GPU-Accelerated High-Level Synthesis for Bitwidth Optimization of FPGA Datapaths

Nachiket Kapre, Ye Deheng
nachiket@ieee.org

Tool support —
XILINX
NVIDIA
Claim
Claim

- GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi “brute-force” evaluation
Claim

• GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi “brute-force” evaluation

• DUMB + clever approach
Claim

- GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi “brute-force” evaluation
- DUMB + clever approach

- **Idea:**
  (1) Use CPU heuristic to narrow search space,
  (2) deploy GPUs when single-shot feasible.
Why Bitwidth Optimisation?
Why Bitwidth Optimisation?

![Diagram](Image)
Why Bitwidth Optimisation?

[FCCM 2012] FX-SCORE — 70% less area
Why Bitwidth Optimisation?

[FCCM 2013] — 4x less area
Why Bitwidth Optimisation?

[FCCM 2014] Mix-FXSCORE—43% less area
But,... slow runtime!
But,… slow runtime!
But,… slow runtime!

- Typical bitwidth optimization challenge:
  - 1K-tap FIR filter design
  - simulate various conditions $\sim 1e^5$ test vector
  - 40 days of CPU runtime
But,… slow runtime!

- Typical bitwidth optimization challenge:
  - 1K-tap FIR filter design
  - simulate various conditions ~$10^5$ test vector
  - 40 days of CPU runtime

G. Caffarena and D. Menard. Many-core parallelization of fixed-point optimization of VLSI circuits through GPU devices. DASIP 2012
Outline

• Bitwidth Optimization Review
   — Parallelism outlook

• GPU Parallelisation
   — CPU-assisted pruning

• Quick-and-dirty HLS

• Experimental Validation

• Outlook
Bitwidth Optimization

Parallel Potential
Outline of algorithm

1. **Interval Analysis**
   - Identify range \([\text{min}, \text{max}]\) per variable
   - One-time pre-processing

2. **Iterative optimization**
   - a. Assign precisions to variables
   - b. Estimate error based on interval and precision
   - c. Optimize/Loop until error meets threshold
Interval Analysis
Interval Analysis

Diagram:
- Input Intervals
- DFG
- Estimate Error, Cost
- Bitwidths

Expression:
\[ [0, 1] \]
\[ a \rightarrow \times \rightarrow b \]
\[ + \rightarrow \times \rightarrow c \]
\[ + \rightarrow + \rightarrow y \]
Interval Analysis

Interval Analysis

Optimization

DFG

Input

Intervals

Estimate Error, Cost

Bitwidths

[0,1]

[0,a]

[0,b]

[0,a]

[0,a]

[0,1]

[0,1]
Interval Analysis

Optimization

DFG

Input

Intervals

Estimate Error, Cost

Bitwidths

Interval Analysis

[0,1]

[a] [0,1]

[0,a] [0,b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]

[0,a] [0,a+b]

[0,a] [0,b]
Interval Analysis

![Diagram showing interval analysis with nodes and intervals]

12
Interval Analysis (IA)
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds
Interval Analysis (IA)

• Propagate the intervals step-by-step through DFG

• IA Pessimistic — does not generate tight bounds

• Loose bounds — larger-than-reqd bitwidths
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds
- Loose bounds — larger-than-reqd bitwidths
- CPUs — affine arithmetic (AA), include correlations between inputs — explosion in error terms!
Interval Analysis (IA)

• Propagate the intervals step-by-step through DFG

• IA Pessimistic — does not generate tight bounds

• Loose bounds — larger-than-reqd bitwidths

• CPUs — *affine arithmetic (AA)*, include correlations between inputs — explosion in error terms!

• GPUs — *sub-interval analysis (Sub-IA)*
Sub-Interval Analysis

[0, 1/2]

[1/2, 1]

[Image of a diagram showing sub-intervals and operations involving variables a, x, b, c, and y.]
Sub-Interval Analysis

[0,1/4]

[1/4,1/2]

[1/2,3/4]

[3/4,1]
Sub-Interval Analysis

(1) Tighter bounds than plain IA
(2) Exponential threads with splits
(3) Completely parallel!
Error+Cost models
Error+Cost models

- Iterative optimization — searches through multiple candidates, refines/improves solution
Error+Cost models

- Iterative optimization — searches through multiple candidates, refines/improves solution

- At each evaluation, must calculate
  — choose bitwidth assignments/variable
  — relative/absolute error at each variable/output
  — physical cost of mapping to FPGA
Idea

Space of Possible Solutions
Idea

Space of Possible Solutions
Idea

Space of Possible Solutions

Pruned set
Idea

Space of Possible Solutions

Pruned set
Error Propagation
Error Propagation

Tight Intervals
Error Propagation

Tight Intervals

Bitwidth Assignment
Error Propagation

Tight Intervals

Bitwidth Assignment

Error Estimates

\[ [0,1] \]
\[ [0,0] \]
\[ [0, a] \]
\[ [0, a+b] \]
\[ [0, a+b+c] \]

\[ + \]

\[ W_1 \]

\[ W_2 \]

\[ W_3 \]

\[ W_4 \]

\[ W_5 \]

\[ a \]

\[ b \]

\[ x \]

\[ c \]

\[ y \]

\[ 2^{-wx} \]

\[ 2^{-w1}, a \cdot 2^{-wx} \]

\[ 2^{-w2}, b \cdot 2^{-wx} \]

\[ 2^{-w3}, a \cdot 2^{-wx} \]

\[ 2^{-w4}, (a+b)/2 \cdot 2^{-wx} \]

\[ 2^{-w5}, (a+b)/2 \cdot 2^{-wx} \]

\[ + \]
Resource Evaluation
Resource Evaluation

RTL for operators

Resource Table/Eqn.
Resource Evaluation

RTL for operators

Resource Table/Eqn.

Bitwidth Assignment
Resource Evaluation

RTL for operators

Resource Table/Eqn.

$A(w_1) = W_x$

Bitwidth Assignment

$A(w_2)$

$A(w_3)$

$A(w_4)$

$A(w_5)$

Resource Prediction
Parallel Explore
Parallel Explore

(1) Evaluate multiple combinations of bitwidths
(2) Common dataflow propagate/evaluate pattern
GPU Architecture
Parallel Potential
Modern GPU Potential
Modern GPU Potential
Modern GPU Potential

- Can run thousands of threads in parallel
  - Each thread do lightweight tasks
  - Suitable for high-throughput computing
Modern GPU Potential

• Can run thousands of threads in parallel
  — Each thread do lightweight tasks
  — Suitable for high-throughput computing

• NVIDIA Tesla K20
  — 26K threads
  — 5G RAM
Modern GPU Potential

• Can run thousands of threads in parallel
  — Each thread do lightweight tasks
  — Suitable for high-throughput computing

• NVIDIA Tesla K20
  — 26K threads
  — 5G RAM

• Common dataflow evaluation pattern
GPU Implementation
GPU Implementation

dataflow graph

input intervals

range_kernel
compute bounds
of all variables
using sub-IA

split sub-intervals
output-intervals

generate bit combinations

error_kernel
calc. relative
error of variables
as fn. of bitwidth

bitwidth combinations
error estimates

area_kernel
compute resource
costs (LUTs, DSPs)
based on analytical models

bitwidth combinations

resource cost

error < ε && min(cost)

final bitwidths for variables
GPU
Implementation

- Three core GPU-accelerated kernels
  - sub-interval analysis
  - error propagation
  - resource evaluation
GPU Implementation

- Three core GPU-accelerated kernels
  - sub-interval analysis
  - error propagation
  - resource evaluation

- Common operation: graph traversal
  - updating state per variable/node
GPU Implementation

- Three core GPU-accelerated kernels
  - sub-interval analysis
  - error propagation
  - resource evaluation

- Common operation: graph traversal
  - updating state per variable/node

- Parallelize to match GPU potential, and application requirement..
Example CUDA code

• Bulk of the work performed by leaf-level CUDA functions

• Example for fixed-point addition

• Range: inputs \([x_0,y_0]\) and \([x_1,y_1]\), compute output \([\text{high, low}]\)

• Error: errors e1, and e2 (inputs) and e3 (depends on bitwidth of add)

• Area: approximate larger of two input precisions to lookup LUTs, DSPs, BRAMs

```c
__device__ void add_range(...)
{
    double t1 = x1+y1;
    double t2 = x0+y0;
    *high=max(t1,t2);
    *low=min(t1,t2);
}

__device__ void add_error(...)
{
    double max = (e1>=e2)?e1:e2;
    if (e3 > max)
        *error = e3 + e1 + e2;
    else
        *error = e2 + e1;
}

__device__ void add_area(...)
{
    *area = (x1>x2)?x1:x2;
}
```
# K20 GPU vs i5-4570 Ideal Performance

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add</td>
</tr>
<tr>
<td>Range analysis</td>
<td>312×</td>
</tr>
<tr>
<td>Error propagation</td>
<td>246×</td>
</tr>
<tr>
<td>Resource estimation</td>
<td>272×</td>
</tr>
</tbody>
</table>
Limits of dump parallelism
Limits of dump parallelism

5—6G RAM
Limits of dump parallelism

1. Cannot enumerate any reasonable solution space, even with 100x speedups — 16 variables, 8 choices/variable — $8^{16}$ (200 peta combinations)
Limits of dump parallelism

1. Cannot enumerate any reasonable solution space, even with 100x speedups
   — 16 variables, 8 choices/var — $8^{16}$
   (200 peta combinations)

2. Limited DRAM capacity on GPU card
   — cannot “store” all solutions!
   — 16 variables, 32b range/var/combination, 4 values/variable
   (2—3M combinations feasible)
Pruning of Search Space

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables \( N \); Targeted Fixed-point Precision

Result: Bounded search space

1. \( \text{bit\_width}(0:N-1) \leftarrow \text{target\_fb} \);
2. \( \textbf{while} \ \text{current\_error} > \text{error\_constraint} \ \textbf{do} \)
3. \( \quad \text{bit\_width}(0:N-1) \ ++; \)
4. \( \textbf{end} \)
5. \( \text{uniform\_bit} = \text{bit\_width}[0]; \)
6. \( \textbf{foreach} \ n=0:N-1 \ \textbf{do} \)
7. \( \quad \textbf{while} \ \text{current\_error} \leq \text{error\_constraint} \ \textbf{do} \)
8. \( \quad \quad \text{bit\_width}(n) \ --; \)
9. \( \quad \textbf{end} \)
10. \( \quad \text{lowest}(n) \leftarrow \text{bit\_width}(n); \)
11. \( \quad \text{bit\_width}(n) \leftarrow \text{uniform\_bit}; \)
12. \( \textbf{end} \)
13. \( \text{bit\_width}(0:N-1) \leftarrow \text{lowest}(0:N-1); \)
14. \( \textbf{while} \ \text{current\_error} \leq \text{error\_constraint} \ \textbf{do} \)
15. \( \quad \text{bit\_width}(0:N-1) ++; \)
16. \( \textbf{end} \)
17. \( \text{highest}(0:N-1) \leftarrow \text{bit\_width}(0:N-1) \ + \ \text{guard\_bit}; \)
Pruning of Search Space

- Start with safe common precision values for all variables

---

**Algorithm 1: Search Space Pruning Heuristic**

**Data:** The number of variables N; Targeted Fixed-point Precision

**Result:** Bounded search space

1. `bit_width(0:N-1) ← target_fb;`
2. **while** `current_error > error_constraint` **do**
3. | `bit_width(0:N-1) ++;`
4. **end**
5. `uniform_bit = bit_width[0];`
6. **foreach** `n=0:N-1` **do**
7. | **while** `current_error ≤ error_constraint` **do**
8. | | `bit_width(n)--;`
9. | **end**
10. | `lowest(n) ← bit_width(n);`
11. | `bit_width(n) ← uniform_bit;`
12. **end**
13. `bit_width(0:N-1) ← lowest(0:N-1);`
14. **while** `current_error ≤ error_constraint` **do**
15. | `bit_width(0:N-1) ++;`
16. **end**
17. `highest(0:N-1) ← bit_width(0:N-1) + guard_bit;`
Pruning of Search Space

- Start with safe common precision values for all variables
- Reduce each variable precisely separately until failure — error > threshold

```
Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision
Result: Bounded search space

bit_width(0:N-1) \leftarrow target_fb;

while current_error > error_constraint do
  bit_width(0:N-1) ++;
end

uniform_bit = bit_width[0];

foreach n = 0:N-1 do
  while current_error \leq error_constraint do
    bit_width(n) --;
  end
  lowest(n) \leftarrow bit_width(n);
  bit_width(n) \leftarrow uniform_bit;
end

bit_width(0:N-1) \leftarrow lowest(0:N-1);
while current_error \leq error_constraint do
  bit_width(0:N-1) ++;
end
highest(0:N-1) \leftarrow bit_width(0:N-1) + guard_bit;
```
Pruning of Search Space

• Start with safe common precision values for all variables

• Reduce each variable precise separately until failure — error>threshold

• Use these as lower limits per variable

```
Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision

Result: Bounded search space

1  bit_width(0:N-1) ← target_fb;
2  while current_error > error_constraint do
3      bit_width(0:N-1) ++;
4  end
5  uniform_bit = bit_width[0];
6  foreach n=0:N-1 do
7      while current_error ≤ error_constraint do
8          bit_width(n) --;
9      end
10     lowest(n) ← bit_width(n);
11     bit_width(n) ← uniform_bit;
12  end
13  bit_width(0:N-1) ← lowest(0:N-1);
14  while current_error ≤ error_constraint do
15      bit_width(0:N-1) ++;
16  end
17  highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```
Pruning of Search Space

- Start with safe common precision values for all variables
- Reduce each variable precise separately until failure — error > threshold
- Use these as lower limits per variable
- Recalculate upper limits for search — recognizing GPU bounds

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision

Result: Bounded search space

1. $bit\_width(0:N-1) \leftarrow target\_fb$
2. while $current\_error > error\_constraint$ do
   3. $bit\_width(0:N-1) \leftarrow +$
4. end
5. $uniform\_bit = bit\_width[0]$
6. foreach $n=0:N-1$ do
   7. while $current\_error \leq error\_constraint$ do
      8. $bit\_width(n) \leftarrow -$
   9. end
10. $lowest(n) \leftarrow bit\_width(n)$
11. $bit\_width(n) \leftarrow uniform\_bit$
12. end
13. $bit\_width(0:N-1) \leftarrow lowest(0:N-1)$
14. while $current\_error \leq error\_constraint$ do
15. $bit\_width(0:N-1) \leftarrow +$
16. end
17. $highest(0:N-1) \leftarrow bit\_width(0:N-1) + guard\_bit$
Simulated Annealing

Space of Possible Solutions
Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber’s Caltech page
  — great email support (to help us get started)
Simulated Annealing

• ASA — Adaptive Simulated Annealing
  — Lester Ingber’s Caltech page
  — great email support (to help us get started)

• This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:
Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber’s Caltech page
  — great email support (to help us get started)

- This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:

- (1) Modified asa_usr_cst.c to support FPGA resource models that we supplied ourselves
Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber’s Caltech page
  — great email support (to help us get started)

- This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:
  
  1. Modified asa_usr_cst.c to support FPGA resource models that we supplied ourselves
  
  2. Links to Gappa for analysing relative error
Quick HLS Flow

Demonstrate ideas
HLS Flow
HLS Flow

• Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development
HLS Flow

• Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development

• Develop a lightweight compilation flow
  — prove your idea/transformation
  — distribute, integrate with LLVM with community support
HLS Flow

• Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development

• Develop a lightweight compilation flow
  — prove your idea/transformation
  — distribute, integrate with LLVM with community support

• Not trying to build production-ready compiler
  — just interested in research/proof of concept
Quick HLS Toolflow

- **DFG**
- **Input Intervals**

**Interval Analysis**

- **Optimization**
- **Estimate Error, Cost**

**Bitwidths**
Quick HLS Toolflow

gappa

gcc

ASA

CUDA

vivado
Quick HLS Toolflow

C + input intervals

gcc GIMPLE

DFG scripts

ASA

CUDA

vivado
Quick HLS Toolflow

C + input intervals

gcc GIMPLE

Gappa scripts

range

error

DFG scripts

ASA

CUDA

vivado
Quick HLS Toolflow

- C + input intervals
- gcc GIMPLE
- gappa scripts
- range
- error
- DFG scripts
- Pruning of Search Space
- Bitwidth bounds
- asa_usr.c

CUDA

vivado
Quick HLS Toolflow

- C + input intervals
- gcc GIMPLE
- DFG scripts
- Gappa scripts
- range
- error

Pruning of Search Space
Bitwidth bounds
FPGA area model
LUTs/DSPs Table

asa_usr.c
Quick HLS Toolflow

\[ C + \text{input intervals} \]

\[ \text{gcc GIMPLE} \]

\[ \text{Gappa scripts} \]

\[ \text{range} \]

\[ \text{error} \]

\[ \text{DFG scripts} \]

\[ \text{asa_usr.c} \]

\[ \text{Pruning of Search Space} \]

\[ \text{Bitwidth bounds} \]

\[ \text{Semi Brute-Force Sweep} \]

\[ \text{FPGA area model} \]

\[ \text{LUTs/DSPs Table} \]
Quick HLS Toolflow

- C + input intervals
- gcc GIMPLE
- DFG scripts
- Pruning of Search Space
- Bitwidth bounds
- Semi Brute-Force Sweep
- FPGA area model
- LUTs/DSPs Table

Gappa scripts

range

error
Experimental Validation
Outline of Results

• Speedup for range analysis
• Speedup for bitwidth optimization
• Quality-Time trends
Sub-interval arithmetic
Sub-interval arithmetic

vs. Gappa dichotomy search
Sub-interval arithmetic
Sub-interval arithmetic slowdown for DFGs with more IOs
Sub-interval arithmetic

1-4x tighter bounds!!
Bitwidth Optimisation

![Bar chart showing speedup for different benchmarks and configurations.](image-url)
Bitwidth Optimisation

brute force!
Bitwidth Optimisation

![Graph showing bitwidth optimisation results with benchmark names like level1_linear, poly, diode, bellido, approx1, poly6, level1_satur, poly8, approx2, caprasse, with speedup on the y-axis and benchmark on the x-axis. The graph highlights that brute force methods are infeasible due to high speedup.]
Bitwidth Optimisation

brute force slow
Quality-Time Tradeoffs
Quality-Time Tradeoffs

The graph illustrates the tradeoff between normalized cost and normalized time for different types of approximations. The long tail indicates that as the normalized time increases, the normalized cost decreases, highlighting the tradeoff between time and quality.
Quality-Time Tradeoffs

one-shot answers
Conclusions and Projections
Roundup

• GPUs can help accelerate FPGA CAD (bitwidth optimization)
  — 100x+ for sub-intervals
  — 10—100x for bitwidth allocation

• **PRUNE+BRUTE** philosophy
  — be prepared to do more work
  — GPUs not just about speed —> optimality

Future Work

• Do more work per GPU thread, only save best, local merge operations — better use of GPU threads

• Affine analysis formulations for GPU parallelism — potentially improve accuracy, converge faster with fewer sub-interval splits

• Modified/parallel Monte-Carlo approaches for covering search space
  — no need to cover every single instance

• Think about prune+brute strategy for other CAD problems
Varying Error Threshold
Fidelity of FPGA models