Chlorophyll

Synthesis-Aided Compiler for Low-Power Spatial Architectures

Phitchaya Mangpo Phothilimthana
Building Compilers is Hard

- Brand new traditional compiler: Takes 10 years to build
- Synthesis-aided compiler: Low-effort & good performance

Why important?

- spatial & temporal partitioning
- small memory
- narrow bitwidth
- crazy ISA

analyze + transform
search for optimal program
Case study: GreenArrays Spatial Processor

On FIR

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.
int k[64];

int sumrotate(int buffer, ...) {
    int sum = buffer + k[i] + message[g];
    ...
}

int@{[0:64]=6} k[64];

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

int@[0:64]=6 k[64];

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

Approach Overview

1. **HL Program**
   - HLP with partition annotation (logical core)

2. **Partitioner**
   - HLP with location annotation & routing info (physical cores)

3. **Layout**
   - Per-core HLPs with communication code

4. **Code Separator**
   - Per-core optimized machine code
Partitioning Synthesizer

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

256-byte mem per core
initial data placement specified

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>2</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

512-byte mem per core
same initial data placement

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;&lt;</td>
<td>2</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;&lt;</td>
<td>5</td>
</tr>
<tr>
<td>105</td>
<td>5</td>
</tr>
<tr>
<td>&lt;&lt;&lt;</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>6</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>
Approach Overview

HL Program → Partitioner → Layout → Code Separator → Code Generator

- Partitioner
  - HLP with partition annotation (logical core)
  - HLP with location annotation & routing info (physical cores)
  - constraint solving
  - off-the-shelf simulated annealing
  - simple transformation
  - superoptimization

- Code Generator
  - Per-core HLPs with communication code
  - Per-core optimized machine code
Modular Superoptimizer

- Synthesize the entire program is not scalable.
- Start-of-the-art synthesizer can generate
  - up to 25 instructions using cluster [Schkufza et al.]
  - up to 51 instructions using cluster with pre-synthesis heuristic [ICFP winner]
- Need decomposition.
Modular Superoptimizer

Naïve Code Gen

loop

if

Superoptimizer

Minimize Running Time

CEGIS

block A

block A'

sliding window
Results

Synthesis vs. Heuristic

Partitioning synthesizer vs. heuristic partitioner
• 5% faster with synthesizing partitioner

Superoptimization vs. no superoptimization
• 15% faster with superoptimization

MD5 Benchmarks (Chlorophyll vs. Expert)

On entire program, Chlorophyll was
• 32% slower and 50% less energy-efficient

On 3 critical functions, Chlorophyll found
• approximately 1.3x – 2.7x shorter and faster code