Autotuning Scientific Applications for Energy Efficiency at Large Scales

Xingfu Wu

Argonne National Laboratory, The University of Chicago

P. Balaprakash, M. Kruse, J. Koo, B. Videau, P. Hovland, V. Taylor (Argonne)

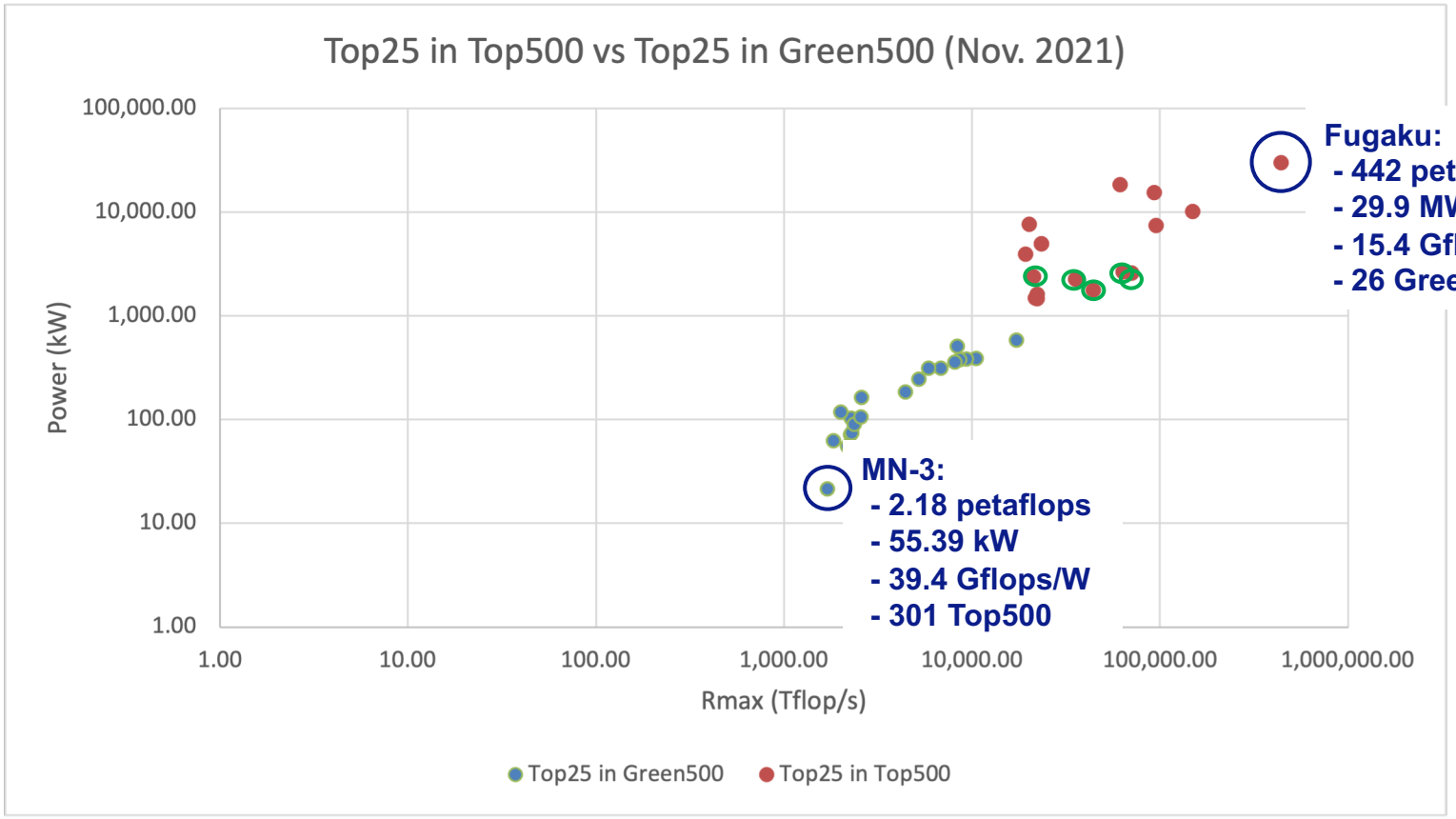
B. Geltz, S. Jana (Intel)

M. Hall (University of Utah)

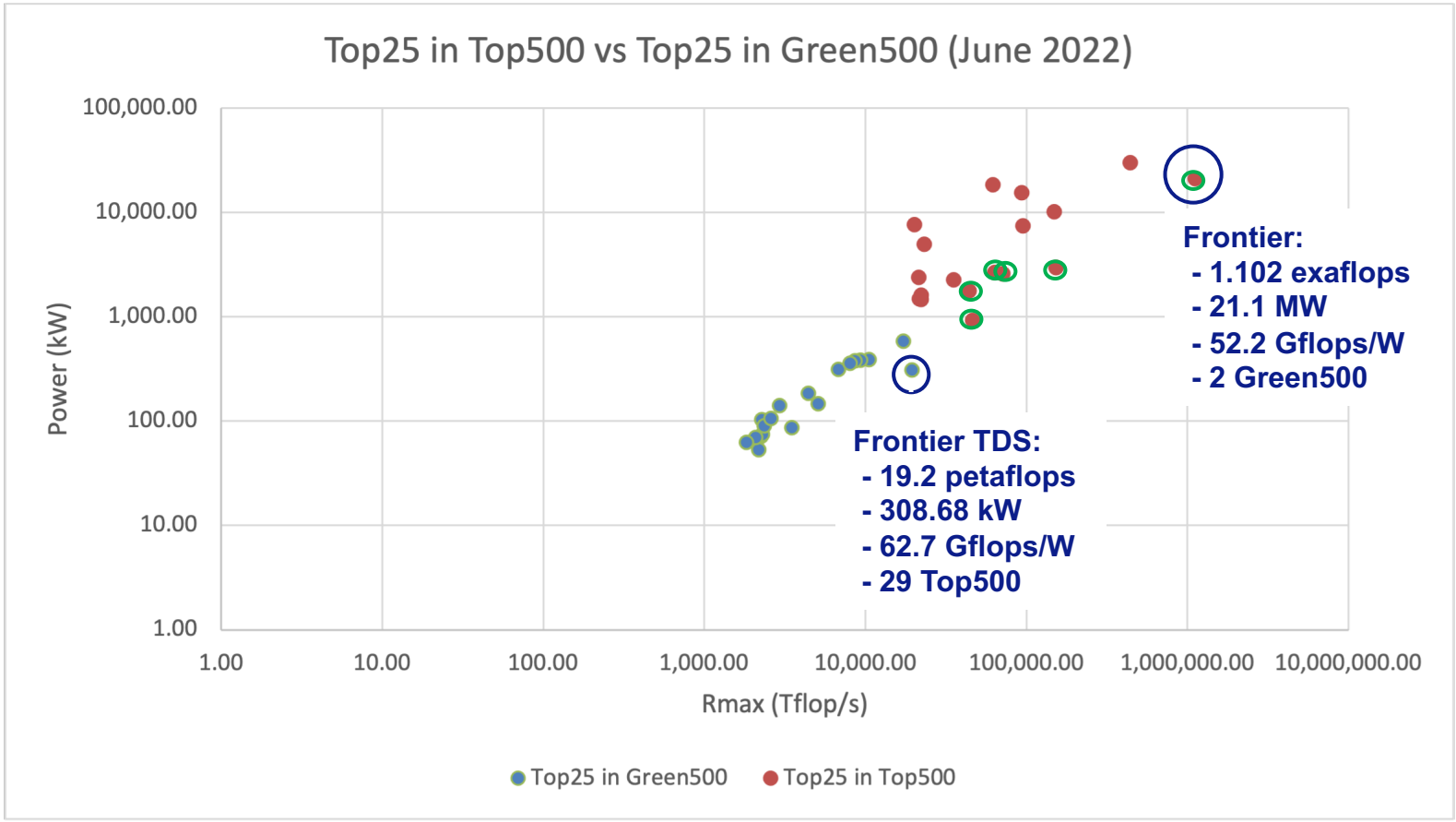
CUG2023, Helsinki, Finland

May 9, 2023

Argonne NATIONAL LABORATORY



Source: www.top500.org, Nov 2021



Source: www.top500.org, June 2022

Background and Motivations

- As we enter the exascale computing era, efficiently utilizing power and optimizing the performance of scientific applications under power and energy constraints has become critical and challenging.
- As the complexity of high performance computing (HPC) ecosystems continues to rise, achieving optimal performance becomes a challenge.
- The number of tunable parameters an HPC user can configure has increased significantly, resulting in a dramatically increased parameter space.
- Exhaustively evaluating all parameter combinations becomes very time-consuming.
- Solution: autotuning for automatic exploration of parameter space is desirable.

- Autotuning is an approach that explores a search space of tunable parameter configurations of an application efficiently executed on an HPC system.
- Typically, one selects and evaluates a subset of the configurations on the target system and/or uses analytical models to identify the best implementation or configuration for high performance or energy efficiency.

Background and Motivations

- Existing autotuning frameworks were for autotuning on a single or a few compute nodes.
- Current large-scale HPC systems such as Theta at ANL and Summit, Frontier at ORNL have complex system architectures and software stacks with many tunable parameters that may affect the system performance and energy.
- Application developers and users often rely on these systems with the default configurations setup by the vendors to run their applications. How efficiently are these applications executed?
- How can we identify the best combination of these parameters for the best system performance or the lowest system energy consumption?
- Can we develop a low-overhead and scalable framework to autotune large-scale applications for performance or energy efficiency on large-scale HPC production systems such as Theta and Summit?

Our Previous Work

- Autotuning framework ytopt was developed (X. Wu, et al., Autotuning PolyBench Benchmarks with LLVM Clang/Polly loop optimization pragmas using Bayesian optimization, *Concurrency and Computation: Practice and Experience*, vol. e6683, https://doi.org/10.1002/cpe.6683, 2021)
  - LLVM Clang/Polly loop optimization pragmas (loop tiling, loop interchange, loop reversal, array packing) focus on a single core optimization
  - Autotuned PolyBench Benchmarks on a single compute node
  - Autotuned a deep learning application MNIST on a single node

- Toward an End-to-End Autotuning Framework in HPC PowerStack (X. Wu, et al., Energy Efficient HPC State of Practice 2020, Kobe, Japan, Sep. 14, 2020)
  - HPC PowerStack - a global consortium of laboratories, vendors, and universities - has highlighted a design shift towards standardization of the HPC power management software stack.
  - This enables seamless integration of software solutions that enable management of energy/power consumption of large scale HPC systems.

  - This work surveyed the high-level objectives of the existing layer-specific tuning approaches, defined the tunable parameters, and proposed and discussed how to autotune the combinations

Our Approaches

- We propose a low-overhead, scalable autotuning framework to tune various hybrid MPI/OpenMP scientific applications at large scales.

- We use this framework to autotune four hybrid MPI/OpenMP ECP proxy applications, namely XSBench, AMG, SWFFT, and SW4lite, using Bayesian optimization with a Random Forest surrogate model to effectively search parameter spaces with up to 6 million different configurations.

- We demonstrate the effectiveness of our autotuning framework to tune the performance or energy consumption of these hybrid applications on up to 4,096 nodes with 262,144 cores.

- The experimental results show that our proposed autotuning framework at large scales has low overhead and good scalability on Theta and Summit.

Methodology of the Autotuning Framework

- We analyze an application code to identify the important tunable application and system parameters (OpenMP runtime environment variables) to define the parameter space.

- We use these tunable parameters to parameterize an application code as a code mold.

- ytopt starts with the user-defined parameter space, the code mold, and user-defined interface that specifies how to evaluate the code mold with a particular parameter configuration.

Methodology of the Autotuning Framework

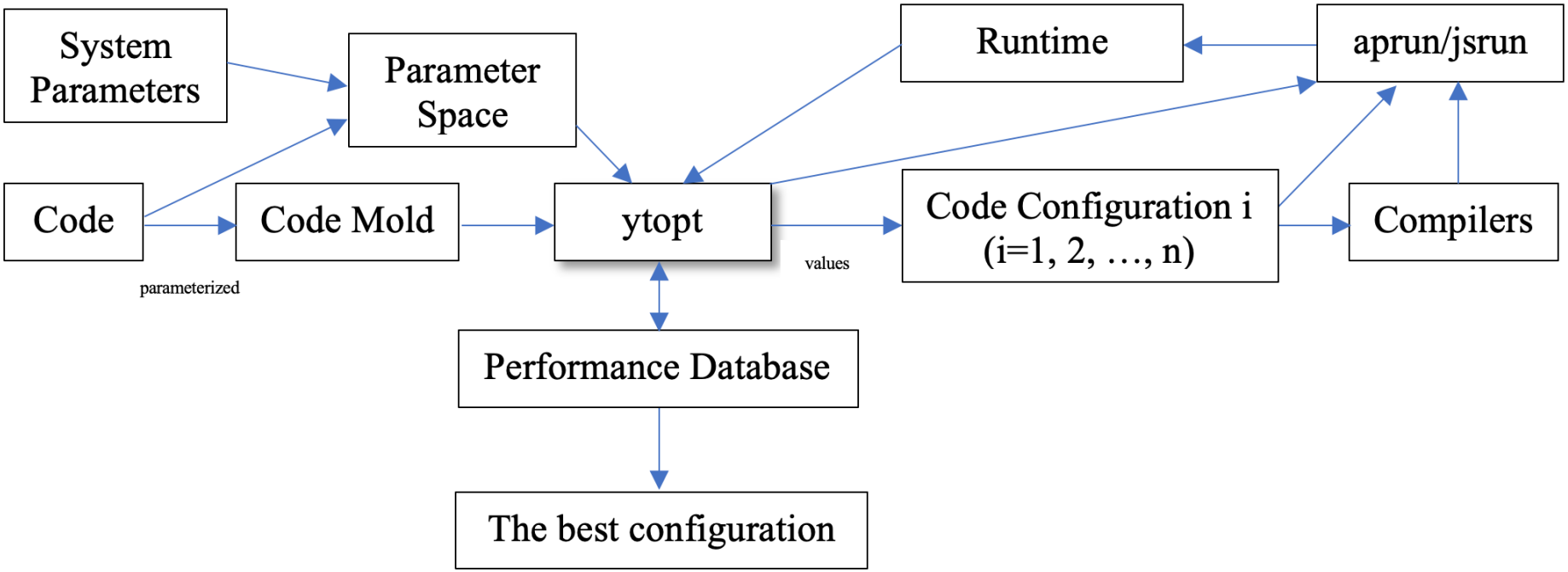
- The search method within ytopt uses Bayesian optimization, where a dynamically updated Random Forest surrogate model that learns the relationship between the configurations and the performance metric, is used to balance exploration and exploitation of the search space.

  – In the exploration phase, the search evaluates parameter configurations that improve the quality of the surrogate model,

  – In the exploitation phase, the search evaluates parameter configurations that are closer to the previously found high-performing parameter configurations.

  – The balance is achieved through the use of the lower confidence bound (LCB) acquisition function that uses
    - the surrogate models' predicted values of the unevaluated parameter configurations
    - and the corresponding uncertainty values (standard deviation).

Proposed Autotuning Framework in Performance



Iterative Phase of the Proposed Framework

- Step 1: Bayesian optimization selects a parameter configuration for evaluation.
- Step 2: The code mold is configured with the selected configuration to generate a new code.
- Step 3: Based on the value of the number of threads in the configuration and the number of nodes reserved, aprun/jsrun command line for the launch of the application on the compute nodes is generated.
- Step 4: The new code is compiled with other codes needed to generate an executable (if needed)
- Step 5: The generated command line is executed to evaluate the application with the selected parameter configuration; the resulting application runtime is sent back to the ytopt and is recorded in the performance database.

- Steps 1–5 are repeated until the maximum number of code evaluations or the wall-clock time is exhausted for the autotuning run.

Some Terms Defined

- The term **ytopt processing time** includes
  - the time spent in the parameter space search,
  - building the surrogate model,
  - processing the selected configuration to generate a new code and the aprun/jsrun command line,
  - compiling the new code,
  - launching the application,
  - and storing the configuration and performance in the performance database (except the application runtime).
- The term **ytopt overhead** stands for the ytopt processing time minus the application compiling time.

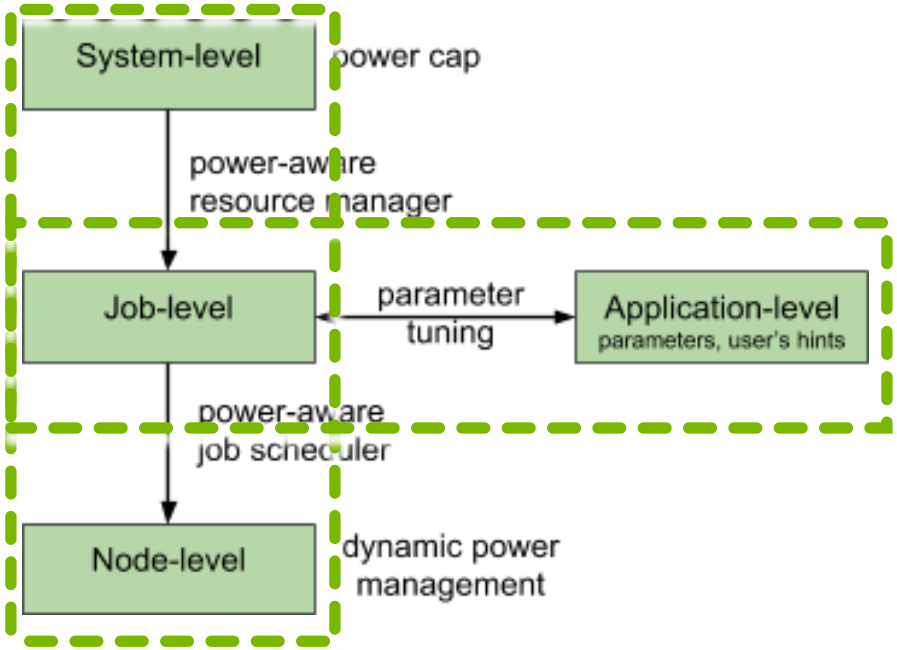
- Compiling time (s) for each application on Theta and Summit

System	XSBench	SWFFT	AMG	SW4lite
Theta	2.021	3.494	2.825	162.066
Summit	4.645	3.781	2.757	58.000

End-to-End Autotuning Framework in PowerStack

Motivation

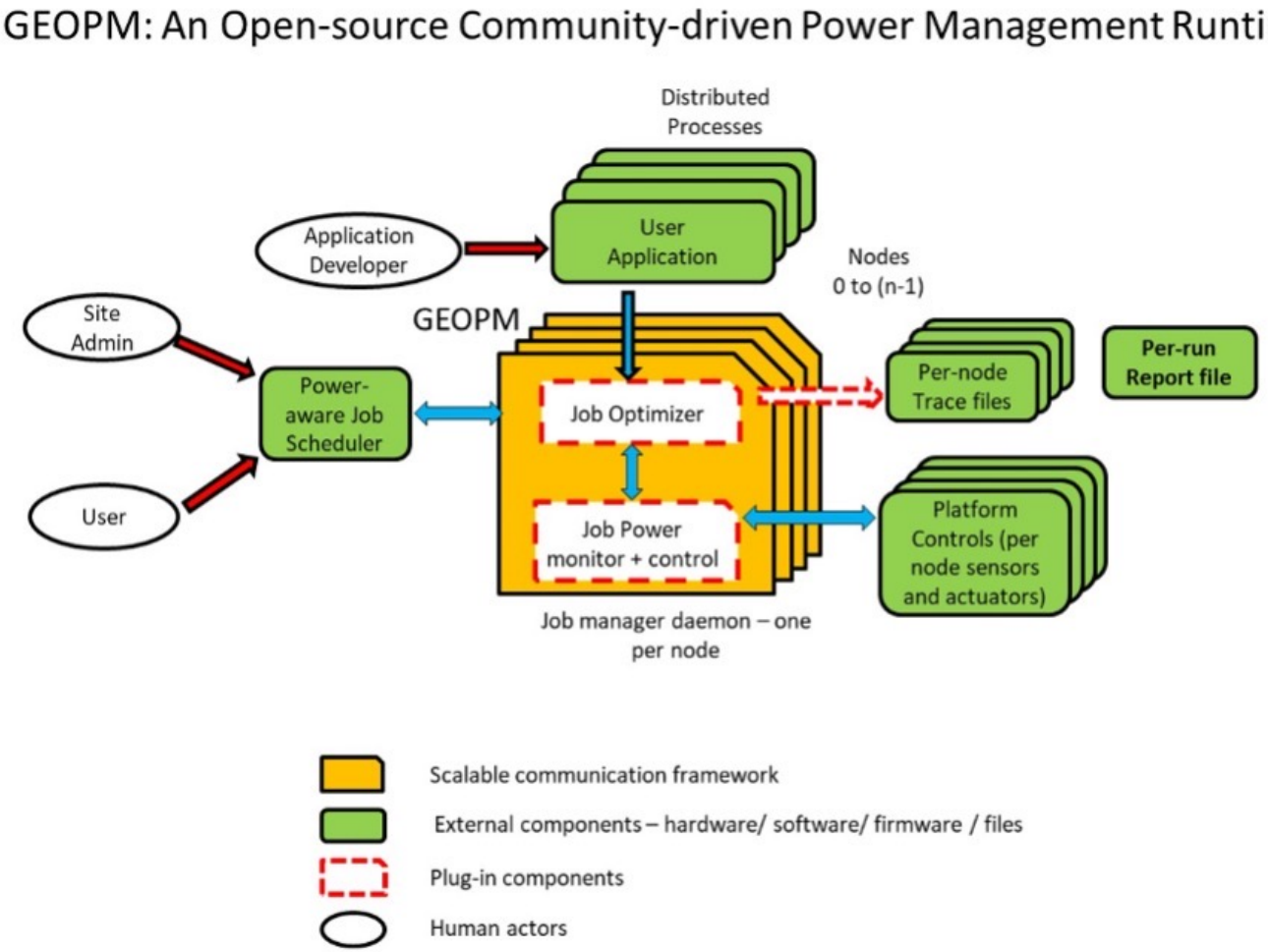
- Existing autotuning primarily layer-specific
- Need to identify, quantify and explore opportunities for cross-layer tuning



Approaches

- Surveyed existing co-tuning work
- Identified open challenges, hard problems in co-tuning
- Provided a platform to drive collaboration on developing co-tuning solution

Global Extensible Open Power Manager (GEOPM)



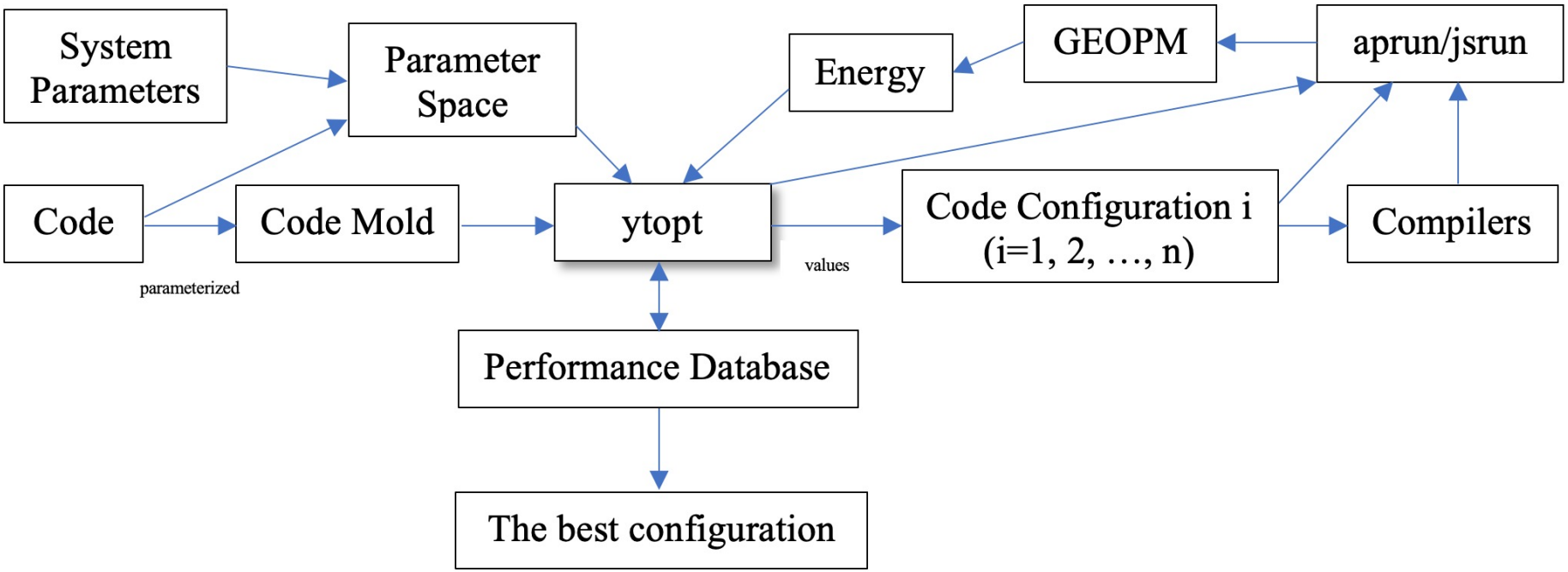
GEOPM: https://geopm.github.io

GEOPM

- It provides multiple interfaces to enable interoperability with external HPC software components such as enabling job schedulers and resource managers to drive job-aware system-wide power efficiency improvements.
- It enables control and monitoring of hardware/software knobs across multiple platforms and architectures such as leveraging multiple power and performance knobs like Intel's hardware power-limiting capability (RAPL) for achieved CPU frequency

- The GEOPM job launch script, geopmlaunch, queries and uses the OMP_NUM _THREADS environment variable to choose affinity masks for each process.
- The principal job of geopmlaunch to aprun is to set explicit per-process CPU affinity masks that will optimize performance while enabling the GEOPM controller thread to run on a core isolated from the cores used by the primary application.
- The geopmlaunch enables the GEOPM library to interpose on MPI using the PMPI interface through the LD_PRELOAD mechanism for unmodified binaries.

Proposed Autotuning Framework in Energy



Methodology of the Proposed Energy Framework

- Steps 1 and 2 are the same.

There are some differences in Steps 3, 4, and 5.

- At Step 3, ytopt sets the OMP_NUM_THREADS environment variable and generates the aprun command line for application launch.
- For the compiling Step 4, the dynamic linking is required with the -dynamic flag.
- At Step 5, ytopt uses the geopmlaunch to launch the aprun command line with the options ”--geopm- ctl=pthread,” which launches the controller as an extra pthread per node, and ”--geopm-report=gm.report,” which creates the summary report file gm.report about performance, power, and energy for each node to evaluate the application with the configuration. ytopt processes the summary report file from GEOPM to record the average node energy in the performance database.

- Steps 1–5 are repeated until the maximum number of code evaluations or the wall-clock time for the run.

Systems: Theta and Summit

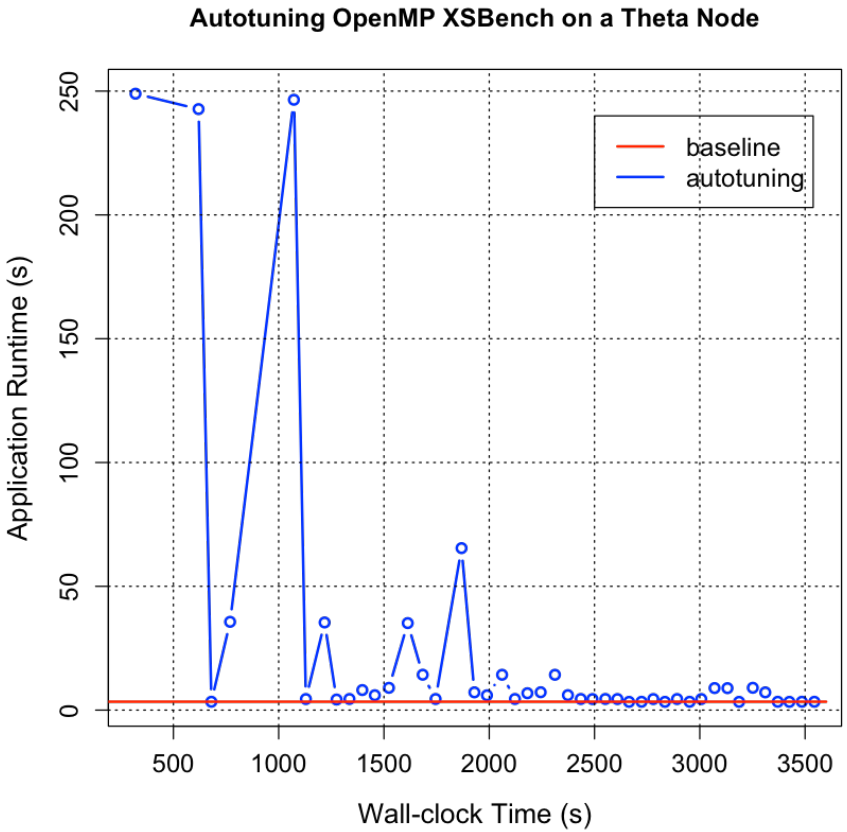
System Name	Cray XC40 Theta	IBM Power9 Summit
Location	Argonne National Lab	Oak Ridge National Lab
Architecture	Intel KNL	IBM Power9 + Nivida GPU
Number of nodes	4,392	4,608
CPU cores per node	64	42
Sockets per node	1	2 for Power9; 2 for GPU sockets
CPU type and speed	Xeon Phi KNL 7230 1.30GHz	IBM Power9 4GHz
GPUs per node	None	6 Nividia Volta GPUs
L1 cache per core	D:32KB, I:32KB	D:32KB, I:32KB
L2 cache per socket	32MB (two cores shared 1MB)	21MB (two cores shared 512KB)
L3 cache per socket	None	120MB (shared)
Threads per core	4	4
Memory per node	16GB MCDRAM, 192GB DDR4	96GB HBM2, 512GB DDR4
Network	Cray Aries Dragonfly	dual-rail EDR InfiniBand
Power tools	GEOPM, CapMC, RAPL	Nvidia-smi, NVML
TDP per socket	215W	190W/Power9; 300W/GPU
File System	Lustre PFS (210GB/s)	IBM GPFS (2.5TB/s)

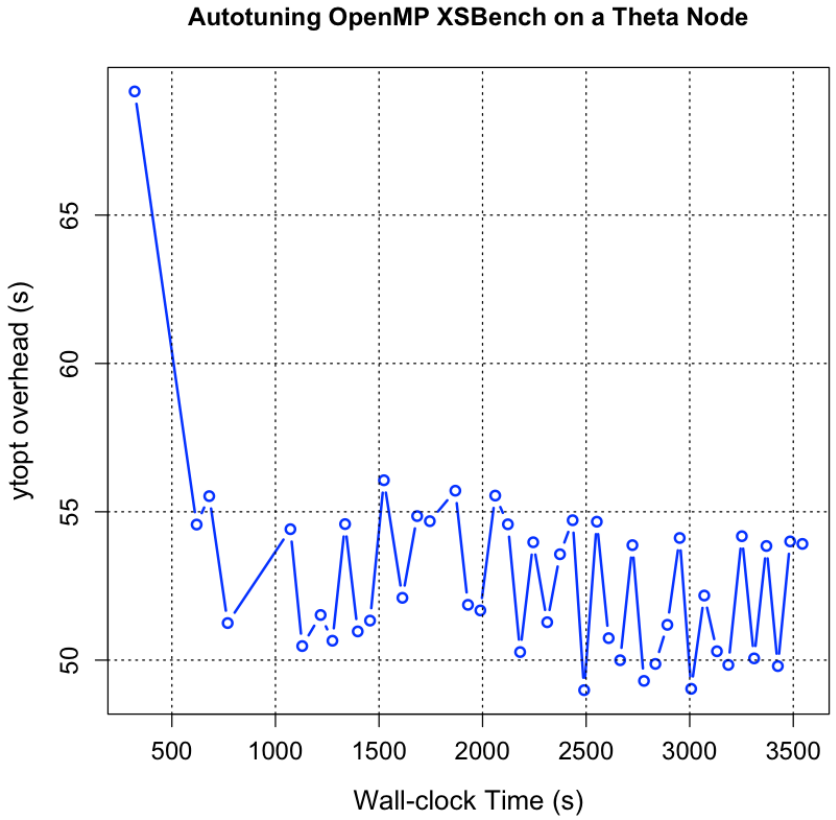
Four ECP Proxy Application and Parameter Spaces

ECP Proxy Apps	System param.	Application param.	Space size
XSBench	4 env. variables	2	51,840
XSBench-mixed	4 env. variables	5	6,272,640
XSBench-offload	5 env. variables	4	181,440
SWFFT	4 env. variables	1	1,080
AMG	4 env. variables	3	552,960
SW4lite	4 env. variables	4	2,211,840

Three weak scaling applications: XSBench, SWFFT, and AMG
One strong scaling application: SW4lite

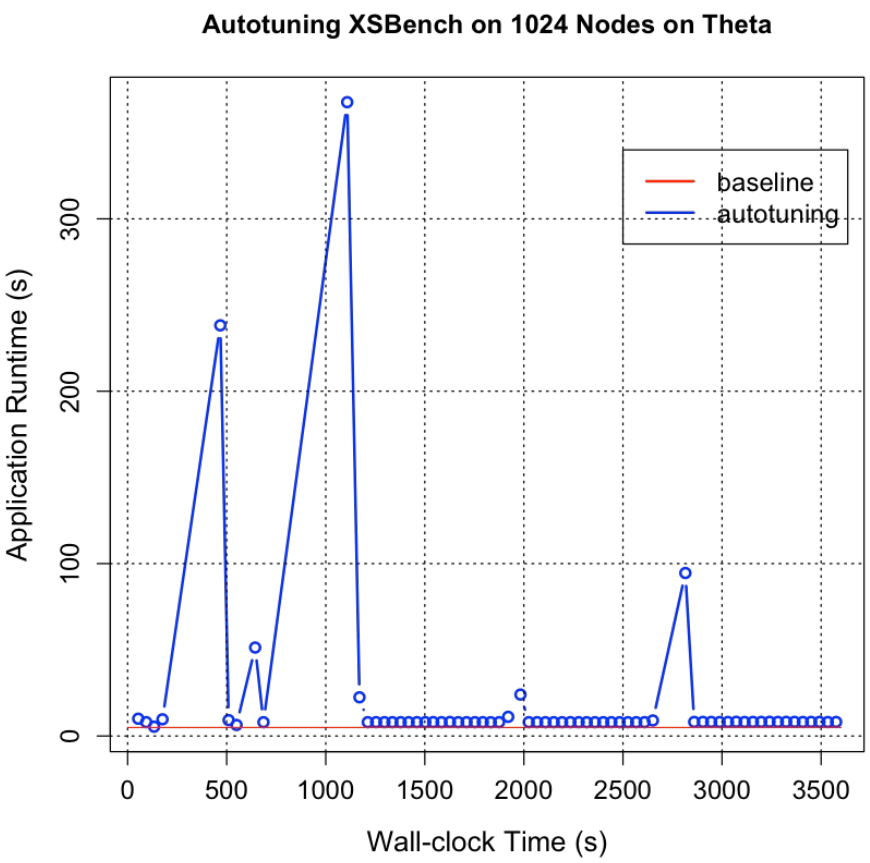
Case Study: Autotuning Performance on a Single Node

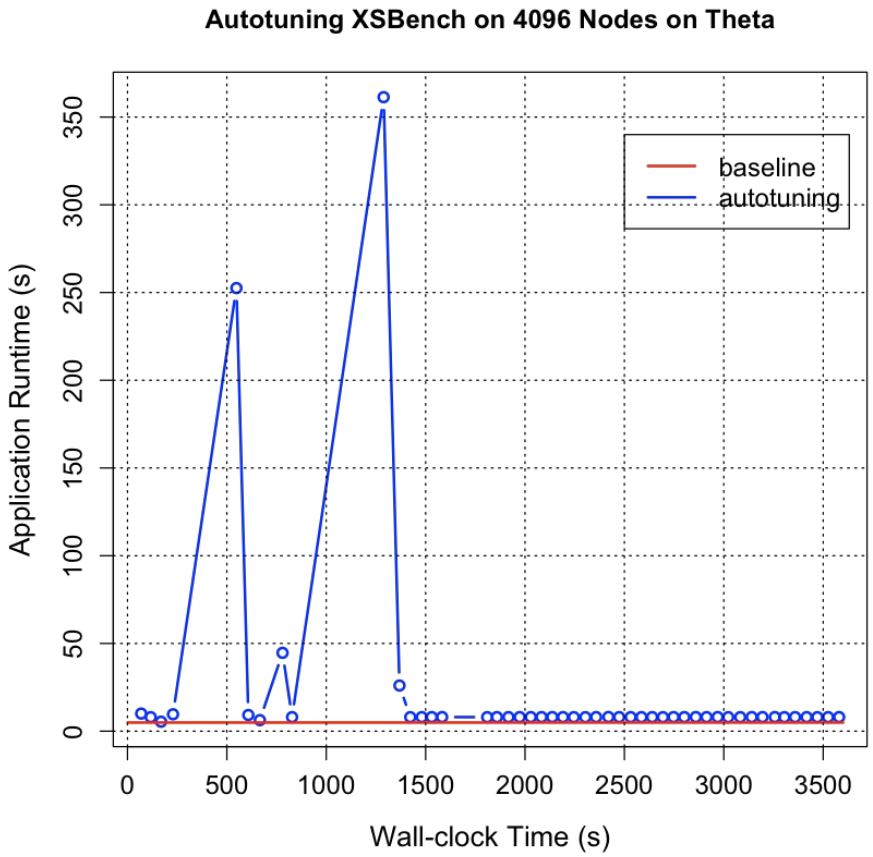




- On Theta, the baseline: 3.395s; Autotuning: 3.339s
- ytopt overhead: less than 70s

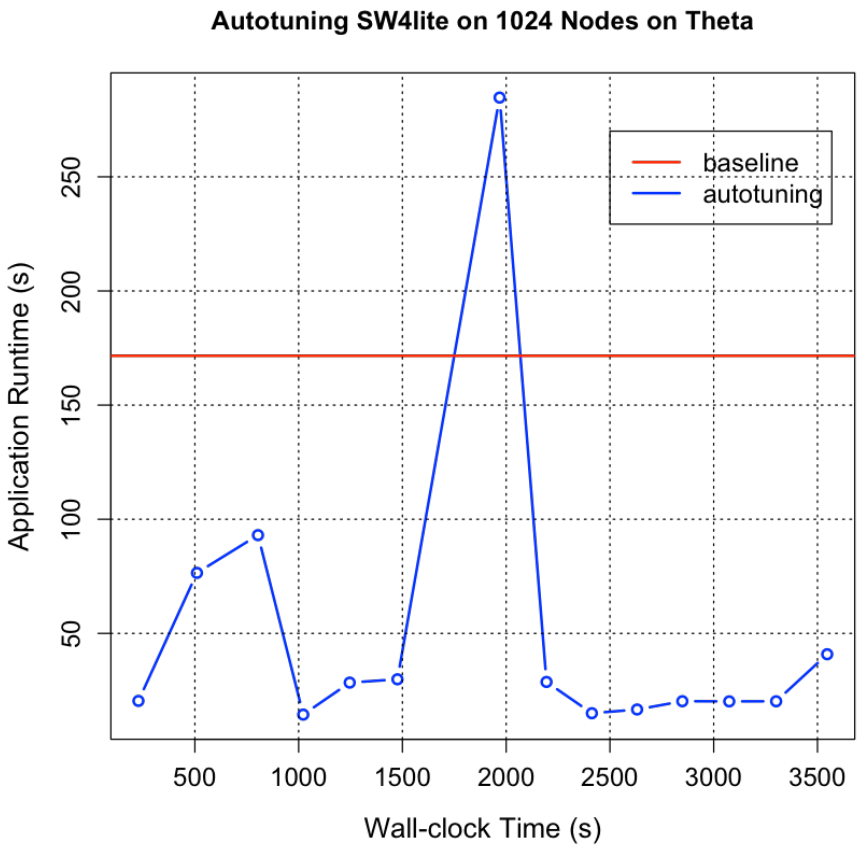
Case Study: Autotuning Performance at Large Scales

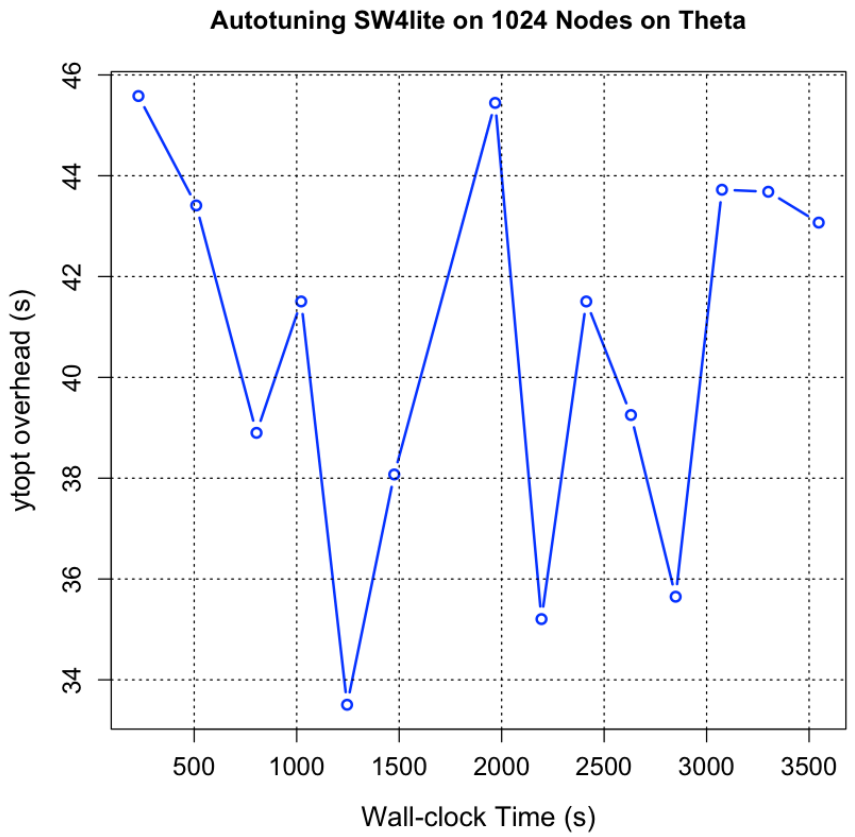




- Autotuning XSBench on 1024 and 4096 nodes on Theta
- ytopt search reaches the good region of the parameter space over time

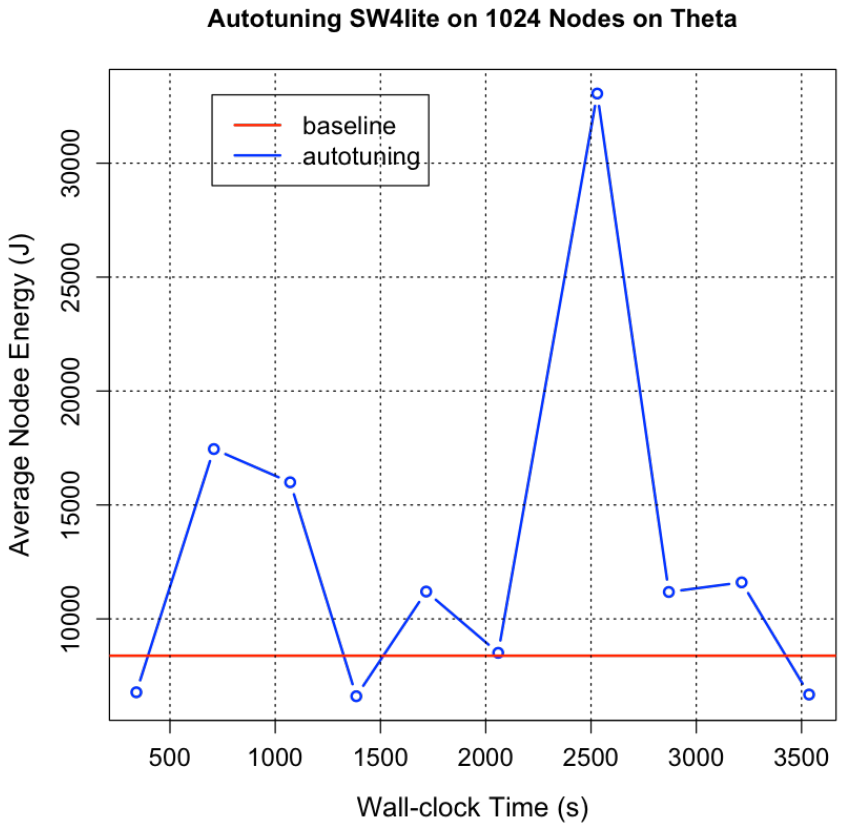
Case Study: Autotuning Performance at Large Scales

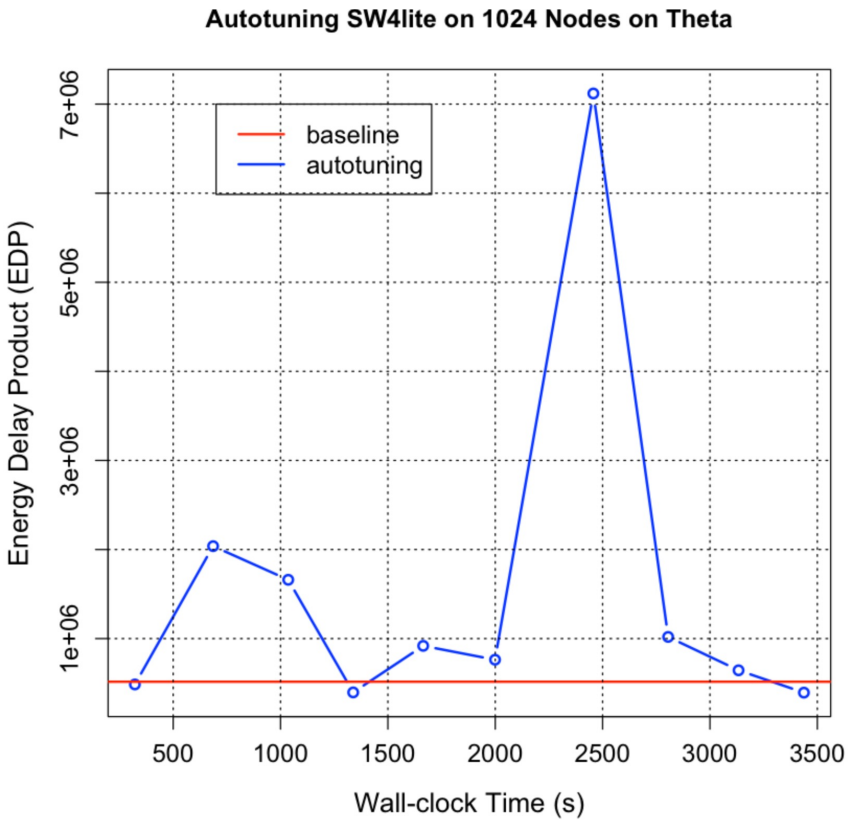




- On Theta, 91.59% performance improvement
- ytopt overhead: less than 46s

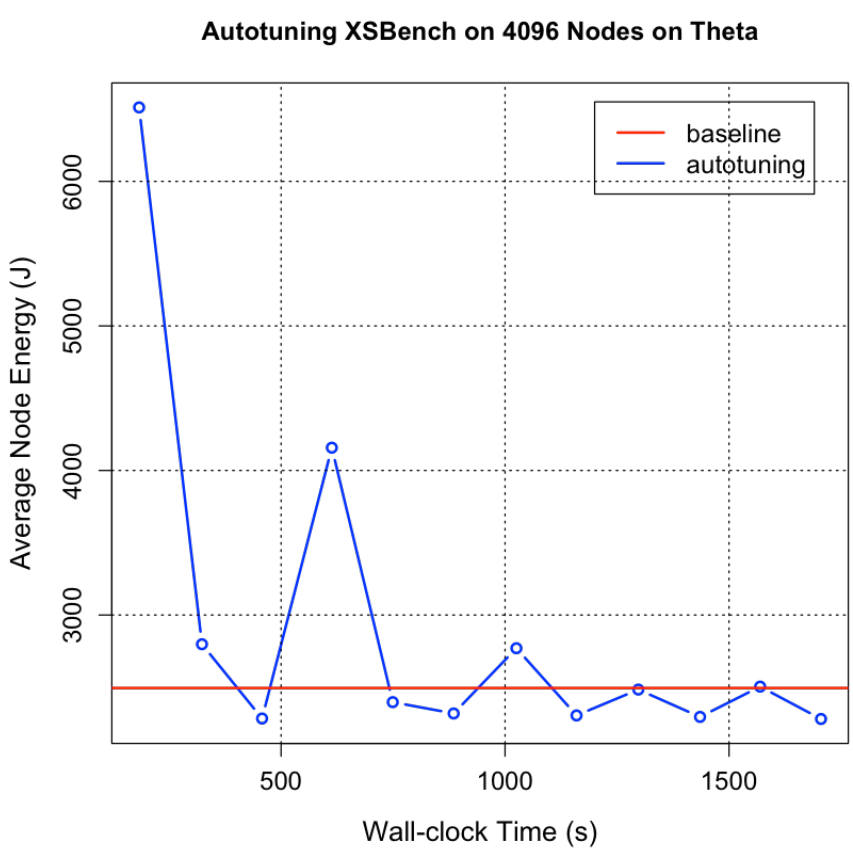
Case Study: Autotuning Energy at Large Scales

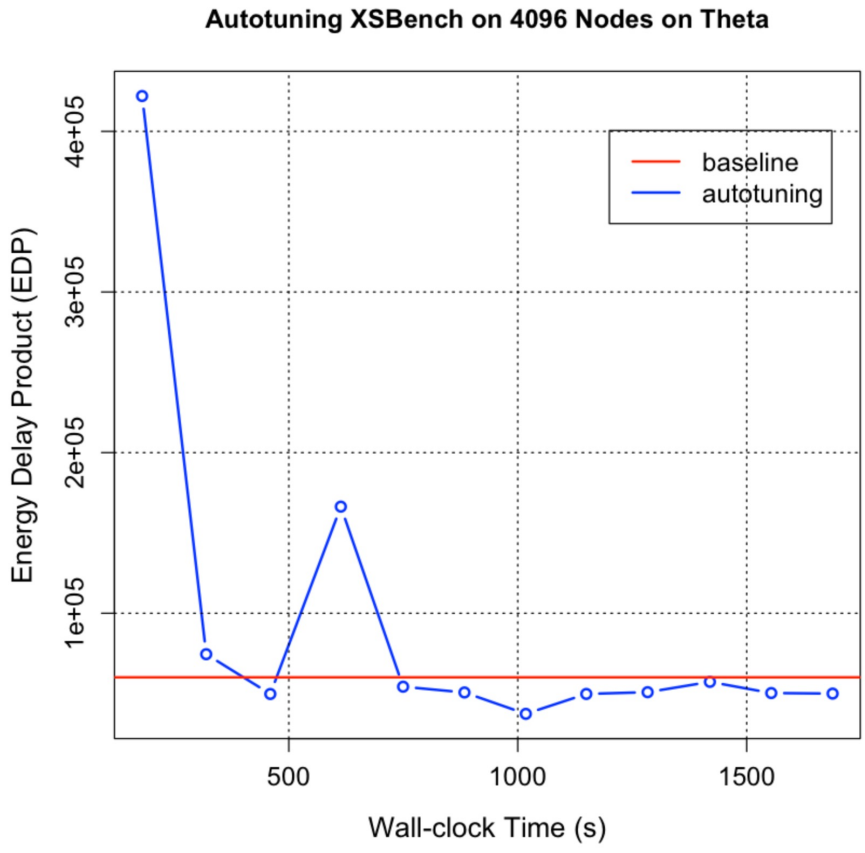




- 21.20% for energy saving
- 23.70% for improving EDP

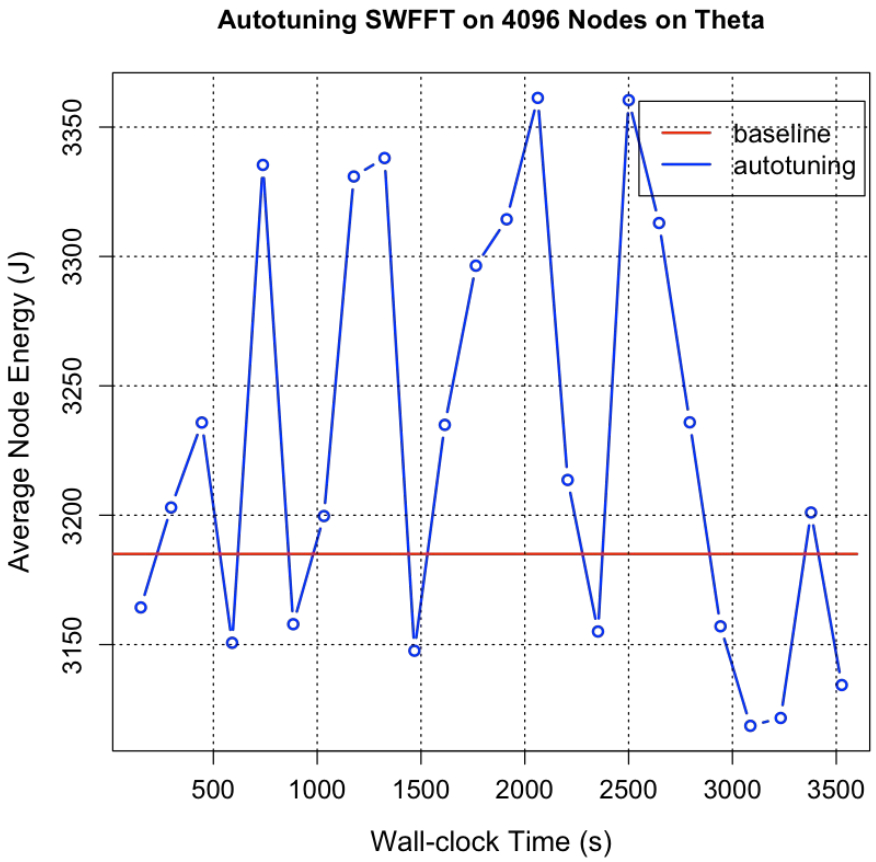
Case Study: Autotuning Energy at Large Scales

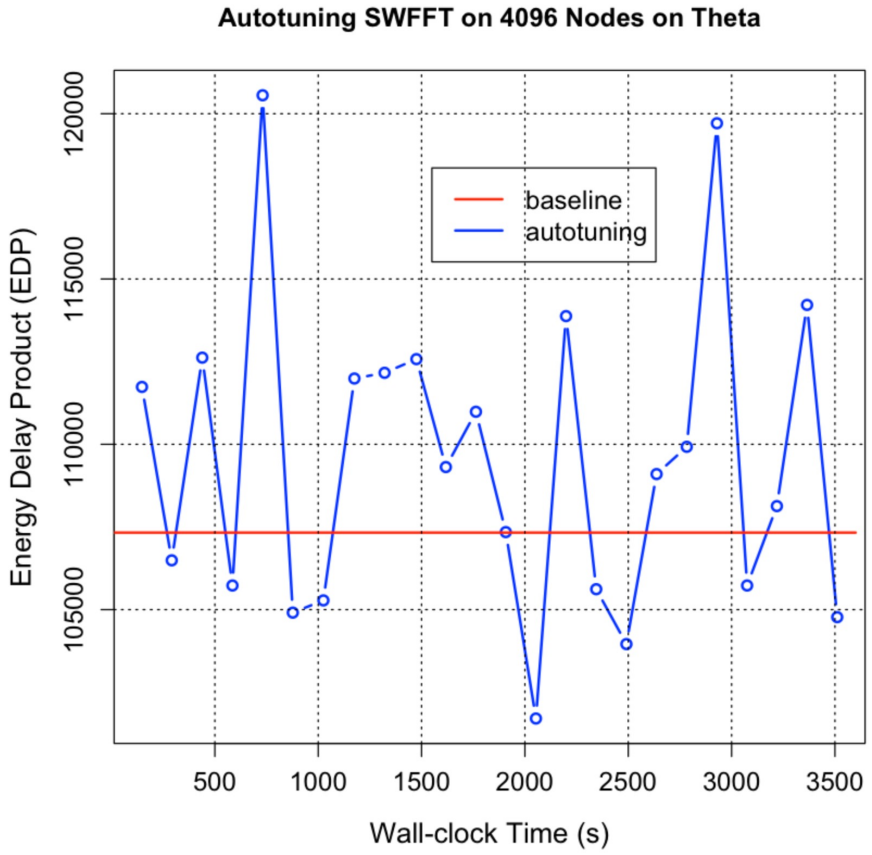




- 8.58% for energy saving
- 37.84% for improving EDP

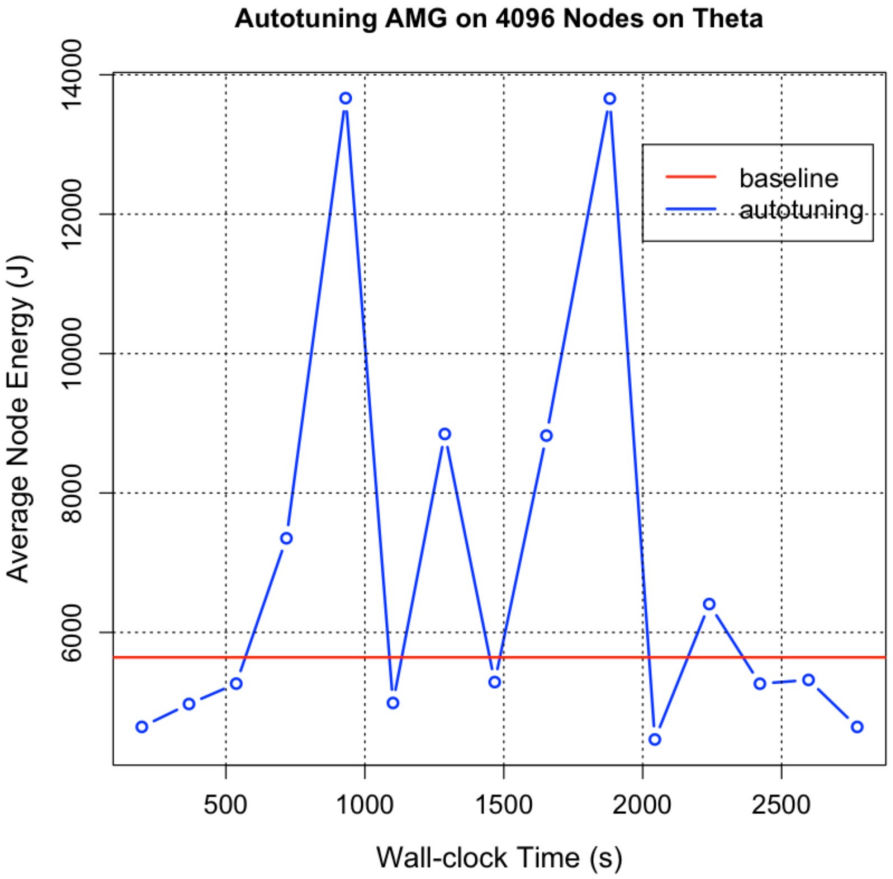
Case Study: Autotuning Energy at Large Scales

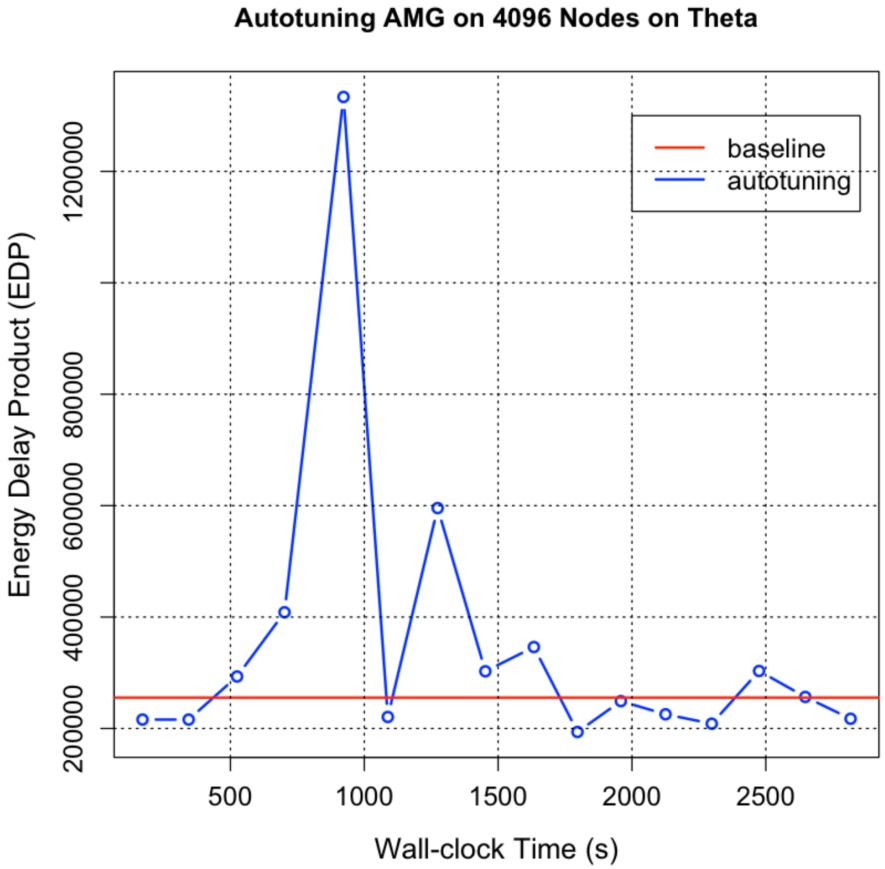




- 2.09% for energy saving
- 5.24% for improving EDP

Case Study: Autotuning Energy at Large Scales





- 20.88% for energy saving
- 24.18% for improving EDP

Summary

- We proposed the low-overhead, scalable autotuning frameworks to autotune four hybrid MPI/OpenMP ECP proxy applications—XSBench, AMG, SWFFT, and SW4lite— for energy efficiency at large scales.
- We used Bayesian optimization with a Random Forest surrogate model to effectively search the parameter spaces with up to 6 million different configurations on Theta and Summit.
- The experimental results showed that our autotuning framework had low overhead and good scalability.
- Using the proposed autotuning framework to identify the best configurations, we achieve up to 91.59% performance improvement, up to 21.2% energy saving, and up to 38.74% improvement in energy delay product (EDP) on up to 262,144 cores.

- This autotuning framework is open source and is available from our github repo: https://github.com/ytopt-team/ytopt

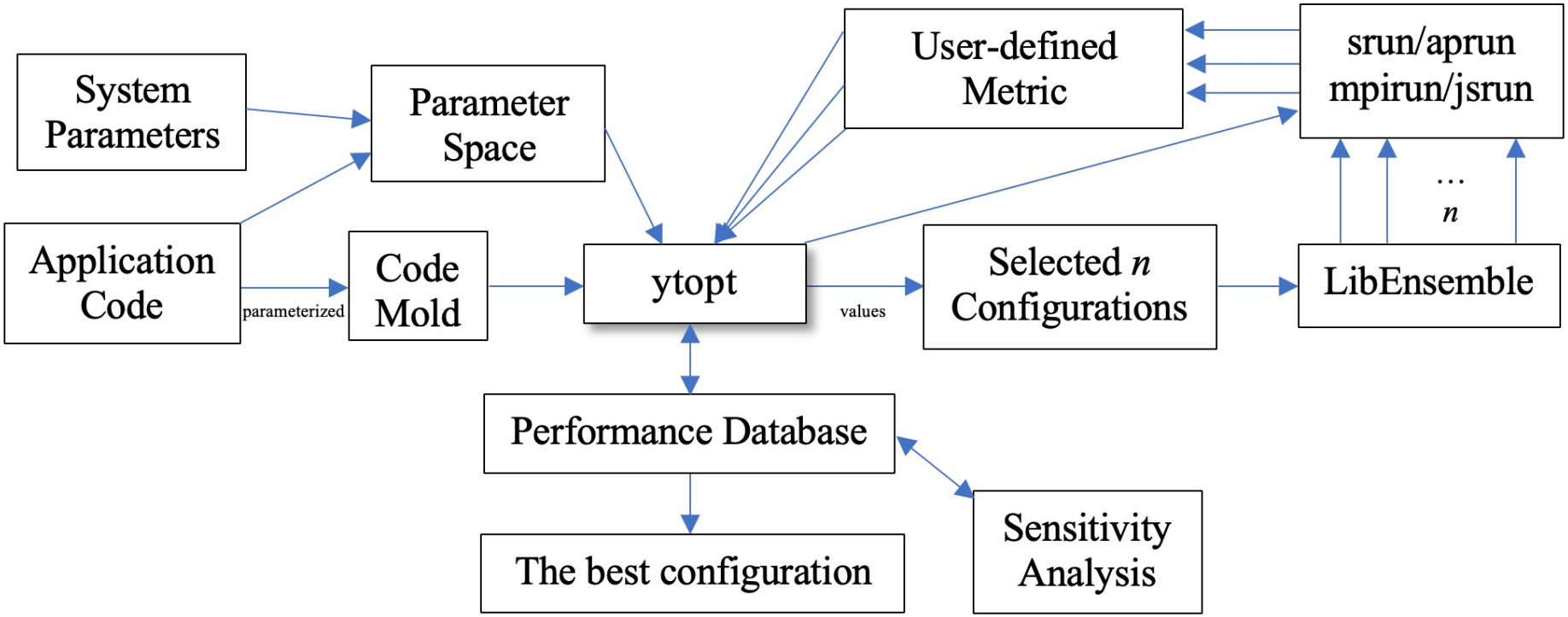
Future Work

- We plan to add transfer learning and online learning to the framework so that it can transfer what it learns from the applications at small scales in problem sizes and system sizes to guide and/or predict the autotuning at large scales.

- We further reduce the ytopt overhead by improving efficiency of python codes, reducing the application compiling time with pre-compiling the unchanged code files or pre-compiling the whole code by passing the parameter values to the command line or do the evaluations in parallel.

- Our autotuning framework ytopt can be applied to autotune a supercomputer with many tunable system parameters and a set of benchmarks for optimal configuration, performance, and energy efficiency.

YTOPT-LIBENSEMBLE AUTOTUNING FRAMEWORK



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

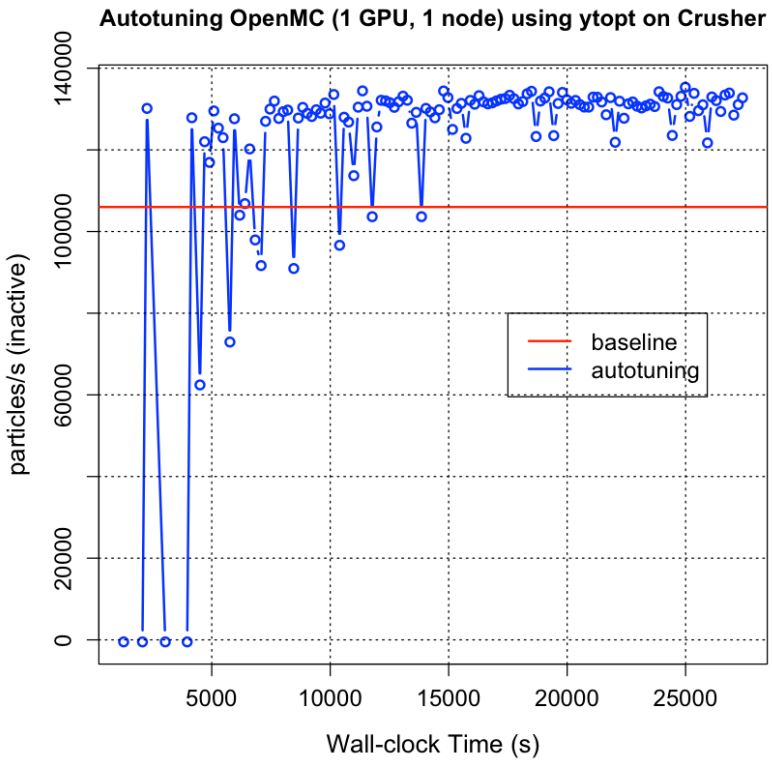
OPENMC PERFORMANCE TUNING KNOBS

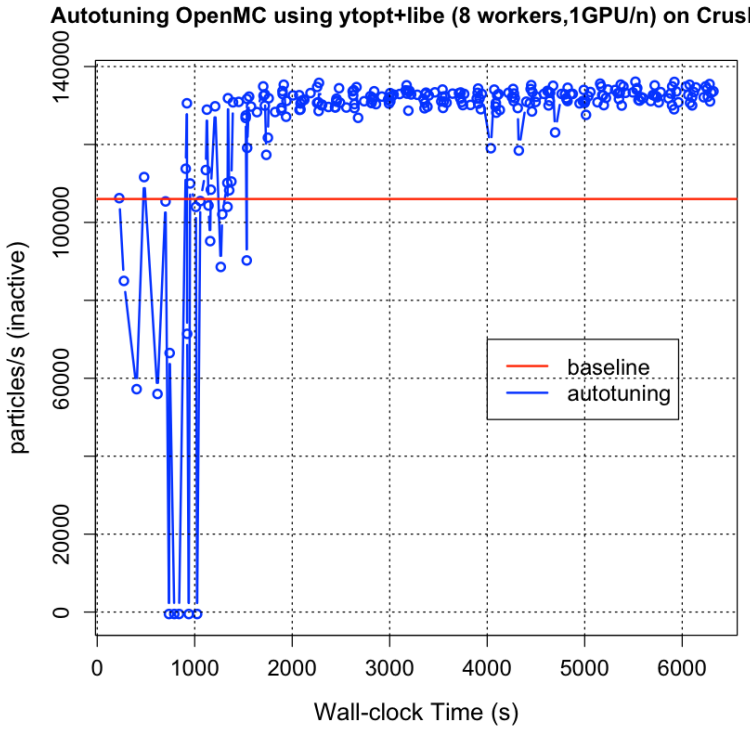
OpenMC Parameter	Type	Range/bounds	Default
Max number of particles in-flight	Integer	100k - max that will fit in memory (~8 million on A100)	1 million
Number of MPI ranks per GPU or tile	Integer	1 - 4	1
Number of logarithmic hash grid bins	Integer	100 - 100k	4,000
Queuing logic type	Mode	Queued vs. Queueless	Queued
Minimum sorting threshold (queued mode only)	Integer	0 (always sort) - infinity (never sort)	20,000

OpenMC: Monte Carlo (MC) neutral particle transport application

Complication: these application parameters are not independent!

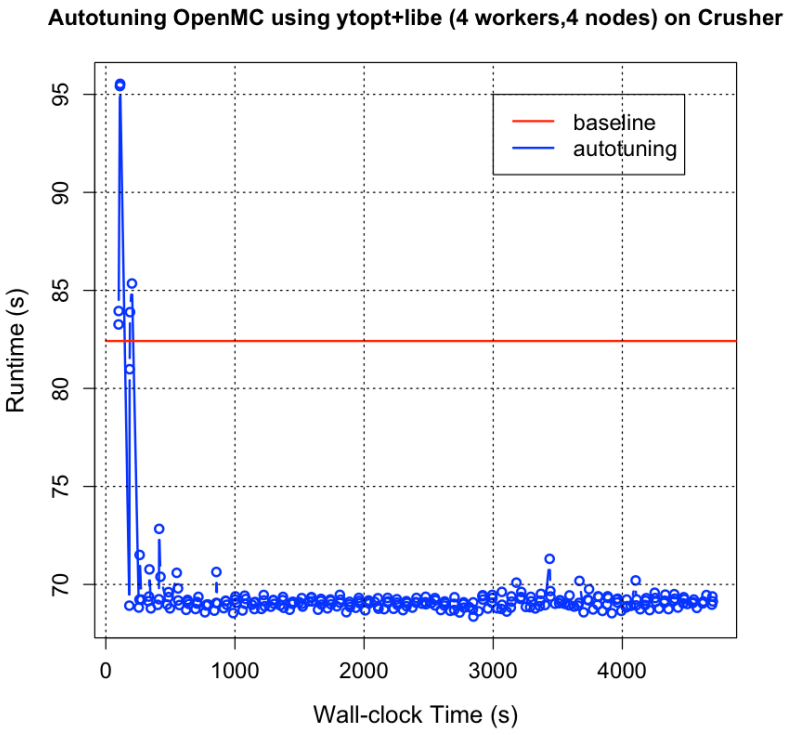
Autotuning OpenMC

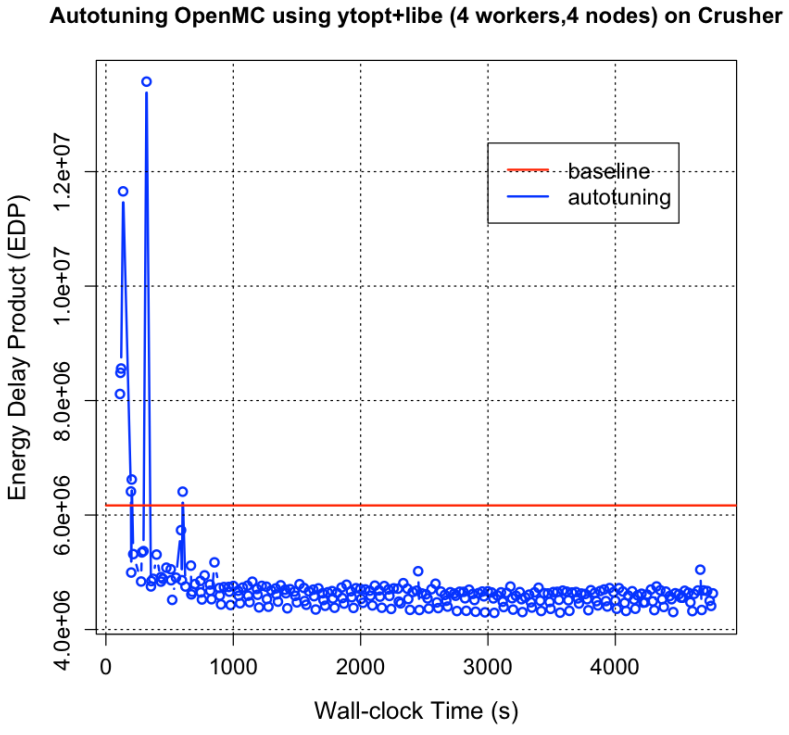




ytopt-libe identified high-performing configurations, resulting in a 28.39% improvement in performance using a single GPU on a single node with 256 evaluations

Autotuning other Metrics using ytopt-libe





ytopt-libe still identified high-performing configurations, resulting in 17.05% improvement in performance and 30.44% improvement in EDP

Acknowledgements

- This work was supported in part by
  - DoE ECP PROTEAS-TUNE
  - DoE ASCR RAPIDS2
  - NSF grant CCF-2119203