BLUE WATERS

SUSTAINED PETASCALE COMPUTING

# Analyses and Modeling of Applications Used to Demonstrate Sustained Petascale Performance on Blue Waters


Torsten Hoefler

With lots of help from the AUS Team and Bill Kramer at NCSA!

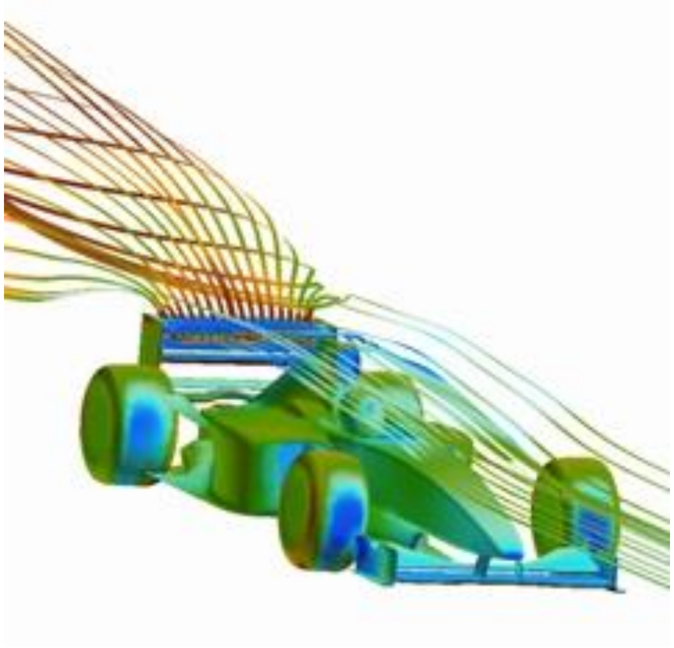

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

# The State of Performance Measurements

- Most used metric: Floating Point Performance
  - That's what limited performance in the 80's!
  - Systems were balanced, peak was easy!
  - FP performance was **the** limiting factor
- Architecture Update (2012):
  - Deep memory hierarchies make systems highly unbalanced
  - Caches mitigate the effect by exploiting algorithmic structure and data locality

# Rough Computational Algorithm Classification

- High locality, moderate locality, low locality
- Highly Structured
  - Dense linear algebra
  - FFT
  - Stencil
- Semi-structured
  - Adaptive refinements
  - Sparse linear algebra
- Unstructured
  - Graph computations



# How do we assess performance?

- Microbenchmarks
  - Libraries (DGEMM, FFT)
  - Communication (p2p, collective)
  - …
- Application Microbenchmark
  - HPL (for historic reasons?)
  - NAS (outdated)
  - …
- Applications

# We still somehow agree on FLOPS

- … because that's what we always did
  - And it's an OK metric
- But the benchmarks should reflect the workload
  - "Sustained performance"
  - Cf. "real application performance"
- In the Blue Waters context
  - "Sustained Petascale Performance" (SPP)
  - Reflects the NSF workload

# The SPP Metric

- Enables us to
  - compare different computer systems
  - Verify system performance and correctness
  - Monitor performance through lifetime
  - Guide design of future systems
- It has to represent the "average workload" and must still be of manageable size
  - We chose ten applications (8 x86, 4 GPU)
  - Performance is geometric mean of all apps

# Blue Waters in a Nutshell

- XE6 with AMD Interlagos 2.3-2.6 (3.0?) GHz
  - ~390k BD modules, ~780k INT cores
- XK6 with Kepler GPUs
  - ~3k
- Gemini Torus
  - Very large (23x24x24), BB-challenged, torus
- How do we make sure the (heterogeneous) system is ready to fulfill it's mission?
  - Well, confirm a certain SPP number (> 1PF!)

# Validating a System Model – Memory I

- Stride-1 word load/store/copy (32 MiB data):
  - 1 int core r/w/c: 3.8 / 4 / 3 GB/s
  - 16 int cores (1 IL) r/w/c: 32 / 16 / 9.6 GB/s
  - 32 int cores (2 IL) r/w/c: 64 / 32 / 19.8 GB/s
- Comments:
  - Very high fairness between cores
  - Very low variance between measurements

**Measured with Netgauge 2.4.7, pattern memory/stream**

# Validating a System Model – Memory II

- CL latency (random pointer chase, 1 GiB data):
  - 1 int core: 110 ns
  - 16 int cores (1 IL): 257 ns
  - 32 int cores (2 IL): 258 ns
- Comments:
  - High fairness between cores
  - Low variance between measurements

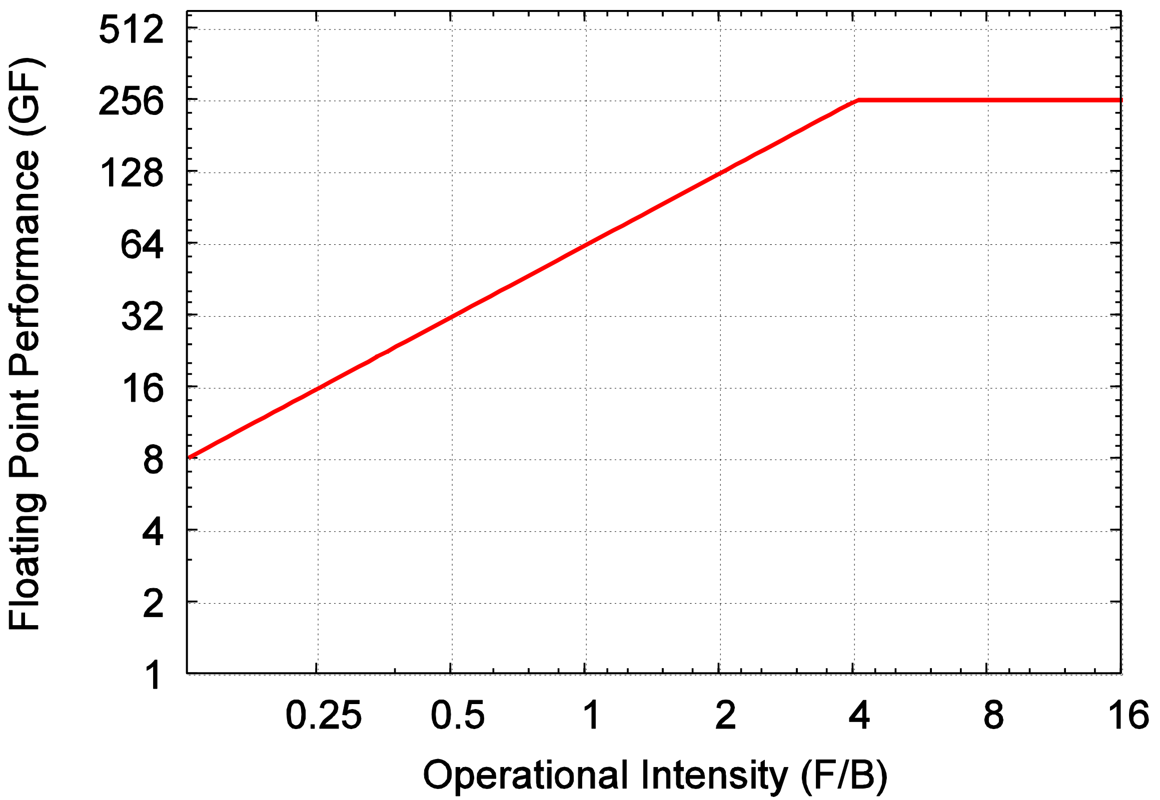
Measured with Netgauge 2.4.7, pattern memory/pchase

# Validating a System Model – Memory III

- Random word access bandwidth (32 MiB data):
  - 1 int core r/w/c: 453 / 422 / 228 MiB/s
  - 16 int cores (1 IL) r/w/c: 241 / 119 / 77 MiB/s
  - 32 int cores (2IL) r/w/c: 241 / 119 / 77 MiB/s
- Comments:
  - Very high fairness between cores
  - Very low variance between measurements

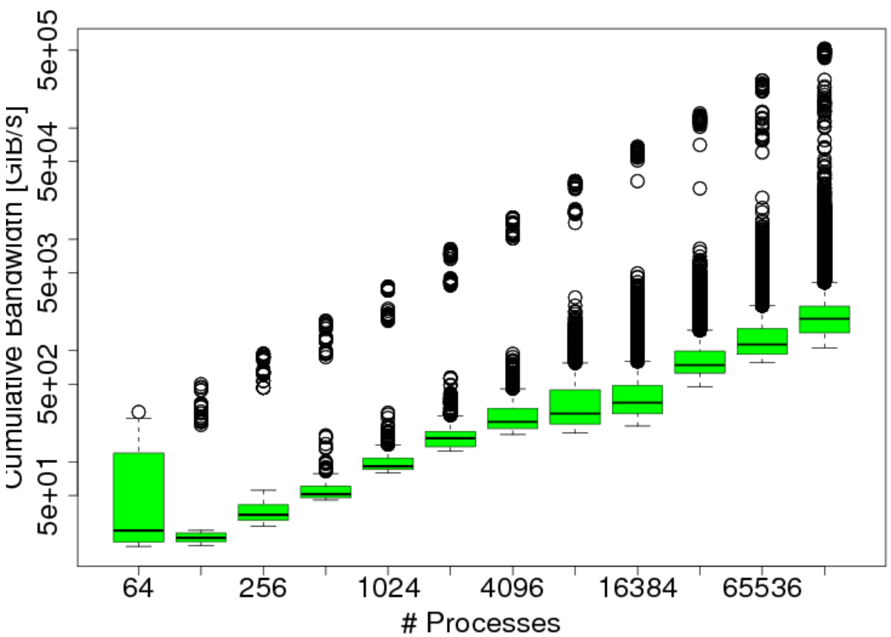
**Measured with Netgauge 2.4.7, pattern memory/rand**

# Roofline Model for Interlagos

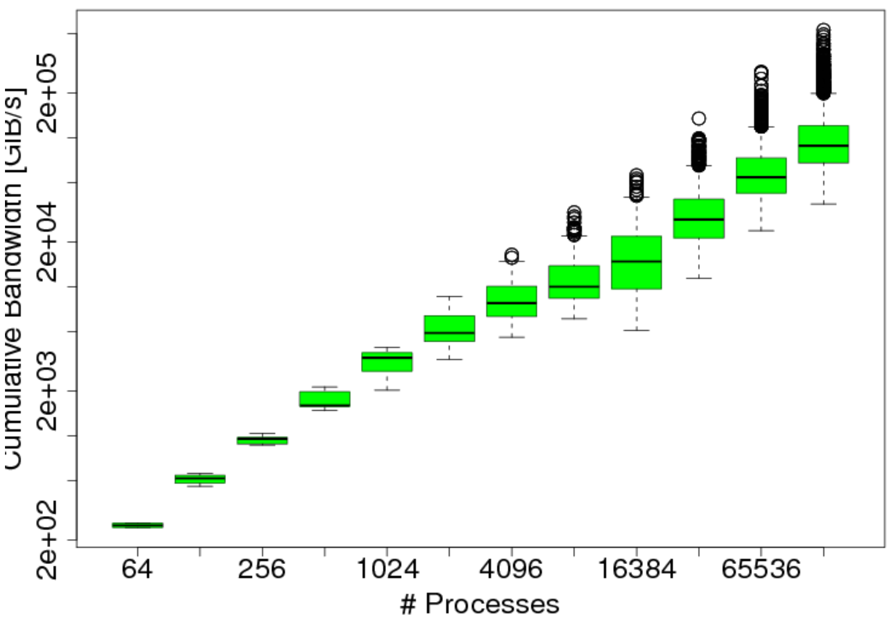


# Validating a System Model – Network Scaling

- Effective Bisection Bandwidth and Variance
  - Expect (3D torus bisection limit): 7.5 TB/s




32 processes per node

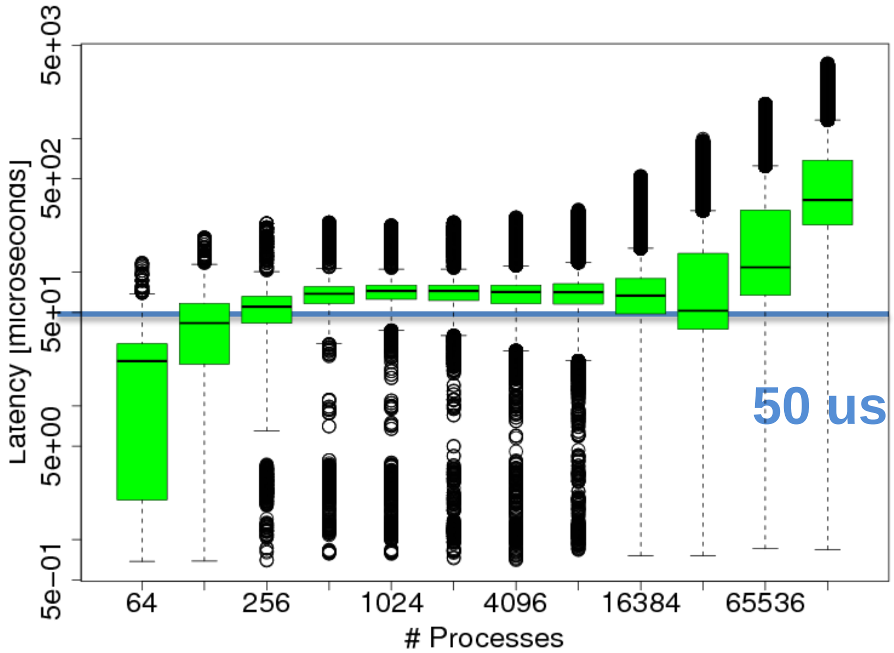



1 process per node

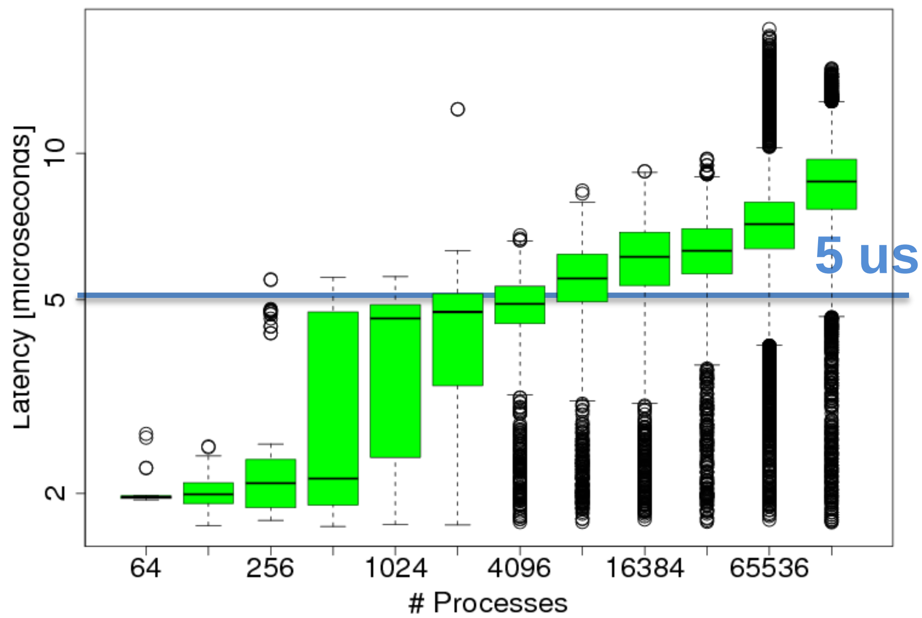
Measured with Netgauge 2.4.7, pattern ebb

# Validating a System Model – Network Scaling

- Average random latency and variance



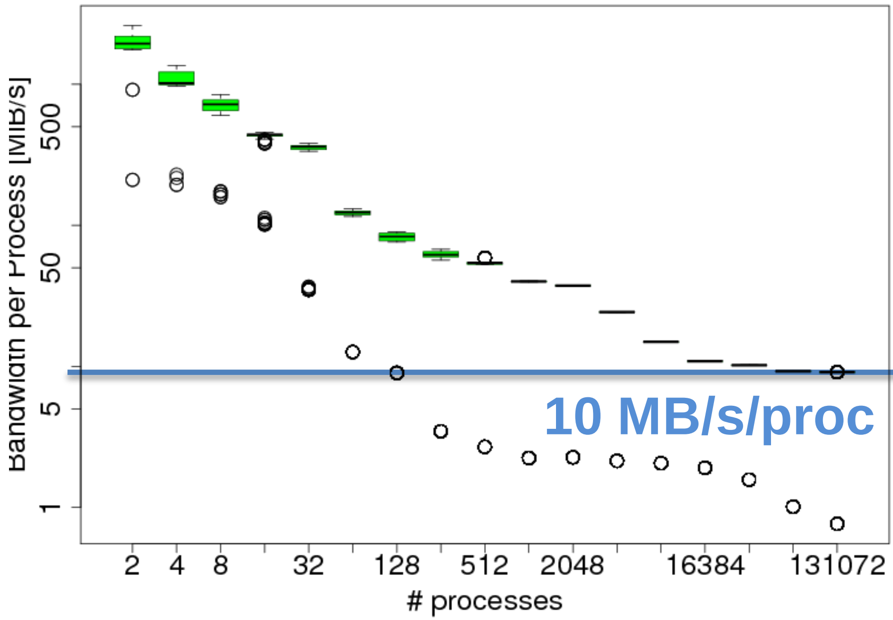

32 processes per node



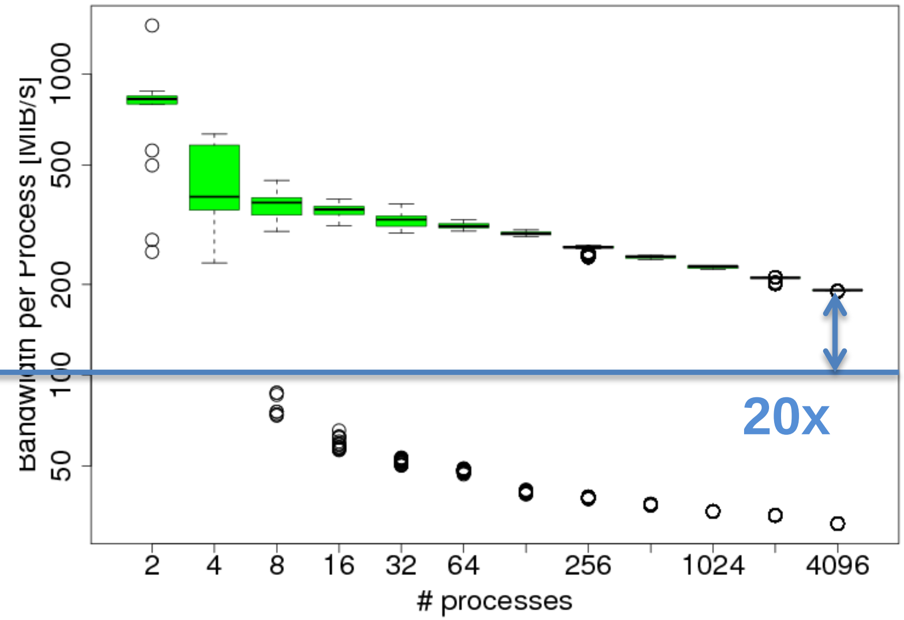

1 process per node

# Validating a System Model – Collectives

- Large message (4k) alltoall performance
  - Model: unclear (depends on mapping etc.)




32 processes per node




1 process per node

**Measured with Netgauge 2.4.7, pattern nbcolls**

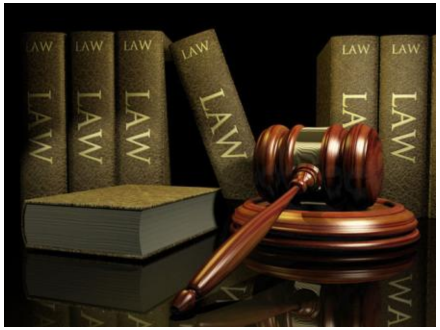
# The SPP Application Mix

- Representative Blue Waters applications:
  - NAMD – molecular dynamics
  - MILC, Chroma – Lattice Quantum Chromodynamics
  - VPIC, SPECFEM3D – Geophysical Science
  - WRF – Atmospheric Science
  - PPM – Astrophysics
  - NWCHEM, GAMESS – Computational Chemistry
  - QMCPACK – Materials Science

# Upping my FLOPS (if I was a vendor)

- Algorithms may have different FLOP counts
  - Slow time to solution but high FLOPS (dense LA)
  - Same time to solution, more FLOPS
  - Single of half FLOPS (esp. GPUs)
  - Redundant FLOPS for parallel codes
- Performance counters are thus not reliable!
  - Just count the observed, not the necessary FLOPS

# Reference FLOP Counts




- We establish "reference FLOP count"
  - Specific to an input problem
  - Ideally established analytically
  - Or (if necessary) on reference code on x86
    - Single-core run (or several parallel runs)
- Input problem needs to be clearly defined
  - Set the right expectations
  - Real, complete science run vs. maximum FLOPS

# The Grand Modeling Vision

- Our very 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

  (Steps 1–4: Analytic)

  5) Determine sequential baseline
  6) Determine communication parameters

  (Steps 5–6: Empiric)

Hoefler, Gropp, Snir, Kramer: Performance Modeling for Systematic Performance Tuning , SC11

# A Simplified Modeling Method

- Fix input problem (omit step 1)
- No fancy tools, simple library using PAPI (libPGT)
- Determine performance-critical kernels
  - We demonstrate a simple method to identify kernels
- Analyze kernel performance
  - Using black-box counter approach
  - More accurate methods if time permits
- Establish system bounds
  - What can be improved? Are we hitting a bottleneck?

# Performance Counter Sanity Checks

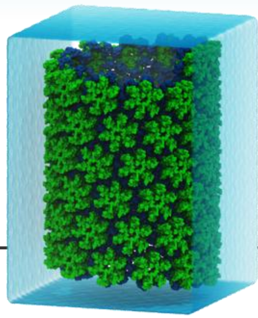
Table 1: Performance characteristics for simple kernels

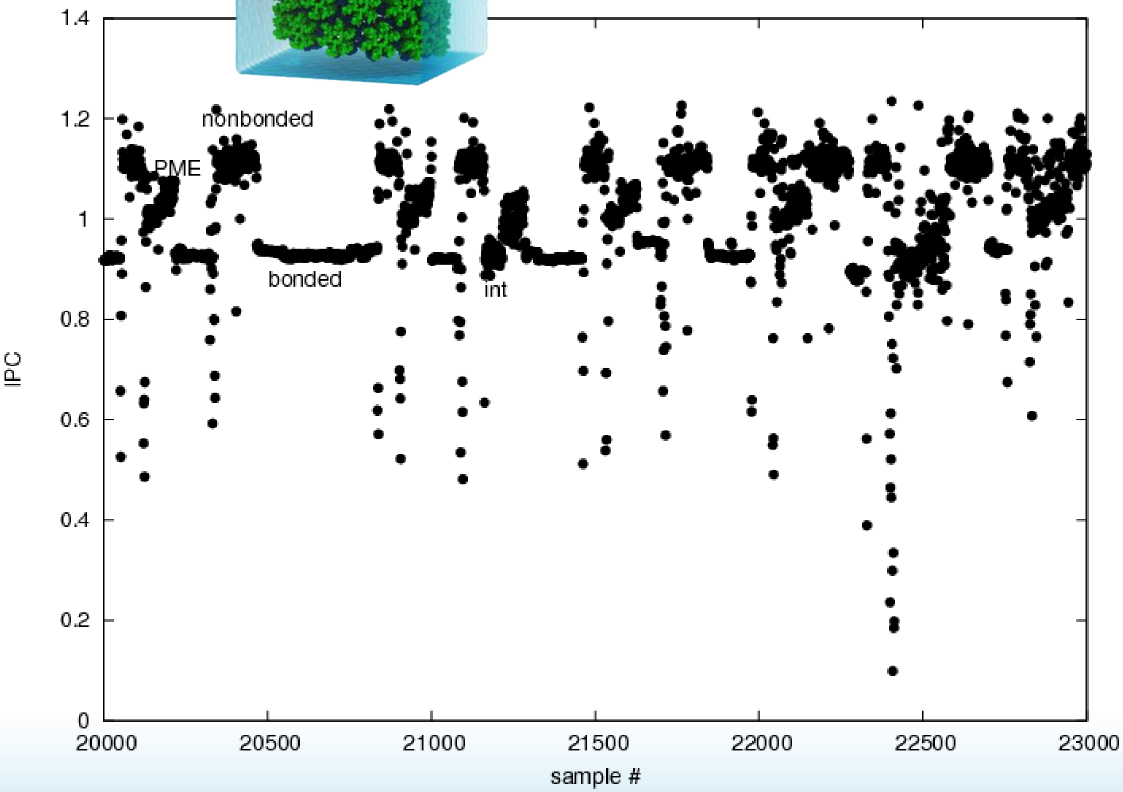
| kernel | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| triad s | 300 | 407 | 3958 | 1.1 | 0.1 | 0.1 | 2.3 |
| triad l | 241 | 156 | 1574 | 1.0 | 0.1 | 0.1 | 2.6 |
| stencil s | 1089 | 2508 | 9172 | 1.4 | 0.3 | 0.5 | 2.3 |
| stencil l | 181 | 458 | 1684 | 1.4 | 0.3 | 0.1 | 2.6 |
| dgemm l | 3690 | 7940 | 3297 | 5.0 | 2.4 | 1.6 | 2.3 |
| reg int | 2000 | 0 | 0 | 0.0 | 0.0 | 0.8 | 2.6 |

- Running small test kernels to check counters
- s=small, l=large
- Stream: 2 GB/s per integer core
- LL_CACHE_MISSES are L2 misses!?
  - Still a proxy metric (use with caution!)

# NAMD

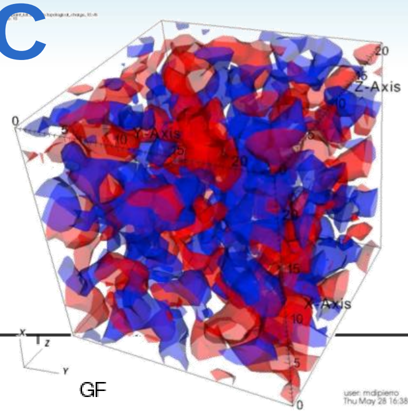
| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| nonbonded | 2460 | 1377 | 7506 | 1.1 | 0.2 | 1.1 | 2.3 |
| PME | 1772 | 1408 | 3299 | 1.7 | 0.4 | 0.8 | 2.3 |
| bonded | 1617 | 723 | 1821 | 0.8 | 0.4 | 0.7 | 2.3 |
| integrate | 1394 | 581 | 4573 | 0.8 | 0.1 | 0.6 | 2.3 |



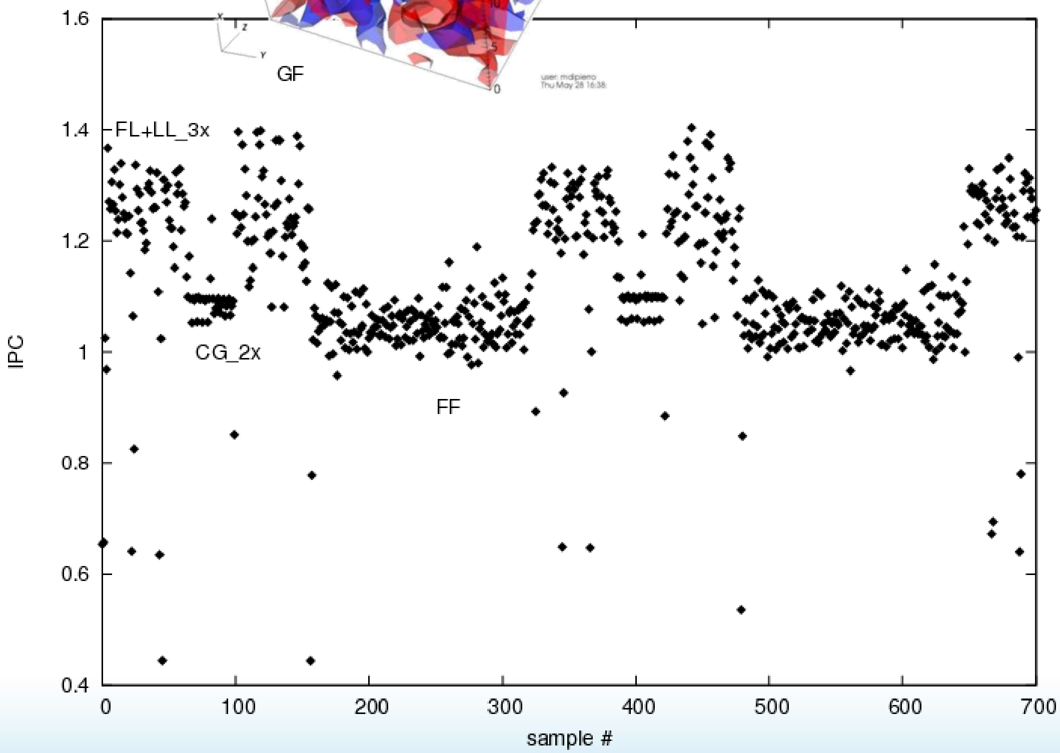



- Dynamic scheduling complicates model
- Excellent cache locality
- PME performs well but will slow down at scale (alltoall)
- Good IPC

# MILC



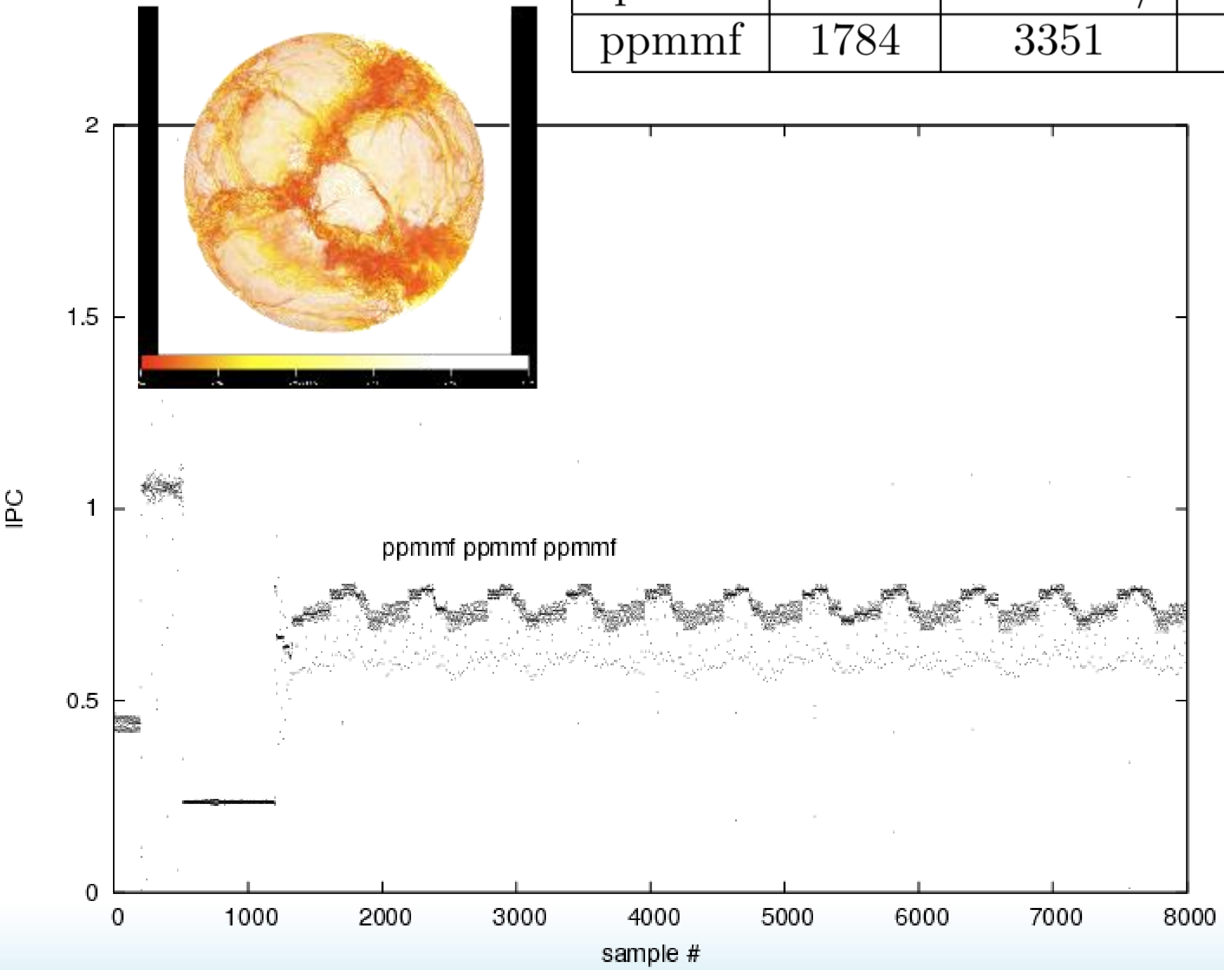

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| LL | 1123 | 707 | 3179 | 1.1 | 0.2 | 0.5 | 2.2 |
| FL | 1475 | 1425 | 3233 | 1.9 | 0.4 | 0.6 | 2.4 |
| FF | 1305 | 1057 | 2055 | 1.2 | 0.5 | 0.5 | 2.4 |
| GF | 1414 | 1087 | 3719 | 1.4 | 0.3 | 0.6 | 2.4 |
| CG | 1353 | 1082 | 3051 | 1.7 | 0.4 | 0.6 | 2.5 |




- Five phases, CG most critical at scale
- Low FLOPs and IPC
  - Turbo boost seems to help here!
- Low FLOPs are under investigation (already using SSE)

# PPM

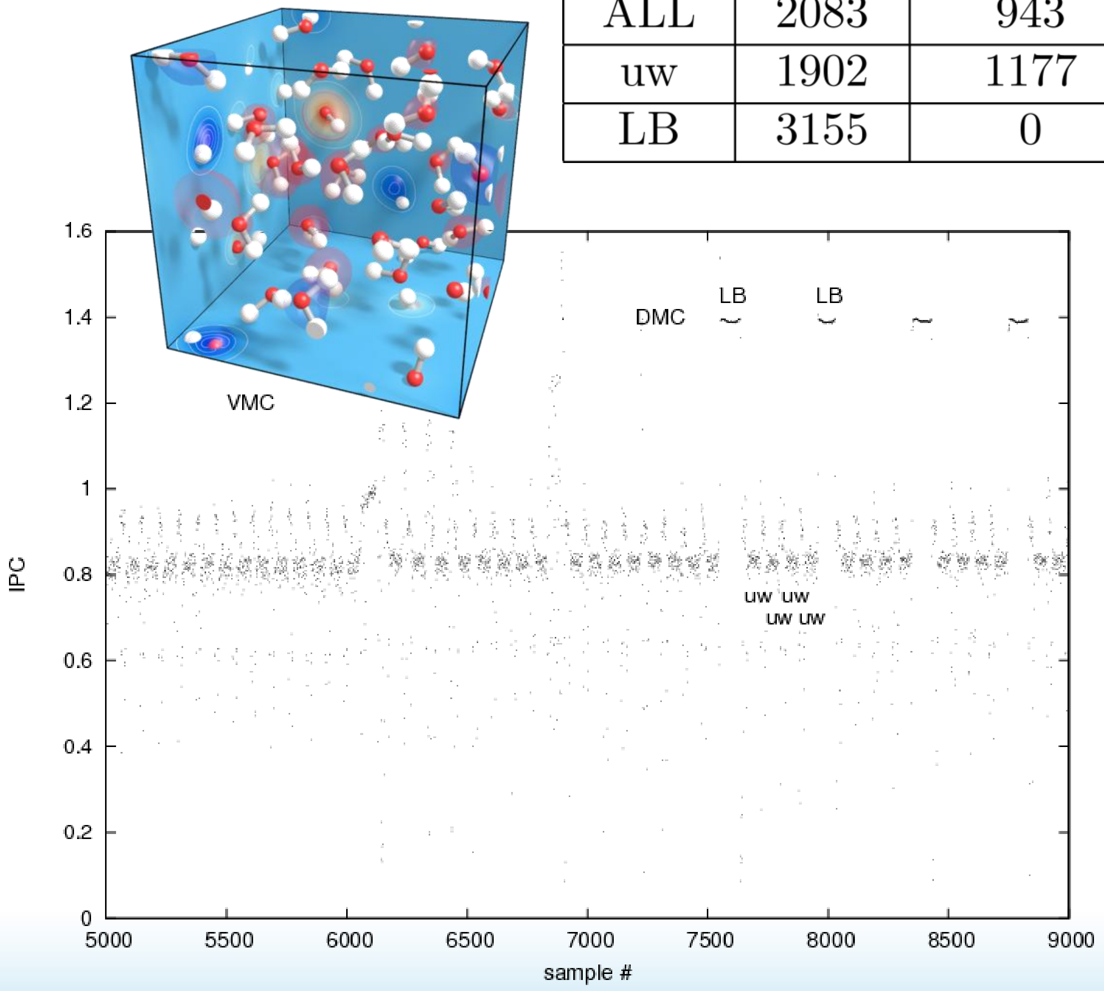
| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| ppmmf | 1784 | 3351 | 2839 | 3.0 | 1.1 | 0.7 | 2.4 |




- Many micro-phases
- Hard to instrument
- Very highly optimized by science team
  - Cache blocking
  - High FLOP rate
  - High locality

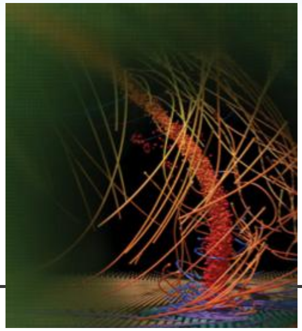
# QMCPACK

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| ALL | 2083 | 943 | 1933 | 1.1 | 0.5 | 0.9 | 2.3 |
| uw | 1902 | 1177 | 2433 | 1.5 | 0.5 | 0.8 | 2.3 |
| LB | 3155 | 0 | 18 | 0.0 | 0.0 | 1.4 | 2.3 |

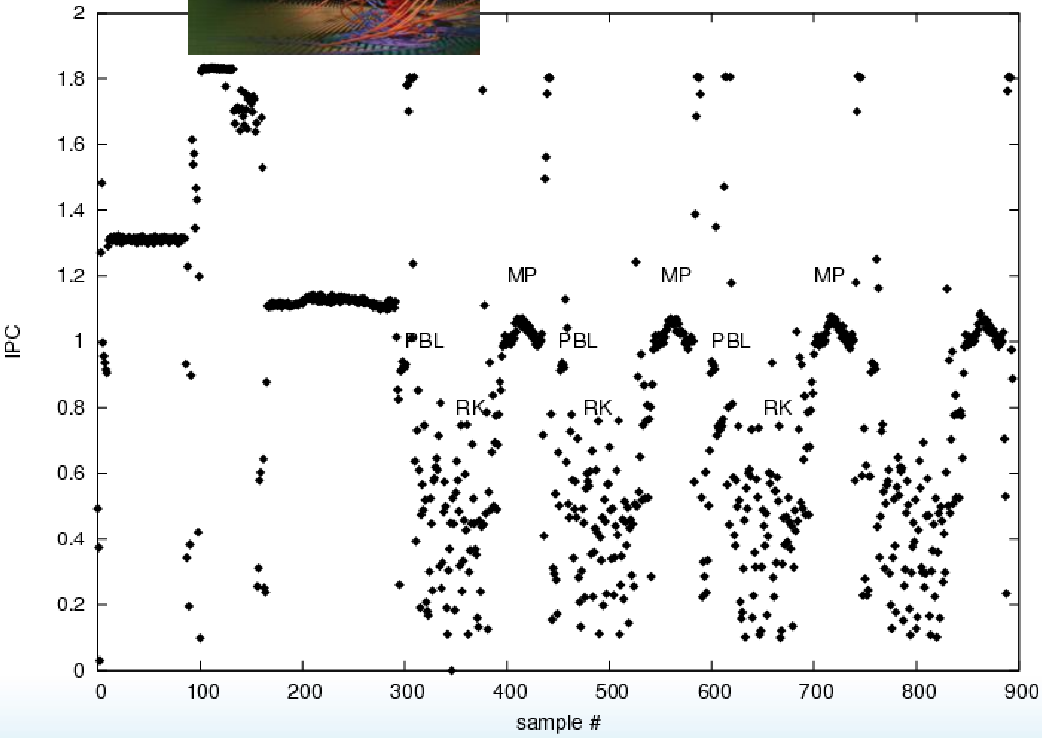



- Variational Monte Carlo initializes
- Performance issues are investigated
- Diffusion Monte Carlo:
  - load balance (LB)
  - update walker (uw)

# WRF



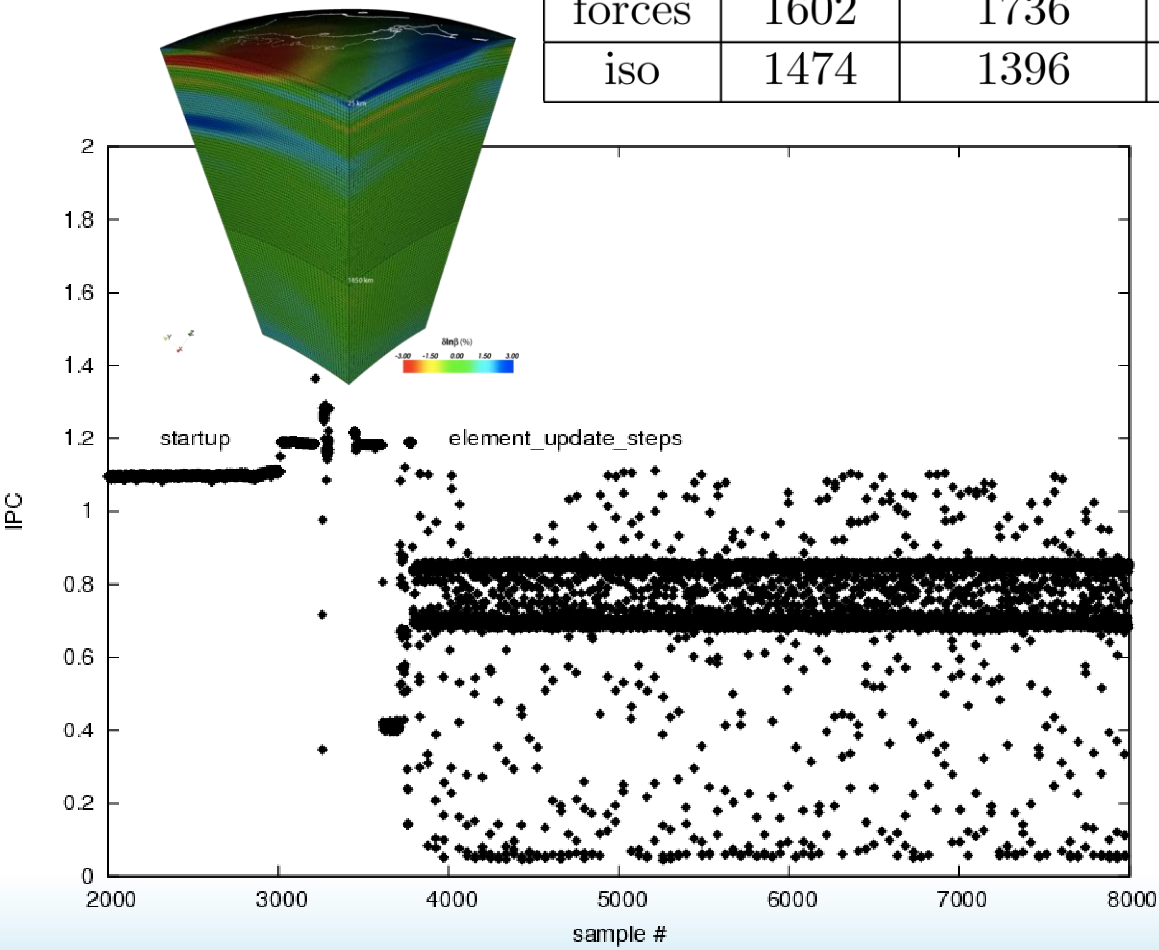
| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| MP | 2647 | 590 | 1288 | 0.5 | 0.5 | 1.0 | 2.6 |
| PBL | 2197 | 566 | 4511 | 0.5 | 0.1 | 0.9 | 2.6 |
| RKt | 1328 | 2695 | 11842 | 2.0 | 0.2 | 0.6 | 2.3 |
| RKs | 1764 | 1120 | 4967 | 0.8 | 0.2 | 0.7 | 2.5 |




- Microphysics dominates
  - Low performance, many branches
- Planet Boundary Layer also problematic
  - Turbo Boost helps!
- Runge Kutta is fast
  - High locality

# SPECFEM3D

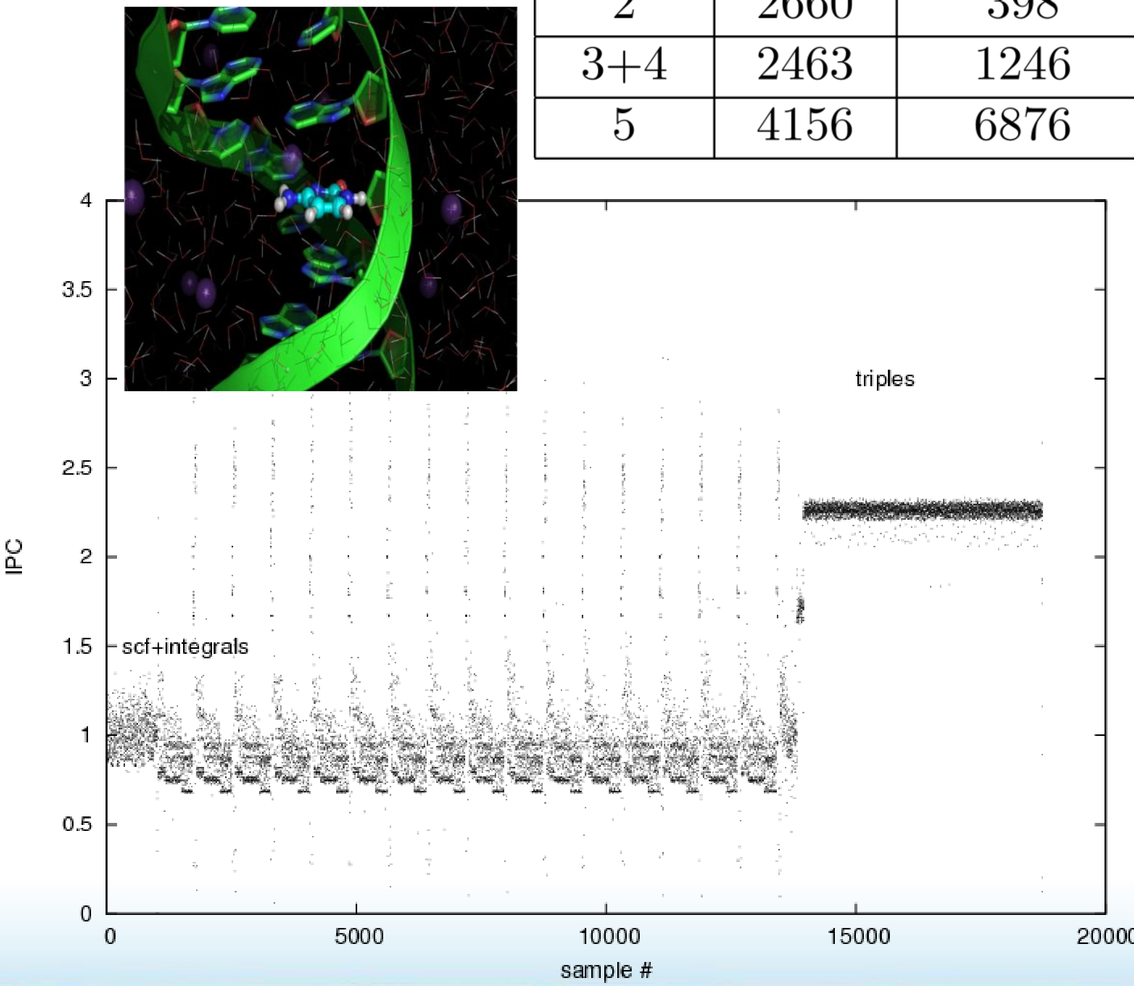
| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| tiso | 1973 | 2010 | 1197 | 1.9 | 1.8 | 0.8 | 2.3 |
| forces | 1602 | 1736 | 4577 | 1.5 | 0.4 | 0.7 | 2.3 |
| iso | 1474 | 1396 | 1617 | 1.6 | 0.9 | 0.6 | 2.3 |




- Two phases, both do small mat-mat mult
- Internal forces perform well

# NWCHEM

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| 1 | 2616 | 431 | 5464 | 0.3 | 0.1 | 1.0 | 2.6 |
| 2 | 2660 | 398 | 4818 | 0.3 | 0.1 | 1.0 | 2.6 |
| 3+4 | 2463 | 1246 | 6030 | 0.9 | 0.2 | 1.0 | 2.6 |
| 5 | 4156 | 6876 | 15583 | 3.5 | 0.4 | 1.6 | 2.6 |




- Highly optimized
  - Even running in turbo boost!
- Very good locality
- Steps 3+4 decent
- Step 5 close to peak!

# Some Early Conclusions

- Average CI: 0.43 FLOPS/B (min: 0.1, max: 1.8)
  - Required CI: 8 GF/s / 4 GB/s → 4 FLOPS/B
- Average Effective Frequency: 2.40 GHz
  - Anticipated frequency: 2.45 GHz
- Average FLOP rate: 1.48 GF (min: 398 GF (WRF), max: 6.876 GF (NWCHEM))
  - 15% of peak ☺
  - Standard deviation: 1.37 GF (!!!)

# Conclusions & Future Work

- We analyzed performance of several SPP applications
  - Performance modeling techniques
- Kernel classification through IPC works well
  - Not automatic yet
- Kernel profiling works mostly
  - Need better/more interpretation of counters
- Extending towards communication models
  - “MPI counters”, congestion, etc.

# Acknowledgments


- Thanks to
  - Gregory Bauer (pulling together the data)
  - Victor Anisimov, Eric Bohm, Robert Brunner, Ryan Mokos, Craig Steffen, Mark Straka (SPP PoCs)
  - Bill Kramer, Bill Gropp, Marc Snir (general modeling ideas/discussions)
  - The Cray performance group (Joe Glenski et al.)
- The National Science Foundation