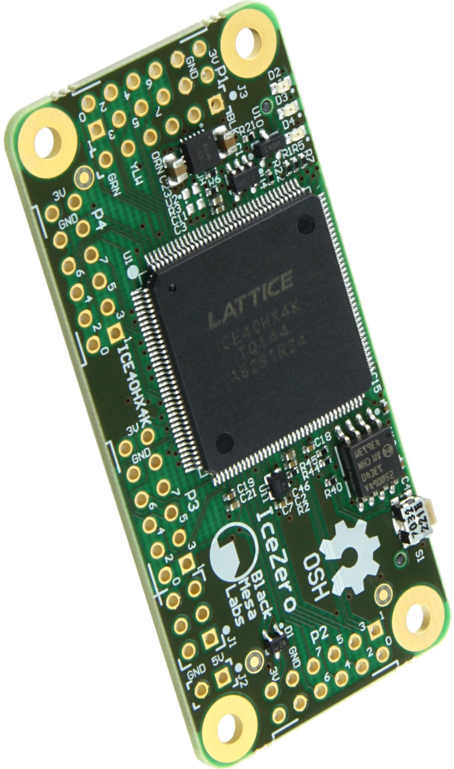
Dahlia

Predictable Accelerator Design with Time-Sensitive Affine types

Rachit Nigam, Sachille Atapattu, Ted Bauer, Adrian Sampson, Zhiru Zhang

What is an FPGA?

Field - Programmable Gate Array

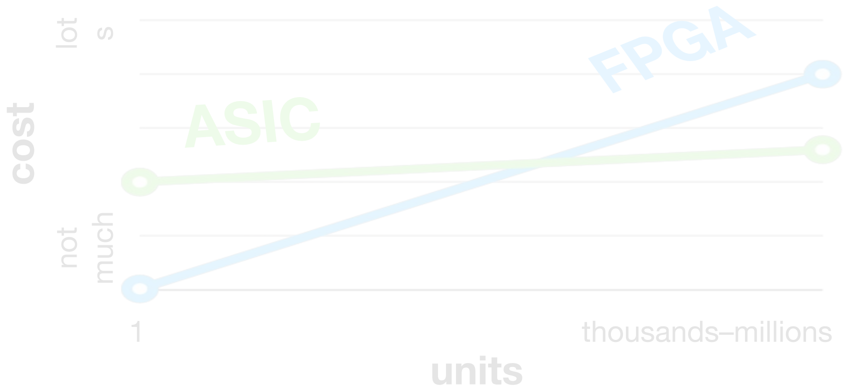


What is an FPGA?

FIELD-PROGRAMMABLE GATE ARRAY

A circuit emulator.

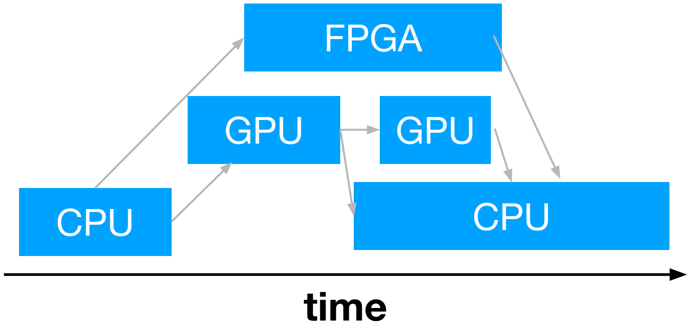
To prototype or replace an ASIC.



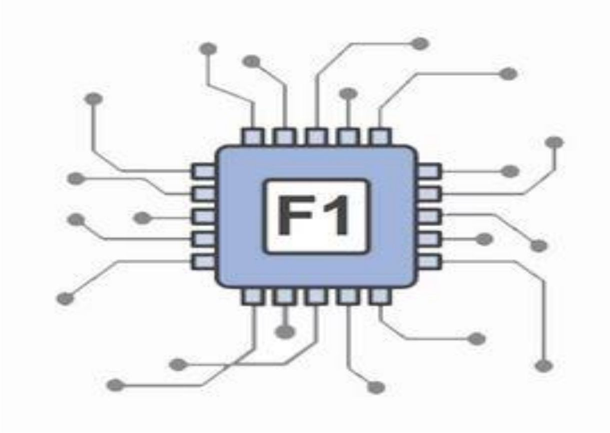
- signal processing
- networking
- embedded devices
- VLSI validation
- radio
- microcontrollers

A general accelerator.

To offload computation from a CPU.



- machine learning
- signal processing
- search
- graph analytics
- genomics
- scientific computing



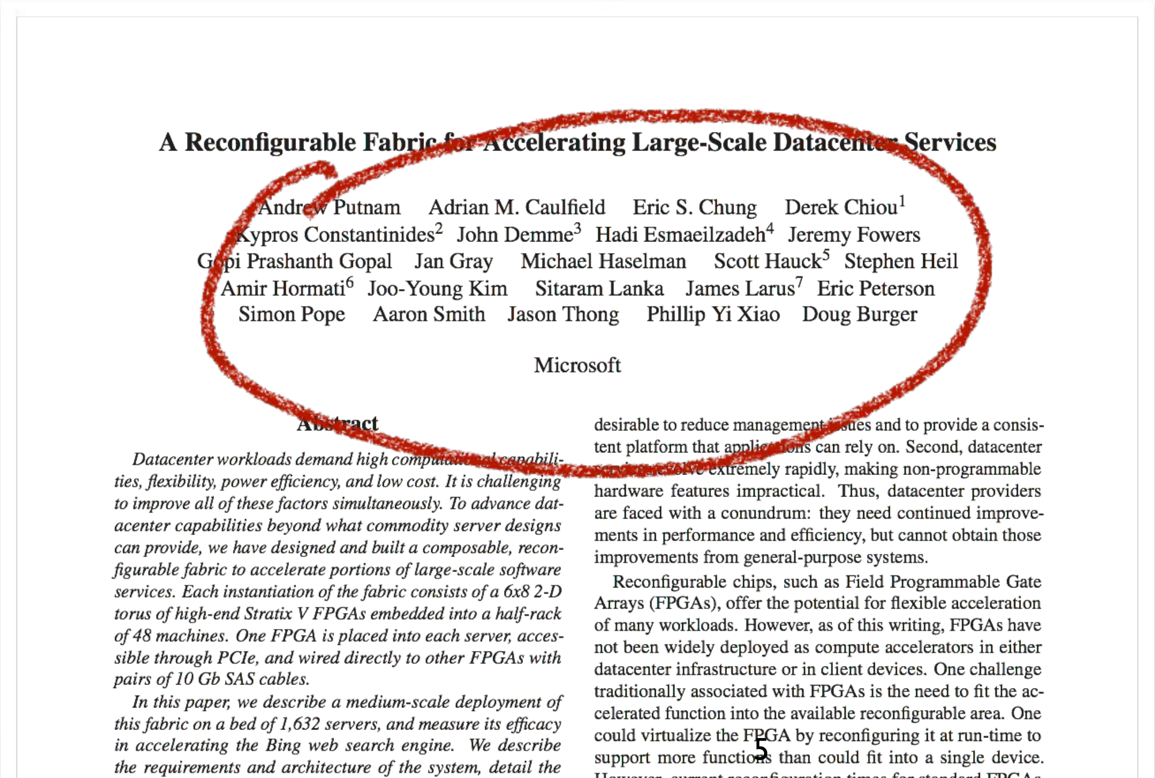
Amazon

EC2 F1 instances let you rent FPGAs by the hour, in the cloud.

2015
Bing Ranking throughput increased by 50%
Bing

Microsoft

Project Catapult: accelerate Bing search.
Project Brainwave: low-latency DNN inference.



A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

Andrew Putnam Adrian M. Caulfield Eric S. Chung Derek Chiou[1]
Kypros Constantinides[2] John Demme[3] Hadi Esmaeilzadeh[4] Jeremy Fowers
Gopi Prashanth Gopal Jan Gray Michael Haselman Scott Hauck[5] Stephen Heil
Amir Hormati[6] Joo-Young Kim Sitaram Lanka James Larus[7] Eric Peterson
Simon Pope Aaron Smith Jason Thong Phillip Yi Xiao Doug Burger

Microsoft

Abstract

Datacenter workloads demand high computational capabilities, flexibility, power efficiency, and low cost. It is challenging to improve all of these factors simultaneously. To advance datacenter capabilities beyond what commodity server designs can provide, we have designed and built a composable, reconfigurable fabric to accelerate portions of large-scale software services. Each instantiation of the fabric consists of a 6x8 2-D torus of high-end Stratix V FPGAs embedded into a half-rack of 48 machines. One FPGA is placed into each server, accessible through PCIe, and wired directly to other FPGAs with pairs of 10 Gb SAS cables.

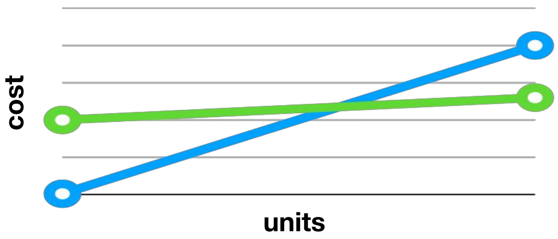
In this paper, we describe a medium-scale deployment of this fabric on a bed of 1,632 servers, and measure its efficacy in accelerating the Bing web search engine. We describe the requirements and architecture of the system, detail the

desirable to reduce management issues and to provide a consistent platform that applications can rely on. Second, datacenter services evolve extremely rapidly, making non-programmable hardware features impractical. Thus, datacenter providers are faced with a conundrum: they need continued improvements in performance and efficiency, but cannot obtain those improvements from general-purpose systems.

Reconfigurable chips, such as Field Programmable Gate Arrays (FPGAs), offer the potential for flexible acceleration of many workloads. However, as of this writing, FPGAs have not been widely deployed as compute accelerators in either datacenter infrastructure or in client devices. One challenge traditionally associated with FPGAs is the need to fit the accelerated function into the available reconfigurable area. One could virtualize the FPGA by reconfiguring it at run-time to support more functions than could fit into a single device. However, current reconfiguration times for standard FPGAs

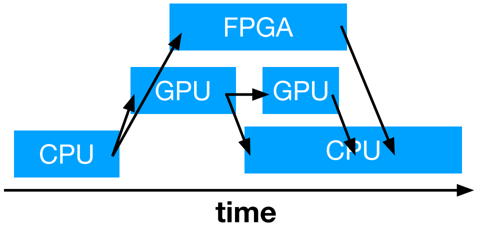
23 AUTHORS!

A circuit emulator.



≠

A general accelerator.



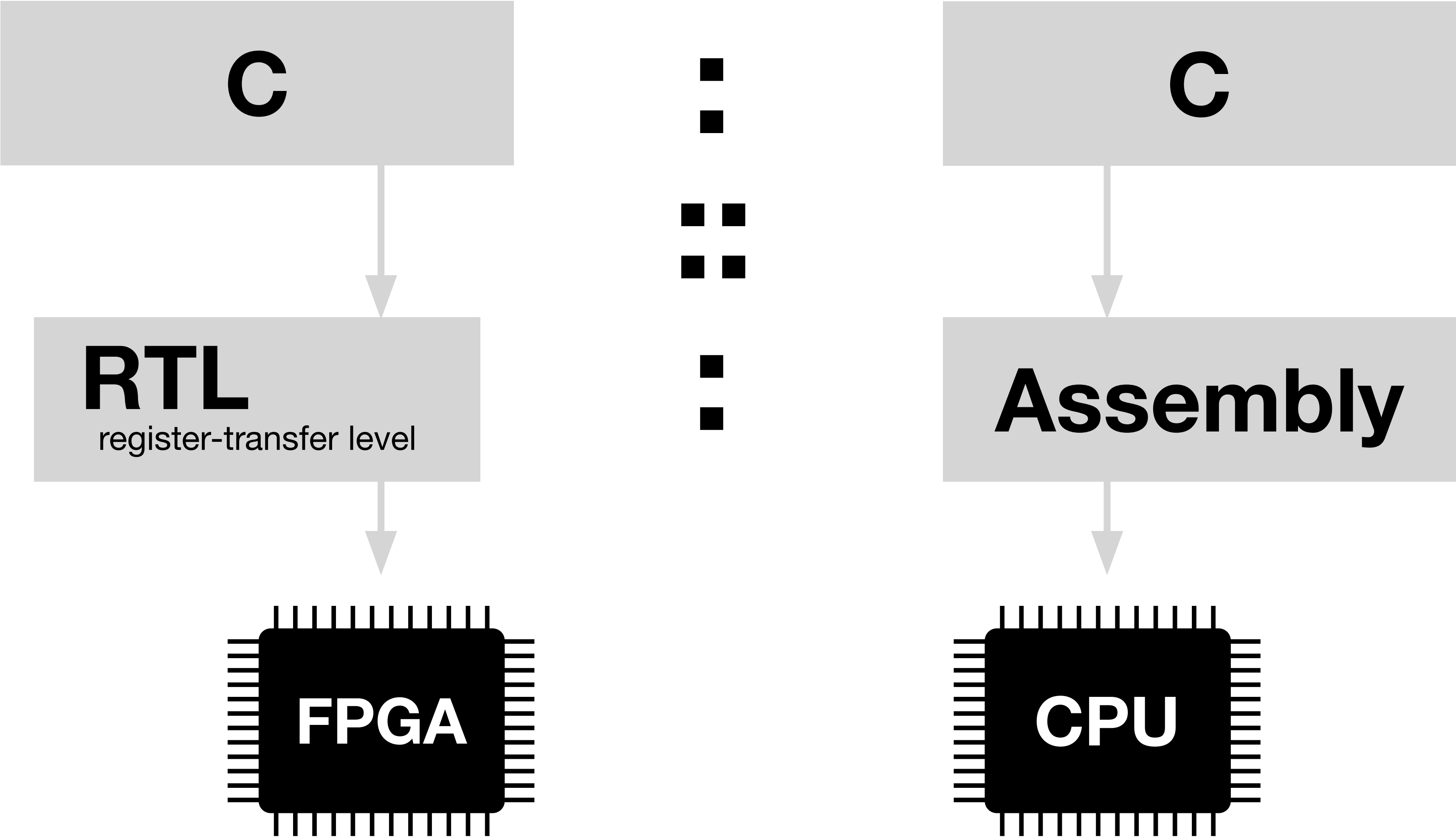
RTL

register-transfer level

Verilog VHDL Bluespec Chisel

Actual Silicon

© imec



C

RTL

register-transfer level

FPGA

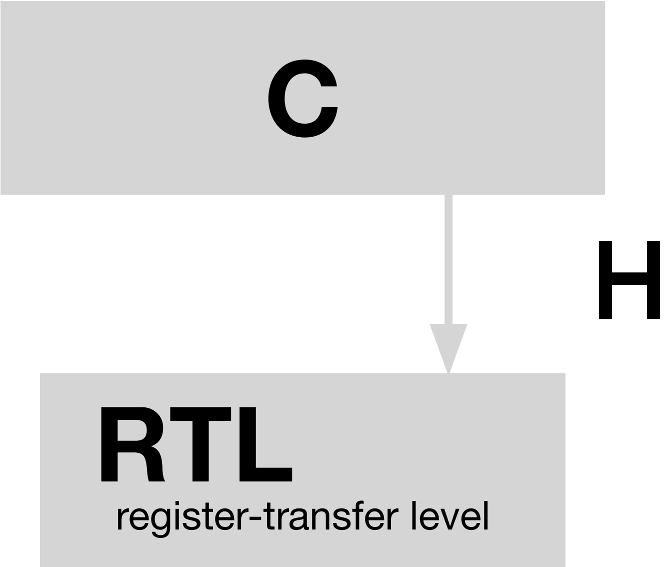
: :: :

C

Assembly

CPU

High-Level Synthesis



C

RTL

register-transfer level

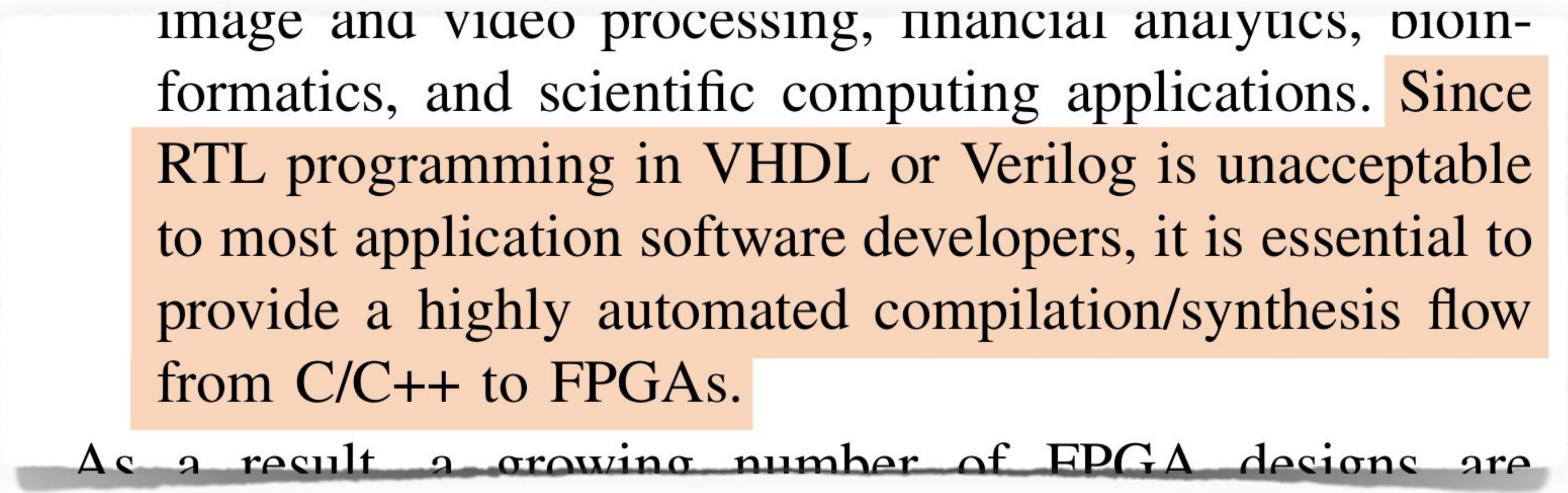


image and video processing, financial analytics, bioinformatics, and scientific computing applications. Since RTL programming in VHDL or Verilog is unacceptable to most application software developers, it is essential to provide a highly automated compilation/synthesis flow from C/C++ to FPGAs.

As a result, a growing number of FPGA designs are

Pitfalls of C-based HLS

HLS tools *transparently* compile C/C++ code and *predictably generate* performant hardware *automatically*.

```
void bias(float* vec, float* biases);
void bias(float* vec, float* biases);
```

Unsupported memory access on the variable 'biases', which is (or contains) an array with unknown size at compile time.

```
int fac(int n, int acc) {
  if (n == 0)
    return acc;
  else
    return fac(n - 1, n * acc);
}
```

```
int fac(int n) {
  if (n == 0)
    return 1;
  else
    return n * fac(n - 1);
}
```

```
int fac(int n, int acc) {
  if (n == 0)
    return acc;
  else
    return fac(n - 1, n * acc);
}
```

✓

```
int fac(int n) {
  if (n == 0)
    return 1;
  else
    return n * fac(n - 1);
}
```

✗

Source: https://forums.xilinx.com/t5/Vivado-High-Level-Synthesis-HLS/Vivado-HLS-and-Tail-Recursive-Functions/td-p/265980

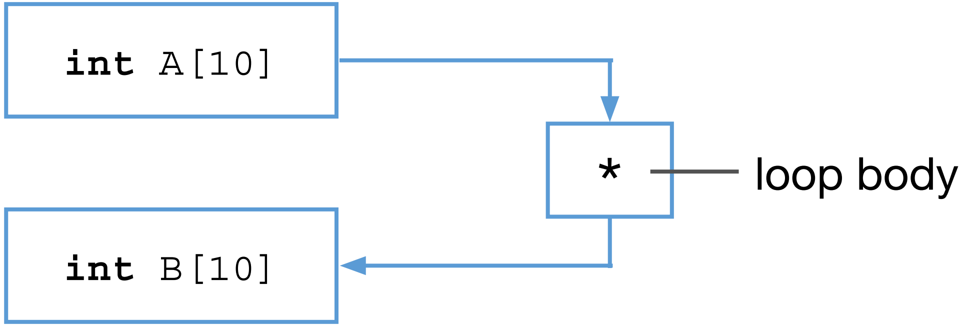
HLS tools *transparently* compile C/C++ code and *predictably generate* performant hardware *automatically*.

- An *undefined subset* of C with simple control and static operators supported.

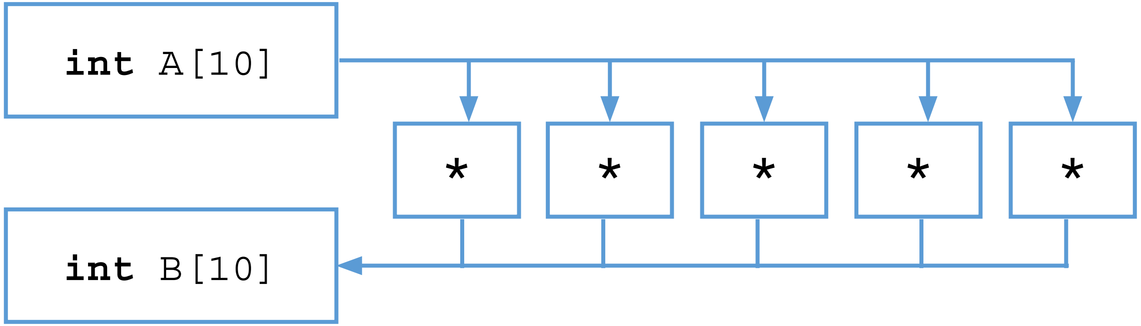
HLS tools ~~*transparently*~~ compile C/C++ code and *predictably generate* performant hardware *automatically*.

- An *undefined subset* of C with simple control and static operators supported.

```
int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

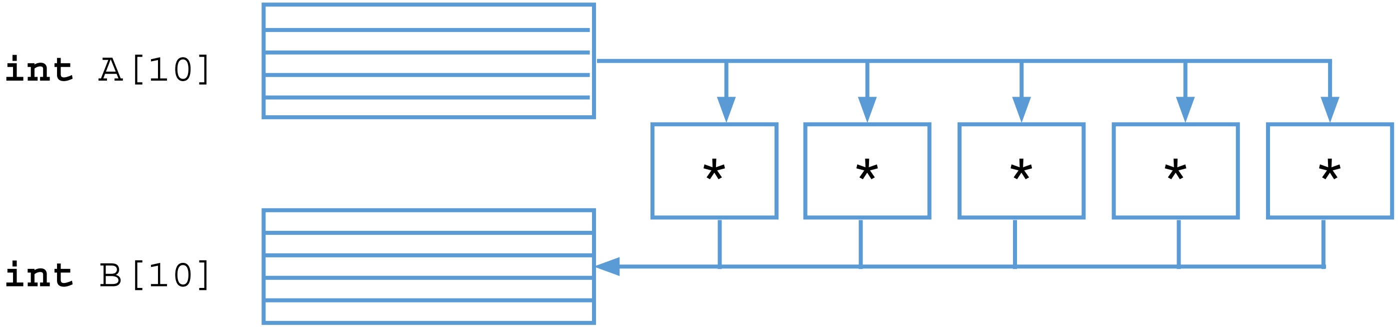


int A[10]
*
loop body
int B[10]



int A[10]
int B[10]
*
*
*
*
*

```
int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

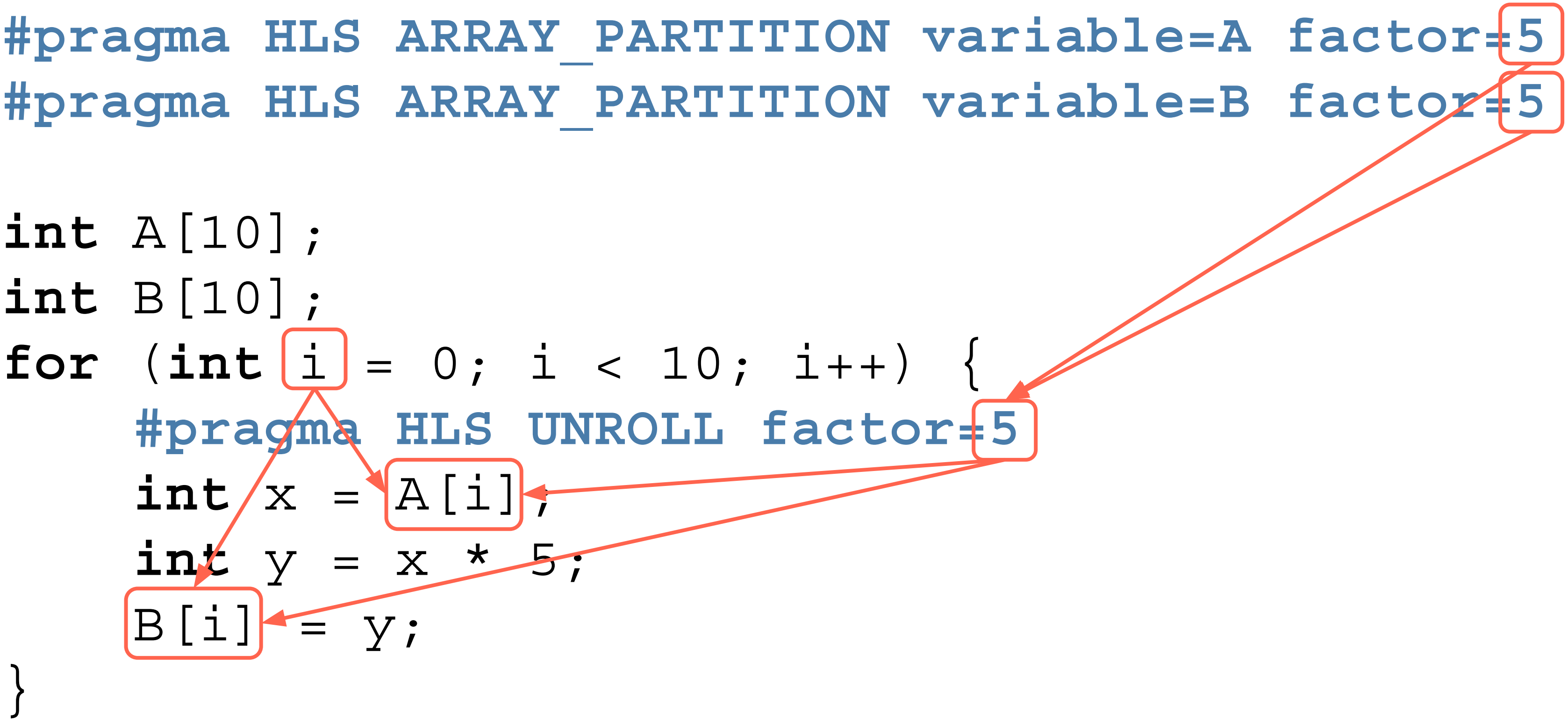


int A[10]

int B[10]

```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```



```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

HLS tools ~~*transparently*~~ compile C/C++ code and *predictably generate* performant hardware *automatically*.

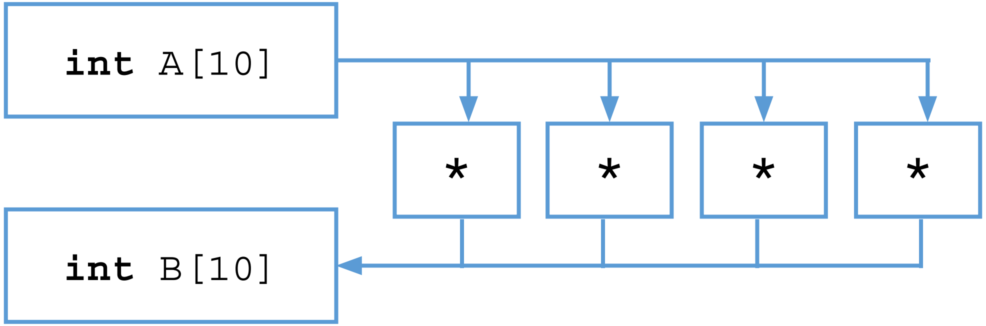
- An *undefined subset* of C with simple control and static operators supported.
- *Second class* primitives for controlling design decisions. User hints needed for high performance design.

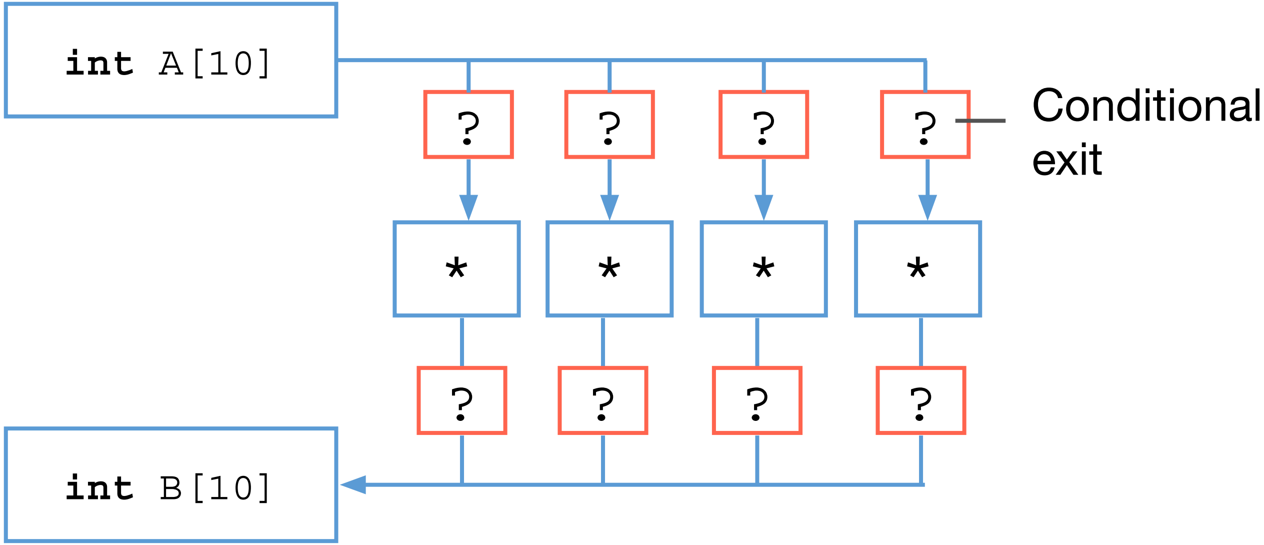
HLS tools *~~transparently~~* compile C/C++ code and *predictably generate* performant hardware *~~automatically~~*.

- An *undefined subset* of C with simple control and static operators supported.
- *Second class* primitives for controlling design decisions. User hints needed for high performance design.

```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=4
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

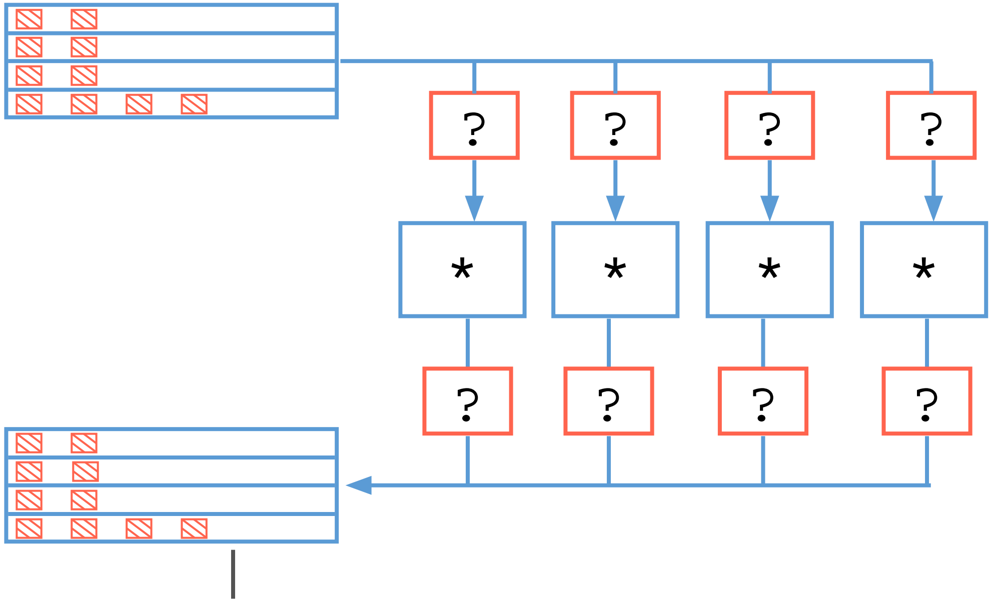

int A[10]
*
*
*
*
int B[10]


int A[10]
?
?
?
?
*
*
*
*
?
?
?
?
int B[10]
Conditional exit

```
#pragma HLS ARRAY_PARTITION variable=A factor=4
#pragma HLS ARRAY_PARTITION variable=B factor=4

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=4
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

int A[10]

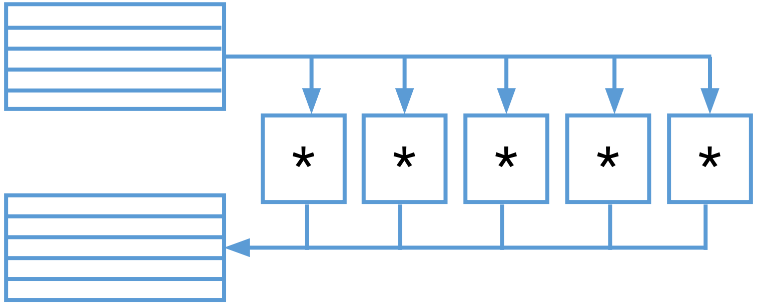


int B[10]

Mismatched partition sizes

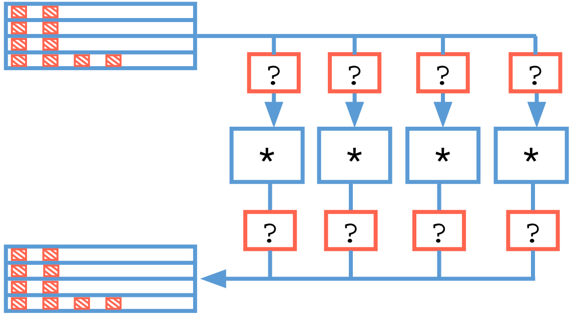
```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```



```
#pragma HLS ARRAY_PARTITION variable=A factor=4
#pragma HLS ARRAY_PARTITION variable=B factor=4

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=4
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```



```
#pragma HLS ARRAY_PARTITION variable=A factor=M
#pragma HLS ARRAY_PARTITION variable=B factor=M

int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=M
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

HLS tools ~~*transparently*~~ compile C/C++ code and *predictably generate* performant hardware ~~*automatically*~~.

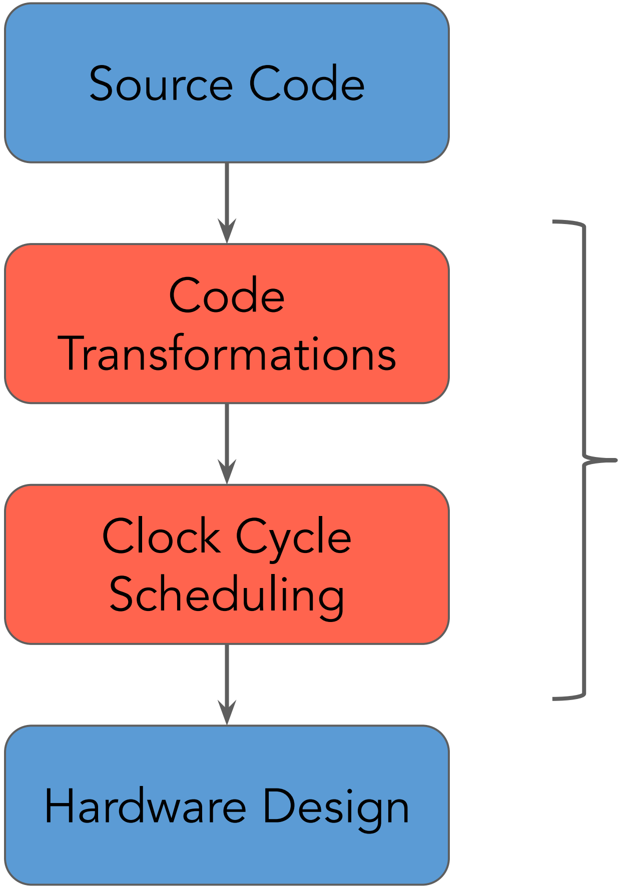
- An *undefined subset* of C with simple control and static operators supported.
- *Second class* primitives for controlling design decisions. User hints needed for high performance design.
- Non trivial additions to designs to support parallel semantics for C.

HLS tools ~~*transparently*~~ compile C/C++ code and ~~*predictably generate*~~ performant hardware ~~*automatically*~~.

- An *undefined subset* of C with simple control and static operators supported.
- *Second class* primitives for controlling design decisions. User hints needed for high performance design.
- Non trivial additions to designs to support parallel semantics for C.

HLS tools ~~transparently~~ compile C/C++ code and ~~predictably generate~~ performant hardware ~~automatically~~.

HLS tools compile an *ill-defined*, imperative language and *unpredictably generate* performant designs with *user aid*.



Outside User Control

(Unpredictable)

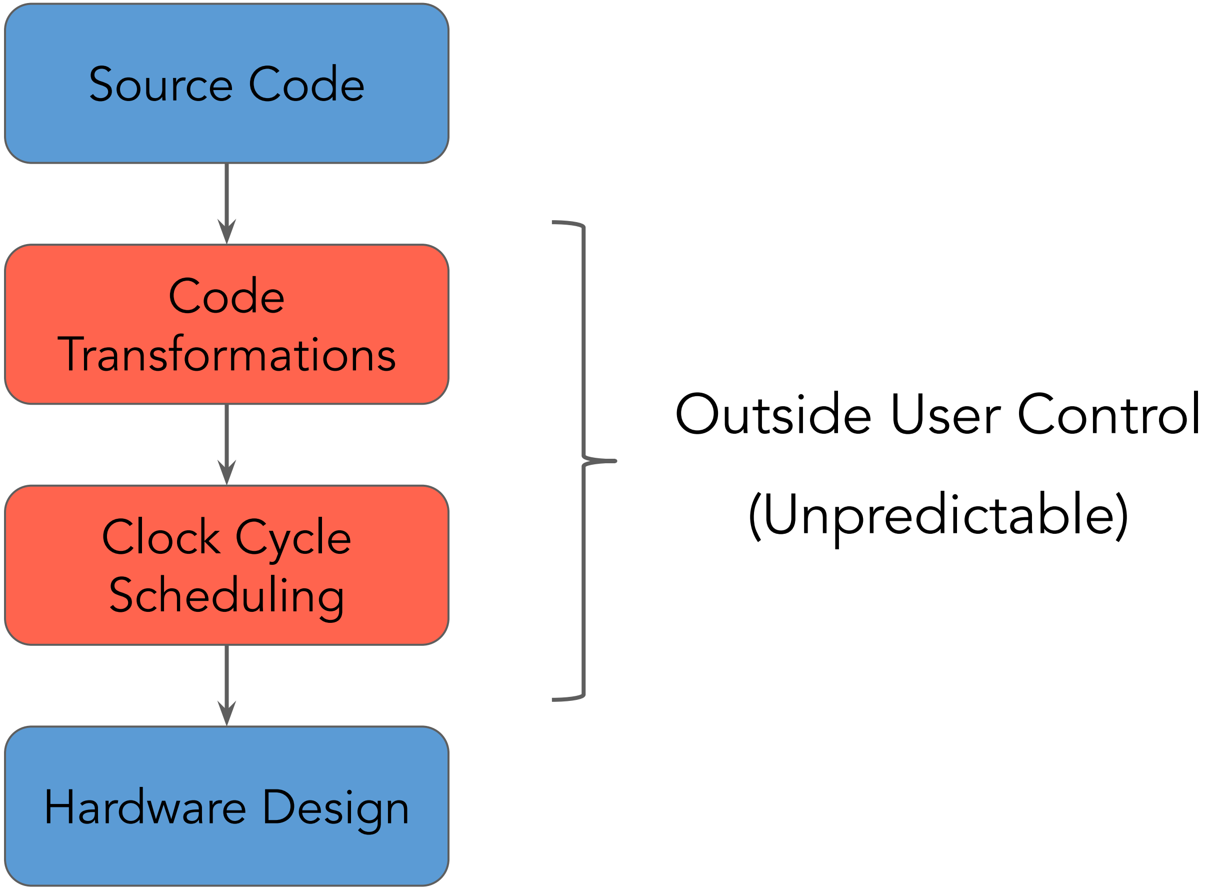
Dahlia

Compiles a *well-defined* imperative language and *predictably* generates performant hardware with *user-aid*

Predictable Hardware Generation

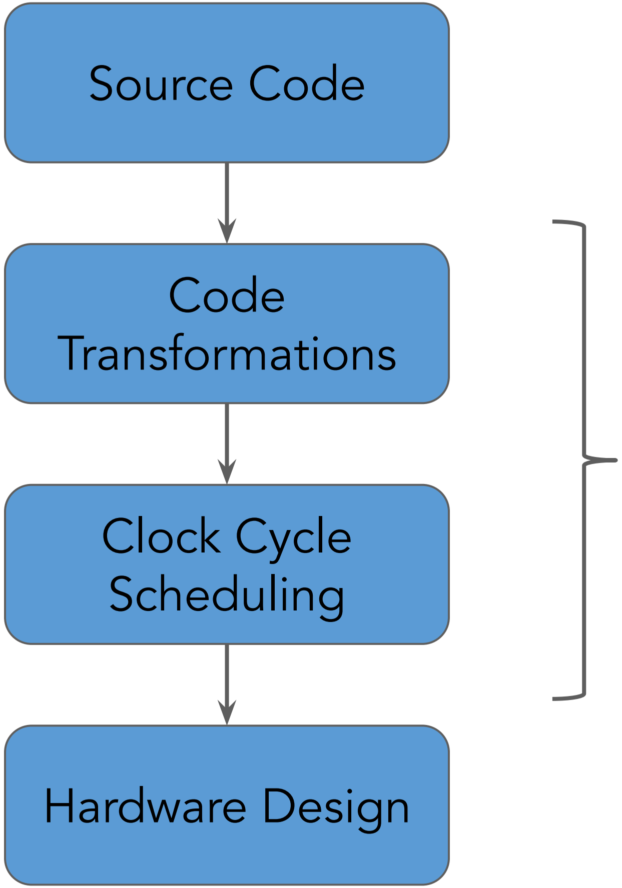
- Only allow programs that represent *hardware cost* at the source level.

- Reject programs that require complicated transformations to work in hardware.



Outside User Control

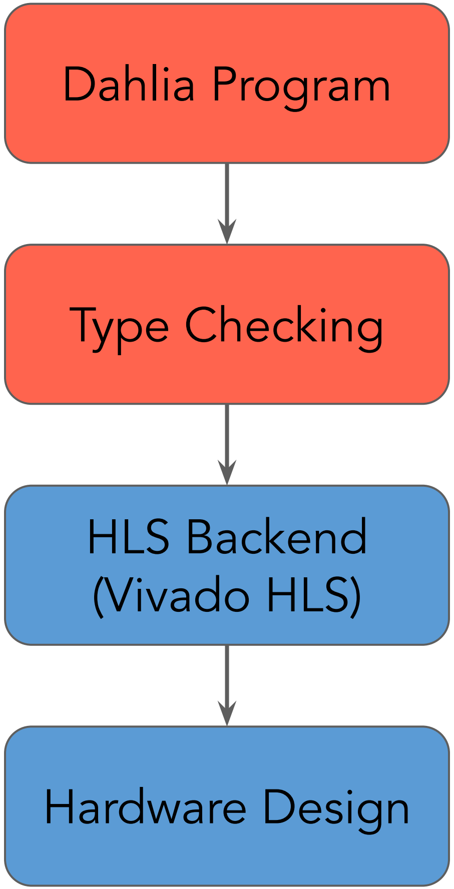
(Unpredictable)



User controlled using source level representation

Design Goals

- *Source-level reasoning* over implicit scheduling decisions.
- *Predictable hardware generation*: Use an affine type system to disallow programs with unpredictable hardware generation.
- *First class scheduling primitives* to control design decisions. Guarantee that scheduling primitives don't change semantics.



Dahlia Program

Type Checking

HLS Backend
(Vivado HLS)

Hardware Design

```
decl A: float[10];
decl A: float[10];
```

Global memories.
Mapped to BRAMs.

```
decl A: float[10];

let x = A[0];
A[9] = 1;
```

```
decl A: float[10];

let x = A[0];
A[9] = 1;
```

Error: Physical resource `A' already used in this context

```
A ; B
```

A and B *may* run in *parallel*

```
let x = A[0];

A[9] = 1;
```

Conflict! Cannot read and write to `A' in the same cycle. Assume all memories are single ported.

```
decl A: float[10];

let x = A[0]
---
A[9] = 1;
```

A logical time step

```
A --- B
```

Run A and then run B.
Create a *logical timestep.*

```
let x = A[0]
---
A[9] = 1
```

No conflict.
Statements run in separate timesteps.

```
{ A --- B };
{ C --- D };
     E
```

Specify complex parallel behavior using --- and ;

```
decl A: bit<32>[M][N];
decl x: bit<32>[N];
decl y: bit<32>[N];
decl tmp: bit<32>[M];

for (let i = 0..N) {
  y[i] := 0.0;
}
---
for (let i = 0..M) {
  tmp[i] := 0.0;
  ---
  for (let j = 0..N) {
    let t = tmp[i];
    ---
    tmp[i] := t + A[i][j] * x[j];
  }
  ---
  for (let j = 0..N) {
    let y0 = y[j];
    ---
    y[j] := y0 + A[i][j] * tmp[i];
  }
}
```

Source level reasoning for cycle boundaries.

```
decl A: bit<32>[M][N];
decl x: bit<32>[N];
decl y: bit<32>[N];
decl tmp: bit<32>[M];

for (let i = 0..N) {
  y[i] := 0.0;
}
---
for (let i = 0..M) {
  tmp[i] := 0.0;
  ---
  for (let j = 0..N) {
    let t = tmp[i];
    ---
    tmp[i] := t + A[i][j] * x[j];
  }
  ---
  for (let j = 0..N) {
    let y0 = y[j];
    ---
    y[j] := y0 + A[i][j] * tmp[i];
  }
}
```

Source level reasoning for cycle boundaries.

```
decl A: bit<32>[10];
decl B: bit<32>[10];
for (i = 0 to 10) {
  A[i] = A[i + 1] - 1;
}
```

Bounded *DOALL* loops. Cross iteration dependencies disallowed.

```
decl A: bit<32>[10];
decl B: bit<32>[10];
for (i = 0 to 10) {
  A[i] = A[i + 1] - 1;
}
```

Error! Cross iteration dependency.

```
decl A: bit<32>[10];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
}
```

First class unrolling primitive. Typechecker reasons about array access exclusion with unrolling.

```
decl A: bit<32>[10];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
}
```

Error: Insufficient resources for parallel access.
Required: 5, Available: 1.

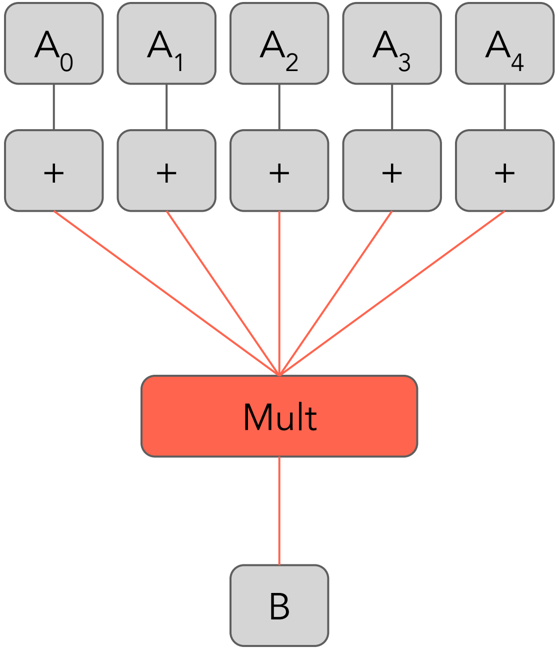
```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
}
```

First class banking primitive. Generates extra resources for parallel accesses.

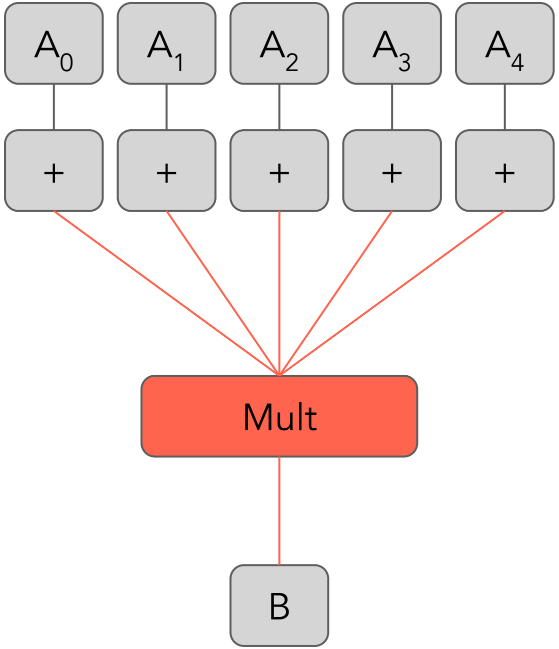
```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```

Explicit instantiation for reduction trees. Can contain arbitrary computation pipelines.

```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```



```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```

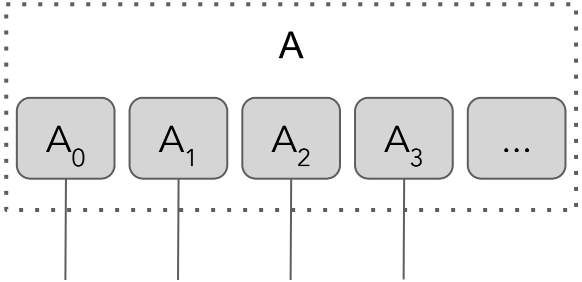


```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```

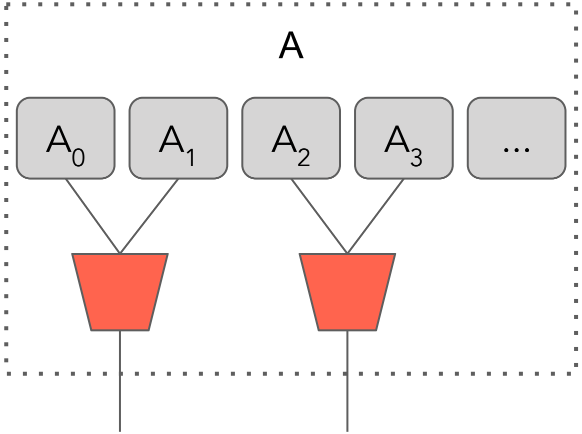
```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```

Error! Banking factor and unrolling factor do not match for access `A[i]’

```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```



```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];
for (i = 0 to 10) unroll 5 {
  let x = A[i] + 1;
} combine {
  B[0] *= x
}
```



```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];

view sh_A = shrink A[by 5];

for (...) unroll 5 {
  let x = sh_A[i] + 1;
} combine {
  B[0] *= x
}
```

Memory View : Generate additional hardware to access `A’ as a 5 banked memory.

```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];

view sh_A = shrink A[by 5];

for (...) unroll 5 {
  let x = sh_A[i] + 1;
} combine {
  B[0] *= x
}
```

Type checking ensures that parallelization/pipelining directives preserve *functional correctness* of the sequential program.

```
decl A: bit<32>[10 bank 10];
decl B: bit<32>[10];

view sh_A = shrink A[by 5];

for (...) unroll 5 {
  let x = sh_A[i] + 1;
} combine {
  B[0] *= x
}
```

Type checking ensures that parallelization/pipelining directives preserve *functional correctness* of the sequential program.

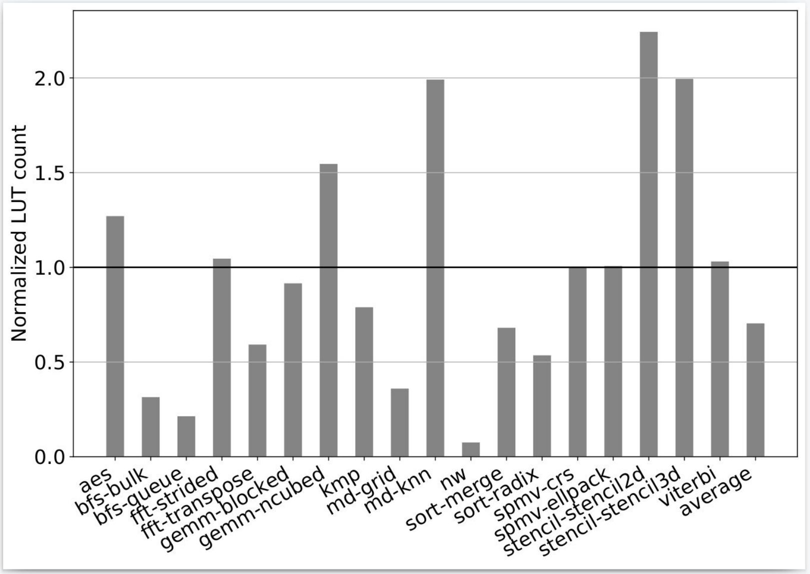
Dahlia

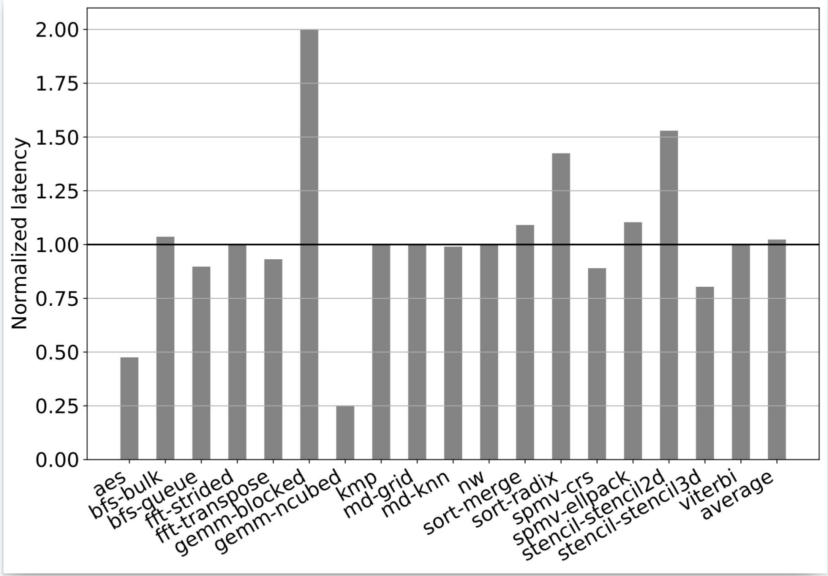
- *Source-level reasoning* over implicit scheduling decisions.
- *Predictable hardware generation*: Use an affine type system to disallow programs with unpredictable hardware generation.
- *First class scheduling primitives* to control design decisions. Guarantee that scheduling primitives don't change semantics.

Evaluation

- Machsuite benchmarks. Port 18 HLS C programs to Dahlia.
- Initial minimal effort porting: Syntactically rewrite C programs to Dahlia.
- Design Space Exploration with Dahlia: Rewrite ports to use Dahlia’s parallelization primitives. Compare with DSE in HLS C.

Evaluation


Normalized LUT count
0.0
0.5
1.0
1.5
2.0
aes
bfs-bulk
bfs-queue
fft-strided
fft-transpose
gemm-blocked
gemm-ncubed
kmp
md-grid
md-knn
nw
sort-merge
sort-radix
spmv-crs
spmv-ellpack
stencil-stencil2d
stencil-stencil3d
viterbi
average


Normalized latency
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
aes
bfs-bulk
bfs-queue
fft-strided
fft-transpose
gemm-blocked
gemm-ncubed
kmp
md-grid
md-knn
nw
sort-merge
sort-radix
spmv-crs
spmv-ellpack
stencil-stencil2d
stencil-stencil3d
viterbi
average

Future Work

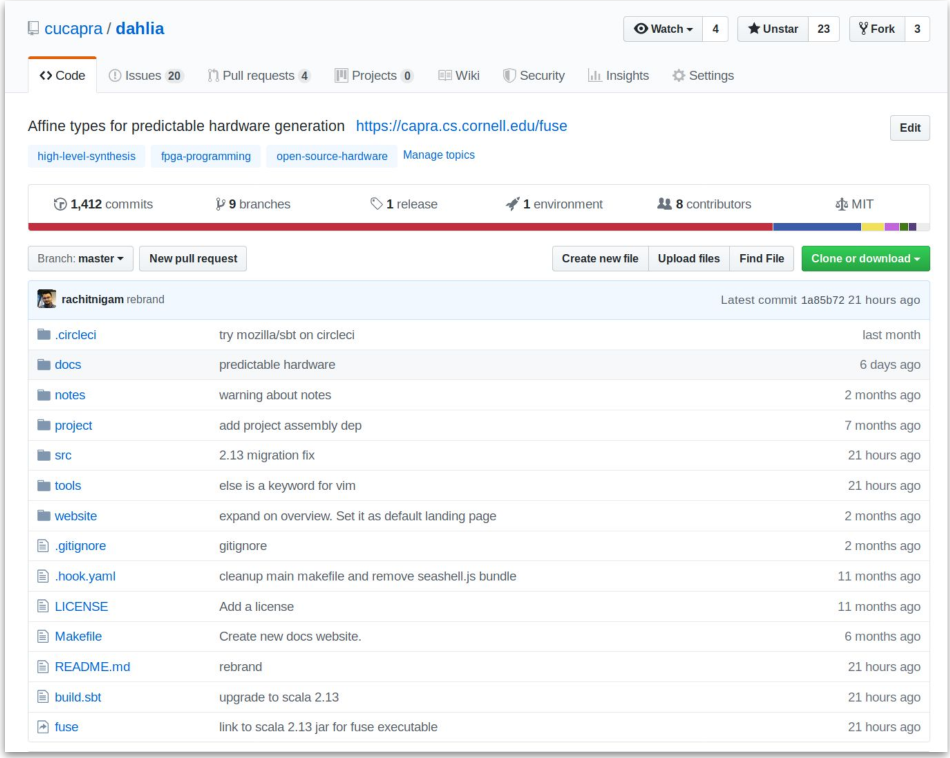
- *Affine functions*: Area-efficiency trade-offs with reusable hardware modules.

- *Multi-ported memories*: Crucial for high performance designs on modern FPGAs.

- *Constraint guided DSE*: Use type constraints to generate constraints for DSE.

Thanks!

capra.cs.cornell.edu/fuse



Affine types for predictable hardware generation https://capra.cs.cornell.edu/fuse

rachitnigam rebrand Latest commit 1a85b72 21 hours ago

.circleci	try mozilla/sbt on circleci	last month
docs	predictable hardware	6 days ago
notes	warning about notes	2 months ago
project	add project assembly dep	7 months ago
src	2.13 migration fix	21 hours ago
tools	else is a keyword for vim	21 hours ago
website	expand on overview. Set it as default landing page	2 months ago
.gitignore	gitignore	2 months ago
.hook.yaml	cleanup main makefile and remove seashell.js bundle	11 months ago
LICENSE	Add a license	11 months ago
Makefile	Create new docs website.	6 months ago
README.md	rebrand	21 hours ago
build.sbt	upgrade to scala 2.13	21 hours ago
fuse	link to scala 2.13 jar for fuse executable	21 hours ago

Constrain directed Optimization

Constrain directed Optimization

```
#pragma HLS ARRAY_PARTITION variable=A factor=M
#pragma HLS ARRAY_PARTITION variable=B factor=L
int A[10];
int B[10];
for (int i = 0; i < K; i++) {
    #pragma HLS UNROLL factor=N
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

Constrain directed Optimization

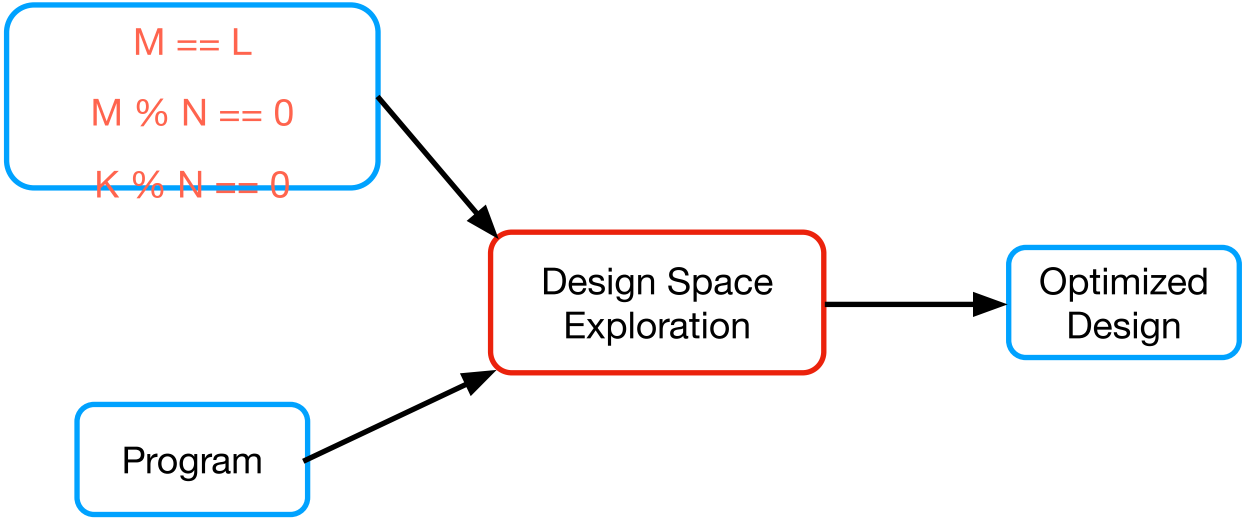
```
#pragma HLS ARRAY_PARTITION variable=A factor=M
#pragma HLS ARRAY_PARTITION variable=B factor=L
int A[10];
int B[10];
for (int i = 0; i < K; i++) {
    #pragma HLS UNROLL factor=N
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

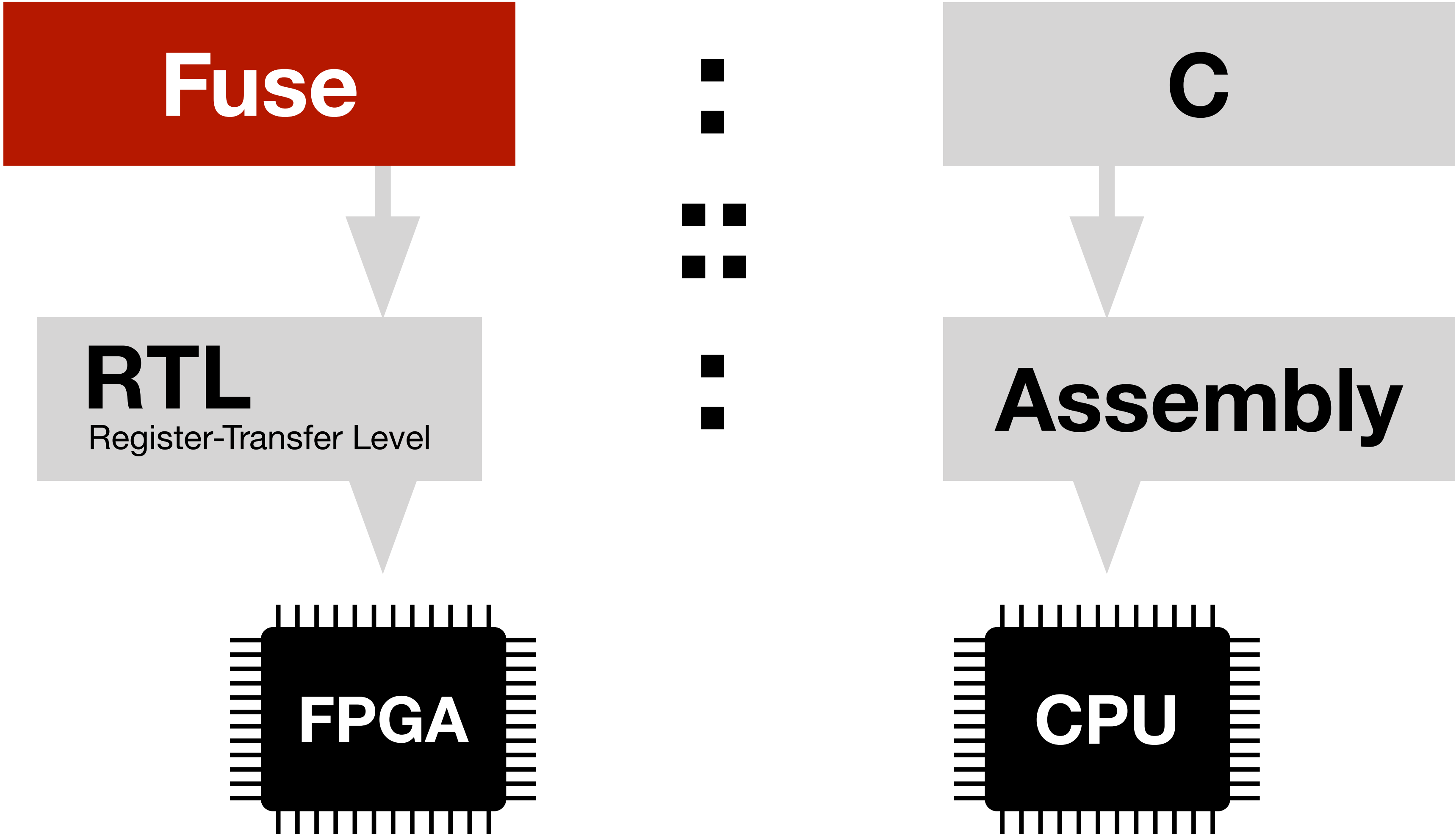
M == L

M % N == 0

K % N == 0

Constrain directed Optimization





Fuse : C

::

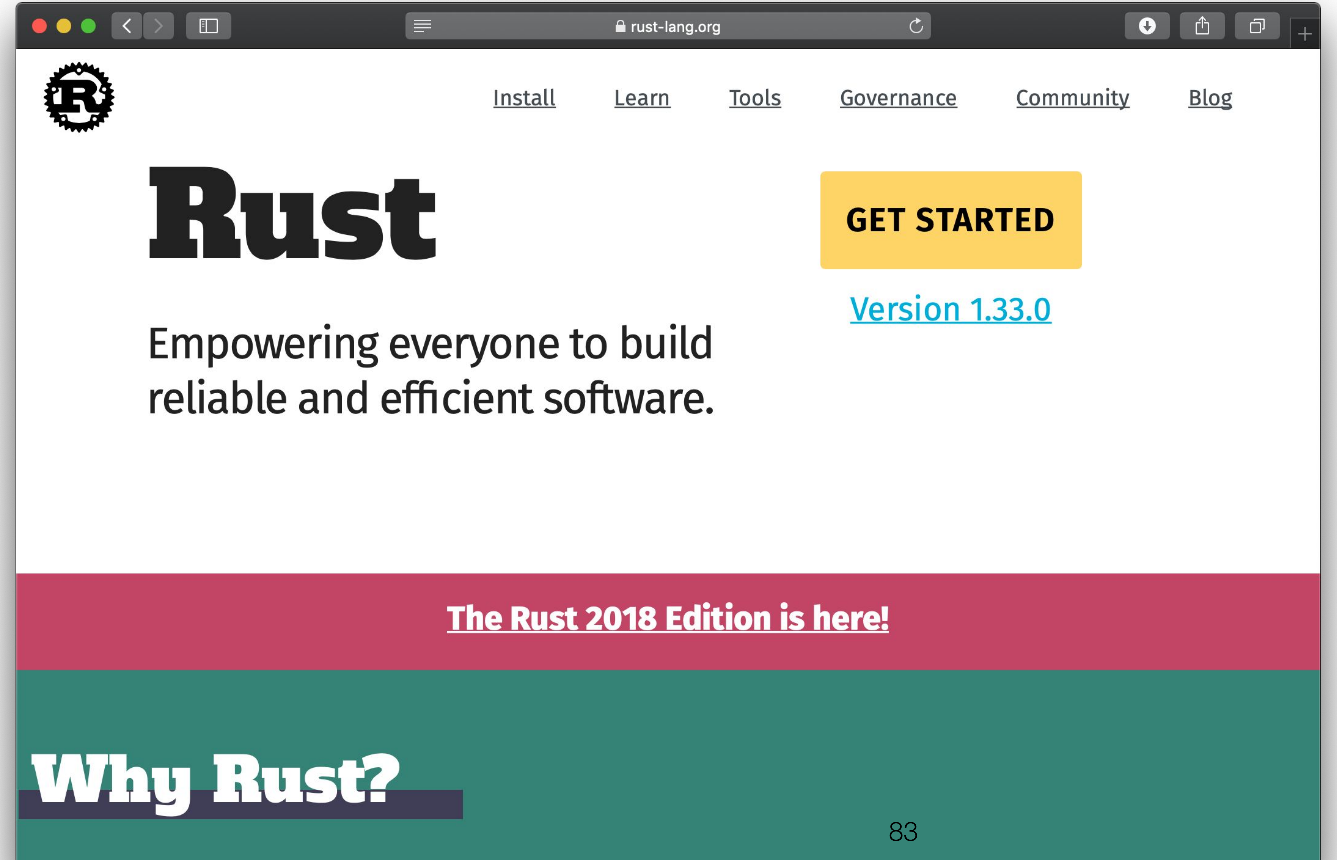
RTL : Assembly

Register-Transfer Level

FPGA

CPU

Affine types and **linear types**, as made famous recently by **Rust**.



rust-lang.org

Install Learn Tools Governance Community Blog

Rust

GET STARTED

Version 1.33.0

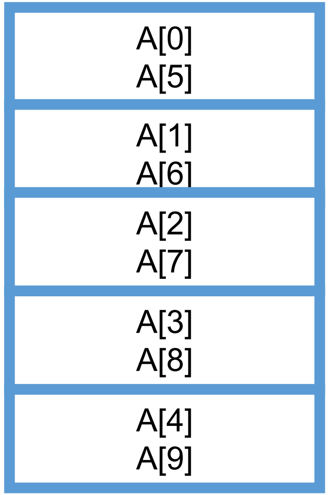
Empowering everyone to build reliable and efficient software.

The Rust 2018 Edition is here!

Why Rust?

Banked memory types

```
memory bank(5) A : int[10];
```



A[0]
A[5]

A[1]
A[6]

A[2]
A[7]

A[3]
A[8]

A[4]
A[9]

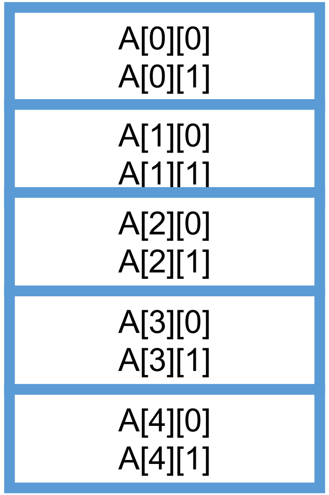
Banked memory types

```
memory bank(5) A : int[10];
for (let i in 0..1) {
    access A[0][i];
    access A[1][i];
    access A[2][i];
    access A[3][i];
    access A[4][i];
}
```

// one access to each
A[j] allowed here

STATIC

DYNAMIC



A[0][0]
A[0][1]

A[1][0]
A[1][1]

A[2][0]
A[2][1]

A[3][0]
A[3][1]

A[4][0]
A[4][1]

Hybrid indices for unrolling

```
memory bank(5) A : int[10];
for (let i in 0..9) unroll 5 {
    access A[??][??];
}
```

Hybrid indices for unrolling

```
memory bank(5) A : int[10];
for (let i in 0..9) unroll 5 {
    access A[i];
}
```