SEJITS: Productive Performance with Pattern-Specific Compilation

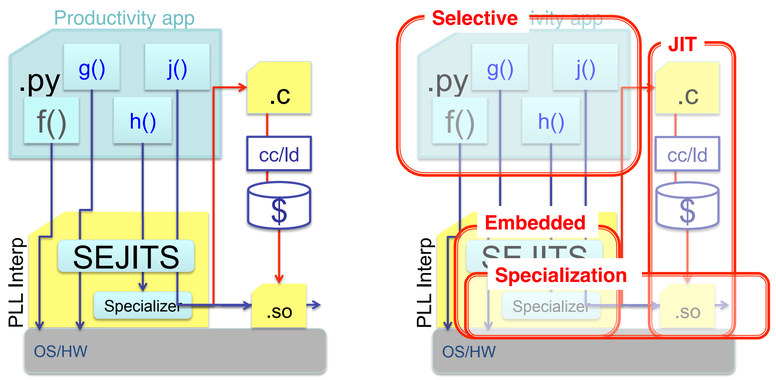
Shoaib Kamil, Armando Fox, Bryan Catanzaro, Katherine Yelick

EECS Electrical Engineering and Computer Sciences

Berkeley Par Lab

PARALLEL COMPUTING LABORATORY

SEJITS Overview



- SEJITS is a methodology for using high-level language capabilities to bring high performance to productivity programmers
- more…

Four Main Ideas

1. Specializer == pattern-specific compiler
   - exploit pattern-specific strategies that may not generalize
   - target specific hardware per pattern
2. Can happen at runtime
3. Productivity language program always valid even without SEJITS support
   - i.e. vs. incompatibly extending syntax; inspired by DSEL vs. DSL argument
4. Specializers can be written in Productivity Language

Producing an Answer vs. Producing Software

- SEJITS delivers adaptive parallel software
  - SEJITS is a highly productive way to produce exactly the code variants you need
- SEJITS makes research code productive
  - Exploit full libraries, tools, etc. of Productivity Lang
  - Performance competitive with Efficiency Lang code
  - Develop specializers to target new HW features => Test designs with real apps

A SEJITS Taxonomy

	Variant Selection	Code Generation
SEJITS-0	None	Statically precompiled
SEJITS-1	In Efficiency Language Library	Statically precompiled
SEJITS-2	In Productivity Language Library	Statically precompiled
SEJITS-3	Single variant generated	Efficiency Language code generated from Productivity Language code
SEJITS-4	In Productivity Language Library	Variants in Efficiency Language code generated from Productivity Language code

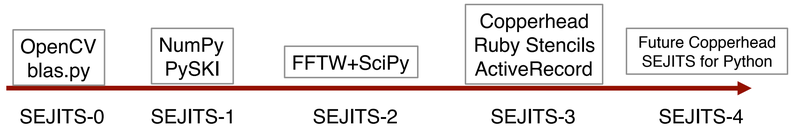
SEJITS-0: Efficiency Lang library exposed to Python

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

SEJITS-3: Efficiency Lang source generated by translating Productivity Lang source and JIT-compiled

SEJITS-4: Multiple Efficiency Lang variants generated from Productivity Lang source, “runtime” empirical planning of which variant to use



Example: Stencils for Ruby

- Ruby class encapsulates SG pattern
  - body of anonymous lambda specifies filter function
- Introspection used to read AST of function body
- Code generator produces OpenMP for multicore x86
  - ~1000-2000x faster than Ruby
  - Minimal per-call runtime overhead
- 90% of pure C performance

```
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
```

```
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;}
```

Two Approaches for Explicit Annotation

- Productivity programmers must specify which code matches specializable patterns

1. Annotate functions that fit into wide, shallow pattern
2. Encapsulate each pattern into its own OO class

Wide Patterns: Data Parallel SEJITS with Copperhead

- Copperhead is a SEJITS Framework for Data Parallelism, embedded in Python
-
- Built on data parallel prelude: `map, reduce, scan, sort, scatter, gather,` …

- Currently has a specializer for CUDA

- See Bryan Catanzaro’s poster for details & demo

```
@cu
def saxpy(a, x, y):
   return [a*xi + yi for xi, yi in zip(x, y)]
```

Narrow Patterns: SEJITS with Python Classes Per-Pattern

- A collection of narrow specializers, each implemented as a Python class

- Shared infrastructure through inheritance
  - AST manipulation, Code Generation, Caching, etc.

- Work in Progress
  - SDK for Specializer Writers
  - Basic Specializers to test infrastructure
  - Parallel Map

- Goal: Democratization of Specializer Creation