Bridging the Performance-Productivity Gap with Selective Embedded Just-In-Time Specialization

Shoaib Kamil
CSAIL, MIT

Armando Fox, Katherine Yelick
UPCRC, EECS Dept, UC Berkeley
Productivity-Performance Gap

• Domain scientists want to write code in high-level languages that match domain
  \[ x = A \backslash b \]
  or
  \[ \text{model} = \text{gmm}(\ldots) \]

• Not worry about typing, parallelism, etc

\[
\text{implicit def s2r}[A,\_,I<:\text{Seq}[A]](xs: I) \\
\{\ldots\}
\]
Productivity-Performance Gap

• Domain scientists want to write code in high-level languages that match domain
• For best performance, must rely on *efficiency programmer*
• Optimized code highly dependent on platform

1/10 LOC  
1/100 Performance

10-100x LOC  
100x Performance

Bryan Catanzaro & PALLAS Group
“Our” Pattern Language (OPL-2010)
(Kurt Keutzer, Tim Mattson)

**Structural Patterns**
- Pipe-and-Filter
- Event-Based/Implicit Invocation
- Puppeteer

**Computational Patterns**
- Graph- and State-Machines
- Divide-and-Bound
- Sparse-Linear-Algebra
- Unstructured-Grids
- Structured-Grids

**Software Stack**
Deals with Implementation

**Concurrent Algorithm Strategy Patterns**
- Task-Parallelism
- Divide and Conquer

**Implementation Strategy Patterns**
- SPMD
- Data-Par/index-space
- Fork/Join
- Task-Queue
- Actors
- Actors (global)
- Task-Graph
- Partitioned Graph
- Distributed-Array
- Shared-Data

**Parallel Execution Patterns**
- MIMD
- SIMD
- Thread-Pool
- Task-Graph
- Transactions

Concurrency Foundation constructs (not expressed as patterns)
- Thread creation/destruction
- Process creation/destruction
- Message-Passing
- Collective-Comm.
- Point-To-Point-Sync. (mutual exclusion)
- collective sync. (barrier)

\[ A = M \times V \]
Example: Stencil Computations

for (int i=1; i<nx-1; i++)
    for (int j=1; j<ny-1; j++)
        output[i,j] = f(output[i,j], neighbors(input[i,j]));

• The function f() changes application-to-application

• Tuning of loops requires information about input set
What is an embedded DSL

- DSL compiler using a host language’s syntax
  - Common example: macro rewriting as in Lisp
  - Difficulty depends on language capabilities
- Leverage capabilities of host language
- Often not same semantics
  - Contrast with APIs & libraries
“Stovepipes”: Using DSELs to Architect Applications

Single program expresses computation, “stovepipes” turn computation into optimized code at run-time.
Overview

- Motivation: Productivity-Performance Gap
- SEJITS Methodology
- Asp & DSELs for Python
- Mechanisms for DSEL Implementation
- Future/Current Work
- Conclusions
Selected Embedded JIT Specialization

Productivity app

Interpreter

DSEL Compiler

Asp Framework

OS/HW

DSMC 2012
Selected Embedded Just-In-Time Specialization (SEJITS)

• Domain scientists (aka *productivity programmers*) write code in embedded DSLs
• *Efficiency programmers* create embedded DSLs instead of one-off libraries or application optimization
• Separation of concerns
• “Invisible” to productivity programmers
  – Except it runs fast
SEJITS Methodology

• Goal: productive portable performance
• Add DSEL support to productivity languages
  – Leverage features of modern “scripting” languages
  – Leverage existing libraries for these languages
• Use external parallelizing/optimizing compilers
  – Leverage existing expertise of efficiency programmers
  – Leverage existing high-performance external libraries
• Use auto-tuning
  – Search over multiple implementations
Auto-tuning: Empirical Search for Best Performance

• A priori determining the best low-level code is difficult, even for expert programmers

• Idea: generate many parameterized versions of a kernel

• Run all of them on the target machine and choose the fastest

• Usually run at install-time
Auto-tuning Matrix Multiply

Better →

Naïve Code

Vendor-provided Expert Code

Auto-tuned Code
Asp is SEJITS for Python

• Proof of concept framework for “easy-to-build” embedded parallel DSLs
• Pragmatic choice: Python used in scientific community
• DSL implementers can use some or all of the building blocks provided

http://sejits.org
# Implemented DSELs/Libraries

<table>
<thead>
<tr>
<th>DSEL/Library</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stencil/Structured Grid</td>
<td>x86+OpenMP</td>
</tr>
<tr>
<td>Semantic Graphs Filtering &amp; Semiring Operations in KDT</td>
<td>x86+MPI</td>
</tr>
<tr>
<td>Parallel Map</td>
<td>x86+processes, cloud</td>
</tr>
<tr>
<td>Gaussian Mixture Modeling</td>
<td>CUDA, Cilk Plus</td>
</tr>
<tr>
<td>CA Matrix Powers for CA Krylov Subspace Methods</td>
<td>x86+pthreads</td>
</tr>
<tr>
<td>Bag of Little Bootstraps*</td>
<td>x86+Cilk Plus, Cloud via Spark</td>
</tr>
<tr>
<td>GraphLab DSEL for Machine Learning via Graphs*</td>
<td>x86+pthreads</td>
</tr>
<tr>
<td>CA Parallel Recursive Structural Pattern*</td>
<td>x86+Cilk Plus</td>
</tr>
</tbody>
</table>
Stencil DSEL Performance

11x faster than auto-parallelizing.
~2.5x faster than state of art non-auto-tuning DSL

Geometric mean of 93% of attainable peak.
Stencil DSEL Performance

Structured Grid Fraction of Peak Performance (boxboro)

- Laplacian
- div
- grad
- hex-div
- trismooth
- g.mean

Fraction of Peak

dsl
pochoir
Communication-Avoiding Recursive Matrix Multiply

• For recursive algorithms with particular branching factor relative to memory usage
• Choose when to perform parallel steps vs serial steps

• Optimal choice attains lower bounds on communication for matrix multiply
CARMA Performance on NUMA Machine

Narrow: $m = n = 64$, $k$ large

Beat MKL by 10x, using MKL.
Mechanism: Code Templates

void vec_add(float *x, float *y) {
    x[0] += y[0];
    x[1] += y[1];
    x[2] += y[2];
}

• Code snippets in backend language with interspersed with Python
• For “simple” code generation
Mechanism: Phased Transformations

1. User code
2. Parse to Python AST
3. Convert to Backend AST
4. Optimize Backend AST
5. Write Out Source Files
6. Call External Compiler
7. Load & Run Shared Lib
8. Return Value

- Optimize IR
- Convert to Domain-Specific IR
Example Code

from stencil_kernel import *

class Laplacian3D(StencilKernel):

    def kernel(self, in_grid, out_grid):
        for x in self.interior_points(out_grid):
            for y in self.neighbors(in_grid, x, 1):
                out_grid[x] += (1.0/6.0) * in_grid[y]
Example Code

```python
from stencil_kernel import *

class Laplacian3D(StencilKernel):
    def kernel(self, in_grid, out_grid):
        for x in self.interior_points(out_grid):
            for y in self.neighbors(in_grid, x, 1):
                out_grid[x] += (1.0/6.0) * in_grid[y]
```
void kernel_optimized(double* in_grid, double* out_grid) {
    #define min(a, b) (_a < _b ? _a : _b)
    #define _idx(d0, d1, d2) ((d0 * 258*258) + (d1 * 258) + d2)

    for (int x1x1 = 1; (x1x1 <= 256); x1x1 = (x1x1 + (1 * 192))) {
        for (int x2x2 = 1; (x2x2 <= 256); x2x2 = (x2x2 + (1 * 160))) {
            #pragma omp parallel for
            for (int x1 = x1x1; (x1 <= min((x1x1 + 191), 256)); x1 = (x1 + 1)) {
                for (int x2 = x2x2; (x2 <= min((x2x2 + 159), 256)); x2 = (x2 + 1)) {
                    #pragma ivdep
                    for (int x3 = 1; (x3 <= (256 - 3)); x3 = (x3 + (1 * 4))) {
                        int x4;
                        x4 = _idx(x1, x2, x3);
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 0))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 0))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 0))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 0))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 0))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 1))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 2))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 1), (x3 + 2))]));
                        x4 = _idx(x1, x2, (x3 + 3));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 1), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 1), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                        out_grid[x4] = (out_grid[x4] + ((1.0 / 6.0) * in_grid[_idx((x1 + 0), (x2 + 0), (x3 + 3))]));
                    }
                }
            }
        }
    }
}

DSMC 2012
Future/Current Work

• Improved auto-tuning via machine learning
• SEJITS + fast hardware prototyping & co-tuning
  – CHISEL project
• Composition in pattern-based frameworks
• Multi-level debugging
• Synthesizing optimized code (versus compiling)
Related Work

• Delite (Stanford)
• Petabricks (MIT)
  – Smart auto-tuning for algorithmic choice
• Auto-tuning compilers (Mary Hall)
  – User-guided auto-tuning for general compilers
  – Difficult to automate due to domain knowledge required
• Auto-tuning motifs
  – PhiPAC, FFTW, ATLAS, OSKI, Spiral, & more
Conclusions

• High performance productive programming is possible with the SEJITS approach
• Also makes easier to write autotuners
• Much work in progress to make it even more easier to use

• BSD Licensed, available
http://www.sejits.org/
Acknowledgements

• Armando Fox, Katherine Yelick
• Parlab professors: Krste Asanović, Ras Bodik, James Demmel, Armando Fox, Kurt Keutzer, John Kubiatowicz, David Patterson, Koushik Sen, David Wessel, Katherine Yelick
• Grad students: Scott Beamer, Derrick Coetzee, Henry Cook, Michael Driscoll, Ekaterina Gonina, Jeffrey Morlan, Jonathan Harper, Erin Carson, Nick Knight
• LBNL/external: Aydin Buluc, Sam Williams, Adam Lugowski, John Gilbert, Leonid Oliker, John Shalf
• Intel/Microsoft: Burton Smith, Tim Mattson, Henry Gabb, Robert Geva, Juan Vargas
• Many undergrads

This work was performed at the UC Berkeley Parallel Computing Laboratory (Par Lab), supported by DARPA (contract #FA8750-10-1-0191) and by the Universal Parallel Computing Research Centers (UPCRC) awards from Microsoft Corp. (Award #024263) and Intel Corp. (Award #024894), with matching funds from the UC Discovery Grant (#DIG07-10227) and additional support from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Oracle, and Samsung.
BACKUP SLIDES
Introspect to Get AST
Transform into IR

Domain Specific Constructs
Transform into Platform AST & Optimize

- Bulk of performance expert’s knowledge
- Use of Asp’s infrastructure for common transformations
- Can generate many variants at once (for auto-tuning)