BLUE WATERS
SUSTAINED PETASCALE COMPUTING

Performance Modeling for Systematic Performance Tuning

Torsten Hoefler
with inputs from William Gropp, Marc Snir, Bill Kramer

Invited Talk RWTH Aachen University
March 30th, Aachen, Germany

NCSA NSF GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

All used images belong to the owner/creator!

The Perspective of a Computing Center

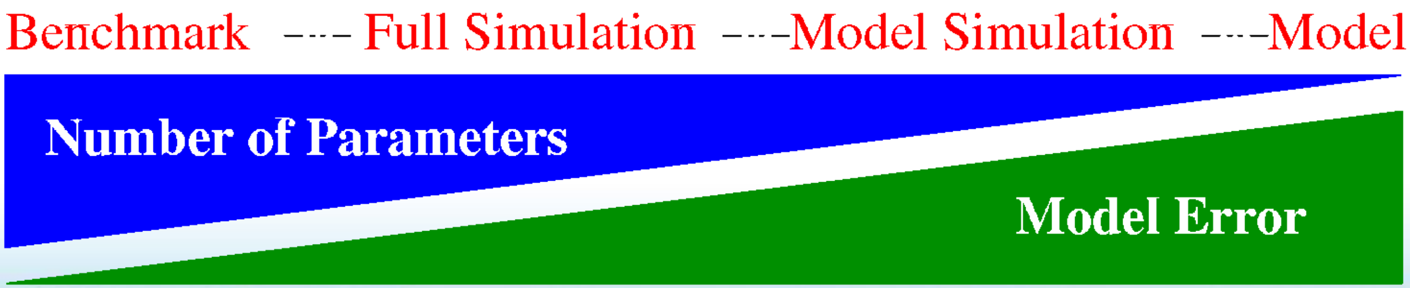
- Performance = “completed science per cost and time”
- Optimizing this metric can be manifold:
  - Application optimization (support application teams)
  - Architecture optimization (select best hardware)
  - Optimize Middleware (scheduler, libraries etc.)
  - Optimize Policies (scheduling, charging etc.)
  - … and many more

Performance Modeling – State of the Practice

- Delivers the “science per cost/time” metric
  - Can be used to drive optimizations!
- Who does performance modeling?
  - Mostly computer scientists, in-house teams
- BUT: most development is done by application developers and/or domain scientists
  - They should develop performance models during software development
    - See performance modeling panel @3:30 in TCC 101

(Ideal) State of the Practice @NCSA

- Propose to use simple performance modeling to characterize the behavior of applications
  - Enables rough optimization (cf. “80/20 rule”)
- We provide a set of simple modeling guidelines
  - Semi-analytic performance modeling
  - Small number of parameters, use other techniques where necessary



Benchmark ---- Full Simulation ----Model Simulation ----Model

Number of Parameters

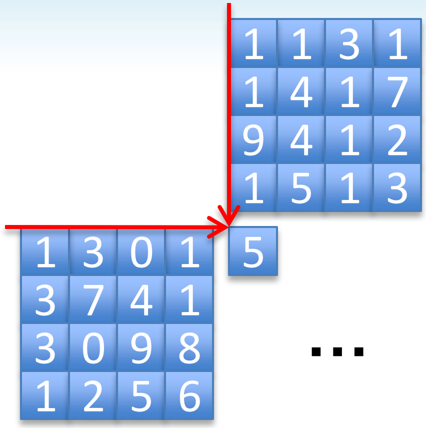
Model Error

Overview of Performance Modeling

- Analytic modeling:
  - Determine application requirements and system speeds to compute time (e.g., bandwidth)
- Empirical modeling (e.g. [1,2]):
  - “Black-box” approach: machine learning, neural networks, statistical learning …
- Semi-empirical modeling:
  - “White box” approach: find asymptotically tight analytic models, parameterize empirically (curve fitting)

[1]: Barnes, Rountree, Lowenthal, Reeves, Supinski, Schulz: A regression-based approach to scalability prediction
[2]: McKee, Singh, Supinski, Schulz: Constructing Application Performance Models Using Neural Networks

A Quick Example - MM



1	1	3	1
1	4	1	7
9	4	1	2
1	5	1	3

1	3	0	1
3	7	4	1
3	0	9	8
1	2	5	6

- Matrix multiplication (N^3 algorithm)

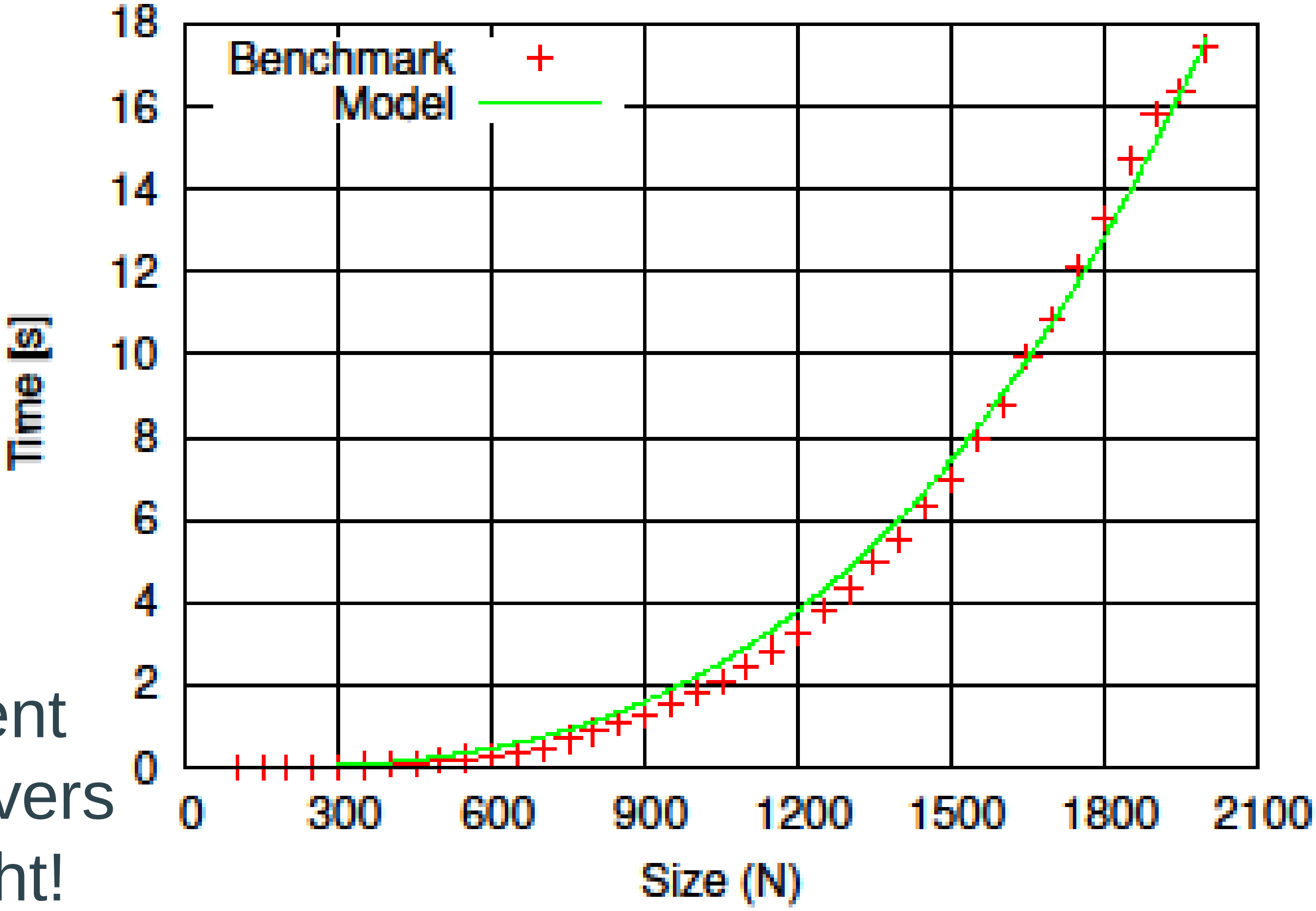
```
for(int i=0; i<N; ++i)
  for(int j=0; j<N; ++j)
    for(int k=0; k<N; ++k)
      C[i+j*N] += A[i+k*N] * B[k+j*N];
```

- Trivial (non-blocked) algorithm

- Analytic Model:
  - N^3 FP add/mult, $4N^3$ FP load/store, +int ops
  - How can we get to an execution time? → very hard!

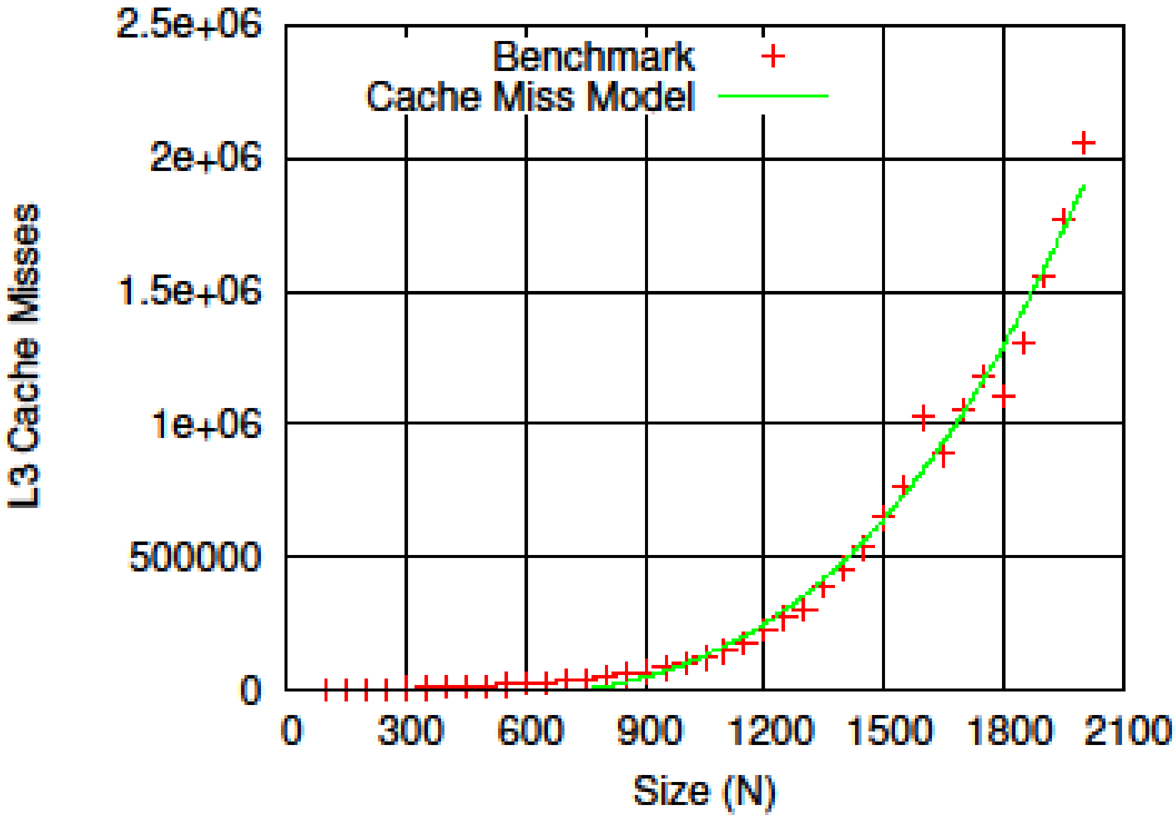
Semi-Empiric Model for MM

- $T(N) = tN^3$
- POWER7
  - t=2.2ns
  - 0.8% err
- Is that all?
  - Requirement Model delivers more insight!



Requirements Model for MM

- Required floating point operations: $2N^3$ (verified)
- Cache misses?
  - Semi-analytic!
  - $C(N) = aN^3 - bN^2$
- POWER7
  - a=3.8e-4
  - a=2.7e-1



Our Ubiquitous Modeling Philosophy

- Modeling during each phase of SW development:
  - Analysis – pick right method (asymptotic models)
  - Design – pick right algorithms (asymptotic models)
  - Implementation – show good usage of machine, e.g., blocking in MM (semi-empirical models)
  - Testing – fulfilling model expectations as correctness criterion (compare tests with models)
  - Maintenance – monitor performance on different architectures (compare times with models)

More uses of Models

- Performance Optimization
  - Identify bottlenecks and problems during porting
- System Design
  - Co-design based on application requirements
- System Deployment and Testing
  - Know what to expect, find performance issues quickly
- During System Operation
  - Detect silent (and slow) performance degradation

Six-Steps to a Model

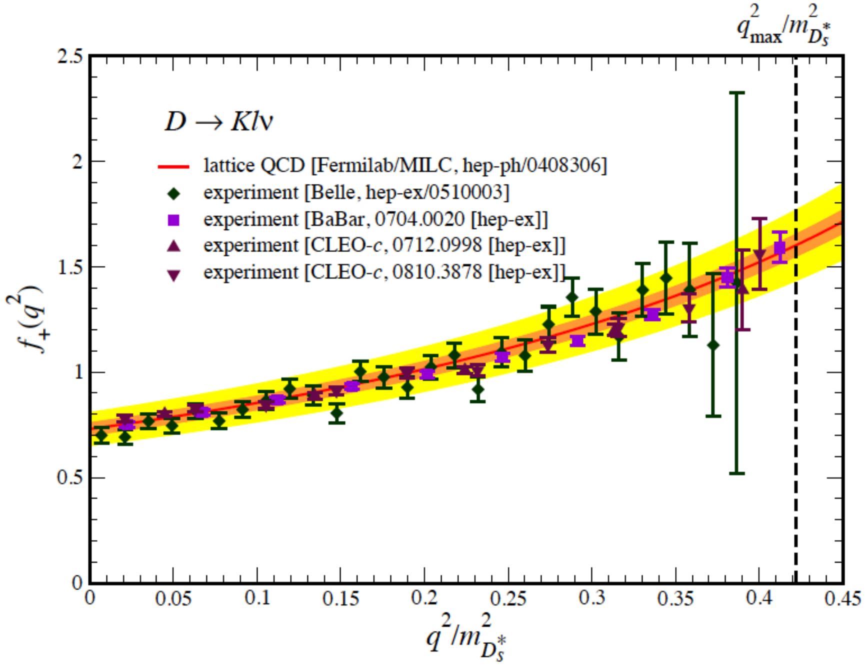
- Our <u>very</u> high-level strategy consists of the following six steps:
  1) Identify input parameters that influence runtime
  2) Identify application kernels
  3) Determine communication pattern
  4) Determine communication/computation overlap

Analytic

  5) Determine sequential baseline
  6) Determine communication parameters

Empiric

All Steps By Example – MILC



- MIMD Lattice Computation
  - Gains deeper insights in fundamental laws of physics
  - Determine the predictions of lattice field theories (QCD & Beyond Standard Model)
  - Major NSF application
- Challenge:
  - High accuracy (computationally intensive) required for comparison with results from experimental programs in high energy & nuclear physics

Bernard, Gottlieb et al.: Studying Quarks and Gluons On Mimd Parallel Computers

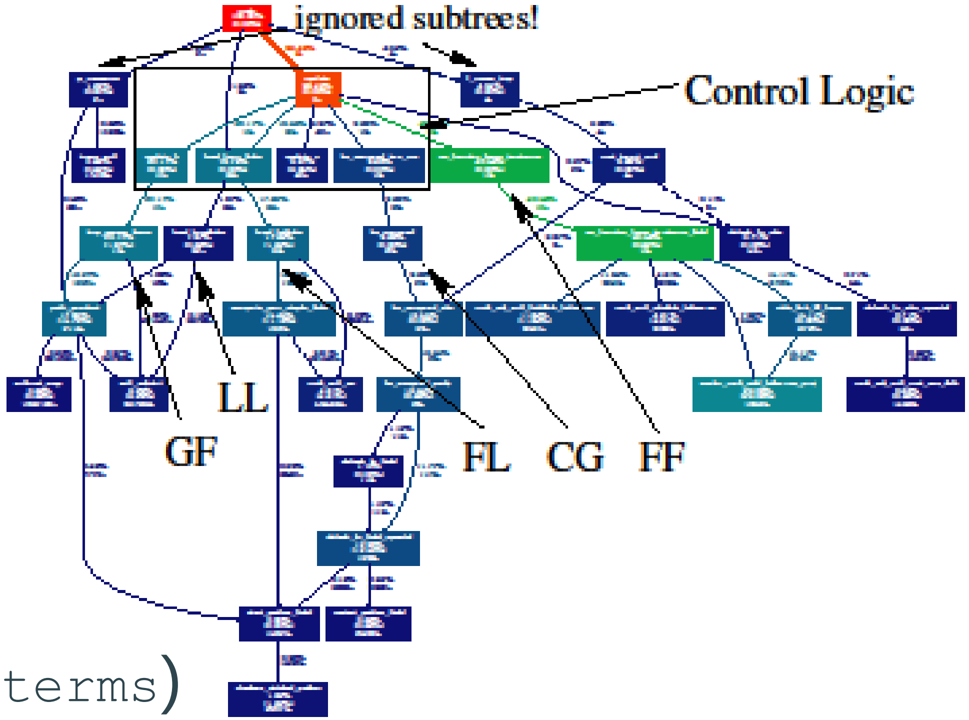
Step 1: Critical Parameters

- Best way: ask a domain expert!
  - Or: look through the code/input file format
- For MILC (thanks to S. Gottlieb):

Name	**Description**
P	number of PEs (intrinsic parameter)
nx, ny, nz, nt	size in x, y, z, t dimension
warms, trajecs	warmup rounds and trajectories (outer loop)
traj_between_meas	measurement "frequency"
steps_per_trajectory	number of "steps" in each trajectory
beta, mass1, ...	physics parameters that influence CG iterations
max_cg_iterations	limits the conjugate gradient iterations

Step 2: Find Kernels

- E.g., investigate call-tree or source-code
- Control logic
  - `update`
- MILC's kernels:
  - LL (`load_longlinks`)
  - FL (`load_fatlinks`)
  - CG (`ks_congrad`)
  - GF (`imp_gauge_force`)
  - FF (`eo_fermion_force_twoterms`)

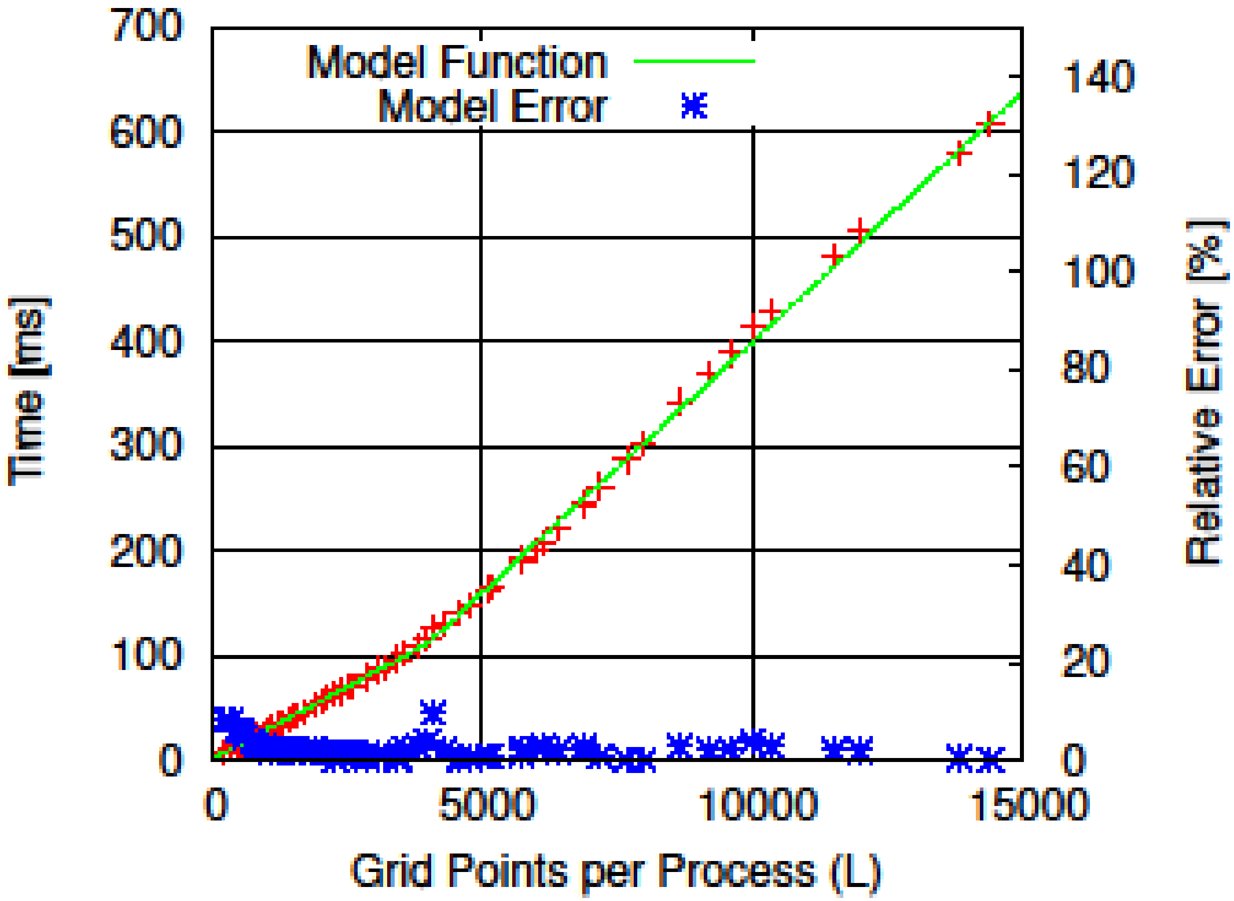


Step 4: Sequential Performance

- MILC “only” loops over the lattice → $\Theta(V)$
- ➢ $T(V) = tV$
  - Wait, it’s not that simple with caches ☹
  - Small V fit in cache!
- ➢ $T(V) = t_1 * \min(s, V) + t_2 * \max(0, V-s)$
  - Cache holds s data elements
  - Three parameters for each kernel

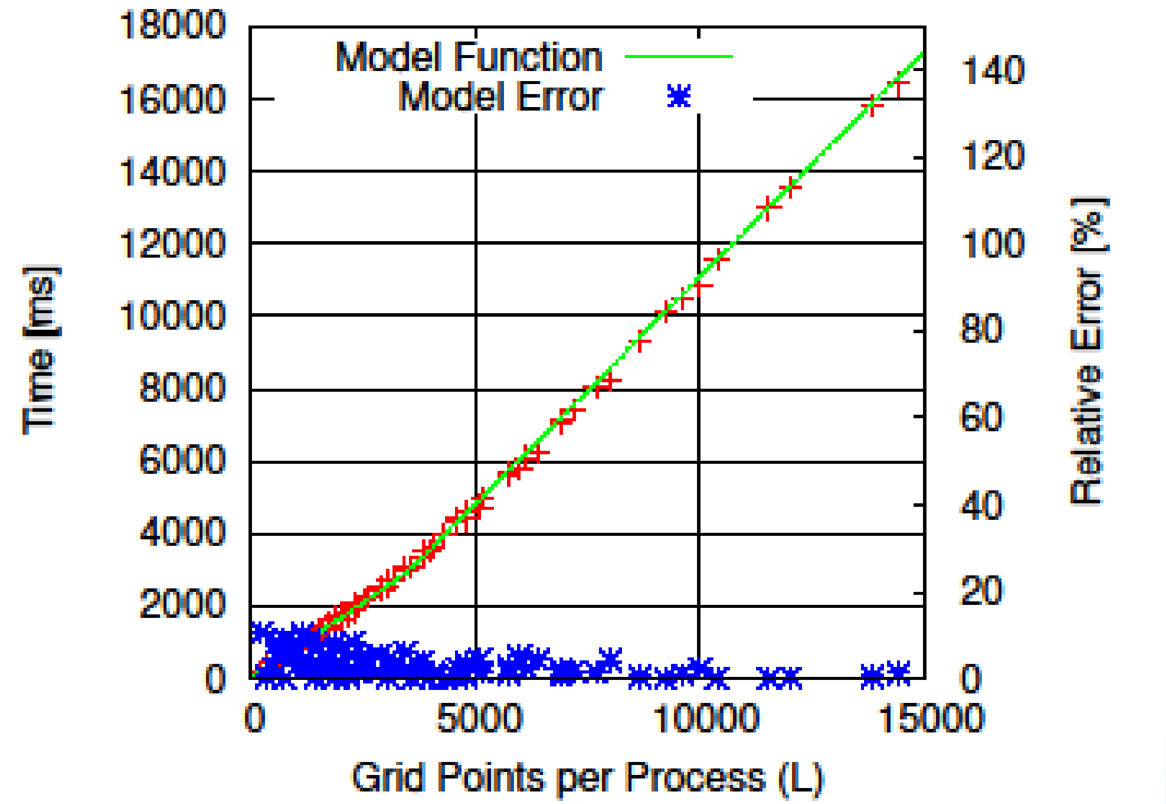
An Example Kernel: GF (Gauge Force)

- On POWER7:
  - t_1=62.4 µs
  - t_2=92 µs
  - s=4.000
- Errors
  - Max <10%
  - Cum <3%



Complete Serial Performance Model

$$T_{serial}(V) = (\mathtt{trajecs} + \mathtt{warms}) \cdot \mathtt{steps} \cdot [T(FF,V) + T(GF,V) + 3(T(LL,V) + T(FL,V))] + \left\lfloor \frac{\mathtt{trajecs}}{\mathtt{meas}} \right\rfloor [T(LL,V) + T(FL,V)] + \mathtt{niters} \cdot T(CG,V)$$



Step 3: Communication Pattern

- 4d domain is cut in all dimensions (cubic)
  - 4d nearest-neighbor communication (8 neighbors)
- Allreduce to check CG convergence
  - One per iteration on full process set
- We counted messages and sizes
  - Separate for each kernel
  - See paper for sizes and full model equation!

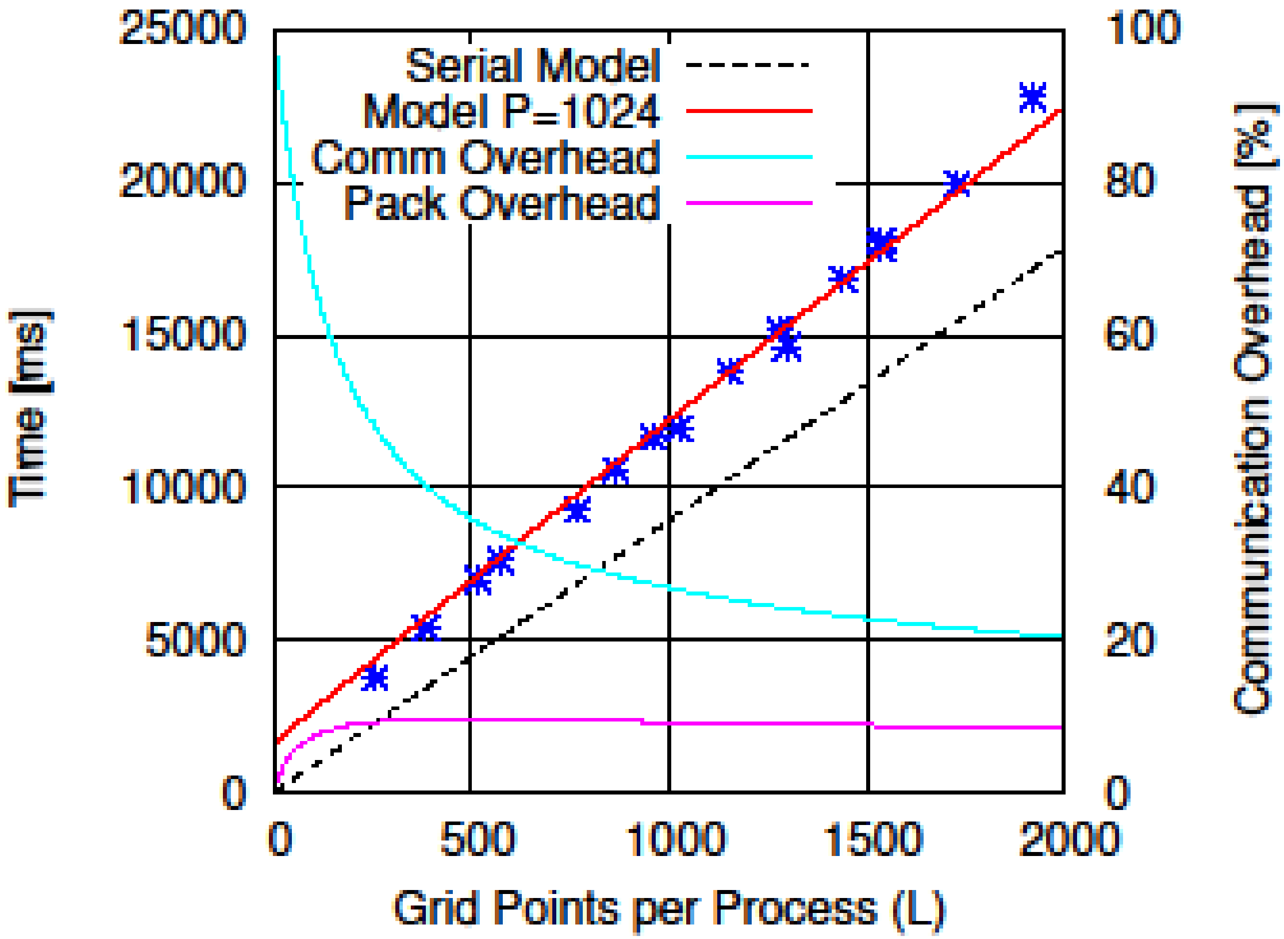
kernel	$\#Messages$
FF	$(\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} \cdot 1616$
GF	$(\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} \cdot 828$
LL	$(3 \cdot \texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + \lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \rfloor) \cdot 8$
FL	$(3 \cdot \texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + \lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \rfloor) \cdot 288$

Step 6: Communication Parameters

- Two options:
  - Semi-empiric – fit measurements to get effective latency and bandwidth
    - Enables to check if they match expectations
  - Analytic – derive parameters separately (e.g., documentation or separate benchmark)
    - Often problematic if they do not match expectations
- Our model was analytic
  - Uses LogGP parameters (measured by Netgauge [1])

[1] Hoefler et al.: Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks

The Fully-Parameterized Parallel Model



Conclusions and Future Work

- Models in use for predictions and optimizations
  - First successes: ~10-20% improved performance [1]
- Simple strategy enables application team models
  - Better chance to be maintained than external models
  - Critical for performance-centric software development
- We need (and work on):
  - More examples for irregular/dynamic codes
  - Better tool support for modeling

[1] Hoefler, Gottlieb.: Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient using MPI Datatypes