Increasing Availability and Impact of Compiler Technology for HPC

Mary Hall, University of Utah

Collaborators and Acknowledgements

Stencils, Bricks and Geometric Multigrid

Protonu Basu (Facebook), Tuowen Zhao, Sam Williams, Brian Van Straalen, Lenny Oliker, Phil Colella, Hans Johansen

Sparse Matrix Computations

Anand Venkat (Intel), Khalid Ahmad, Payal Nandy, Michelle Strout, Mahdi Mohammadi

Tensor Contractions

Thomas Nelson, Axel Rivera (Intel), Prasanna Balaprakash, Paul Hovland, Liz Jessup, Boyana Norris

LLVM Polly Optimization and Pragma Autotuner

Michael Kruse, Hal Finkel, Vinu Sreenivasan

Funded in part by Department of Energy Office of Advanced Scientific Computing Research under the Exascale Computing Project, awards DE-SC0008682 and Scientific Discovery through Advanced Computation (SciDAC) award DE-SC0006947, and by the National Science Foundation award CCF-1018881.

Two Parts of Talk

- Performance portability with autotuning compiler technology
- Migrating autotuning compiler technology into Open Source compilers, leveraging deep learning investments

Part I: Autotuning Compiler Technology for Performance Portability

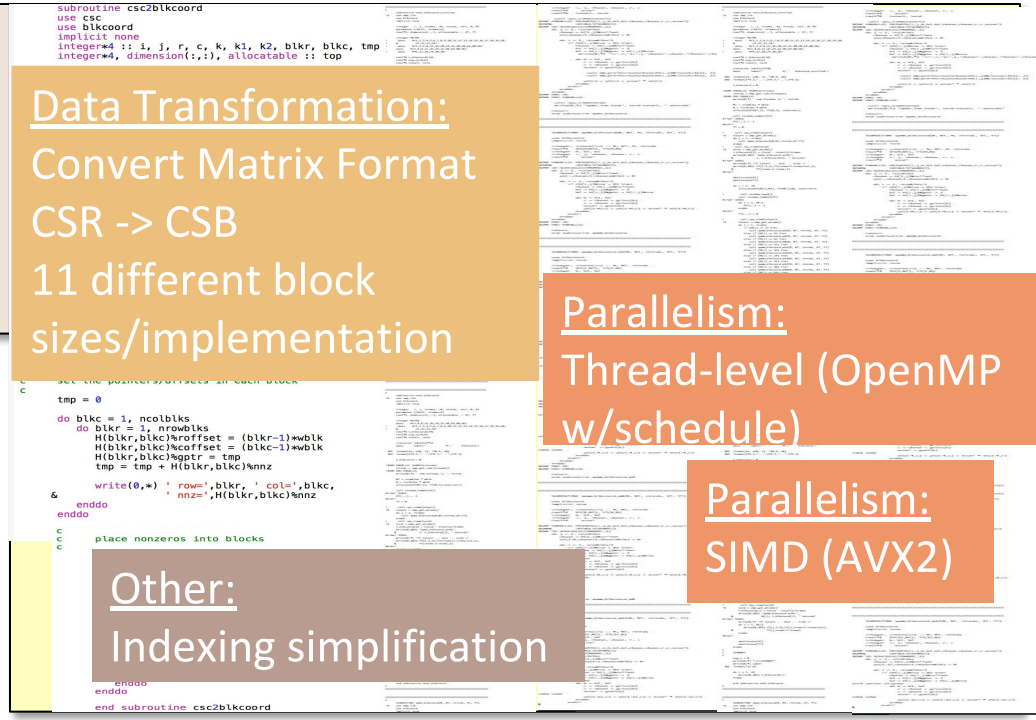
Optimizing Sparse Codes: Which Version Would You Prefer to Write?

```
/* SpMM from LOBCG on symmetric matrix */
for( i =0; i < n ; i ++) {
  for ( j = index [ i ]; j < index [ i +1]; j ++)
    for( k =0; k < m ; k ++);
      y [ i ][ k ]+= A [ j ]* x [ col [ j ]][ k ];
  /* transposed computation exploiting symmetry*/
  for ( j = index [ i ]; j < index [ i +1]; j ++)
    for( k =0; k < m ; k ++)
      y [ col [ j ]][ k ]+= A [ j ]* x [ i ][ k ];
}
```

Code A:
Multiple SpMV computations (SpMM), 7 lines of code

Question: Can a compiler generate **Code B** starting with **Code A**?

Answer: *YES*



Code B: Manually-optimized SpMM from LOBCG, 2109 lines of code

THE UNIVERSITY OF UTAH

Optimizing Stencil Codes: Which Version Would You Prefer to Write?

```
/* Laplacian 7-point Variable-Coefficient Stencil */
for (k=0; k<N; k++)
  for (j=0; j<N; j++)
    for (i=0; i<N; i++
      temp[k][j][i] = b * h2inv * (beta_i[k][j][i+1] * ( phi[k][j][i+1] – phi[k][j][i] )
          -beta_i[k][j][i] * ( phi[k][j][i] – phi[k][j][i-1] )
        +beta_j[k][j+1][i] * ( phi[k][j+1][i] – phi[k][j][i] )
        -beta_j[k][j][i] * ( phi[k][j][i] – phi[k][j-1][i] )
        +beta_k[k+1][j][i] * ( phi[k+1][j][i] – phi[k][j][i] )
        -beta_k[k][j][i] * ( phi[k][j][i] – phi[k-1][j][i] ) );
```

```
/* Helmholz */
for (k=0; k<N; k++)
  for (j=0; j<N; j++)
    for (i=0; i<N; i++)
      temp[k][j][i] = a * alpha[k][j][i] * phi[k][j][i] – temp[k][j][i];
```

```
/* Gauss-Seidel Red Black Update */
for (k=0; k<N; k++)
  for (j=0; j<N; j++)
    for (i=0; i<N; i++){
      if ((i+j+k+color)%2 == 0 )
      phi[k][j][i] = phi[k][j][i] – lambda[k][j][i] * (temp[k][j][i] –
rhs[k][j][i]);}
```

Memory Hierarchy

Prefetch

Data staged in registers/buffers

AVX SIMD intrinsics

Parallelism

Ghost zones:
Tradeoff computation for communication

Parallel Wavefronts:
Reduce sweeps over 3D grid

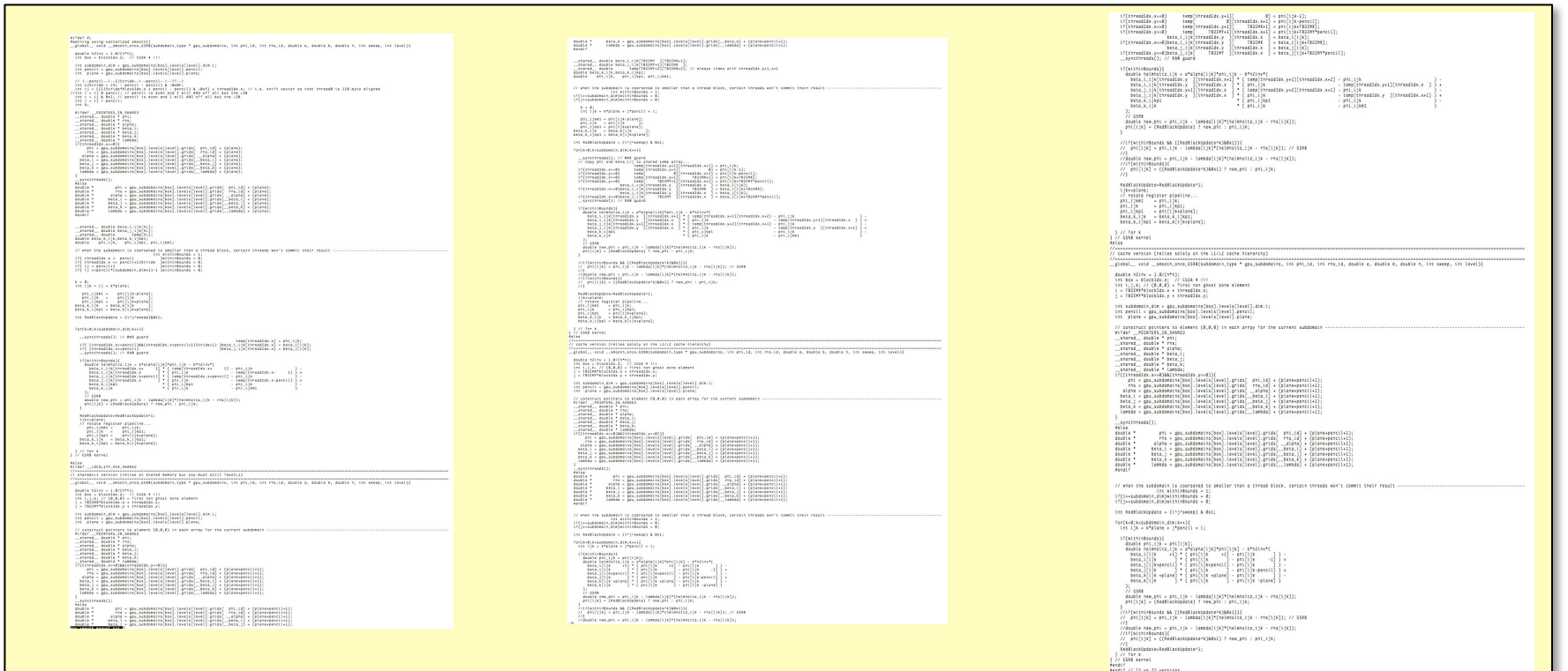
Nested OpenMP and MPI

Spin locks in OpenMP

Code A: miniGMG baseline smooth operator
approximately 13 lines of code

Code B: miniGMG optimized smooth operator
approximately 170 lines of code

And now GPU code?



Code C: miniGMG optimized smooth operator for GPU, 308 lines of code for just kernel

Autotuning Compiler: Transforming Code A to Code B,C

```
/* Laplacian 7-point Variable-Coeffi
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++
         temp[k][j][i] = b * h2inv * (beta
               -beta_i[k][j][i] * ( phi[k][j]
            +beta_j[k][j+1][i] * ( phi[k][j
            -beta_j[k][j][i] * ( phi[k][j][i]
            +beta_k[k+1][j][i] * ( phi[k+
            -beta_k[k][j][i] * ( phi[k][j][i
```

```
/* Helmholz */
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++)
         temp[k][j][i] = a * alph
```

```
/* Gauss-Seidel Red Black U
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++){
         if ((i+j+k+color)%2 ==
         phi[k][j][i] = phi[k][j][i
rhs[k][j][i]);}
```

Code A

```
/* jacobi_box_4_64.py, 27-pt stencil, 64^3 box size */
from chill import *

#select which computation to optimize
source('jacobi_box_4_64.c')
procedure('smooth_box_4_64')
loop(0)
original() # fuse wherever possible

#create a parallel wavefront
skew([0,1,2,3,4,5],2,[2,1])
permute([2,1,3,4])

#partial sum for high order stencils and fuse result
distribute([0,1,2,3,4,5],2)
stencil_temp(0)
stencil_temp(5)
fuse([2,3,4,5,6,7,8,9],1)
fuse([2,3,4,5,6,7,8,9],2)
fuse([2,3,4,5,6,7,8,9],3)
```

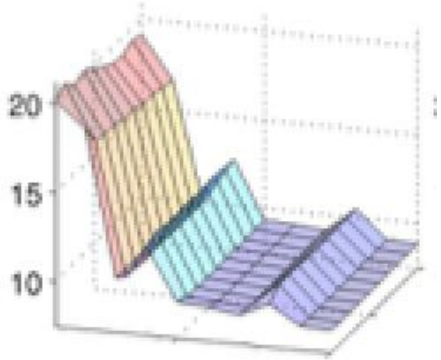
```
/* gsrb.lua, variable coefficient GSRB, 64^3 box size */
init("gsrb_mod.cu", "gsrb",0,0)
dofile("cudaize.lua") # custom commands in lua

# set up parallel decomposition, adjust via autotuning
TI=32 TJ=4 TK=64 TZ=64

tile_by_index(0, {"box","k","j", "i"},{TZ,TK, TJ, TI},{l1_control="bb", l2_control="kk",
l3_control="jj", l4_control="ii"},{"bb","box","kk","k","jj","j","ii","i"})

cudaize(0, "kernel_GPU",{_temp=N*N*N,_beta_i=N*N*N,
_phi=N*N*N},{block={"ii","jj","box"}, thread={"i","j"}},{})
```

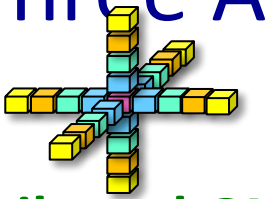
Transformation Recipes, Separate Code from Architecture Mapping



Autotuning to search across potential implementations

THE UNIVERSITY OF UTAH

Three Application Domains, One Compiler



Stencils and GMG

- *Memory-bandwidth bound:* Communication-avoiding optimizations
- *Compute bound:* Eliminate redundant computation (partial sums or bricks)

Sparse Tensors

- *Specialize matrix representation:* Data transformations
- *Incorporate runtime information:* Inspector/executor
- *Support non-affine input & transformations*

$$\underline{v} = (I \bigotimes I \bigotimes C)\underline{u}$$

$$v_{i,j,k} = \sum_{p} C_p^k * u_{i,j,p}$$

Tensor Contractions

- *Reduce computation:* Reassociate
- *Optimize memory access pattern:* Modify loop order to best match data layout and memory hierarchy
- *Adjust parallelism*

THE UNIVERSITY OF UTAH

Similar Idea is Gaining Traction for Domain-Specific Frameworks

Halide

a language for image processing and computational photography

```
vectorize(x_inner, factor), equivalent to gradient.split(x, x, x_inner, 4);
gradient.vectorize(x_inner);
gradient.parallel(tile_index);
gradient.split(x, x_outer, x_inner, 2);
gradient.unroll(x_inner), equivalent to gradient.unroll(x, 2);
gradient.tile(x, y, x_outer, y_outer, x_inner, y_inner, 4, 4);
gradient.reorder(y, x); // similar to transpose
gradient.split(x, x_outer, x_inner, 2)
fuse(x, y, fused)
```

saml

TVM Stack

`tile`(*x_parent, y_parent, x_factor, y_factor*)

Perform tiling on two dimensions

The final loop order from outmost to inner most are [x_outer, y_outer, x_inner, y_inner]

Parameters:
- **x_parent** (*IterVar*) – The original x dimension
- **y_parent** (*IterVar*) – The original y dimension
- **x_factor** (*Expr*) – The stride factor on x axis
- **y_factor** (*Expr*) – The stride factor on y axis

Returns:
- **x_outer** (*IterVar*) – Outer axis of x dimension
- **y_outer** (*IterVar*) – Outer axis of y dimension
- **x_inner** (*IterVar*) – Inner axis of x dimension
- **p_y_inner** (*IterVar*) – Inner axis of y dimension

`unroll`(*var*)

Unroll the iteration.

Parameters: **var** (*IterVar*) – The iteration to be unrolled.

`vectorize`(*var*)

Vectorize the iteration.

Parameters: **var** (*IterVar*) – The iteration to be vectorize

Successes so far using this approach

Partner with applications team - sustainable?

Produce tuned code that is integrated into application, does not evolve

- Nek5000
- CPPTRAJ, analysis code for Amber

Point to performance issues, and then code team rewrites

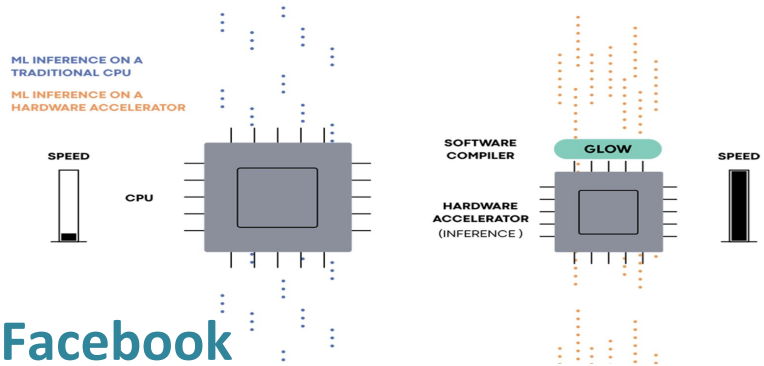
- PFLOTRAN
- MPAS-OCEAN

Part II: Leveraging investments in “Deep Learning” to build open source HPC compiler technology

Deep Learning Compiler Technology

POSTED ON SEP 13, 2018 TO AI RESEARCH, ML APPLICATIONS

Glow: A community-driven approach to AI infrastructure



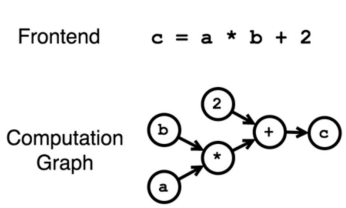
Facebook

Google

MLIR: Multi-Level Intermediate Representation Compiler Infrastructure

2019 European LLVM Developers Meeting

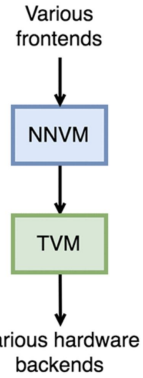
A typical framework



GPU-optimized * and +
CPU-optimized * and +

Amazon

NNVM Compiler



Challenges and opportunities:

- Domain-specific
- Many frontends
- Many target architectures
- Abundant parallelism and data reuse
- Must scale to large problems

http://code.fb.com/ml-applications/glow-a-community-driven-approach-to-ai-infrastructure/

http://aws.amazon.com/blogs/machine-learning/introducing-nnvm-compiler-a-new-open-end-to-end-compiler-for-ai-frameworks/

Convolutional Neural Network Forward Layer Code (in C)

```
for (n=0; n<N; n++) { // minibatch size
  for (k=0; k<K; k ++) { // output feature map
    for (c=0; c<C; c ++) { // input feature map
      for (p=0; p<P; p ++) { // output height
        ij = p * u; // input height
        for (q =0; q<Q; q ++) { // output width
          ii = q * v; // input width
          for (r=0; r<R; r ++) { // filter height
            for (s =0; s< S; s ++) {// filter width
              output_seq[n][k][p][q] +=
                    input [n][c][ij+r][ii+s] * weight[k][c][r][s];
} } } } } } }
```

Current State of HPC Compilers

Proprietary

- Robust
- High-quality implementations for supported architectures
- Support HPC community
- Code not performance portable across systems
- Often conservative

Open Source

- Research compilers
  - State-of-the-art
  - Experimental, untrusted
  - Difficult to track language changes
  - Gaps, such as Fortran frontend
- LLVM and gcc
  - Gaps in HPC support
  - Conservative

Current State of HPC Compilers, cont.

Challenges:

- HPC market not large enough to drive significant change to open source or even proprietary compilers
- Meanwhile, research systems not sufficiently robust for production codes

Impact:

- Productivity improvements for HPC not being exploited
- Heterogeneity will make this a bigger concern

Goal: HPC Support in Open Source Compilers

Short-term

(ECP time frame)

Extend LLVM

- Parallel IR
- Loop transformations
- OpenMP/OpenACC
- Autotuning
- Fortran frontend

Longer term

(But need to start now)

Work with Google on MLIR?

- Higher level of abstraction
- Composability of different views (parallelism?)
- Built-in polyhedral transformations and code generation
- Multiple backends via LLVM
- Missing frontends

ECP Y-Tune Efforts I Didn't Discuss

Transformation recipes using loop transformation pragmas in Clang/LLVM/Polly

- Effort to standardize in OpenMP

Pragma autotuner

- Used for OpenMP or loop transformation pragmas

Application and library parameter tuning

- Sherry Li's talk tomorrow

Search using Random Forests (SuRF)

- Some search framework must support all of these