UCLA Samueli School of Engineering

ACM SIGSOFT International Symposium on Software Testing and Analysis

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

github stackoverflow

Debugging and Testing Tools for Big Data

Spark hadoop

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

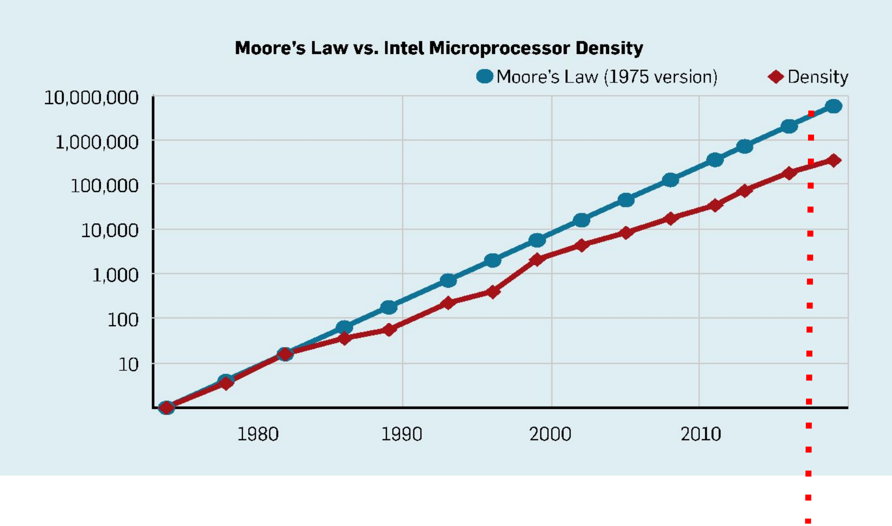
SW developer tools <> Heterogeneous HW

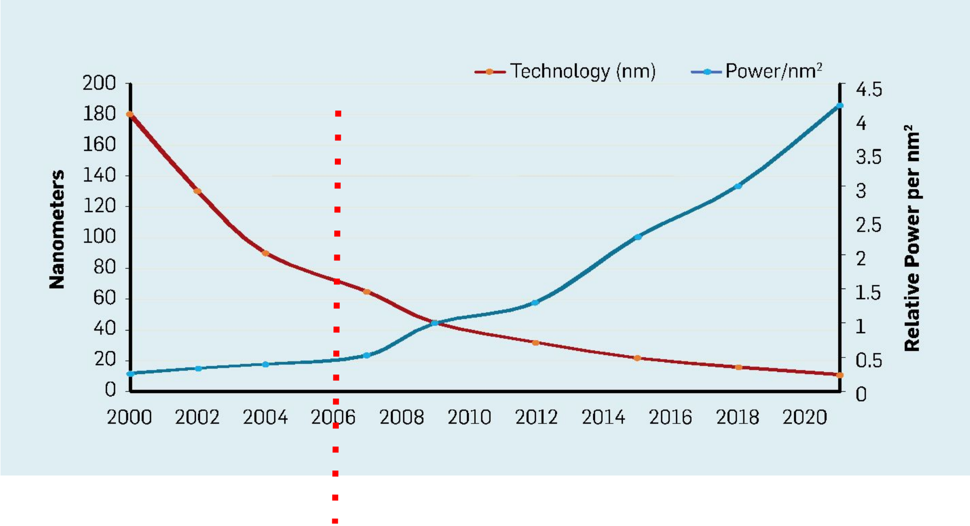
Outline

- 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

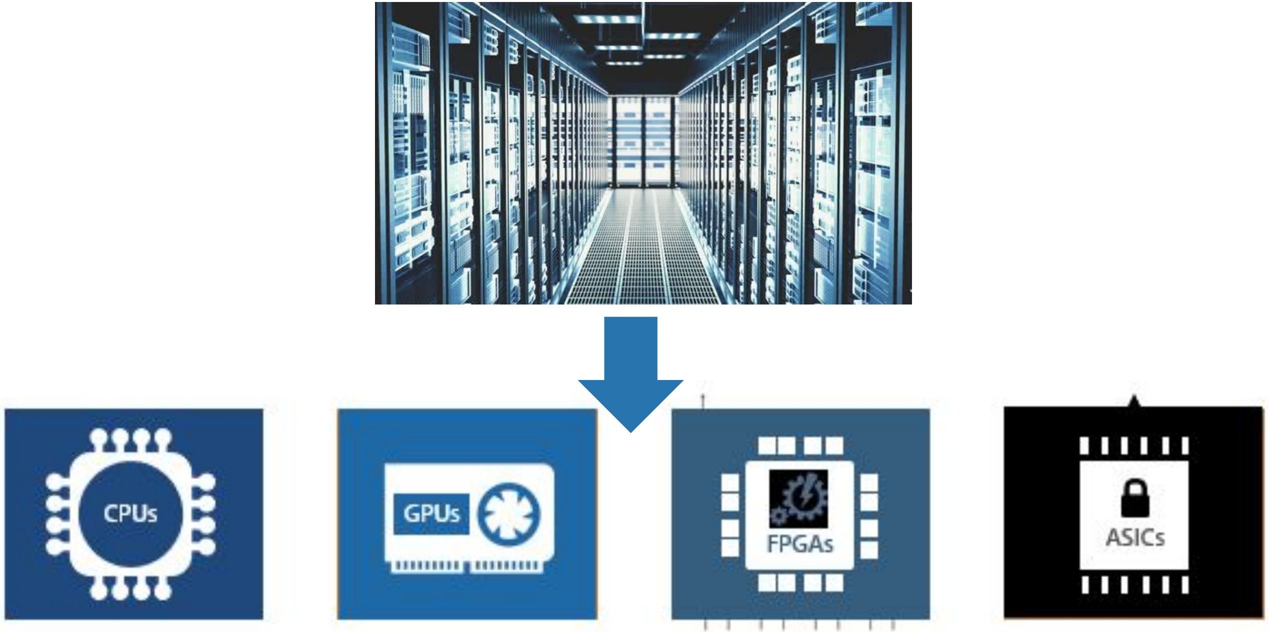




2018 2006

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

amazon web services™ EC2

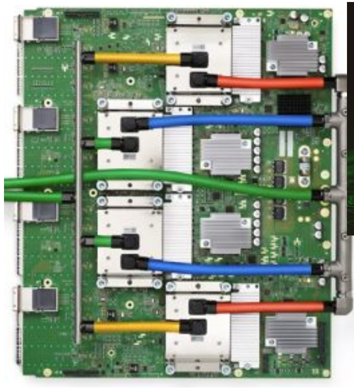
intel FPGA DEVCLOUD

Introducing oneAPI intel
No transistor left behind

Microsoft Azure

XILINX
ALL PROGRAMMABLE™

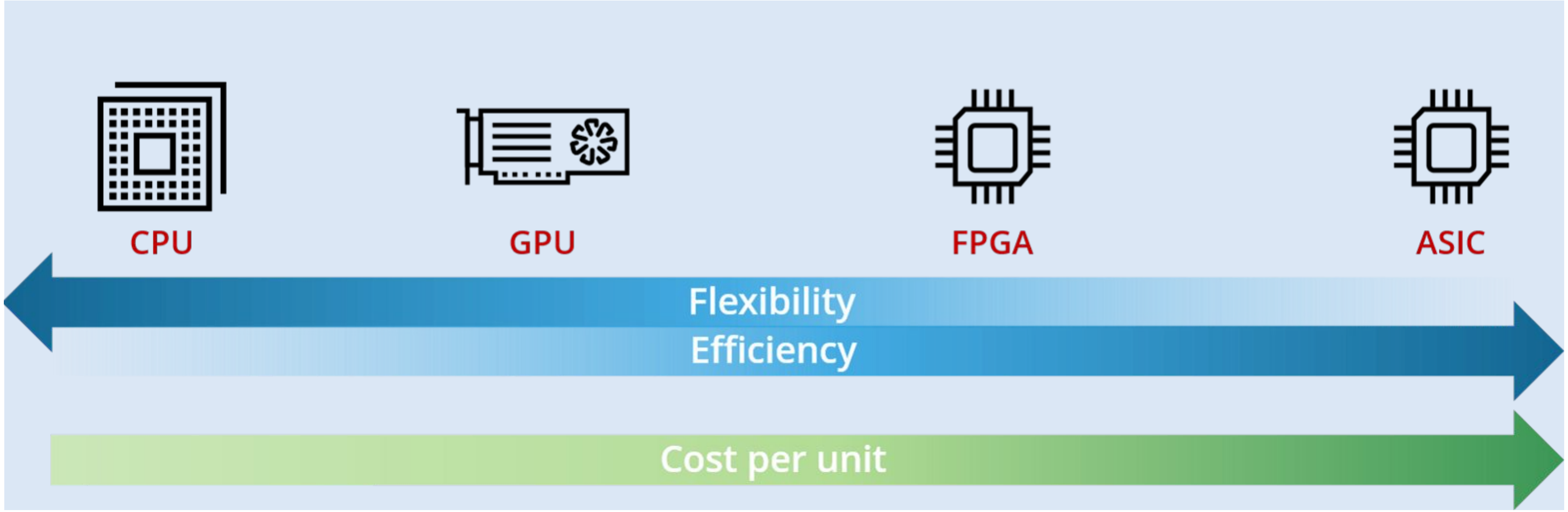
Cloud TPU



Nimbix Supercomputing

What does heterogeneity look like?

CPU, GPU, FPGA and ASICs tradeoffs

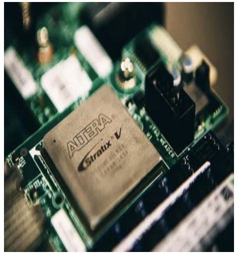


Field programmable gate array (FPGA)

Programmable logics, interconnects, and customizable building blocks

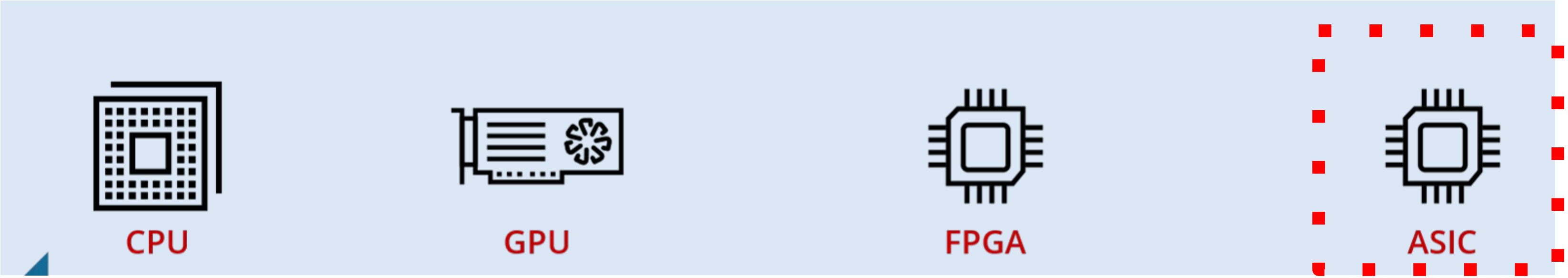
Microsoft

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



Difficult to programs in RTL languages

Application-specific integrated circuit (ASIC)

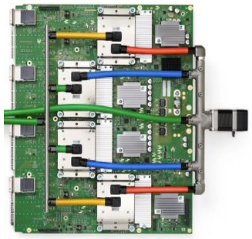


TPU for accelerating deep-learning workloads

80X performance–per-watt advantage over CPU

Design cycle is long and costly.

Google



What are the implications of heterogeneity?

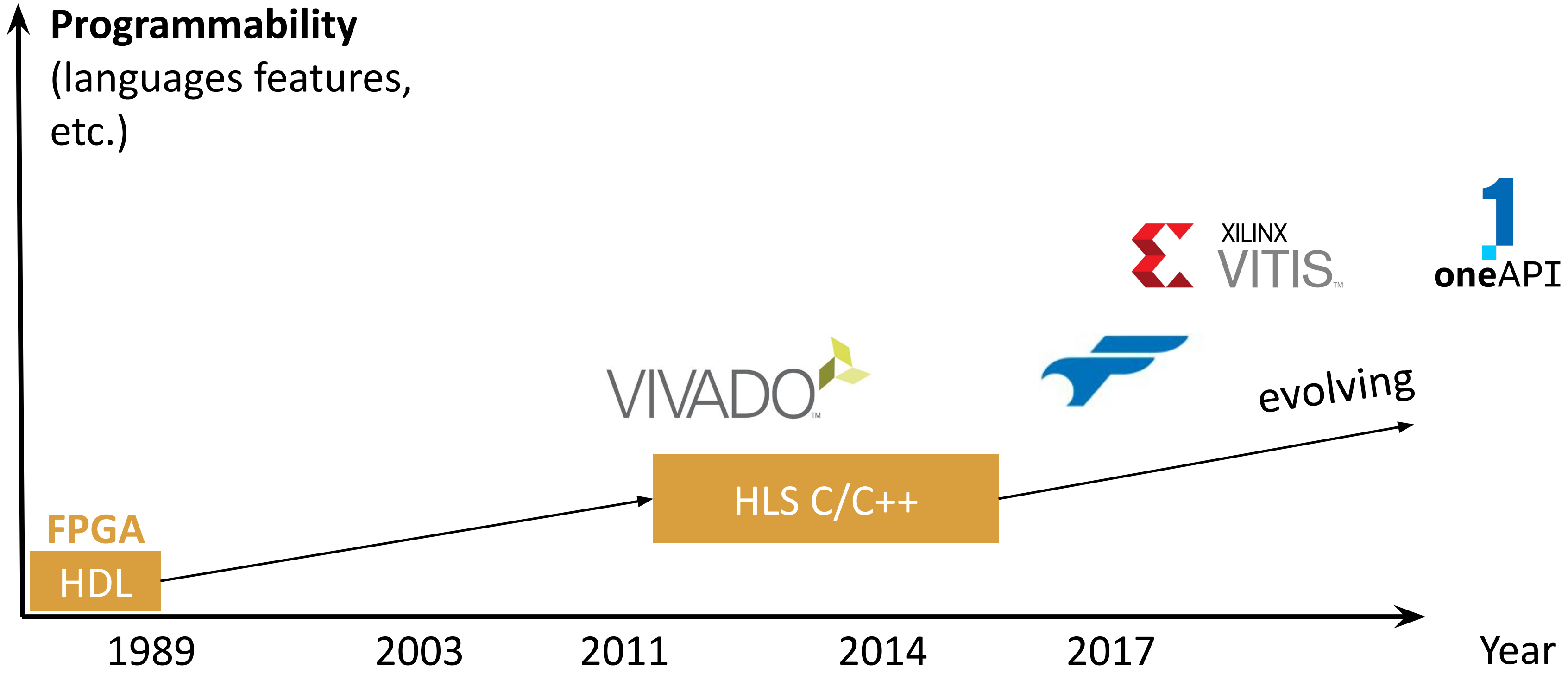
US bureau of labor statistics

1.8 M software developer

70000 hardware engineers

Towards Democratized IC Design and Customized Computing, 2022

Raising the abstraction level of HW design

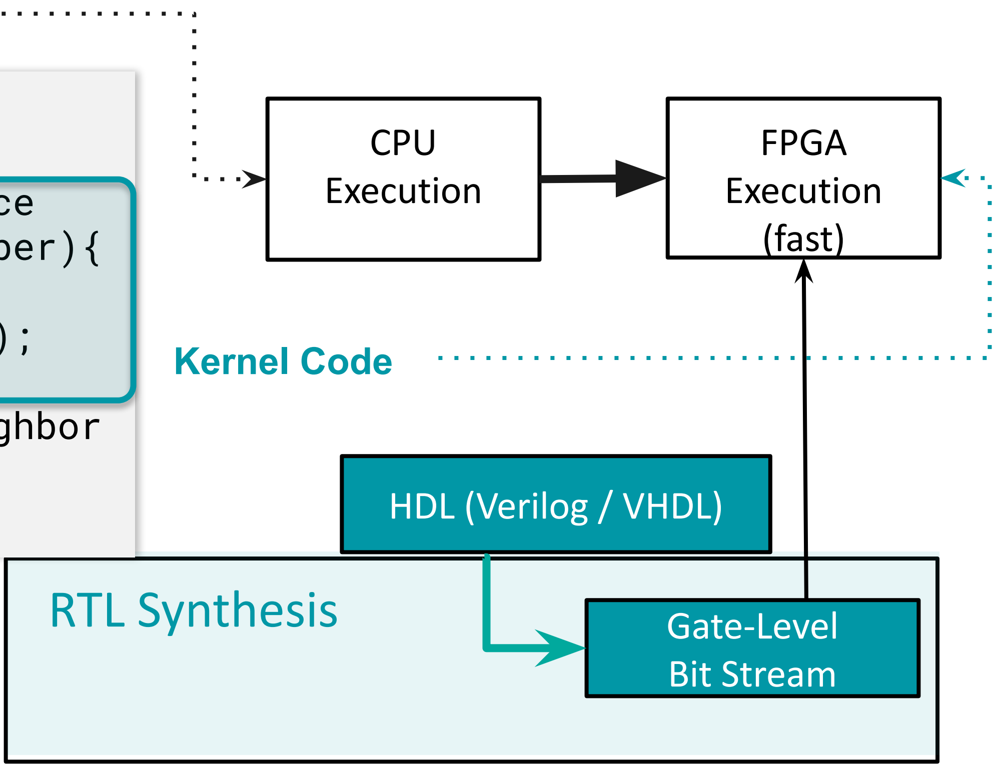


What is developer workflow with high level synthesis?

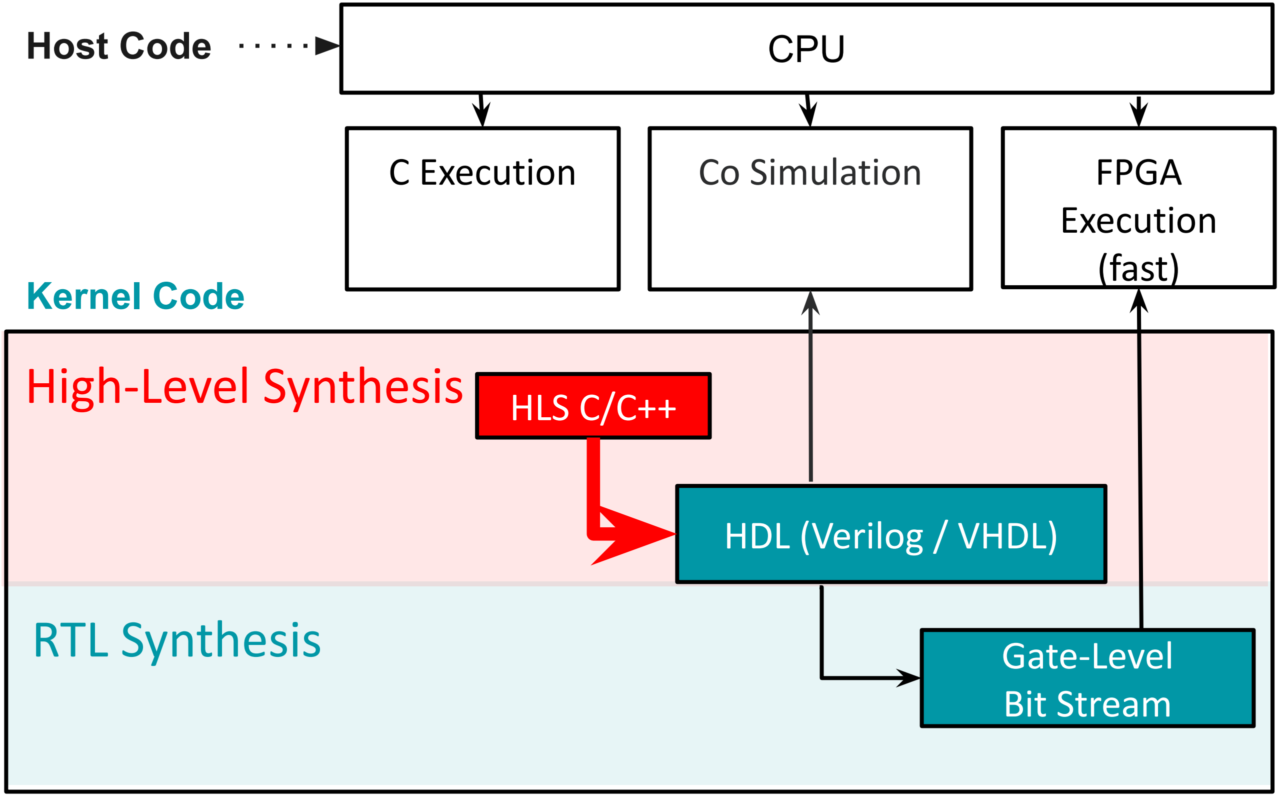
Traditional FPGA design flow

Host Code

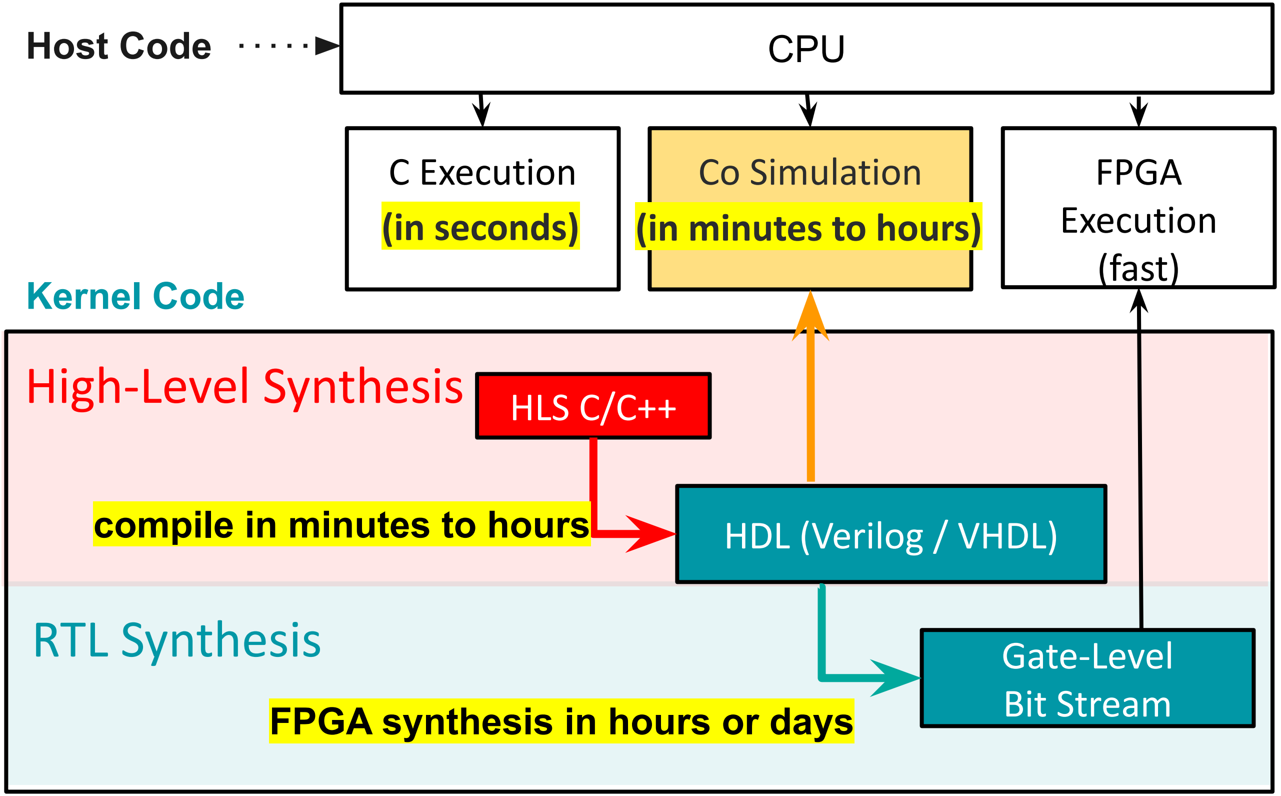
```
int KNN()
  ...
// Calculate distance
   for (i = 0 to number){
   dist[i] =
l2norm(data[i], dim);
   }
//Top 1 nearest neighbor
    ...
}
```



High level synthesis (HLS) for FPGA



High level synthesis (HLS) for FPGA



What is developer workflow with HLS?

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

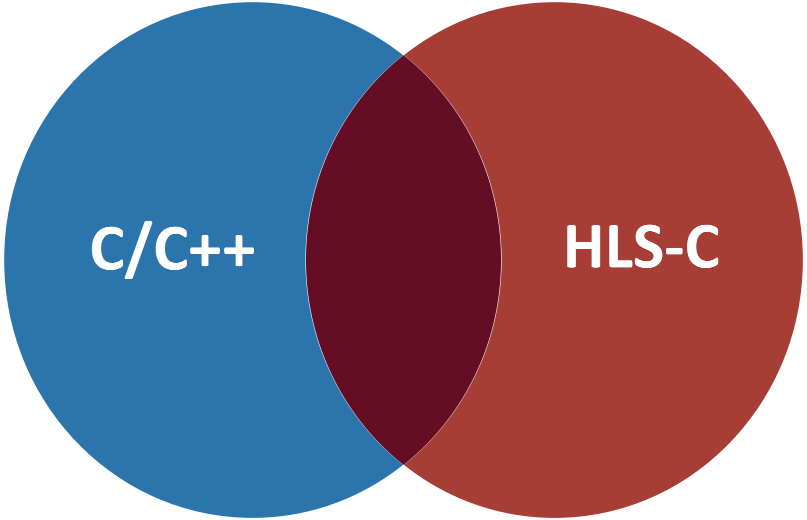
7X speed up on FPGA

1. Performance profiling
2. Kernel function identification in C
3. Manual **rewriting** from C to HLS-C — **HLS compilation to RTL 6 minutes**
4. **Differential testing** with input samples (RTL simulation vs. C execution) — **CPU-FPGA co-simulation 8 minutes**
5. Iterative optimization

Repeat

FPGA synthesis in 2.5 hours

HLS tools are not easy to use for SW developers



Manual rewriting for synthesizability and optimization

No developer tools for code translation

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

HLS-C requires specifying *bitwidth* for each type

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

C Program

```
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

HLS-C uses a *custom* floating point type

```
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;
}
```

C Program

```
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 Program

HLS-C requires *finitizing* 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);
}
```

HLS compile error

C Program

```
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()) {
       ...}}
```

HLS-C Program

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

UCLA Samueli School of Engineering

Divergence errors between CPU and FPGA

```
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;
  }
}
```

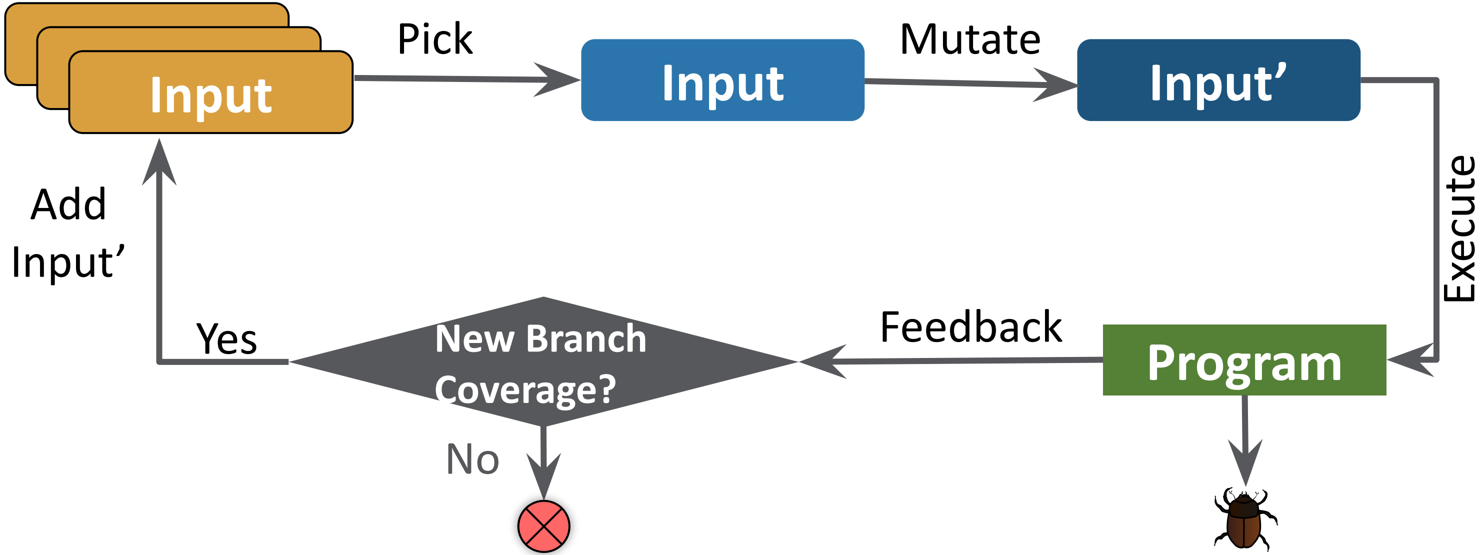
Host Code

```
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;
}
```

Kernel Code

Input	CPU	FPGA
[1,1,1,253]	no errors	div/0 in host
[2,1,1,253]	257	1

Is *fuzz testing* applicable?



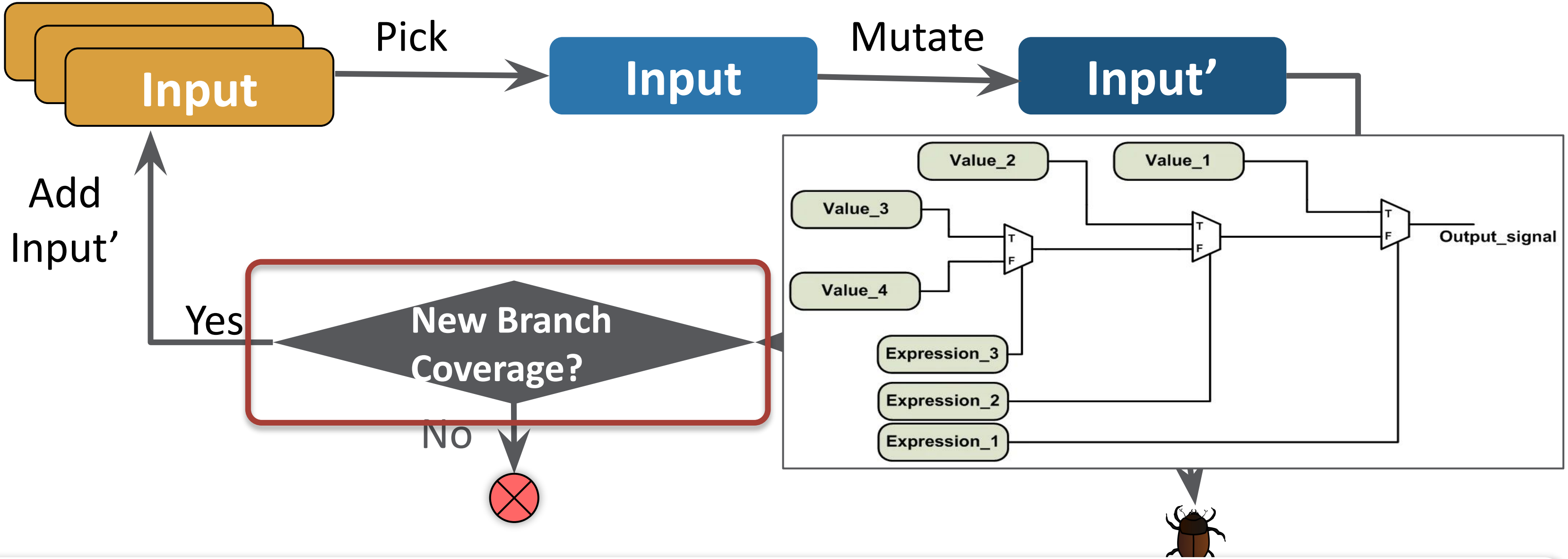
AFL running time for finding errors

Table 1: Examples of Behavior Divergence Between CPU and FPGA

ID	Description	Time
894069 [57]	Segmentation fault when allocating a big array `int x[1920][1080]` on FPGA yet no error on CPU	8.5h
595225 [58]	Different outcome caused by HLS `dataflow` directive	47.7h
438446 [59]	Different outcome caused by FPGA fetching incorrect `struct vector training_set[MAXSZ]`	10.6h
754676 [60]	Different outcome caused by bitwidth `typedef ap_fixed<25,1,AP_RND> s25f24_type`	2.3h
785019 [61]	Getting all zeros when shifting an array caused by `#pragma HLS RESET`	3.1h
907213 [62]	Undecided output when overwriting a same variable within the loop yet no error on CPU	79.4h
1166264 [63]	`EMFILE` error when loading 2048 files with `#pragma HLS ARRAY_PARTITION` yet no error on CPU	11.7h
1126600 [64]	The 25-tap FIR filter bypassess some input multiplications with `#pragma HLS PIPELINE`	93.2h

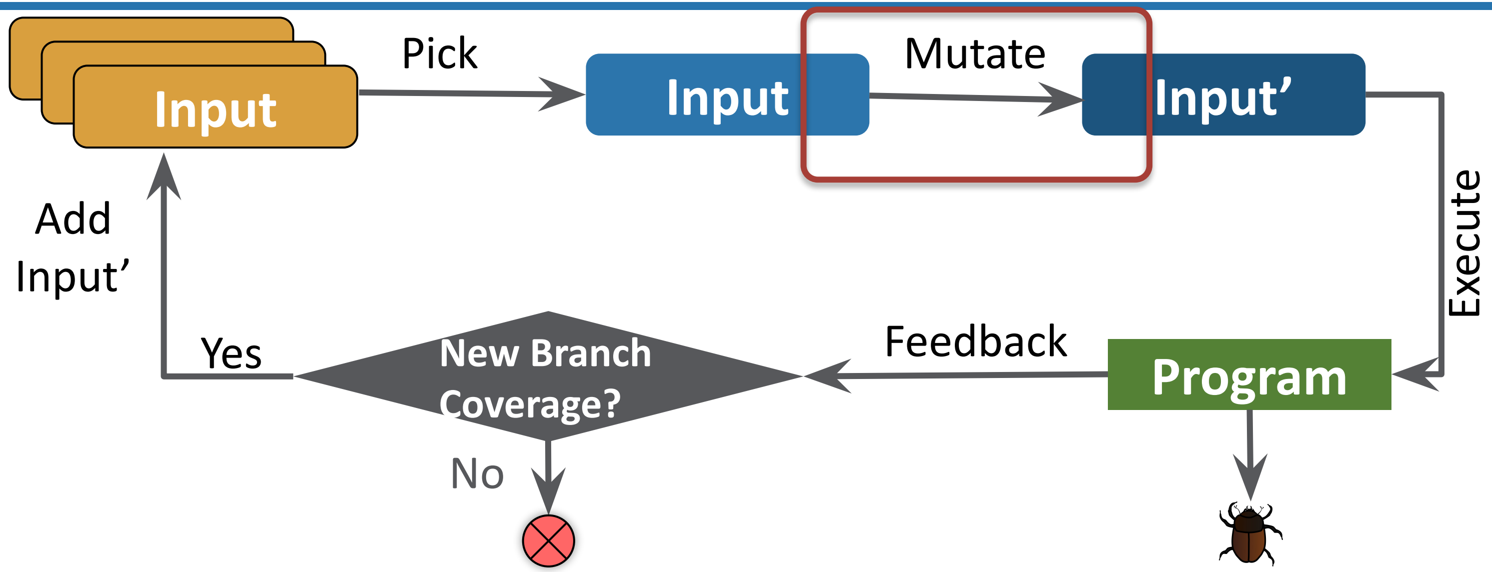
AFL: American Fuzzy Lop (a well known fuzz testing framework)

Challenge 1: lack of guidance in HW



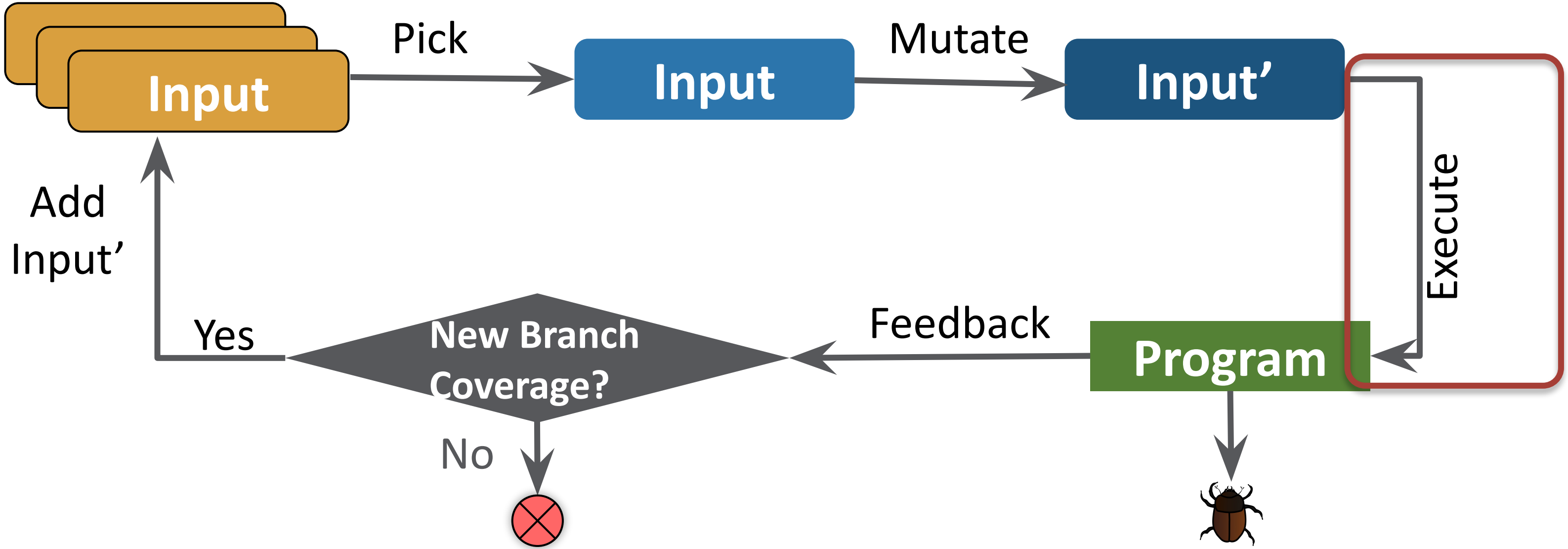
Branch coverage is not meaningful in HW

Challenge 2: lack of effective mutations



Input mutations must stretch HW behavior in terms of finitized resource usages to induce errors

Challenge 3: long simulation time



Fuzzing assumes the program under test can execute quickly in the order of milliseconds.

HeteroFuzz Overview

Challenge	Heterogeneous Applications
lack of guidance in HW →	Accelerator Spectra Monitoring
lack of effective mutations →	Probabilistic Mutation
long simulation time →	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;
}
```

Kernel Code

```
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;
  }
}
```

Host Code

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

Static analysis of HLS pragmas

Inject accelerator specific monitors

HeteroFuzz evaluation

57%

Effectiveness

divergence-inducing inputs with accelerator spectra monitoring

17.5X

Speed up

with dynamic probabilistic mutation

8.8X

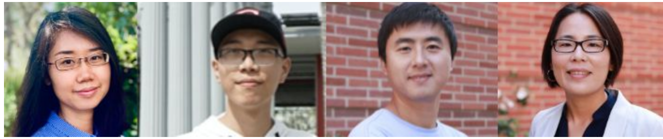
Efficiency

with selective HLS invocation

754X

Speed up

with three-pronged optimizations in finding divergence errors



HeteroGen: Transpiling C to Heterogeneous HLS Code with Automated Test Generation and Program Repair

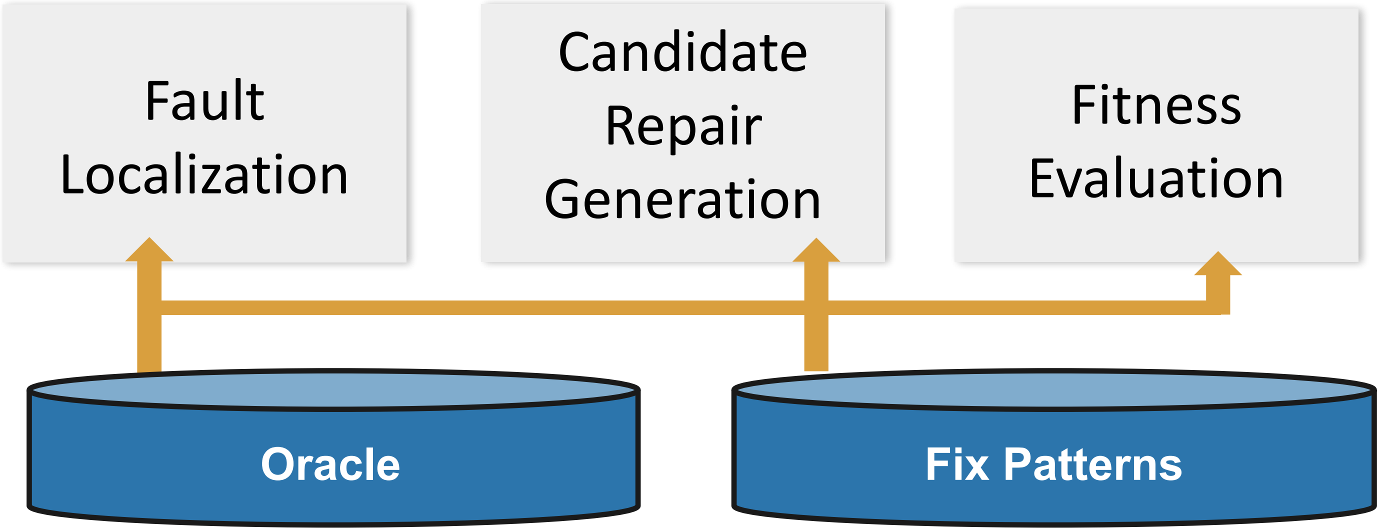
Qian Zhang, Jiyuan Wang, Harry Xu, Miryung Kim

ASPLOS 2022

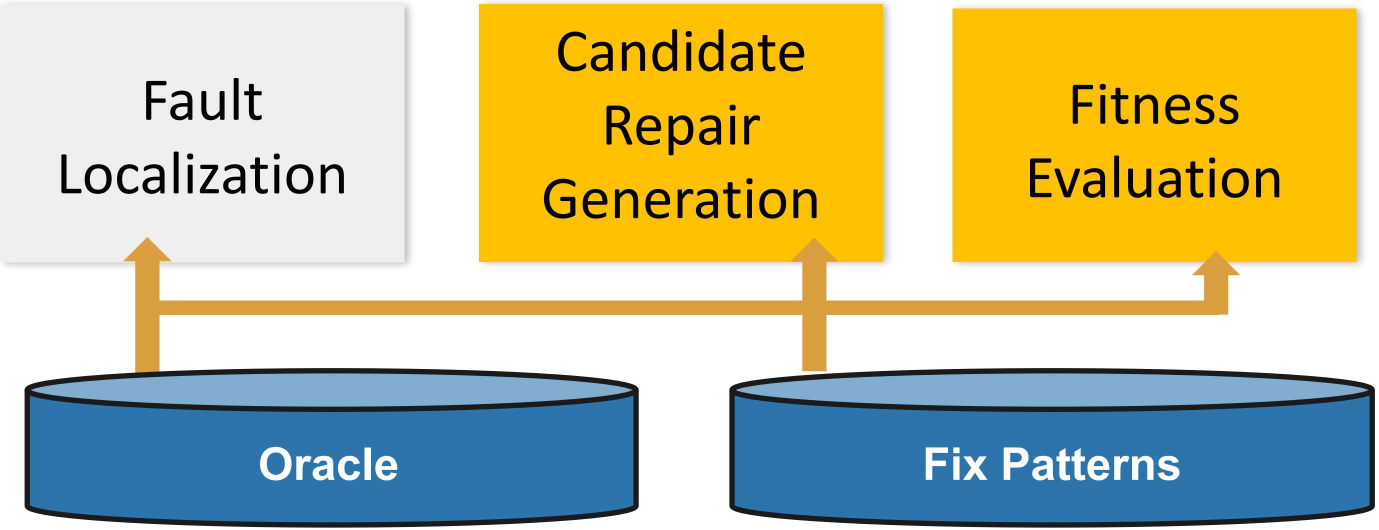
UCLA Samueli School of Engineering

Is *search-based repair* applicable?

Automated program repair (2008 ~)



(1) Long compilation & run (2) a large search space



0.2s in GenProg *vs.* 14 minutes in HLS

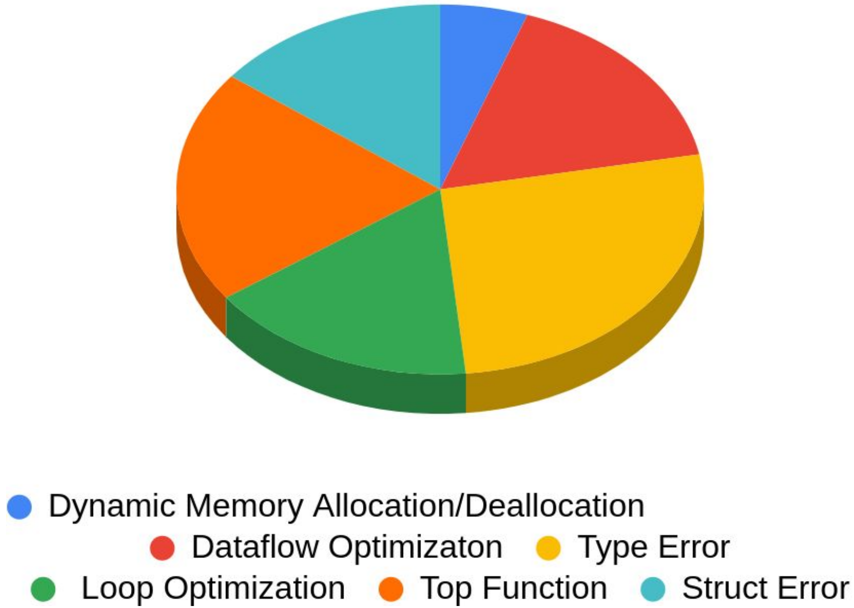
Genprog: a generic method for automatic software repair 2011

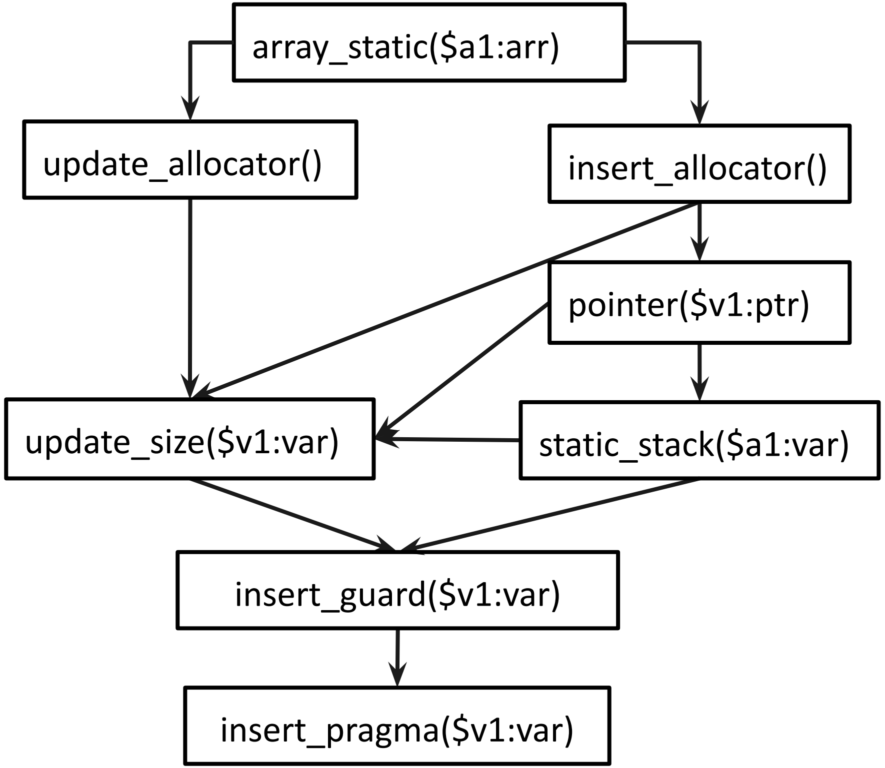
HeteroGen overview

Challenge	Heterogeneous Applications
The search-space of small edits is too large	Encode composite edits; Dependence-guided repair
HLS compilation and simulation takes minutes to hours	Dependence-guided repair; Early candidate rejection

1. Encode HLS repairs with composite edits

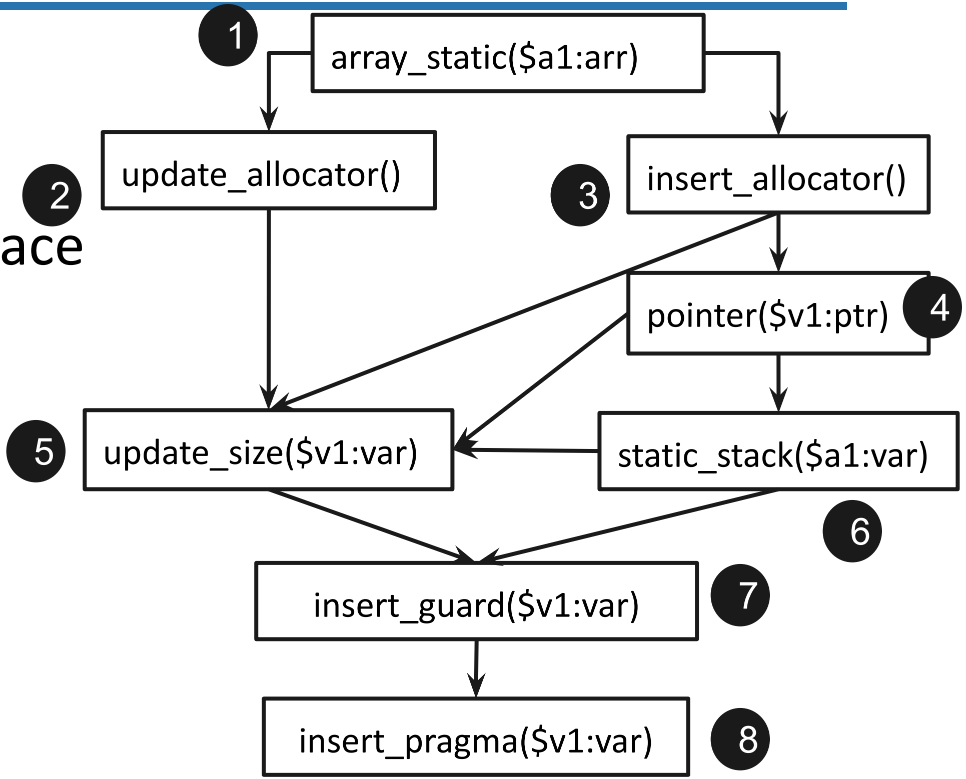
An error study based on Xilinx 1000 posts.





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

```
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

90%

Effectiveness

HeteroGen produces an HLS-compatible version for 9 out of 10.

97%

Coverage

Auto-generated ~2500 inputs cover 97%, while pre-existing tests reach 36% coverage.

35X

Speed-up

Dependence-based search contributes to 35X speedup than the one without.

~438 lines

Automation

It automates upto 438 lines.

1.6X

Latency

It produces a HLS version 1.63X faster than the original C

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

https://github.com/ucla-seal/

Developer Tools for Heterogeneous Computing	HeteroGen, HeteroFuzz, HeteroRefactor, QDiff
Debugging and Testing Tools for Big Data	BigDebug, BigSift, BigTest, BigFuzz, PerfDebug, FlowDebug, OptDebug, Titian
Testing, Debugging and Refactoring for Java + Code Mining GitHub	ExampleCheck, ExampleStack, Examplore, Jdebloat, Jshrink, Critics, Lase, Alice, etc.
Systems and Runtimes for Memory Disaggregation	Semeru, Dorylus, Mapo, etc.

Q&A