PySKI: The Python Sparse Kernel Interface

Erin Carson, Ben Carpenter, Armando Fox, and James Demmel

Motivation

- Efficiency: Low-level Auto-tuning libraries, such as OSKI, enable better performance for scientific computations
  - Complex matrix tuning optimizations
  - C code enables near peak performance
  - Hard to write

- Productivity: Higher level languages, such as Python, enable faster/better code development
  - 2-5x faster (P. Hudak and M. P. Jones, 1994)
  - Less efficiency

- Can we combine the benefits of both?

OSKI

- C library used in solver libraries
- BLAS-style interface (SpMV, SpTS, etc)
- Provides automatically tuned kernels for sparse matrices
  - Optimal tuning choices are often non-obvious

Library Install-Time (offline) → Application Run-Time

- 1. Build for Target Arch
- 2. Benchmark
- 1. Evaluate Models
- 2. Select Data Struck & Code

PySKI Ideas

- Provide Python bindings for OSKI via scipy.sparse
  - A python sparse matrix package with some overlap with OSKI
  - OSKI maintains data structures plus "shadow" data structures for tuning
  - Abstract datatypes wrap pointers to these structures

- Expose higher-level abstract datatypes & methods to productivity programmer
  - low-level OSKI objects become invisible to mainline computation

- Idea: separate tuning hints from main source code
  - changes to policy don’t contaminate source
  - policy experimentation can proceed in parallel
  - Enables performance portability

DECORATOR CODE

```python
def check_OSKI(*args):
    if OSKI is installed:
        if check_for_hint():
            set_hints()
            tune_mats()
        call OSKI SpMV
        gather profiling data
    else:
        fall through to scipy SpMV code
```

USER PROGRAM

```python
import scipy.sparse
A = csr_matrix()
b = array()
c = A*b
```

SPICY SOURCE CODE

```python
@check_OSKI
def mul_(*args):
    perform matmul
```

Relation to SEJITS

- PySKI and SEJITS will use the same infrastructure
  - All part of the PLL Interpreter
  - Specializer recognizes that OSKI has a tuned version of the function being called

- PySKI (OSKI) is just a first attempt
  - There are many other auto-tuning libraries that can be incorporated into higher level languages
  - Broader goal: enable the user to write high performance code productively

Future Work

- Obtain experimental results for prototype
- Implement and release fully functional version of PySKI
- Integration with SEJITS
- Investigate the problems of co-tuning and maintaining databases for tuning results

Summary: The goal of the PySKI project is to bridge the gap between the productivity and efficiency layers of programming. PySKI can allow scientists to write their code in an expressive, simple language, while still taking advantage of the increased performance that OSKI provides through automated tuning.

Challenges

- Identification of the Call Site
  - Must associate explicit tuning hints given by the user with specific calls
  - Need to know when and where the call happens
  - Current approach: using tags
  - How much should the user specify?
  - What if co-tuning is required?

- Handling history
  - Currently OSKI records the calling arguments on every call.
  - This could in principle be saved across multiple runs and time-stamped
  - Future work: maintain tuning database

Conclusions and Future Work

- Future Work
  - Obtain experimental results for prototype
  - Implement and release fully functional version of PySKI
  - Integration with SEJITS
  - Investigate the problems of co-tuning and maintaining databases for tuning results

- Summary: The goal of the PySKI project is to bridge the gap between the productivity and efficiency layers of programming. PySKI can allow scientists to write their code in an expressive, simple language, while still taking advantage of the increased performance that OSKI provides through automated tuning.