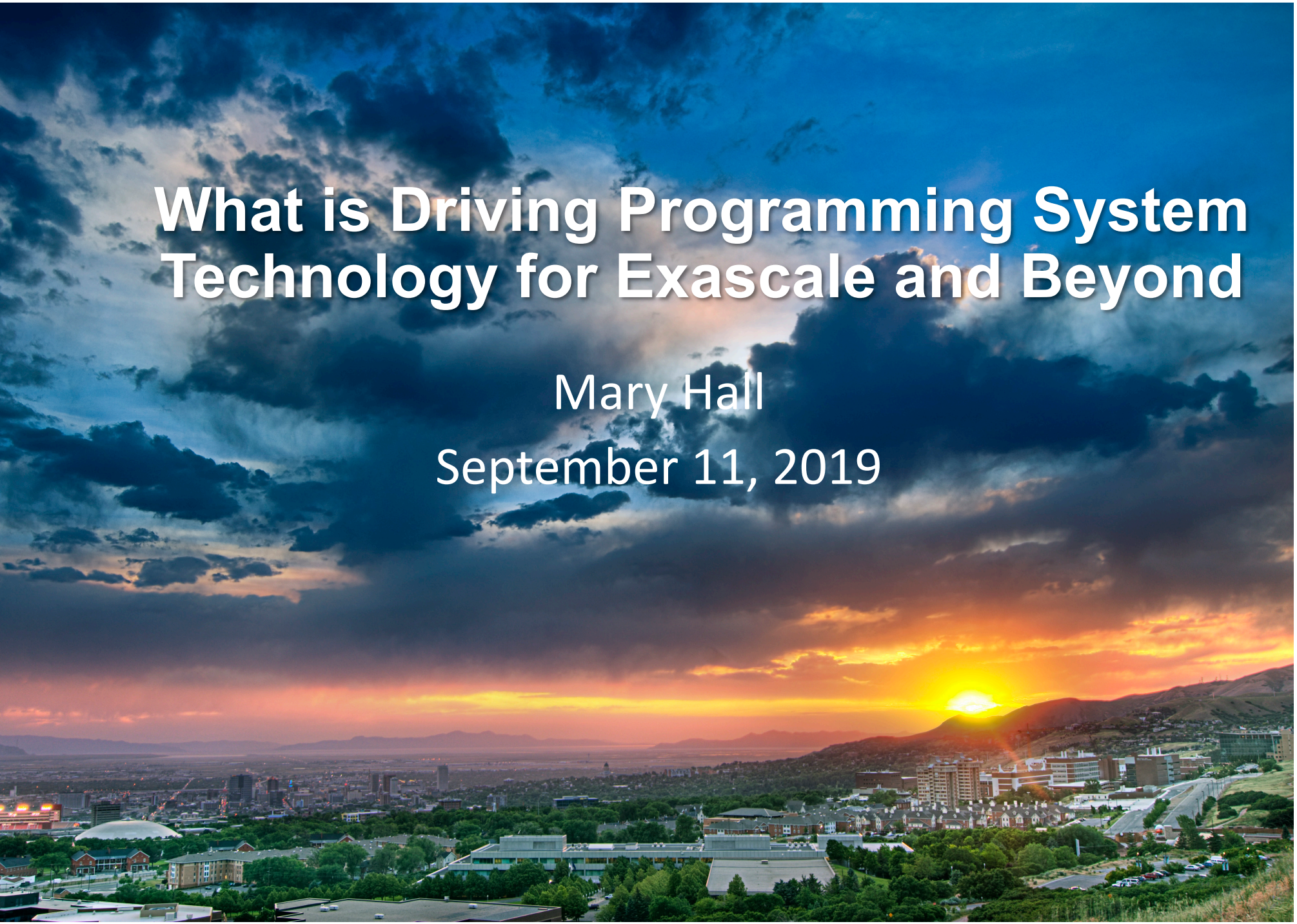


What is Driving Programming System Technology for Exascale and Beyond

Mary Hall

September 11, 2019

Collaborators and Acknowledgements

Stencils, Bricks and Geometric Multigrid

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

Autotuning Search and Pragma Autotuner

Prasanna Balaprakash, Paul Hovland, Vinu Sreenivasan, Rajath Javali

LLVM and Polly Optimization

Michael Kruse, Hal Finkel, Vinu Sreenivasan

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This research used resources in Lawrence Berkeley National Laboratory and the National Energy Research Scientific Computing Center, which are supported by the U.S. Department of Energy Office of Science's Advanced Scientific Computing Research program under contract number DE-AC02-05CH11231.

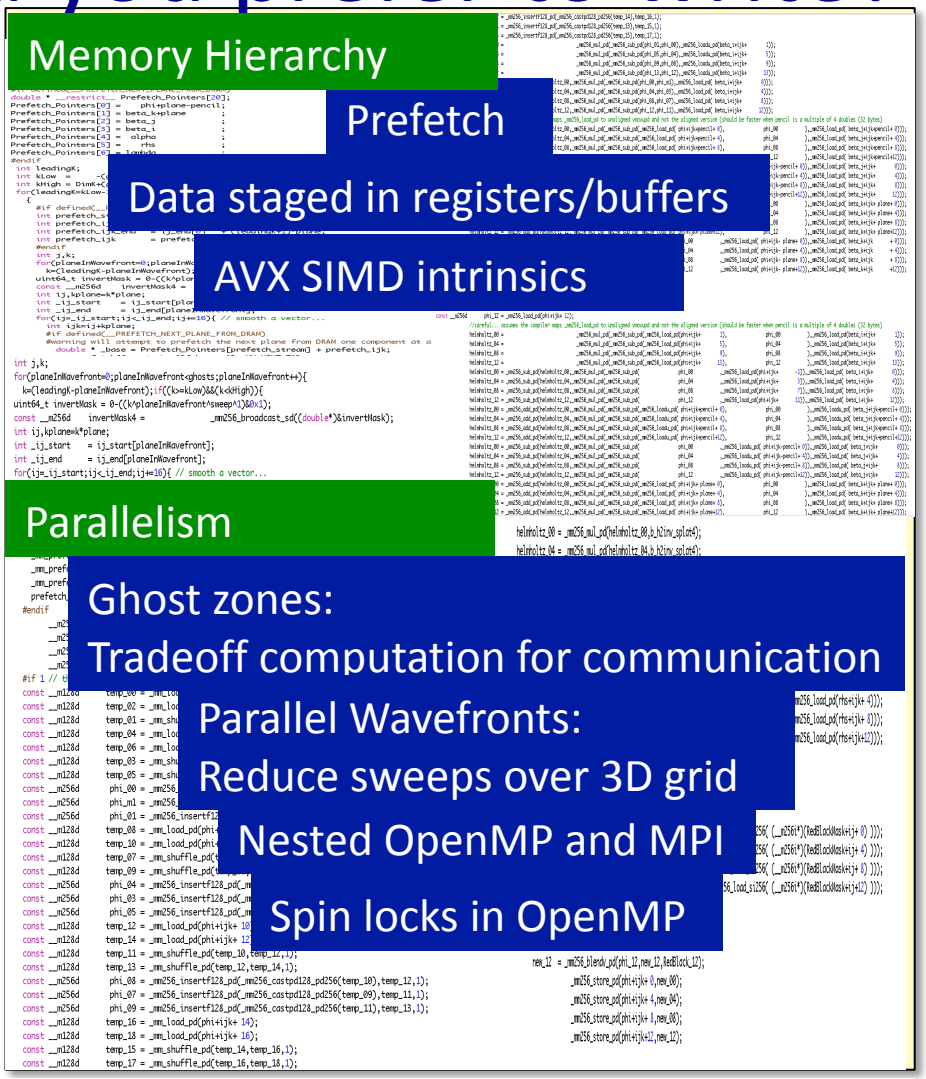
Which version would you prefer to write?

```
/* 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]);}
```

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

Goal of Research

Programmer writes

```
/* 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_i[k][j+1][i] * ( phi[k][j+1][i] – phi[k][j][i] )
       -beta_i[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][i][i] = a * alpha[k][i][i] * phi[k][i][i] –
                      temp[k][i][i];
```

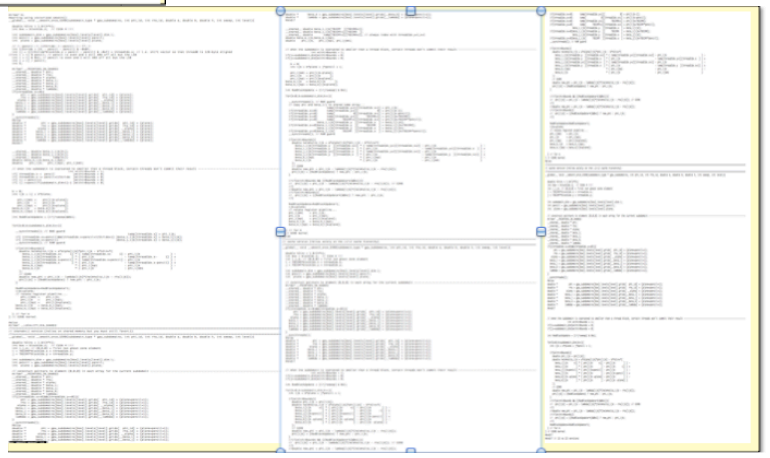
```
/* Gauss-Seidel Red Black Update */
for (k=0; k<N; k++)
  for (j=0; j<N; j++)
    for (i=0; i<N; i++){
      if ((i+j+k+color)%2 == 0 )
      phi[k][i][i] = phi[k][i][i] – lambda[k][i][i] *
        (temp[k][i][i] – rhs[k][i][i]);}
```

Code A

Programming system derives

Code B (CPU)

Code C (GPU)



Also, Codes D, E and F….

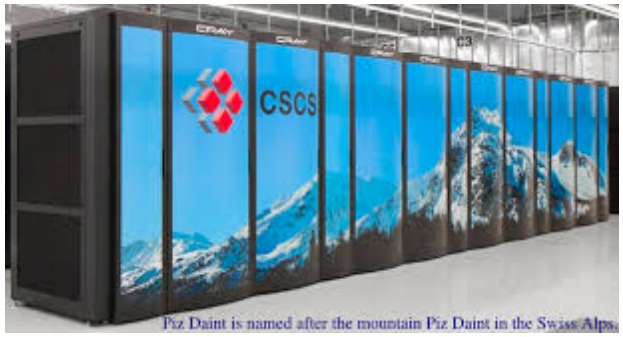
Theme 1: Performance Portability

Can the same program perform well on diverse supercomputing platforms? (e.g., Top 500 list, top500.org)

#1: Summit, IBM Power9+V100 GPUs

#3: TaihuLight, Sunway

#4: Tianhe-2, Intel Xeon Phis



#6: Piz Daint, Intel Xeon+P100 GPUs

#8: ABCI Intel Xeon Gold And V100 GPUs

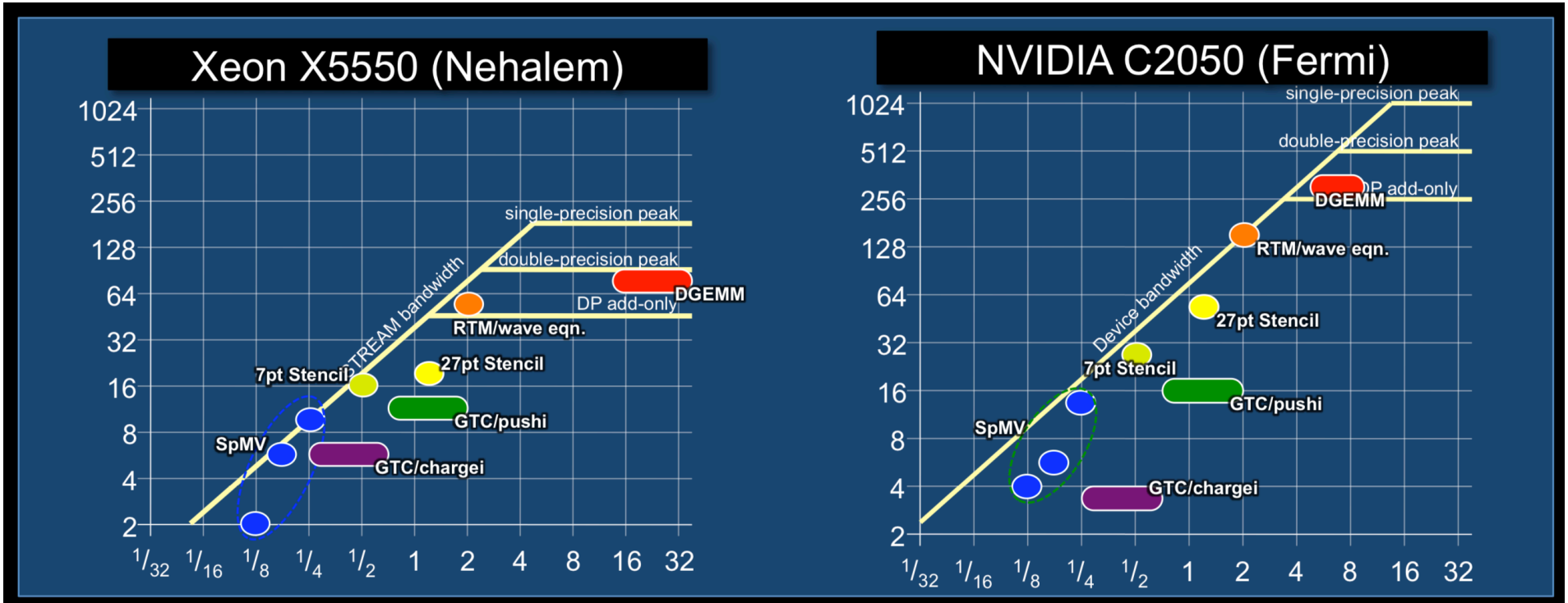
What's Coming Next?

Fugaku (Riken), ARM + custom optimizations

Aurora, Intel Xeon + Intel X Compute

Frontier, AMD EPYC CPU + AMD GPU

Theme 2: Data Movement



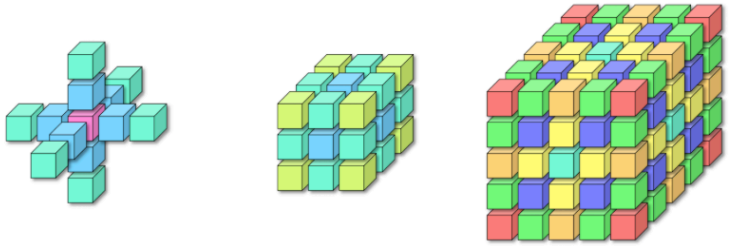
Communication wall will get worse (dominates energy and time)

- Optimizing for memory/network more important than ever
- Automatic data movement (caches, VM) can be wasteful
- Autotuning (search) helps reach bandwidth limits

Slide source: Kathy Yelick, UC Berkeley, **More Data, More Science and... ! Moore's Law, 2015.**

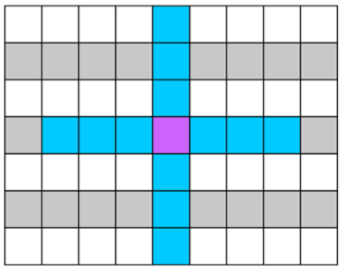
Stencil Computations

- Solve partial differential equations
  - Points are computed using neighbors
- Low order stencil
  - Lower accuracy
  - Low arithmetic intensity (FLOP per byte) typically memory bound
- **High order stencil**
  - High arithmetic intensity and could be compute bound
  - Conventional wisdom: memory optimization is not as important
- Diameter of stencil related to order of stencil
  - Low order stencil - smaller diameter
  - High order stencil - larger diameter

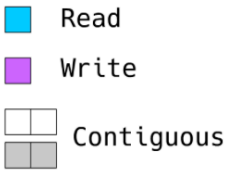


Data Movement Arising from Stencils

- Hardware prefetching streams
- TLB entries
- Worst case usage (cube-shaped):
  - ~ diameter in 2D
  - ~ diameter^2 in 3D

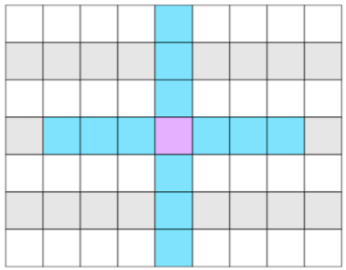


- Usage limits parallelism
  - e.g. number of threads < streams
- Problem exacerbated with tiling
- Tiling factors are architecture specific
  - Size of cache, page size, number of prefetching stream

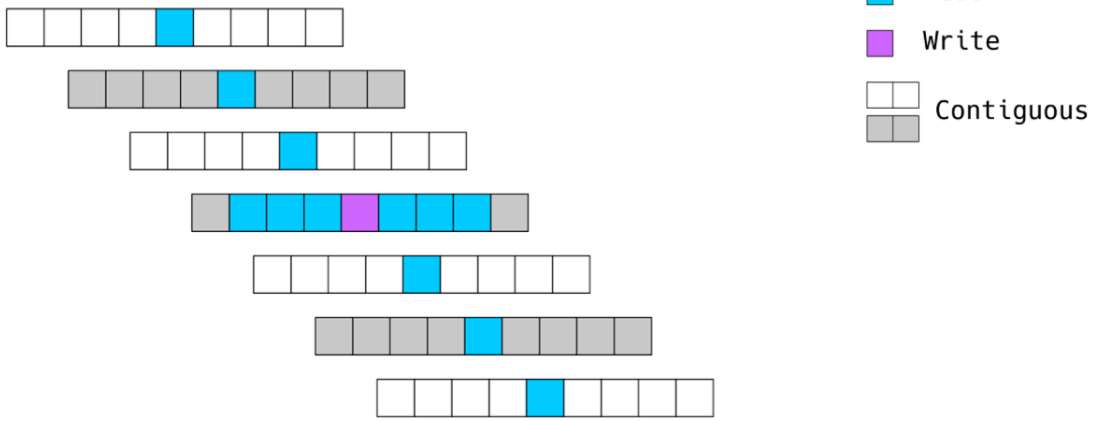


Data Movement Arising from Stencils

- Hardware prefetching streams
- TLB entries
- Worst case usage (cube-shaped):
  - ~ diameter in 2D
  - ~ diameter^2 in 3D



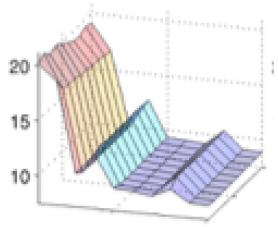
- Usage limits parallelism
  - e.g. number of threads < streams
- Problem exacerbated with tiling
- Tiling factors are architecture specific
  - Size of cache, page size, number of prefetching stream



Approach #1: Code A to Codes B & C

- Extended compiler transformation and code generation framework with ***domain-specific specialization*** (supports C-like C++)
  - Target is loop-based scientific applications and related tensor computations such as CNNs
  - ***Composable*** transformations
- Optimization strategy can be specified or derived with ***transformation recipes***
  - Also optimization parameters exposed
  - ***Separates code from mapping!***
- ***Autotuning***
  - Systematic exploration of alternate transformation recipes and their optimization parameter values
  - Search technology to prune combinatorial space

```
for (i=0;i<N;i++) {
   for (j=1;j<M;j++) {
S0:    a[i][j] = b[j] –
                 a[i][j-1];
I = {[i,j] | 0<=i<N ∧
             1<=j<=M}
```



Transformation Recipes, Codes B & C

/* 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)
fuse([2,3,4,5,6,7,8,9],4)
```

/* 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*N,_beta_i=N*N*N*N,
_phi=N*N*N*N},{block={"ii","jj","box"}, thread={"i","j"}},{})
```

Similar Idea is Gaining Traction in 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.

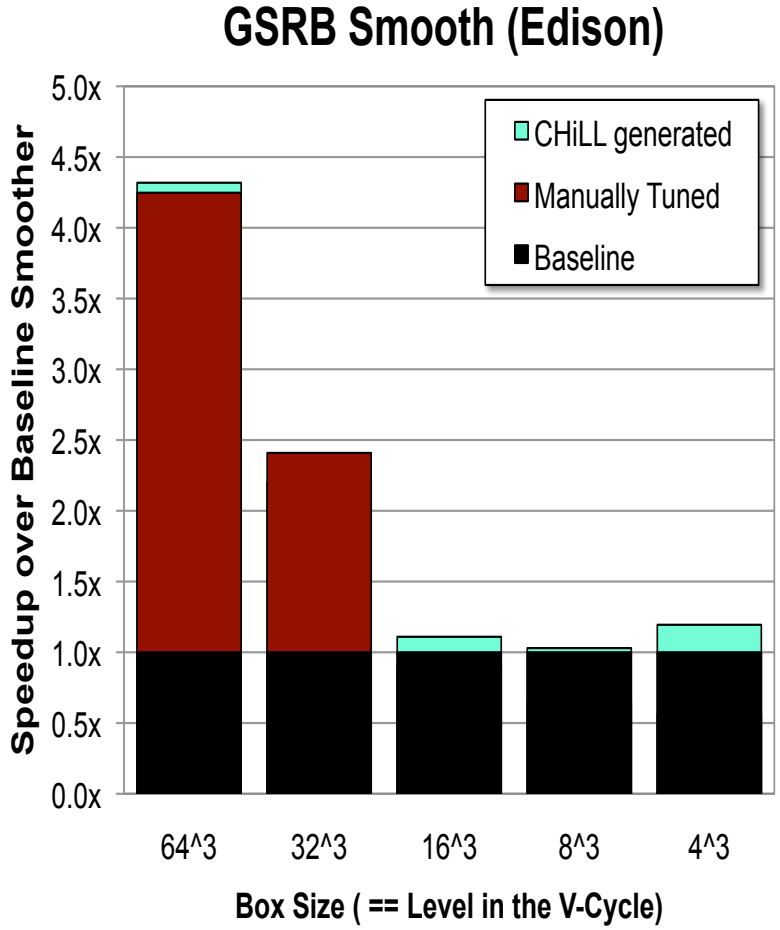
vectorize(*var*)

Vectorize the iteration.

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

Communication Avoiding: Sometimes Code A Beats Code B!

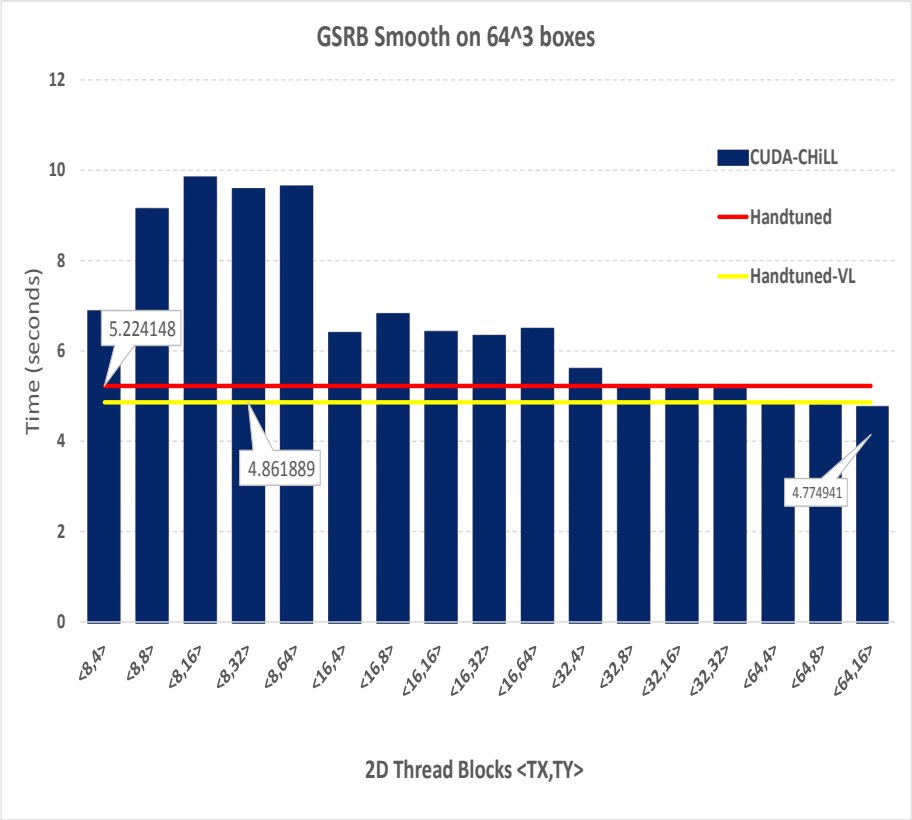
- miniGMG w/CHiLL
  - Fused operations
  - Communication-avoiding wavefront
  - **Parallelized (OpenMP)**
- **Autotuning** finds the best implementation for each box size
  - wavefront depth
  - nested OpenMP configuration
  - inter-thread synchronization (barrier vs. point-to-point)
- For fine grids (large arrays) CHiLL attains nearly a **4.5x speedup** over baseline



Basu et al., HiPC 2013, IPDPS 2015.

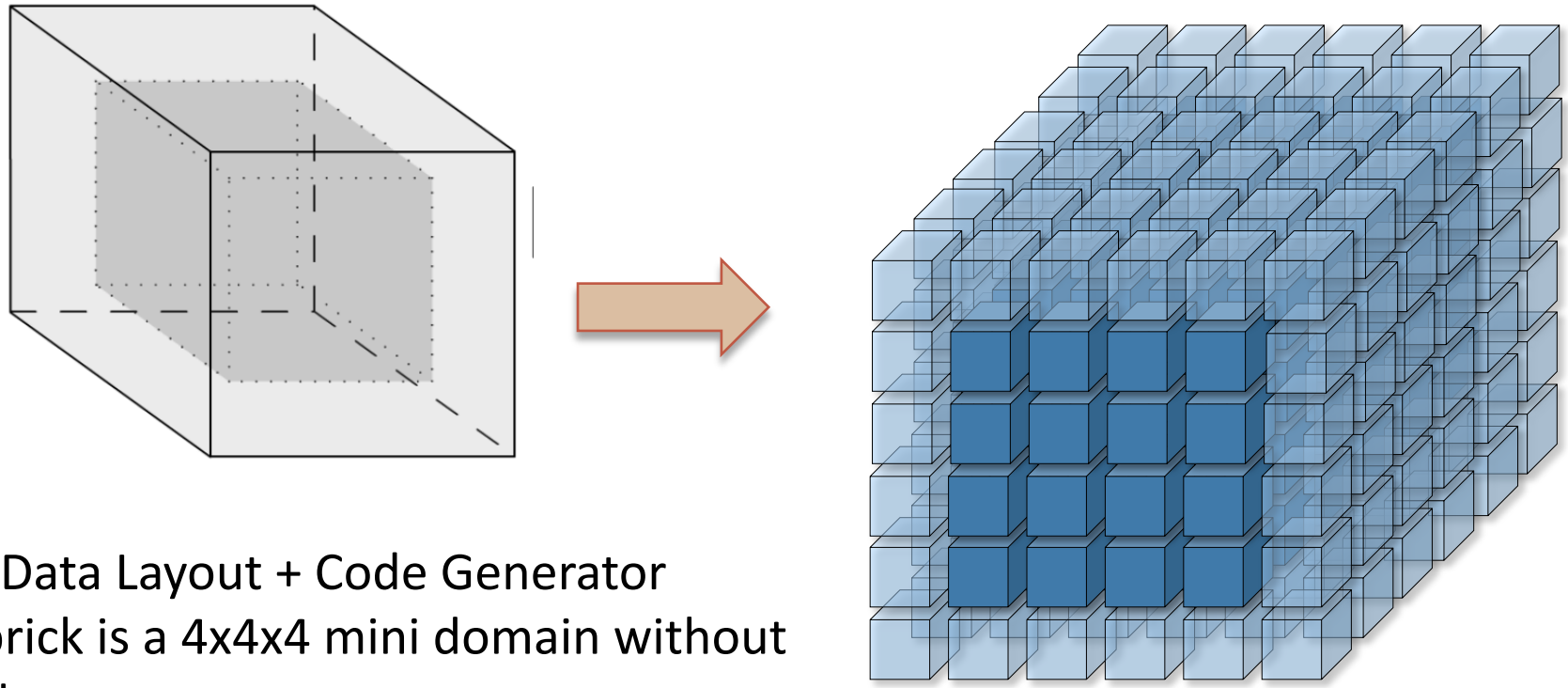
Retargetable and Performance Portable: Optimized Code A can beat Code C!

- CHiLL can obviate the need for architecture-specific programming models like CUDA
  - CUDA-CHiLL took the sequential GSRB implementation (.c) and **generated CUDA** that runs on NVIDIA GPUs
  - CUDA-CHiLL autotuned over the thread block sizes and is ultimately **2% faster** than the hand-optimized minigmg-cuda (**Code C**)
  - Adaptable to new GPU generations



Basu et al., PARCO 2017.

Approach #2: Code A to Codes B & C



Brick Data Layout + Code Generator

- A brick is a 4x4x4 mini domain without a ghost zone
- Application of a stencil reaches into other bricks (affinity important)
- Implemented with contiguous storage and adjacency lists

Code A uses Brick Domain-Specific Library

- Bricks are programmed using brick library for 3D stencils
  - Creation
  - Deletion
  - Access
- Brick library handles cases when access across brick boundary
- Vector code generation is carried out by a code generator

Array

```
float c = prev[k][j][i] * coeff[0] + (
  prev[k][j][i+1] + prev[k][j][i-1] +
  prev[k][j+1][i] + prev[k][j-1][i] +
  prev[k+1][j][i] + prev[k-1][j][i]) *
  coeff[1];
next[k][j][i] = c * vel[k][j][i];
```

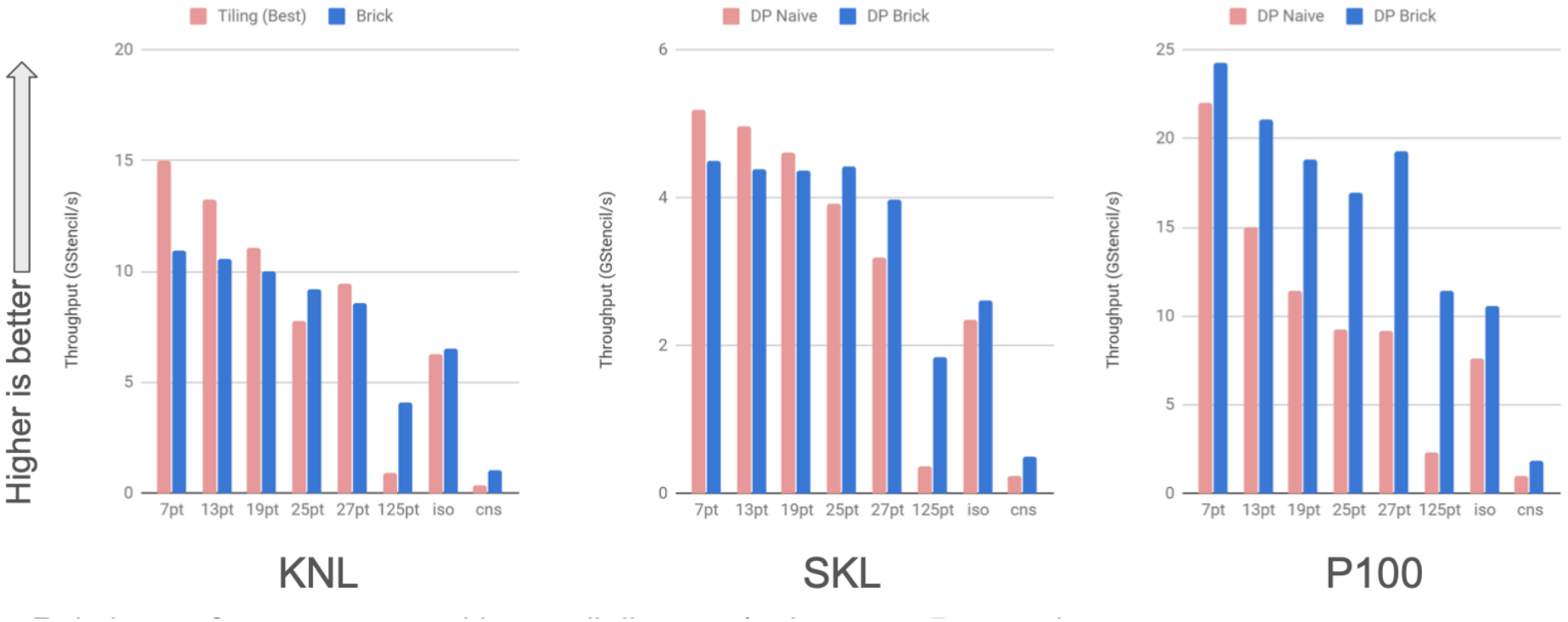
Brick

```
float c = prev.elem(b,k,j,i)*coeff[0]+(
  prev.elem(b,k,j,i+1)+prev.elem(b,k,j,i-1)+
  prev.elem(b,k,j+1,i)+prev.elem(b,k,j-1,i)+
  prev.elem(b,k+1,j,i)+prev.elem(b,k-1,j,i))*
  coeff[1];
next.elem(b,k,j,i)=c*vel.elem(b,k,j,i);
```

Bricks Address Themes 1 and 2

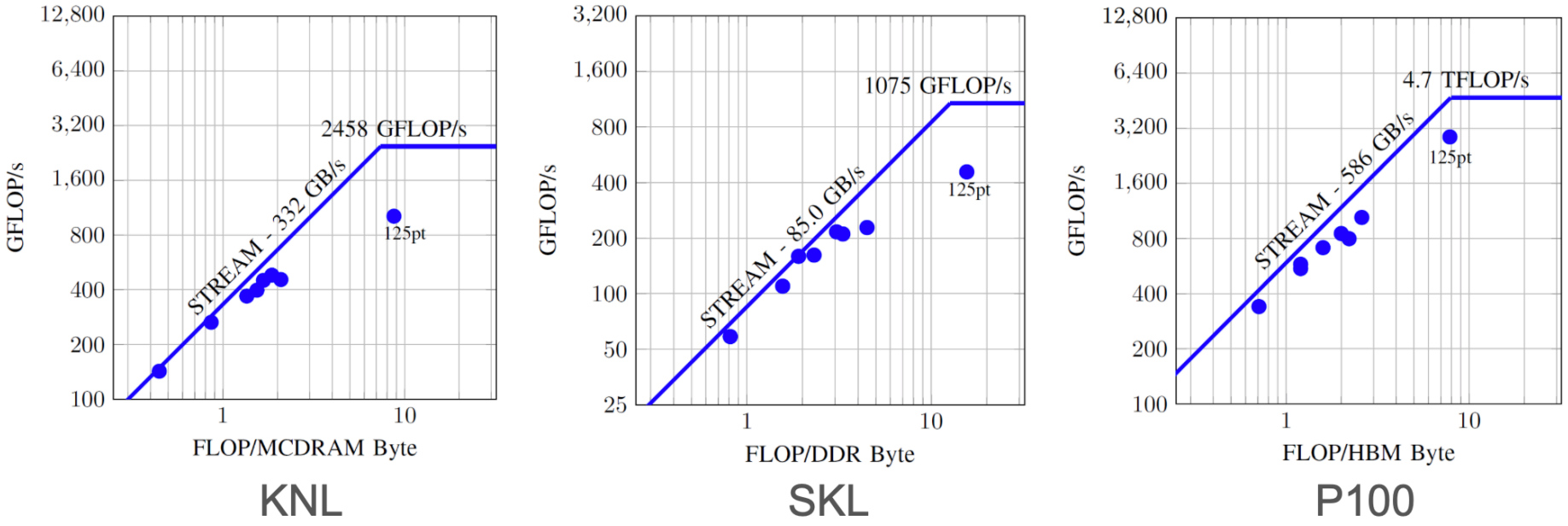
- Performance portability
  - Automation of architecture-specific code generation
  - Same abstraction, but different low-level instructions and “vector” widths
- Data movement
  - Contiguous storage of subdomain reduces overhead of automatic data movement (prefetch, TLB, cache)
  - Adjustable brick size adapts to node architecture limits
  - Indirection to represent neighbor lists gives freedom to adapt co-located bricks to architecture
    - (Ongoing) And to adapt layout to optimize communication

Performance Results (Node)



- Bricks achieve best performance for higher-order stencils, up to 5X!
- Always profitable on P100

Roofline Performance Results



- Bricks achieve performance close to memory bandwidth limit
- 125pt stencil approaches compute limit, has non-float operations

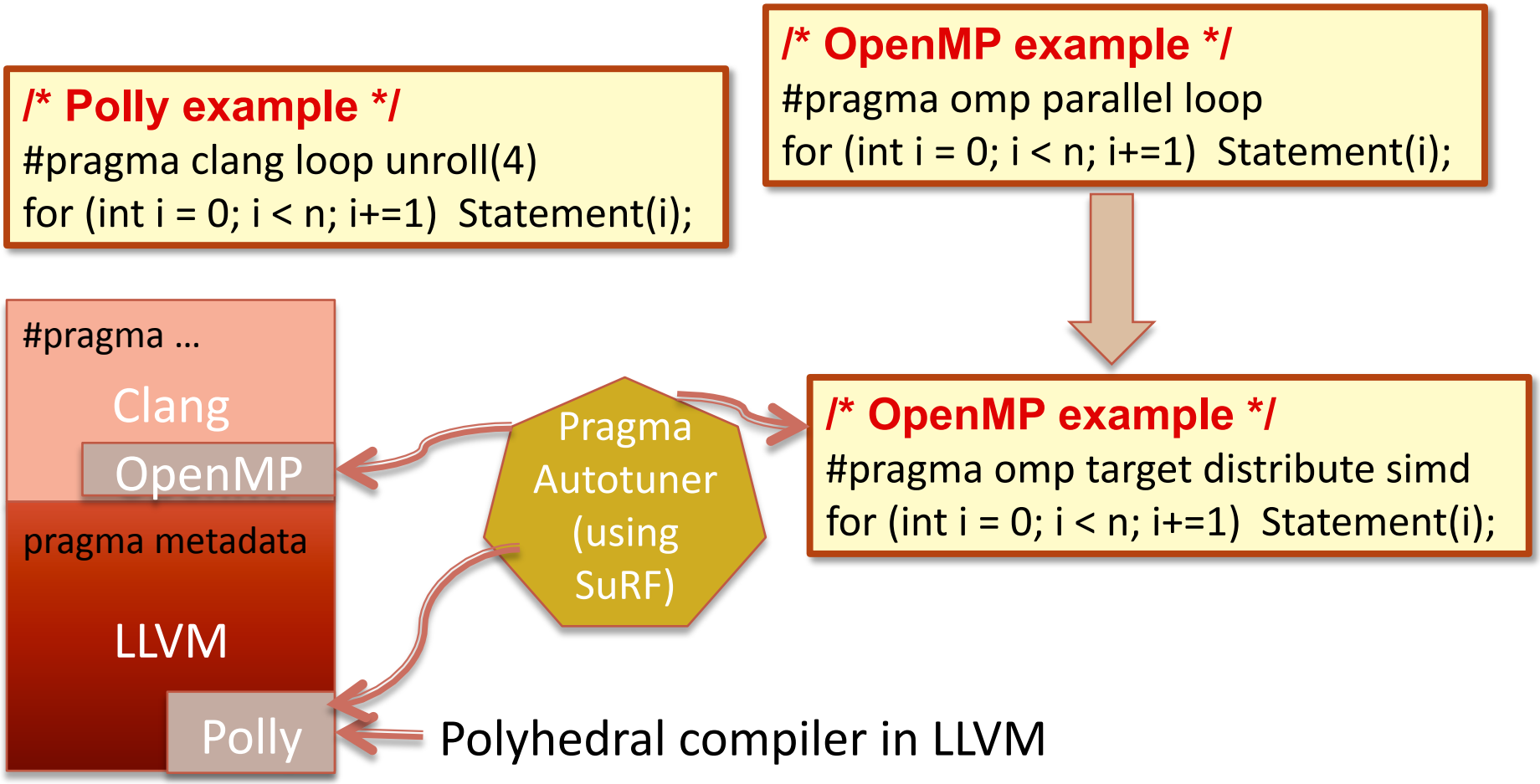
Zhao et al., PP3HPC 2018.
Zhao et al., SC19.

More on Autotuning Research: Automating Finding Codes B and C

- Bricks
  - What brick size?
  - How many bricks per core? Per node?
- Program transformations
  - Which transformations to use?
  - Parameters to optimizations, such as tile size?
- Other things to tune
  - Pragmas, e.g., OpenMP
  - Application parameters, e.g., in a library like SuperLU

Pragma Autotuner

- Search Using Random Forest (SuRF) for autotuning search (may not involve compiler)



Autotuning Barriers to Adoption in HPC

- Overhead
  - Tuning search can be expensive
  - Off-line tuning expensive, programmer burden
  - Specifying search space, transformations
  - Selection and configuration of algorithms
- Scope
  - Tuning must be repeated for new execution contexts
  - Exascale resources vary during execution, platform may not be available for training
  - Economies of data scale: Learning based on a community's code
- Other programmer concerns
  - Correctness concerns with dynamically-changing code
  - Long-term tool availability

Autotuning in High-Performance Computing Applications, Balaprakash, Dongarra, Gamblin, Hall, Hollingsworth, Norris, Vuduc, Special Issue of Proceedings of the IEEE, Nov. 2018.

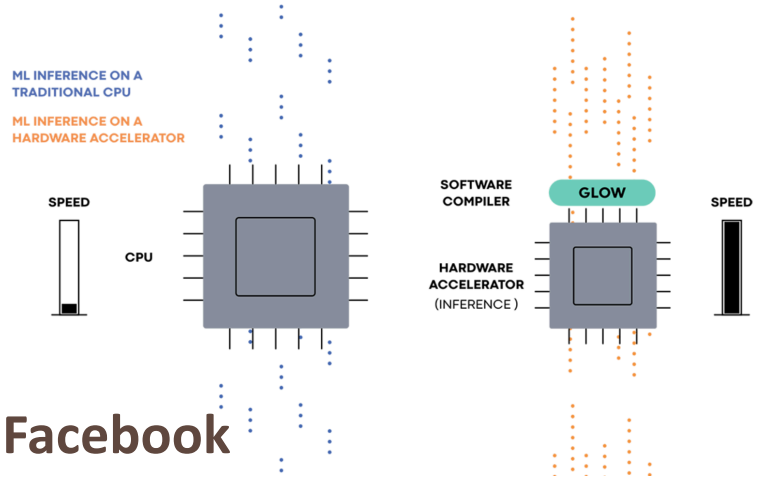
Conclusion

- The plethora of architecture technologies will make programming future supercomputers even more of a nightmare
- Programming system technology is desperately needed to address programmer productivity
  - Separating specification from architecture mapping
  - Architecture-specific code generation
  - Autotuning
- HOW TO BUILD THIS TECHNOLOGY???

Theme 3: Leveraging Investment in Deep Learning Compilers

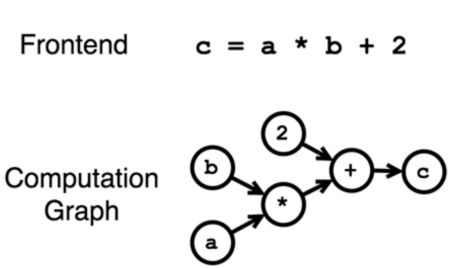
POSTED ON SEP 13, 2018 TO AI RESEARCH, ML APPLICATIONS

Glow: A community-driven approach to AI infrastructure

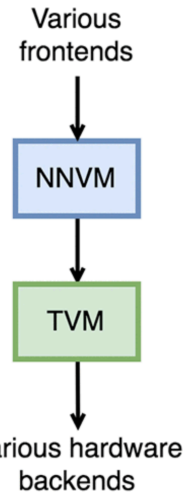


Facebook

A typical framework



NNVM Compiler



Amazon

Google

MLIR: Multi-Level Intermediate Representation Compiler Infrastructure

2019 European LLVM Developers Meeting

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];
} } } } } } }
```