PySKI: THE PYTHON SPARSE KERNEL INTERFACE

Erin Carson
Ben Carpenter
Armando Fox
James Demmel
THE NEED FOR AUTO-TUNING

Mflops/s for Various Block Sizes in MatMul Operation

$k_0 = 1$

333 MHz Sun Ultra 2i

2-D slice of 3-D space; implementations color-coded by performance in Mflop/s

16 registers, but 2-by-3 tile size fastest
WHAT IS OSKI?

- “Optimized Sparse Kernel Interface”
- C Library used in solver libraries
- BLAS-style interface
  - SpMV, SpTS, etc.
- Automatically tuned computational kernels on sparse matrices
  - Optimal tuning choices are often non-obvious
  - 3 Types of Tuning
    - Static tuning (based on system)
    - implicit dynamic tuning (performance monitoring)
    - explicit dynamic tuning (workload hints)
PYSKI MOTIVATION

- Productivity code: express computation in the easiest way possible
- Efficiency code: tuning and other implementation choices to get best performance
- C/OSKI code conflates the two
  - When to change representation of a matrix?
  - When to do expensive "unmarshal" of a representation?
  - When to tune and re-tune?
    - Setting explicit tuning hints
EXAMPLE: TUNING WITH EXPLICIT HINTS

```c
oski_matrix_t A_tunable = oski_CreateMatCSR( ... );

/* Tell OSKI we will call SpMV 500 times (explicit workload hint) */
oski_SetHintMatMult(A_tunable, OP_NORMAL, α, x_view, β, y_view, 500);

/* Tell OSKI we think the matrix has 8x8 blocks (structural hint) */
oski_Set Hint(A_tunable, HINT_SINGLE_BLOCKSIZE, 8, 8);

/* Ask OSKI to tune */
oski_TuneMat(A_tunable);

for( i = 0; i < 500; i++ )
    oski_MatMult(A_tunable, OP_NORMAL, α, x_view, β, y_view);
```
PYSKI SEPARATES TUNING FROM COMPUTATION

- Provide Python bindings for OSKI via scipy.sparse
  - 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 into another file
  - changes to policy don't contaminate source
  - *policy experimentation can proceed in parallel*
BACKUP SLIDES
HOW OSKI TUNES (OVERVIEW)

1. Build for Target Arch.
2. Benchmark

Library Install-Time (offline)  Application Run-Time

1. Evaluate Models
2. Select Data Struct. & Code

Workload from program monitoring

Matrix

Generated code variants
Benchmark data

History
Heuristic models

To user: Matrix handle for kernel calls

Extensibility: Advanced users may write & dynamically add “Code variants” and “Heuristic models” to system.
SUMMARY OF PERFORMANCE OPTIMIZATIONS

- **Optimizations for SpMV**
  - **Register blocking (RB):** up to 4x over CSR
  - **Variable block splitting:** 2.1x over CSR, 1.8x over RB
  - **Diagonals:** 2x over CSR
  - **Reordering** to create dense structure + **splitting:** 2x over CSR
  - **Symmetry:** 2.8x over CSR, 2.6x over RB
  - **Cache blocking:** 2.8x over CSR
  - **Multiple vectors (SpMM):** 7x over CSR
  - And combinations...

- **Sparse triangular solve**
  - Hybrid sparse/dense data structure: **1.8x** over CSR

- **Higher-level kernels**
  - $A \cdot A^T \cdot x$, $A^T \cdot A \cdot x$: **4x** over CSR, 1.8x over RB
  - $A^2 \cdot x$: 2x over CSR, 1.5x over RB
  - $[A \cdot x, A^2 \cdot x, A^3 \cdot x, .. , A^k \cdot x]$