Software Developer Tools for Democratizing Heterogeneous Computing

ISSTA 2022 Keynote

Miryung Kim
UCLA Computer Science
Software Engineering Analysis Laboratory
Software Engineering Analysis Lab at UCLA

- Code Mining, Debugging and Refactoring for Java
- Debugging and Testing Tools for Big Data
- Systems and Runtimes
- Developer Tools for Heterogeneous Computing

DA4SE
Data Analytics for Software Engineering

SE4DA
Software Engineering for Data Analytics

A new wave of SE tools for data intensive computing
SW developer tools
<> Heterogeneous HW
Outline

• Why heterogeneity now?
• What does heterogeneity look like?
• What are the implications of heterogeneity?
• High-level synthesis developer workflow
• Examples of SW developer tools for heterogeneity
• Opportunities and challenges
Why heterogeneity now?
A new era of golden age of architectures

End of Moore’s Law and Dennard Scaling [CACM 2019]
Cloud is shifting to HW heterogeneity

Increasing Heterogeneity of Cloud Hardware [SIGSOPS 2020]
Hardware accelerators are widely available
What does heterogeneity look like?
CPU, GPU, FPGA and ASICs tradeoffs
Field programmable gate array (FPGA)

Programmable logics, interconnects, and customizable building blocks

Catapult – Bing search with FPGA-enabled servers 50% throughput increase and 25% latency reduction.

Difficult to program in RTL languages
Application-specific integrated circuit (ASIC)

**TPU** for accelerating deep-learning workloads

80X performance–per-watt advantage over CPU

Design cycle is long and costly.
What are the implications of heterogeneity?
US bureau of labor statistics

1.8 M software developer

70000 hardware engineers

Towards Democratized IC Design and Customized Computing, 2022
Raising the abstraction level of HW design

Programmability (languages features, etc.)

- FPGA
- HDL
- HLS C/C++
- 1989
- 2003
- 2011
- 2014
- 2017
- VIVADO
- XILINX VITIS
- oneAPI
- evolving
- Year

UCLA School of Engineering
What is developer workflow with high level synthesis?
int KNN()
...

// Calculate distance
for (i = 0 to number){
    dist[i] = l2norm(data[i], dim);
}
// Top 1 nearest neighbor
...

// Top 1 nearest neighbor
...
High level synthesis (HLS) for FPGA

- Host Code
- CPU
  - C Execution
  - Co Simulation
  - FPGA Execution (fast)
- Kernel Code
- High-Level Synthesis
  - HLS C/C++
  - HDL (Verilog / VHDL)
- RTL Synthesis
  - Gate-Level Bit Stream
High level synthesis (HLS) for FPGA

- Host Code
  - C Execution (in seconds)
  - Co Simulation (in minutes to hours)
  - FPGA Execution (fast)

- Kernel Code
  - High-Level Synthesis
    - HLS C/C++ (compile in minutes to hours)
    - HDL (Verilog / VHDL)
  - RTL Synthesis
    - FPGA synthesis in hours or days

- FPGA synthesis in hours or days

- Gate-Level Bit Stream
int KNN()
...

// Calculate distance
for (i = 0 to number){
    dist[i] = l2norm(data[i], dim);
}
// Top 1 nearest neighbor
...

1. Performance profiling
2. Kernel function identification in C
3. Manual rewriting from C to HLS-C
4. Differential testing with input samples (RTL simulation vs. C execution)
5. Iterative optimization

7X speed up on FPGA

FPGA synthesis in 2.5 hours

CPU-FPGA co-simulation 8 minutes

HLS compilation to RTL 6 minutes

Repeat
HLS tools are not easy to use for SW developers

- Resource finitization
- Hardware expertise and pragmas for optimization
- Partitioning, parallelization, pipelining, etc.

Manual rewriting for synthesizability and optimization
No developer tools for code translation
HLS-C requires specifying \textit{bitwidth} for each type.

\begin{Verbatim}
float vecdot(
    float a[],
    float b[],
    int n) {
    for (int i = 0; i < n; i++)
        sum += a[i] * b[i];
    return sum;
}
\end{Verbatim}

\begin{Verbatim}
float vecdot(
    float a[],
    float b[],
    \texttt{fpga\_int<7>} n) {
    for (\texttt{fpga\_int<7>} i = 0; i < n; i++)
        sum += a[i] * b[i];
    return sum;
}
\end{Verbatim}
HLS-C uses a *custom* floating point type

C Program

```c
float vecdot(
    float a[],
    float b[],
    fpga_int<7> n) {
    for (fpga_int<7> i = 0; i < n; i++)
        sum += a[i] * b[i];
    return sum;
}
```

HLS-C Program

```c
fpga_float<8,15> vecdot(
    fpga_float<8,15> a[],
    fpga_float<8,15> b[],
    fpga_int<7> n) {
    for (fpga_int<7> i = 0; i < n; i++)
        sum += a[i] * b[i];
    return sum;
}
```
HLS-C requires \textit{finitializing} resources

```
struct Node {
    Node *left, *right;
    int val;
};
void init(Node **root) {
    *root = (Node *)malloc(sizeof(Node));
}
void delete_tree(Node *root) {
    free(root);
}
void traverse(Node *curr) {
    if (curr == NULL) return;
    int ret = visit(curr->val);
    traverse(curr->left);
    traverse(curr->right);
}
```

```
Node Node_arr[NODE_ARR_SIZE];
struct Node {
    Node *left, *right;
    int val;
};
void delete_tree(Node_ptr root) {
    ... node_free(root);
}
void traverse_converted(Node_ptr curr) {
    stack<context> s(STACK_SIZE);
    while (!s.empty()) {
        ...}
    ```
Performance boost is *not automatic* with HLS

// Convolution
for (int j = 0; j < NumIn; ++j) {
    for (int h = 0; h < ImSize; ++h) {
        for (int w = 0; w < ImSize; ++w) {
            for (int po = 0; po < ParallelOut; po++) {
                for (int p = 0; p < kKernel; ++p) {
                    for (int q = 0; q < kKernel; ++q)
                        C[po][h][w] += weight(i, po, j, p, q) * input(j,h + p,w + q);
                }
            }
        }
    }
}

Source: “Towards Democratized IC Design and Customized Computing, 2022

7-line CNN: **Initially 108X slower** with a commercial HLS tool.
After 28 pragmas and proper restructuring, **89X faster**.
Computing power locked in a few hands

Less than 5% of software developers are able to make use of HLS effectively.
SW developer tools for democratizing heterogeneity
HeteroFuzz: Fuzz Testing to Detect Platform Dependent Divergence for Heterogeneous Applications

Qian Zhang, Jiyuan Wang, Miryung Kim

ESEC/FSE 2021
Divergence errors between CPU and FPGA

### Host Code

```c
int main(int argc, char *argv[]){
    int data[] = gradient(argv[1]);
    int sum;
    float th = argv[2];
    int size = data.size();
    accumulate(data[size]);
    for(i = 0 to size){
        data[i] /= sum;
        if(data[i] > th)
            discard;
    }
}
```

### Kernel Code

```c
int accumulate(int data[size]){
    typedef ap_uint<8> bit8;
    #define max M;
    bit8 sum = 0;
    bit8 data_fpga[M];
    for(i = 0 to M){
        data_fpga[i]=(bit8)data[i];
    }
    SUM_LOOP for(i = 0 to M){
        #pragma HLS unroll factor=2
        sum += data_fpga[i];
    }
    return sum;
}
```

<table>
<thead>
<tr>
<th>Input</th>
<th>CPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,1,1,253]</td>
<td>no errors</td>
<td>div/0 in host</td>
</tr>
<tr>
<td>[2,1,1,253]</td>
<td>257</td>
<td>1</td>
</tr>
</tbody>
</table>

Testing

Translation
Is **fuzz testing** applicable?

![Diagram showing the process of fuzz testing]

1. **Input**
   - Pick
   - Mutate
   - Execute

   - Add Input’
     - Yes
     - New Branch Coverage?
       - Yes
       - Feedback
         - Program
       - No
         - X

   - Test
AFL running time for finding errors

---

### Table 1: Examples of Behavior Divergence Between CPU and FPGA

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>894069</td>
<td>Segmentation fault when allocating a big array int x[1920][1080] on FPGA yet no error on CPU</td>
<td>8.5h</td>
</tr>
<tr>
<td>595225</td>
<td>Different outcome caused by HLS dataflow directive</td>
<td>47.7h</td>
</tr>
<tr>
<td>438446</td>
<td>Different outcome caused by FPGA fetching incorrect struct vector training_set[MAXSZ]</td>
<td>10.6h</td>
</tr>
<tr>
<td>754676</td>
<td>Different outcome caused by bitwidth typedef ap_fixed&lt;25,1,AP_RND&gt; s25f24_type</td>
<td>2.3h</td>
</tr>
<tr>
<td>785019</td>
<td>Getting all zeros when shifting an array caused by #pragma HLS RESET</td>
<td>3.1h</td>
</tr>
<tr>
<td>907213</td>
<td>Undecided output when overwriting a same variable within the loop yet no error on CPU</td>
<td>79.4h</td>
</tr>
<tr>
<td>1166264</td>
<td>EMFILE error when loading 2048 files with #pragma HLS ARRAY_PARTITION yet no error on CPU</td>
<td>11.7h</td>
</tr>
<tr>
<td>1126600</td>
<td>The 25-tap FIR filter bypasses some input multiplications with #pragma HLS PIPELINE</td>
<td>93.2h</td>
</tr>
</tbody>
</table>

AFL: American Fuzzy Lop (a well known fuzz testing framework)
Challenge 1: lack of guidance in HW

Branch coverage is not meaningful in HW
Challenge 2: lack of effective mutations

Input mutations must stretch HW behavior in terms of finitized resource usages to induce errors
Fuzzing assumes the program under test can execute quickly in the order of **milliseconds**.
HeteroFuzz Overview

Challenge:
- lack of guidance in HW
- lack of effective mutations
- long simulation time

Heterogeneous Applications:
- Accelerator Spectra Monitoring
- Probabilistic Mutation
- Selective HLS Invocation
Accelerator spectra monitoring

```
int accumulate(int data[size]){
    typedef ap_uint<8> bit8;
    #define max M;
    bit8 sum = 0;
    bit8 data_fpga[M];
    for(i = 0 to M){
        data_fpga[i]=(bit8)data[i];
    }
    SUM_LOOP for(i = 0 to M){
        #pragma HLS unroll factor=2
        sum += data_fpga[i];
    }
    return sum;
}
```

```
int main(int argc, char *argv[]){
    int data[] = gradient(argv[1]);
    int sum;
    float th = argv[2];
    int size = data.size();
    accumulate(data[size]);
    for(i = 0 to size){
        data[i] /= sum;
        if(data[i] > th)
            discard;
    }
    return 0;
}
```

Fuzzing Guidance
Kernel input: [1,1,1,9]
Accelerator Feedback
Data_fpga: [1,9]
Sum: [2,12]
Accessed offsets: [0,1,2,3]
loop iterations: 2

Host Feedback
The activated branches

Inject accelerator specific monitors

Static analysis of HLS pragmas

Testing
Translation
## HeteroFuzz evaluation

<table>
<thead>
<tr>
<th>Effectiveness</th>
<th>Speed up</th>
<th>Efficiency</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>divergence-inducing inputs with accelerator spectra monitoring</td>
<td>with dynamic probabilistic mutation</td>
<td>with selective HLS invocation</td>
<td>with three-pronged optimizations in finding divergence errors</td>
</tr>
</tbody>
</table>

| 57% | 17.5X | 8.8X | 754X |

Testing

Translation

_UCLA Samuehi School of Engineering_
HeteroGen: Transpiling C to Heterogeneous HLS Code with Automated Test Generation and Program Repair

Qian Zhang, Jiyuan Wang, Harry Xu, Miryung Kim

ASPLOS 2022
Is search-based repair applicable?

Automated program repair (2008 ~)

Fault Localization  Candidate Repair Generation  Fitness Evaluation
(1) Long compilation & run (2) a large search space

Fault Localization -> Candidate Repair Generation -> Fitness Evaluation

Oracle -> Fix Patterns

0.2s in GenProg vs. 14 minutes in HLS

Genprog: a generic method for automatic software repair 2011
HeteroGen overview

Challenge

The search-space of small edits is too large

Heterogeneous Applications

Encode composite edits

Dependence-guided repair

Early candidate rejection

HLS compilation and simulation takes minutes to hours
1. Encode HLS repairs with composite edits

An error study based on Xilinx 1000 posts.

- Dynamic Memory Allocation/Deallocation
- Dataflow Optimization
- Type Error
- Loop Optimization
- Top Function
- Struct Error

![Pie chart with error categories]

![Diagram of HLS repair components]

- array_static($a1:arr)
- update_allocator()
- insert_allocator()
- pointer($v1:ptr)
- update_size($v1:var)
- static_stack($a1:var)
- insert_guard($v1:var)
- insertPragma($v1:var)
2. Dependence-guided repair exploration

- **Dependence-guided search** helps construct valid edits and prune the search space of potential repairs
  - 1
  - 1 -> 2
  - 1 -> 3
  - 1 -> 2 -> 5
3. Early candidate rejection

- **LLVM-level style check**
  - If a repair does not conform to HLS coding styles, it does not need to be compiled.

14 mins full HLS compilation and HW simulation

**vs.**

1 second conformance checking

```c
void foo (...) {
    int8 array1[M];
    int12 array2[N];
    ...
    #pragma HLS unroll
    skip_exit_check factor=4
    loop_2: for(i=0;i<M;i++) {
        array1[i] = ...
        array2[i] = ...
        ...
    }
    ...
}
```
## HeteroGen evaluation

<table>
<thead>
<tr>
<th>Effectiveness</th>
<th>Coverage</th>
<th>Speed-up</th>
<th>Automation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>90%</strong></td>
<td><strong>97%</strong></td>
<td><strong>35X</strong></td>
<td><strong>~438 lines</strong></td>
<td></td>
</tr>
<tr>
<td>HeteroGen produces an HLS-compatible version for 9 out of 10.</td>
<td>Auto-generated ~2500 inputs cover 97%, while pre-existing tests reach 36% coverage.</td>
<td>Dependence-based search contributes to 35X speedup than the one without.</td>
<td>It automates up to 438 lines.</td>
<td>It produces a HLS version 1.63X faster than the original C</td>
</tr>
</tbody>
</table>
How can our SE community contribute?
1. Programmability

Context:
- domain specific language [Halide] [HeteroCL] [SPIRAL]
- one API to target many platforms
- cross-industry, multi-vendor programming model [Intel’s oneAPI] [DOE’s IRIS runtime]

Opportunities:
- automated refactoring
- code recommendation for inserting pragmas (HW hints)

Challenges:
- fewer examples than Python/Java/C/C++
- ML accuracy vs. performance tradeoffs
2. Debuggability

Context:
• in-circuit debugging [Kourfali et al.]
• HLS debugging via source to source transformation [Calagar et al. Hemmert et al.]
• software monitors and tracing [MOP] [Phosphor], etc.

Opportunities:
• combine SW monitoring and HW probes

Challenges:
• difficulty with injecting HW probes
• slow execution and simulation
• overhead
3. Testing

Context:
• grey-box fuzzing [AFL] [HeteroFuzz]
• symbolic execution [Klee] [JPF] [Cute]
• search-based testing [EvoSuite], etc.

Opportunities:
• HW acceleration for fuzzing
• fuzzing guidance with HW probes
• efficient search strategies based on HW design hints

Challenges:
• slow execution and simulation
4. Compiler correctness

**Context**
- deep layers of compilation flows [Halide] [HeteroCL]
- frequent compiler extension [MLIR]

**Opportunities:**
- automatic program generation for testing compilers & extensions

**Challenges:**
- slow execution and simulation
- large design space exploration search space
Thank you!

Thanks to Qian Zhang, Jiyuan Wang, Muhammad Ali Gulzar, Jason Lau, Aishwarya Sivaraman, Jason Cong, Harry Xu, Hongbo Rong, Adrian Sampson, Rohan Padhye, Jason Teoh, Fabrice Harel-Canada, Yifan Qiao, Haoran Ma
## Debugging and Testing Tools for Big Data

- **HeteroGen, HeteroFuzz, HeteroRefactor, QDiff**
- **BigDebug, BigSift, BigTest, BigFuzz, PerfDebug, FlowDebug, OptDebug, Titian**
- **ExampleCheck, ExampleStack, Examplore, Jdebloat, Jshrink, Critics, Lase, Alice, etc.**
- **Semeru, Dorylus, Mapo, etc.**

## Systems and Runtimes for Memory Disaggregation

- **Developer Tools for Heterogeneous Computing**
- **HeteroGen, HeteroFuzz, HeteroRefactor, QDiff**
- **BigDebug, BigSift, BigTest, BigFuzz, PerfDebug, FlowDebug, OptDebug, Titian**
- **ExampleCheck, ExampleStack, Examplore, Jdebloat, Jshrink, Critics, Lase, Alice, etc.**
- **Semeru, Dorylus, Mapo, etc.**

## Example Checkers

- **Jdebloat, Jshrink, Critics, Lase, Alice, etc.**
Q&A