Programming Model and Synthesis for Low-power Spatial Architectures

Phitchaya Mangpo Phothilimthana
Nishant Totla

University of California, Berkeley

Architecture's Heterogeneity is Inevitable

Why heterogeneous system/architecture?

- Energy efficiency
- Runtime performance

We want both!

What is the future architecture? Convergence point unclear.
But it will be some combination of

1. **Many small cores** (less control overhead, smaller bitwidth)
2. **Simple interconnect** (reduce communication energy)
3. **New ISAs** (specialized, more compact encoding)

What we are working on

- Programming model for future heterogeneous architectures
- Synthesis-aided "compiler"

Energy Efficiency vs. Programmability

Future architectures		Challenges
Many small cores	→	Fine-grain partitioning
New ISAs	→	New compiler optimizations
Simple interconnect	→	SW-controlled messages

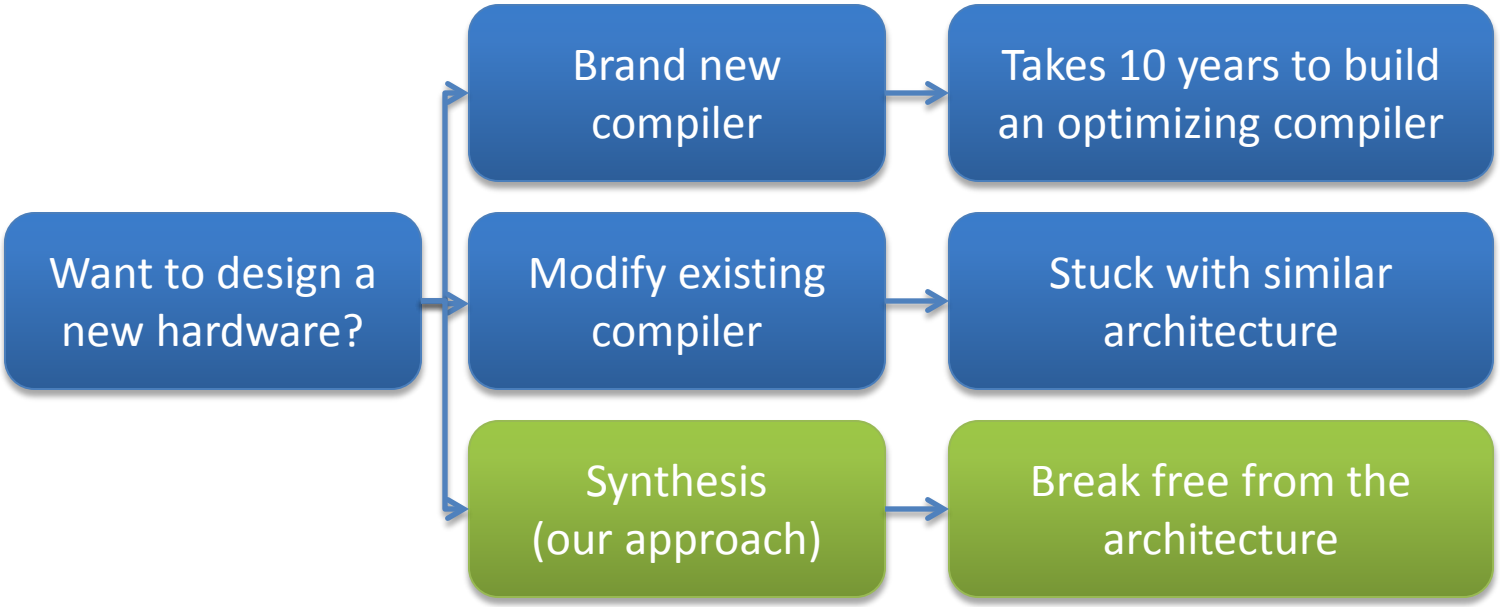
"The biggest challenge with 1000 core chips will be programming them."[1]
- William Dally (NVIDIA, Stanford)

On NVIDIA's 28nm chips, getting data from neighboring memory takes 26x the energy of addition.[1]

Cache hit uses up to 7x energy of addition.[2]

1: http://techtalks.tv/talks/54110/ 2: http://cva.stanford.edu/publications/2010/jbalfour-thesis.pdf

Compilers: State of the Art



If you limit yourself to traditional compilation framework, you limit yourself to similar architectures or wait years to prove the success.

Synthesis, an alternative to compilation

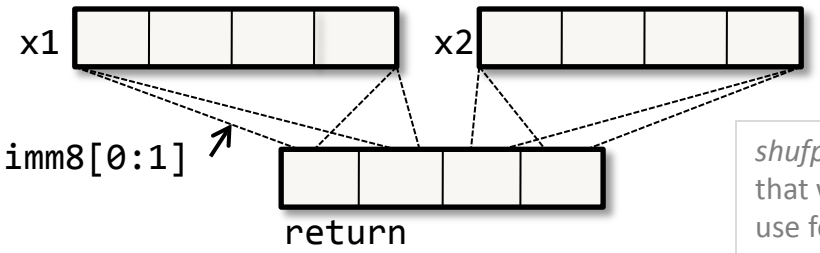
- *Compiler*: transforms the source code
- *Synthesis*: searches for a correct, fast program

Program Synthesis (Example)

Specification:

```
int[16] transpose(int[16] M) {
  int[16] T = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
      T[4 * i + j] = M[4 * j + i];
  return T;
}
```

Specification is a correct and easy-to-implement function.



shufps is an SSE insturction that we might be able to use for speeding up the transpose function

Sketch:

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;
  repeat (??) S[??::4] = shufps(M[??::4], M[??::4], ??);
  repeat (??) T[??::4] = shufps(S[??::4], S[??::4], ??);
  return T;
}
```

Sketch is a skeleton of a faster output program with unknowns (??) which are usually hard to get right.

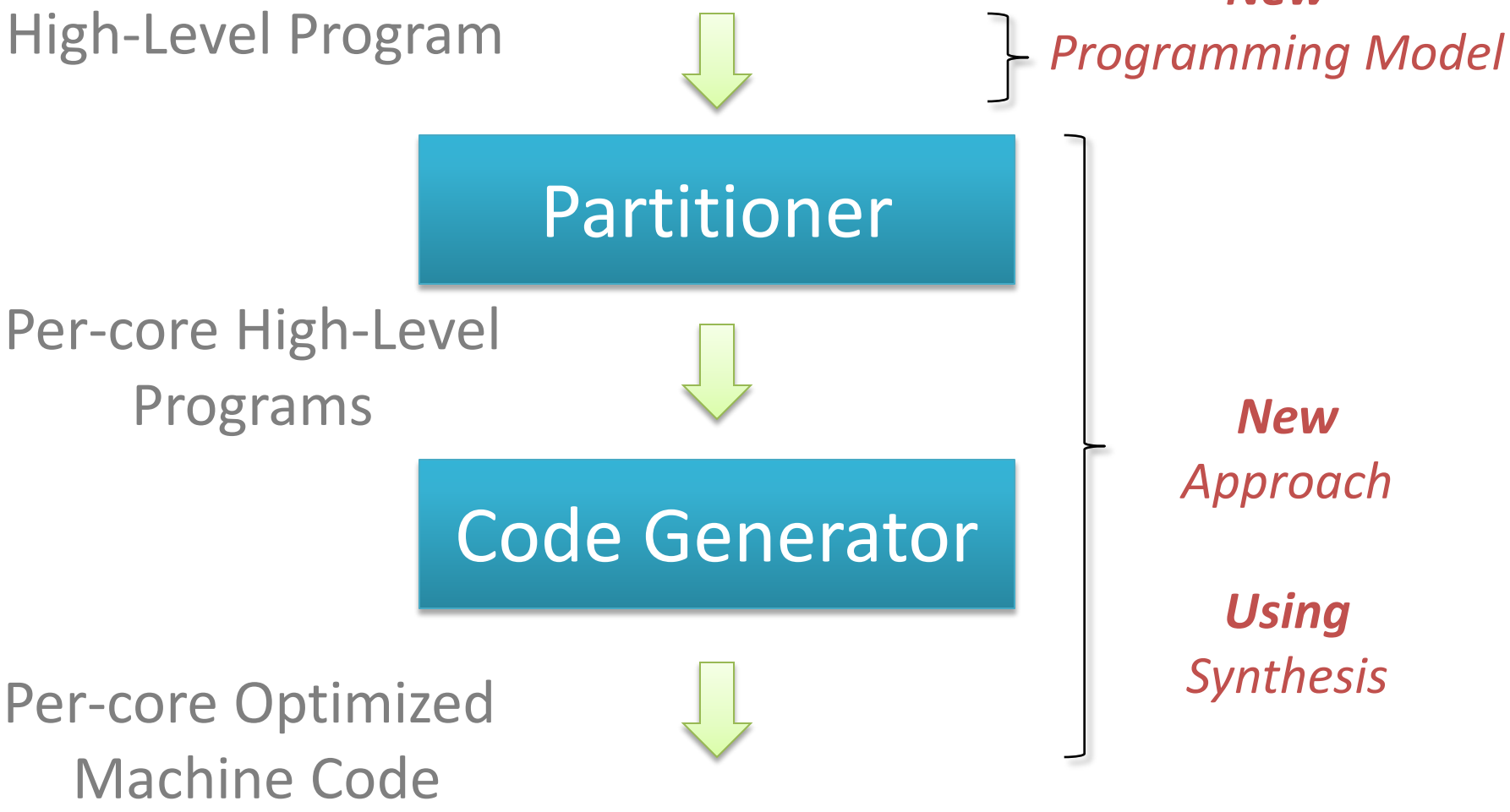
Synthesized program:

```
int[16] trans_sse(int[16] M) implements trans {
  S[4::4]   = shufps(M[6::4],   M[2::4],  11001000b);
  S[0::4]   = shufps(M[11::4],  M[6::4],  10010110b);
  S[12::4]  = shufps(M[0::4],   M[2::4],  10001101b);
  S[8::4]   = shufps(M[8::4],   M[12::4], 11010111b);
  T[4::4]   = shufps(S[11::4],  S[1::4],  10111100b);
  T[12::4]  = shufps(S[3::4],   S[8::4],  11000011b);
  T[8::4]   = shufps(S[4::4],   S[9::4],  11100010b);
  T[0::4]   = shufps(S[12::4],  S[0::4],  10110100b);
  return T;
}
```

Synthesized program is a completion of the sketch that behaves like the specification.

Synthesis time < 10 seconds.
Search space > 10^{70}

Our Plan


High-Level Program
Partitioner
Per-core High-Level Programs
Code Generator
Per-core Optimized Machine Code
New Programming Model
New Approach
Using Synthesis

Case study: GreenArrays Spatial Processors

of Instructions/second vs Power

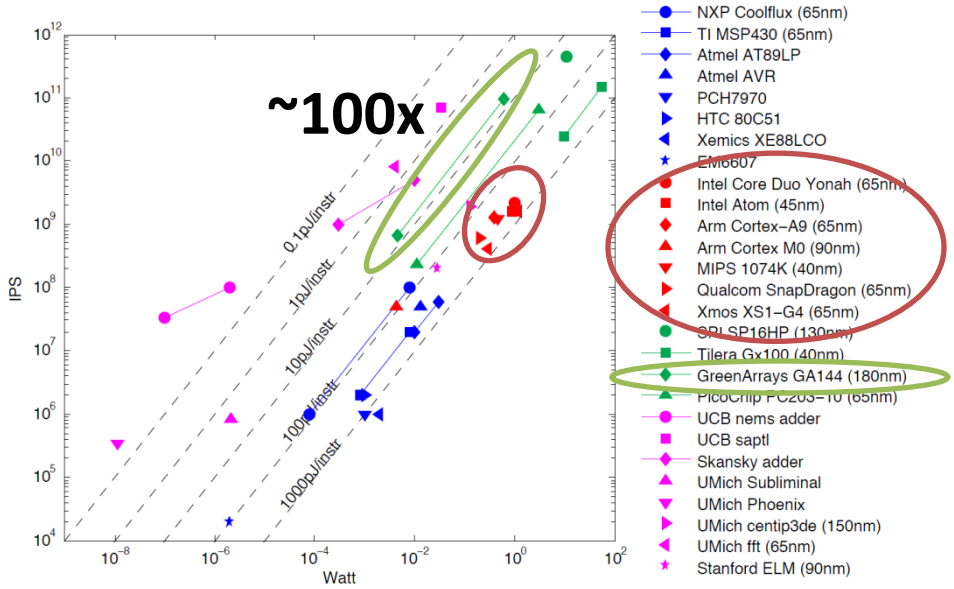


Figure from Per Ljung

Finite Impulse Response Benchmark

GA144 is 11x faster and simultaneously 9x more energy efficient than MSP 430.

Data from Rimas Avizienis

Specs

- Stack-based 18-bit architecture
- 32 instructions
- 8 x 18 array of asynchronous computers (cores)
- No shared resources (i.e. clock, cache, memory). Very scalable architecture.
- Limited communication, neighbors only
- < 300 byte memory per core

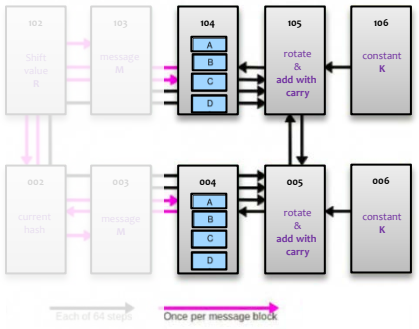
Example challenges of programming spatial architectures like GA144:

- *Bitwidth slicing:* Represent 32-bit numbers by two 18-bit words
- *Function partitioning:* Break functions into a pipeline with just a few operations per core.

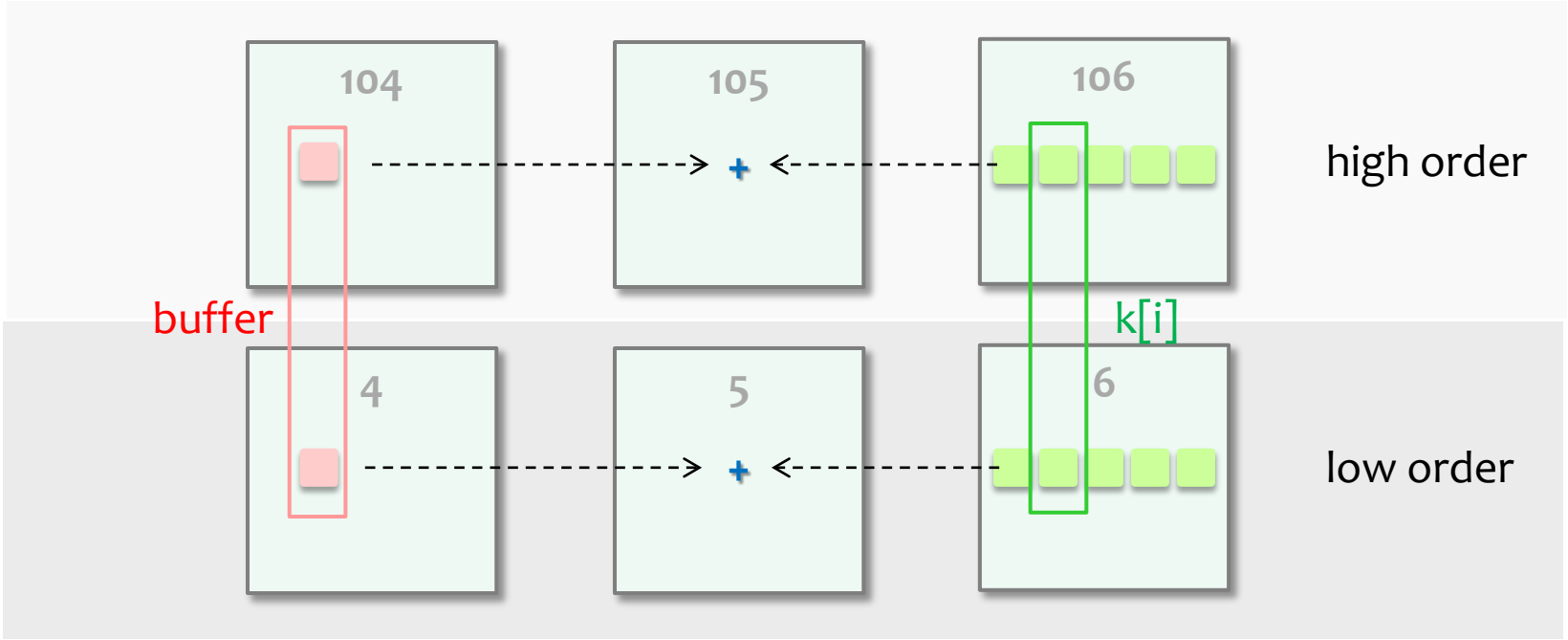
Spatial programming model

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@here sum = buffer +@here k[i] + message[g];
  ...
}
```

MD5 Implementation on GA144 >>
From GreenArrays App Note

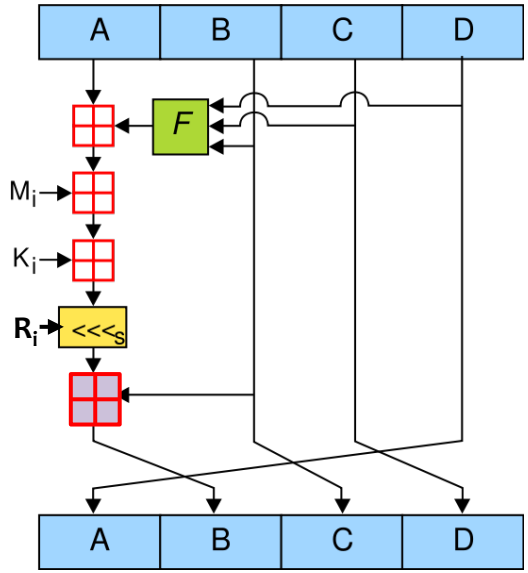



```
buffer is at (104,4)
+      is at (105,5)
k[i]   is at (106,6)
```



Optimal Partitions from Our Synthesizer

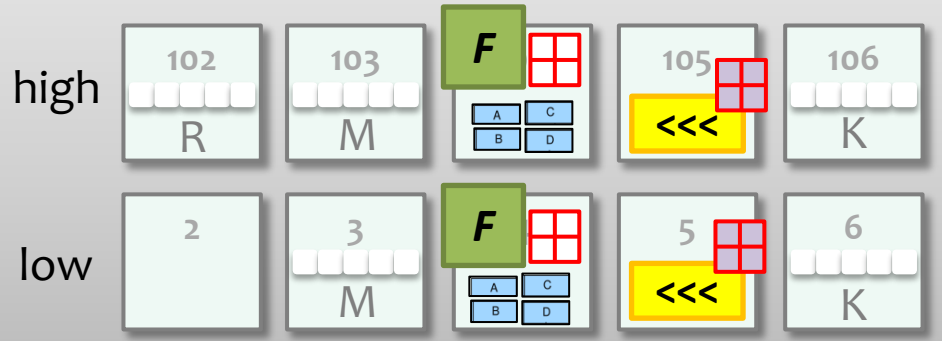
- Benchmark: simplified MD5 (one iteration)
- Partitions are automatically generated.



1

256-byte mem per core

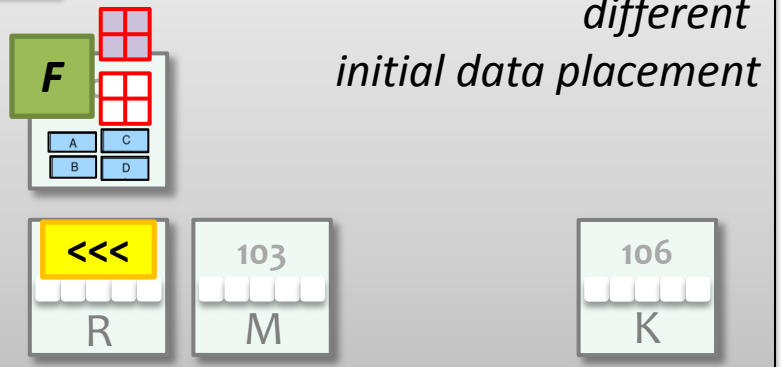
initial data placement specified



3

512-byte mem per core

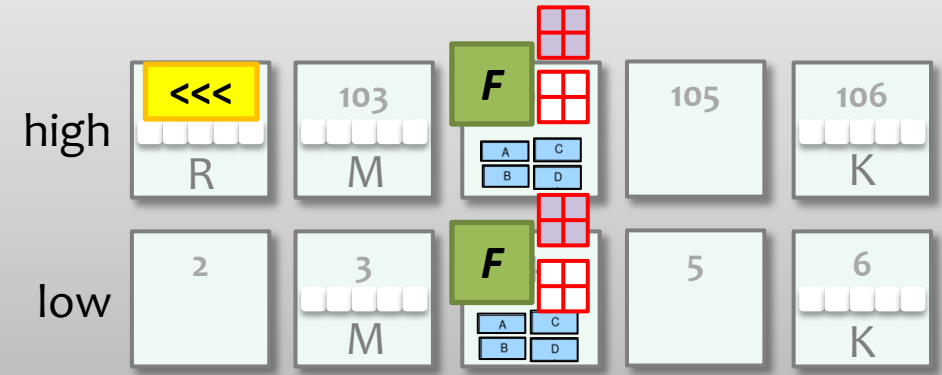
different initial data placement



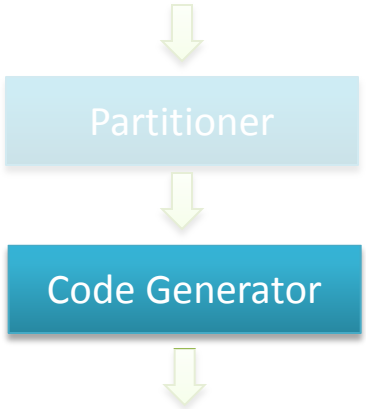
2

512-byte mem per core

same initial data placement



Retargetable Code Generation



Traditional compiler needs many tasks:

implement optimizing transformations,
including hardware-specific code generation
(e.g. register allocation, instruction scheduling)

Synthesis-based code translation needs only these:

- define space of programs to search, via code templates
- define machine instructions, as if writing an interpreter

Example: define exclusive-or for a stack architecture

```
xor = lambda: push(pop() ^ pop())
```

Synthesizer can generate code from

- a template with holes as in transpose example --> *sketching*
- an unconstrained template --> *superoptimization*

Code Generation via Superoptimization

Current prototype synthesizes a program with

8 unknown instructions	in	2 to 30 seconds
~25 unknown instructions	within	5 hours

Synthesized functions are

1.7x – 5.2x faster and 1.8x – 4x shorter than
naïve implementation of simple GreenArrays functions

1.1x-1.4x faster and 1.5x shorter than optimized hand-written
GreenArrays functions by **experts** (MD5 App Note)

Synthesize efficient division by constant

quotient = (?? * n) >> ??

Program	Solution
x/3	(43691 * x) >> 17
x/5	(52429* x) >> 18
x/6	(43691 * x) >> 18
x/7	(149797 * x) >> 20

Demo and Future

Current Status

- Partitioner for straight-line code
- Superoptimizer for smaller code

Future Work

- Make synthesizer retargetable
- Release it!
- Design spatial data structures
- Build low-power gadgets for audio, vision, health, …

We will answer

"how minimal can hardware be?"

"how to build tools for it?"