Compiler Technology and Autotuning in the Exascale Computing Project

Approved for public release

Mary Hall

University of Utah

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.

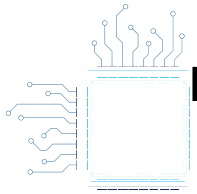
Learn More

- Exascale Computing Project:
  - https://exascaleproject.org/about/
- MLIR for High Performance Computing
  - https://www.cs.utah.edu/mlir4hpc/

What is the Exascale Computing Project (ECP)?

ECP is an accelerated research and development project funded by the US Department of Energy (DOE) to ensure all necessary pieces are in place to deliver the nation's first, capable, exascale ecosystem, including mission critical applications, an integrated software stack, and advanced computer system engineering and hardware components.

What is a “capable” exascale computing ecosystem?



Hardware

At least two diverse system architectures

Delivers 50x the performance of today’s 20 petaflop systems and 5x the performance of Summit, Oak Ridge National Laboratory’s supercomputer— i.e., allows at least a quintillion floating point operations per second

Functions with sufficient resiliency: an average fault rate of ≤1 per week

Software

Includes a software stack that meets the needs of a broad spectrum of applications and workloads

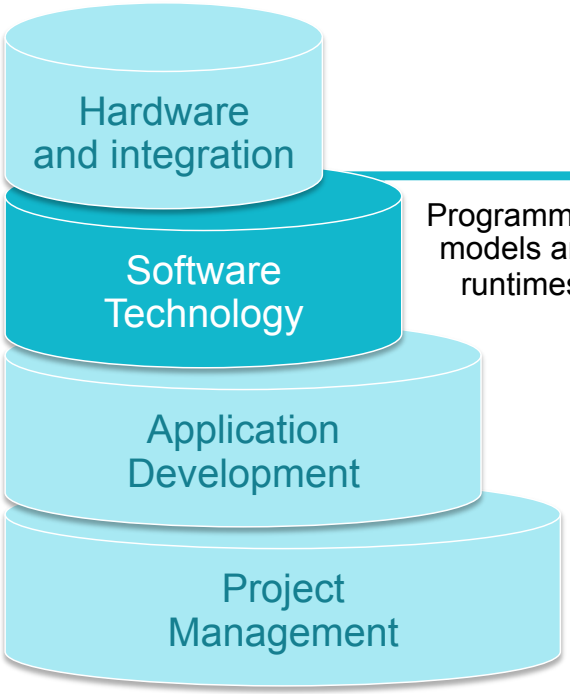
Applications

Supports a wide range of applications that deliver high-fidelity solutions in less time to problems of greater complexity

Exascale means real capability improvement in the science we can do, and how fast we can do it

Software Technology

Develop the exascale software stack and deliver using Software Development Kits (SDKs)



Programming models and runtimes | Development tools | Mathematical libraries | Data and visualization | Software ecosystem and delivery

ST Elements

- 55 WBS L4 subprojects executing RD&D
- ST efforts currently contribute to 89 unique products

Challenges: Qualitative changes; Massive concurrency; Multi-scale, multi-physics, data-driven science; Ecosystem integration

PROTEAS-TUNE Project Overview

PROTEAS-TUNE is not a one product project; it has multiple technical thrusts.

Challenges

- Performance portability is a critical challenge for sustainable HPC software
- Heterogeneous and manycore processors (GPUs, FPGAs, Manycore),
- Deep memory hierarchies incl nonvolatile memory systems (NVM);
- Consistent with recent expectations for FRONTIER, AURORA
  - But don't specialize

Solutions

- Programming systems
  - OpenACC, CUDA, OpenCL, OpenMP, SYCL
  - Optimization techniques and algorithms
  - Autotuning
- Open LLVM Compiler
  - OpenACC support, parallel IR, loop transformations
  - Contributions to upstream LLVM
- Portable, scalable performance analysis via TAU Performance System and its integration
- Strategies for algorithm design for new architectures: bricks
- Scalable approaches to using NVM, such as embedded KV stores (papyrus)

Impact

- Engage with ECP applications and software teams
  - Multiple engagements with AD and ST teams
- Engage with broader community
  - OpenACC Forum
  - OpenMP Forum
  - LLVM development community
  - Tutorials, Dev Meetings, etc
- Substantive connections with all vendors (even non-ECP vendors)
  - IBM, AMD, Intel, NVIDIA, Cray, etc

Project Team

- Oak Ridge National Laboratory
  - Jeffrey S Vetter (PI), Joel Denny, Jungwon Kim, Seyong Lee, Dick Glassbrook (PM)
- University of Oregon
  - Allen Malony (Co-PI), Sameer Shende,
  - Kevin Huck, Camille Coti
- Los Alamos National Laboratory
  - Kei Davis (Co-PI), David Ringo
- Argonne National Laboratory
  - Hal Finkel (Co-PI), Prasanna Balaprakash, Johannes Doerfert, Michael Kruse
- University of Utah
  - Mary Hall (Co-PI)
- Lawrence Berkeley National Laboratory
  - Sam Williams (Co-PI), Hans Johansen

- FY20 officially begins the merged project: PROTEAS + Y-TUNE -> PROTEAS-TUNE
- Operating as a combined team since March 2019 including AHM at Utah

Y-TUNE Toolkit: Distinguishing Features

Y-Tune Toolkit

Compiler transformation & code generation
Brick data layout for stencils
Autotuning Search (ytopt)
Pragma autotuner
Technology transfer to Clang/LLVM

Primary goal:
Single source performance portability for ECP codes using autotuning

CHiLL: Transformation & Code Generation

- CHiLL released and integrated into Spack
- Focus in Phase 2 shifted to LLVM and MLIR with CHiLL as prototype

Brick Data Layout for Stencils

- Fine-grain data blocking and cross-architecture code generation reduces data movement through hierarchical memory systems
- Architecture-specific code gen. makes bricks performance portable to CPU & GPU

Autotuning Search

- Search using Random Forests (SuRF) iteratively refines model in promising input region by obtaining new measurements at unevaluated input configurations

Pragma Autotuner

- Vary pragmas for compiler to investigate alternative implementations

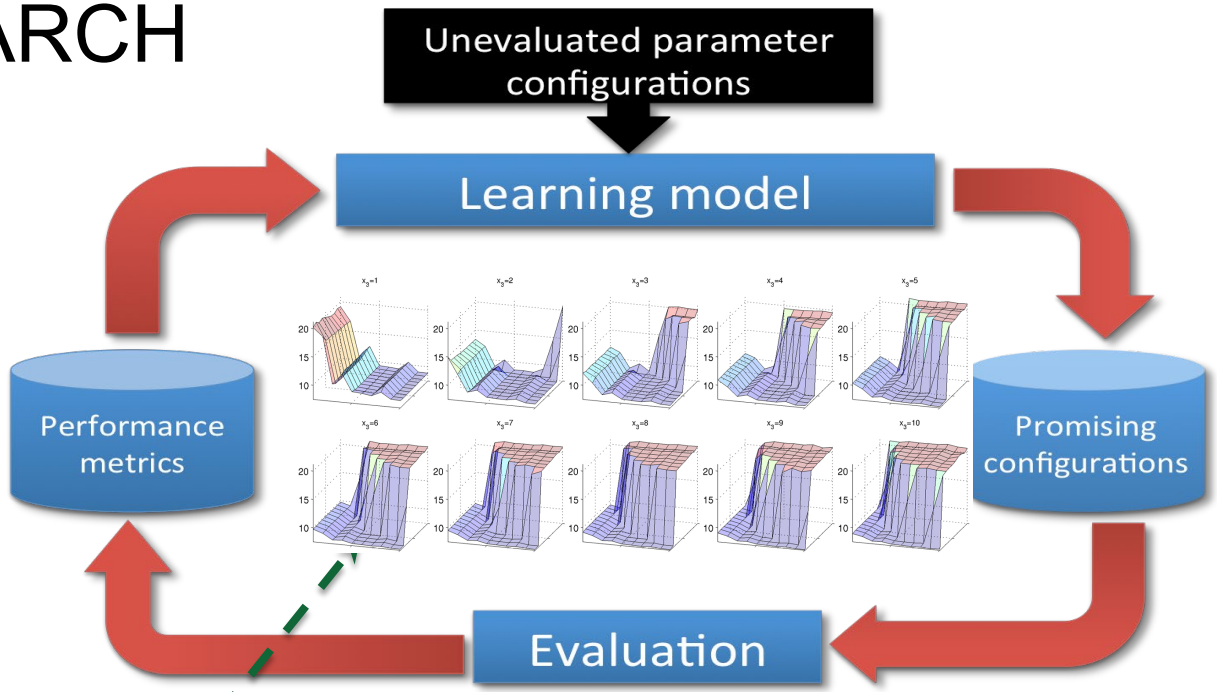
Technology Transfer to Clang/LLVM (and MLIR)

- Integrate transformation recipes into Clang/LLVM/Polly via transformation pragmas

YTOPT/SURF: AUTOTUNING SEARCH

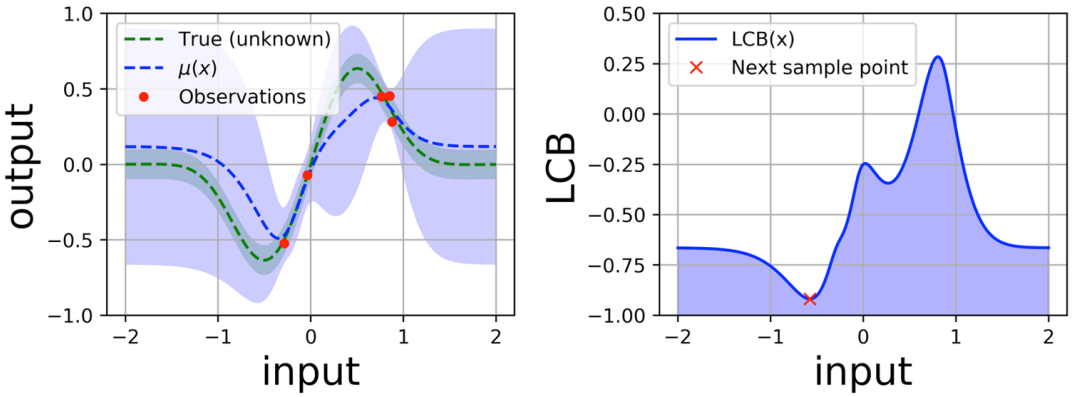
- Framework:
  - Initialization phase
    - Random or Latin hypercube sampling
  - Iterative phase
    - Fit model
    - Sample using the model

https://github.com/ytopt-team/ytopt



Example Surrogate Model Fitted to Sampled Performance

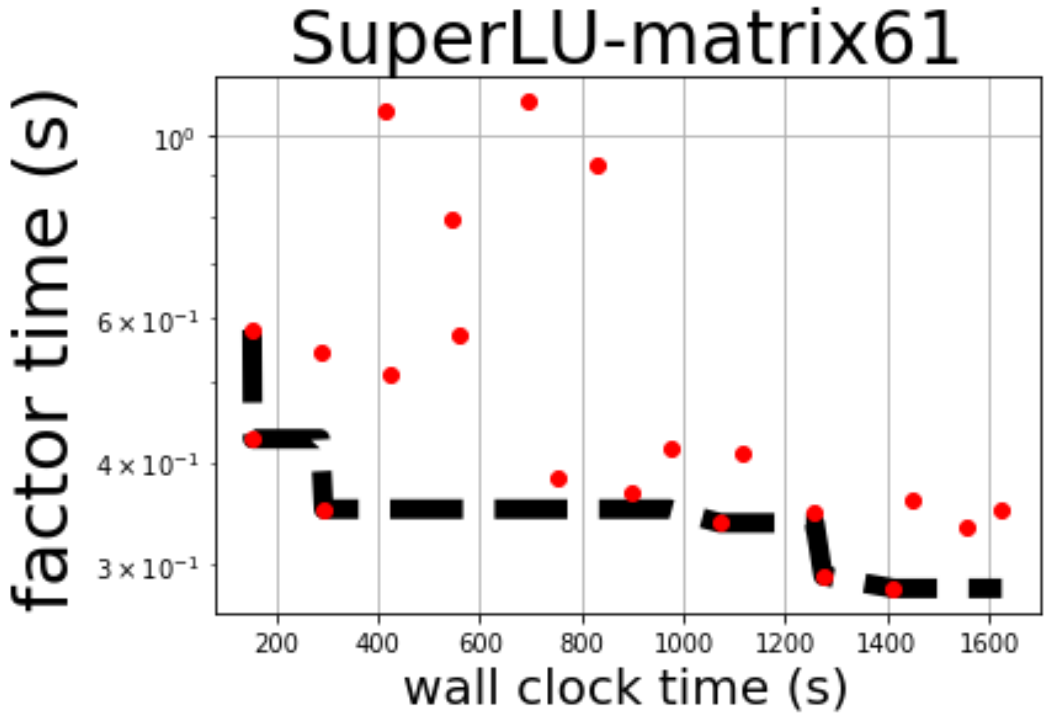
(iterative refinement improves the learning model)

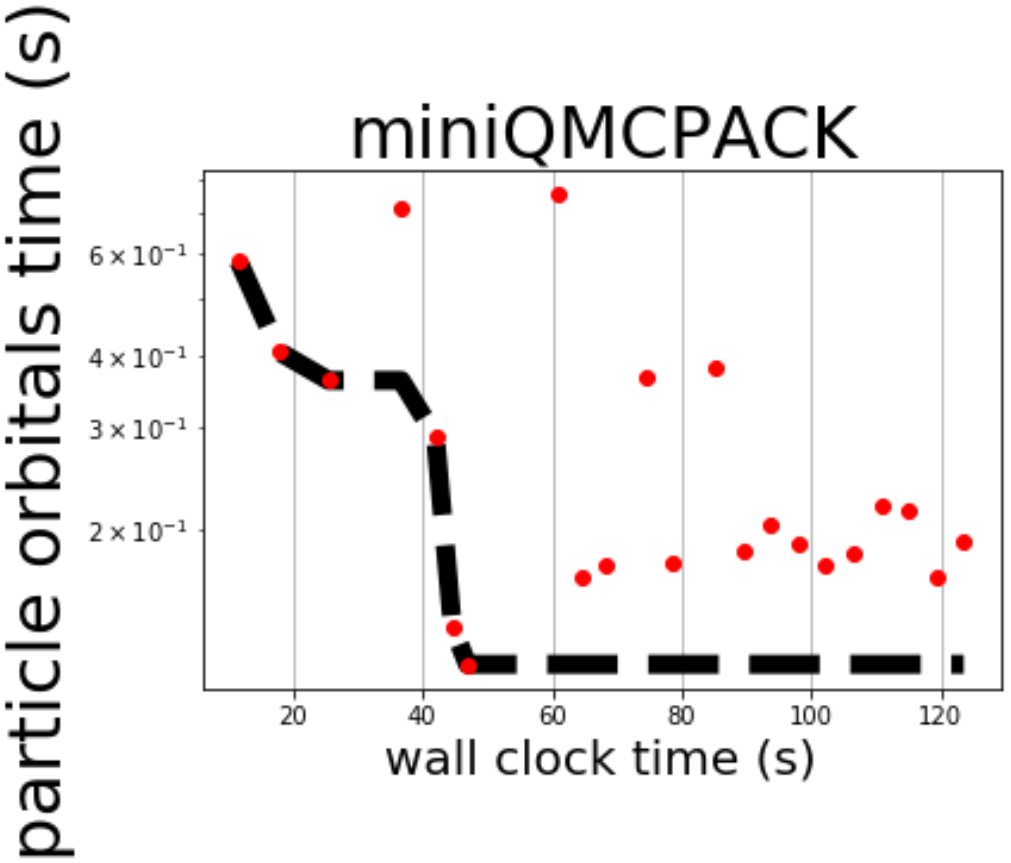


Bayesian optimization idea for sampling new point

Y-TUNE Self Assessment: Clients and Users

Initial Results on SuperLU and QMCPACK





SuRF obtained speedups comparable to domain-expert/manual tuning

Autotuning advantages:

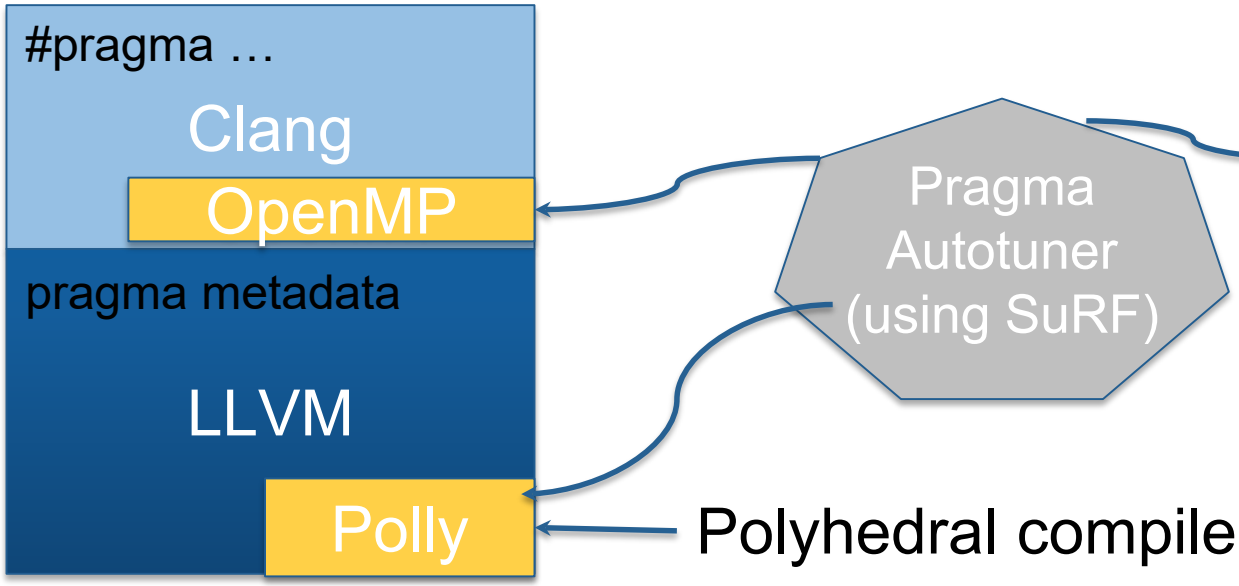
(1) Performance/portability; (2) Productivity; (3) Maintainability

Pragma Autotuner

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

```
/* Polly example */
#pragma clang loop unroll(4)
for (int i = 0; i < n; i+=1)  Statement(i);
```

```
/* OpenMP example */
#pragma omp parallel loop
for (int i = 0; i < n; i+=1)  Statement(i);
```



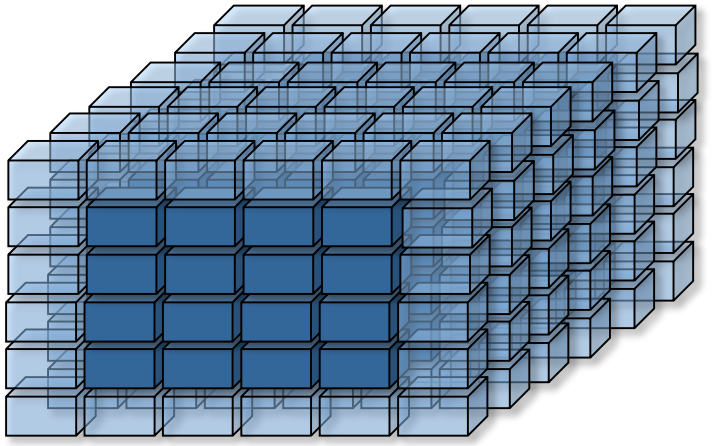

```
/* OpenMP example */
#pragma omp target distribute simd
for (int i = 0; i < n; i+=1)  Statement(i);
```

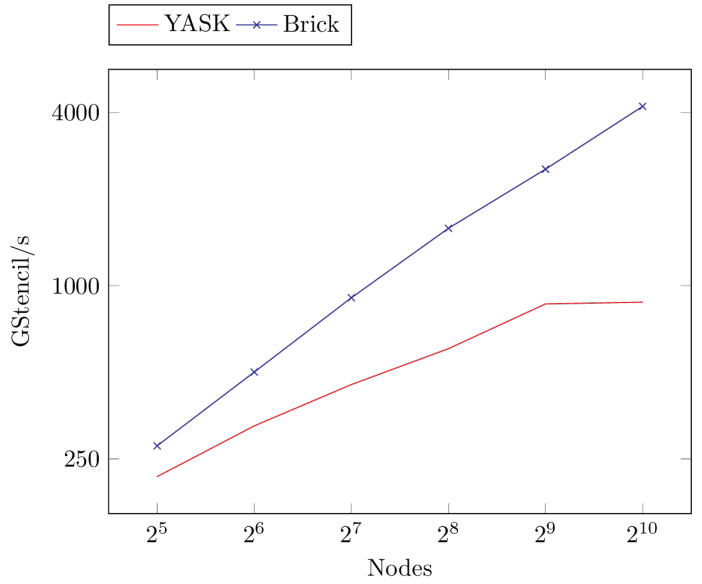
A Framework for OpenMP Autotuning, V. Sreenivasan, R. Javali, M. Hall, T. Scogland, B. de Supinski, IWOMP'19.

Polyhedral compiler in LLVM

Brick Data Layout, Applied to High-Order Stencils

- Data layout ***reduces data movement***
  - Brick: 4x4x4 subdomain w/o a ghost zone
  - Uses contiguous storage and adjacency lists
- Achieves ***performance portability***
  - Automation of architecture-specific “vector” code generation
  - Dramatically reduces vertical (register/cache/TLB) and horizontal (MPI) data movement
  - Adjustable brick size and indirection adapts to architecture limits
  - Indirection eliminates message packing and reduces communication overhead





with Tuowen Zhao (Utah), Sam Williams, Hans Johansen (LBNL), To appear SC’19

Workshop on MLIR for HPC

October 21, 2019, Georgia Institute of Technology, Atlanta, GA

Rooms 1123 and 1116, Klaus Advanced Computing Building

Held in conjunction with International Workshop on Languages and Compilers for Parallel Computing (LCPC 2019)

Organizing Committee:

Uday Bondhugula, Indian Institute of Science
Albert Cohen, Google
Tobias Grosser, ETH
Mary Hall, University of Utah
Santosh Pande, Georgia Tech
P. Sadayappan, University of Utah
Vivek Sarkar, Georgia Tech
Michelle Strout, University of Arizona
Reid Tatge, Google

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

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

Goal: HPC Support in Open Source Compilers

Short-term

(ECP time frame)

Extend LLVM

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

Potential longer term

(But need to start now)

Collaborate on MLIR

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