SIGIL
A tool for assisting acceleration selection

Siddharth Nilakantan, Parnian Mokri, Mike Lui
and Mark Hempstead
Drexel University
Outline

- **Accelerator Selection Problem**
  - Example

- Sigil Overview

- Sigil Methodology for Accelerator Selection

- Partitioning Example
What is accelerator selection?

- Which functions to accelerate?
What is accelerator selection?

- Which functions to accelerate?
  - What are limiting factors for selection?

Dependence Chain

A → B → C → D
What is accelerator selection?

- Which functions to accelerate?
  - What are limiting factors for selection?

**Dependence Chain**

<table>
<thead>
<tr>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
</tbody>
</table>
What is accelerator selection?

- Which functions to accelerate?
- What are limiting factors for selection?

Dependence Chain:
- A → B → C → D

Software:
- A  B  C  D

Compute Operations:
- A  B  C  D
Accelerator selection

Selection 1

S/W

A

C

D

H/W

B

Time

CPU time

Comm. time

Accel. time
Accelerator selection

Selection 1

S/W: A | C | D
H/W: B

Selection 2

S/W: A | C
H/W: B | D

Time

- Dark blue: CPU time
- Yellow: Comm. time
- Light blue: Accel. time
Accelerator selection

Selection 1

Selection 2

Selection 3

Time

CPU time
Comm. time
Accel. time
Platform-independent metrics

- Accelerator time and communication time are implementation-dependent!
  - Large design space for implementations

- Early stage design approach: Capture platform-independent metrics as proxy
  - Accelerator time $\rightarrow$ Compute operations
  - Communication Time $\rightarrow$ I/O set of bytes for each function
Capturing Input/Output Set

- Input/Output set: **NOT all** memory reads and writes, only unique ones
- Biggest challenge: Measuring *unique* communication

![Diagram showing flow of I/O bytes between Function A and Function B with nodes for Compute, Wr Addr, and Rd Addr, indicating bytes added to I/O set and bytes marked as reuse.]
Novelty: **Sigil** measures these metrics *automatically*
- Classifying communication (unique and total bytes)
- Compute operations for each function
- Produces control data flow graph (CDFG) representations

Revisit the Q: Which functions to accelerate?
- Apply HW/SW partitioning algorithm to graphs!
- Goals of algorithm
  - Minimize *unique* communication
  - Maximizing coverage in HW
Outline

- Accelerator selection problem

- Sigil Overview

- Sigil Methodology for Accelerator Selection

- Partitioning Example
Sigil Implementation

- Implemented on top of Callgrind
- Works on binary, no source changes
- Can be implemented on any framework. Requires
  - Functions
  - Load/Store addresses

- Control data flow graph
- Unique and local communication costs and edges
- Cache simulation
- Branch prediction
- Dynamic binary instrumentation
- VEX IR generation
Tracking unique communication
Tracking unique communication

1. Function A
   - Write
   - Addr. 1

2. Function B
   - Read
   - Addr. 1

3. Monitor

4. Shadow Memory
Tracking unique communication

1. Write Addr. 1
2. Read Addr. 1

Function A

Function B

Monitor

Update last writer

Shadow Memory
Tracking unique communication

Function A
- Write
  - Addr. 1

Function B
- Read
  - Addr. 1

Monitor

Shadow Memory
Tracking unique communication

1. Function A
   - Write
   - Addr. 1

2. Function B
   - Read
   - Addr. 1

3. Monitor

4. Shadow Memory

5. Last Writer

Diagram shows the flow of communication functions and how they interact with a monitor and shadow memory.
Tracking unique communication

1. Write Addr. 1
2. Read Addr. 1
3. Last Writer
4. Unique/non-unique bytes
Inside Shadow Memory

ST Addr, Register in Function A

LD Register, Addr in Function B

Primary Map

Addr[34:16]

Addr[15:0]

Secondary Maps

Last Writer = Func A

Last Reader = None

Last Reader Call = 0
Inside Shadow Memory

Primary Map

Addr[34:16]

0 0 ....

Addr[15:0]

Secondary Maps

ST Addr, Register in Function A

Last Writer = Func A

Last Reader = Func B

Last Reader Call = 1

ST Addr, Register in Function A

LD Register, Addr in Function B
Outline

- Accelerator selection problem
- Sigil Overview

- **Sigil Methodology for Accelerator Selection**
  - Control Data flow graphs
  - Partitioning process

- Partitioning Example
Control Data Flow graphs

- Function calltrees...
  - Hierarchical representation of functions in application
  - Obtained via Callgrind
Control Data Flow graphs

- Function calltrees annotated with *unique* communication flow
- Obtained via Sigil
Function calltrees annotated with *unique* communication flow
- Add computation costs in as well
- Also obtained via Sigil
HW/SW partitioning process

- How to pick accelerator candidates in hierarchical CDFG?
HW/SW partitioning process

How to pick accelerator candidates?

- Leaf nodes are self contained – Natural candidates
  - If coverage of work too low?
HW/SW partitioning process

- How to pick accelerator candidates?
  - Leaf nodes are self contained – Natural candidates
  - Non-leaf nodes? *Include* functionality of sub-calltree
Calculate inclusive costs

- Non-leaf nodes: Merge sub-calltree
  - Inclusive computation costs – Add up operations
  - Inclusive communication costs – Edges crossing the box
Outline

- Accelerator selection problem
- Sigil Overview
- Sigil methodology for accelerator selection

- Partitioning examples
  - In-depth look: 456.Hmmer
  - Results: Multiple benchmarks
Partitioning algorithm

- Employ any partitioning algorithm

- Existing algorithms
  - Intuitive: Computation to Communication ratio
  - State-of-the-art: Simulated Annealing, Genetic algorithms

- We use a **demonstrative** algorithm utilizing:
  - software time — from Callgrind
  - communication time — from Sigil
  - compute time — from Sigil

- Does not indicate amenability of functions
  - HLS tools show amenability
Partitioning example: Spec 456.Hmmer

Assumptions
- For SW time – 2.5GHz CPU
- For Comm. Time – 16GB/s transfer rate

After partitioning
- Call edges shown
- Communication edges not shown
- Candidates in box
Partitioning example: Spec 456.Hmmer

Comm. → % Communication cost of merged function/Total communicated bytes in program

- Rank statistics
  - S/W, Flops/llops and Communication bytes coverage %

- FChoose
  - Cycles 14%
  - Ops 10%
  - Comm. 0.39%

- Random Seq.

- Gauss Random
  - Cycles 0.05%
  - Ops 0.01%
  - Comm. 1.82%

- Digitize Seq.
  - Cycles 10%
  - Ops 4.7%
  - Comm. 49%

- P7 Viterbi
  - Cycles 74%
  - Ops 83%
  - Comm. 4.62%

- Other functions
Functions from *demonstrative* partitioning for PARSEC benchmarks

<table>
<thead>
<tr>
<th>Rank</th>
<th>Blackscholes</th>
<th>Freqmine</th>
<th>Dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>String to float</td>
<td>sort</td>
<td>sha1_block_data_order</td>
</tr>
<tr>
<td>2</td>
<td>ieee754_exp</td>
<td>FP_Array_scan2*</td>
<td>sha1_block_data_order</td>
</tr>
<tr>
<td>3</td>
<td>ieee754_expf</td>
<td>sort</td>
<td>compress2*</td>
</tr>
<tr>
<td>4</td>
<td>ieee754_logf</td>
<td>FP_Array_scan2*</td>
<td>write_file*</td>
</tr>
</tbody>
</table>

- `ieee_754/mul` – IEEE “math” library functions
- `sha1_block_data_order` – core of SHA1 calculation
- `FP_Array_scan2` – Builds “prefix-tree” for frequent pattern mining [1]

* → merged function

Partitioning Results - PARSEC

- **S/W Coverage with accelerator candidates**

The diagram shows the normalized software time for various applications. The applications include `blackscholes`, `bodytrack`, `canneal`, `dedup`, `fluidanimate`, `facesim`, `ferret`, `fregmine`, `raytrace`, `streamcluster`, `swaptions`, `vips`. The bars are divided into two colors: yellow for software functions and red for accelerator candidates.
Looking forward

- We plan on releasing results from SPEC, PARSEC, BioBench and more

- Commonality of functions between applications
  - Area may be free, design and verification are not

- Need more applications!
  - Run Sigil on your workload and tell us what you find
Getting Sigil

- Available open source
  - `git clone https://github.com/snilakan/Sigil`
  - Documentation included

- Tested and validated in Linux
  - Officially supported distros: CentOS6, Ubuntu 12.04 LTS, Ubuntu 14.04 LTS
  - Theoretically, supported by any system supported by Valgrind (3.10.1)
Automated script
- Checks dependencies and builds Valgrind with Sigil/Callgrind
- Configures post processing scripts

Manual build process
- Typical autotools flow (./configure, make, etc)
- Will have to modify post processing scripts
  - See documentation
Running Sigil

- Compile `<user_program>` with debug flags

- Generate CDFGs
  - `./run_sigil.sh <user_program>`

- Partitioning the graph
  - `./aggregate_costs.py -help`
  - Can plug in your own partitioning algorithm!
Sigil Release

- Official Website
  - `git clone https://github.com/snilakan/Sigil`
  - Or download zip from link
  - Contact: michael.d.lui@drexel.edu

- Related Publications
  - “Platform-independent Analysis of Function-level Communication in Workloads”, Siddharth Nilakantan and Mark Hempstead, IISWC 2013
  - “Metrics for Early-Stage Modeling of Many-Accelerator Architectures”, Siddharth Nilakantan, Steven Battle and Mark Hempstead, CAL July-Dec 2012
BACKUP SLIDES
Partitioning steps

- First, use a metric to compare nodes against parents
  - Merge nodes when parents make better candidates
- Second, rank leaf nodes by same metric
Metric for merging & ranking

- Breakeven-speedup
  - Minimum factor of computational acceleration, given communication
  - For calculation of communication; we can plug in a transfer rate

Breakeven-speedup = \[
\frac{t_{sw}}{t_{sw} - (t_{comm:ip:accel} + t_{comm:op:accel})}
\]