High-Speed Soft-Processor Architecture
For FPGA Overlays

Charles Eric LaForest

SGS Final Oral Examination
December 5th, 2014
Motivation

- Designing on FPGAs remains difficult
  - Larger systems
  - Longer CAD processing times
  - Increases time-to-market and engineering costs
FPGA Design Processes

- Hardware Description Languages (Verilog, VHDL)
  - Precise implementation
  - Low-level and tedious
  - Long CAD processing time
FPGA Design Processes

- High-Level Synthesis (LegUp, Bluespec)
  - Easier, faster design and exploration
  - Mostly same performance as HDL
  - “Black-Box” implementations
  - Long CAD processing time
FPGA Design Processes

- Overlays (soft-processors)
  - Easiest and fastest: design as software
  - Co-design hardware only if necessary
  - Fast overall design cycle
  - Lower performance
  - Higher area
FPGA Design Processes

- Soft-processor vs. underlying FPGA (Stratix IV)
  - Logic Fabric: 800 MHz
  - Block RAM: 550 MHz
  - DSP Block: 480 MHz
  - Nios II/f: 240 MHz
FPGA Design Processes

- How do we improve overlay performance?
Overlay Design Goals

• Abundant Parallelism
  – SIMD and MIMD
Overlay Design Goals

- Abundant Parallelism
  - SIMD and MIMD
- High Clock Frequency (Fmax)
Overlay Design Goals

- Abundant Parallelism
  - SIMD and MIMD
- High Clock Frequency (Fmax)
- Low Architectural Overhead
  - Low CPI, instruction count
Overlay Design Goals

- Abundant Parallelism
  - SIMD and MIMD
- High Clock Frequency (Fmax)
- Low Architectural Overhead
  - Low CPI, instruction count
- Few Stalls
  - Data and Control dependencies, Memory latency
Overlay Design Goals

- Abundant Parallelism
  - SIMD and MIMD
- High Clock Frequency (Fmax)
- Low Architectural Overhead
  - Low CPI, instruction count
- Few Stalls
  - Data and Control dependencies, Memory latency
- Simple and Minimal
Overlay Design Goals

- Abundant Parallelism
  - SIMD and MIMD
- High Clock Frequency (Fmax)
- Low Architectural Overhead
  - Low CPI, instruction count
- Few Stalls
  - Data and Control dependencies, Memory latency
- Simple and Minimal
- Congruent to underlying FPGA
  - Word widths, pipeline depths, primitives
Multi-Threaded Overlay Architecture

• Must pipeline to absorb FPGA delays
  – Problem: dependencies between pipeline stages
Multi-Threaded Overlay Architecture

• Must pipeline to absorb FPGA delays
  – Problem: dependencies between pipeline stages

• Proposed solution: fully-pipelined multi-threading
  – Full pipelining to maximize Fmax
  – “Single-cycle” thread instructions over entire pipeline
  – Multiple threads for SIMD and MIMD parallelism
Multi-Threaded Overlay Architecture

- Must pipeline to absorb FPGA delays
  - Problem: dependencies between pipeline stages

- Proposed solution: fully-pipelined multi-threading
  - Full pipelining to maximize Fmax
  - “Single-cycle” thread instructions over entire pipeline
  - Multiple threads for SIMD and MIMD parallelism

- Only allow fixed round-robin scheduling
  - Unlike HEP, Tera, UTMT II, CUSTARD, NetThreads
  - No pipeline dependencies (almost…)
  - Determinism enables thread composition
Self-Loop Characterization (BRAM)

- Accounts for interconnect and clock-to-out delay

398 MHz
Self-Loop Characterization (BRAM)

398 MHz  656 MHz

- Accounts for interconnect and clock-to-out delay
Self-Loop Characterization (BRAM)

- Accounts for interconnect and clock-to-out delay

398 MHz  656 MHz

531 MHz
Self-Loop Characterization (BRAM)

398 MHz 656 MHz

531 MHz 710 MHz

- Accounts for interconnect and clock-to-out delay
Self-Loop Characterization (BRAM)

- Accounts for interconnect and clock-to-out delay
- Minimum clock pulse width of 500 to 550 MHz
- Absolute upper frequency limit on Stratix IV

398 MHz  656 MHz

531 MHz  710 MHz
Overlay High-Level Architecture

BRAMs

A

ALU

B

I

R
Overlay High-Level Architecture

BRAMs

A

B

I

ALU

R
Overlay High-Level Architecture

BRAMs → A → ALU → R
B → I
Memory High-Level Architecture
Memory High-Level Architecture
Dual-Pipeline Multiplier
Dual-Pipeline Multiplier

480 MHz

clk/2

480 MHz

A

B

clk

ALU0 ALU1 ALU2

P
Dual-Pipeline Multiplier

600 MHz
Fully-Pipelined ALU
Fully-Pipelined ALU
Fully-Pipelined ALU

600 MHz
Fully-Pipelined ALU
Octavo Soft-Processor

- Reaches 550 MHz on Stratix IV FPGA
- 8 threads (fixed round-robin)
- 1024 36-bit integer words for each I/A/B memory
Octavo Soft-Processor

- Reaches 550 MHz on Stratix IV FPGA
- 8 threads (fixed round-robin)
- 1024 36-bit integer words for each I/A/B memory
Empty Pipeline Stages

- Derived from BRAM self-loop characterization
- Used for special functions later...
A and B Data Memories

- Memory-mapped I/O ports for Accelerators
• Computes next PC for each thread (8 PCs)
• Output (R) written to all memories
Data Path

- 8 stages (2 read, 4 compute, 2 write)
Control Path

- 8 stages to match Data Path
- Offset due to empty stages (1,2,3)
- 1-cycle RAW hazard from ALU to Instr. Mem.
Table 3.1: Octavo’s Instruction Word Format.

<table>
<thead>
<tr>
<th>Field</th>
<th>4 bits</th>
<th>a bits</th>
<th>a bits</th>
<th>a bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode (OP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination (D)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source (A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source (B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Octavo’s Instruction Set and Opcode Encoding.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>0000</td>
<td>D ← A ⊕ B</td>
</tr>
<tr>
<td>AND</td>
<td>0001</td>
<td>D ← A ∧ B</td>
</tr>
<tr>
<td>OR</td>
<td>0010</td>
<td>D ← A ∨ B</td>
</tr>
<tr>
<td>SUB</td>
<td>0011</td>
<td>D ← A − B</td>
</tr>
<tr>
<td>ADD</td>
<td>0100</td>
<td>D ← A + B</td>
</tr>
<tr>
<td>—</td>
<td>0101</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0110</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0111</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>Multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MHS</td>
<td>1000</td>
<td>D ← A · B (High Word Signed)</td>
</tr>
<tr>
<td>MLS</td>
<td>1001</td>
<td>D ← A · B (Low Word Signed)</td>
</tr>
<tr>
<td>MHU</td>
<td>1010</td>
<td>D ← A · B (High Word Unsigned)</td>
</tr>
<tr>
<td>Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>1011</td>
<td>PC ← D</td>
</tr>
<tr>
<td>JZE</td>
<td>1100</td>
<td>if (A = 0) PC ← D</td>
</tr>
<tr>
<td>JNZ</td>
<td>1101</td>
<td>if (A ≠ 0) PC ← D</td>
</tr>
<tr>
<td>JPO</td>
<td>1110</td>
<td>if (A ≥ 0) PC ← D</td>
</tr>
<tr>
<td>JNE</td>
<td>1111</td>
<td>if (A &lt; 0) PC ← D</td>
</tr>
</tbody>
</table>
### Table 3.1: Octavo’s Instruction Word Format.

<table>
<thead>
<tr>
<th>Size:</th>
<th>4 bits</th>
<th>$a$ bits</th>
<th>$a$ bits</th>
<th>$a$ bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field:</td>
<td>Opcode (OP)</td>
<td>Destination (D)</td>
<td>Source (A)</td>
<td>Source (B)</td>
</tr>
</tbody>
</table>

### Table 3.2: Octavo’s Instruction Set and Opcode Encoding.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>0000</td>
<td>D ← A ⊕ B</td>
</tr>
<tr>
<td>AND</td>
<td>0001</td>
<td>D ← A ∧ B</td>
</tr>
<tr>
<td>OR</td>
<td>0010</td>
<td>D ← A ∨ B</td>
</tr>
<tr>
<td>SUB</td>
<td>0011</td>
<td>D ← A − B</td>
</tr>
<tr>
<td>ADD</td>
<td>0100</td>
<td>D ← A + B</td>
</tr>
<tr>
<td>—</td>
<td>0101</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0110</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0111</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>Multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MHS</td>
<td>1000</td>
<td>D ← A · B (High Word Signed)</td>
</tr>
<tr>
<td>MLS</td>
<td>1001</td>
<td>D ← A · B (Low Word Signed)</td>
</tr>
<tr>
<td>MHU</td>
<td>1010</td>
<td>D ← A · B (High Word Unsigned)</td>
</tr>
<tr>
<td>Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>1011</td>
<td>PC ← D</td>
</tr>
<tr>
<td>JZE</td>
<td>1100</td>
<td>if (A = 0) PC ← D</td>
</tr>
<tr>
<td>JNZ</td>
<td>1101</td>
<td>if (A ≠ 0) PC ← D</td>
</tr>
<tr>
<td>JPO</td>
<td>1110</td>
<td>if (A ≥ 0) PC ← D</td>
</tr>
<tr>
<td>JNE</td>
<td>1111</td>
<td>if (A &lt; 0) PC ← D</td>
</tr>
</tbody>
</table>
Table 3.1: Octavo’s Instruction Word Format.

<table>
<thead>
<tr>
<th>Field</th>
<th>4 bits</th>
<th>(a) bits</th>
<th>(a) bits</th>
<th>(a) bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode (OP)</td>
<td>4 bits</td>
<td>(a) bits</td>
<td>(a) bits</td>
<td>(a) bits</td>
</tr>
</tbody>
</table>

Table 3.2: Octavo’s Instruction Set and Opcode Encoding.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>0000</td>
<td>D ← A ⊕ B</td>
</tr>
<tr>
<td>AND</td>
<td>0001</td>
<td>D ← A ∧ B</td>
</tr>
<tr>
<td>OR</td>
<td>0010</td>
<td>D ← A ∨ B</td>
</tr>
<tr>
<td>SUB</td>
<td>0011</td>
<td>D ← A − B</td>
</tr>
<tr>
<td>ADD</td>
<td>0100</td>
<td>D ← A + B</td>
</tr>
<tr>
<td>—</td>
<td>0101</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0110</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>—</td>
<td>0111</td>
<td>(Unused, for expansion)</td>
</tr>
<tr>
<td>Multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MHS</td>
<td>1000</td>
<td>D ← A · B (High Word Signed)</td>
</tr>
<tr>
<td>MLS</td>
<td>1001</td>
<td>D ← A · B (Low Word Signed)</td>
</tr>
<tr>
<td>MHU</td>
<td>1010</td>
<td>D ← A · B (High Word Unsigned)</td>
</tr>
<tr>
<td>Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>1011</td>
<td>PC ← D</td>
</tr>
<tr>
<td>JZE</td>
<td>1100</td>
<td>if (A = 0) PC ← D</td>
</tr>
<tr>
<td>JNZ</td>
<td>1101</td>
<td>if (A ≠ 0) PC ← D</td>
</tr>
<tr>
<td>JPO</td>
<td>1110</td>
<td>if (A ≥ 0) PC ← D</td>
</tr>
<tr>
<td>JNE</td>
<td>1111</td>
<td>if (A &lt; 0) PC ← D</td>
</tr>
</tbody>
</table>
High Fmax over Design Space

![Graph showing Fmax vs. Word Width for different pipeline lengths (8, 10, 12, 14, 16 stages). The graph indicates the maximum frequency (Fmax) at various word widths for different pipeline lengths.]
High Fmax over Design Space

![Graph showing Fmax vs Word Width with different pipeline lengths: 8, 10, 12, 14, and 16 stages. The graph indicates that 36 bits achieve the highest Fmax.](image)

- Pipeline Length:
  - 8 stages
  - 10 stages
  - 12 stages
  - 14 stages
  - 16 stages

- NiosII/f

- Fmax (MHz)
- Word Width (bits)

- 36 bits achieve the highest Fmax.
Tiling Overlay Architectures

- Tiling: duplicating in 2-D for parallelism
  - Datapaths: SIMD
  - Processors: MIMD
Tiling Overlay Architectures

• Tiling: duplicating in 2-D for parallelism
  – Datapaths: SIMD
  – Processors: MIMD

• CAD optimizations now worsen performance!
Tiling Overlay Architectures

- Tiling: duplicating in 2-D for parallelism
  - Datapaths: SIMD
  - Processors: MIMD

- CAD optimizations now worsen performance!

- Simple way to steer CAD tool
  - ...without source annotations or per-node CAD
  - ...without increasing CAD processing time
Tiling Overlay Architectures

- Tiling: duplicating in 2-D for parallelism
  - Datapaths: SIMD
  - Processors: MIMD

- CAD optimizations now worsen performance!

- Simple way to steer CAD tool
  - ...without source annotations or per-node CAD
    - ...without increasing CAD processing time

- Meshes of Scalar and SIMD Octavo Cores
Tiling Datapaths for SIMD
Multi-Local Logic

- Same simultaneous inputs and/or states
- CAD tool “de-duplicates” to save area
Harmful Optimization

- Same simultaneous inputs and/or states
- CAD tool “de-duplicates” to save area
- But creates artificial critical paths!
Logical Partitioning

- Partition each Lane as a separate netlist
- Prevents optimizations across partitions
- Easily avoids harmful optimizations...
- ...without preventing useful ones!
Impact on Speed

The graph illustrates the impact of various lane configurations on speed, represented by the average frequency (MHz) as a function of the SIMD lane count. The lines and markers show the performance of different lane configurations:

- Red dashed line and markers: Flat (nrdr)
- Blue line and markers: Per-Lane
- Red solid line: Flat

The y-axis represents the average frequency (MHz), while the x-axis shows the SIMD lane count. The graph demonstrates how the speed changes as the lane count increases for each configuration.
Partitioning a 32-Way SIMD Octavo

Flat: 373 MHz
Partitioning a 32-Way SIMD Octavo

Flat: 373 MHz

Per-Lane: 489 MHz
Meshes of Octavo Cores

- Scalar or SIMD Cores
- New Multi-Localities
Meshes of Octavo Cores

- Scalar or SIMD Cores
- New Multi-Localities
  - 3-bit thread counter
  - 1 per Core
Meshes with 32 Datapaths Total

Fmax (MHz)

- 4x8 (Scalar) 350 MHz
- 4x4 (2-way) 370 MHz
- 2x4 (4-way) 390 MHz
- 1x1 (32-way) 410 MHz
Meshes with 32 Datapaths Total

Fmax (MHz)

<table>
<thead>
<tr>
<th>Flat</th>
<th>Per-Core</th>
<th>Per-Lane</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x8 (Scalar)</td>
<td>470</td>
<td></td>
</tr>
<tr>
<td>4x4 (2-way)</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>2x4 (4-way)</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>1x1 (32-way)</td>
<td>380</td>
<td></td>
</tr>
</tbody>
</table>
Meshes with 32 Datapaths Total

- Flat
- Per-Core
- Per-Lane

Fmax (MHz)

<table>
<thead>
<tr>
<th></th>
<th>Flat</th>
<th>Per-Core</th>
<th>Per-Lane</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x8 (Scalar)</td>
<td>350</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4x4 (2-way)</td>
<td>370</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2x4 (4-way)</td>
<td>390</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1x1 (32-way)</td>
<td>410</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mesh of 102 Octavo Cores

- Flat
- Per-Core

Fmax (MHz)

17x6 (Scalar)
Mesh of 102 Scalar Octavos (17x6)

Flat: 331 MHz
Mesh of 102 Scalar Octavos (17x6)

Flat: 331 MHz

Per-Lane: 489 MHz
Overhead-Free Execution

• Problems
  – Speedup ultimately limited by execution overhead
  – Addressing and flow-control (per thread)
  – Worsened by hardware assistance
Overhead-Free Execution

• Problems
  - Speedup ultimately limited by execution overhead
  - Addressing and flow-control (per thread)
  - Worsened by hardware assistance

• Solutions
  - Extract overhead as “sub-programs” (per thread)
  - Execute them in parallel along the pipeline
  - Decreases Fmax 6.1%, increases area 73%*
Sequential Sub-Programs in MIPS

outer: seed_ptr = ptr_init
inner: temp = MEM[seed_ptr]
    if (temp < 0):
        goto outer
    temp2 = temp & 1
    if (temp2 == 1):
        temp = (temp * 3) + 1
    else:
        temp = temp / 2
    MEM[seed_ptr] = temp
seed_ptr += 1
OUTPUT = temp
goto inner

- Flow-control
- Addressing
- Useful work
Sequential Sub-Programs in Octavo

outer:
ADD seed_ptr, ptr_init, 0

inner:
LW temp, seed_ptr
BLTZn outer, temp
BEVNNn even, temp
MUL temp, temp, 3
ADD temp, temp, 1
JMP output

even:
SRA temp, temp, 1

output:
SW temp, seed_ptr
ADD seed_ptr, seed_ptr, 1
SW temp, OUTPUT
JMP inner

- Flow-control
- Addressing
- Useful work
Removing Flow-Control Overhead

outer:  ADD seed_ptr, ptr_init, 0
inner:  LW  temp, seed_ptr
        BLTZn outer, temp
        BEVNNn even, temp
        MUL temp, temp, 3
        ADD temp, temp, 1
        JMP output

even:   SRA temp, temp, 1
output:  SW  temp, seed_ptr
        ADD seed_ptr, seed_ptr, 1
        SW  temp, OUTPUT
        JMP inner

- Flow-control
- Addressing
- Useful work
Parallel Sub-Programs in Octavo

outer:  ADD seed_ptr, ptr_init, 0
inner:  LW temp, seed_ptr
        BLTZn outer, temp
        BEVNN even, temp
        MUL temp, temp, 3
        ADD temp, temp, 1
        JMP output

even:  SRA temp, temp, 1
output: SW temp, seed_ptr
        ADD seed_ptr, seed_ptr, 1
        SW temp, OUTPUT
        JMP inner

• Flow-control
• Addressing
• Useful work
Parallel Sub-Programs in Octavo

outer:    ADD seed_ptr, ptr_init, 0
inner:    LW temp, seed_ptr
          MUL temp, temp, 3 ; BEVNNn even ; BLTZn outer
          ADD temp, temp, 1 ; JMP output
even:     SRA temp, temp, 1
output:   SW temp, seed_ptr
          SW temp, OUTPUT ; JMP inner

• Flow-control (folded, cancelling, multi-way)
• Addressing (indirect with post-increment)
• Useful work
Parallel Sub-Programs in Octavo

outer:  ADD seed_ptr, ptr_init, 0
inner:  LW  temp, seed_ptr
        MUL temp, temp, 3 ; BEVNN even ; BLTZn outer
        ADD temp, temp, 1 ; JMP output
even:   SRA temp, temp, 1
output: SW temp, seed_ptr
        SW temp, OUTPUT  ; JMP inner

• Flow-control (folded, cancelling, multi-way)
• Addressing (indirect with post-increment)
• Useful work
Speedups

- **Unrolled ("perfect" MIPS)**
- **Looping (modified Octavo)**

**Chart:**
- **Hailstone**: Unrolled: 1.9, Looping: 1.9
- **Increment**: Unrolled: 1.6, Looping: 1.7
- **Reverse**: Unrolled: 1.3, Looping: 1.2
- **FIR**: Unrolled: 1.1, Looping: 1.1
- **FSM**: Unrolled: 1.0, Looping: 1.0
Reduced-Overhead Octavo
Reduced-Overhead Octavo

Branch Trigger Module (BTM)

(Branches not in fetched instructions!)
Reduced-Overhead Octavo

Address Offset Module (AOM)

(One entry for each instruction operand)
Benchmarking

- Compares FPGA design processes:
  - Octavo (Multi-Threaded Soft-Processor)
Benchmarking

• Compares FPGA design processes:
  – Octavo (Multi-Threaded Soft-Processor)
  – MXP (Soft Vector Processor)
Benchmarking

- Octavo (Multi-Threaded Soft-Processor)
- MXP (Soft Vector Processor)
- LegUp (plain C HLS)
Benchmarking

• Compares FPGA design processes:
  - Octavo (Multi-Threaded Soft-Processor)
  - MXP (Soft Vector Processor)
  - LegUp (plain C HLS)
  - Verilog (hand-optimized HDL for speed)
Benchmarking

- Compares FPGA design processes:
  - Octavo (Multi-Threaded Soft-Processor)
  - MXP (Soft Vector Processor)
  - LegUp (plain C HLS)
  - Verilog (hand-optimized HDL for speed)

- All results relative to a Scalar Octavo core
• 1 to 32 SIMD Datapaths total (-L1 to -L32)
• Each Datapath has Accelerators on I/O Ports:
  – Accumulator
  – Array Reversal Channel
-V1 to -V32 Vector Lanes, with table-lookups
Sequential Speedup

- Reverse-3
- Hailstone-S
- Hailstone-A
- FSM-S
- FSM-A

Speedup vs. Scalar Octavo

MXP-V1
MXP-Nios
LegUp
HDL
Sequential Speedup

- MXP-V1
- MXP-Nios
- LegUp (20.6)
- HDL (8.81)

Speedup vs. Scalar Octavo

- Reverse-3
- Hailstone-S
- Hailstone-A
- FSM-S
- FSM-A
Parallel Speedup

![Graph showing speedup comparison between different algorithms and configurations. The x-axis represents algorithms (Increment, Hailstone-N, Reverse-4, FIR) and the y-axis represents speedup vs. Scalar Octavo. The graph includes bars for Octavo-L2, Octavo-L4, Octavo-L8, Octavo-L16, Octavo-L32, MXP-V1, MXP-V2, MXP-V4, MXP-V8, MXP-V16, and MXP-V32.]
Parallel Speedup

(23.6)

Octavo-L2
Octavo-L4
Octavo-L8
Octavo-L16
Octavo-L32
MXP-V1
MXP-V2
MXP-V4
MXP-V8
MXP-V16
MXP-V32

Speedup vs. Scalar Octavo

Increment Hailstone-N Reverse-4 FIR
Summary of Contributions

- How do we improve overlay performance?
Summary of Contributions

• How do we improve overlay performance?

1. Octavo: 500+ MHz soft-processor
   • Operates at >90% of absolute maximum on Stratix IV FPGA
Summary of Contributions

• How do we improve overlay performance?

1. Octavo: 500+ MHz soft-processor
   • Operates at >90% of absolute maximum on Stratix IV FPGA

2. Preserved performance under scaling
   • e.g.: Fmax -22% over 102x scaling
   • C. E. LaForest, J. G. Steffan, “Maximizing Speed and Density of Tiled FPGA Overlays via Partitioning”, ICFPT 2013
Summary of Contributions

• How do we improve overlay performance?

1. Octavo: 500+ MHz soft-processor
   • Operates at >90% of absolute maximum on Stratix IV FPGA

2. Preserved performance under scaling
   • e.g.: Fmax -22% over 102x scaling
   • C. E. LaForest, J. G. Steffan, “Maximizing Speed and Density of Tiled FPGA Overlays via Partitioning”, ICFPT 2013

3. Overlap execution overhead with useful work
   • Better speedup than loop unrolling on “perfect” MIPS CPU
Future Work

- Programming Support
Future Work

• Programming Support

• Further reduce execution overhead
Future Work

- Programming Support
- Further reduce execution overhead
- Increase resource diversity
Future Work

- Programming Support
- Further reduce execution overhead
- Increase resource diversity
- Increase efficiency
  - Non-branching code
  - Internal Memory bandwidth
  - Bit-level Parallelism
  - ALU Utilization
  - Measure/Reduce Power
Future Work

https://github.com/laforest/Octavo
Extra Slides
Impact on Area

![Graph showing the impact on area with various lines representing different scenarios such as Fmax, Fmax (nrdr), eALMs, eALMs (nrdr), Density, and Density (nrdr)].

- **Axis Labels:**
  - Vertical: Avg. % Difference
  - Horizontal: SIMD Lane Count

- **Legend:**
  - Red solid line: Fmax
  - Red dashed line: Fmax (nrdr)
  - Green solid line: eALMs
  - Green dashed line: eALMs (nrdr)
  - Blue solid line: Density
  - Blue dashed line: Density (nrdr)
Impact on Area

5% area gap
AOM/BTM Configurations

The graph shows the average Fmax (MHz) plotted against the number of entries per AOM and/or BTM instances. The graph includes various configurations:

- **Original Octavo** (black circles)
- 1 entry/AOM, 1-8 BTM instances (blue triangles)
- 1 BTM instance, 1-8 entries/AOM (green squares)
- 1-8 entries per AOM and BTM instances (red circles)
- 2/4, 2/8, 3/6, 4/8 entries per AOM/BTM instances (black stars)

Design points above the line can reach 500 MHz.
Logic Element (LE)
Logic Cluster (LC)
Switch Boxes and Connection Blocks

Diagram showing components and connections.
Generic Island-Style FPGA
Hard Blocks (RAM, DSP)
Interconnect Delay Dominates
Fmax vs. Memory Depth

[Graph showing the relationship between Fmax (MHz) and Memory Depth (Words) for various word widths (8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28) with corresponding Fmax values (30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72).]
Fmax vs. Memory Depth

![Graph showing Fmax vs. Memory Depth with different word widths and memory depths.](image)
Table 5.1: Octavo's Instruction Word Format with Extended Write Address Space

<table>
<thead>
<tr>
<th>Size:</th>
<th>4 bits</th>
<th>12 bits</th>
<th>10 bits</th>
<th>10 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field:</td>
<td>Opcode (OP)</td>
<td>Destination (D)</td>
<td>Source (A)</td>
<td>Source (B)</td>
</tr>
</tbody>
</table>

Write Address

0

A

B

I

H

Read Address

0

A operand (10 bits)

B operand (10 bits)

PC (10 bits)

Not Readable

D operand (12 bits)

2047

1023

0

3071

1023

0

4095
Branch Trigger Module
Branch Trigger Module
Address Offset Module
Address Offset Module
Efficiency Increase

- Unrolled ("perfect" MIPS)
- Looping (modified Octavo)

- Hailstone
- Increment
- Reverse
- FIR
- FSM
Efficiency Increase

- Unrolled ("perfect" MIPS)
- Looping (modified Octavo)

1.1 Increment
1.2 Hailstone
1.3 Reverse
1.4 FIR
1.5 FSM

(0.828)
Parallel Speedup

(23.6)

Speedup vs. Scalar Octavo

Increment Hailstone-N Reverse-4 FIR

Octavo-L2 Octavo-L4 Octavo-L8 Octavo-L16 Octavo-L32
MXP-V1 MXP-V2 MXP-V4 MXP-V8 MXP-V16 MXP-V32
LegUp HDL
Planning for Larger Systems

● Problems
  – Inefficient use of memory (overlapping)
  – Wasting I/O ports on low-traffic control hardware
  – Software busy-wait loops for I/O
Planning for Larger Systems

• Problems
  - Inefficient use of memory (overlapping)
  - Wasting I/O ports on low-traffic control hardware
  - Software busy-wait loops for I/O

• Solutions
  - Extending the write address space (2 spare bits)
  - Predicating instructions on I/O port readiness
  - Decreases Fmax 4.5%, increases area 5.5%
Instruction I/O Predication

- Empty/Full bit on each read/write I/O port
- Annul instruction if not all addressed ports ready
- Re-issue instruction next thread cycle