Numerical Optimization for Automatic Tuning of Codes

Stefan Wild

Joint work with Prasanna Balaprakash, Paul Hovland, and Boyana Norris

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL

November 19, 2012

Motivation: A Looming Storm for Software & Libraries

Architectures are getting increasingly complex

- Multiple cores, deep memory hierarchies, software-controlled storage, shared resources, SIMD compute engines, heterogeneity, ...

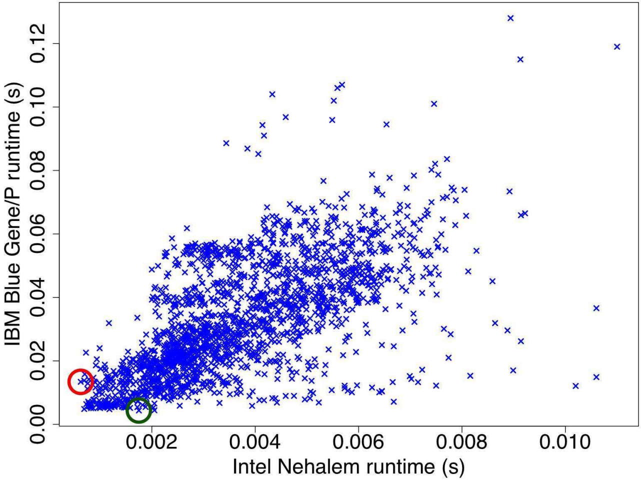
Performance optimization is getting more important

- Today's sequential and parallel applications may not be faster on tomorrow's architectures
- Growing complexity of scientific applications: tradeoffs between performance and maintainability (e.g., good software engineering practices)
- Managing data locality at least as important as optimizing parallelism
- Managing power of growing importance

Performance portability

- Tuning for a particular architecture potentially hinders performance on other architectures

Overtuning Can Destroy Performance Portability



Each × denotes a DGEMM variant

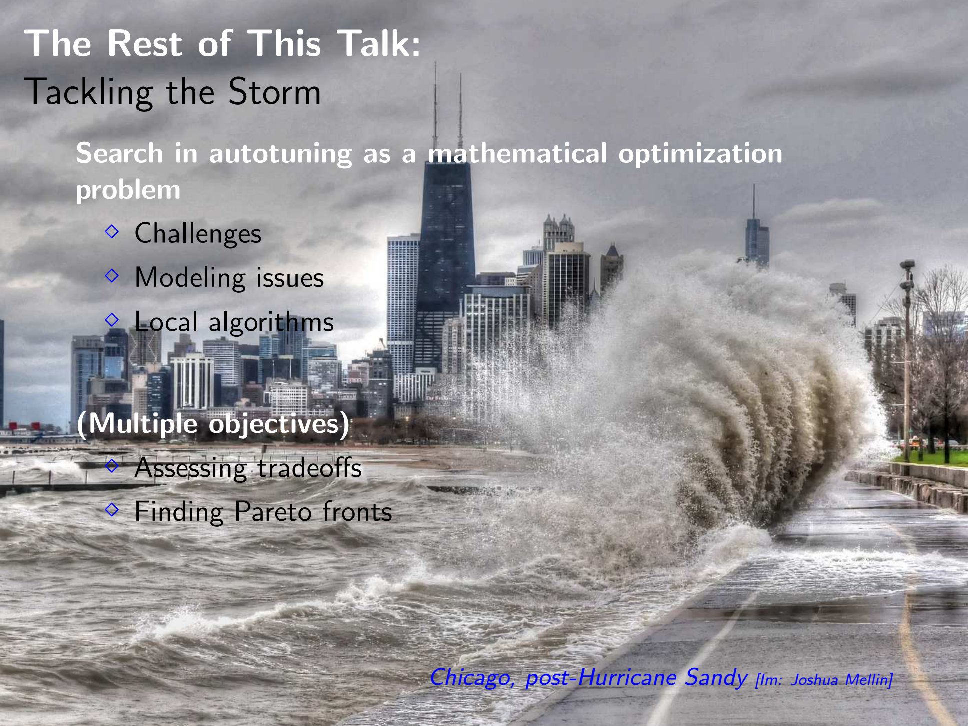
The Rest of This Talk:
Tackling the Storm

Search in autotuning as a mathematical optimization problem

- Challenges
- Modeling issues
- Local algorithms

(Multiple objectives)

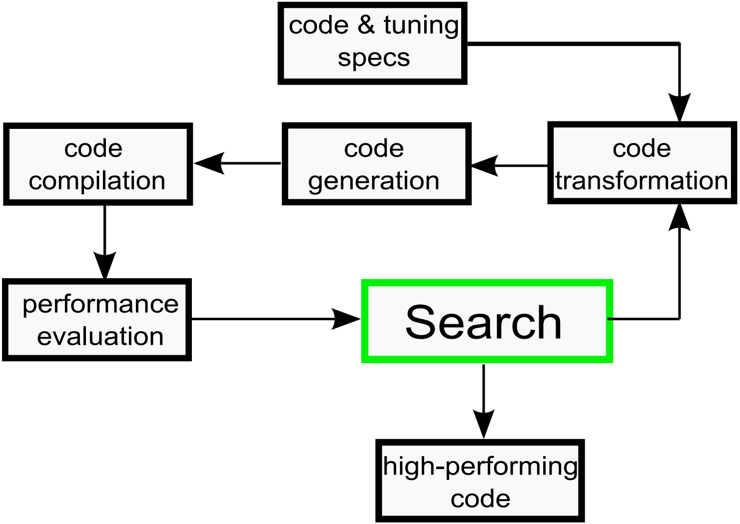
- Assessing tradeoffs
- Finding Pareto fronts



Chicago, post-Hurricane Sandy [Im: Joshua Mellin]

Automating Empirical Performance Tuning

Given a computation kernel and transformation space:



Search in Autotuning

Alternatives:

- Complete enumeration
  - Prohibitively expensive (10^{50} variants!)
  - Unnecessary?
- Pruning
  - Careful balancing act (between aggressive and conservative strategies)

Helpful (necessary?) precursors:

The expert still plays a role!

- Identify variable space (parameters to be tuned, ranges, constraints)
- Quantify measurement limitations and noise
- Incorporate known theoretical considerations (models)
- Construct meaningful objectives

→ Reduce search space and/or number of variants that need to be examined

Search in Autotuning

Alternatives:

- Complete enumeration
  - Prohibitively expensive (10^{50} variants!)
  - Unnecessary?
- Pruning
  - Careful balancing act (between aggressive and conservative strategies)

Helpful (necessary?) precursors:

The expert still plays a role!

- Identify variable space (parameters to be tuned, ranges, constraints)
- Quantify measurement limitations and noise
- Incorporate known theoretical considerations (models)
- Construct meaningful objectives

→ Reduce search space and/or number of variants that need to be examined

Search in Autotuning

Alternatives:

- Complete enumeration
  - Prohibitively expensive (10^{50} variants!)
  - Unnecessary?
- Pruning
  - Careful balancing act (between aggressive and conservative strategies)

Helpful (necessary?) precursors:

The expert still plays a role!

- Identify variable space (parameters to be tuned, ranges, constraints)
- Quantify measurement limitations and noise
- Incorporate known theoretical considerations (models)
- Construct meaningful objectives

→ Reduce search space and/or number of variants that need to be examined

Our goal

Design, implement, and analyze *efficient optimization (=search) algorithms*

. . . for tuning kernels in small computation budgets

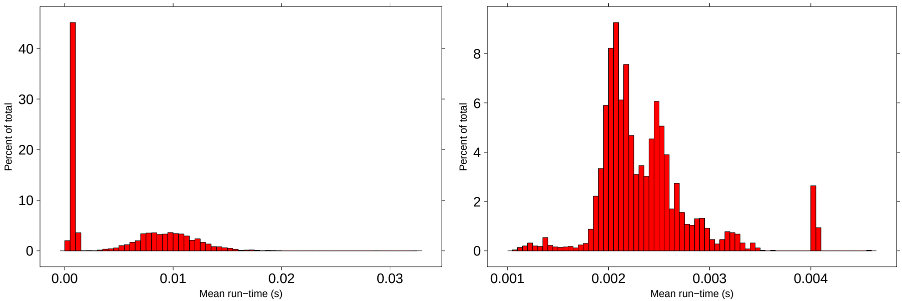
Is a Sophisticated Search Algorithm Needed?

[Seymour, You, & Dongarra, Cluster Computing '08]: Random search performs better than alternatives as the number of tuning parameters grows **!**

Is a Sophisticated Search Algorithm Needed?

[Seymour, You, & Dongarra, Cluster Computing '08]: Random search performs better than alternatives as the number of tuning parameters grows **!**

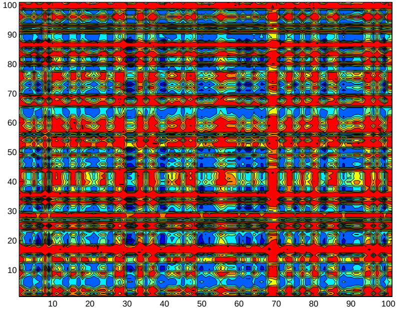
Depends on distribution of high-performing variants:

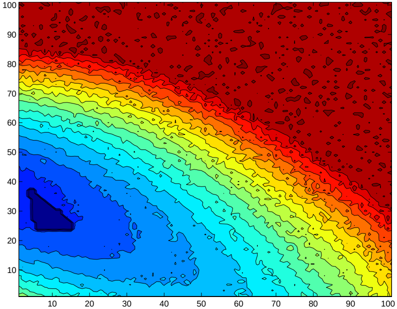


(5000 semantically equivalent variants each)

Is a Sophisticated Search Algorithm Useful?

Depends on structure of the (modeled) search space:





Both 2-dimensional problems have the same histogram

Must learn/model/exploit this structure to quickly find high-performing variants

Formulation and Modeling: Optimization is Optimization

Finding the best configuration is a mathematical optimization problem

$$\min_x \{f(x) : x = (x_\mathcal{I}, x_\mathcal{B}, x_\mathcal{C}) \in \mathcal{D} \subset \mathbb{R}^n\}$$

x multidimensional parameterization (compiler type, compiler flags, unroll/tiling factors, internal tolerances, ...) for a code variant

$f(x)$ empirical performance metric of x such as FLOPS, power, or run time (requires a run)

$\mathcal{D}$ search domain (constraints for feasible transformation, no errors, ...)

- bound: unroll $\in [1, \ldots, 30]$; RT $= 2^i$, i=[0,1,2,3]
- known: $(RT_I * RT_J \leq 150)$ (cheap); power consumption ≤ 90 W (expensive)
- hidden: transformation errors (relatively cheap), compilation (expensive), and run time (very expensive) failures

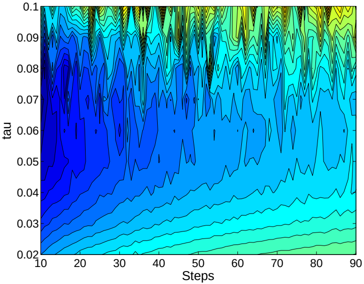
See [Balaprakash, Hovland, & W., iWAPT '11]

Optimization Challenges in Autotuning

$\min_x \{f(x) : x = (x_{\mathcal{I}}, x_{\mathcal{B}}, x_{\mathcal{C}}) \in \mathcal{D} \subset \mathbb{R}^n\}$

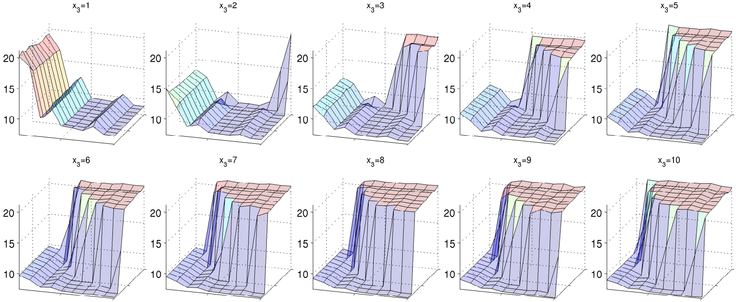
- f noisy, expensive, black box
- Discrete x unrelaxable
- $\nabla_x f$ unavailable/nonexistent
- "Cliffs", many distinct/local solutions?

Calls for **Derivative-Free Optimization**



↓ Integer Space: `MM` (MatMult)

↑ Mixed-Integer: Lattice QCD code



Search Problems in Automatic Performance Tuning

Problems: SPAPT [Balaprakash, Norris, & W., ICCS '12]

- 12 kernel codes
  - elementary linear algebra, linear solver, stencil codes, and elementary statistical computing configurations
- SPAPT problem = code + set of transformations + parameter specifications + constraints + input size

Code transformation: Orio performance tuning framework

Performance metric $f(x)$: mean run time of 10 runs

Kernel	Transformations	n_i	n_b	$\|\mathcal{D}\|$
ADI	CT, RT, UJ	16	4	2.818e+21
ATAX	CT, RT, UJ	13	6	6.115e+17
BiCG	CT, RT, UJ	9	4	2.654e+12
COR	CT, RT, UJ	16	4	2.818e+21
DGEMV	CT, RT, UJ	38	11	1.241e+53
FDTD4d2d	CT, RT, UJ	25	5	1.616e+33
GEMVER	CT, RT, UJ	17	6	1.409e+23
GESUMMV	CT, RT, UJ	8	3	5.308e+10
HMC	CT, RT, UJ	7	8	5.308e+10
Jacobi-1d	CT, RT, UJ	8	3	5.308e+10
LU	CT, RT, UJ	9	5	2.654e+12
MM	CT, RT, UJ	10	4	3.732e+11

SPAPT: Orio-ready Implementation

[Norris, Hartono, & Gropp, '07]

- Extensible empirical tuning system
- Allows inserting annotations as structured comments
- Supports architecture independent and specific optimizations

```
/* AXPY Kernel */
for (i=0; i<=n-1; i++)
   y[i]=y[i]+a1*x1[i]+a2*x2[i]+a3*x3[i]+a4*x4[i];
```

↓

/* Tuning specifications */ UF = {1,. . . ,30}; PAR = {True, False}

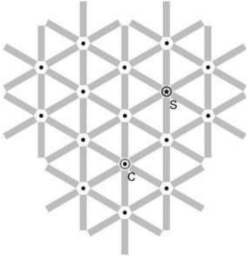
```
/*@ begin Loop (
  transform Unroll(ufactor=UF, parallelize=PAR)
    for (i=0; i<=n-1; i++)
       y[i]=y[i]+a1*x1[i]+a2*x2[i]+a3*x3[i]+a4*x4[i];
      )
@*/
```

Derivative-Free Optimization Algorithms

"Dibs:" An algorithm for Chicago winter parking reservations

Classical Algorithms for Performance Tuning

Global search



◇ exploration and exploitation
◇ find the globally best*
◇ long search time
◇ parameter sensitive

Local search

◇ limited exploration
◇ find the locally best
◇ short search time
◇ risk of bad local solution

Hypothesis: customized local search algorithms are effective for short computational budgets

Previous Algorithms for Performance Tuning

[Seymour, You, & Dongarra, Cluster Computing '08] and [Kisuki, Knijnenburg, & O'Boyle, PACT '00] compared several global and local algorithms

- Random search outperforms a genetic algorithm, simulated annealing, particle swarm, Nelder-Mead, and orthogonal search !
- Large number of high-performing parameter configurations → easy to find one of them

[Norris, Hartono, & Gropp, *Computational Science* '07] used several global and local algorithms but no comparison

- Nelder-Mead simplex method, simulated annealing, a genetic algorithm

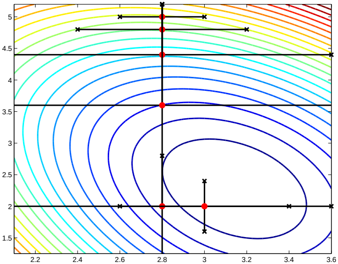
Other local search algorithms without comparison to global search:

- Orthogonal search in ATLAS [Whaley & Dongarra, SC '98]
- Pattern search in loop optimization [Qasem, Kennedy, & Mellor-Crummey SC '06]
- Modified Nelder-Mead simplex algorithm in Active Harmony [Tiwari, Chen, Chame, Hall, & Hollingsworth, IPDPS '09]

Local Algorithms: Direct Search Methods

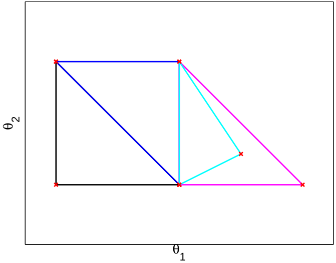
See [Kolda, Lewis, & Torczon, *SIREV* '03]

Pattern Search



Easy to parallelize f evaluations

Nelder-Mead

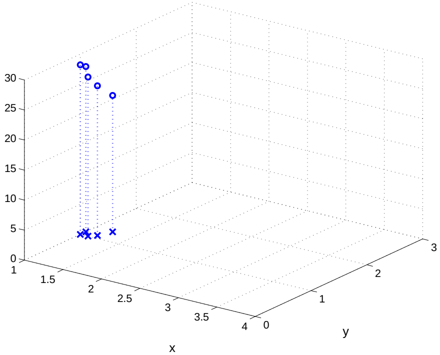


Popularized by *Numerical Recipes*

- Rely on indicator functions: $[f(x_k + s) <^? f(x_k)]$
  - Ignore valuable information on relative magnitudes of $f(x_k)$

Making the Most of Little Information on f

- f is expensive $\Rightarrow$ can afford to make better use of points
- Overhead of the optimization routine is minimal (negligible?) relative to cost of empirical evaluation



Bank of data, $\{x_i, f(x_i)\}_{i=1}^{k}$:

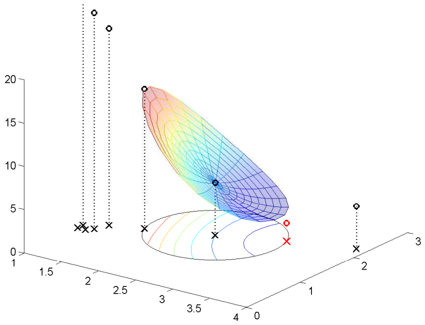
$=$ Everything* known about f

Idea:

- Make use of growing bank as optimization progresses

Making the Most of Little Information on f

- f is expensive $\Rightarrow$ can afford to make better use of points
- Overhead of the optimization routine is minimal (negligible?) relative to cost of empirical evaluation



Bank of data, $\{x_i, f(x_i)\}_{i=1}^k$:

= Everything* known about f

Idea:

- Make use of growing bank as optimization progresses

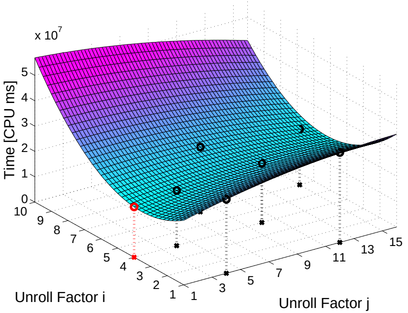
Surrogate-Based Trust-Region Algorithms

Substitute $\min \{m(x) : x \in \mathcal{B}_k\}$ for $\min f(x)$

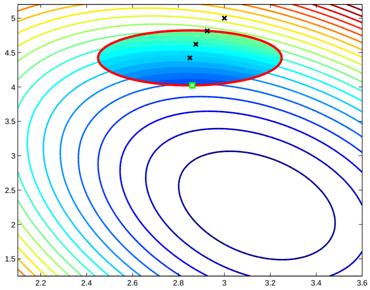
f expensive, no ∇f

m cheap, analytic derivatives

Surrogate based on known f values

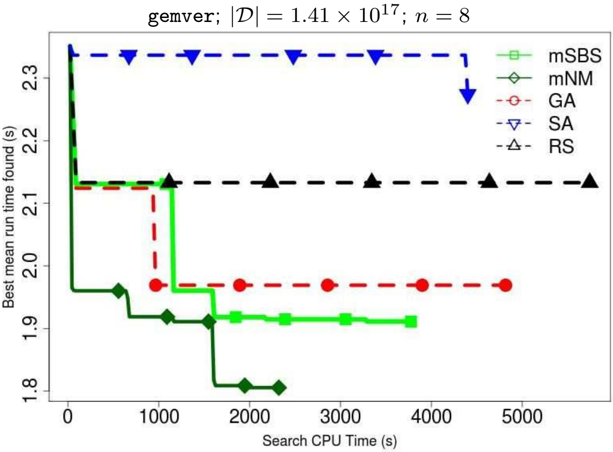


Trust $m \approx f$ in $\mathcal{B}_k$



Surrogates: predict improvement

Sample Comparison: Unoptimized Default Starting Point

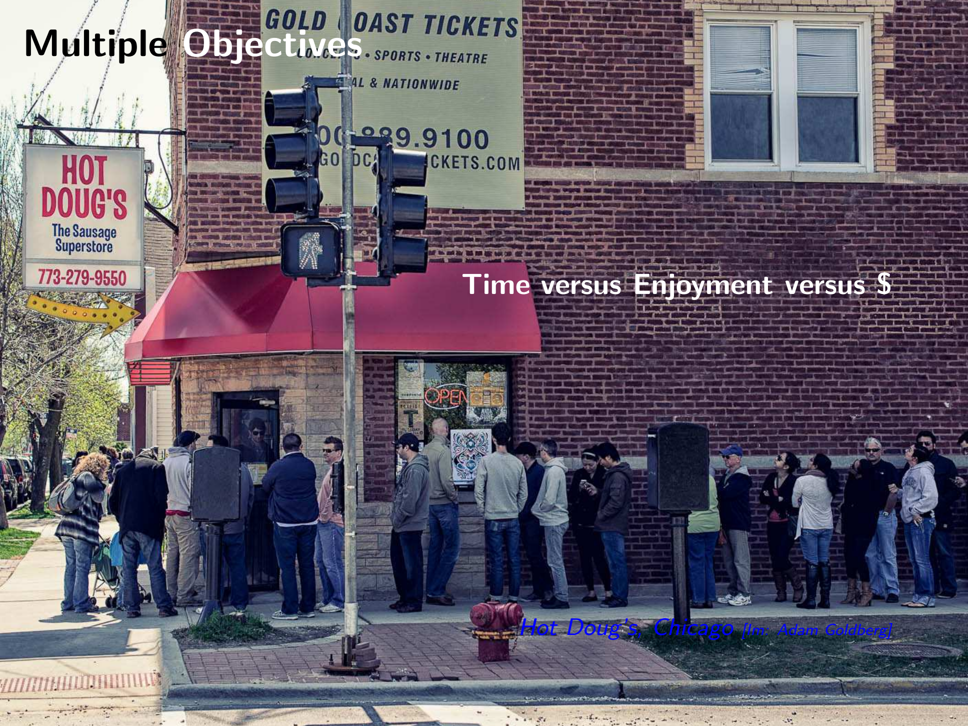


- **Double win: Better solutions, less time** (=time/evaluation)
- 10/12 SPAPT problems local search outperforms global search

(Down and to the left is better; Markers every 20 evaluations)

See [Balaprakash, Hovland, & W., VecPar '12]

Multiple Objectives



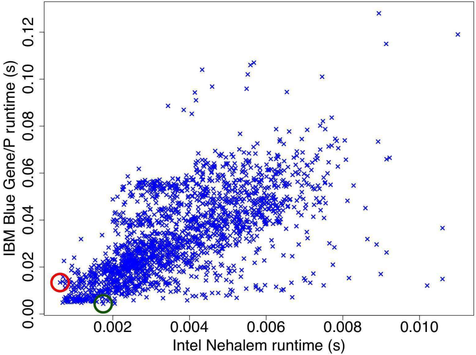
Time versus Enjoyment versus $

Hot Doug's, Chicago [Im: Adam Goldberg]

Simultaneously Optimizing Multiple Objectives

$$\min_{x\in\mathcal{D}}\{f_1(x), f_2(x), \ldots, f_p(x)\}$$

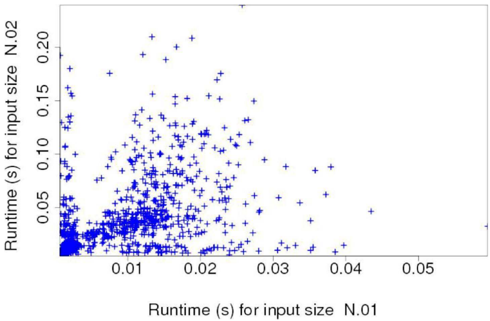
- No *a priori* weights w_i $\left(\sum_i w_i f_i(x)\right)$
- Dominated points $\tilde{x}$: $\exists x^* \in \mathcal{D}$ with
  $f_i(\tilde{x}) \geq f_i(x^*)\ \forall i,$
  $f_j(\tilde{x}) > f_j(x^*)$ some j
- Seek Pareto front of non-dominated points



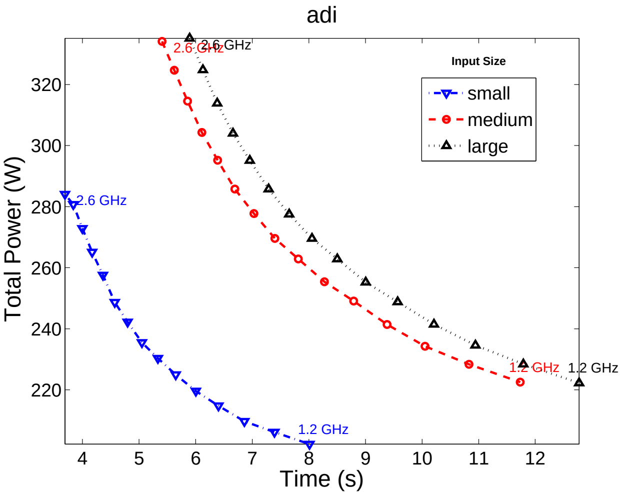
Simultaneously Optimizing Multiple Objectives

$$\min_{x \in \mathcal{D}} \{ f_1(x), f_2(x), \ldots, f_p(x) \}$$

- No *a priori* weights w_i $\left(\sum_i w_i f_i(x)\right)$
- Dominated points $\tilde{x}$: $\exists x^* \in \mathcal{D}$ with
  $f_i(\tilde{x}) \geq f_i(x^*) \, \forall i$,
  $f_j(\tilde{x}) > f_j(x^*)$ some j
- Seek Pareto front of non-dominated points

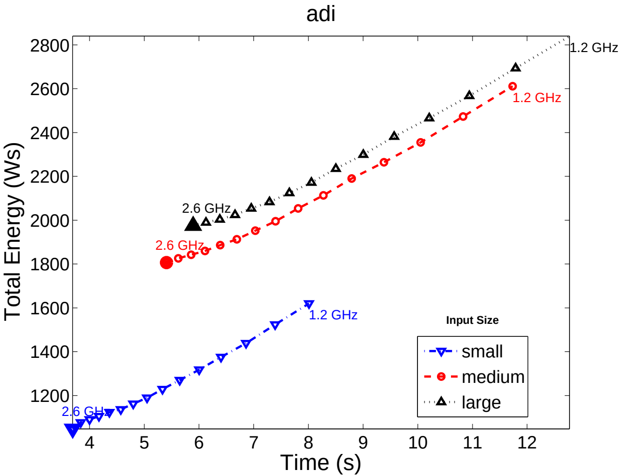


Multiple Objectives: Time, Power, Energy



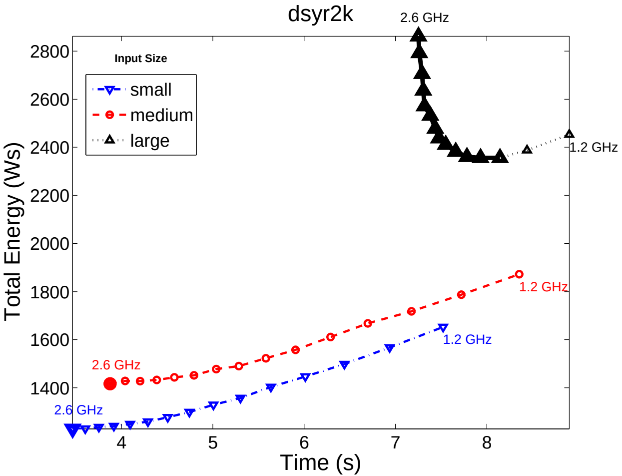
◇ Tradeoffs in power do not imply tradeoffs in energy

Multiple Objectives: Time, Power, Energy



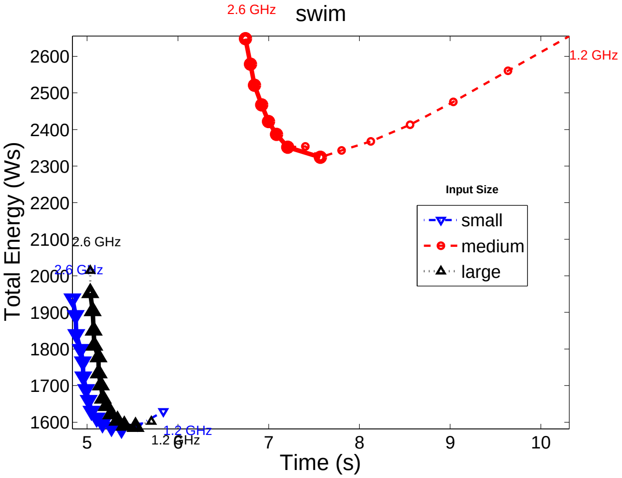
- Tradeoffs in power do not imply tradeoffs in energy
- Objectives may not be conflicting: "Race to idle

Multiple Objectives: Time, Power, Energy



- Tradeoffs in power do not imply tradeoffs in energy
- Objectives may not be conflicting: “Race to idle
- Tradeoffs occur for different sizes

Multiple Objectives: Time, Power, Energy



- Tradeoffs in power do not imply tradeoffs in energy
- Objectives may not be conflicting: "Race to idle
- Tradeoffs occur for different sizes
- Tradeoffs occur at different frequencies

Summary and Links

- Performance tuning increasingly necessary, not yet "automatic"
- Derivative-free optimization is a powerful, practical tool

When the available tuning time is limited:

- Global exploration less useful
- Problem formulation and starting point play important roles

Future work includes:

- Incorporation of models, binary parameters, constraints (from models or otherwise), online restart strategies,role in full application codes, . . .

→ always collecting new search/optimization problems

. . . especially those with structure

Some preprints `http://mcs.anl.gov/~wild`

Orio `http://trac.mcs.anl.gov/projects/performance/wiki/Orio`

SPAPT `http://trac ··· /performance/browser/orio/testsuite/SPAPT.v.01`

Summary and Links

- Performance tuning increasingly necessary, not yet "automatic"
- Derivative-free optimization is a powerful, practical tool

When the available tuning time is limited:

- Global exploration less useful
- Problem formulation and starting point play important roles

Future work includes:

- Incorporation of models, binary parameters, constraints (from models or otherwise), online restart strategies,role in full application codes, . . .

→ always collecting new search/optimization problems

. . . especially those with structure

Some preprints `http://mcs.anl.gov/~wild`

Orio `http://trac.mcs.anl.gov/projects/performance/wiki/Orio`

SPAPT `http://trac ··· /performance/browser/orio/testsuite/SPAPT.v.01`

→ Thank you!