BOF Days

February 10 - 12, 2026

https://cass.community

The Consortium for the Advancement of Scientific Software (CASS)

https://cass.community/news/2026-02-10-cass-bof-days.html

CASS: Stewardship and Advancement of the Scientific Software Ecosystem

- **Inward-facing activities:** Strengthening software products
  - Improve development practices, sustainability, quality, and trustworthiness
  - Enhance user experience and integration within the broader ecosystem
- **Outward-facing activities:** Community engagement and discovery
  - Curate and evolve the software portfolio
  - Help teams connect with and grow their user communities
  - Enable the broader community to discover and adopt useful software

CASS Members

CORSA
Partnering with foundations to provide sustainable pathways for scientific software

FASTMATH
Stewardship, advancement, and integration for math and ML/AI packages

PESO
Stewarding, evolving and integrating a cohesive ecosystem for DOE software

RAPIDS
Stewardship, advancement, and integration for data, visualization and ML/AI packages

S4PST
Stewardship, advancement and engagement for programming systems

STEP
Stewardship, advancement of software tools for understanding performance and behavior

Sponsored by the Department of Energy, Office of Advanced Scientific Computing Research

Engage with CASS!

- Learn about CASS:
  - https://cass.community/about/
- Join the CASS Announcement list (low-volume):
  - http://eepurl.com/iRiSnY
- Find out more about our **software products**
  - Catalog: https://cass.community/software/
  - Collected as part of the Extreme-Scale Scientific Software Stack (E4S)
- Participate in **CASS Working Groups**
  - Impact Framework, Integration, Metrics, Software Ecosystem, User-Developer Experience, Workforce
  - https://cass.community/working-groups/

Autotuning HPC Codes and ML Pipelines with GPTune

Yang Liu, Sherry Li
Lawrence Berkeley National Laboratory

Feb 11, 2026
CASS BOF DAYS

Team

Jim Demmel
UC Berkeley

Sherry Li
LBNL

Yang Liu
LBNL

Younghyun Cho
Santa Clara Univ.

Hengrui Luo
Rice Univ.

Igor Kozachenko
UC Berkeley

Grace Dinh
Cornell Univ.

Past members:

Wissam Sid-Lakhdar
Univ. Tennessee

Osni Marques
LBNL

Mohsen Mahmouddi
Univ. Texas A&M

Chang Meng
Emory Univ.

Xinran Zhu
Cornell Univ.

Acknowledgement

- 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 was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research's Applied Mathematics program under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

Autotuning

- Problem

Given a target problem (task) and a parameterized code to solve it, find the parameter configuration (combination of parameter values) that optimizes the code performance

  - Metrics: solution time, memory or energy usage, etc. (or combined)

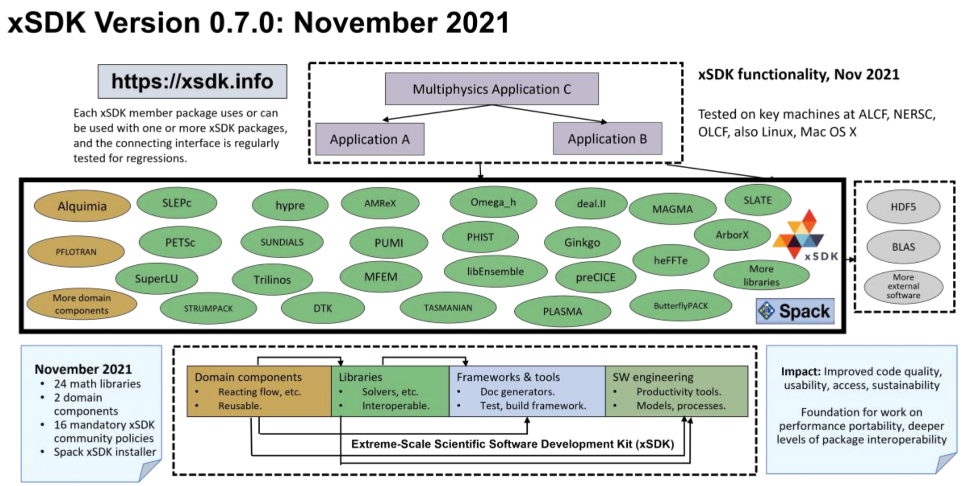
- Real application codes are costly
  - Run on large supercomputers, for a long time

- Real datasets can be large
  - Streaming measurement data can be large
  - Online shared database

- Goal: make best use of limited budget and available data

Applications of GPTune

- Tuning mathematical software on exascale computing platforms



xSDK: Extreme-scale Scientific Software Development Kit (figure obtained from https://xsdk.info/packages/)

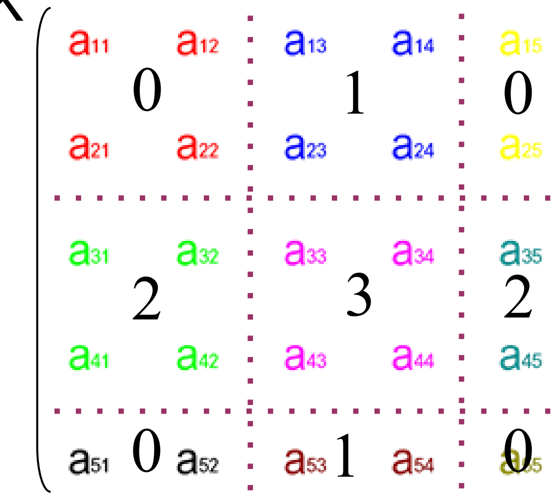
- Tuning computational science codes and experimental control
  - Fusion simulation: NIMROD, M3DC1
  - Electromagnetics simulation: cavity modeling, photonics crystal simulation
  - Accelerator physics: RHIC beamline

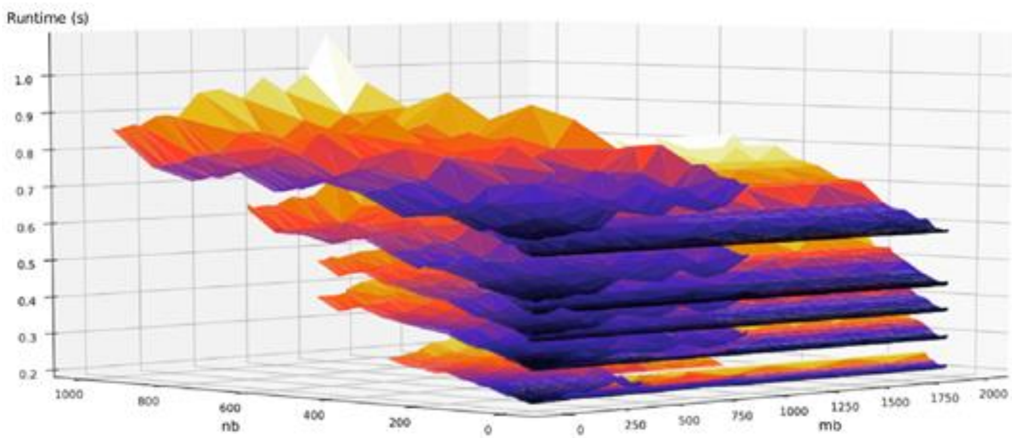
- Physics-informed uncertainty quantification
  - Poisson equation, wave equations, etc.

Example: semi-exhaustive search

$m = n = 5, mb = nb = 2, p$=2

- Parallel dense QR factorization in ScaLAPACK
- 2D block-cyclic layout
- Task is defined by [m, n] pair
- 3 Parameters: {mb, nb, p} (nprocs = pxq)





1 node, 24 cores
$m = n = 2000$
x-axis: mb, y-axis: nb
each layer is one *pxq* config.

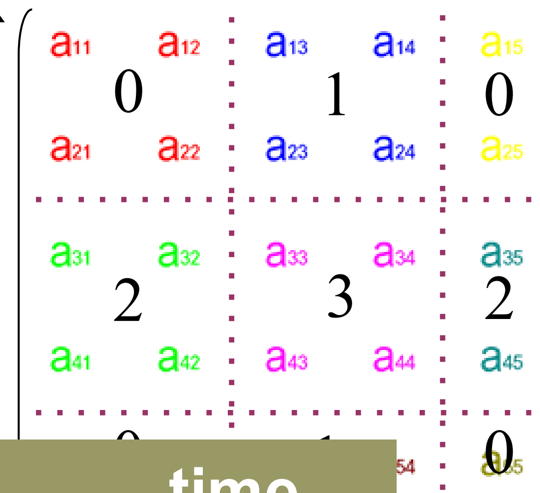
- Rule of thumb for best performance (from algorithm viewpoint)
  - Process grid as square as possible
  - Blocks as square as possible

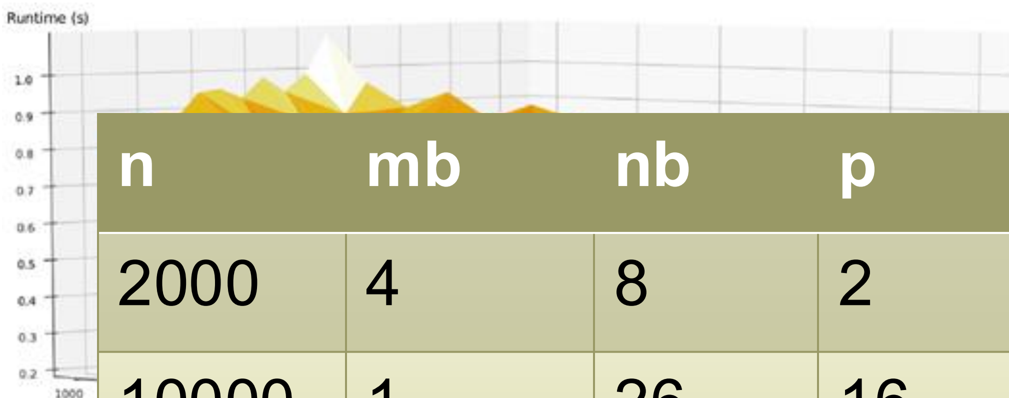
https://github.com/gptune/GPTune

Example: semi-exhaustive search

$m = n = 5, mb = nb = 2, p=2$

- Parallel dense QR factorization in ScaLAPACK
- 2D block-cyclic layout
- Task is defined by [m, n] pair
- 3 Parameters: {mb, nb, p} (nprocs = pxq)





n	mb	nb	p	q	time
2000	4	8	2	12	0.20
10000	1	26	16	192	1.61

- Rule of thumb for best performance (from algorithm viewpoint)
  - Process grid as square as possible
  - Blocks as square as possible

Characteristics of the optimization problems

- No analytical formulation of
  - objective function (runtime, memory, energy, …)
  - gradient
  - problem constraints

- Function evaluation == expensive application run (up to weeks!)
  - large variability related to hardware (e.g., network, disk I/O)

- Non-convex problems and non-linear constraints

- Discrete and continuous search spaces
  - Parameters can be *Real, Integer, Categorical*

Bayesian optimization

- Problem: $\min_x y(x)$, x : parameter configuration
- Bayesian statistical inference is an iterative model-based approach
  - versatile framework for black-box derivative-free global optimization

Sampling: $y_1 \ldots y_{n_0}$

Prior distribution

Surrogate Model $f(x)$

$\min f \approx \min y$

Update Posterior Distribution

Search Next:

$x_{n+1} \rightarrow y_{n+1}$

To optimize Acquisition Function

$p(y) = \mathcal{N}(\mu_0, \Sigma_0)$

Multivariate normal dist.

Bayes Rule

Update model with new data

Bayesian optimization

- Problem: $\min_{x} y(x)$, x : parameter configuration
- Bayesian statistical inference is an iterative model-based approach
  - versatile framework for black-box derivative-free global optimization

Sampling: $y_1 \dots y_{n_0}$

Prior distribution

Surrogate Model $f(x)$

$\min f \approx \min y$

Update Posterior Distribution

Search Next:

$x_{n+1} \rightarrow y_{n+1}$

To optimize Acquisition Function

Update model with new data

$p(y) = \mathcal{N}(\mu_0, \Sigma_0)$

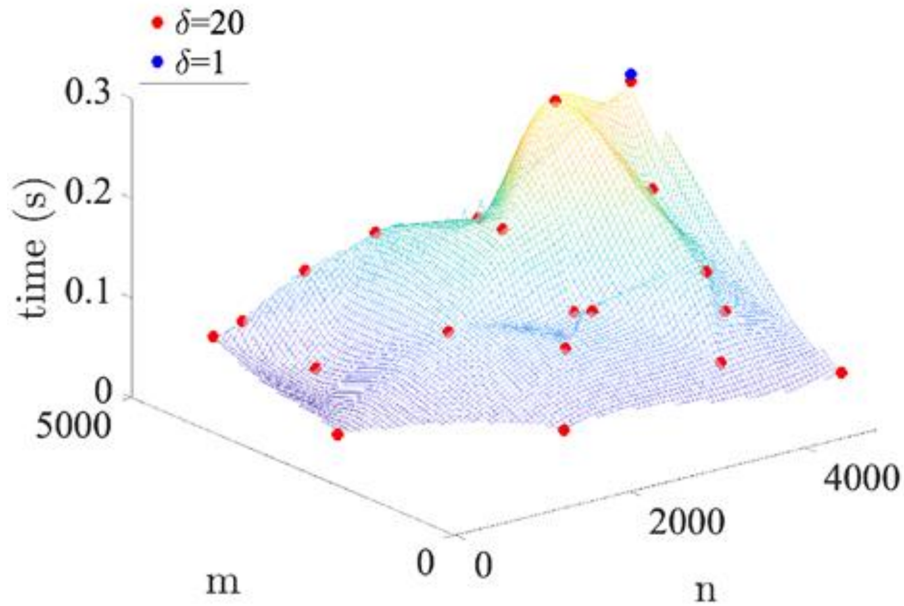
Multivariate normal dist.

Bayes Rule

- Sampler: Monte Carlo Sampling, Latin hypercube sampling, etc.

- Models: Kernels (RBF, Exponential, Matern, Wendland), LCM, multi-armed bandit, clustering, MCTS, transfer learning, etc.
- Optimization: LBFGS, MCMC

- Acquisition Function: Expected Improvement (EI), UCB, MSPE, etc.
- Optimization: GA, PSO, CMA-ES, LBFGS, etc.



Modeling

Gaussian Process Regression

"Gaussian Processes for Machine Learning", Rasmussen and Williams 2006

Modeling: Gaussian Process with e.g. RBF Kernels

- GP defines a distribution over functions, and inference takes place in the space of functions
  - Every finite subset of variables follows multivariate normal distribution
- GP is specified by the mean function and covariance function $k(x, x')$ (kernel)

$$f(x) \sim GP\big(\mu(x), k(x, x')\big)$$
$$\mu(x) = \mathbb{E}[f(x]$$
$$k(x, x') = \mathbb{E}[(f(x) - \mu(x))(f(x') - \mu(x'))]$$

- RBF kernel

$$k(x, x') = \ \sigma^2 \exp(-\sum_{i=1}^{D} \frac{(x_i - x_i')^2}{l_i})$$

covariance is large if two points are close

$$-l(\theta) = \frac{1}{2}\log|K| + \frac{1}{2}(y-\mu)^T K^{-1}(y-\mu) + \frac{n}{\pi}\log(2\pi) \qquad -\nabla l(\theta)_j = \frac{1}{2}\mathrm{tr}\big(K^{-1}\partial_j K\big) - \frac{1}{2}y^T K^{-1}\partial_j K K^{-1} y$$

Hyperparameter $\theta = (\sigma^2, l_i,$ etc.) optimization: MLE, MCMC, etc.

Modeling: GP model prediction

- Given s observation pairs:

$$X = [x^1, x^2, \dots, x^s] \quad Y = [y(x^1), y(x^2), \dots, y(x^s)]$$

Add new point x^*, posterior prob. distribution is : $p(y^*|X) = \mathcal{N}(\mu_*, \sigma_*^2)$
mean (prediction) and variance (confidence) for $y(x^*)$ are:

$$\mu_* = K(x^*, X)\, K(X, X)^{-1}\, Y$$
$$\sigma_*^2 = K(x^*, x^*) - K(x^*, X)\, K(X, X)^{-1} K(x^*, X)^T$$

Dimension of covariance matrix $K(X, X)$ = number of samples

Inversion of K and its log determinant can be quite expensive to compute

Modeling: Other Kernels, Optimizers and Algorithms

Tuner	Kernel, modeler	Model optimizer	Fast inversion	Acquisition function	Search algorithm
GPTune (single-objective)	RBF, Exponential, Matern, LCM, WGP	lbfgs	ScaLAPACK	EI, UCB, MSPE, q-UCB, q-EI	pso, ga, cmaes, l-bfgs-b, dual_annealing, trust-constr,shgo
GPTune (multi-objective)	RBF, Exponential, Matern, LCM	lbfgs	ScaLAPACK	EI, UCB, MSPE, q-UCB, q-EI, UCB-HVI†	nsga2, nspso, maco, moead
GPTuneBand	LCM and multi-armed bandit	lbfgs	ScaLAPACK	EI, UCB	pso, successive halving
cGP	Matern and clustering	lbfgs, MCMC	N/A	EI, MSPE	lbfgs
GPTuneHybrid	Matern and MCTS	lbfgs	N/A	GP-EI, GP-UCB, UCTS, Multinomial	lbfgs
GPTuneGeorge	RBF, Matern, LCM, sparse kernels	lbfgs, MCMC	HODLR, SuperLU_DIST	EI, UCB, MSPE, q-UCB, q-EI	pso, ga, cmaes, l-bfgs-b, dual_annealing, trust-constr,shgo

Table 2: Algorithm components of each supported tuner in GPTune. †: UCB-HVI uses the same search algorithm as in GPTune (single-objective).

Search Phase

- Where to place the new sample point?
  - Maximize certain Acquisition Function
  - Optimization based on surrogate model (easier!)

- Balance between exploitation and exploration
  - **Exploitation**: local search within promising regions
  - **Exploration**: global search of new regions with more uncertainty

- Given a new sample point, need quickly update the model

Acquisition Function (1)

- **Expected Improvement (EI)** – most commonly used AF

For a new point x^*, expected difference from current best is

$$I(x) = \max(y^{min} - f(x^*),\ 0)$$

$$EI(x^*) = \mathbb{E}\big(I(x)\big) = (y^{min} - \mu(x^*))\ \Phi(Z) + \sigma(x^*)\phi(Z)$$

- $Z = \frac{y^{min} - \mu(x^*)}{\sigma(x^*)}$
- $\Phi(.)$ = CDF of the standard normal distribution
- $\phi(.)$ = PDF of standard normal distribution

[Jones et al. 1998]

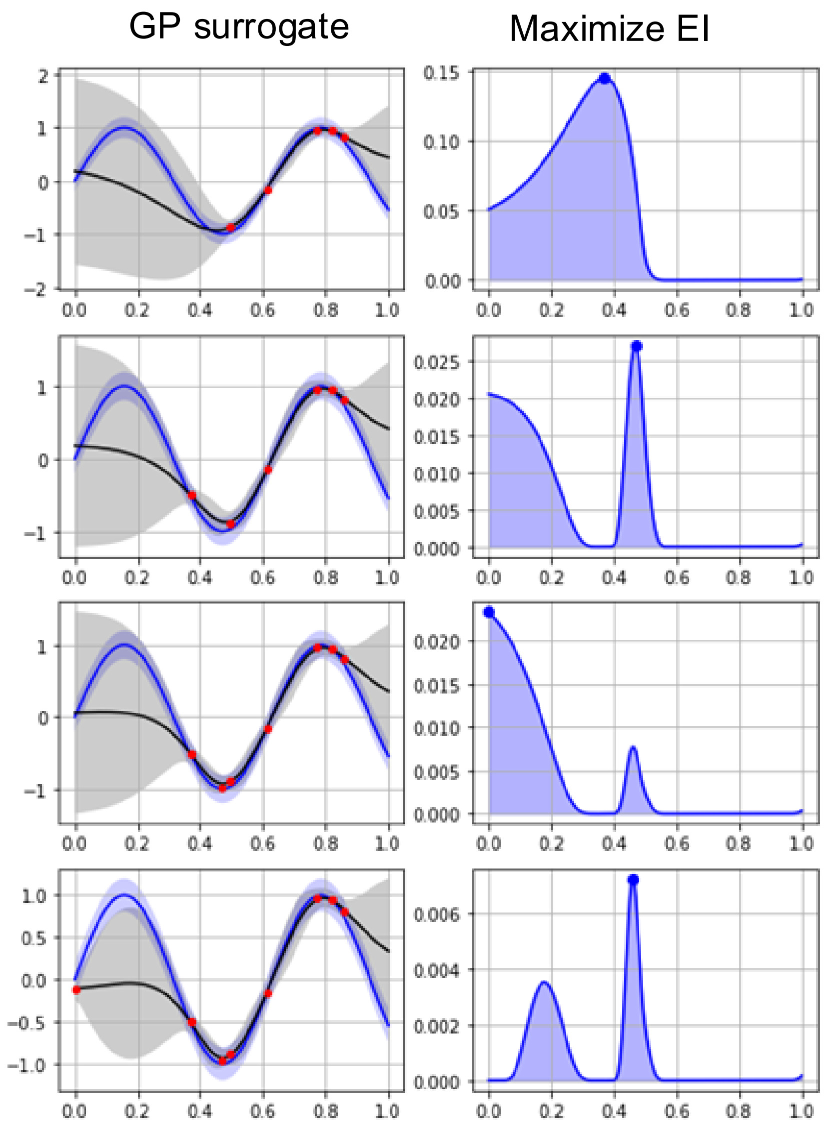
Acquisition Function (2)

- **Lower Confidence Bound (LCB)**

$$x^* = argmin_x \, LCB(x) = \mu(x) - \lambda \, \sigma(x)$$

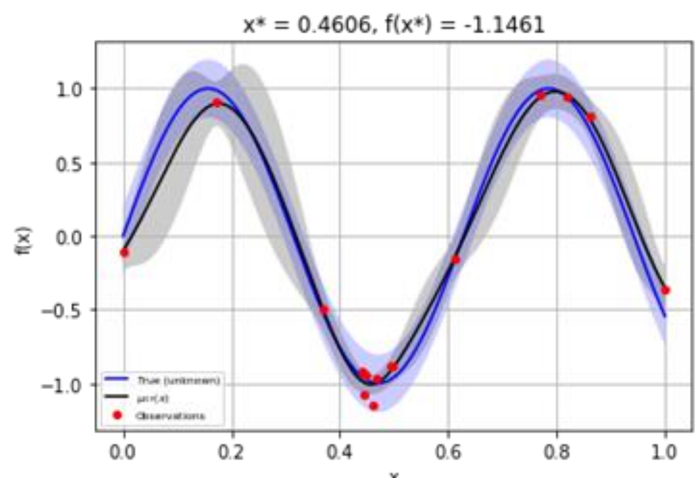
- $\mu(x) \rightarrow$ exploitation: favor points predicted to have low values
- $-\lambda\sigma(x) \rightarrow$ exploration: subtracting uncertainty, prefer points where the lower bound could be much smaller than the mean
- $\lambda \rightarrow$ trade-off parameter controlling exploration vs exploitation

1D example: black-box function $y(x) = sin(10x)$



5 initial samples
4 additional steps

- Blue line: true function
- Red dots: function evaluations
- Black line: mean function of the fitted surrogate model
- Grey shaded area is 95% confidence interval



Search: Other Acquisition Functions and Algorithms

Tuner	Kernel, modeler	Model optimizer	Fast inversion	Acquisition function	Search algorithm
GPTune (single-objective)	RBF, Exponential, Matern, LCM, WGP	lbfgs	ScaLAPACK	EI, UCB, MSPE, q-UCB, q-EI	pso, ga, cmaes, l-bfgs-b, dual_annealing, trust-constr,shgo
GPTune (multi-objective)	RBF, Exponential, Matern, LCM	lbfgs	ScaLAPACK	EI, UCB, MSPE, q-UCB, q-EI, UCB-HVI†	nsga2, nspso, maco, moead
GPTuneBand	LCM and multi-armed bandit	lbfgs	ScaLAPACK	EI, UCB	pso, successive halving
cGP	Matern and clustering	lbfgs, MCMC	N/A	EI, MSPE	lbfgs
GPTuneHybrid	Matern and MCTS	lbfgs	N/A	GP-EI, GP-UCB, UCTS, Multinomial	lbfgs
GPTuneGeorge	RBF, Matern, LCM, sparse kernels	lbfgs, MCMC	HODLR, SuperLU_DIST	EI, UCB, MSPE, q-UCB, q-EI	pso, ga, cmaes, l-bfgs-b, dual_annealing, trust-constr,shgo

Table 2: Algorithm components of each supported tuner in GPTune. †: UCB-HVI uses the same search algorithm as in GPTune (single-objective).

GPTune Software: github.com/gptune/GPTune

- Python interface, leverage existing Python packages
  - GPy, scipy, scikit-learn, scipy, scikit-optimize, MPI4py, pymoo, pygmo …
- HPC code for parallel covariance matrix computation/inversion: BLAS, ScaLAPACK, ButterflyPACK, George, SuperLU_DIST
- Part of xSDK, E4S, spack

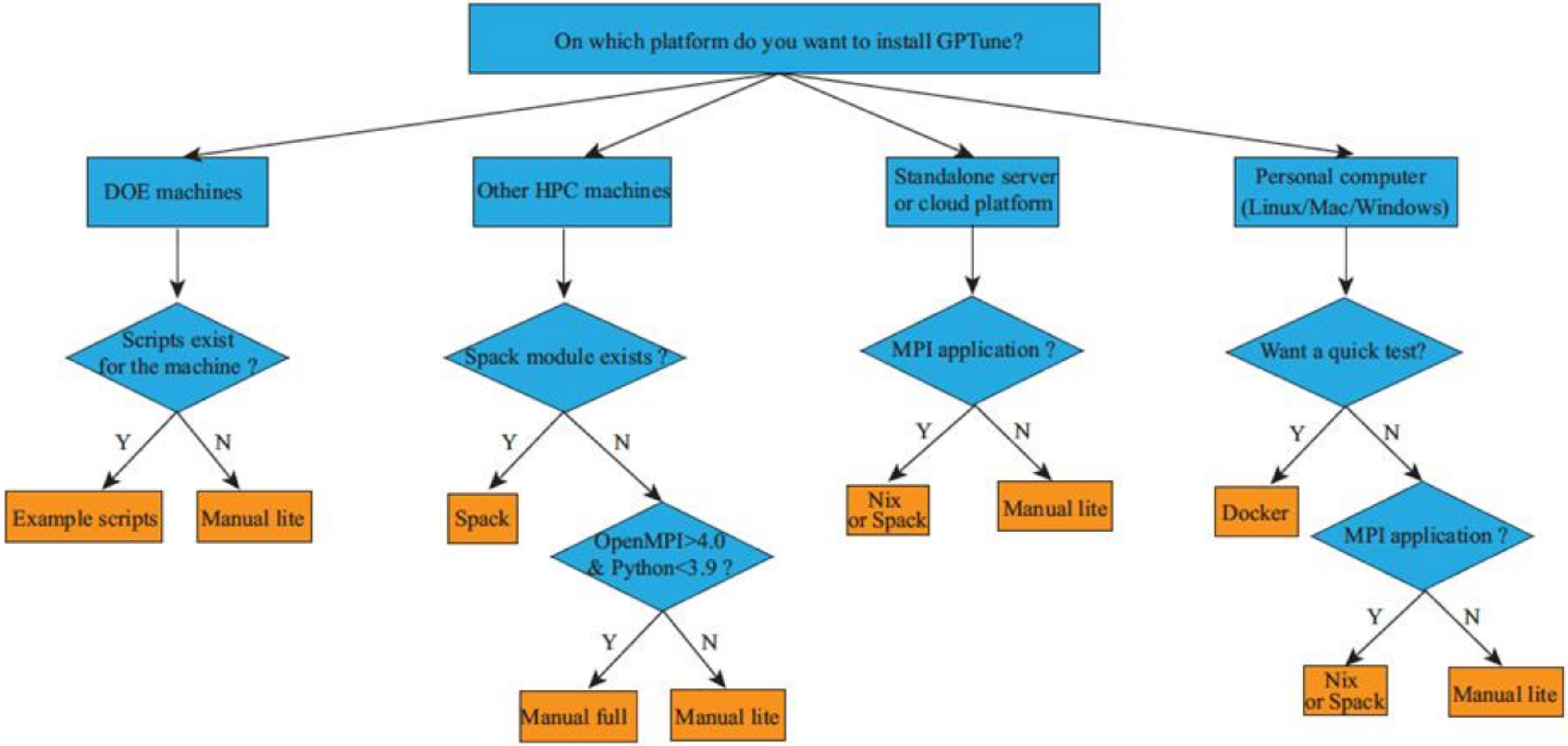
- User input:
  - Task parameter input space ( ): the space of tasks parameters
    - QR example: m = n = 20000
  - Tuning parameter space ( ): the space of tuning parameters
    - (categorical, integer, real), and ranges
    - QR example: {mb, nb, nprow}
  - Output space ( ): the space of objective function values
    - runtime
  - Define application as a black-box function
    - Python to C / Fortran interface

(Optionally)

- Define constraints in parameter search space
- Define performance models
- Choose a search method
- ……

GPTune Software: Installation Options

- Manual full, manual lite, Spack, Nix, Docker
- For quick test, use the Docker image https://ggle.io/5X02



https://github.com/gptune/GPTune

Easy-to-Use Interface in Python

A model problem to illustrate user interface

$$y(t,x) = sin(10x + 2t)$$

3 tasks: $t = 0, 0.5, 1.0$

Use Python classes to:

- Express arbitrary complex sets of constraints
- Provide arbitrary sets of tuning choices, as first class objects

```
from autotune.problem import *
from autotune.space import *
from autotune.search import *
from GPTune.gptune import *
```

Import internal Python classes

```
….
input_space = Space( [Real(0., 10., name="t")] )
parameter_space = Space( [Real(0., 1., name="x")] )
output_space = Space( [Real(-Inf, Inf, name="y")] )
```

Define application parameters (categorical, integer, real) and ranges

```
def objectives(point):
    t = point['t']
    x = point['x'];
    f = np.sin(10*x + 2*t)
    return [f]
```

Define application as a black-box function

```
constraints = {"cst1": "x >= 0. and x <= 1."}
```

***Define constraints in parameter search space** [optional]*

```
def analytical_model1(point):
    f = np.sin(10*x + 2*t)
    return [f*(1+np.random.uniform()*0.1]
models = {'model1': analytical_model1}
```

***Define performance models** [optional]*

```
constants={"AAA":aaa,"BBB":bbb}
```

***Define constants** [optional]*

```
problem = TuningProblem(input_space, parameter_space,
output_space, objectives , constraints, models, constants)
```

```
options = Options()
options['XXX'] = 'YYY'
```

Set GPTune options

```
gt = GPTune(problem, computer, data, options, … )
gt.MLA(NS, giventask, NI, NS1=int(NS/2))
```

Use different tuners

https://github.com/gptune/GPTune

Unified interface to many tuners

- External tuners: opentuner, hpbandster, etc.
- Our tuners: GPTune, GPTuneBand, cGP, GPTuneHybrid, etc.

```
# Define objectives, constraints, spaces, options, computer
...
# Define the "autotune" interface to all tuners
problem = TuningProblem(IS, PS, OS, objectives, constraints)
# Call different tuners
if(TUNER_NAME=='GPTune'):
    gt = GPTune(problem, computer=computer, options=options)
    (data, model, stats) = gt.MLA(NS=NS, Igiven=giventask)
if(TUNER_NAME=='opentuner'):
    (data,stats)=OpenTuner(T=giventask,NS=NS,problem,computer)
if(TUNER_NAME=='hpbandster'):
    (data,stats)=HpBandSter(T=giventask,NS=NS,problem,computer)
if(TUNER_NAME=='GPTuneBand'):
    gt = GPTune_MB(problem, computer=computer,NS=1,options=options)
    (data, stats, data_hist)=gt.MB_LCM(NS=1,Igiven=giventask)
if(TUNER_NAME=='cgp'):
    (data,stats)=cGP(T=giventask,problem,computer,options=options)
```

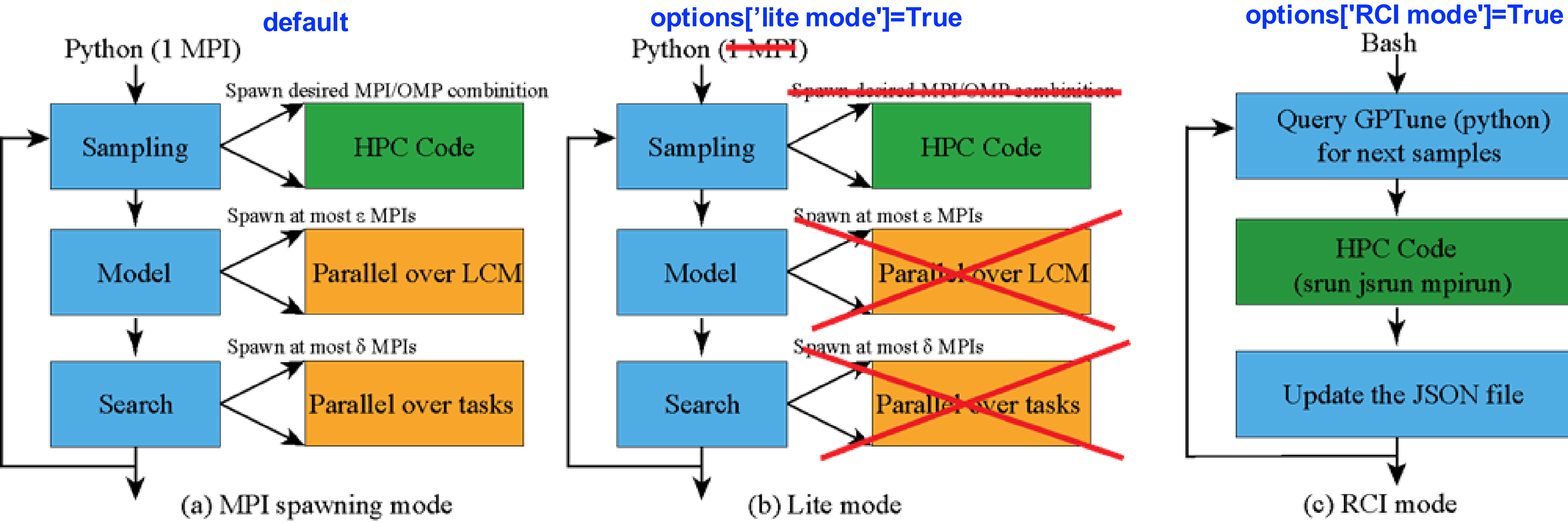
Variants of Algorithms and Features in GPTune

- Multi-task learning algorithm (GPTune MLA): $y(t,x)$, t denotes task parameters
- Transfer learning algorithm (GPTune TLA): from trained $y(t_1,x)$ to improve $y(t_2,x)$
- Multi-fidelity tuning (GPTuneBand): $y(s,t,x)$, s denotes a fidelity level
- Multi-objective tuning (GPTune): $y^l(t,x)$ denotes the l^{th} objective function
- Mixed-variable tuning (GPTuneHybrid)
- Non-smooth objectives (cGP)
- Database, visualization, and crowd tuning (https://gptune.lbl.gov/repo/dashboard/)
- Other features: parallel implementation, checkpointing, sensitivity analysis, importance analysis, coarse performance models

	Tuner	MLA	TLA	multi-fidelity	multi-objective	parallel tuning	database logging	checkpoint & restart
	GPTune	✓	✓		✓	✓	✓	✓
GPTune variants	GPTuneBand	✓		✓		✓	✓	✓
	cGP						✓	
	GPTuneHybrid						✓	
External	HpBandSter						✓	
	OpenTuner						✓	

Parallel architecture of GPTune

- Three execution modes of GPTune: MPI spawning mode, lite mode and reverse communication interface (RCI) mode

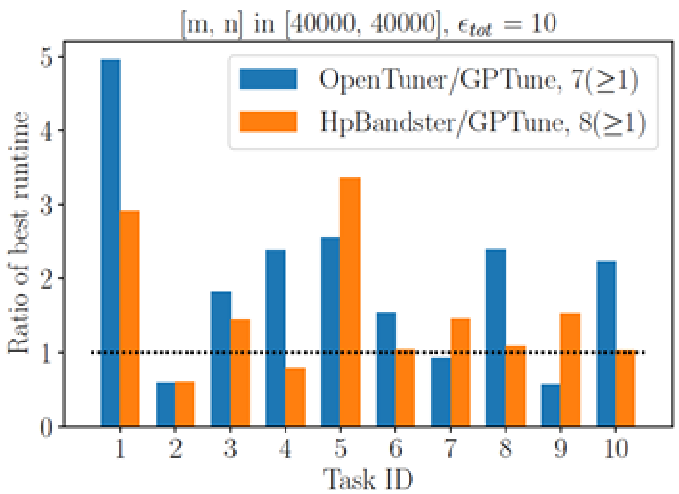


	spawning	lite	RCI
call application	from python	from python	from bash
parallel model/search	yes	no	yes
code instrumentation	yes	no	no

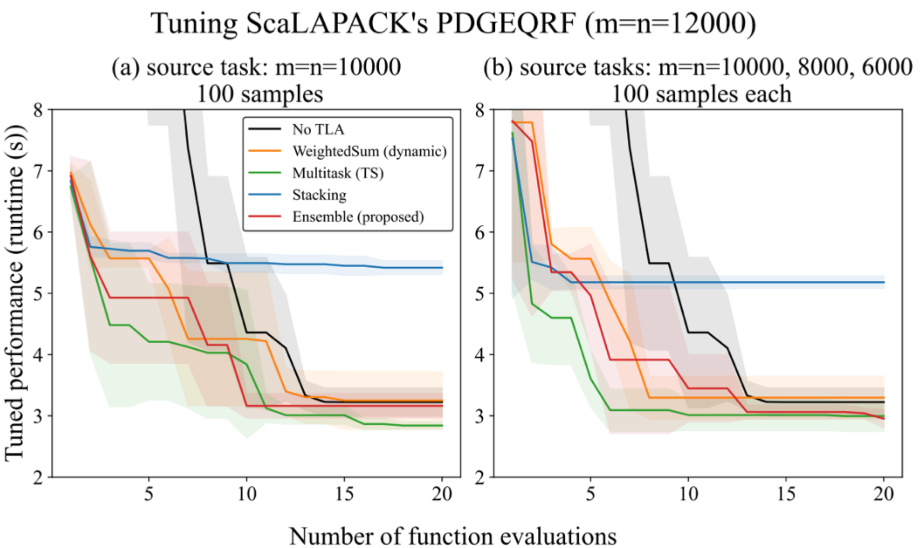
https://github.com/gptune/GPTune

Feature 1: Multitask Learning (MLA) and Transfer Learning (TLA)

- Consider a set of correlated objective functions $\{y_i(X)\}_{i\in 1..\delta}$, we want to tune them all together (MLA) or use the models of $y_i, i = 1,..,\delta-1$ to tune y_δ (TLA).
- Both MLA and TLA increase the effective number of samples



MLA tuning runtime of ScaLAPACK QR of 10 matrices

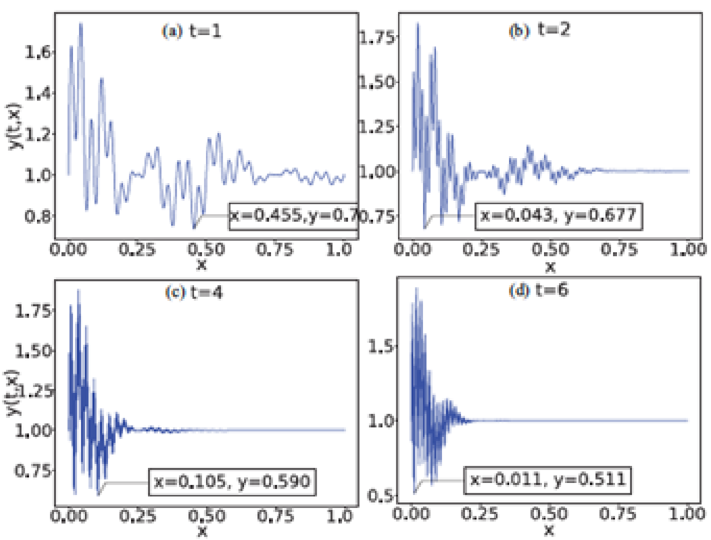


TLA tuning runtime of ScaLAPACK QR of 1 target matrix from 1 source matrix

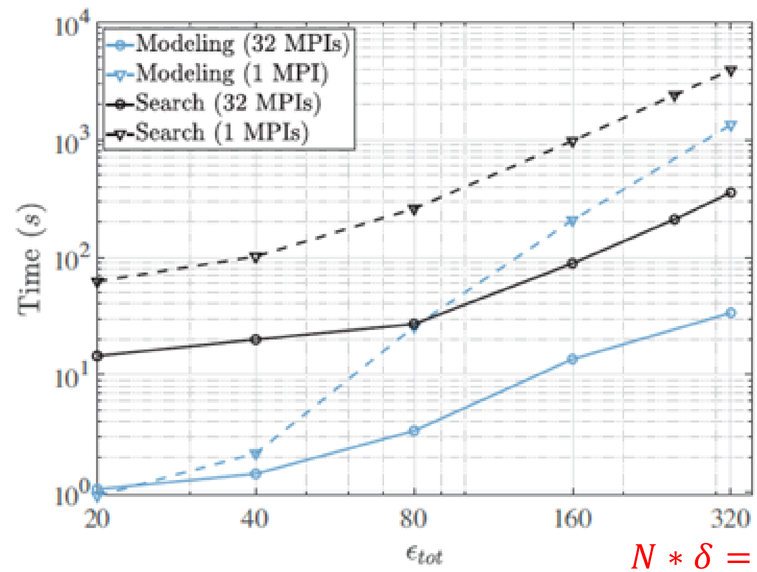
Feature 2: Fast K Computation: ScaLAPACK

Consider an analytical function, t, x: task and tuning parameters, $\delta = 20$ tasks N : number of samples per task.

$$y(t,x) = 1 + e^{-(x+1)^{t+1}} \cos(2\pi x) \sum_{i=1}^{3} \sin\left(2\pi x (t+2)^i\right)$$



Objective functions



$N * \delta = 6400$

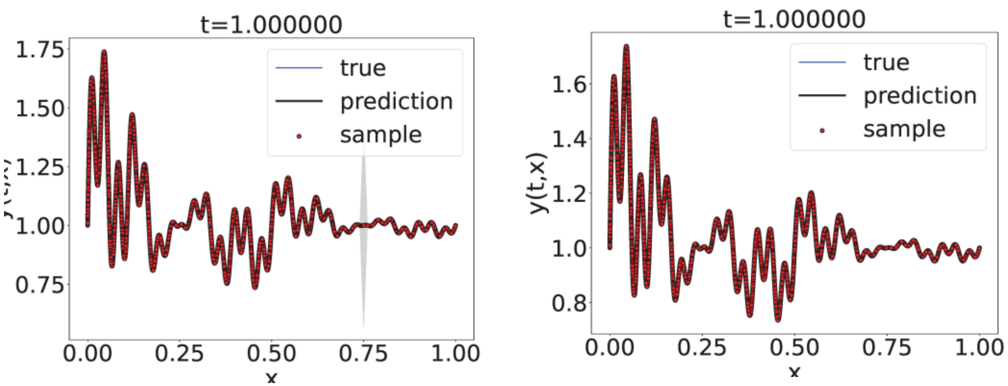
Parallel performance

* GPTune: Multitask learning for autotuning exascale applications, Yang Liu, Wissam M Sid-Lakhdar, Osni Marques, Xinran Zhu, Chang Meng, James W Demmel, Xiaoye S Li, PPoPP'21

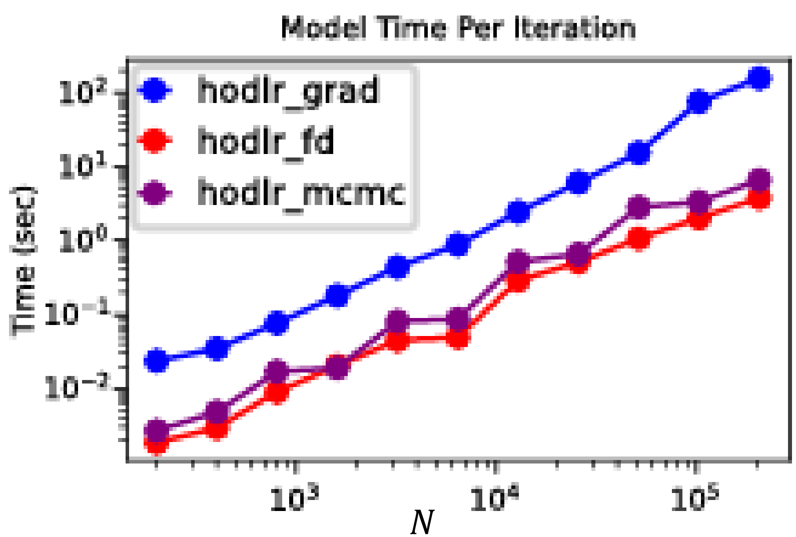
Feature 2: Fast K Computation: Hierarchical Matrices

Idea: leverage low-rank compression to exploit data-sparsity of dense kernel matrices

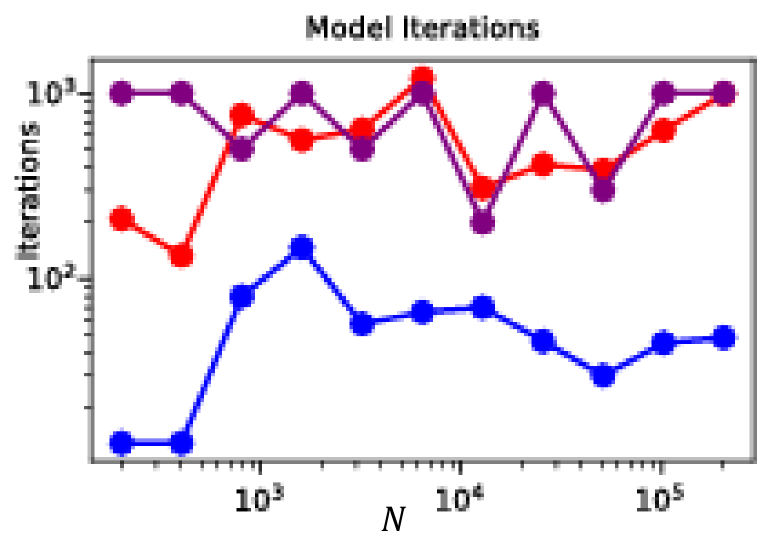
Setup: generate N random samples, build the GP model, and predict the $(N+1)^{th}$ sample



$N = 102400.$
Left: LBFGS.
Right: MCMC



$-l(\theta)$ and $-\nabla l(\theta)_j$ per MCMC or LBFGS iteration



Iteration counts of MCMC or LBFGS

Feature 2: Fast K Computation: Sparse Kernel Computation via SuperLU_DIST

$$k(x,x') = \sigma^2 \exp\left(-\sum_{1=1}^{D}\frac{\left(x_i - x_i^i\right)^2}{l_i^2}\right) \times \left(1-\frac{r}{r_c}\right)^4\left(\frac{4r}{r_c}+1\right)$$

RBF **Wendland**

Negative log-likelihood: (LBFGS, MCMC, etc.)

$$-l(\theta) = \frac{1}{2}\log|K| + \frac{1}{2}(y-\mu)^T K^{-1}(y-\mu) + \frac{n}{\pi}\log(2\pi)$$

The gradient (LBFGS):

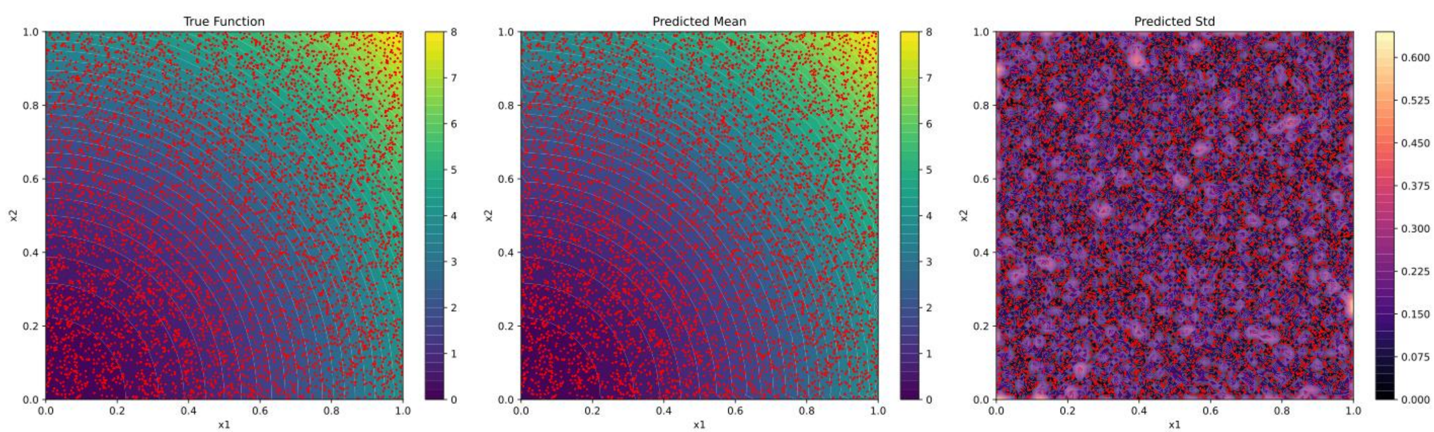
$$-\nabla l(\theta)_j = \frac{1}{2}\mathrm{tr}\left(K^{-1}\partial_j K\right) - \frac{1}{2}y^T K^{-1}\partial_j K K^{-1} y$$

Randomized Hutchinson trace estimator:

$$\mathrm{tr}\left(K^{-1}\partial_j K\right) \approx \frac{1}{m}\sum_{l=1}^{m} u_l^T K^{-1}\partial_j K u_l$$

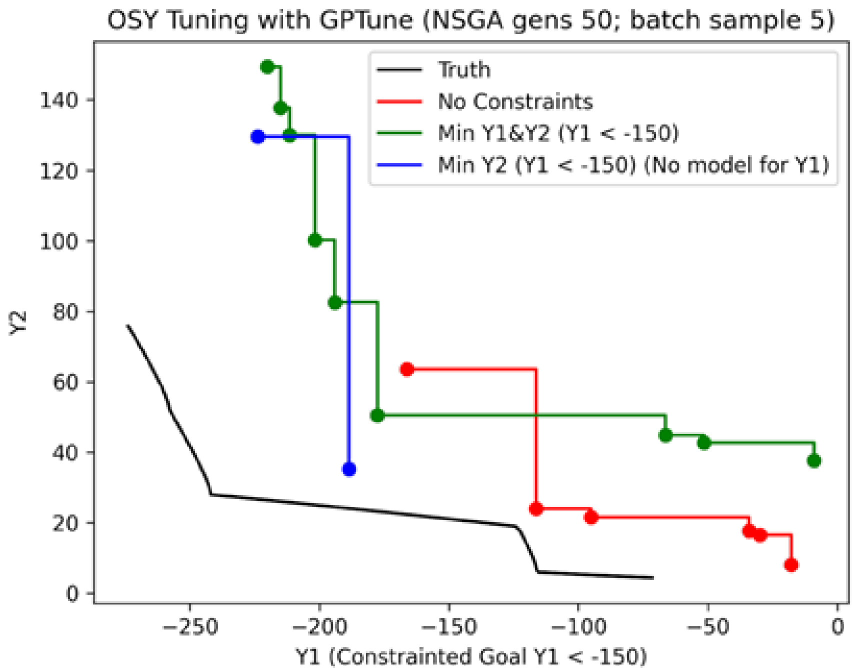
Leverage the newly developed SuperLU_DiST python interface via file-based IO

$N = 6400$



Feature 3: Multi-objective tuning

- Part II.2 of https://ggle.io/5X02
- Pareto optimal: no other PS points dominate over this point in all objectives.
- Unconstrained and constrained multi-objective tuning **output_space = Space([y1, y2, ...])**
  - No constraint, optimize: **yk = Real(float("-Inf"), float("Inf"), name="yk")**
  - Constrained, optimize: **yk = Real(float("-Inf"), 10.0, name="yk")**
  - Constrained, not-to-optimize: **yk = Real(float("-Inf"), 10.0, name="yk", optimize=False)**



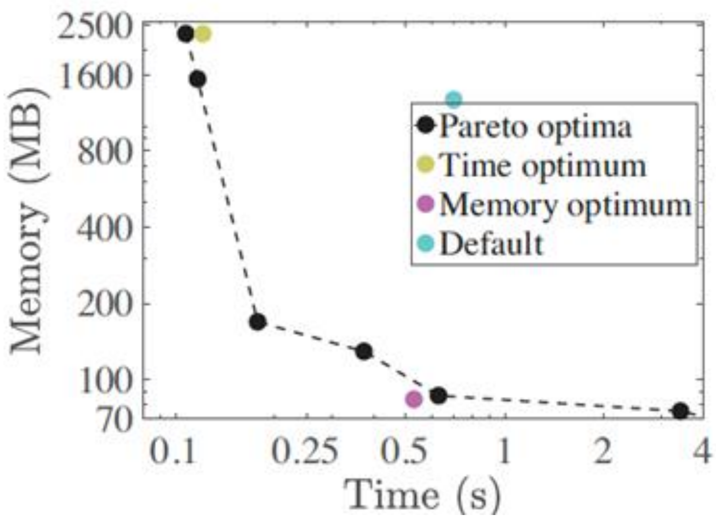
https://github.com/gptune/GPTune

Feature 3: Multi-objective tuning of SuperLU_DIST

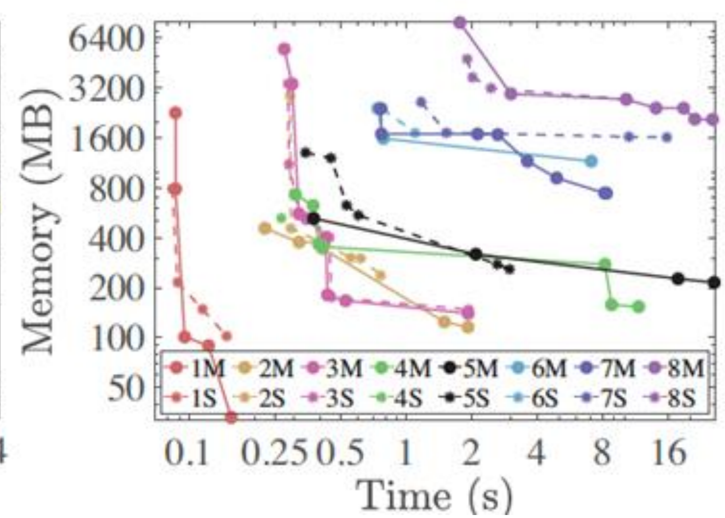
- Unconstrained two objective tuning
- Part II.2 of https://ggle.io/5X02

Multi-objective EGO: One LCM per objective, NSGA-II in search

- $\mathbb{IS}$=[matrix name] $\mathbb{PS}$=[COLPERM, NSUP, NREL, nprows].
- Multi-objective: $\mathbb{OS} = [time, memory]$, single-objective: $\mathbb{OS} = [time]$ or $[memory]$. MLA $\delta = 8$ or single-task. 256 cores.
- Pareto optimal: no other $\mathbb{PS}$ points dominate over this point in both objectives.



single-task



MLA

Feature 4: Incorporation of Coarse Performance Model

- Part II.4 of https://ggle.io/5X02
- An analytical formula can significantly improve tuning performance

A coarse performance model $\tilde{y}(t,x)$ (per task) can be built into $\mathbb{PS}$: $x \rightarrow [x, \tilde{y}(t,x)]$. $\tilde{y}(t,x)$ can also be parameterized.

A simple performance model for PDGEQRF

$$\tilde{y}(t,x) = C_{flop} \times t_{flop} + C_{msg} \times t_{msg} + C_{vol} \times t_{vol} \quad (1)$$

with the number of floating point operations C_{flop}, the number of messages C_{msg} and the volume of messages C_{vol}

$$C_{flop} = \frac{2n^2(3m-n)}{2p} + \frac{b_r n^2}{2p_c} + \frac{3b_r n(2m-n)}{2p_r} + \frac{{b_r}^2 n}{3p_r} \quad (2)$$

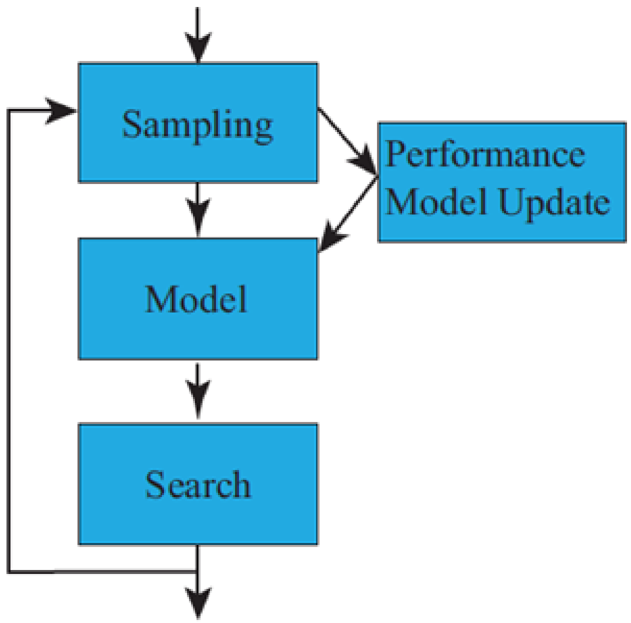
$$C_{msg} = 3n \log p_r + \frac{2n}{b_r} \log p_c \quad (3)$$

$$C_{vol} = \left(\frac{n^2}{p_c} + b_r n\right) \log p_r + \left(\frac{mn - n^2/2}{p_r} + \frac{b_r n}{2}\right) \log p_c \quad (4)$$

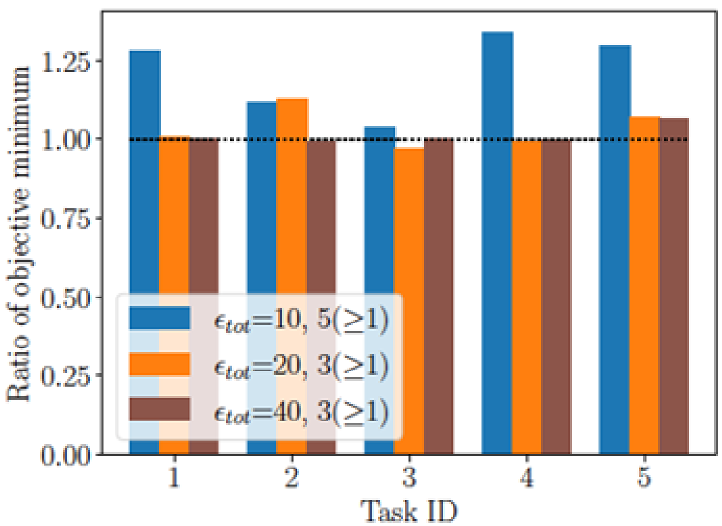
Feature 4: Incorporation of Coarse Performance Model

- An analytical formula can significantly improve tuning performance
- The formula can have hyperparameters as well

A coarse performance model $\tilde{y}(t,x)$ (per task) can be built into $\mathbb{PS}$: $x \rightarrow [x, \tilde{y}(t,x)]$. $\tilde{y}(t,x)$ can also be parameterized.



(a) Incorporate performance model



(b) PDGEQRF: ratio between best runtime with and without the performance model

Feature 5: Multi-fidelity tuning with GPTuneBand

- Part II.5 of https://ggle.io/5X02

Multi-fidelity tuning of hypre

- Convection-diffusion equation on a n^3 grid:
$$-c\Delta u + a\nabla \cdot u = f$$
- $\mathbb{IS}$=[a, c], $\mathbb{PS}$=12 integer/real/categorical, $\mathbb{OS}$=[runtime]
- Fidelity/budget $\sim n^3$.

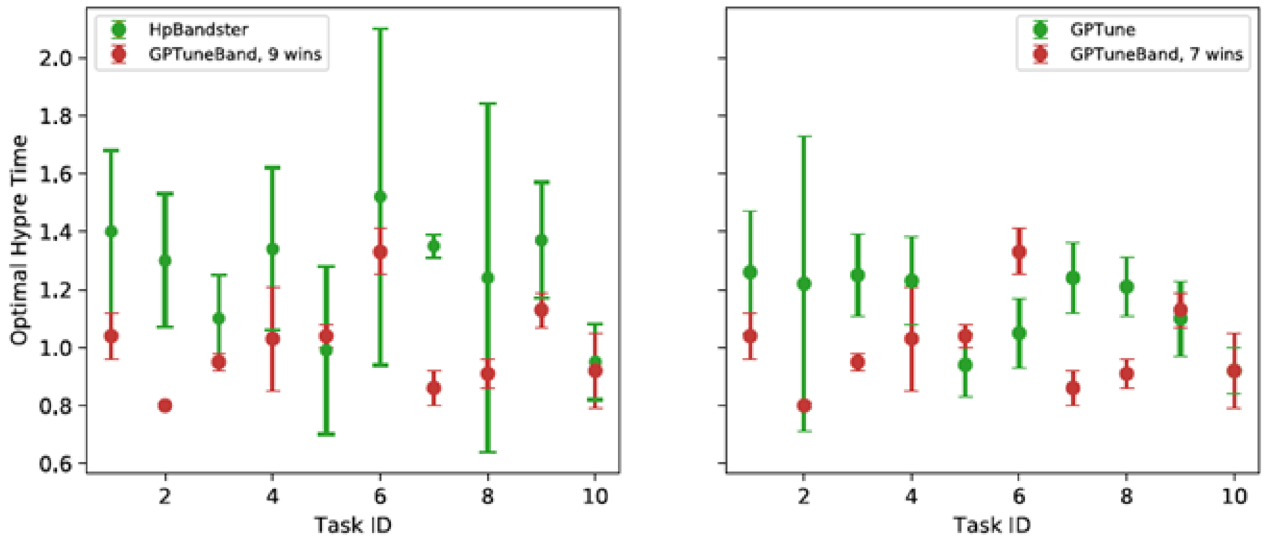
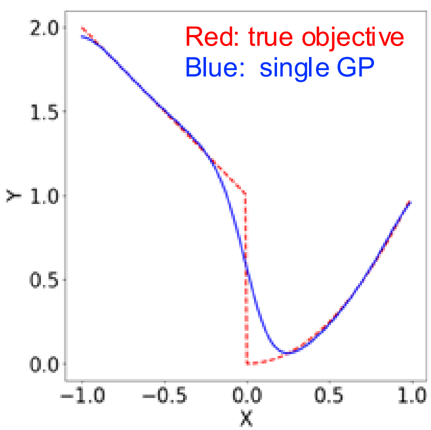
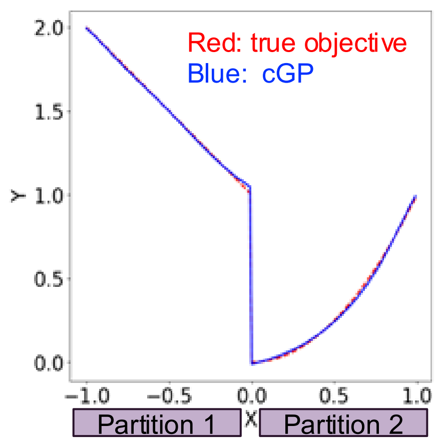


Figure: Comparison of GPTuneBand (multi-fidelity, MLA), GPTune (MLA), and HpBandster (multi-fidelity)

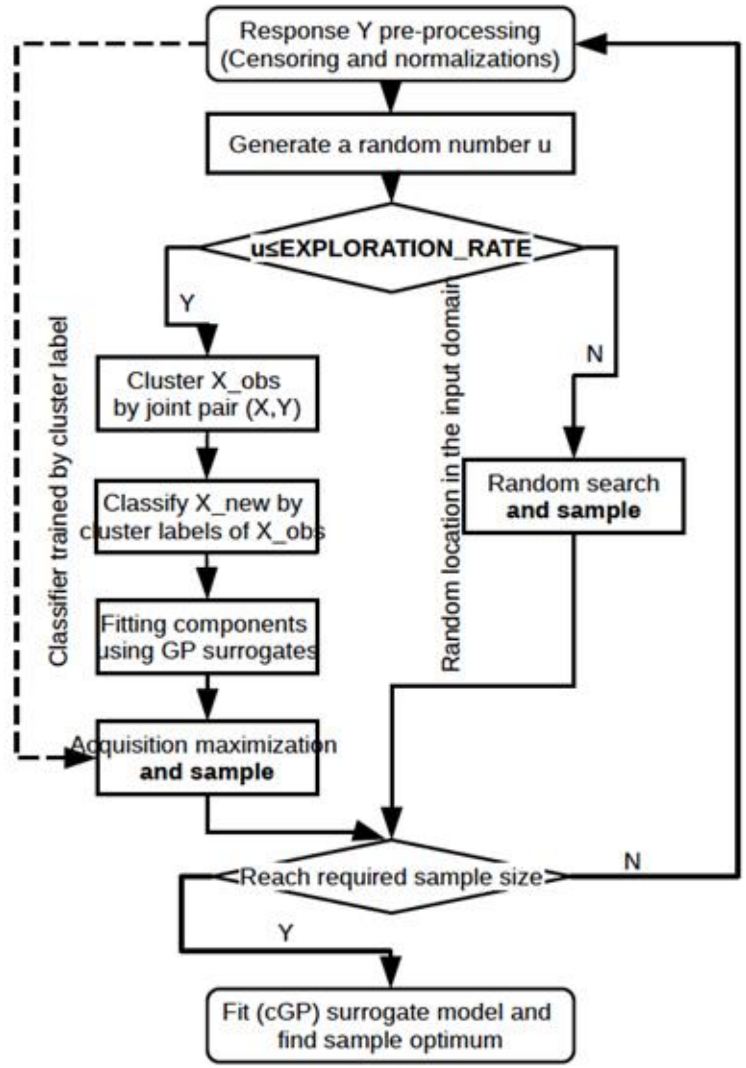
Feature 6: Handling non-smooth objectives with cGP

- Non-smooth objective (discontinuity): single GP performance poor
- cGP (clustered GP): auto-partitioning of the parameter domain and build GP on each





- cGP key options:
  - N_COMPONENT: maximal number of clusters.
  - EXPLORATION_RATE: probability of generating new samples from the acquisition instead of random



cGP workflow

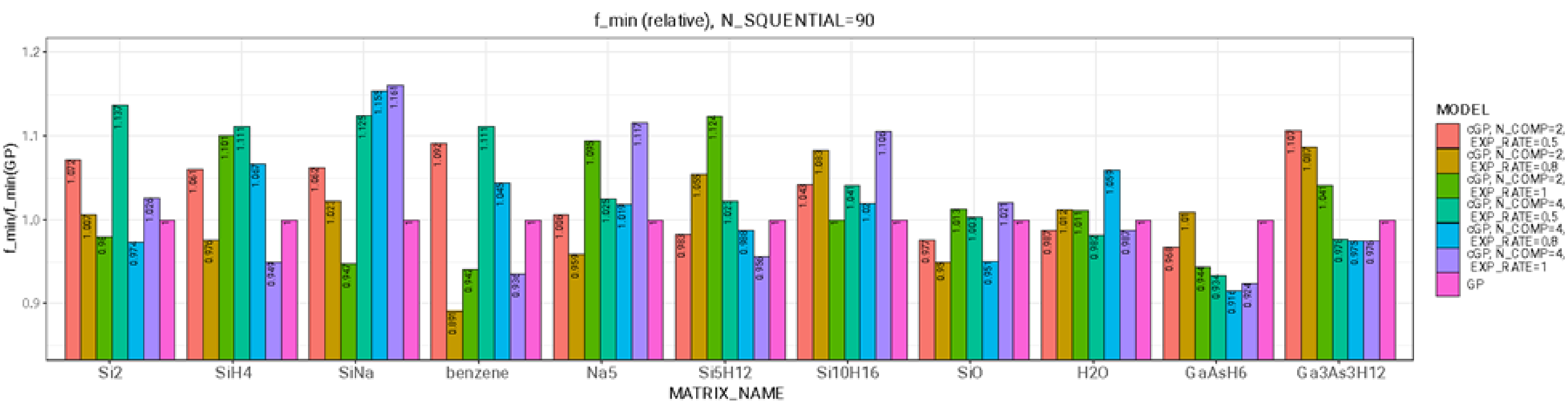
https://github.com/gptune/GPTune

Feature 6: Handling non-smooth objectives with cGP

- Part II.6 of https://ggle.io/5X02

Tuning SuperLU_DIST factorization time

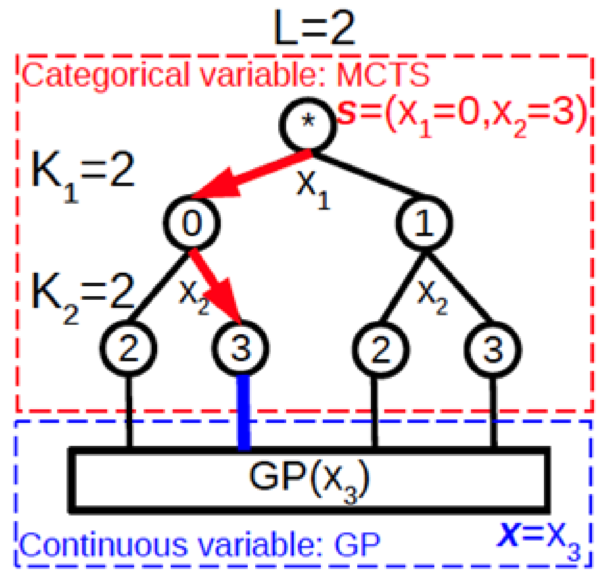
- $\mathbb{IS}$=[matrix name] $\mathbb{PS}$=[NSUP, NREL, nprows].
- Objective: $\mathbb{OS} = [\mathit{time}]$. Single-task. 256 cores.
- N_COMPONENT and EXPLORATION_RATE are performance critical



cGP outperforms single GPs by up to 11.5% for 90% of all 11 matrices

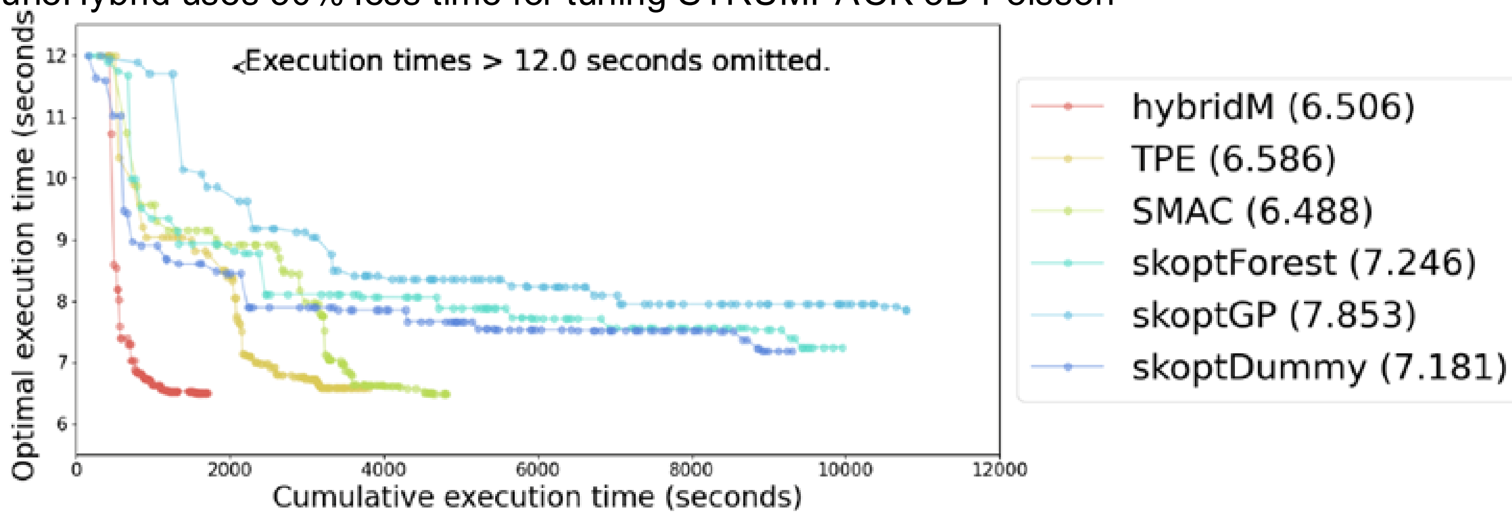
Feature 7: Handling mixed-variables with GPTuneHybrid

- Part II.7 of https://ggle.io/5X02
- Mixed-variable (categorical variables + continuous variables): GP performance poor
- GPTuneHybrid: Monte Carlo Tree Search (MCTS) for categorical variables and GP for continuous variables.



rid workflow

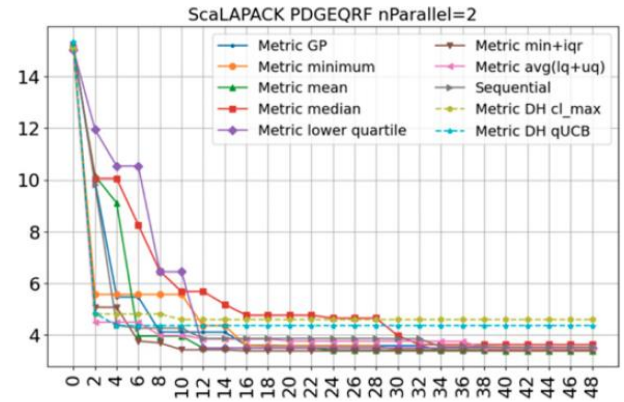
GPTuneHybrid uses 50% less time for tuning STRUMPACK 3D Poisson



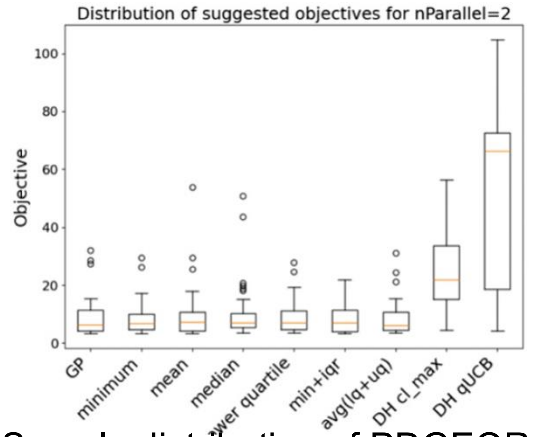
Feature 8: Multiple Sample Generation in the Search Phase

- Goal: generate multiple new samples per BO iteration
  - Reduce the number of hyperparameter optimization model stages (q-EI, not Liar strategy)
  - Leverage distributed-memory parallelism to run multiple function evaluations

- Methods:
  - Multi-point acquisition functions (e.g., q-EI): providing joint expected improvement of q samples.
  - GP-mean-based Liar strategy: use **model predicted mean** as the function value, update the model, after q steps, evaluate the true function.
  - Constant Liar strategies: use **statistical metrics (e.g., min, max, mean, etc.) of existing samples** as the function value, update the model, after q steps, evaluate the true function.



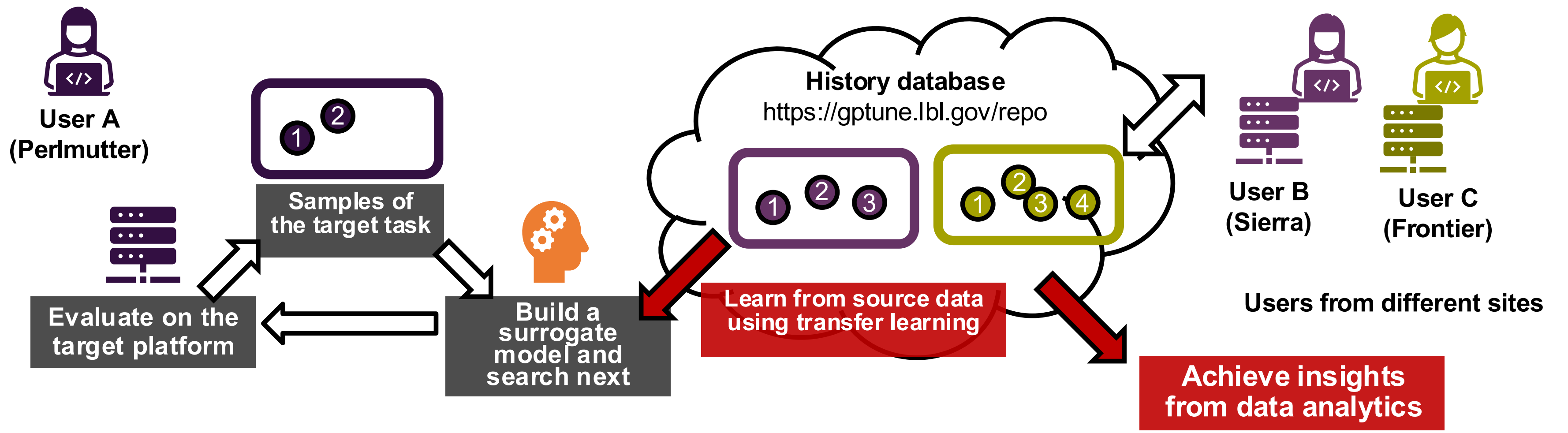
Convergence of PDGEQRF tuning with GPTune and DeepHyper



Sample distribution of PDGEQRF tuning with GPTune and DeepHyper

* Parallelizing autotuning for HPC applications: Unveiling the potential of the speculation strategy in Bayesian optimization, AP Dieguez, S Ockerman, T Aikman, Y Cho, Y Liu, KZ Ibrahim, IJHPCA, 2025

Feature 9: Crowd Tuning: Harnessing the Crowd for Autotuning



- Observation: for popular applications, there are multiple users need tuning
- Our goals:
  1. Build an infrastructure to collect obtained knowledge
  2. Leverage previously collected data for better tuning (fewer evaluations)
  3. Provide useful data analytics from the shared database (history database)

* Harnessing the crowd for autotuning HPC applications, Y. Cho, J. Demmel, J. King, X. S. Li, Y. Liu, H. Luo, IPDPS’23

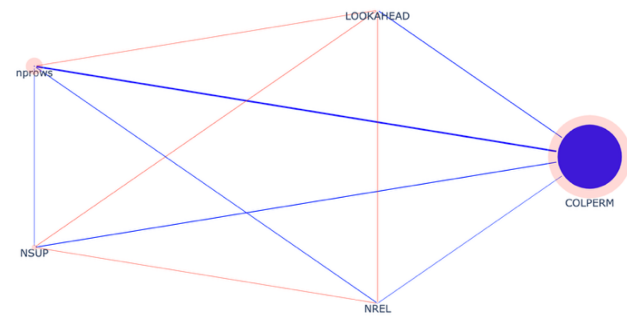
* Enhancing autotuning capability with a history database, Y.Cho, J. Demmel, X. S. Li, Y. Liu, H. Luo, MCSoC’21

Feature 9: Crowd Tuning: Sensitivity Analysis

- Sobol's sensitivity analysis: variance-based analysis to estimate the sensitivity of each variable and interactions of the variables (variable = tuning parameter)
  - **S1: the influence on a single parameter**
  - **ST: the influence of a parameter that includes all interactions with other parameters**
  - **S1/ST_conf: confidence interval**

Matrix: Si5H12	S1	S1_conf	ST	ST_conf
COLPERM	0.795619	0.061109	0.860267	0.070658
LOOKAHEAD	0.004315	0.009732	0.011590	0.002510
nprows	0.109990	0.049130	0.169862	0.032508
NSUP	0.008409	0.021093	0.055794	0.015733
NREL	0.004392	0.008828	0.011989	0.003025

Example result of SuperLU_DIST



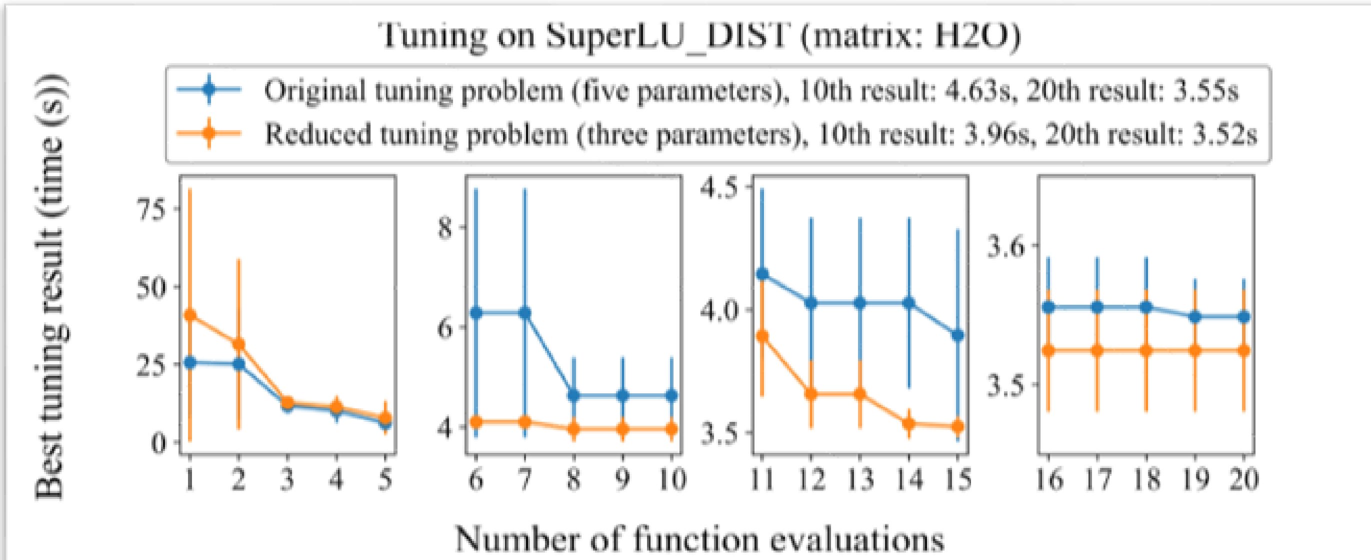
Example graphical representation on the web dashboard (matrix: Si5H12)
(thanks to Mohammad Zaeed and Tanzima Islam)

- We can reduce the tuning space based on the analysis

Note: analysis results are based on surrogate-based sampling, so the results can vary depending on the sampling setting

- Sobol analysis: https://en.wikipedia.org/wiki/Variance-based_sensitivity_analysis
- We internally use SALib for the sensitivity analysis computation: https://salib.readthedocs.io/en/latest/

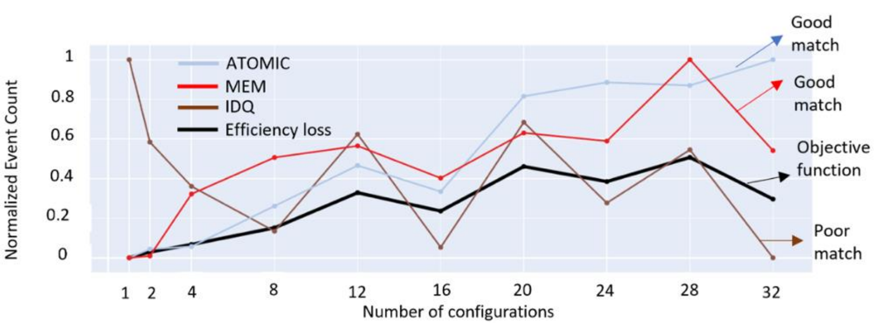
Feature 9: Crowd Tuning: Sensitivity Analysis



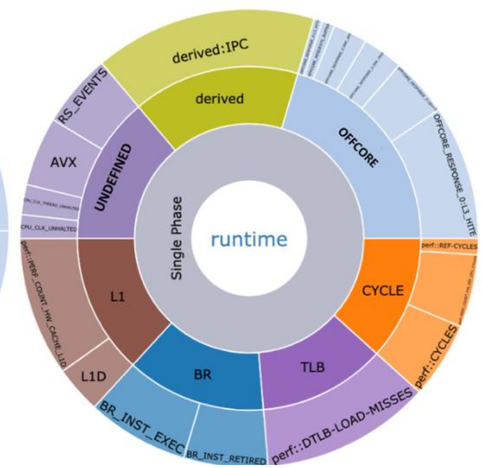
- SuperLU_DIST on four Haswell nodes in Cori
  - **Original tuning parameters: COLPERM, LOOKAHEAD, nprows, NSUP, NREL**
  - **Reduced tuning parameters: COLPERM, nprows, NSUP**

Feature 9: Crowd Tuning: Importance Analysis

- Incorporated Dashing(*)'s analysis/visualization in the history database
  - **Importance analysis on tuning parameters or hardware performance counters (and groups of them)**
  - **Interactive visualization on web (examples: PLASMA's DGEMM/DG at https://gptune.lbl.gov/repo)**
  - **Visualization for trend matching**



NIMROD (analysis on tuning parameters)



PLASMA's DGEMM (analysis on performance counters)

Analysis and Visualization of Important Performance Counters To Enhance Interpretability of Autotuner Output

Mohammad Zaeed[1], Tanzima Z. Islam[1], Younghyun Cho[3], Xiaoye Sherry Li[2], Hengrui Luo[2], Yang Liu[2]
[1]Texas State University, [2]Lawrence Berkeley National Laboratory, [3]University of California, Berkeley

(*) Islam et al., "Toward a programmable analysis and visualization framework for interactive performance analytics," IEEE ProTools, 2019