Gesture Recognition Application on GA144

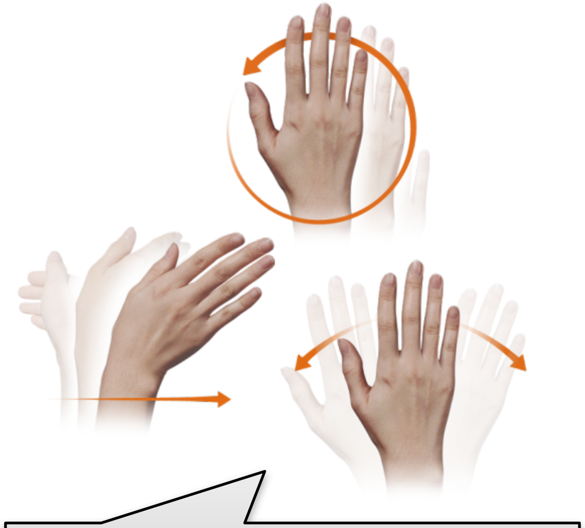
Phitchaya Mangpo Phothilimthana
Michael Schuldt

Berkeley
UNIVERSITY OF CALIFORNIA

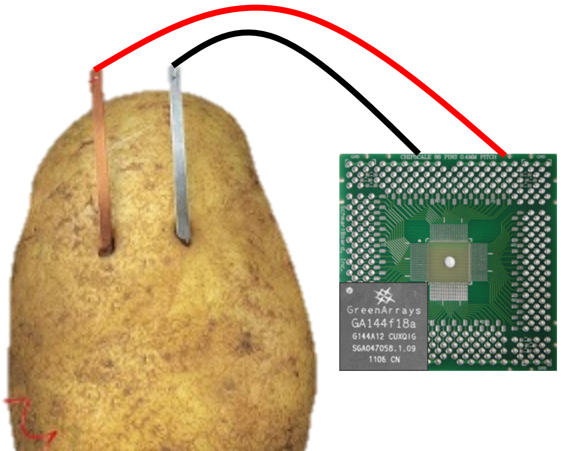
Rastislav Bodik

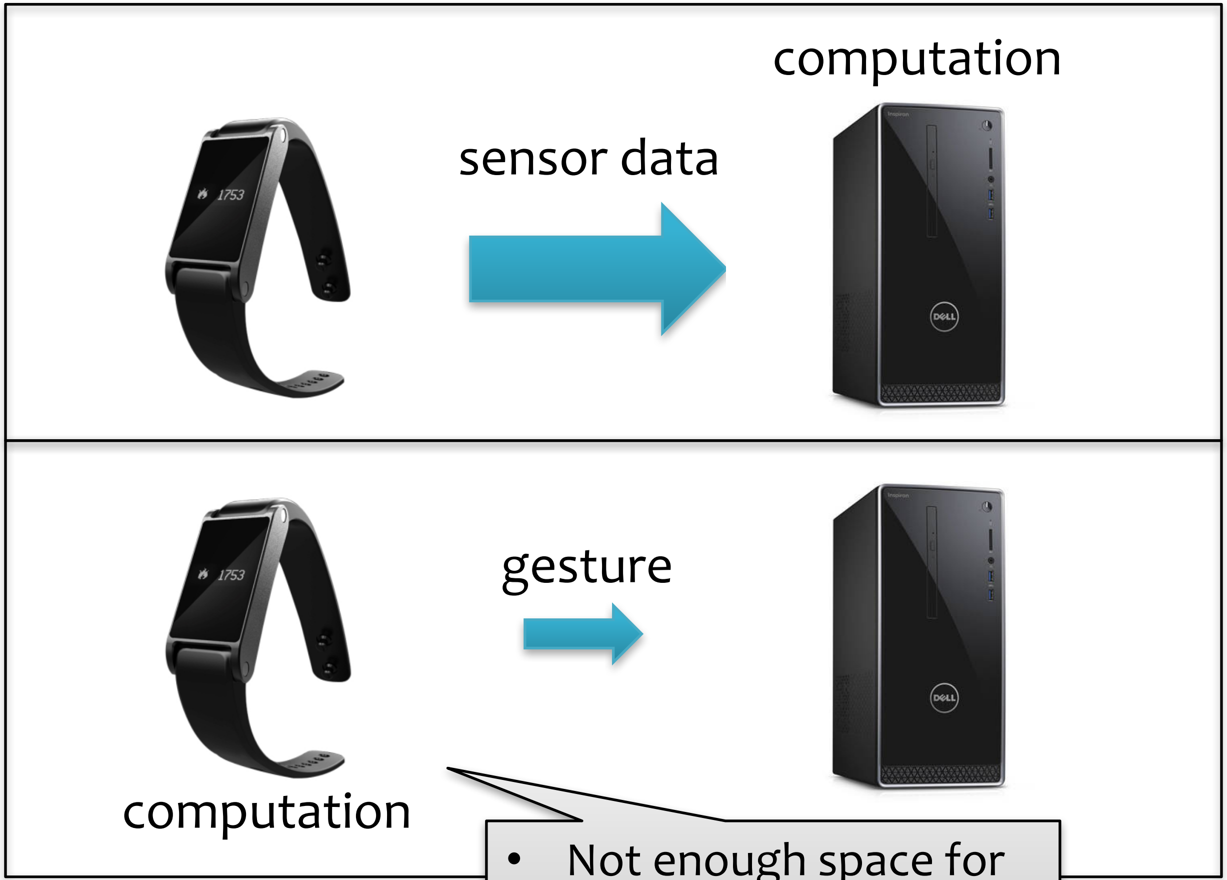
W UNIVERSITY of WASHINGTON

Classification Applications



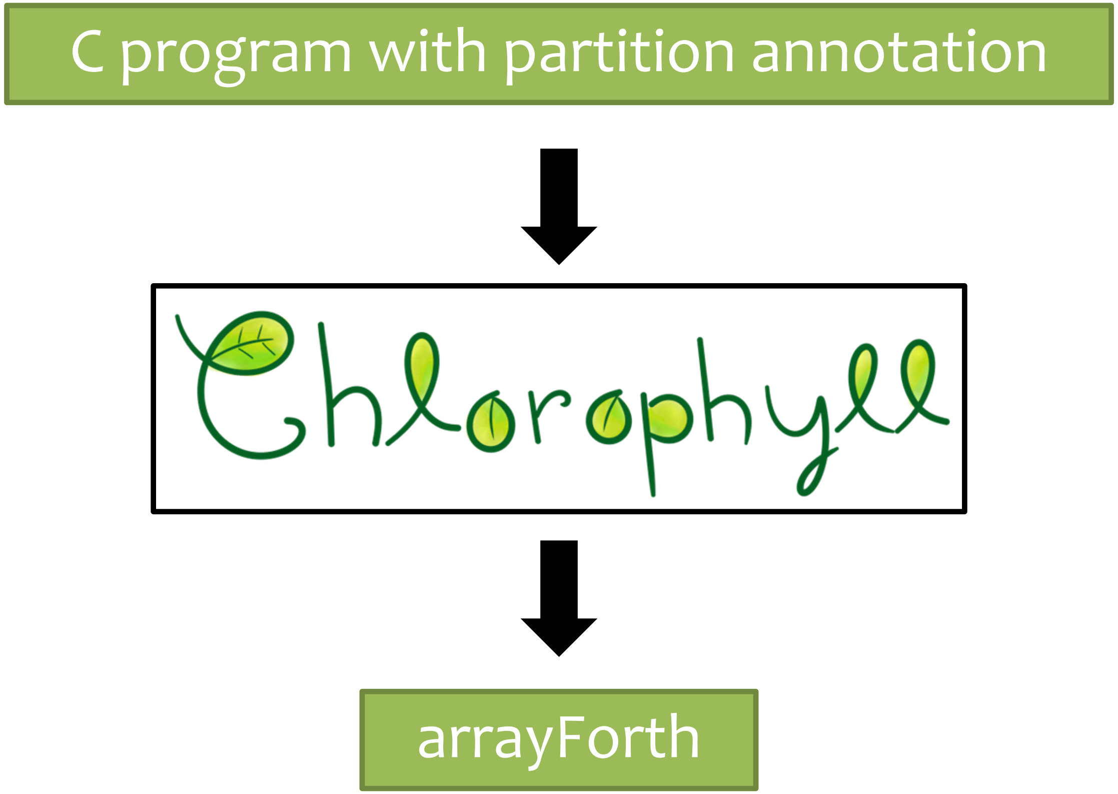
- Intensive computation
- Large data storage for gesture models





- Not enough space for data and program
- Not enough energy
- Too slow

Compiler



C program with partition annotation

Chlorophyll

arrayForth

Spatial programming model

```
int x[12];
for(i from 1 to 12)
  x[i] += x[i-1];
```

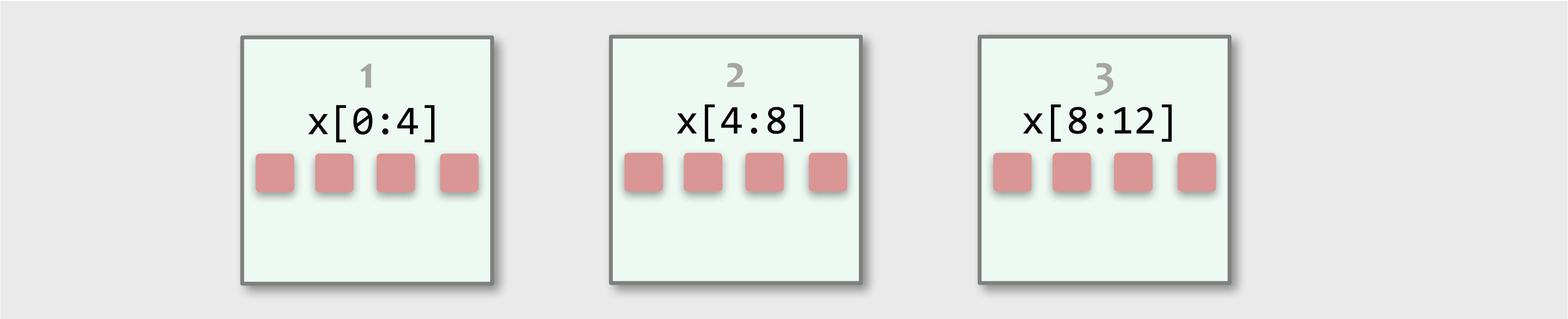
Spatial programming model

```
int[4]@{1,2,3} x[12];
for(i from 1 to 12)
  x[i] += x[i-1];
```

Partition Type

pins data and operators to specific partitions (logical cores)

Similar to [Chandra *et al*. PPoPP'08]



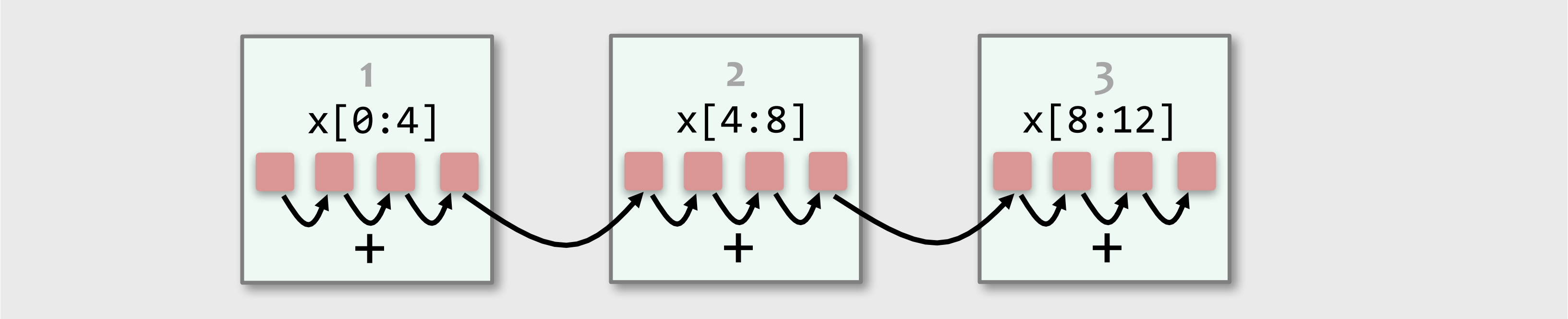
Spatial programming model

```
int[4]@{1,2,3} x[12];
for(i from 1 to 12)
  x[i] +=@loc(x[i]) x[i-1];
```

Partition Type

pins data and operators to specific partitions (logical cores)

Similar to [Chandra *et al.* PPoPP'08]



Incomplete Annotations

```
int[4] x[12];
for(i from 1 to 12)
  x[i] += x[i-1];
```

Incomplete Annotations

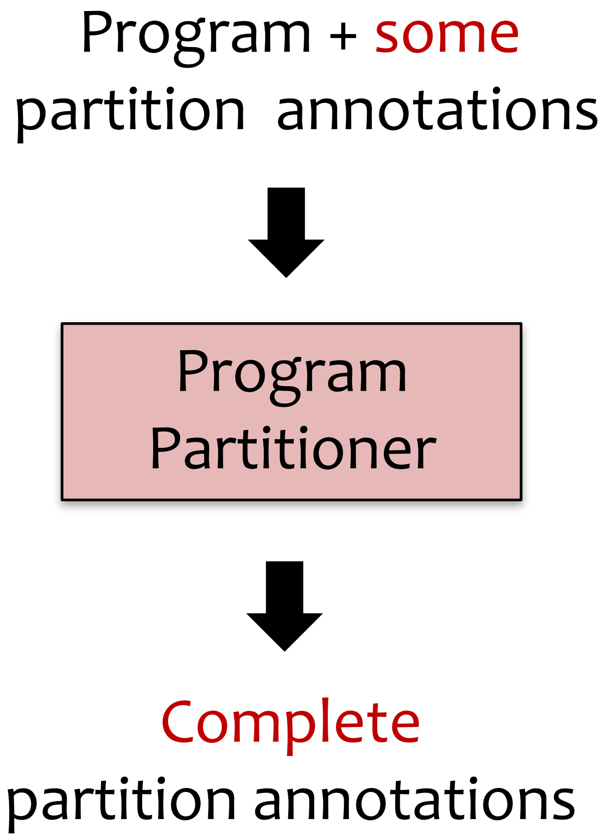
```
int[4]@?? x[12];
for(i from 1 to 12)
  x[i] +=@?? x[i-1];
```

Hard constraint:

Code fits in each logical core (partition).

Objective:

Minimize number of messages sent between partitions.



Program + some partition annotations

Program Partitioner

Complete partition annotations

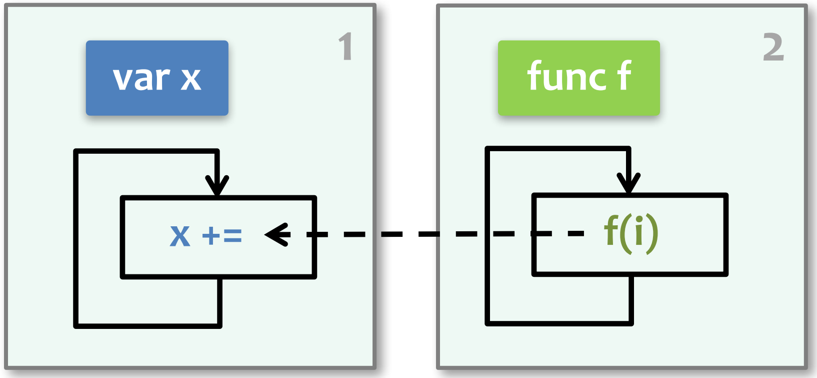
Problems

Problem I	Generated code is too large.
Cause	**Control flow statements partitioning** strategy exploits **code replication** but not enough **communication**.

Control Flow Partitioning

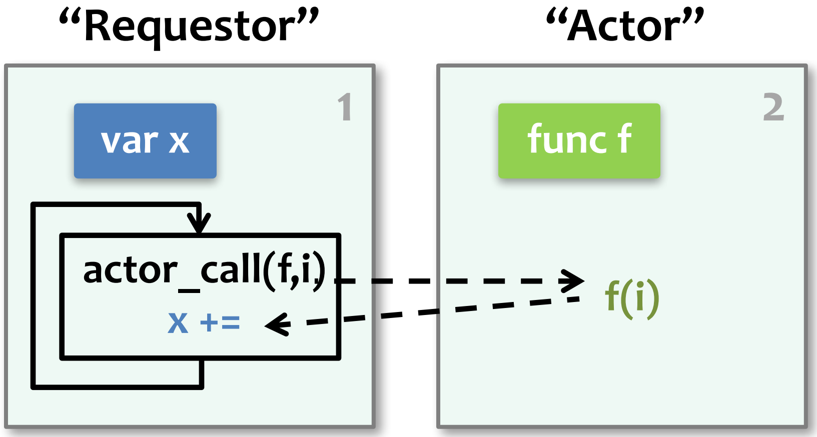
SPMD strategy (original)

```
// source
int@2 f(int@2 i)
{ ... }

int@1 x;
for(i from 0 to 100)
  x += f(i);
```



Replicates relevant control flow constructs onto every partition.

More code
Less communication



Sends a request to execute code to avoid control flow duplication.

Less code
More communication

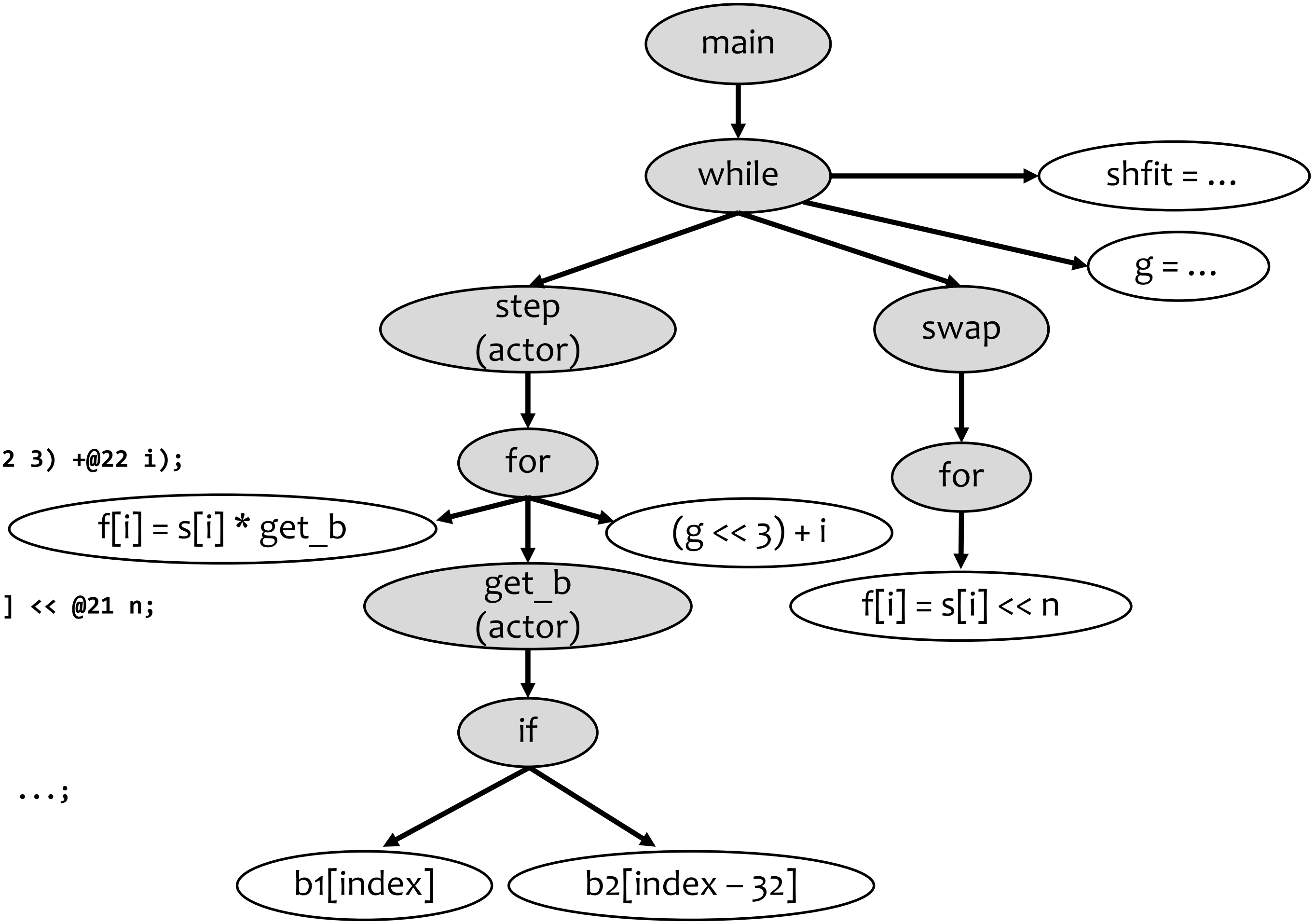
Actor strategy

```
// annotate with
actor f;
```

Compiling with Mixed Strategy

```
fix1_t@21 f[8];     fix1_t@11 s[8];
fix1 t@23 b1[32];   fix1 t@13 b2[32];

actor get_b;
fix1 t@23 get_b(int@23 index) {
  if (index <@23 32)
    return b1[index] ;
  else
    return b2[index -@13 32];
}

actor step;
void step(int@22 g) {
  for (i from 0 to 8)
    f[i] = s[i] *@22 get_b((g <<@22 3) +@22 i);
}

void swap(int@21 n) {
    for (i from 0 to 8) s[i] = f[i] << @21 n;
}

void main() {
  while(1) {
    int@32 g = ...; int@32 shift = ...;
    step(g);
    swap(shift);
  }
}
```

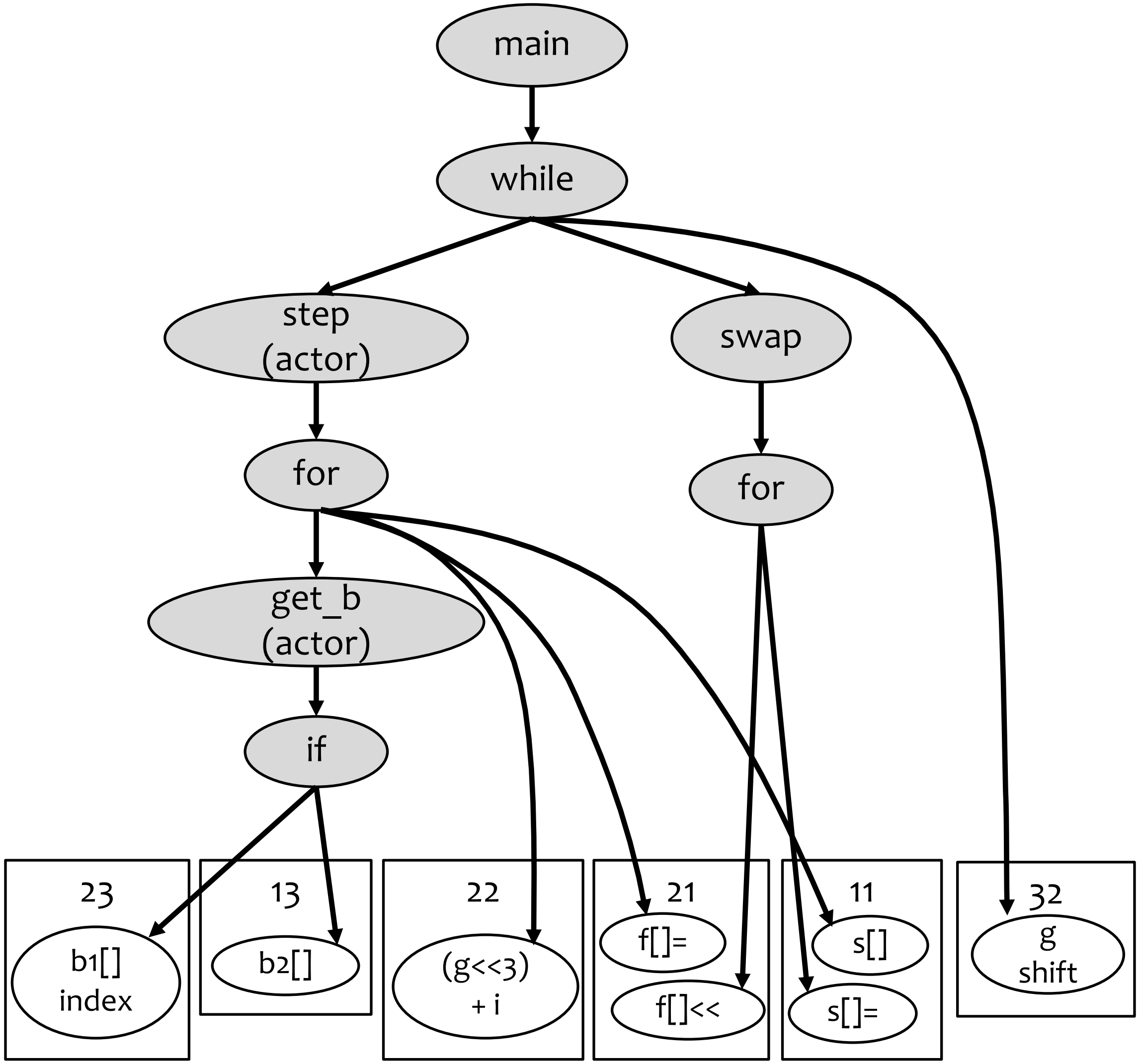
Control Dependence Graph (CDG)



Compiling with Mixed Strategy

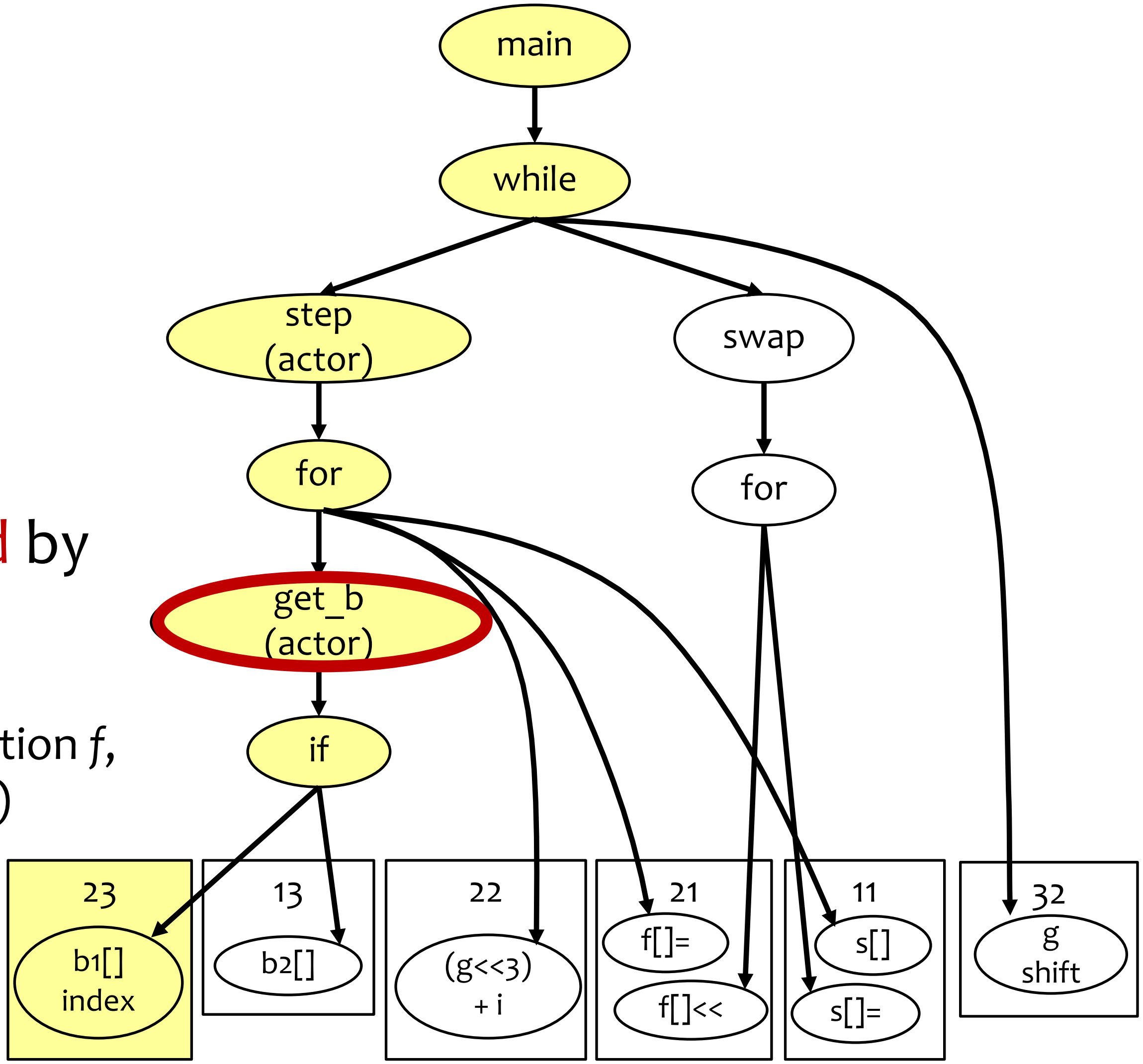
```
fix1_t@21 f[8];     fix1_t@11 s[8];
fix1 t@23 b1[32];   fix1 t@13 b2[32];

actor get_b;
fix1 t@23 get_b(int@23 index) {
  if (index <@23 32)
    return b1[index] ;
  else
    return b2[index -@13 32];
}

actor step;
void step(int@22 g) {
  for (i from 0 to 8)
    f[i] = s[i] *@22 get_b((g <<@22 3) +@22 i);
}

void swap(int@21 n) {
    for (i from 0 to 8) s[i] = f[i] << @21 n;
}

void main() {
  while(1) {
    int@32 g = ...; int@32 shift = ...;
    step(g);
    swap(shift);
  }
}
```

Control Dependence Graph (CDG)



Compiling with Mixed Strategy

Control Dependence Graph (CDG)



Partition 23 is dominated by actor function get_b.

(Partition *p* is dominated by function *f*, if all paths from main to *p* pass *f*.)

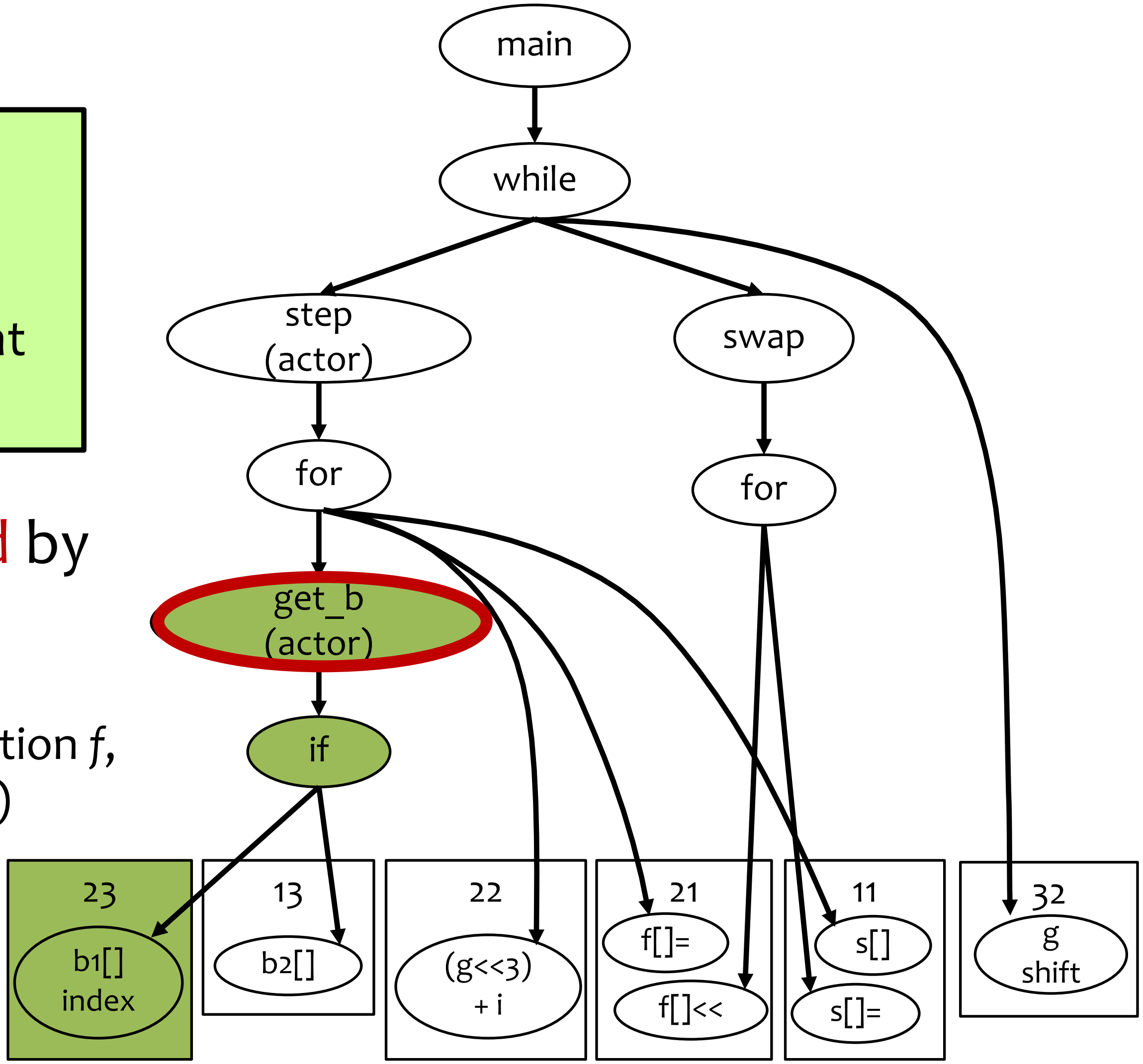
Compiling with Mixed Strategy

Therefore,
- Partition 23 is an actor of **get_b**.
- Control flow of 23 starts at **get_b**.

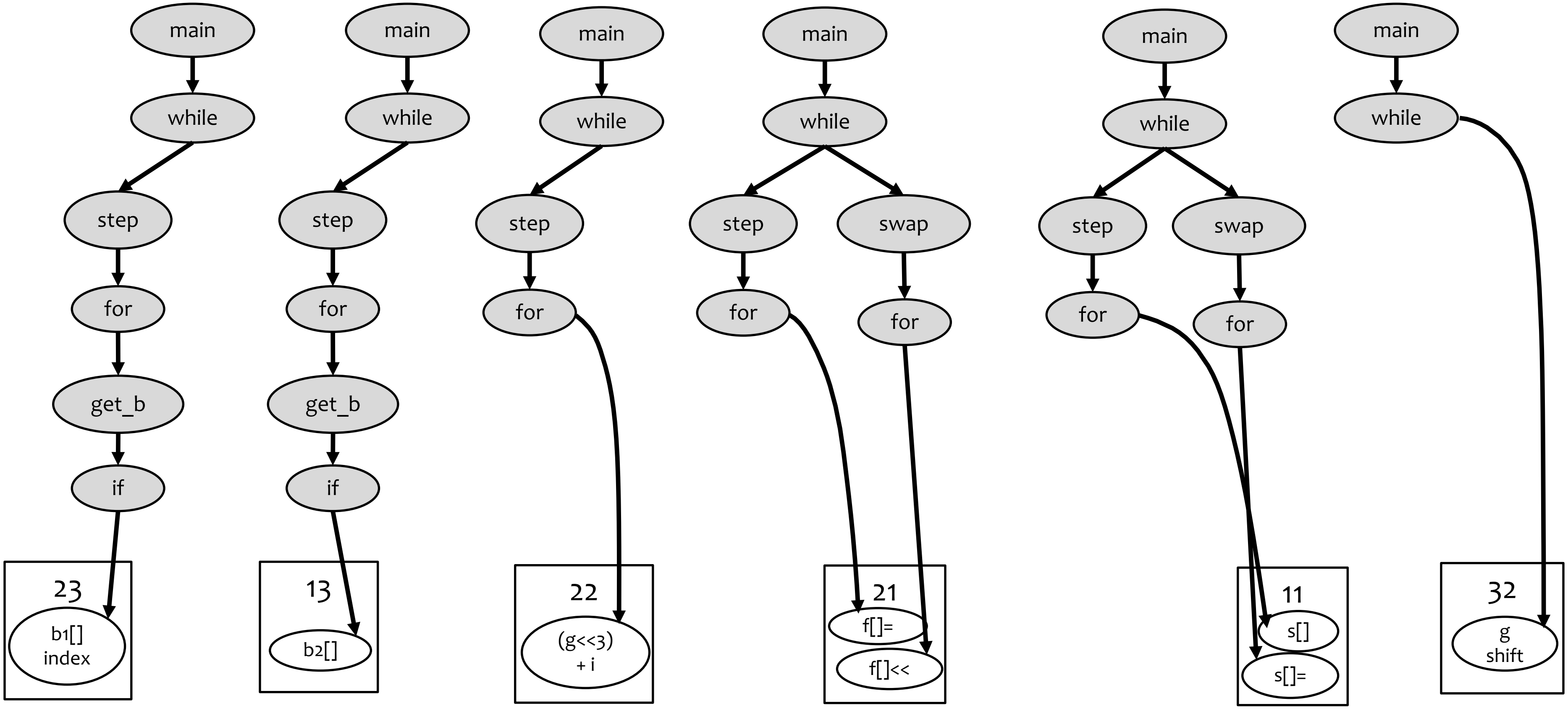
Partition 23 is dominated by actor function **get_b**.

(Partition *p* is dominated by function *f*, if all paths from main to *p* pass *f*.)

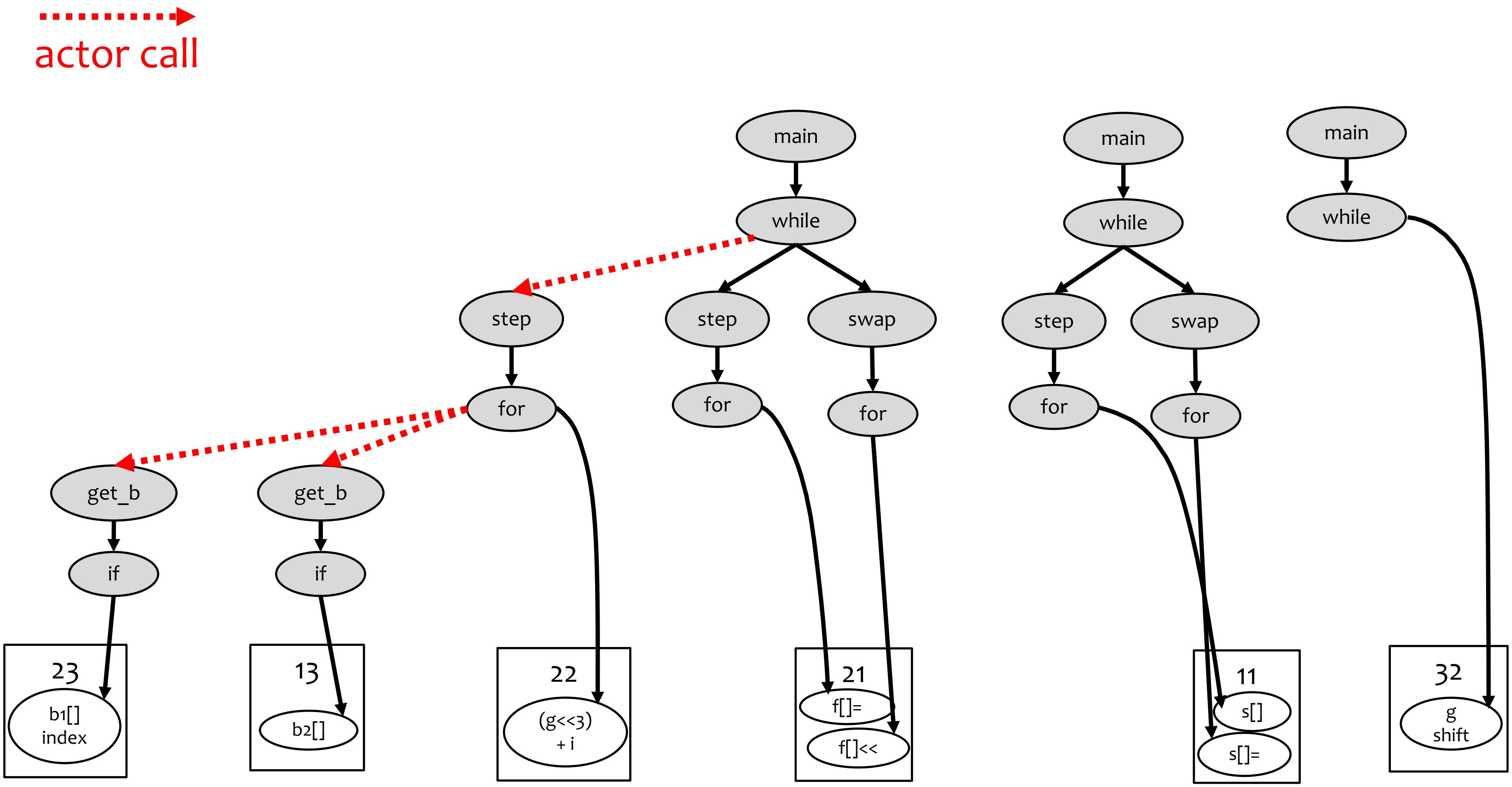
Control Dependence Graph (CDG)



SPMD Strategy



SPMD + Actor Strategy



Problems

Problem I Generated code is too large.

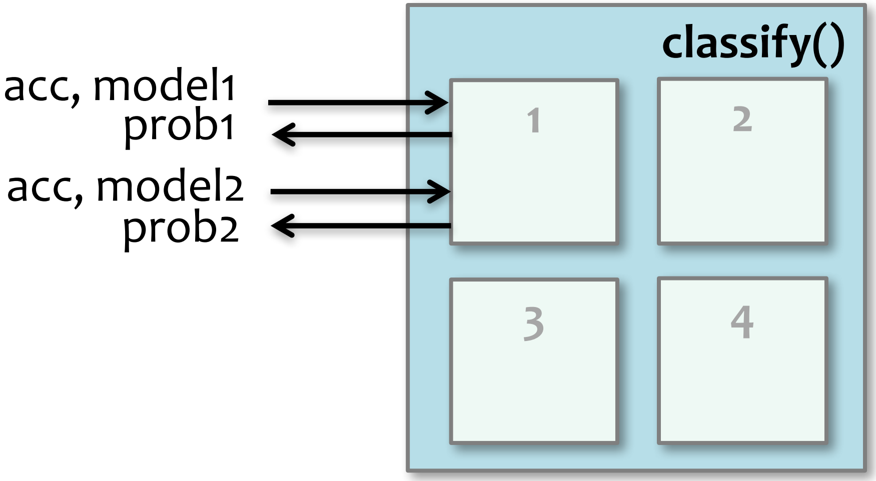
Cause **Control flow statements partitioning** strategy exploits **code replication** but not enough **communication.**

Problem II Slow execution time (no parallelism)

Cause **Data & computations partitioning** strategy does not exploit **code replication**.

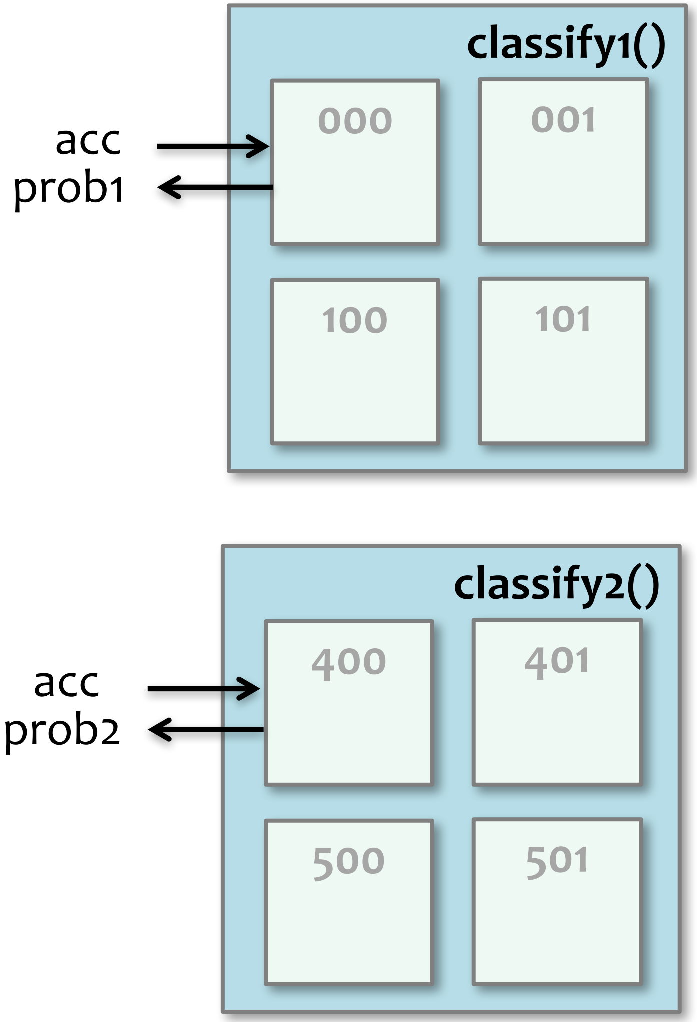
Original: No Task Parallelism

```
fix1_t@1 classify(fix1_t@1 acc[3], fix1_t@2 model[N]) {...}

prob1 = classify(acc, model1);
prob2 = classify(acc, model2);
```



Solution: Automatic Replication

```
// Define module
module Classifier(model_init) {
  fix1_t@1 model[N] = model_init ;
  fix1_t@2 classify(fit1_t@2 acc[3]) {
    ... }
}

// Create module instances
C1 = new Classifier(model1);
C2 = new Classifier(model2);

// Call two different functions
C1.classify(acc);
C2.classify(acc);
```

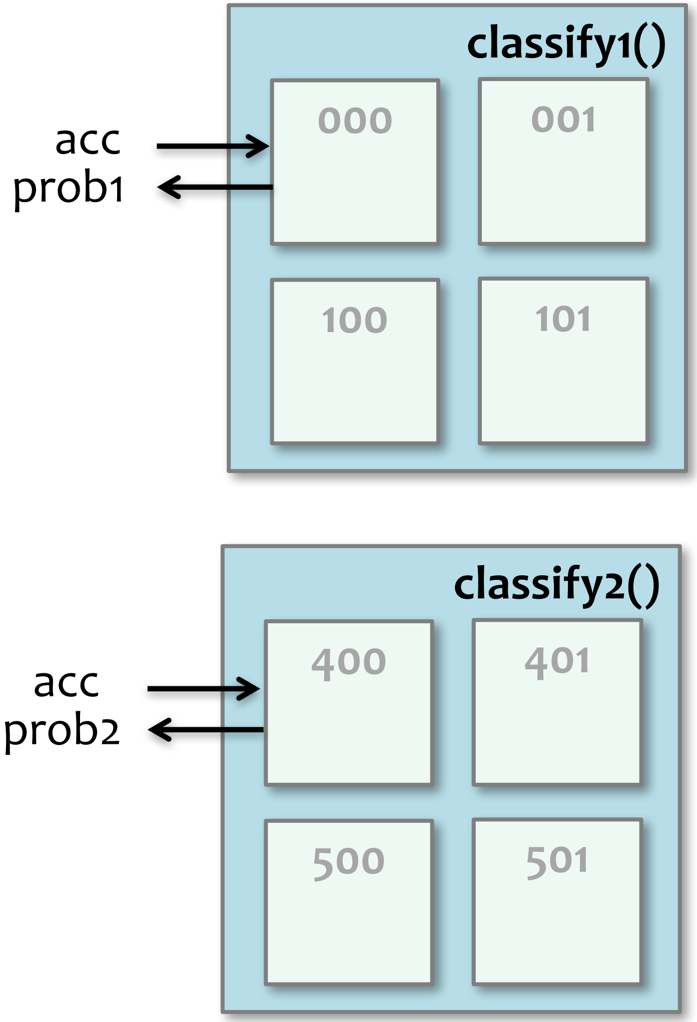


Pinning Partitions to Cores

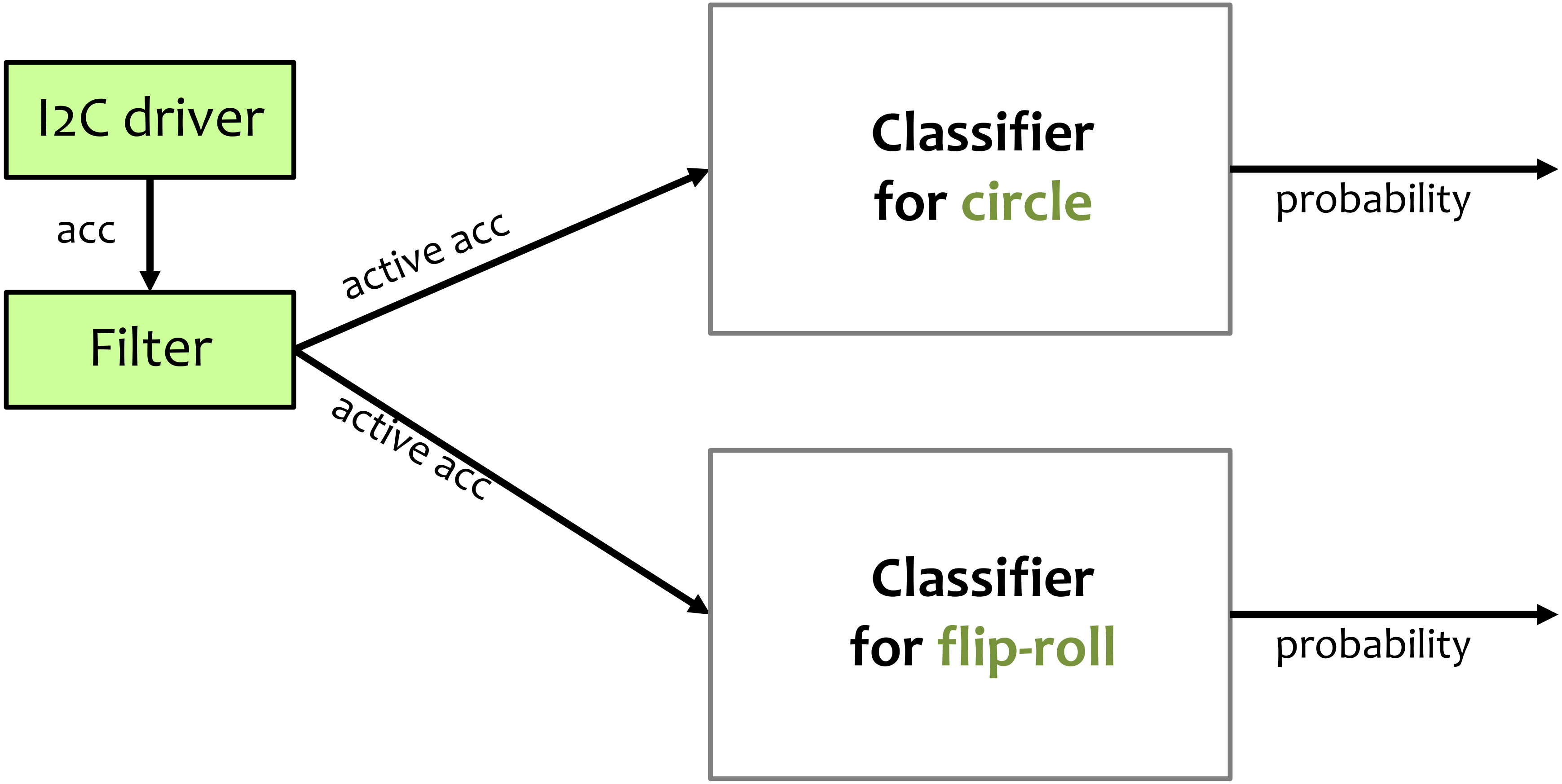
```
// Define module
module Classifier(model_init) {
  # 1 --> 000
  # 2 --> 001
  # 3 --> 100
  # 4 --> 101
  fix1_t@1 model[N] = model_init ;
  fix1_t@2 classify(fit1_t@2 acc[3]) {
    ... }
}

// Create module instances
C1 = new Classifier(model1)@REG(000,101);
C2 = new Classifier(model2)@REG(400,501);

// Call two different functions
C1.classify(acc);
C2.classify(acc);
```

top-left

bottom-right



Hand Gesture Recognition



I2C driver

acc

Filter

active acc

active acc

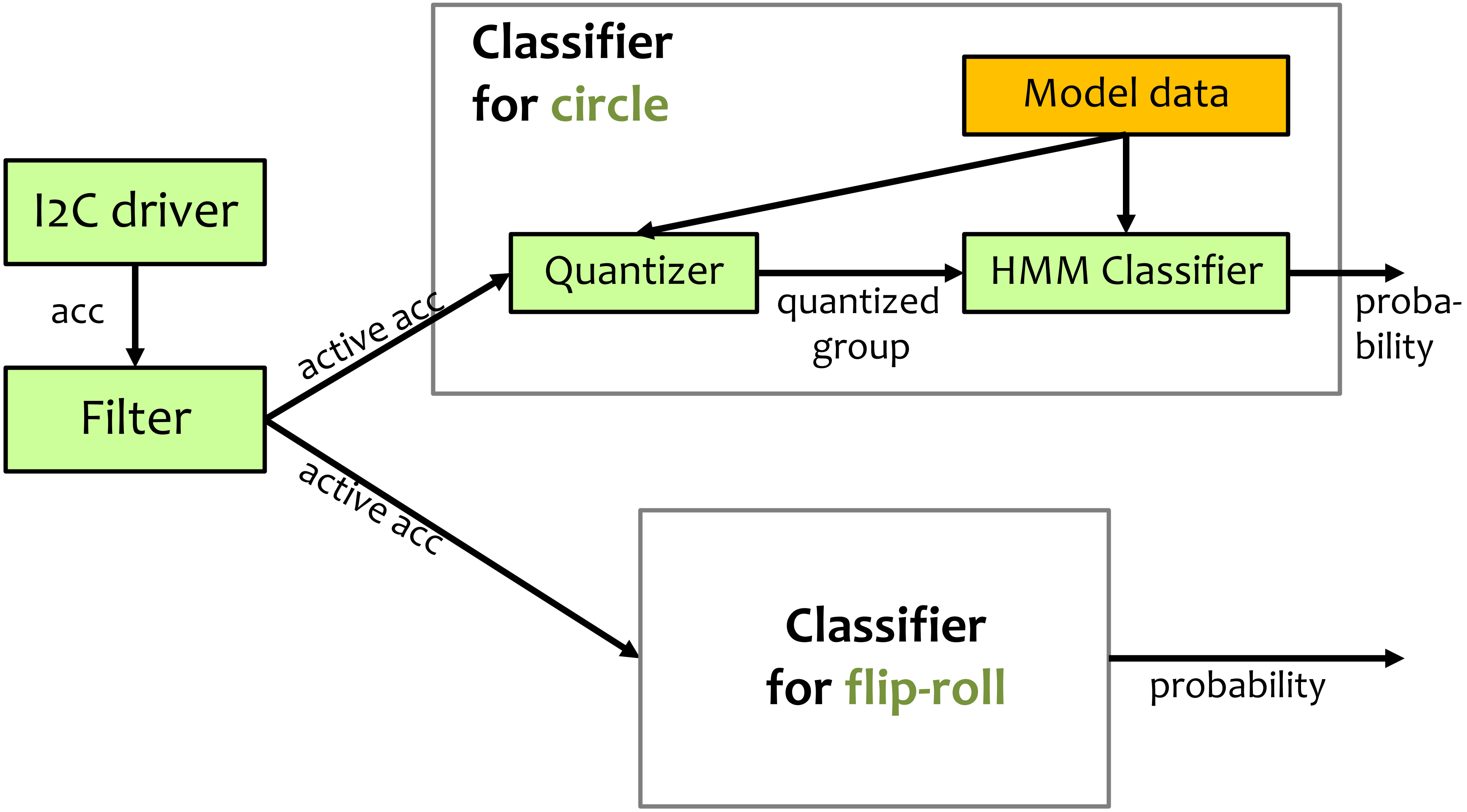
Classifier for circle

Classifier for flip-roll

probability

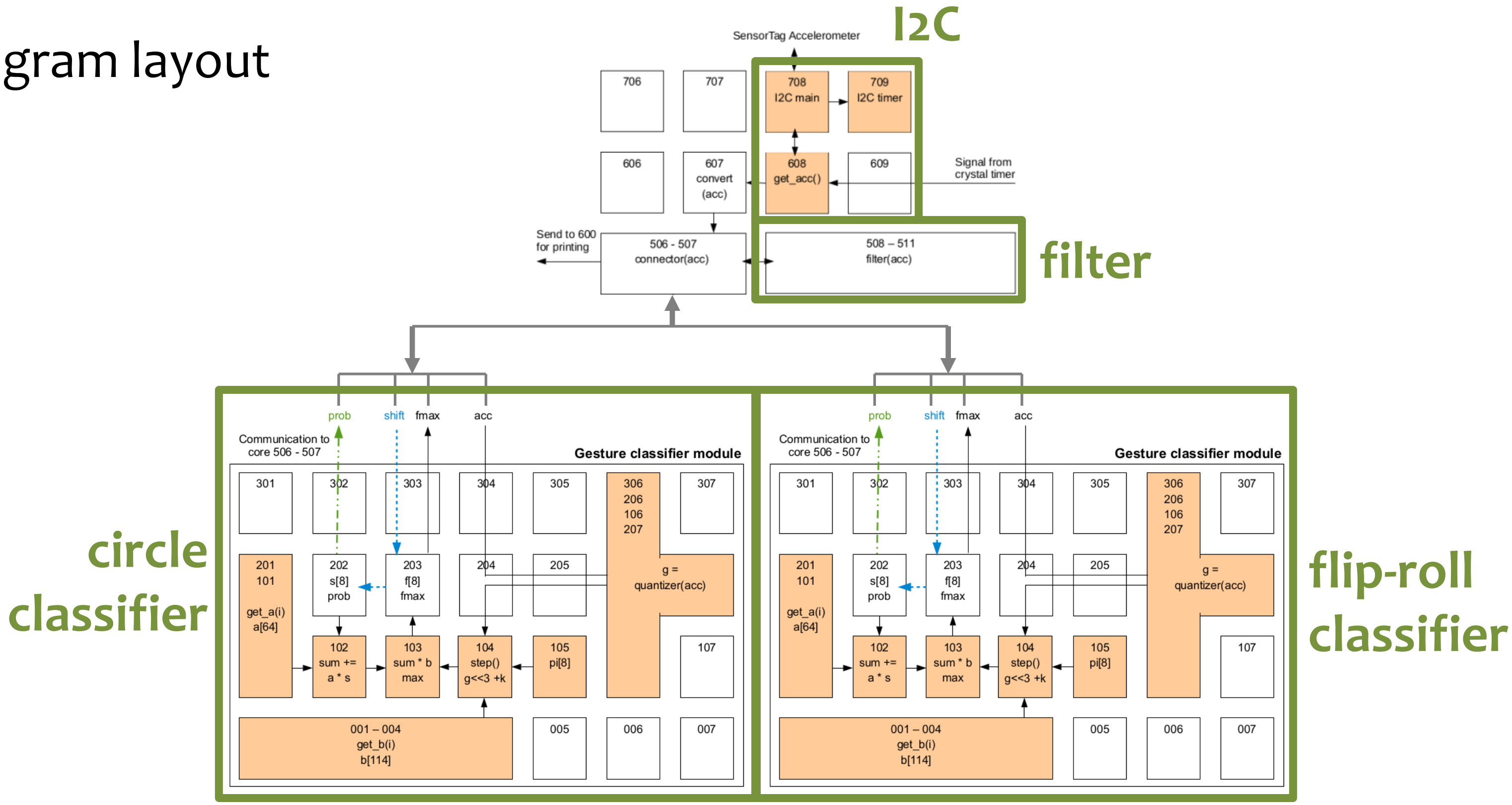
probability

Hand Gesture Recognition



Implementation

Program layout



1. Use **mixed partitioning strategy** to make the application fit on GA144.
   orange = actor cores
2. Use **parallel module** to classify circle and flip-roll gestures in parallel.

Result

Can we use Chlorophyll with our extensions to generate code for the gesture recognition application for GA144?

Partitioning strategy	Number of cores	Overflowed cores	Size of largest core (words)
SPMD	90	12	87
SPMD + Actor	82	0	64

Note: each core can contain up to 64 words.

Code occupies 82 out of 144 cores.

Prediction accuracy = 80-91% (similar to Wigee [Schlomer et al. 08])

GA144 vs. MSP430

How much energy consumption can we reduce by being able to compile for GA144?

Energy comsumption per one round of accelerometer reading
(not including energy consumed when idle).

Processor	Execution time (ms)	Energy consumption (uJ)		
		Accelerometer	Computation	Total
GA144	2.6	1.7	0.6	2.2
MSP430	61.3	0.8	41.2	41.9

↑ 23x faster

↑ 19x more energy-efficient

GA144 vs. MSP430

Average power consumption

Processor	Overall (active + idle)	Idle
GA144 (at 1.8V)	0.436 mA	38 uA
MSP430 (at 2.2V)	0.748 mA	1 uA

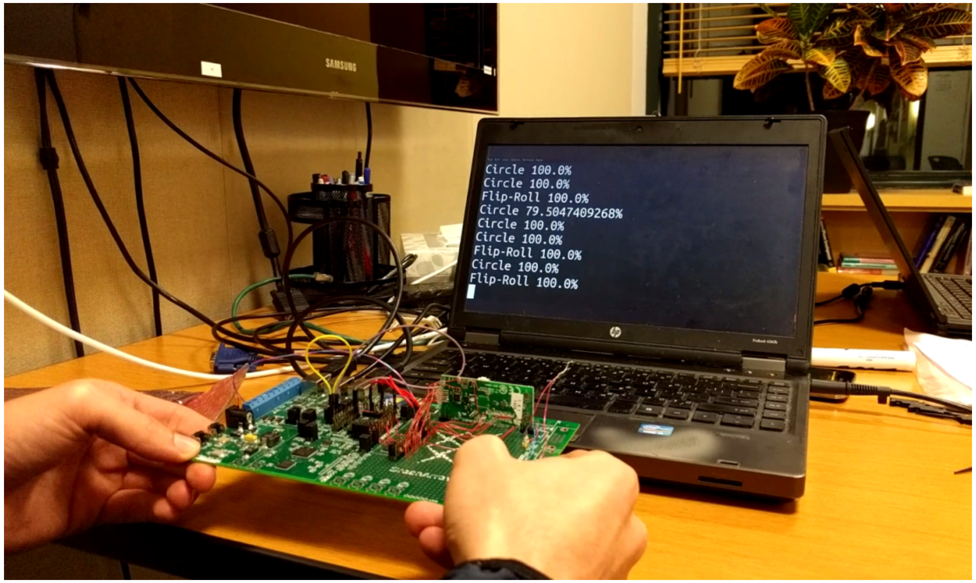
Note that MSP430 performed less computation than GA144:

- GA144 ran 200 iterations per second.
- MSP430 ran 16 iterations per second.
  (MSP430 could not run any faster.)

Limitation

- Automatic program partitioning and layout for complex programs is difficult.
  - It failed to generate a valid solution (code does not fit).
  - This is because it ignores space taken by routing code.
  - Solution: need a better algorithm or
    let programmers help (current strategy).
- Small on-chip memory is troublesome.
  - We can only support two gestures because of the memory limit.
  - Solution: use the combination of native computation + VM

Demo



```
Circle 100.0%
Circle 100.0%
Flip-Roll 100.0%
Circle 79.5047409268%
Circle 100.0%
Circle 100.0%
Flip-Roll 100.0%
Circle 100.0%
Flip-Roll 100.0%
```