Our next Research IT Reading Group topic will be: HPC optimization and auto-tuning for Python code w/SEJITS

The presentation and discussion will be facilitated by Chick Markley, Staff Programmer for the Aspire Lab in EECS.

Chick writes: “If you write code in the high-level language Python, or wish that you could instead of writing in a low-level language like C/Fortran/OpenCL by hand, then SEJITS is an option that allows you to take advantage of high performance hardware such as GPUs and MICs by allowing you to write specialized kernels in Python that bridge that gap. Our framework provides tools to generate code at runtime, transparent to the user, with autotuning that specifically targets the application and the hardware environment.

We are looking for use cases with Python applications and domain experts that might benefit from SEJITS specializers.”

Here's the background material for review prior to our meeting:

SEJITS (Selected Embedded Just-In-Time Specialization) implements software patterns associated with computational or energy intensive algorithms (see Berkeley software motifs). These specializers exploit run-time information and auto-tuning to generate low level code that achieves near maximal performance in this domain.

SEJITS was developed by the AspireLab, formerly ParLab, with large contributions from Shoaib Kamal, originally at ParLab and now at MIT. The main paper describing this work is here.

---

More details are available on the main website at: sejits.org

All of the code is open source and available on github.com/ucb-sejits

The core code is CTree, A framework for writing SEJITS specializers

Also on the website is a collection of specializers for applications, the most mature specializers are for stencil code and also hinde mith (development branch) which is collection of linear algebra specializers optimized for optical flow applications.

Attending

- Chick Markley, Staff Programmer for the Aspire Lab in EECS
- Lenny Truong, Undergraduate Research Assistant on the SEJITS project
- Michael Gerstil, Aspire Lab

- Aaron Culich, CIO-RIT
- Aron Roberts, CIO-RIT
- Camille Villa, CIO-RIT
- Chris Hoffman, CIO-RIT
- David Greenbaum, CIO-RIT
- Patrick Schmitz, CIO-RIT
- Quinn Dombrowski, CIO-RIT
- Rick Jaffe, CIO-RIT
- Steve Masover, CIO-RIT
- Bill Allison, IST-API
- Ryan Lovett, SCF
- Dav Clark, D-Lab
- Bob Nadaeu, Seismological Research Lab
- Rick Katz, SALT
- Ronald Sprouse, Linguistics
- James McCarthy, SSL
- Nico Tripcevich, Archaeology
- Rachel Slaybaugh, Nuclear Engineering
- Brian Hamlin, OSGeo, California Chapter

Framing Questions

HPC optimization and auto-tuning for Python code w/SEJITS (Selected Embedded Just-In-Time Specialization)
1. How do I know my use case is a good fit for SEJITS?
2. Will my existing Python code benefit without rewriting part of it?
3. Is it worth investing time to write a specializer if I move code to other systems which may not have SEJITS installed?
4. Can I install it myself on my own laptop or cluster, or does it require special expertise and privileges to take advantage of the hardware?

**Slides**

ResearchIT-2014-10-09-SEJITS-talk.pdf

**Notes**

[Chick Markley, SEJITS Project, Aspire Lab: slide show -- to be linked soon]

ASPIRE: Algorithms and Specializers for Provably Optimal Implementations with Resiliency and Efficiency

SEJITS sweet spot: runtime info affects performance; runtime tuning can find optimal configuration, meta-specialization combines patterns

Main audience is people who can write "specializers" that fit in the SEJITS framework. Includes domain and efficiency experts, specialized hardware producers, and ultimately research scientists who can benefit from the efficiencies gained.

Aaron: How do I know if my use case is a good fit for SEJITS

Chick: Hard to answer. If your algorithm runs slow, that's a good clue. This is a big question for us.

Lenny: Looking at specific sections of code -- for loops, iterating over large data sets ... that's where the performance gains potentially occur. Specific, intense computations, things that are called a lot or form the bulk of processing time: that's where the wins are.

Dav: NUMBA (Python library). Takes 10 sec to use. I think you guys are doing much more efficient and specialized things ... so NUMBA might handle typical scientific use cases.

Chick: I suspect we'd do better than NUMBA, but haven't run head-to-head comparison. Stencils -- computation in which you have 2 or 3 dimensional matrix, need to commute new matrix in which each node is a function of neighboring points -- big wins for these types of compute.

Lenny: SEJITS differs from Numba: Numba optimizes NumPy arrays. We focus on optimizations in domain specific languages, abstracting away specifics of particular solutions and optimize the operations, e.g. matrix operations. Have not done performance evals, or ease of installation.

Chick: SEJITS focused on *single* node performance. Possibly w/ multiple cores.

Aaron: So who can install?

Chick: Python, PIP LLVM ....

Lenny: Conda ships with what you need. But OpenMP not in standard (mainline) version of LLVM, requires compile with certain flags. But if you're running on laptop, on your mac, it's about as simple as PIP.

Dav: Conda (package manager) has good facilities for compilation, including for example a forked version of LLVM, could be managed by Conda

Chick: Would be very interested in doing that.

Dav: BCE is "Berkeley Common Environment" -- aimed at common environment on VirtualBox or VMWare or Docker; also build makes it trivial to deploy to AWS.

[discussion re: what services are offered by Unix team in IST -- can they produce a VM from a VMWare image -- do they offer Docker-containerized software]

Aaron: Portability of optimization

Chick: Can examine back end ... converting to C alone is a huge performance win

Lenny: Even if you can't get the GPU access performance gains. Even if you develop on a laptop and deploy to a cloud, where different code is compiled, the goal is that one writes the same code.

Patrick: Would be interesting to see which Savio users have enough Python code to look at this.
Chick: Spark (open source cluster computing - from AMPLab) people have a Python wrapper for their Scala implementation. Opportunity there.

SEJITS on the individual nodes running in a Spark cluster.

Rick: Audio. Follow an instrument in an ensemble.

Chick: Gaussian model. My guess is yes, but am not familiar with that space.

Rick: Gunshot monitors?

Chick: Not sure what temporal limits of detecting a signature. What I've mostly heard has to do with audio stream.

Rick: Linguistics use cases, Ron?

Ron: Potentially, though current code is not in Python. Feature detection. Tends to be single speaker analysis. Would be interesting if we could rewrite in Python. Currently old C code, would be great to move off that dependency.

Aaron: Gains not so much re: code already in Fortran or C, but greenfield where researchers don't want to invest in learning low level languages

Chick: Sideloader -- a path we're exploring -- use Python as a code generator, write Python and compile it into an .so using SEJITS

Ryan: Startup penalty for using SEJITS the first time

Chick: It's there. One operation -- not worth it. Don't have metrics, but build is sub-second. Using MIT project OpenTuner for exploring parameter space in the code; also have brute force tuner of our own.

Lenny: How does it compare to static library already compiled? Have found use cases where SEJITS performs better because of ability for recompilation with parameters specific to a particular run.

Ryan: Accounting for number of cores?

Lenny: Not in our current model, but theoretically this is something that we could do.

Chick: SEJITS does query GPU re: how many cores it has, etc. This influences optimization. Not MP, but can imagine using multicore as a sort of GPU.

Patrick: Can take advantage of compilation when using it again on the same node; what about leveraging that compilation on other nodes

Chick: Hmm... That's version 3 ....

Michael: On tuning just-in-time. First 150 times the specializer is called it will try a different version of a program, then it will thereafter select the one that has performed best. You could imagine that kind of tuning across nodes that attempt different versions then communicate which runs fastest across the group of nodes.

Brian: From coders point of view, people here are fully dialed in to optimization questions, and work on that; but vast majority of Python programmers won't be at that level. How to bring people to the table, make it easy for suggestions to be visible to Python programmers and for the optimization to happen deeper in the machine. One project: A vehicle miles traveled analysis module ... used Python, ordinary Python arrays (not NumPy) ... could be one, hundreds, or thousands of cells being question -- a couple megabytes of floats in Python ... so there's a practical example. I'm aware that in Paul Wadell's project they moved a lot of calculations into NumPy arrays first, and consciously structured operations to take advantages of that. Whatever you can do to help NumPy.

Dav: Synthicity has some good coders, worth considering those use cases. Astro folks are doing lots of Python code over ginormous data sets. Worth connecting with them. Josh Bloom and Saul Perelmutter. AstroPy developers -- one of them is in that group.

Aaron: How would people interface with you who have a good use case.

Chick: start with me chick@eecs.berkeley.edu

Rachel: The Hacker Within. W 3:30-5:30, meeting in 190 Doe. Most people are doing and talking about Python. Mostly nuclear engineers, but hoping to expand. This might be a fruitful group in which to discuss SEJITS.

Rick Katz: SAIT has schedule builder -- sets of potential schedules for students planning courses -- back-end of a website, dual processor, Python 2.6.8 in a virtual environment; but Python 2.4.3 is more normally installed in our environment.

Rachel: I missed presentation part of this ... but can I ask about messy, multiple language codebases?

Lenny: Most of our work has been Python, but in theory you could handle crosstalk.

Rick Katz: Decorator?

Chick: Yes, Decorator pattern or otherwise overwriting a particular class.

Ryan: Does C code get preserved?

Lenny: Not currently, but it could be. Caching generated programs.

Brian: Root project at CERN? Interactive C++ -- in a shell.

Aaron: What languages do you support?
Chick: So far just Python. Pressure to support Scala, but there are hurdles there. Down the road: FPGA 'glued' to core, so hardware runs specialized parts of algorithm. Intel is on a path to offering this, but it's not for sale yet.

Dav: Bitcoin group might be folks interested in working on that.

Chick: Pressamonious. Floating point optimization.