Automated Performance Analysis with PETSc

Matthew G. Knepley
Computer Science & Engineering, University at Buffalo
knepley@gmail.edu

Albert Cowie
Computer Science & Engineering, University at Buffalo
arcowie@buffalo.edu

Justin Chang
National Renewable Energy Laboratory
justin.chang@nrel.gov

We acknowledge support from DOE Applied Math Research, Contract DE-AC02-06CH11357, NSF SI2-SSI: 1450339, and the UB Center for Computational Research.

Why do we need new performance analysis?

Traditional performance analysis treats all work (flops, memory references, etc.) as *equally effective*. Hpwever, work done in service of an algorithm with faster convergence rate is worth more than normal work, if we use the metric of marginal decrease in error per flop. Thus flops done for multigrid are different than those done for CG/ILU, and flops done for Newton are different than those done for Richardson.

Metrics such as flop rate and bandwidth achieved are excellent for understanding hardware utilization, which accounts for the popularity of Roofline analysis, tools such as STREAM, PAPI, OpenSpeedShop and HPCToolKit. However they do not shed much light on algorithmic tradeoffs For that, we also need to understand how different algorithms make progress toward the solution.

Automation

PETSc provides an *extensible* logging framework, where users can add custom events and choose arbitrary aggregation stages for results. We have added the ability to report both problem sizes and solution errors for multiple fields inside any logging event. This means the structure is also extensible to different error measures, such as divergence norms, dispersion measures, and hypernorms. We plan to explore the consequences for algorithm choices in future work.

PETSc API (www.mcs.anl.gov/petsc)

In order to input data, we have added:

```
PetscErrorCode PetscLogEventSetDof(PetscLogEvent event, PetscInt n, PetscLogDouble dof);
PetscErrorCode PetscLogEventSetError(PetscLogEvent event, PetscInt n, PetscLogDouble error);
```

The user may output the performance data using options:

`-log_view :log.out`	Human-readable ASCII
`-log_view :log.out:ascii_info_detail`	Python module source
`-log_view :log.out:ascii_csv`	CSV file

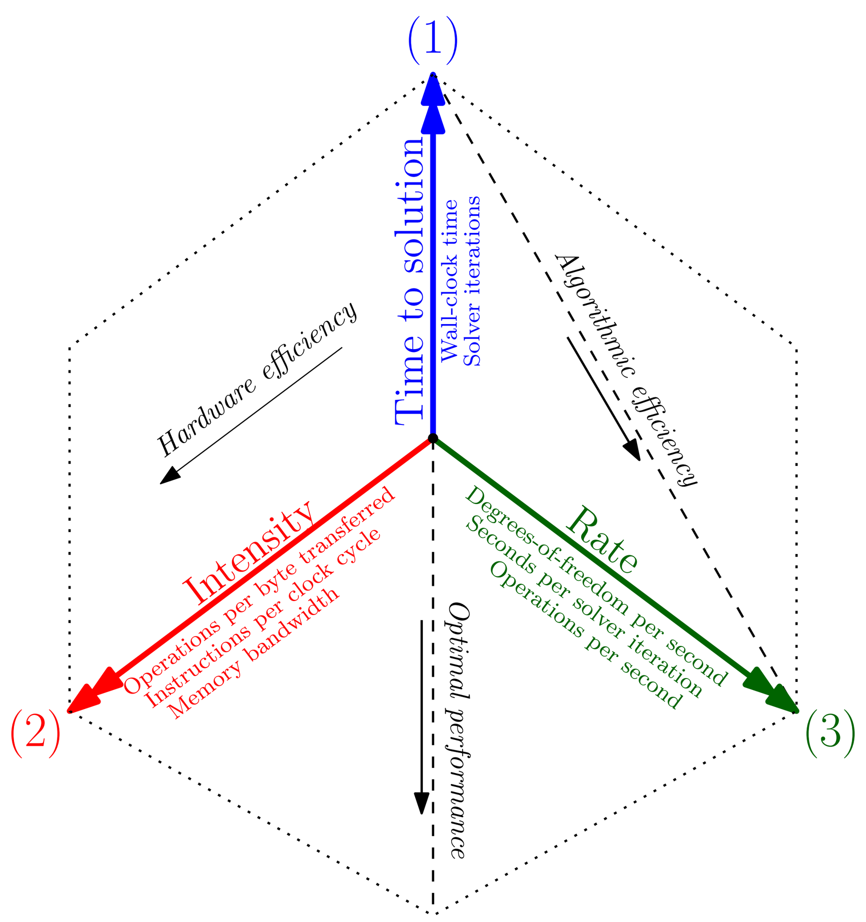
Metrics

Primary Data		Derived Quantities	
d	Spatial dimension	Resolution h	$N = Dh^{-d}$
T	Runtime	Digits of Accuracy (DoA)	$-\log_{10} E$
N	Problem size	Digits of Size (DoS)	$\log_{10} N$
E	Solution error	Digits of Efficacy (DoE)	$-\log_{10}(ET)$

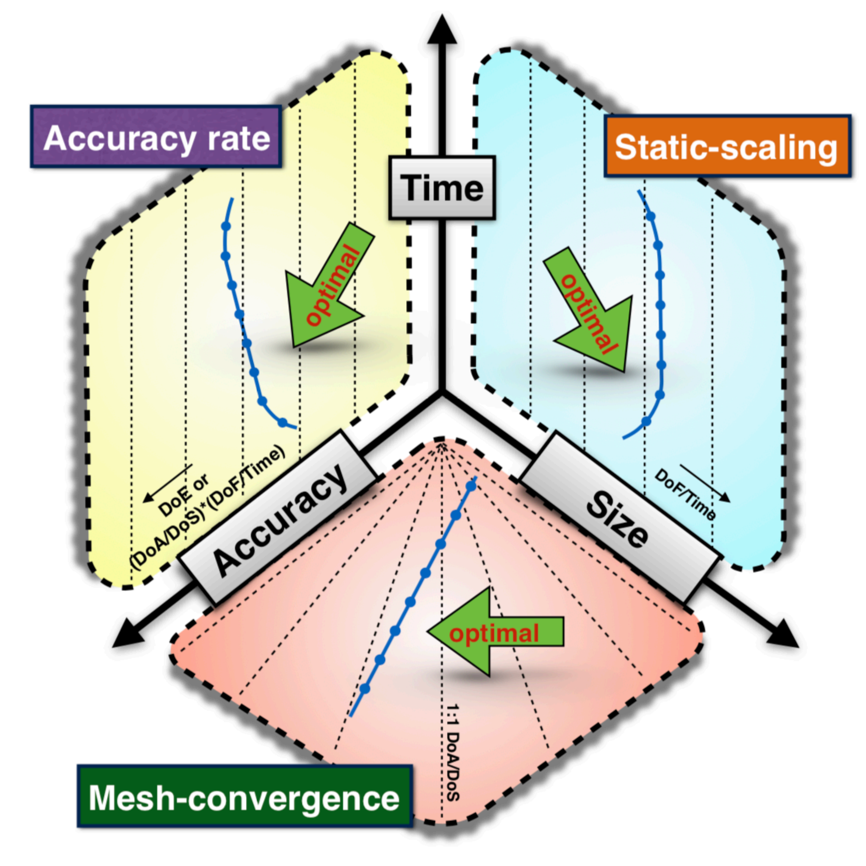
Strong Scaling: Problem Size Fixed Concurrency Increased
Weak Scaling: Problem Size Increased Concurrency Increased
Static Scaling: Problem Size Increased Concurrency Fixed

What is TAS analysis?

The Time-Accuracy-Size (TAS) performance analysis uses measures of the execution time, solution accuracy, and problem size or work, to examine algorithmic tradeoffs. It is detalied in (Chang, Fabien, Knepley, Mills, 2018. arxiv:1802.07832). For example, the traditional mesh convergence diagram plots accuracy against size. The static scaling analysis introduced by Brown and Adams for the HPGMG Benchmark (hpgmg.org) plot computation rate against runtime for fixed parallelism. We introduce a new analysis comparing accuracy-time against runtime, again at fixed parallelism. The diagram below shows the relation between these analyses.



Chang et. al., *A performance spectrum for parallel computational frameworks that solve PDEs*, Concurrency and Computation, 30(11), 2017.



Chang et. al., *Comparative study of finite element methods using the Time-Accuracy-Size (TAS) spectrum analysis*, SIAM J. Sci. Comput., 40(6), 2018.

Optimal Solvers

A convergent discretization satisfies $E \leq Ch^{-\alpha}$, thus our mesh convergence slope is

$$\frac{DoA}{DoS} = -\frac{\log_{10}\left(Ch^{\alpha}\right)}{\log_{10}\left(Dh^{-d}\right)} \approx \frac{\alpha}{d}$$

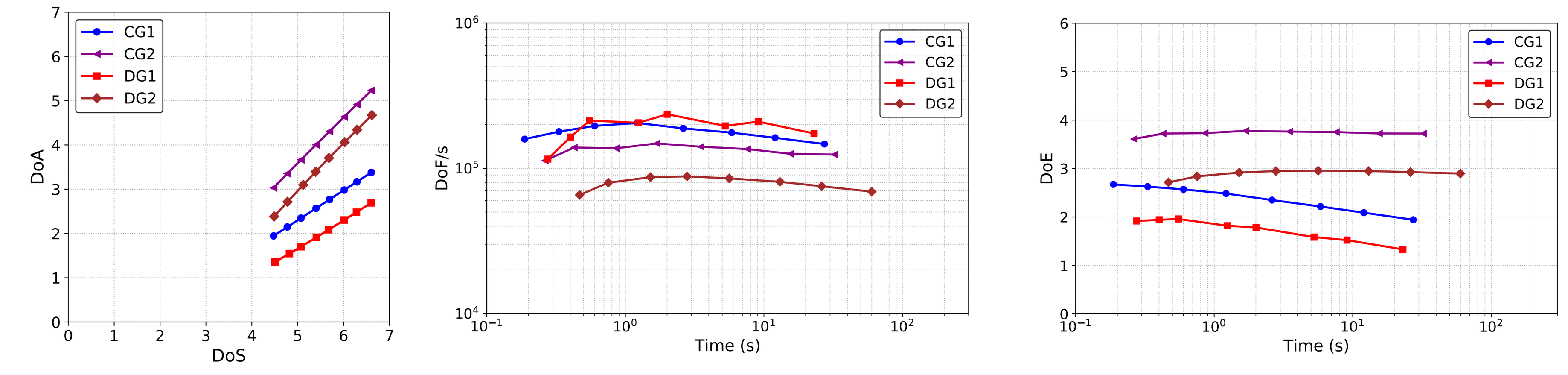
Optimal solvers do linear work, so the static scaling curve should be flat, until strong scaling effects take over at the left, or algorithmic inefficiencies dominate at the right.

The marginal production of accuracy declines as problem sizes increases, which makes it a poor measure for performance across problem scales. Instead, we define the *efficacy*, or $E \cdot T$, as the quantity to be minimized by the solver. For an optimal solver,

$$-\log_{10}(E \cdot T) = -\log_{10}\left(Ch^{\alpha} \cdot Wh^{-d}\right) \qquad = (d-\alpha)\log_{10} h - \log_{10}(CW).$$

so that the optimal slope of our efficacy graph is $d - \alpha$. Note also that higher DoE indicates a more efficient algorithm. Thus efficacy can be likened to an *algorithmic action*, in analogy with the mechanical action $\mathrm{Energy} \times \mathrm{Time}$, and minimzation of the efficacy should be an algorithmic design goal.

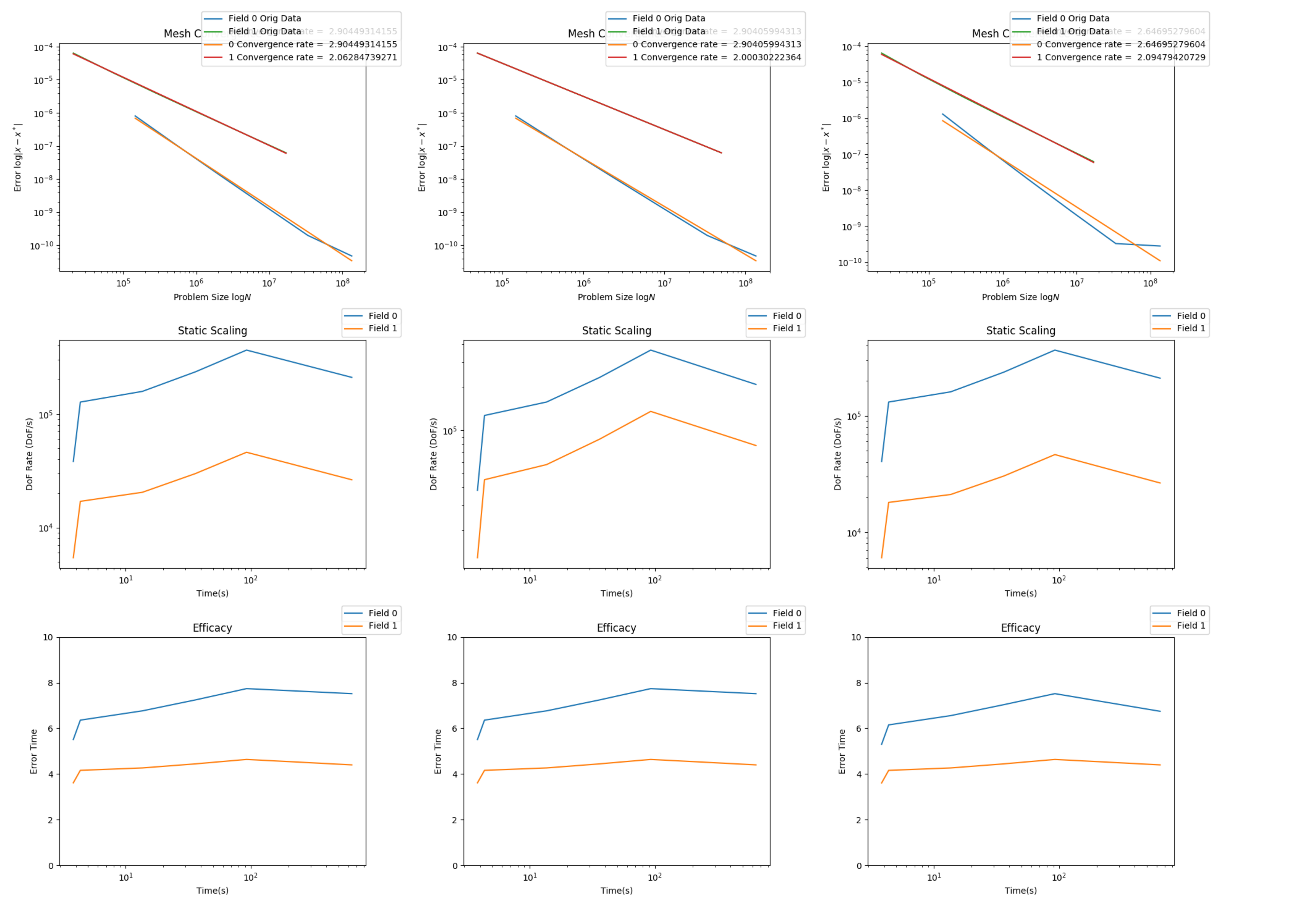
3D Poisson Problem: CG vs. DG, 1 PE



Mesh Convergence
Static Scaling
Efficacy

2D Stokes Problem: Comparing $Q_2 - Q_1$ vs. $Q_2 - Q_{-1}$ vs $P_2 - P_1$

SNES ex62 run on 1152 PE of the Knepley/Jadamec Geosolver Cluster at UB CCR. Both velocity and pressure fields are shown on graphs.



Summary

- TAS Analysis can evaluate algorithmic tradeoffs more precisely (arxiv:1802.07832)
- PETSc now (extensibly) automates the information gathering for TAS
- The next release of PETSc will automate vizualization of TAS
- See also poster on *Composable Block Solvers*, Joshaghani et.al. (arxiv:1808.08328) for a TAS analysis of multilevel block preconditioners.