Improving High-Level Synthesis with Decoupled Data Structure Optimization

Ritchie Zhao, Gai Liu, Shreesha Srinath,
Christopher Batten, Zhiru Zhang

Computer Systems Lab
Electrical and Computer Engineering
Cornell University

Cornell University

Vision for Future HLS

Current HLS tools

Limited usage by hardware designers in a few specialized domains

Many challenges still remain

Our contribution:
Extend HLS to efficiently handle new programs

Future HLS tools

Wide usage by mainstream programmers who are not hardware experts

Traditional HLS Programs

- **program = algorithm + data structure**
  - Interface = data structure methods

Traditional HLS Programs

Algorithms

+

Simple Data Structures
- e.g., arrays, queues
- Fixed latency methods

CRC Error Detection

```
unsigned crc( msg[32], len ) {
  R = 0;
  for (i = 0; I < len; ++i) {
    R ^= msg[i] << (3*8);
    for (bit = 8; bit > 0; --bit) {
      if (R & (1 << 31))
        R ^= 0xD8;
      R = R << 1;
    }
  }
  return R;
}
```

HLS with Complex Data Structures

- **program = algorithm + data structure**
  - Interface = data structure methods

<u>Emerging HLS Programs</u>

Algorithms

+

Complex Data Structures

- e.g., trees, heaps
- Significant compute
- Variable latency methods

```
void priority_queue::push( val ) {
  data[size] = val;
  unsigned curr = size;
  ++size;

  while (curr != 0) {
    prev = (curr-1) >> 1;
    if (data[curr] > data[prev])
      swap(&data[curr], &data[prev]);
    else
      break;
    curr = prev;
  }
}
```

HLS with Complex Data Structures

- Many programs base their efficiency on complex data structures, which are poorly handled by existing HLS tools
- **Definition:** A data structure is complex if its key methods exhibit variable latency

Example complex data structures from the C++ STL containers library

Container	Underlying Data Structure	Key Methods	Variable Latency Operations
map, set	Red-black tree	insert, delete	Tree traversal and rotations
unordered_map, unordered_set	Hash table	insert, delete	Collision chain traversal
priority_queue	Heap	push, pop	Maintaining heap condition

Complex Method Example

Dijkstra's Algorithm

```
s = u.begin_neighbors();
e = u.end_neighbors();
// inner loop
for (v = s; v < e; ++v) {
  alt = dist[u] + edge[u][v];
  if (dist[v] > alt) {
    dist[v] = alt;
    // priority queue push
    Q.push(v, dist[v]);
  }
}
```

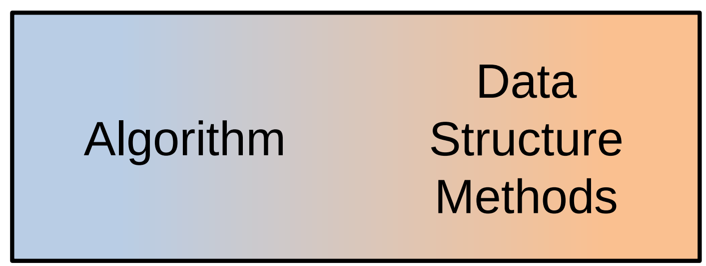
Algorithm

Complex Method

Conventional HLS Flow

1. **Static schedule** for the entire program
2. Monolithic hardware executes **in lockstep** adhering to the schedule

Generated Hardware



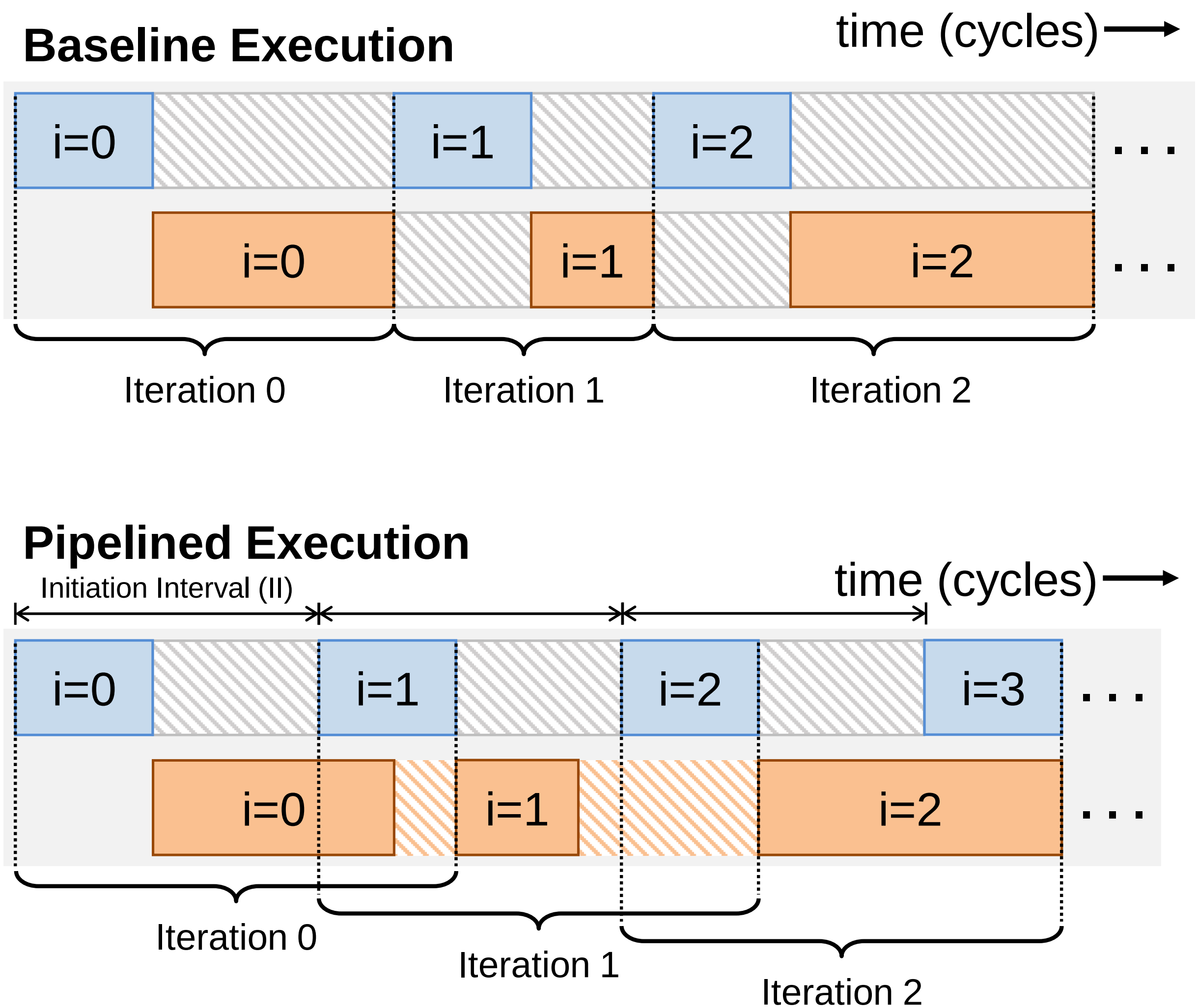
Complex Method Example

Dijkstra's Algorithm

```
s = u.begin_neighbors();
e = u.end_neighbors();
// inner loop
for (v = s; v < e; ++v) {
  alt = dist[u] + edge[u][v];
  if (dist[v] > alt) {
    dist[v] = alt;
    // priority queue push
    Q.push(v, dist[v]);
  }
}
```

Algorithm

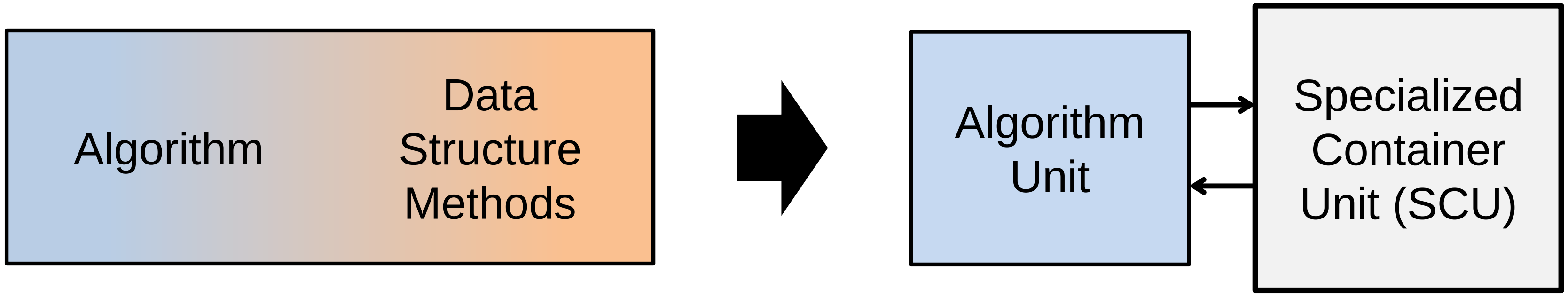
Complex Method



Decoupled Data Structure Synthesis

1. **Decouple** complex methods from the algorithm using a latency-insensitive interface
   – Separation of concerns, eliminate lockstep execution

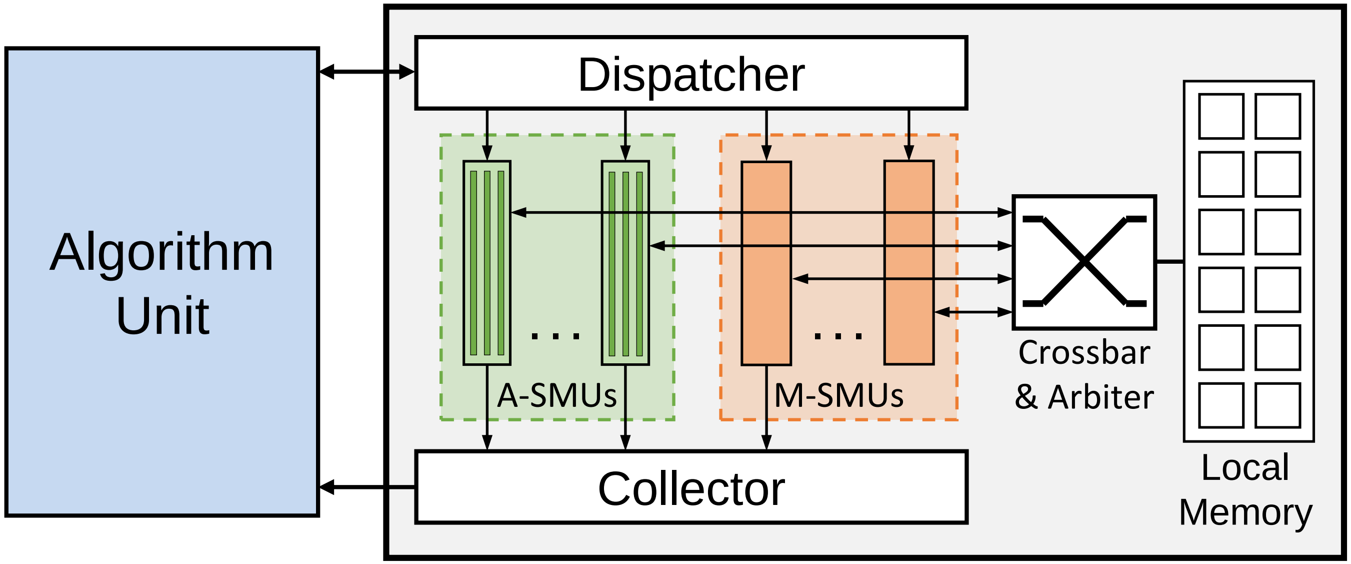
2. **Map** the complex data structure to a **specialized container unit (SCU)**
   – Potential for parallel and out-of-order method execution



Previous Work

- Individual data structure accelerators [Xu et al., *CISP'08*] [Huang et al., *FPL'14*] [Oberg et al., *FPL'12*]
  - Complementary to our approach
- Memory operation decoupling [Cheng & Wawrzynek, *FPT'14*]
  - We decouple entire methods, which may contain many loads and/or stores

Specialized Container Unit



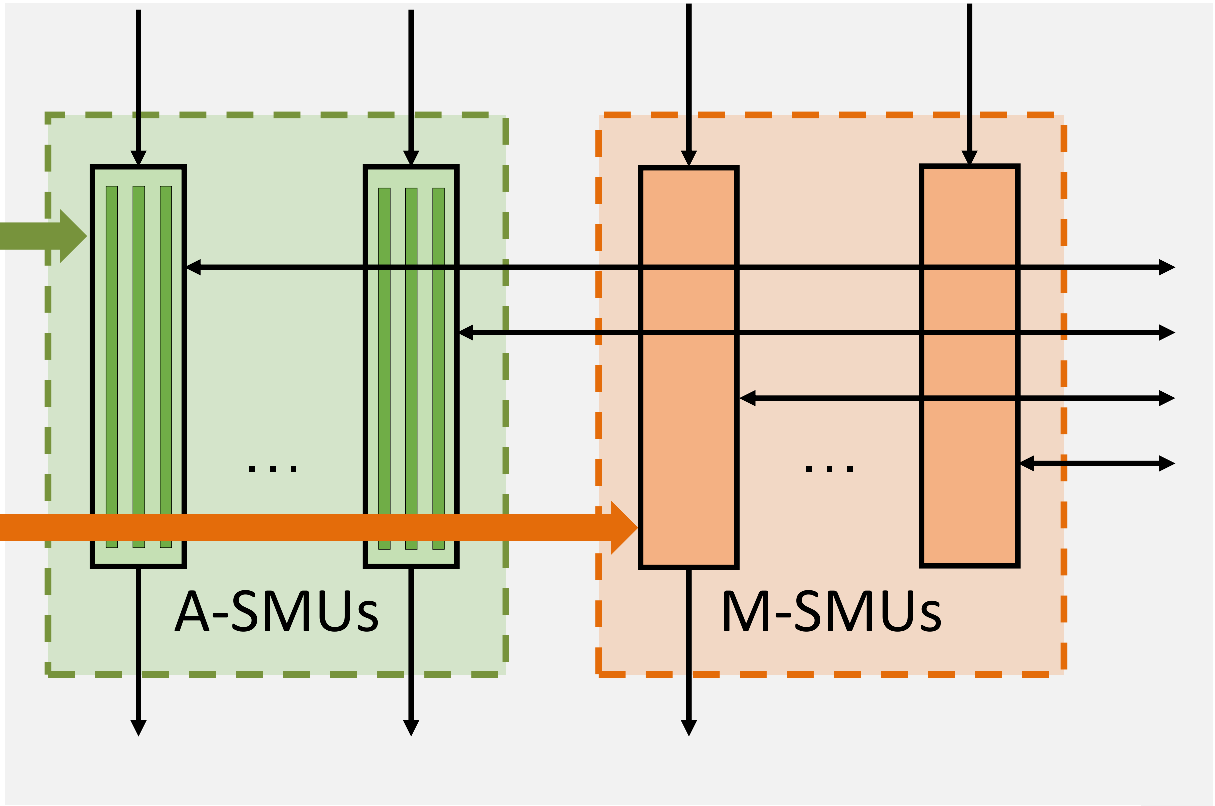
Specialized Container Unit (SCU)

- Architectural template
- Arrows indicate latency-insensitive interfaces (e.g., val-ready)
- Complex method calls → request/response messages

Specialized Method Units

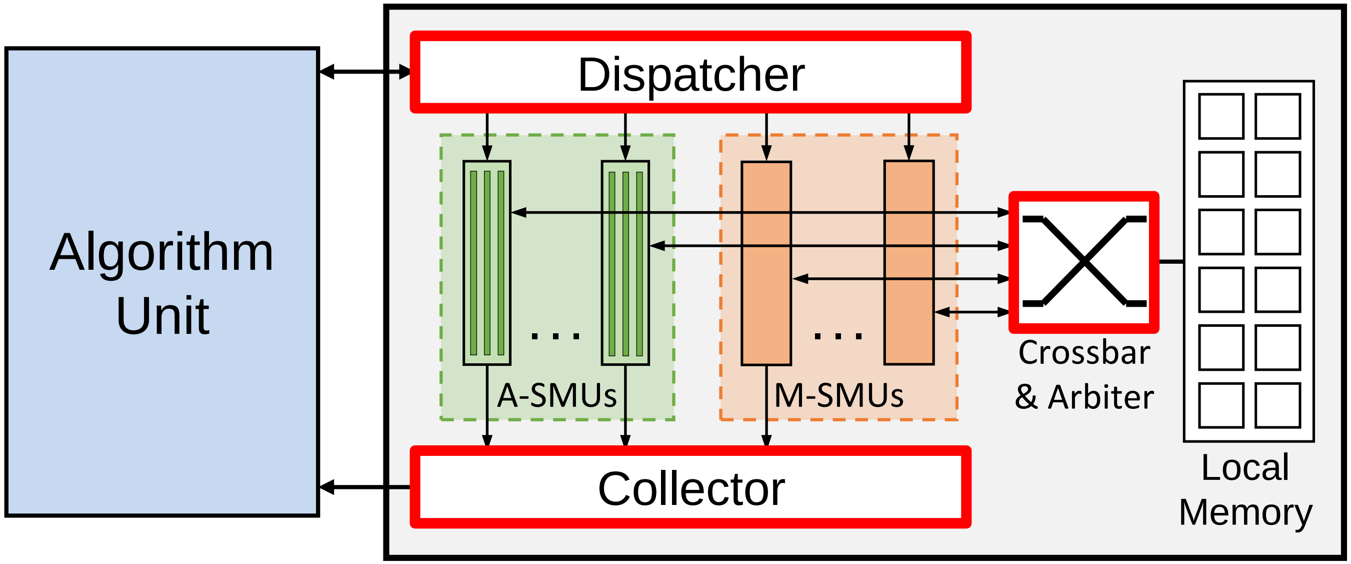
Accessor SMUs contain multiple lanes for parallel and out-of-order execution

Mutator SMUs may only execute when all other SMUs are idle (conservative assumption)



- Complex method code is removed from the program and synthesized into **specialized method units** (SMUs)
  - Accessor-SMUs (A-SMUs)
  - Mutator-SMUs (M-SMUs)

Other SCU Blocks



Specialized Container Unit (SCU)

- **Dispatcher**: handles requests and safely invokes the SMUs
- **Collector**: gathers results and returns them in calling order
- **Crossbar & Arbiter**: allows SMUs to share memory ports

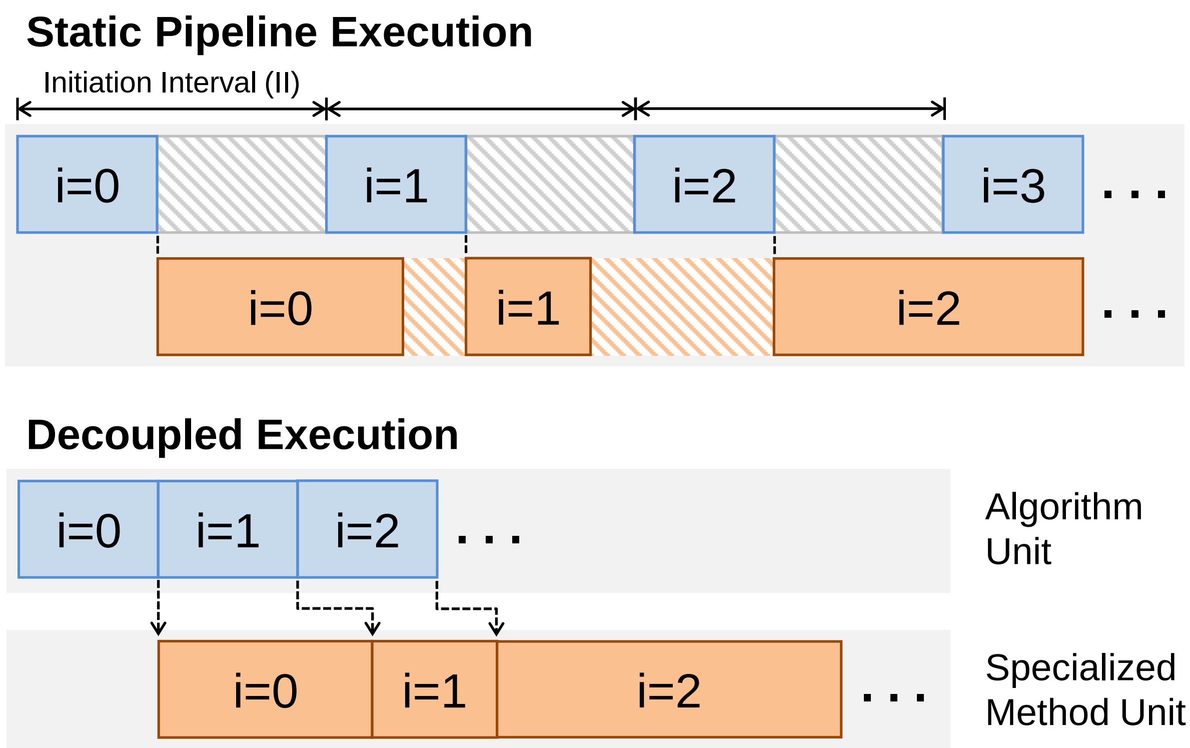
SCU Execution (Mutator)

Dijkstra's Algorithm

```
s = u.begin_neighbors();
e = u.end_neighbors();
// inner loop
for (v = s; v < e; ++v) {
  alt = dist[u] + edge[u][v];
  if (dist[v] > alt) {
    dist[v] = alt;
    // priority queue push
    Q.push(v, dist[v]);
  }
}
```

Algorithm

Complex Method



- Decoupling enables continuous execution without a static schedule and dynamically exploits parallelism

SCU Execution (Accessor)

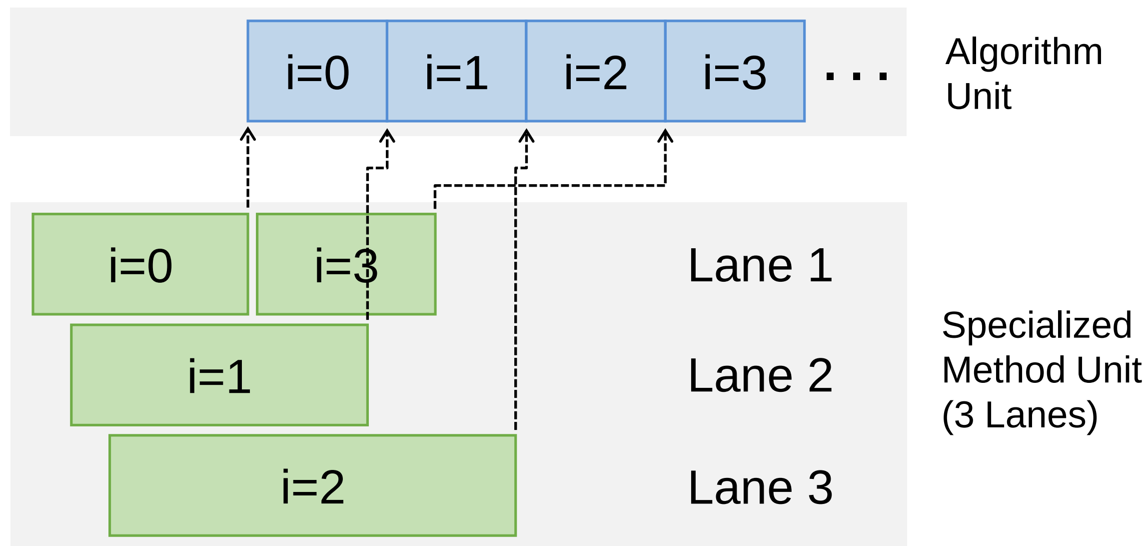
KeySearch Kernel

```
void key_search() {
  for (i = …) {
    // hash find
    n = table.find(k);
    // algorithm
    if (n != NULL)
      vals[i] = n.val;
  }
}
```

Algorithm

Complex Method

Decoupled Execution

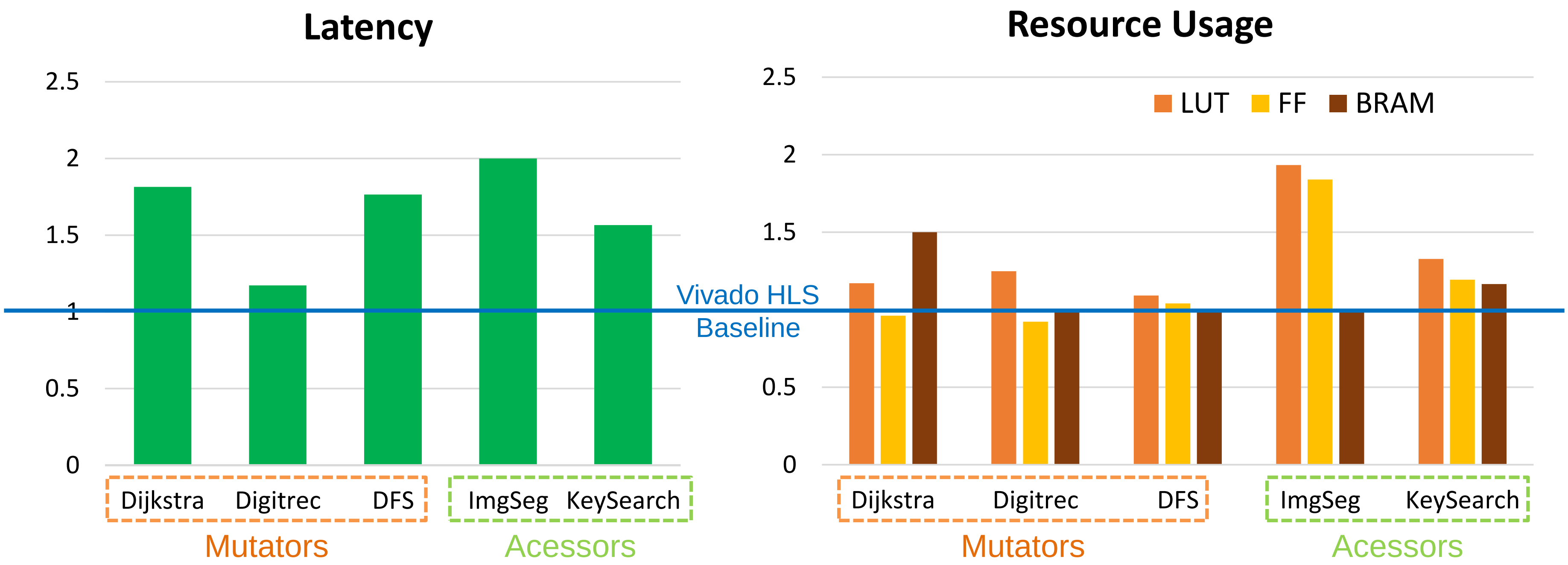


- Multiple lanes enable parallel and out-of-order method execution to greatly improve performance

Experimental Setup

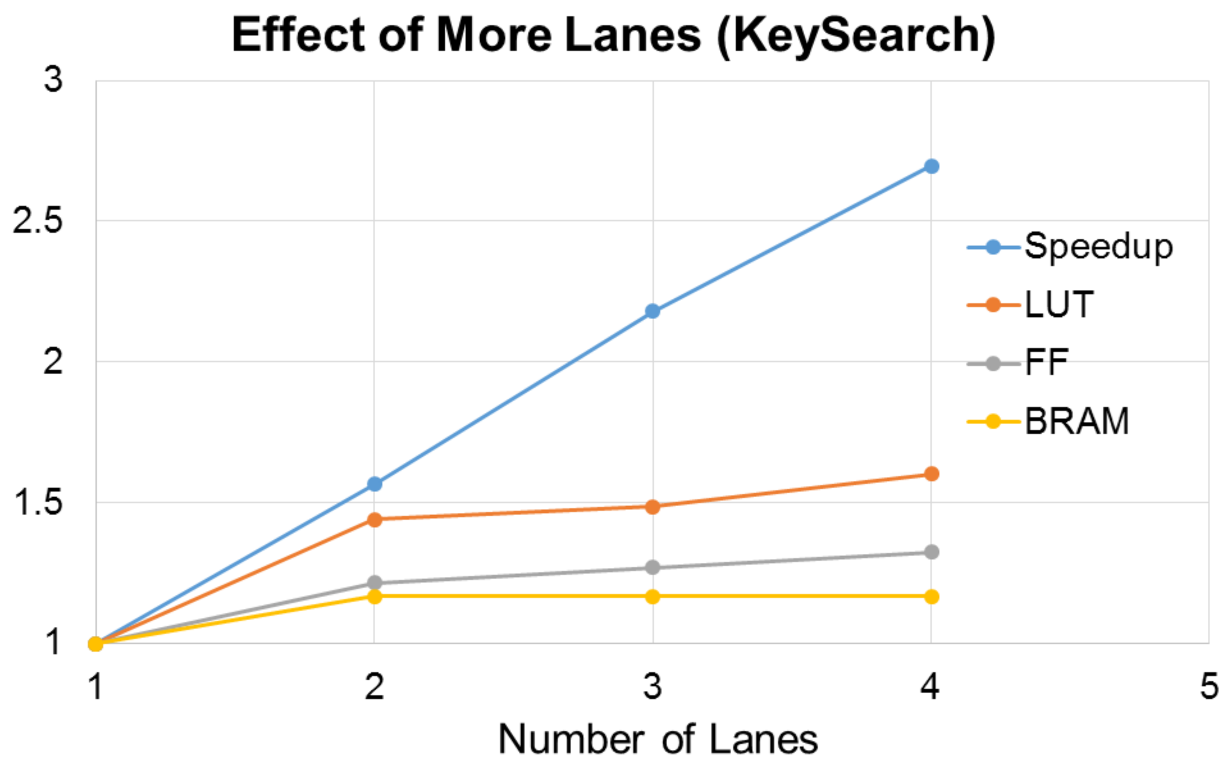
- **Baseline**
  - HLS program written in synthesizable C++
- **SCU Flow**
  - Extract the complex method code and use it to synthesize SMUs
  - Synthesize dispatcher, collector, etc. from C++ templates
- **Tools**
  - Vivado HLS as the HLS tool
  - Vivado 2015.3 to implement the generated HDL

Performance and Area Comparison



- Target device is Virtex-7, target clock period is 5ns
- <u>Average Speedup</u>: 1.6x
- <u>Average Area Overhead</u>: 30% LUT, 20% FF, 10% BRAM

Scalability



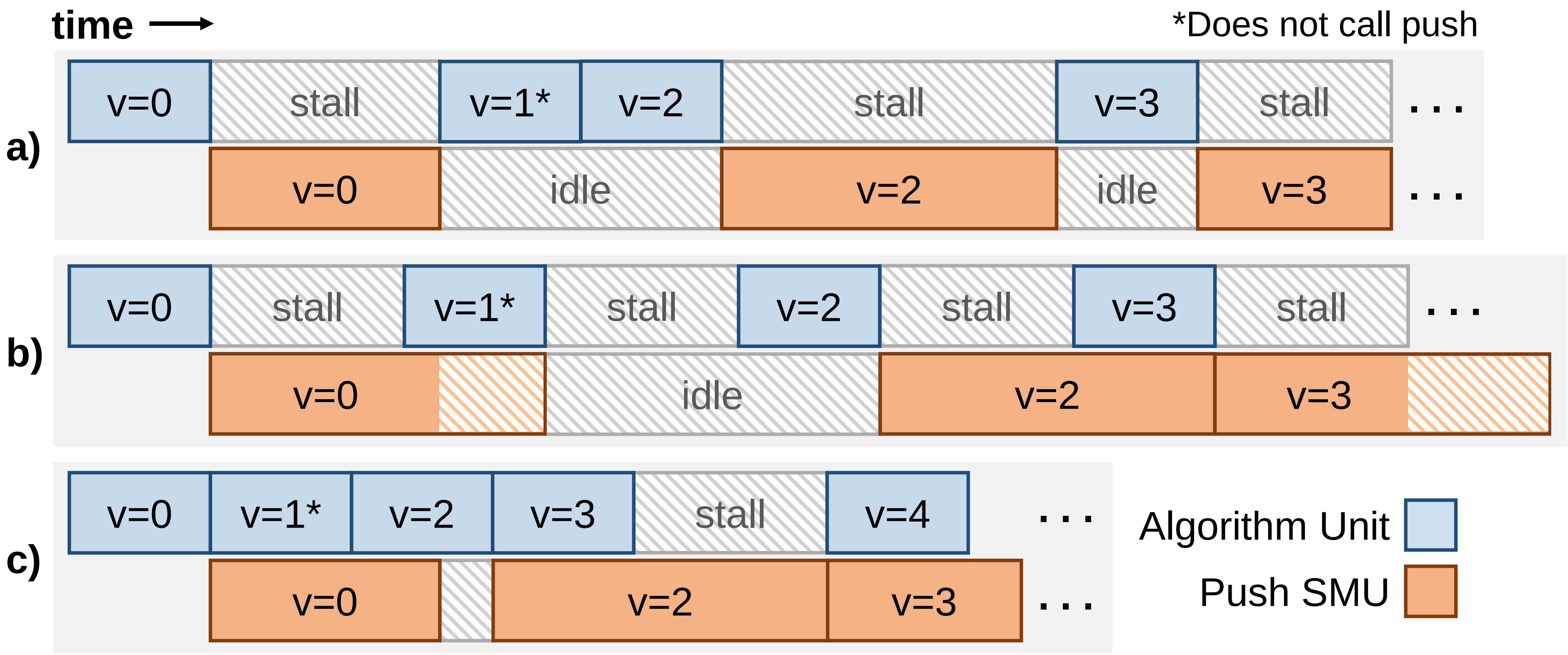
- Area overhead remains fairly constant while speedup continues to improve with more lanes

Conclusions

- Current and future HLS applications will contain both fixed and variable-latency code
- Decoupling the two parts and separately optimizing them is a promising approach
- **Where** to decouple and **what** optimizations to apply are key questions to address

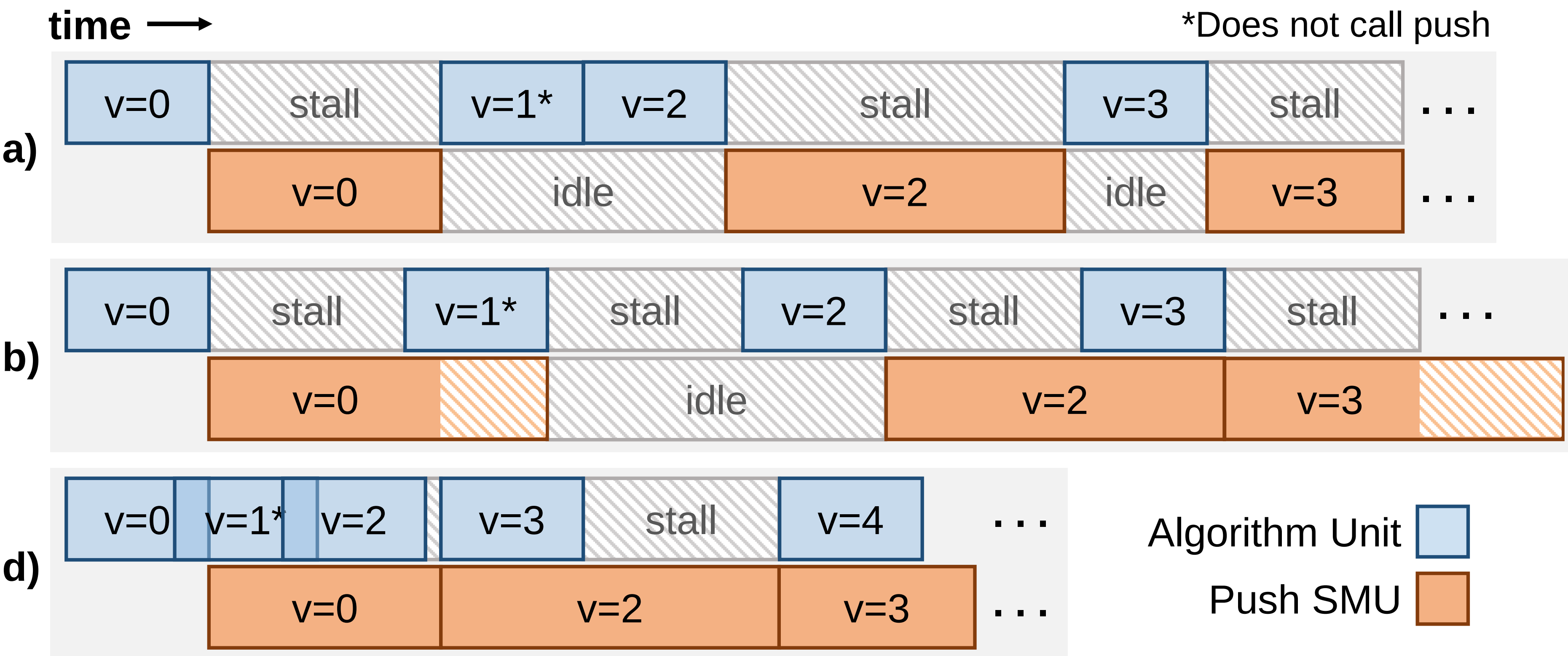
- **Future work**
  - Intelligent prefetching in the SCU
  - Overlapped mutator-SMU execution

Example Execution (Mutator)



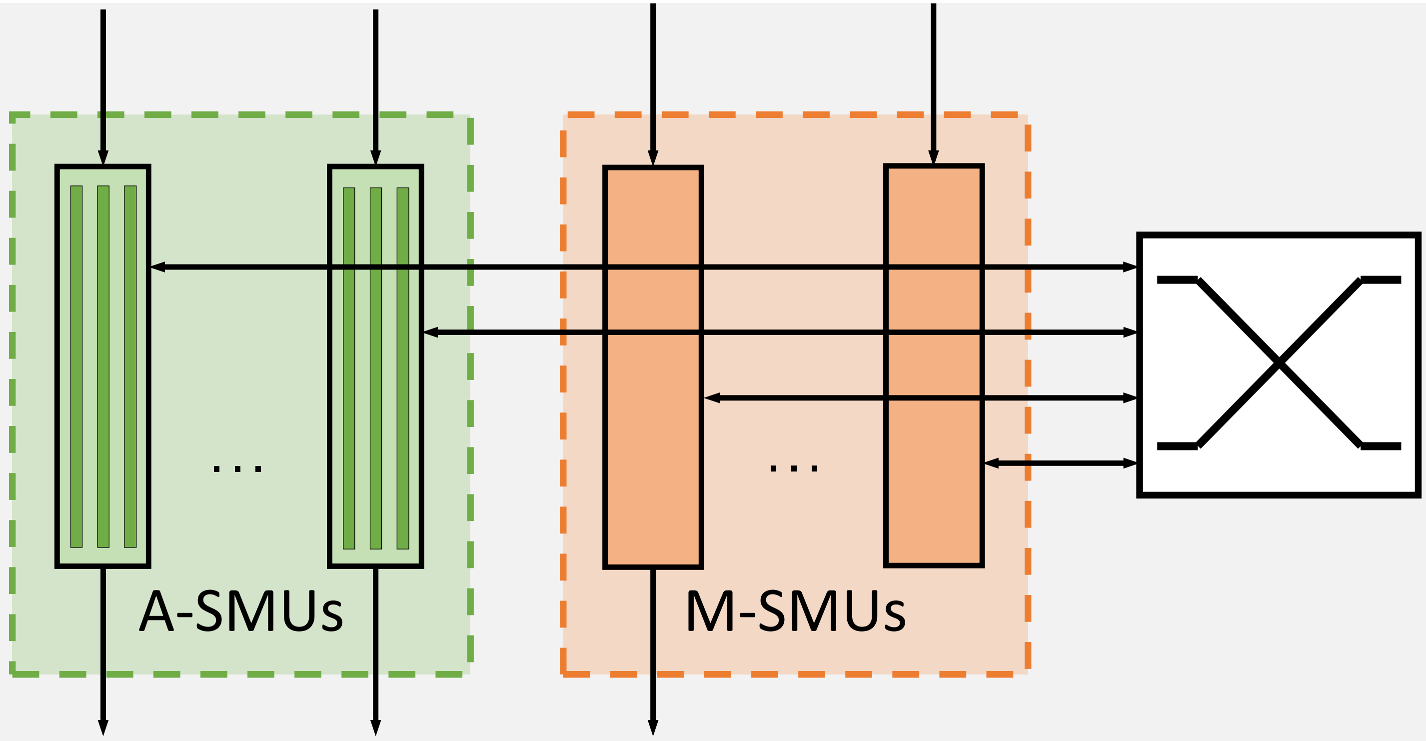
a) Baseline, non-pipelined
b) Baseline, pipelined
c) Decoupled, algorithm unit pipelined

Example Execution (Mutator)



a) Baseline, non-pipelined
b) Baseline, pipelined
c) Decoupled
d) Decoupled, algorithm pipelined

Specialized Method Units (SMUs)



- Complex method code is synthesized into **specialized method units** (SMUs)
- SMUs can be **accessor** or **mutator** SMUs