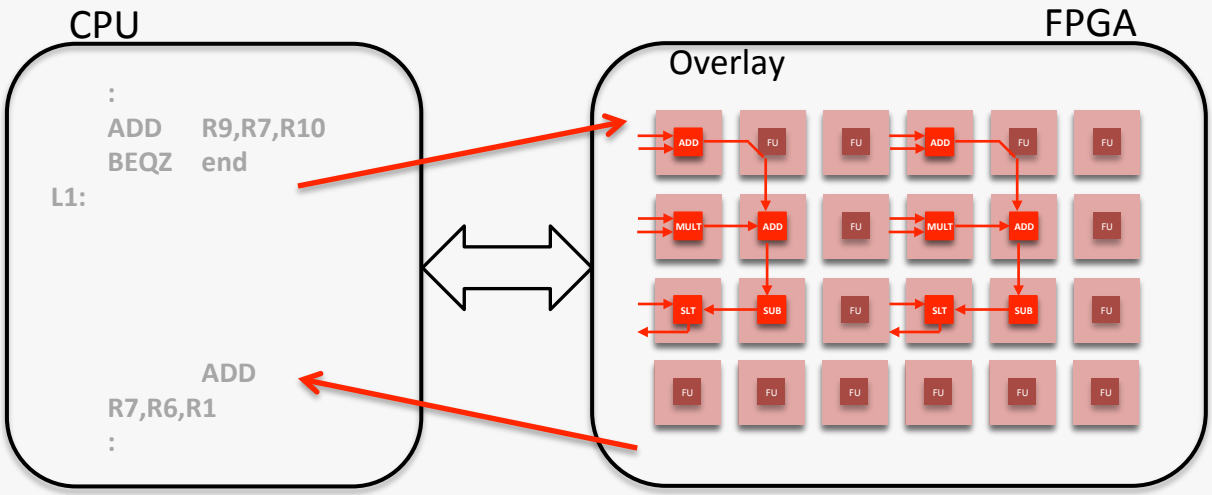
- Extract DFG and configure the overlay

JIT Compilation to Hardware



- Re-write the code

JIT Compilation to Hardware



- Transfer execution to the overlay

User-Transparent Dynamic Program Acceleration

A Prototype JIT Compiler

- Target: Intel QuickAssist Platform

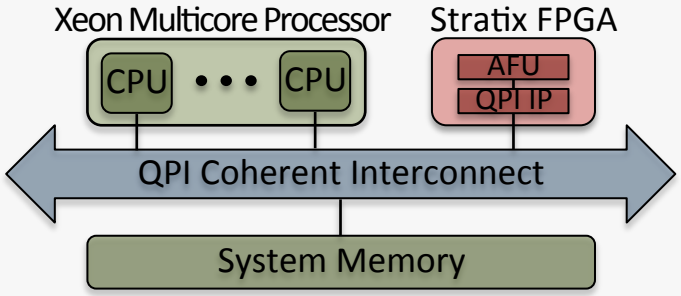
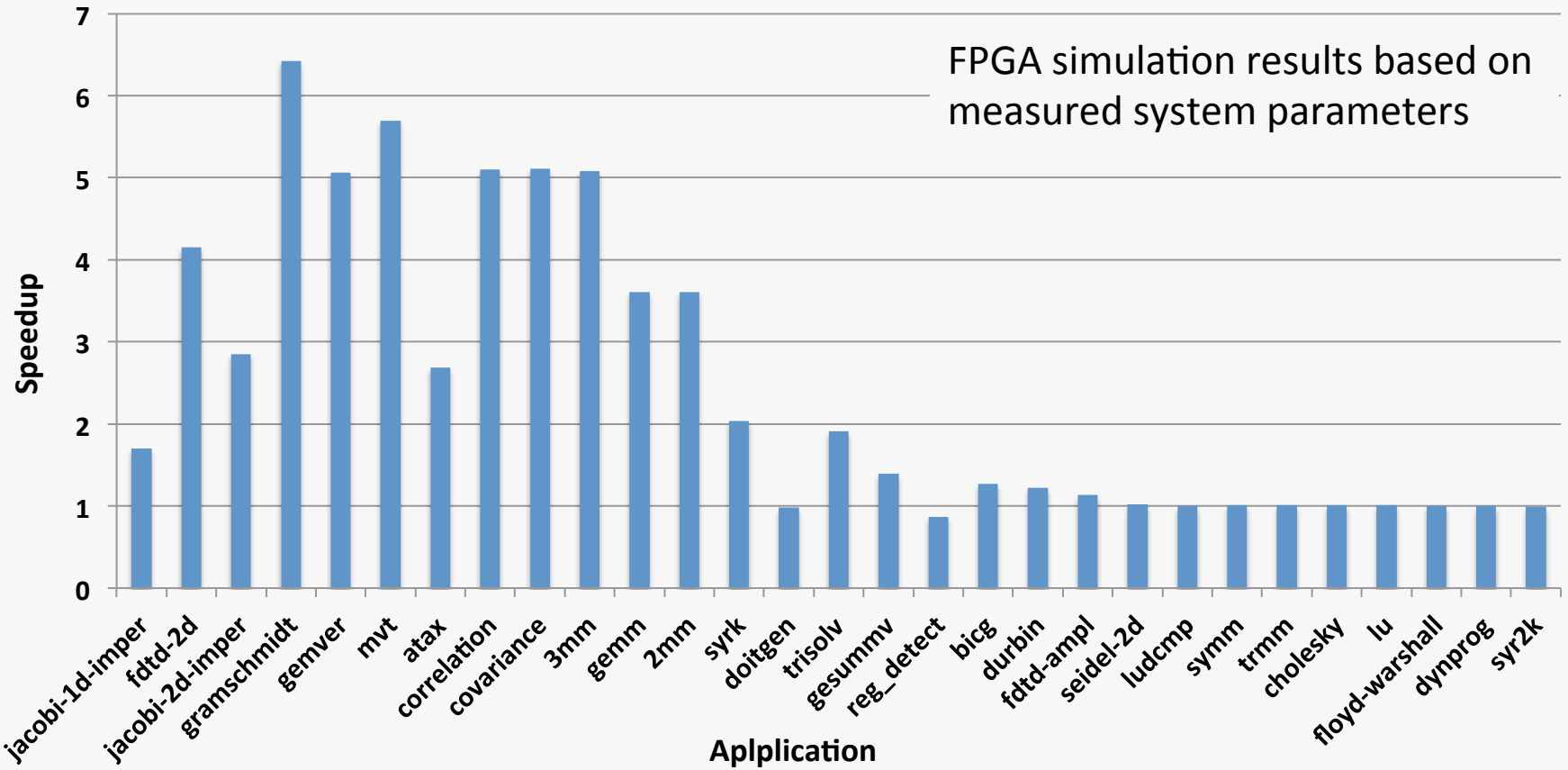


Figure after Intel literature

- The compiler prototype:
  - Built around LLVM, targets innermost loops of scientific code
  - Mitigates much of the run-time overhead to compile time

- Overlay currently being integrated into the target platform

Acceleration Potential

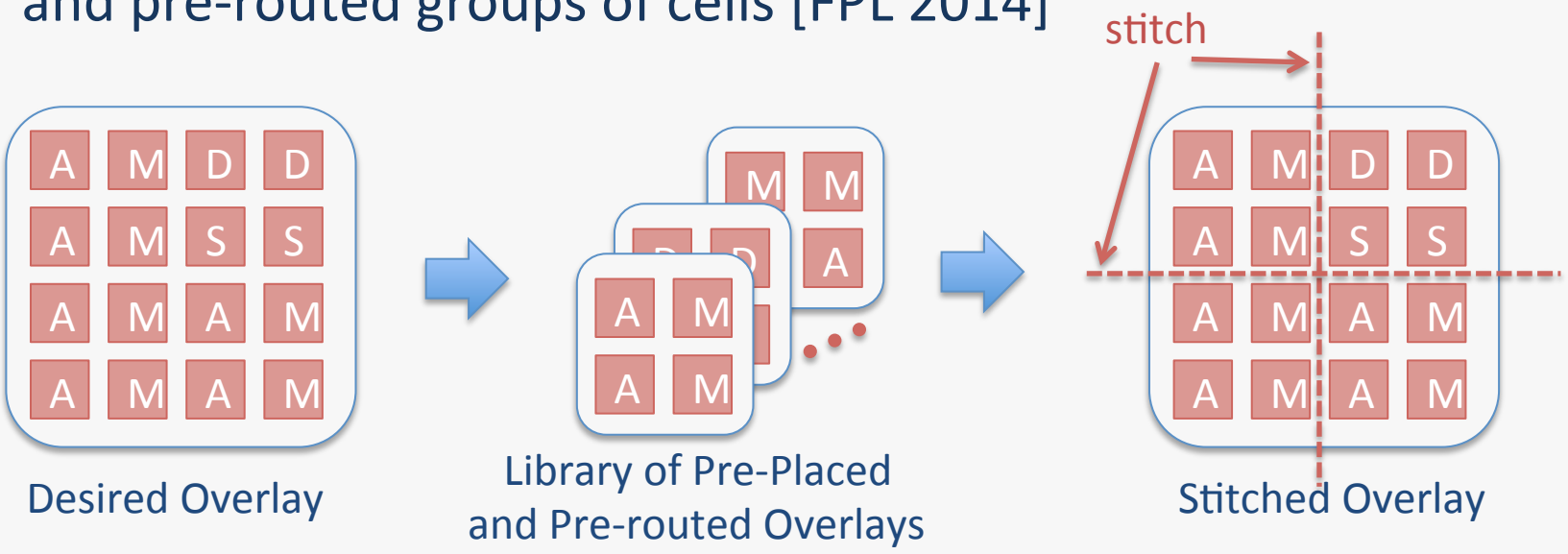


Customizability

- One of the key advantages of FPGAs is that they can be customized for applications
- Overlays can also be “user” customizable
  - With minimal usage of FPGA design tools
- In the context of our mesh-of-FUs, we can vary the choice of the FU at each location of the mesh, i.e., the functional layout, to the overlay more efficient for an application

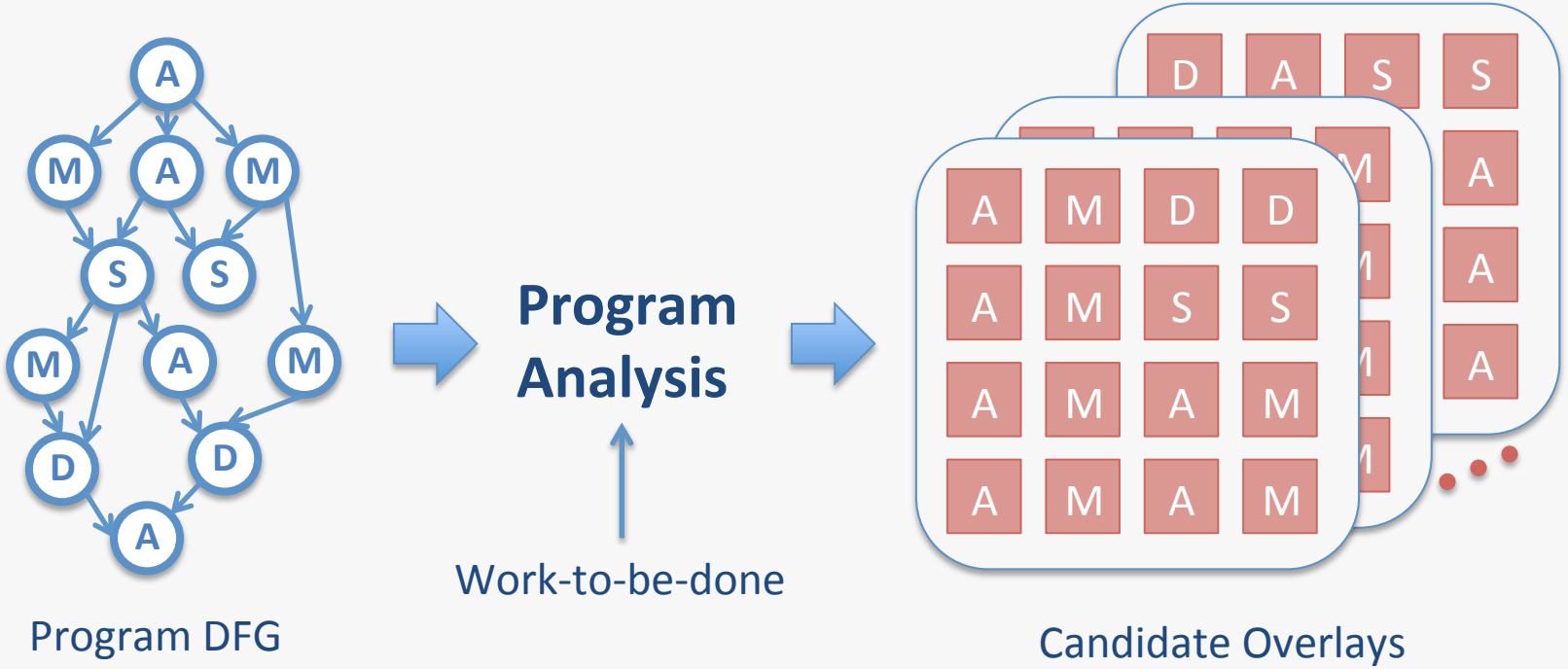
A Library-Based Approach

- Bottom-Up flow allows (restricted) relocation of pre-placed and pre-routed groups of cells [FPL 2014]

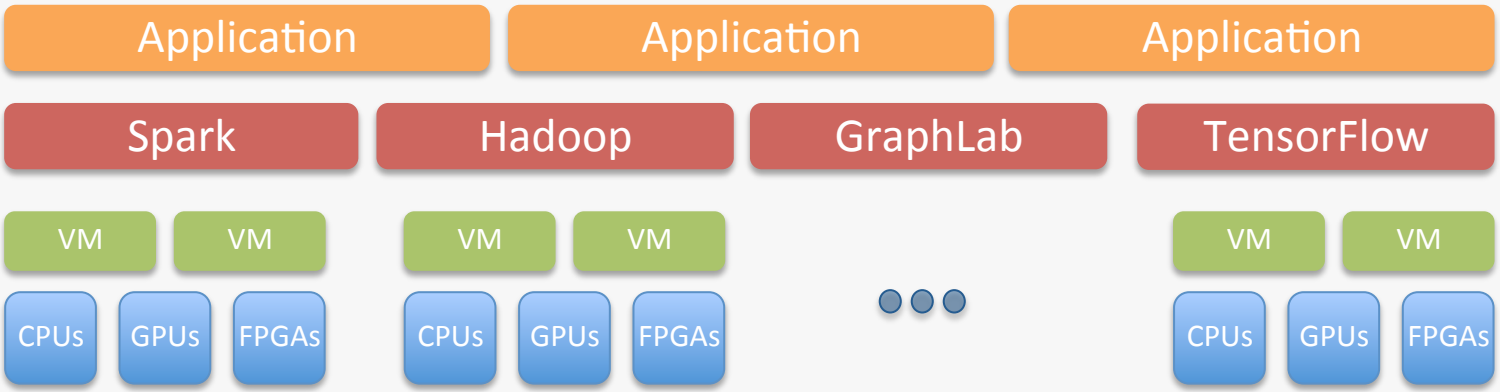


- Example 12x15 overlay: 35 minutes vs. 15 hours

Program Analysis for Customization



System Integration



- Must be able to virtualize the FPGA
  - Take snapshots
  - Migrate
  - Share and manage as a resource

Overlays Facilitate Virtualization

- FPGA virtualization only now being explored
  - Requires specialized hardware
  - A very large “state”

- Overlays naturally have a much smaller state, facilitating snapshots and context switching
  - We would like to explore this support in our mesh-of-FUs overlay

Challenges to Overlays

- Resource overhead
  - That is, the FPGA resources used by the overlay compared to a dedicated circuit (HDL) that implements the same application
  - ~4X for our FP overlay and can be higher
  - Difficult to quantify design effort

  - FPGAs are are increasing in size
  - Hard floating point units
  - Hardening the overlay once design is over?

Challenges to Overlays – Cont'd

- Overlay architectures need more exploration: which architecture for a given application domain
  - How to ensure scalability?
  - Taking into account the underlying FPGA device constraints
  - How to implement well (e.g., data-driven execution, FIFOs, etc.)?
  - Fixed function vs. multi-function FUs?
  - How to reducing resource overhead?
  - Time multiplexed?
  - Multiple devices?

Challenges to Overlays – Cont'd

- Evolving the FPGA design tools
  - Modular architectures do not lead to modular circuits
    - The tools do not understand the modularity
    - At present we must "fight with them" [FPL 2014]
  - The tools must evolve to allow developers to express and to recognize the modularity of the architecture
    - Scalable circuits from scalable architectures

Concluding Remarks

- A case for overlays
  - Performance, software-friendliness, customizability and system integration
- They can serve as “middle ground” between hardware design and software programming
  - Either for production or for debugging and prototyping
- Challenges to architecture, programming models, implementation and resource overhead

Questions?