Automating Compiler-Directed Autotuning for Phased Performance Behavior

Tharindu Rusira[†], Mary Hall[†], Protonu Basu[*]

[†]School of Computing, University of Utah

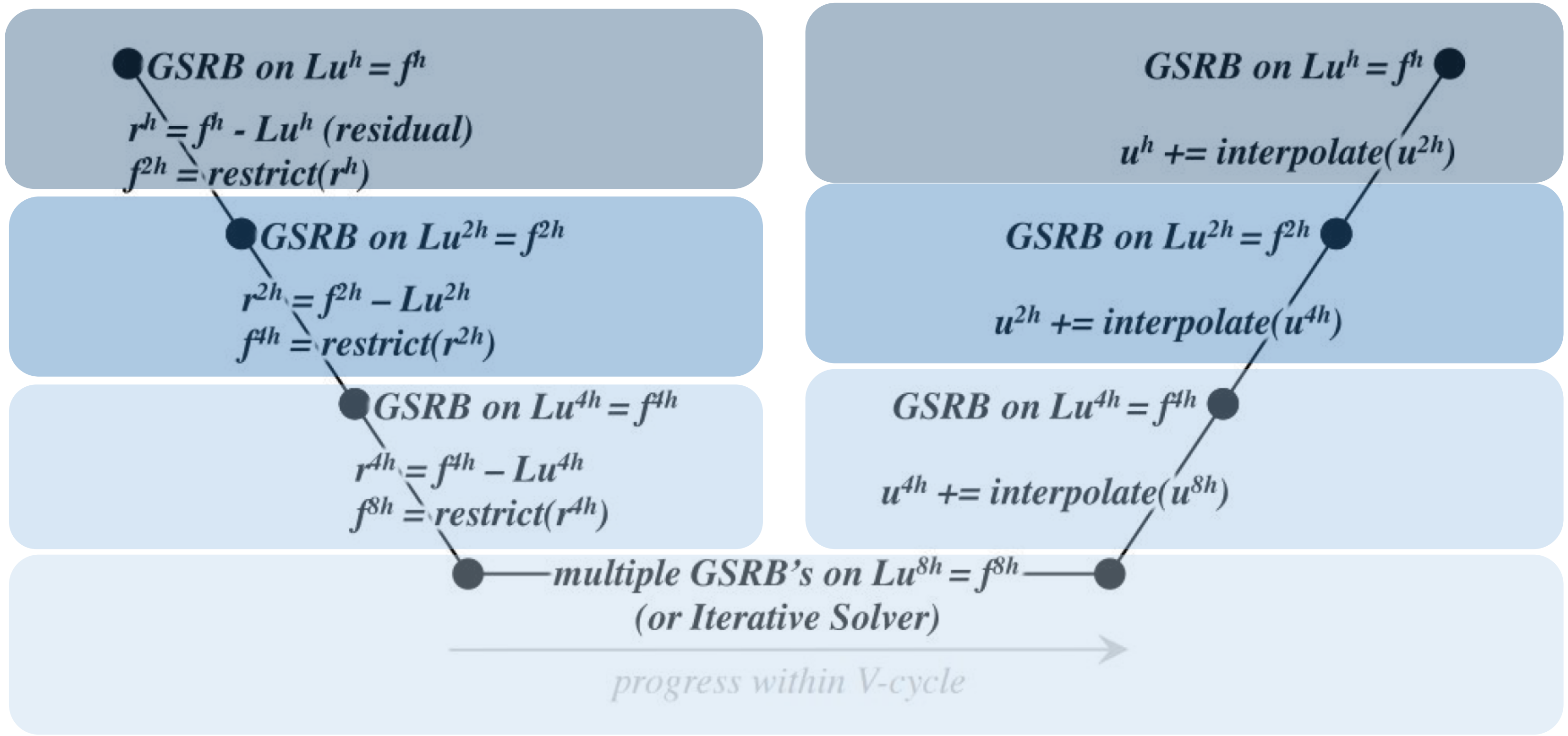
[*] Lawrence Berkeley National Laboratory

Outline

- GMG and phased performance behavior
- Domain-specific optimizations
- Superscripts
- Autotuning
- Results
- Future work

GMG

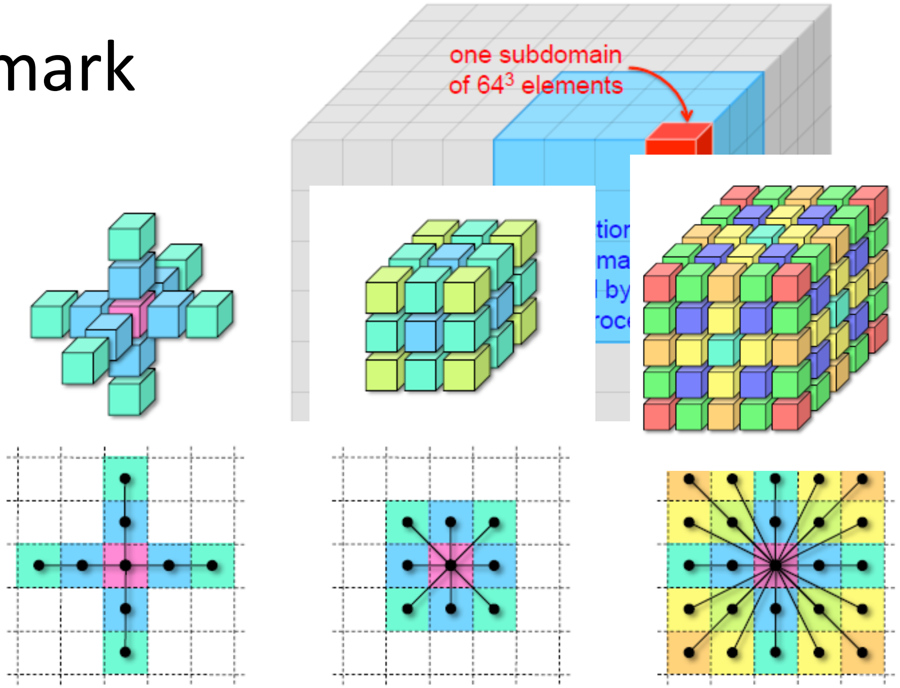
- V-cycle – Hierarchical linear solver with varying grid resolutions



[Image] S. Williams, D. D. Kalamkar, A. Singh, A. M. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker. Optimization of geometric multigrid for emerging multi-and manycore processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 96. IEEE Computer Society Press, 2012.

GMG and stencils

- **miniGMG**[1], Geometric Multigrid benchmark
- Domain decomposition
- Higher order stencils (13pt, 27pt, 125pt)



[1] Williams, Samuel. "Implementation and optimization of miniGMG-a compact geometric multigrid benchmark." (2014).

Motivation

- Each GMG level has different performance requirements
- Compiler has to decide which optimizations to apply
- Many performance factors
  - Architecture
  - GMG level (input size)
  - GMG operator (computation)
  - Stencil
  - Interaction between operators

Smooth operator (Jacobi)

- Performance dominant operation in V-cycle (multiple applications of ‘smooth’ at each level, usually communication-bound)
- S0, S1, S2 involve stencil computations and memory updates of two grids **temp** and **phi**

```
// statement S0
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]));
```

```
// statement S1
temp[k][j][i] = a * alpha[k][j][i] * phi[k][j][i]-temp[k][j][i];
```

```
// statement S2
phi[k][j][i] = phi[k][j][i]-lambda[k][j][i]*(temp[k][j][i]-rhs[k][j ][i]);
```

```
if(smooth_application % 2 == 0)
   {

  for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
       even_S0();

  for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
       even_S1();

  for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for(i=0;i<N;i++)
       even_S2();
 } else{

   for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
       odd_S0();

  for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
       odd_S1();

  for (k=0;j<N;k++)
    for (j=0;j<N;j++)
      for(i=0;i<N;i++)
       odd_S2();
 }
```

Optimizations

- Loop fusion
- Wavefront computation
  - With deep ghost zones (halo regions)
- Partial sums
- Nested parallelism (OpenMP)

Loop fusion

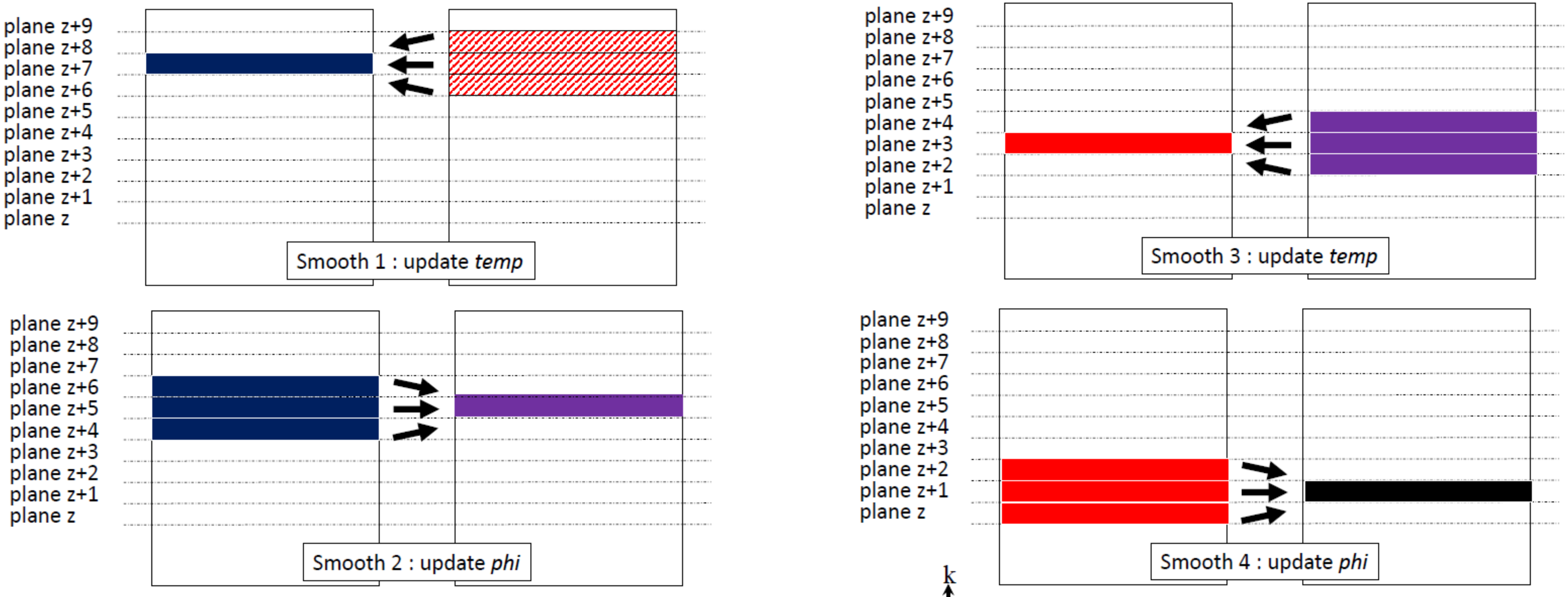
Reduces vertical (across memory hierarchy) memory traffic

```
if(smooth_sweep % 2 == 0){
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
        even_S0(); //Laplace
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
        even_S1(); //Hemholtz
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for(i=0;i<N;i++)
        even_S2(); //Jacobi relaxation
} else{
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
        odd_S0();
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++)
        odd_S1();
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for(i=0;i<N;i++)
        odd_S2();
}
```

```
if(smooth_sweep % 2 == 0){
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++){
        even_S0(); //Laplace
        even_S1(); //Hemholtz
        even_S2(); //Jacobi relaxation
}} else{
  for (k=0;k<N;k++)
    for (j=0;j<N;j++)
      for (i=0;i<N;i++){
        odd_S0();
        odd_S1();
        odd_S2();
}}
```

Wavefronts

Reduces vertical memory traffic by simultaneously running multiple smooths

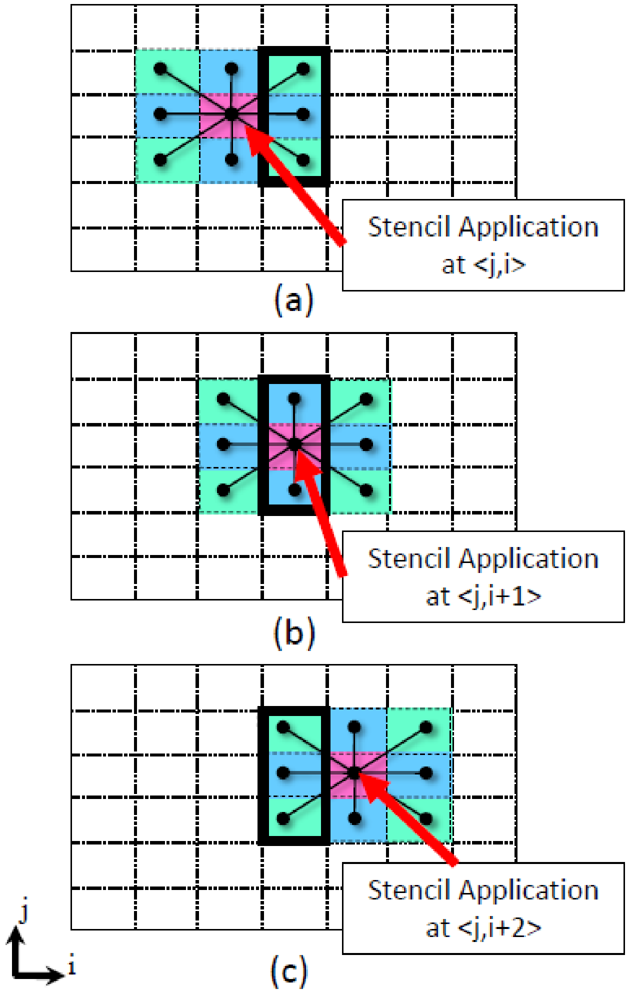


Basu, Protonu. *Compiler optimizations and autotuning for stencils and geometric multigrid*. Diss. The University of Utah, 2016.

Partial sums

2D 9pt stencil

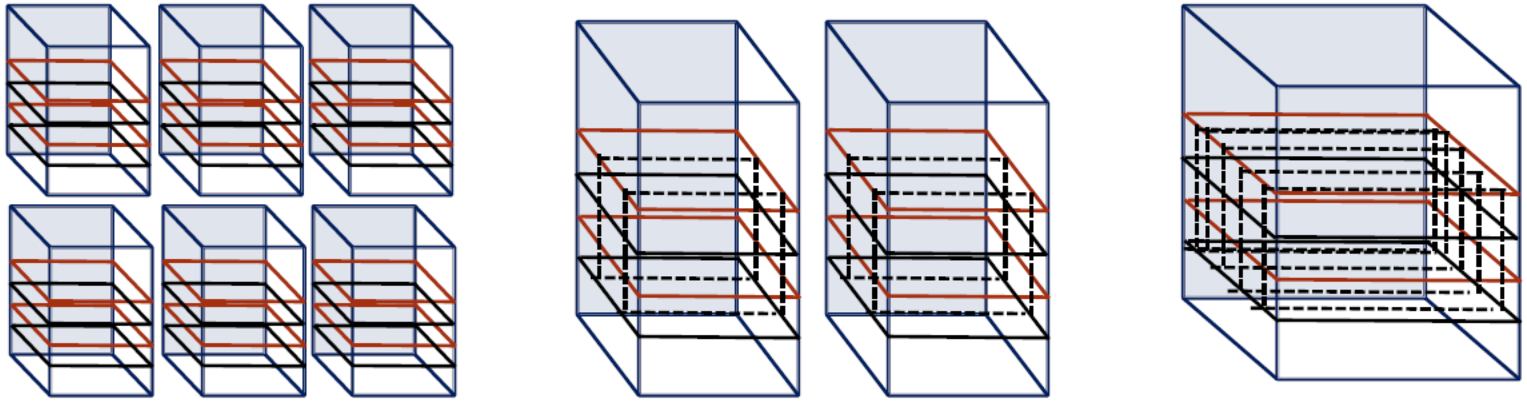
```
for (j=0; j<N; j++)
  for (i=0; i<N; i++)
        out[j][i] = w1*(
                in[j-1][i  ] + in[j+1][i  ] +
                in[j  ][i-1] + in[j  ][i+1]
        ) +
                w2*(
                in[j-1][i-1] + in[j+1][i-1] +
                in[j-1][i+1] + in[j+1][i+1]
        ) +
        w3* (in[j  ][i  ] );
```



Nested Parallelism (Collaborative Threading)

6 threads working in parallel

<X,Y> : X boxes, Y threads/box



Inter-Box Parallelism
Thread Configuration <6,1>

Nested Parallelism
Thread Configuration <2,3>

Intra-Box Parallelism
Thread Configuration <1,6>

Code variants and Search space

- Search space for 125pt stencil

  Wavefront size = {2,4,8}

  Partial sums = {ON, OFF}

  Collaborative threading = {1x12, 2x6, 3x4, 4x3, 6x2, 12x1}

  Smooth variants for a level ($>4^3$) = 3 x 2 x 6 = 36

  Smooth variants for bottom layer (4^3) = 2 x 2 x 6 = 24

  Total smooth variants for the V-cycle = 36^4 x 24 = **40, 310, 784**

- Question! How do we explore the search space?

CHiLL loop transformations

```
original()
skew ([0,1,2,3,4,5], 2, [3,1])
permute ([2,1,3,4])

partial_sums(0)
partial_sums(5)

fuse([0,1,2,3,4,5,6,7,8,9], 4)

omp_par_for(4,3)
```

[CHiLL] C. Chen, J. Chame, and M. Hall. CHiLL: A framework for composing high-level loop transformations. Technical report, Citeseer, 2008.
[Partial sums] Basu, Protonu, et al. "Compiler-directed transformation for higher-order stencils." Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International. IEEE, 2015.

CHiLL superscripts

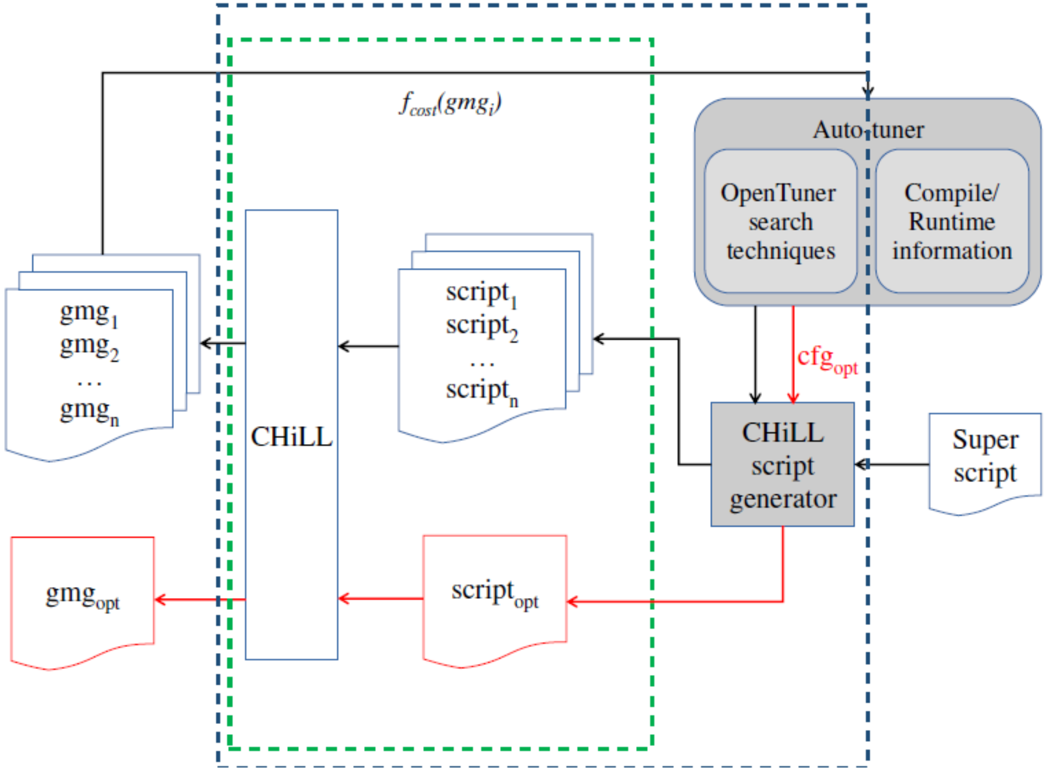
```
original()
skew ([0,1,2,3,4,5], 2, [3,1])
permute ([2,1,3,4])

partial_sums(0)
partial_sums(5)

fuse([0,1,2,3,4,5,6,7,8,9], 4)

omp_par_for(4,3)
```

```
original()
skew ([0,1,2,3,4,5], 2, [3,1])
permute ([2,1,3,4])

partial_sums(0)
partial_sums(5)

fuse([0,1,2,3,4,5,6,7,8,9], 4)

@begin_param_region
param(x,enum,[1,2,3,4,6,12])
param(y,enum,[1,2,3,4,6,12])
constraint(x*y==12)
omp_par_for(x,y)
@end_param_region
```

```
import chillsuper as cs
cs.generate_parameter_domain('superscript')
```

https://github.com/TharinduRusira/CHiLLsuper

Autotuning




```python
wav = ['wav0','wav1','wav2','wav3','wav4']
collab = ['c0','c1','c2','c3','c4']
ps = ['ps0','ps1','ps2','ps3','ps4']
def manipulator(self):
  manipulator = ConfigurationManipulator()
  for x in collab:
    manipulator.add_parameter(EnumParameter(x,['1x12','2x6','3x4','4x3','6x2','12x1']))
  for y in wav:
    if y is 'wav4':
      manipulator.add_parameter(EnumParameter(y,[2,4]))
    else:
      manipulator.add_parameter(EnumParameter(y,[2,4,8]))
  for z in ps:
    manipulator.add_parameter(EnumParameter(z,['on','off']))
  return manipulator
```

Code variant evaluation

- Total ‘smooth’ time

$$f_{cost} = \sum_{i=0}^{levels} t_s^i$$

- Customizable cost function

Tuned configurations

TABLE I: Tuned configuration for 13pt stencil, # of tests=1454

box size	64^3	32^3	16^3	8^3	4^3
fused	Yes	Yes	Yes	Yes	Yes
wavefront	No	No	No	No	No
partial sums	Yes	Yes	Yes	Yes	Yes
collab. threading	12×1	3×4	1×12	1×12	1×12

TABLE II: Tuned configuration for 27pt stencil, # of tests=1518

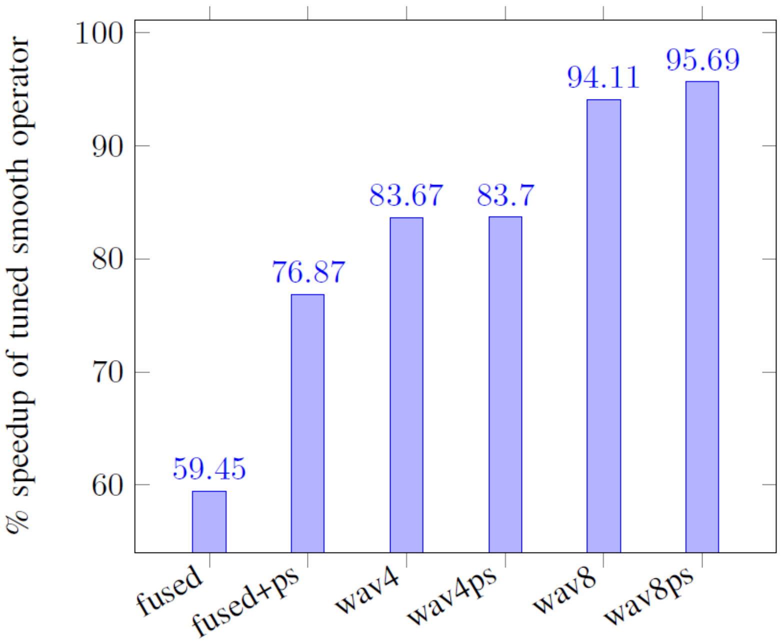
box size	64^3	32^3	16^3	8^3	4^3
fused	Yes	Yes	Yes	Yes	Yes
wavefront	No	No	No	No	2
partial sums	Yes	Yes	Yes	Yes	Yes
collab. threading	4×3	3×4	12×1	4×3	6×2

TABLE III: Tuned configuration for 125pt stencil, # of tests=1078

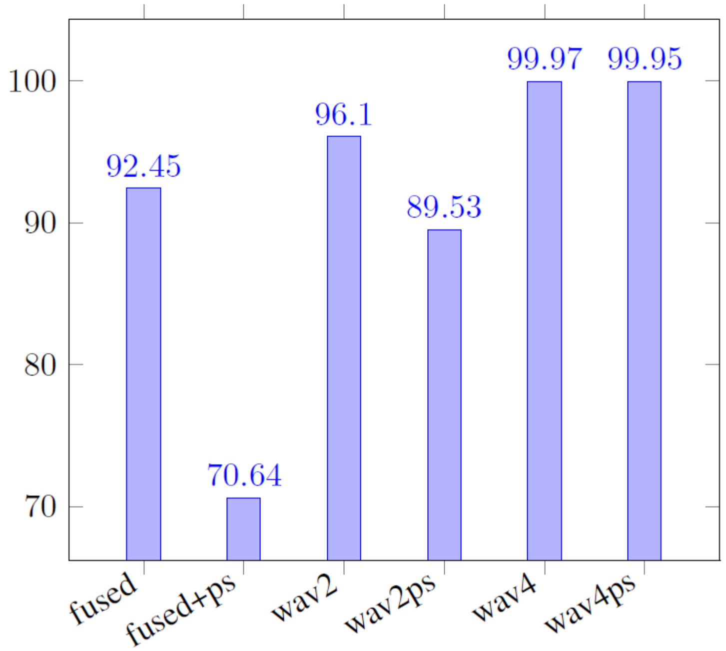
box size	64^3	32^3	16^3	8^3	4^3
fused	Yes	Yes	Yes	Yes	Yes
wavefront	No	No	No	No	No
partial sums	Yes	Yes	Yes	Yes	Yes
collab. threading	3×4	6×2	6×2	6×2	12×1

Results

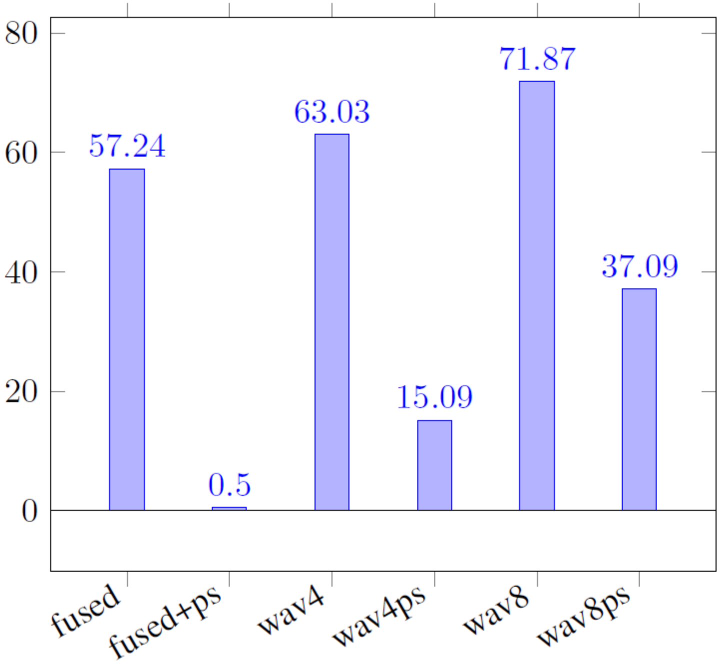
NERSC's Edison (8 MPI tasks, 4 nodes), ICC with -O3 -xAVX --D__COLLABORATIVE_THREADING=12 --D__MPI
GMG configuration: 512^3 full domain distributed among 8 tasks (256^3 subdomain each). 5 GMG levels with $64x64^3$ at the top level (64^3,32^3,16^3,8^3,4^3)



13pt Jacobi smooth



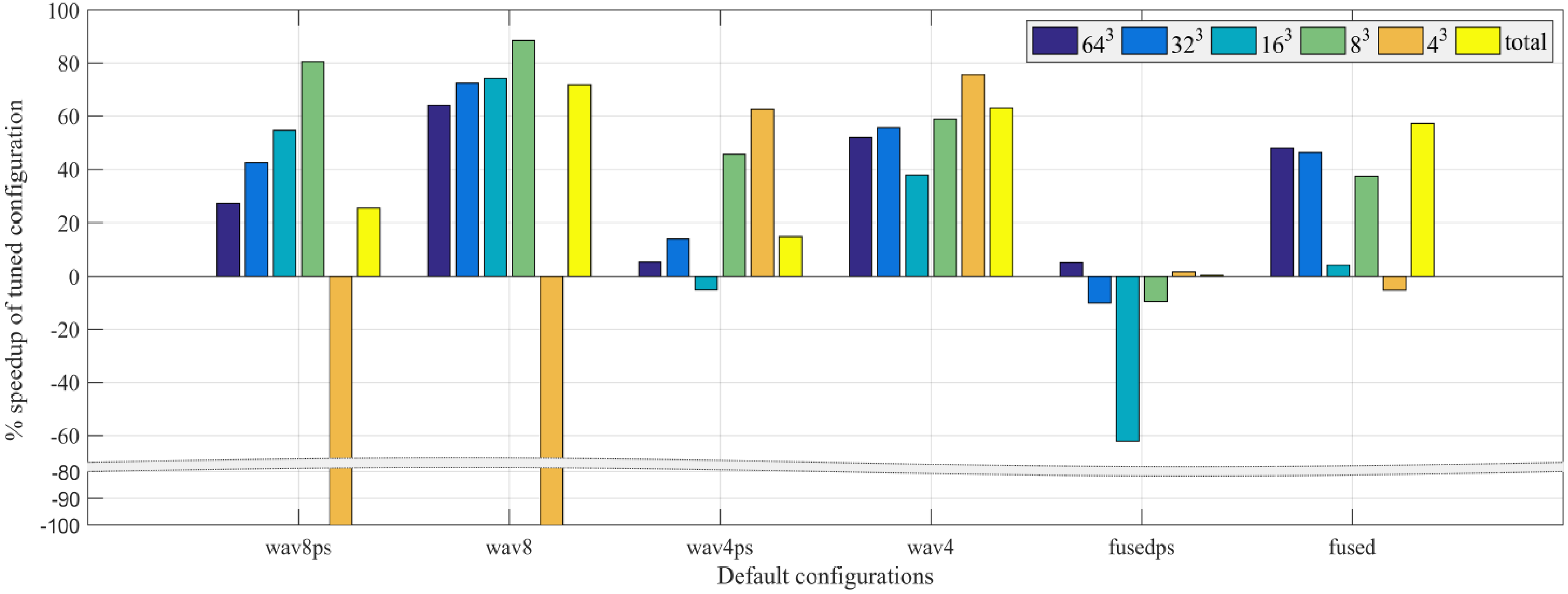
27pt Jacobi smooth



125pt Jacobi smooth

Effect of autotuning on phased performance

Comparison of speedups by each GMG level (64^3,32^3,16^3,8^3,4^3) and overall speedup of the *tuned configuration* against corresponding V-cycle levels in *default configurations*



Related Work

- [OpenTuner] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe. Opentuner: an extensible framework for program autotuning. In Proceedings of the 23rd international conference on Parallel architectures and compilation, pages 303–316. ACM, 2014.
- [ActiveHarmony+CHiLL] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. K. Hollingsworth. A scalable auto-tuning framework for compiler optimization. In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pages 1–12. IEEE, 2009.
- [Orio1] A. Hartono, B. Norris, and P. Sadayappan. Annotation-based empirical performance tuning using Orio. In *Proceedings of the 2009 IEEE international symposium on parallel & distributed processing*, pages 1–11. IEEE Computer Society, 2009.
- [Orio2] T. Nelson, A. Rivera, P. Balaprakash, M. Hall, P. D. Hovland, E. Jessup, and B. Norris. Generating efficient tensor contractions for gpus. In Parallel Processing (ICPP), 2015 44th International Conference on, pages 969–978. IEEE, 2015.

Future work

- Leverage support for arbitrary HPC applications
- Loop transformation sequences
- Better search space pruning with domain-specific information
- Guided search for fast convergence using analytical models/domain-specific heuristics