Achieving Superscalar Performance w/o Superscalar Overheads – A Dataflow Compiler IR for Custom Computing

Ali Mustafa Zaidi
David Greaves (supervisor)
University of Cambridge
Computer Laboratory

![Graph showing relative performance and power dissipation comparison between Superscalar and Custom solutions]
The Dark Silicon Problem

Amdahl's Law

+ 

Utilization Wall

= 

Dark Silicon

\[ \text{Perf} = \frac{1}{(1-f) + \frac{f}{s_{\text{seq}} n}} \]

The Dark Silicon Problem

Amdahl's Law + Utilization Wall = Dark Silicon

\[ \text{Perf} = \frac{1}{(1-f)} + \frac{f}{s_{\text{seq}} \cdot n} \]

2.1GHz @ 90nm (80W)  5.2GHz @ 45nm (80W)  7.3GHz @ 32nm (80W)

18%  7%  3%

The Dark Silicon Problem

Amdahl's Law

\[ \text{Utilization Wall} = \frac{1}{(1-f)} + \frac{f}{s_{\text{seq}}} \]

\[ \text{Dark Silicon} \]

Utilization Wall

45nm → 8nm (32x resources)

- CPU: 3.5x, GPU 2.4x (Cnsrv.)
- CPU: 7.9x, GPU 2.7x (ITRS)

The Dark Silicon Problem

Amdahl's Law

Utilization Wall

= Dark Silicon

Need for both high 'Sequential' Performance, AND Very High Energy Efficiency

45nm → 8nm (32x resources)
- CPU: 3.5x, GPU 2.4x (Cnsrv.)
- CPU: 7.9x, GPU 2.7x (ITRS)

The Dark Silicon Problem

Amdahl's Law

+ Utilization Wall

= Dark Silicon

Superscalar Processors

- Only Option for Seq. Performance!
- Power scales exponentially with Performance

The Dark Silicon Problem

Amdahl's Law

\[ + \]

Utilization Wall

\[ = \]

Dark Silicon

Superscalar Processors

- Only Option for Seq. Performance!
- Power scales exponentially with Performance

Conventional

Spatial?

Custom Hardware:

- 10 – 1000 x Efficiency!
- Not for Sequential!

The Dark Silicon Problem

Amdahl's Law

\[ \text{Utilization Wall} = \text{Dark Silicon} \]

Superscalar Processors

- Only Option for Seq. Performance!
- Power scales exponentially with Performance

Custom Hardware:

- 10 – 1000 x Efficiency!
- Not for Sequential!

Can we achieve Superscalar Performance, w/o Superscalar Overheads?

The Dark Silicon Problem

Amdahl's Law + Utilization Wall = Dark Silicon

Superscalar Processors
- Only Option for Seq. Performance!
- Power scales exponentially with Performance

Conventional
Spatial?

Custom Hardware:
- 10 – 1000 x Efficiency!
- Not for Sequential!

Can we compile from HLL code to custom hardware?
Can we match (or beat) superscalar performance?

Key Reasons for High Performance of Complex, OOO Superscalars:

- Aggressive Control-flow Speculation
- Dynamic, out-of-order execution scheduling

Custom hardware has very limited speculation:

- Single flow of control
- If-conversion & hyperblock formation for forward branches.
- **No acceleration of backwards branches!**

Solution: Custom Hardware?

---

McFarlin et al., “Discerning the dominant out-of-order performance advantage: is it speculation or dynamism?”, ASPLOS ’13
Solution: Custom Hardware!

Control-Data Flow Graph

```
Start
```

```
i = 0
```

```
A[i] > 0
```

```
i++
```

```
i < 100
```

Our Solution

Instead of

CDFG IR + Compile-time Execution Scheduling

We Employ

VSFG IR + Dataflow Execution Model

```
for (i = 0; i < 100; i++)
    if (A[i] > 0) foo();
    bar();
```
Overcoming Control-Flow with the VSFG

for (i = 0; i < 100; i++)
    if (A[i] > 0) foo();
bar();

- **Hierarchical Dataflow Graph**
  - Instead of [Basic Blocks + Control Flow], we have [Nested Subgraphs + Dataflow]
  - Functions → nested subgraphs
  - Loops → tail-recursive functions.

- **Dataflow execution of operations**
  - Multiple Subgraphs may execute concurrently in Dataflow order (unlike basic blocks).
  - Exposes **Multiple Flows of Control!**
Overcoming Control-Flow with the VSFG

Infinite DAG
- Loops represented as Tail Recursion
- Branches represented via if-conversion
- Enables **Aggressive Speculation**!

No single 'Flow of Control'
- Instead, control implemented via 'Boolean Predicate Expressions'.
- Logic minimization can simplify expressions, facilitating **Control Dependence Analysis**!

```c
for (i = 0; i < 100; i++)
    if (A[i] > 0) foo();
bar();
```
Overcoming Control-Flow with the VSFG

```plaintext
for (i = 0; i < 100; i++)
    if (A[i] > 0) foo();
    bar();
```

- **Hierarchical Dataflow Graph**
  - Subgraphs may be 'predicated', or executed speculatively (via 'if-conversion').
  - 'Flattening' loop tail-call subgraphs → loop unrolling/pipelining.
  - Multiple loops in a loop-nest may be unrolled independently to expose ILP
Overcoming Control-Flow with the VSFG
Overcoming Control-Flow with the VSFG
High Level Synthesis Case Study

Any High Level Language → LLVM → VSFG → Low-Level IR → Bluespec SystemVerilog → ASIC / FPGA

%1 = mul i32 %x, %y;
%2 = srem i32 %1, %z;
%3 = icmp slt i32 %2, %1;

FIFOF(int) x ← mkFIFOF1;
FIFOF(int) y ← mkFIFOF1;
FIFOF(int) z ← mkFIFOF1;
FIFOF(int) srem_1 ← mkFIFOF1;
FIFOF(int) icmp_1 ← mkFIFOF1;
FIFOF(int) icmp_2 ← mkFIFOF1;
FIFOF(int) out_3 ← mkFIFOF1;

rule mul_inst;
  let val1 = x.first; x.deq;
  let val2 = y.first; y.deq;
  let rslt = val1 * val2;
  srem_1.enq (rslt);
  icmp_1.enq (rslt);
endrule

rule srem_inst;
  let val1 = srem_1.first; srem_1.deq;
  let val2 = z.first; z.deq;
  let rslt = val1 % val2;
  icmp_2.enq (rslt);
endrule
Hardware Oriented Dataflow IR

- Performance and Energy Evaluation by comparing with
  - LegUp HLS Tool, & Altera Nios IIIf Processor, implemented on Altera Stratix IV GX FPGA.
  - Nehalem Core i7 (Sniper interval simulator from Intel).
  - In all cases, memory access latency assumed == 1 Cycle.

- LegUp
  - LLVM 2.9
  - O2
  - No LTO, no LTI
  - No Op Chaining
  - Statically Scheduled CFG

- Our Toolchain
  - LLVM 2.6
  - O2
  - No LTO, no LTI
  - No Op Chaining
  - Dynamically Scheduled VSFG
Performance (Cycle Counts)

Matrix Transpose (x1k cycles)

<table>
<thead>
<tr>
<th></th>
<th>Core i7</th>
<th>Nios 2f</th>
<th>LegUp</th>
<th>VSFG_0</th>
<th>VSFG_1</th>
<th>VSFG_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>3400</td>
<td>1078</td>
<td>1062</td>
<td>528</td>
<td>265</td>
<td></td>
</tr>
</tbody>
</table>

adpcm (x1k cycles)

<table>
<thead>
<tr>
<th></th>
<th>Core i7</th>
<th>Nios 2f</th>
<th>LegUp</th>
<th>VSFG_0</th>
<th>VSFG_1</th>
<th>VSFG_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>120</td>
<td>71</td>
<td>58</td>
<td>52</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

dfsin (x1k cycles)

<table>
<thead>
<tr>
<th></th>
<th>Core i7</th>
<th>Nios 2f</th>
<th>LegUp</th>
<th>VSFG_0</th>
<th>VSFG_1</th>
<th>VSFG_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>1421</td>
<td>106</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>

Neural Net Simulator (x1M cycles)

<table>
<thead>
<tr>
<th></th>
<th>Core i7</th>
<th>Nios 2f</th>
<th>LegUp</th>
<th>VSFG_0</th>
<th>VSFG_1</th>
<th>VSFG_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>373</td>
<td>142</td>
<td>114</td>
<td>98</td>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

Compared to Nios II/f & Intel Nehalem Core i7 (SniperSim)
Power and Speculation Overheads

Power estimation assuming 250MHz operating frequency
Power and Speculation Overheads

Power estimation assuming 250MHz operating frequency
Normalized Energy

![Normalized Energy Chart]

Legend:
- LegUp
- VSFG_0
- VSFG_1
- VSFG_3
- Nios
Sources of Energy Inefficiency

- Energy Cost Comparison:
  - vs Nios II/f: 0.25 x (GEOMEAN)
  - vs LegUp: 3-4 x (GEOMEAN)

- Overheads of Speculation
  - Balance between speculation & predication must be found for efficiency & performance

- Part of power dissipation proportional to Area
  - Clock Gating for predicated regions to reduce dynamic power
    - (consider asynchronous Ckts)
  - Power gating for predicated regions to reduce static power?
  - Selective loop unrolling.
Limitations on Performance

- 35% better performance than statically scheduled CFG, without any optimizations:
  - Improvements due to dynamic scheduling, MFC & CDA
  - Unrolling helps, but speed-up saturates quickly.

- Further Improvements possible:
  - Balance between **predication** & **speculation**, to improve speed-up without unrolling (thus reducing area and energy costs)
  - State-edge is on critical path – limits both unrolling & MFC.
    - Last remnant of 'sequential' nature of program.

- Frequency Scaling limited by Memory Interconnect
  - Partition memory & pipeline memory access tree
Thank You
Solution: Spatial Architectures?

- Custom Hardware, FPGAs, CGRAs, MPPAs, etc.

- **Advantages**
  - Scalable, Decentralized architectures, with short, p2p wiring.
  - High Computational Density
  - 10-1000x Energy and Performance efficiency.

- **Issues**
  - Poor Programmability: often requiring low-level hardware knowledge
  - Limited Amenability: poor performance on sequential, irregular, or complex control-flow code.

- **Examples**
  - Conservation Cores: Performance ≈ in-order MIPS24KE core
  - Phoenix CASH Hardware: Performance 30% less than 4-way OOO Core.