EECS
Electrical Engineering and Computer Sciences

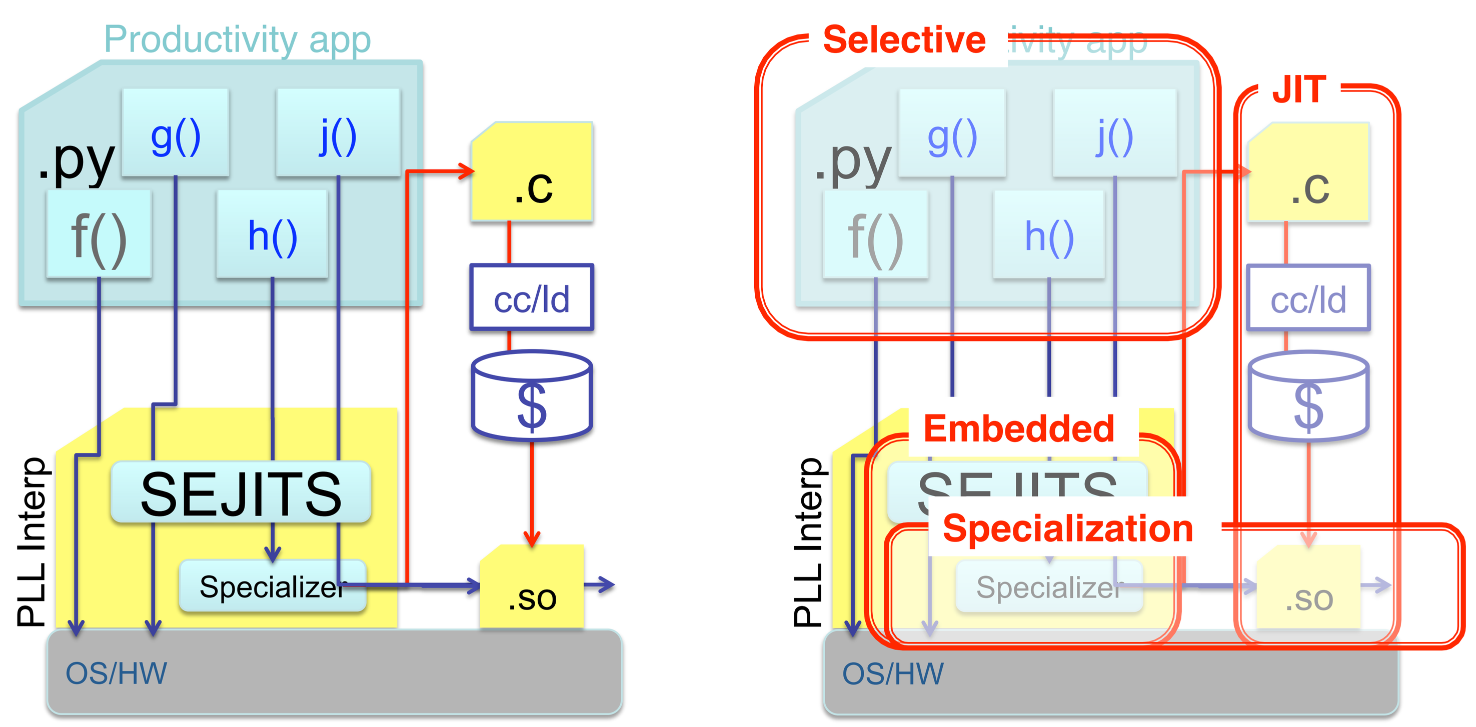
ASP: A SEJITS Implementation for Python
Status, Lessons & Future Plans

Shoaib Kamil, Armando Fox, Katherine Yelick, and many others

BERKELEY PAR LAB

PARALLEL COMPUTING LABORATORY

SEJITS Overview



Specializer == pattern-specific JIT compiler

- *Code templates* hand-authored by efficiency programmers in efficiency language (eg C++)
- *AST transformation* of VHLL code to instantiate templates
- Compile & run specialized code, return results to PLL
- Occurs invisibly to programmer

ASP: A SEJITS Implementation for Python

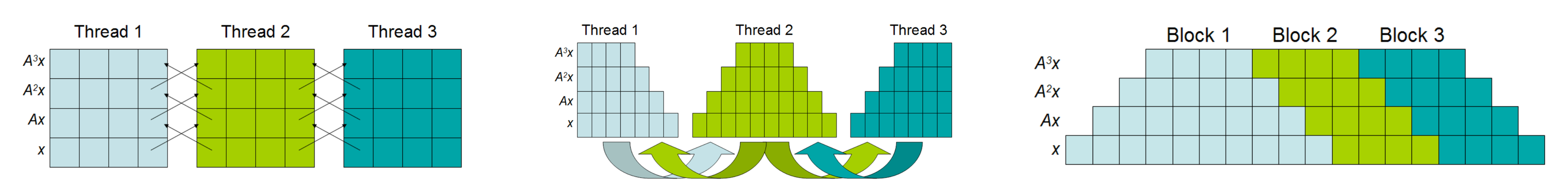
- Programmers write their apps in Python
  - Supports code generation in C/C++/CUDA
  - Under rapid development (patches welcome!)
- Public source repo:
  `git://github.com/shoaibkamil/asp.git`
- Wiki:
  `http://aspsejits.pbwiki.com/`
- Graduate course project: implement a specializer used in one of the ParLab apps

QUESTIONS

- How hard to convert existing efficiency code into a specializer? (Do you need to be a compiler jock?)
- Can specializers be composed, or will we end up with $O(n^2)$ specializers if n patterns?

A^kx Specializer

- Computes Krylov subspace basis vectors $\{x, Ax, A^2x, \ldots, A^kx\}$ using parallel and communication-avoiding methods
  - Building block for many sparse solvers
  - Depending on matrix properties, different methods give better performance
- Built by turning existing implementations into parameterized templates for 3 algorithms



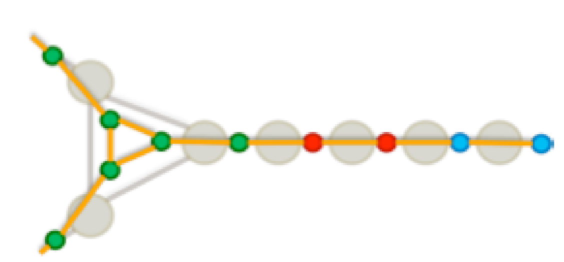
Parallel Algorithm — Communication-Avoiding Algorithm — Cache-blocked Algorithm

- Speedup vs Serial: **2.8x** to **11.7x** on tested matrices

Bloodflow Simulation (Circle of Willis) in Stroke Victims

- Combination of stencil and Newton-Raphson specializers
  - Artery interior points computed using stencils, junction points using nonlinear N-R solve

Example of artery and junction, showing border points (blue), interior points (red), and junction points (green).



- Multilevel parallelism
  - pthread per artery/junction, OpenMP parallelism in stencil
- **>10x** faster than pure Python with large room for improvement if composability of parallel libraries is improved

Gaussian Mixture Model Specializer

- Expectation-Maximization algorithm for Gaussian Mixture Modeling on CUDA-based GPUs
- See poster by Henry Cook and Ekaterina Gonina

Future Plans: Composition

- Motivation: activating OpenMP parallelism in the stencil portion of bloodflow simulation causes overprovisioning of hardware contexts
  - pthreads and OpenMP **both think they "own"** all available hardware contexts
  - problem is not unique to ASP!
- Lithe: Par Lab answer to composable libraries
  - Provides *hart* (hardware thread) abstraction that corresponds 1:1 with hardware context
  - Modified OpenMP/pthreads/TBB etc run on top of Lithe
- *Composability of specializers will depend on using Lithe abstractions*

Future Plans: Calling Back Into Python

- Currently, due to limitations of Python interpreter, can't call back interpreted functions from parallel regions
- Current workaround: mutual exclusion around queue of work going to interpreter thread
- Long term: improve AST analysis & code generation to cover most "simple" functions handed to specializer

Future Plans: ASPdb

- SaaS-based database to aggregate knowledge about optimal parameters for specializers
- Specializers submit own results to ASPdb, query for hints about tuning parameters for current platform

Conclusions

- Wrapping existing ELL code in specializers doesn't require compiler-fu
  - But more challenging if need new *abstraction*
- ASP is viable way to deliver autotuned code
- Composition presents resource-management challenges, but optimistic that Lithe can help

End-to-end Python+ASP apps now feasible & running