A SEJITS Taxonomy & Implementation for Python

Shoaib Kamil
Armando Fox, Katherine Yelick,
Bryan Catanzaro, Krste Asanovic, Jim Demmel
UPCRC Retreat Summer 2010
What is a Specializer

- Captures a pattern of serial or parallel control/computation
  - may be specific to data structure (graph, array, tree)
  - or exist independently (map)
- Input
  - size of data or other data parameters
  - one or more functions to apply
  - in degenerate case, takes in just parameters
- Specializer engine: Combines pattern-specific intelligence with codegen
- Output: Compiled generated code that has been specialized and optimized
Captures a *higher order pattern*

- ie, pattern “parametrized” by 1 or more *functions*

**Input:** pattern + function(s)

- Class captures pattern
- class’s method(s) take function(s) as arguments

**Specializer engine generates OpenMP stencil code that applies function at each pt**

**Allows pattern-specific optimizations**

- e.g. time skewing, transformation to red-black
Example pattern-specific compiler: Structured grid in Ruby

- Ruby class encapsulates SG pattern
  - body of anonymous lambda specifies filter function
- Code generator produces OpenMP for multicore x86
  - ~1000-2000x faster than Ruby
  - Minimal per-call runtime overhead

```ruby
class LaplacianKernel < Kernel
  def kernel(in_grid, out_grid)
    in_grid.each_interior do |point|
      in_grid.neighbors(point, 1).each do |x|
        out_grid[point] += 0.2 * x.val
      end
    end
  end
end
```

```c
VALUE kern_par(int argc, VALUE* argv, VALUE self) {
  unpack_arrays into in_grid and out_grid;

  #pragma omp parallel for default(shared)
  private (t_6, t_7, t_8)
  for (t_8=1; t_8<256-1; t_8++) {
    for (t_7=1; t_7<256-1; t_7++) {
      for (t_6=1; t_6<256-1; t_6++) {
        int center = INDEX(t_6, t_7, t_8);
        out_grid[center] = (out_grid[center] + (0.2 * in_grid[INDEX(t_6-1, t_7, t_8)]));
        ...;
        out_grid[center] = (out_grid[center] + (0.2 * in_grid[INDEX(t_6, t_7, t_8+1)]));
      }
    }
  }

  return Qtrue;
}
```
Ruby class encapsulates SG pattern
- body of anonymous lambda specifies filter function

Code generator produces OpenMP for multicore x86
- ~1000-2000x faster than Ruby
- Minimal per-call runtime overhead

90% of pure C performance without auto-tuning

```ruby
class LaplacianKernel < Kernel
  def kernel(in_grid, out_grid)
    in_grid.each_interior do |point|
      in_grid.neighbors(point, 1).each do |x|
        out_grid[point] += 0.2*x.val
      end
    end
  end
end
```

```c
VALUE kern_par(int argc, VALUE* argv, VALUE self) {
    unpack_arrays into in_grid and out_grid;

    #pragma omp parallel for default(shared)
    private (t_6,t_7,t_8)
    for (t_8=1; t_8<256-1; t_8++) {
        for (t_7=1; t_7<256-1; t_7++) {
            for (t_6=1; t_6<256-1; t_6++) {
                int center = INDEX(t_6,t_7,t_8);
                out_grid[center] = (out_grid[center] +(0.2*in_grid[INDEX(t_6-1,t_7,t_8)]));
                ...
                out_grid[center] = (out_grid[center] +(0.2*in_grid[INDEX(t_6,t_7,t_8+1)]));
            }
        }
    }
    return Qtrue;
}
```
Example pattern-specific compiler: Structured grid in Ruby

- AST is transformed, analyzed, and codegen’d
- Ruby iterator AST nodes become for loops
  - simple analysis to determine ghost zones
  - innermost loop is unrolled
  - analysis is pattern-specific

```ruby
class LaplacianKernel < Kernel
  def kernel(in_grid, out_grid)
    in_grid.each_interior do |point|
      in_grid.neighbors(point, 1).each do |x|
        out_grid[point] += 0.2 * x.val
      end
    end
  end
end
```

```ruby
VALUE kern_par(int argc, VALUE* argv, VALUE self) {
  unpack_arrays into in_grid and out_grid;
  #pragma omp parallel for default(shared)
  private (t_6, t_7, t_8)
  for (t_8=1; t_8<256-1; t_8++) {
    for (t_7=1; t_7<256-1; t_7++) {
      for (t_6=1; t_6<256-1; t_6++) {
        int center = INDEX(t_6, t_7, t_8);
        out_grid[center] = (out_grid[center] +
          (0.2 * in_grid[INDEX(t_6-1, t_7, t_8)]));
        ...}
      out_grid[center] = (out_grid[center] +
        (0.2 * in_grid[INDEX(t_6, t_7, t_8+1)]));
    }
  }
  return Qtrue;
}
```
A SEJITS Taxonomy

- 5 “Levels” of SEJITS: SEJITS-0 to SEJITS-4
- 2 “interfaces”: function-level decorators or class-encapsulated specializers
- Show scope and range of kinds of codes implementing aspects of SEJITS
- SEJITS-0 is traditional library

OpenCV blas.py

SEJITS-0  SEJITS-1  SEJITS-2  SEJITS-3  SEJITS-4
A SEJITS Taxonomy

- **SEJITS-1**: Efficiency lang code statically precompiled w/ variant selection in efficiency lang code
- **SEJITS-2**: Efficiency lang code statically precompiled w/ variant selection in productivity lang code

SciPy examines array sizes/strides and tries to do a better job than FFTW’s variant selection
A SEJITS Taxonomy

- **SEJITS-3**: Pattern-specific efficiency language source generated by translating productivity language source and JIT-compiled.
A SEJITS Taxonomy

- **SEJITS-4**: Multiple pattern-specific efficiency lang variants generated from productivity lang source, “runtime” empirical planning of which variant to use
A Python Implementation of SEJITS: Asp

- Productivity Language Code
- Domain-Specific Specializer Classes
- Shared Infrastructure
  - Custom Domain-Specific Analysis & Code Generation
  - AST Representation
  - Tree Transformation
  - Basic Codegen
  - Caching

- Efficiency Language Code
  - Lithe
  - Nexus (Cloud)
  - TBB
  - Etc
Simple Specializers to Test Infrastructure

- **Parallel Map**
  - Apply a C function to each item in a collection
  ```ruby
  abc = ParMap.new(…)
  arr = […]
  abc.map(arr, 'my_sofile#my_func')
  ```

- **BoxBlur**
  - Fast blur stencil for images
  - Generates code for specific radius
  - A (very limited) port of Ruby Stencil specializer
Conclusion

- SEJITS uses high-level language features to bring performance to productivity languages
- A variety of levels to match needs
  - depending on capabilities needed
- Working on a OO-based set of specializers to implement SEJITS in Python
  - Provide SDK for specializer writers
- Goal: Democratization of specializer creation