User Transparent Acceleration of Loops on Field Programmable Gate Arrays

Leslie Barron and Tarek Abdelrahman

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

November 1st, 2016

User Transparent Acceleration of Loops on Field Programmable Gate Arrays

A Work In Progress

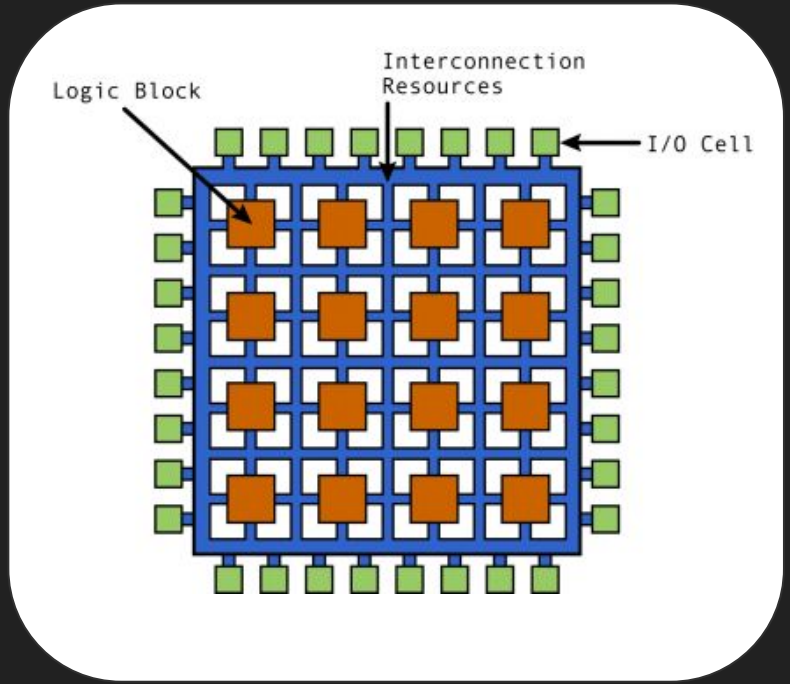
Leslie Barron and Tarek Abdelrahman

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

November 1st, 2016

Motivation

- Field Programmable Gate Arrays (FPGAs) are configurable circuits that can be reprogrammed to suit various applications
  - Potential to greatly increase performance for some applications
  - Increased performance per watt
  - FPGA-accelerated systems from IBM, Intel and Xilinx



Motivation - Cont’d

- FPGAs have a programmability wall for software developers
  - Requires experience in hardware design
  - Hardware CAD tools lead to longer development cycles
- Creates large barrier to entry relative to software
- One goal of our research group is to tackle the programmability wall
- My work specifically tackles it through JIT compilation to FPGAs

Research Goal

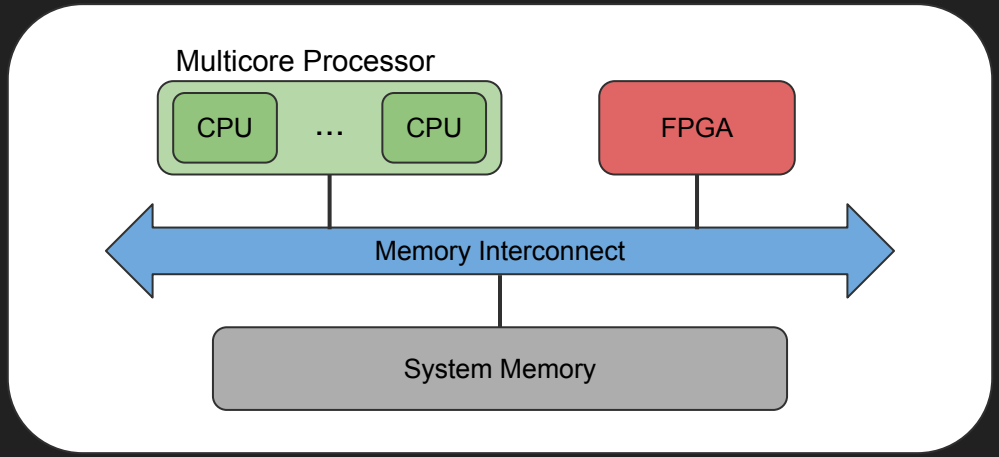
- Design a JIT compiler that
  - Profiles code
  - Identifies hot loops
  - Extracts dataflow graphs from hot loops
  - Accelerates loops on FPGA

- Gives us user transparent FPGA acceleration

- Potentially feasible because of two recent advancements

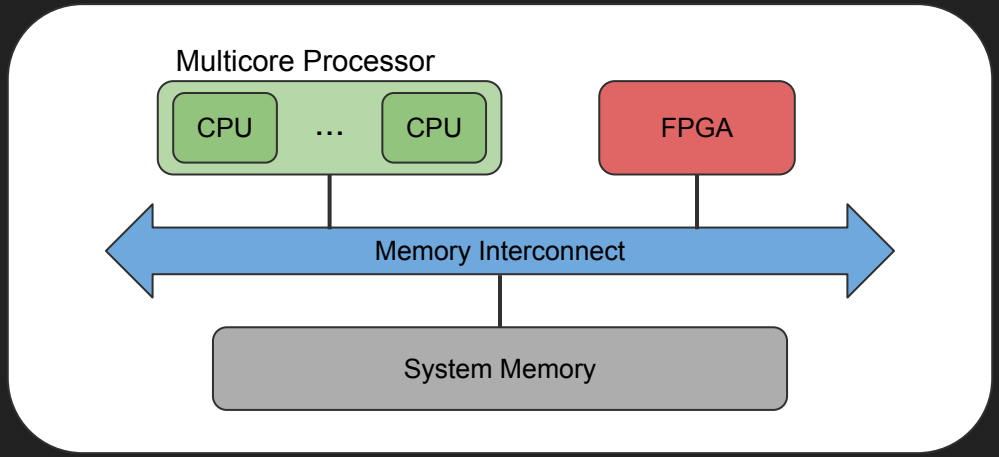
Heterogeneous Systems

- New heterogeneous CPU-FPGA systems being developed
  - Tightly integrate FPGAs and processors with shared memory
  - Examples include Intel's QuickAssist, IBM's Capi, Xilinx's Zynq



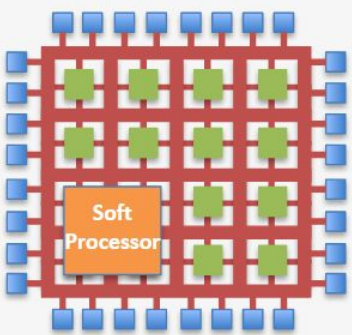
Heterogeneous Systems

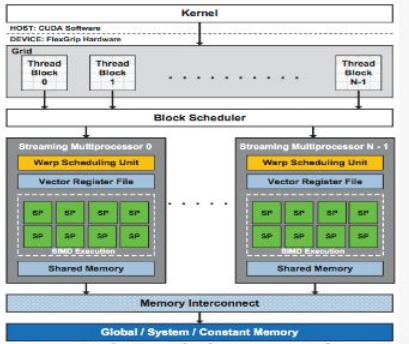
- New heterogeneous CPU-FPGA systems being developed
  - Tightly integrate FPGAs and processors with shared memory
  - Examples include Intel's QuickAssist, IBM's Capi, Xilinx's Zynq



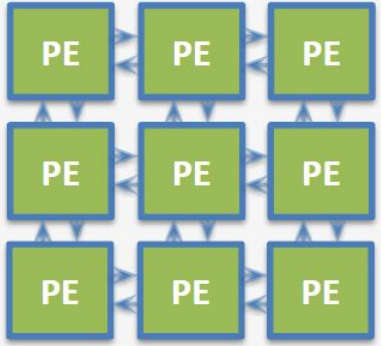
Overlays: Run-time Configurable Circuits

- Pre-compiled FPGA circuits that are in themselves configurable, i.e., run-time configurable



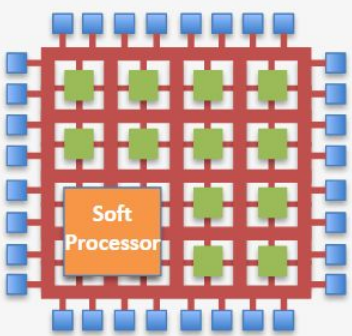


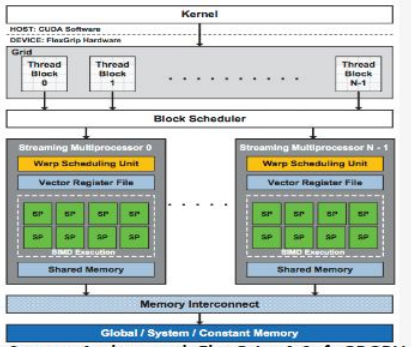
Source: Andryc et al: FlexGrip: A Soft GPGPU for FPGAs, FPT 13



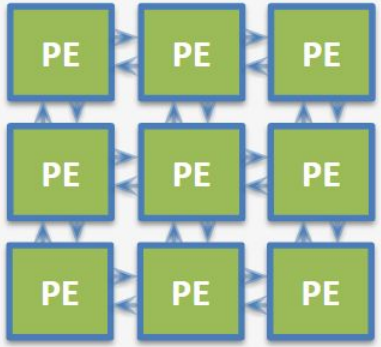
Overlays: Run-time Configurable Circuits

- Pre-compiled FPGA circuits that are in themselves configurable, i.e., run-time configurable





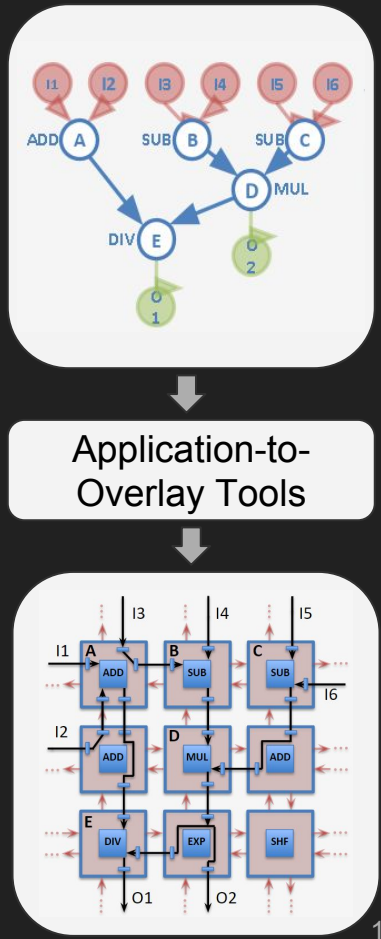
Source: Andryc et al: FlexGrip: A Soft GPGPU for FPGAs, FPT 13



- Raises level of abstraction and alleviates need for CAD tools

Target Overlay

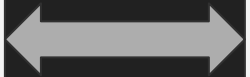
- Application (floating point) expressed as a Dataflow Graph (DFG)
- Application-to-overlay tools map DFG to overlay and produce a configuration bitstream
- Overlay executes DFG instances in a pipelined fashion
  - Throughput is improved



JIT Compilation to Overlay

CPU

```
        .
        .
        ADD     R9,R7,R10
        BEQZ    end
L1:     ADD     R1,R3,R7
        MULT    R11,R12,R13
        ADD     R8,R1,R11
        SUB     R9,R8,#8
        SLT     R8,R9,R7
        BNZ     R8, L1
        ADD     R7,R6,R1
        .
        .
```



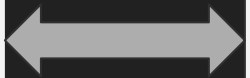
Overlay

1. Profile code

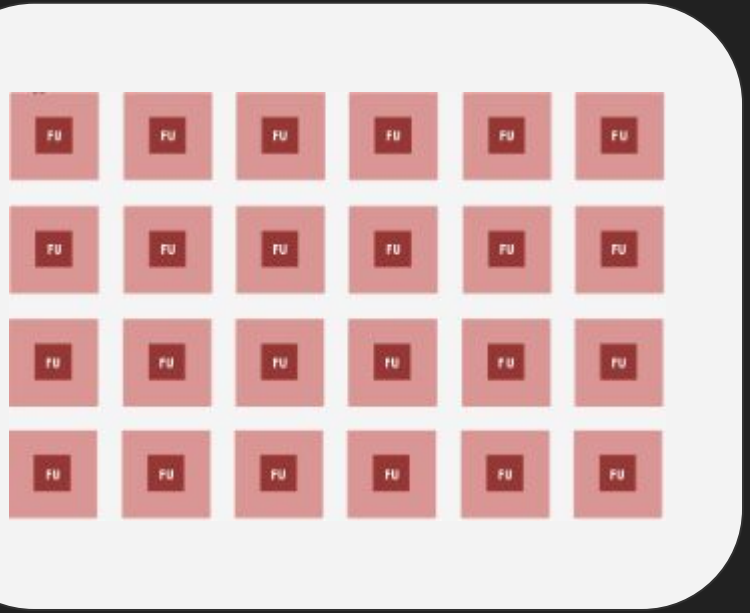
JIT Compilation to Overlay

CPU

```
        .
        .
        ADD     R9,R7,R10
        BEQZ    end
L1:     ADD     R1,R3,R7
        MULT    R11,R12,R13
        ADD     R8,R1,R11
        SUB     R9,R8,#8
        SLT     R8,R9,R7
        BNZ     R8, L1
        ADD     R7,R6,R1
        .
        .
```



Overlay



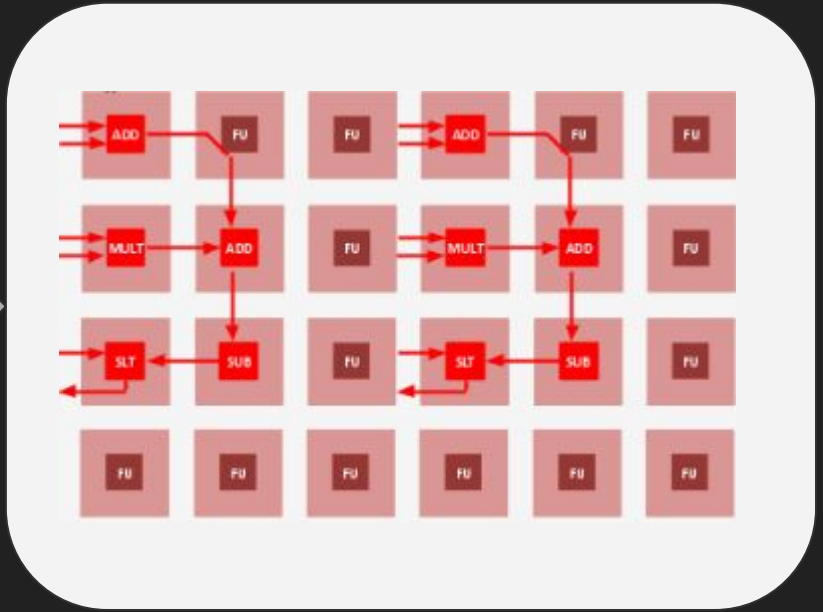
2. Identify hot loops

JIT Compilation to Overlay

CPU

```
        .
        .
        ADD     R9,R7,R10
        BEQZ    end
L1:     ADD     R1,R3,R7
        MULT    R11,R12,R13
        ADD     R8,R1,R11
        SUB     R9,R8,#8
        SLT     R8,R9,R7
        BNZ     R8, L1
        ADD     R7,R6,R1
        .
        .
```

Overlay



3. Extract DFG, generate bitstream and configure overlay

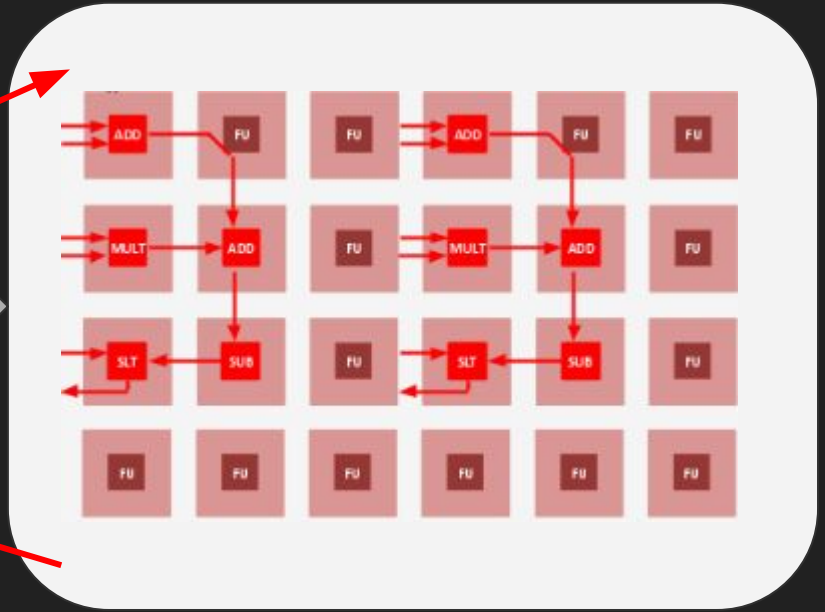
JIT Compilation to Overlay

CPU

```
        .
        .
        ADD     R9,R7,R10
        BEQZ    end
L1:

        ADD     R7,R6,R1
        .
        .
```

Overlay



4. Re-write code to transfer execution to Overlay

Target Applications and System

- This work specifically targets....
  - Scientific applications with heavy computation that is focused in loops
  - On an existing overlay that supports pipelined execution of DFGs
  - On the heterogeneous Intel QuickAssist FPGA platform

Challenges - JIT Compilation to FPGA

	Traditional JIT	Our JIT
Code Segment	Functions	Innermost Loops
Optimizing	1. Optimization Passes 2. Compile to native code	1. Extract DFG from loop 2. Convert DFG to bitstream
Transfer of Execution	1. Switch contexts 2. Branch to function pointer	1. Prepare shared data 2. Transfer execution to FPGA

Execution Model Restrictions

The pipelined execution of DFGs gives rise to a number of restrictions

- No back edges, only innermost loop bodies
- Writes must be unconditional
- No loop dependencies

Challenges - Loops

- DFGs cannot be extracted from all loops

Challenges - Loops

- DFGs cannot be extracted from all loops
  - No while loops

```
while(condition) {


}
```

Challenges - Loops

- DFGs cannot be extracted from all loops
  - No while loops
  - No function calls

```
while(condition) {

        doStuff();

}
```

Challenges - Loops

- DFGs cannot be extracted from all loops
  - No while loops
  - No function calls
  - No dependencies

```
for(int i = 1; i < 5; i++) {

        a[i] = a[i - 1];

}
```

Challenges - Loops

- DFGs cannot be extracted from all loops
  - No while loops
  - No function calls
  - No dependencies
- Not all accelerated loops improve performance over CPU

```
for(int i = 0; i < 5; i++) {

        a[i] = b[i];

}
```

Challenges - Overhead

- Many sources of overhead
  - Determining loop admissibility
  - Extracting DFG from loop
  - Generating overlay configuration bitstream
  - Configuring overlay with bitstream

- Overcoming them all at runtime is challenging

Overcoming Overhead

- Move as much to compile time as possible
  - Identify admissible loops to potentially be accelerated
  - Generate and embed bitstreams in binary
  - Instrument profiling code and transfer logic for admissible loops
- Program startup
  - Configure FPGA with overlay at program startup
  - Currently assume DFGs of all admissible loop(s) are preloaded at startup
- Runtime
  - Communicating with overlay and accelerate hottest loop(s)

Challenges - FPGA Performance

- FPGAs strengths lie in increased parallelism
  - CPU loop time $\propto$ InstructionsPerLoop * TripCount
  - FPGA loop time $\propto$ CriticalPathLatency + TripCount

Challenges - FPGA Performance

- FPGAs strengths lie in increased parallelism
  - CPU loop time $\propto$ InstructionsPerLoop * TripCount
  - FPGA loop time $\propto$ CriticalPathLatency + TripCount
- However, ...
  - CPU is clocked ~10x faster than FPGAs
  - CPU has its own forms of parallelism

Challenges - FPGA Performance

- FPGAs strengths lie in increased parallelism
  - CPU loop time $\propto$ InstructionsPerLoop * TripCount
  - FPGA loop time $\propto$ CriticalPathLatency + TripCount
- However, ...
  - CPU is clocked ~10x faster than FPGAs
  - CPU has its own forms of parallelism
- Solution - Duplicate DFG
  - Execute multiple iterations at once, all pipelined
  - Duplication factor only limited by memory bandwidth and FPGA space

Loop Instrumentation

- Multiple ways to instrument loops
  - Send only hottest traces of loop body
    - Most efficient use of space on FPGA
    - Very high profiling overhead
  - Or send entire loop body
    - Less complicated to implement
    - Lower profiling overhead
  - No difference between approaches for straight line code
- We opt to send entire loop body

Loop Instrumentation

- Transfer of execution logic can go inside or outside loop
  - If inside loop can be transferred between iterations
  - Also has larger overhead
- Keeping it outside the loop has advantages
  - Lowest overhead
  - Simplest to implement
  - Cannot accelerate loop on first invocation
- We opt for the transfer logic to go outside

Loop Instrumentation

```
for(int i = 0; i < N; i++) {
    // Original loop
}
```

Loop Instrumentation

- Two profiling counters are created per loop

```
uint64_t LoopIters = 0;
uint64_t LoopRun = 0;

for(int i = 0; i < N; i++) {
    // Original loop
}

LoopIters += N;
LoopRun++;
```

Loop Instrumentation

- Two profiling counters are created per loop
- Overlay bitstream is generated and embedded into a byte array

```
char* FPGABitstream = {.....}
uint64_t LoopIters = 0;
uint64_t LoopRun = 0;

for(int i = 0; i < N; i++) {
    // Original loop
}

LoopIters += N;
LoopRun++;
```

Loop Instrumentation

- Two profiling counters are created per loop
- Overlay bitstream is generated and embedded into a byte array
- Heuristic and Transfer logic is added

```
char* FPGABitstream = {.....}
uint64_t LoopIters = 0;
uint64_t LoopRun = 0;

if(shouldRunOnFPGA(LoopIters, LoopRun, ...))
    /* Prepare shared data */
    runOnFPGA(FPGABitstream, ...);
else
    for(int i = 0; i < N; i++) {
        // Original loop
    }

LoopIters += N;
LoopRun++;
```

Current Status

- Built a prototype system using llvm
  - Implemented using two code passes
  - Pass that determines loop admissibility and extracts DFGs
  - Pass that instruments admissible loops

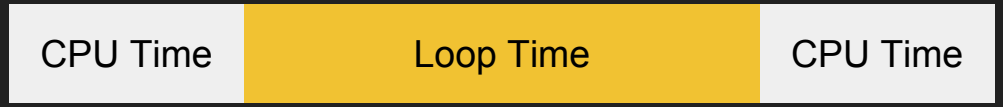
- Overlay not fully implemented on target system
  - Work in progress
  - Created performance model based on microbenchmarks on the system
  - Built functional DFG simulator

Performance Estimation

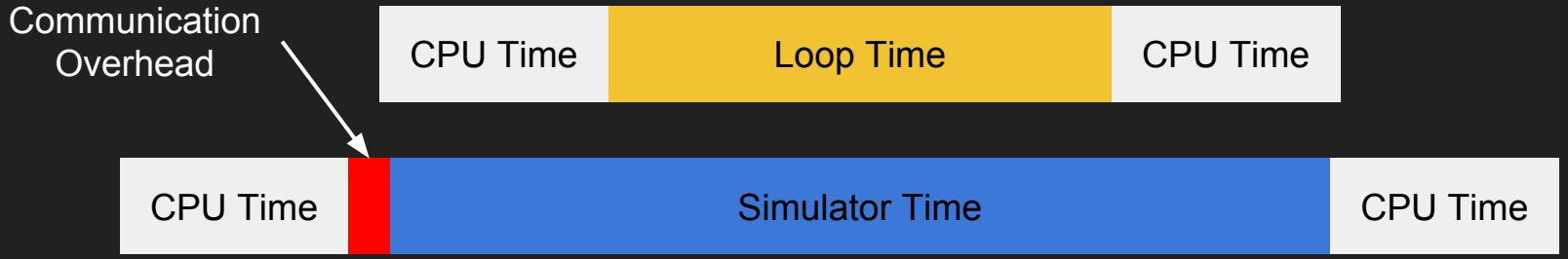
- Interested in determining speedup of raw CPU execution versus the accelerated execution

- Measuring runtime of raw CPU execution is easy
- How do we determine runtime of accelerated execution?
  - Instrument code and execute using DFG simulator in place of FPGA
  - Includes data communication overheads
  - Subtract off time spent in simulator
    - add modeled FPGA time in its place

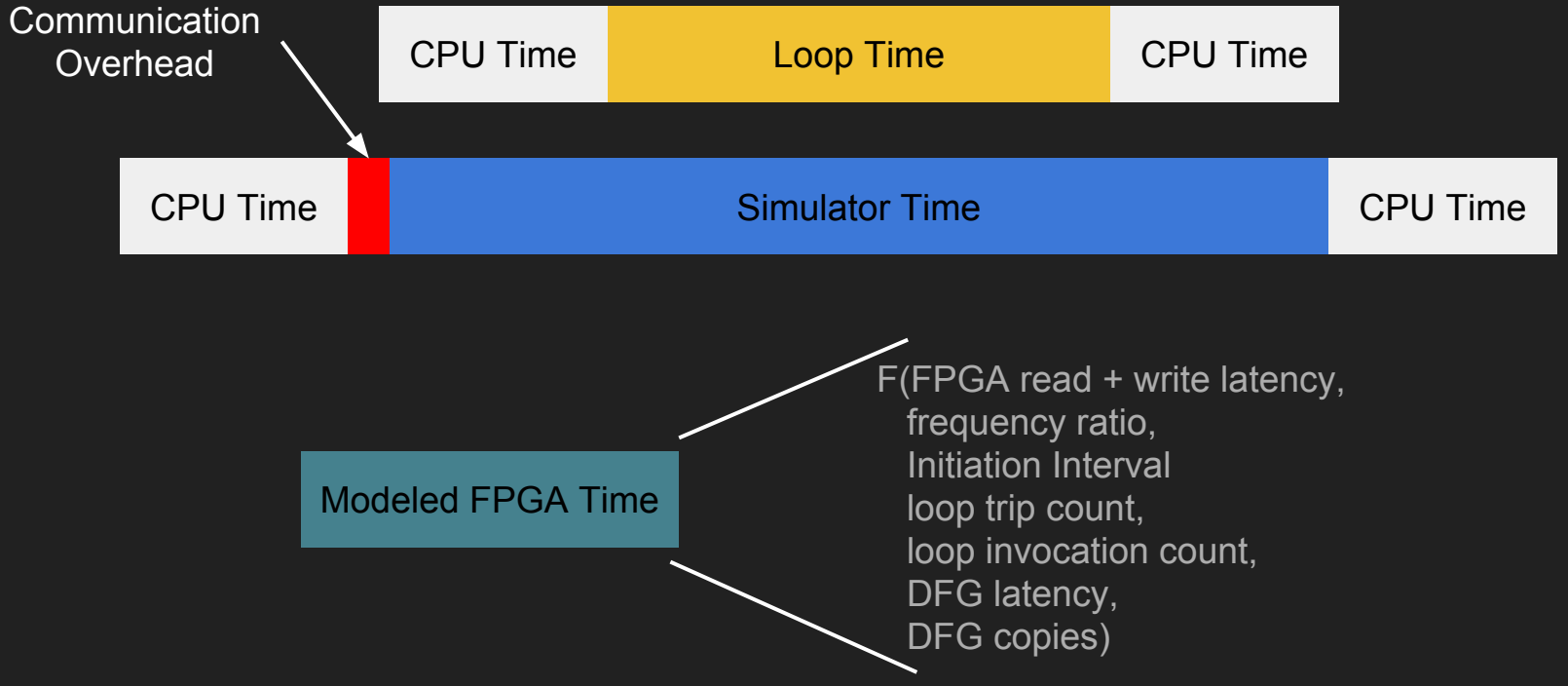
Performance Estimation



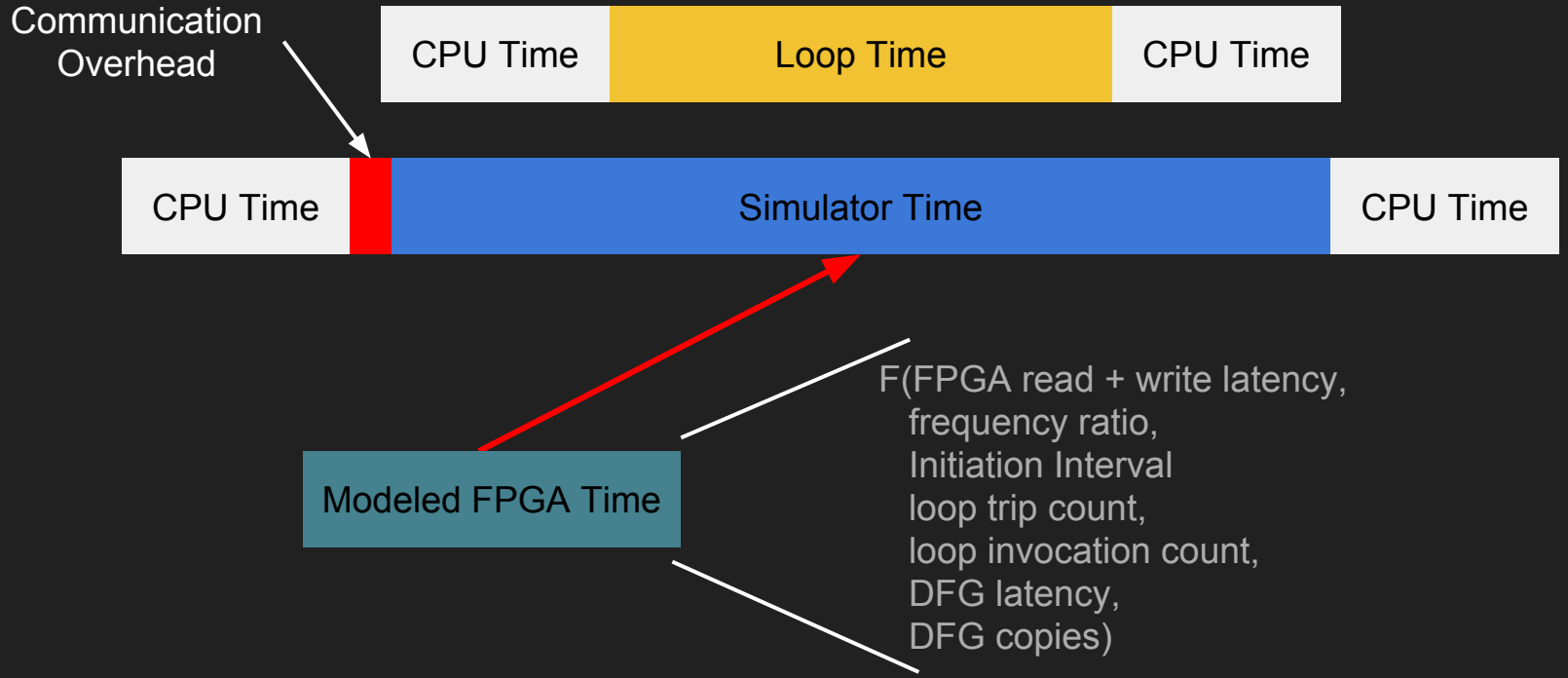
Performance Estimation



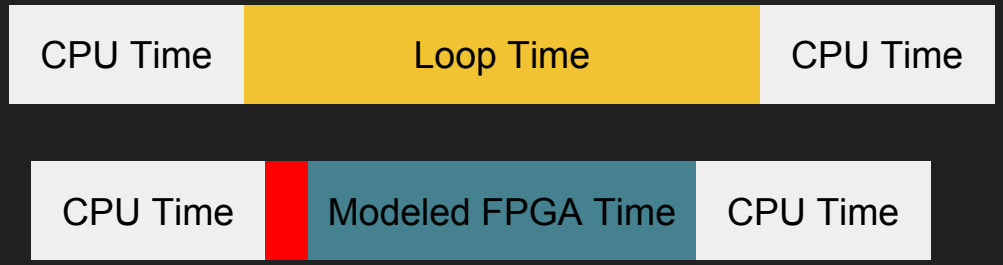
Performance Estimation



Performance Estimation



Performance Estimation



Evaluation - Experimental Parameters

- Assumed a frequency ratio of 10 (2-4 GHz vs 200-400 MHz)
- Assumed read/write latency of 36/48 cycles
- Assumed 16 copies of each DFG
- Assumed Initiation Interval of 1
  - Thus the results are an upper bound on performance
- Results collected on a machine with an Intel i5-3570 processor with 8gb of ram

Evaluation - Benchmarks

- Evaluated on 30 benchmarks in polybench suite
  - Contains floating point code generally found in scientific applications that process large arrays
  - Examples include matrix multiply, lu decomposition, stencils, etc..

- The 30 benchmarks contained 119 candidate loops
  - Which are innermost loops which perform computation

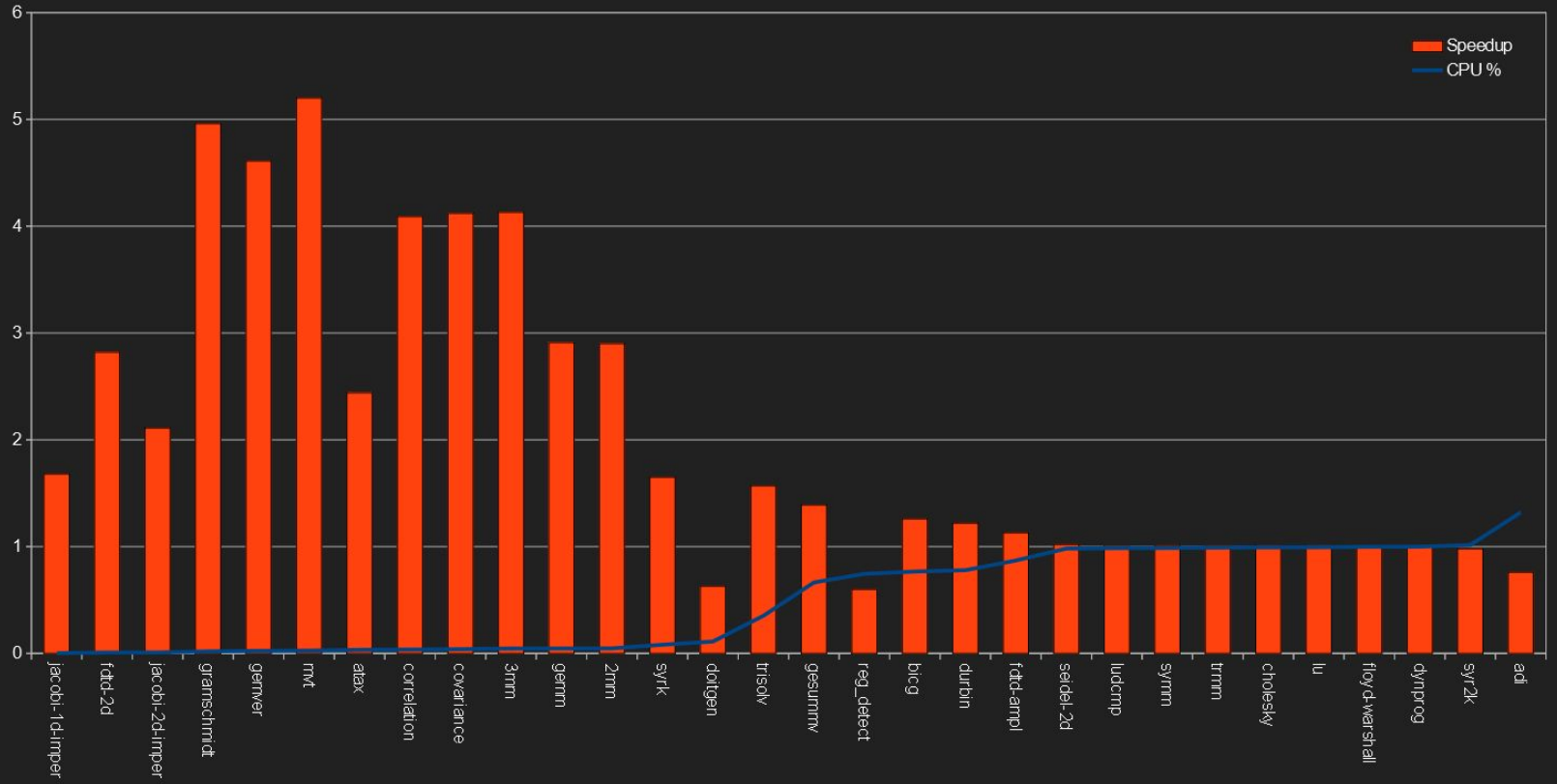
Evaluation - Metrics

- Used two metrics to describe our results
  - Relative speedup - Speedup between the accelerated runtime and raw CPU runtime
  - CPU percentage - CPU runtime during the simulated run (without the simulation time) divided by full CPU runtime
- CPU percentage determines the upper bound of speedup we can achieve
  - This is just the sequential fraction in Amdahl’s Law

Evaluation - Preliminary Results

- 92 of the 119 candidate loops were found to be admissible and therefore able to be accelerated
- Achieved average speedup of 2.04 over all 30 benchmarks
- Of the 14 benchmarks where all candidate loops were admissible we achieved an average speedup of 3.16
- Only 3 benchmarks had slowdowns greater than 2%

Evaluation - Preliminary Results



Concluding Remarks

- Explored the design of a JIT compiler to accelerate programs on an FPGA
  - Accelerates programs user transparently
  - Requires no prior knowledge or source code modification
- Achieved an average (calculated) speedup of 2.04 over 30 benchmarks
- Technique was applicable to 77% of candidate loops in the benchmarks

Limitations

- Measured using a performance model
  - Results represent an upper bound on performance
- Assumed overlay was preprogrammed at startup
- Not invoking overlay tools or generating bitstream

- Nonetheless we show a high potential
  - Even a fraction of the benefit would be great

Future Work

- Run on hardware
- Measure performance per watt
- Larger programs where not all DFGs can fit simultaneously

Questions