


Scalable System Measurement and Performance Analysis: Recent Progress

**Rob Fowler
RENCI**

**Workshop on
Performance & Productivity
of Extreme-Scale Parallel Computing
Oct. 16 2006.**





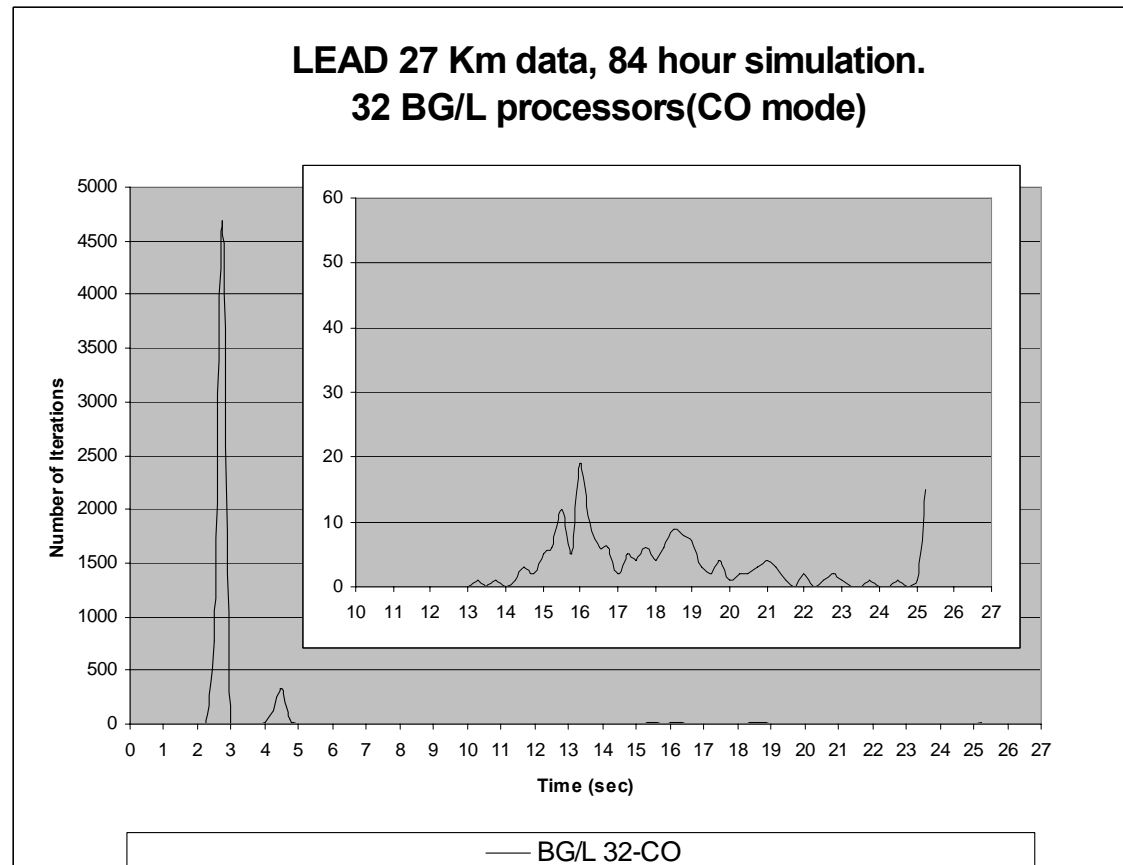
(Scalable System) Measurement and Performance Analysis: Recent Progress

**Rob Fowler
Workshop on
Performance & Productivity
of Extreme-Scale Parallel Computing
Oct. 16 2006.**

The I/O Issue and Scaling.

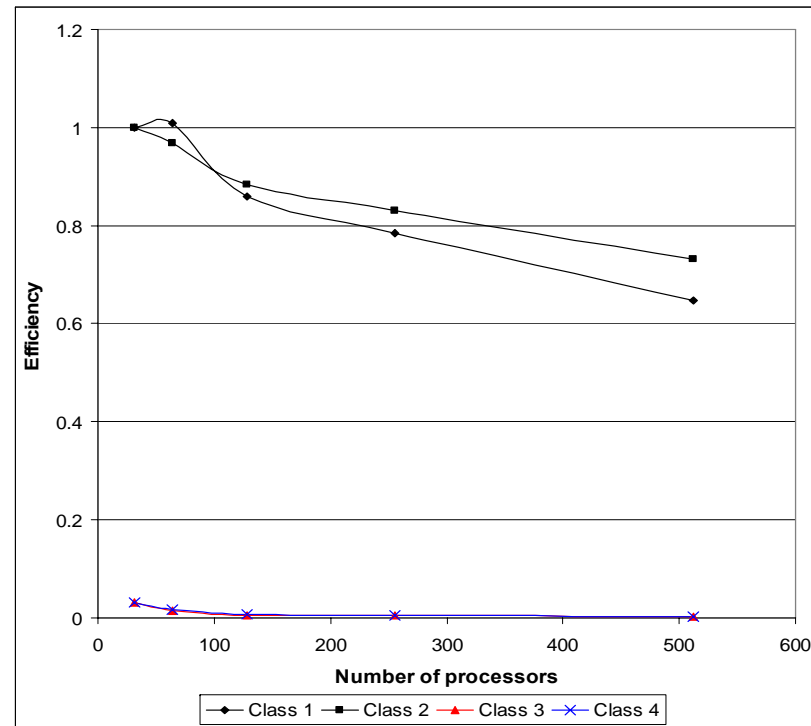
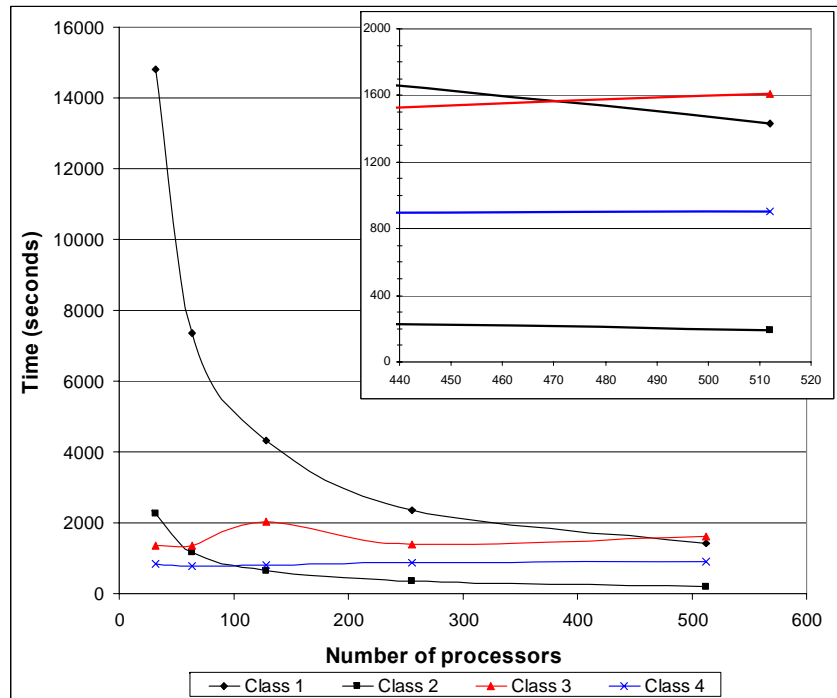
- **Seymour Cray (1976):**
 - I/O has certainly been lagging the last decade.
- **D. Kuck (1988):**
 - Also, I/O needs lots of work.
- **Dave Patterson (1994):**
 - Terabytes >> Teraflops or Why Work on Processors When I/O is Where the Action is?
- **Seymour Cray:**
 - A supercomputer is a device that turns a compute-bound problem into an I/O-bound problem.

A "Real" WRF problem on BG/L



- WRF run for LEAD : CONUS 27km grid, 84 simulated hours, hourly simulation output, checkpoint every 12 simulated hours.
- NetCDF used for I/O.

Speedup and efficiency of WRF on BG/L



- Four classes of iterations: 1 & 2 just compute, 3 & 4 do file I/O using NetCDF.
- Computation scales reasonably well. I/O does not scale at all.
- What about weak scaling, i.e. run on a “petascale challenge input”:
 - With a bigger problem, the computation will scale better on more processors.
 - With a bigger problem on more data, I/O will be even more of a bottleneck.
- Procurement benchmarks: Output only once, at the end of the run.

A Simple Model of This WRF Run

- Postulate an Amdahl's law model for each class of iteration:

$$t(P) = t_{parallel} / P + t_{sequential}$$

- For each class of iteration, fit the parameters:

(lst. sq., $R^2 > .98$)

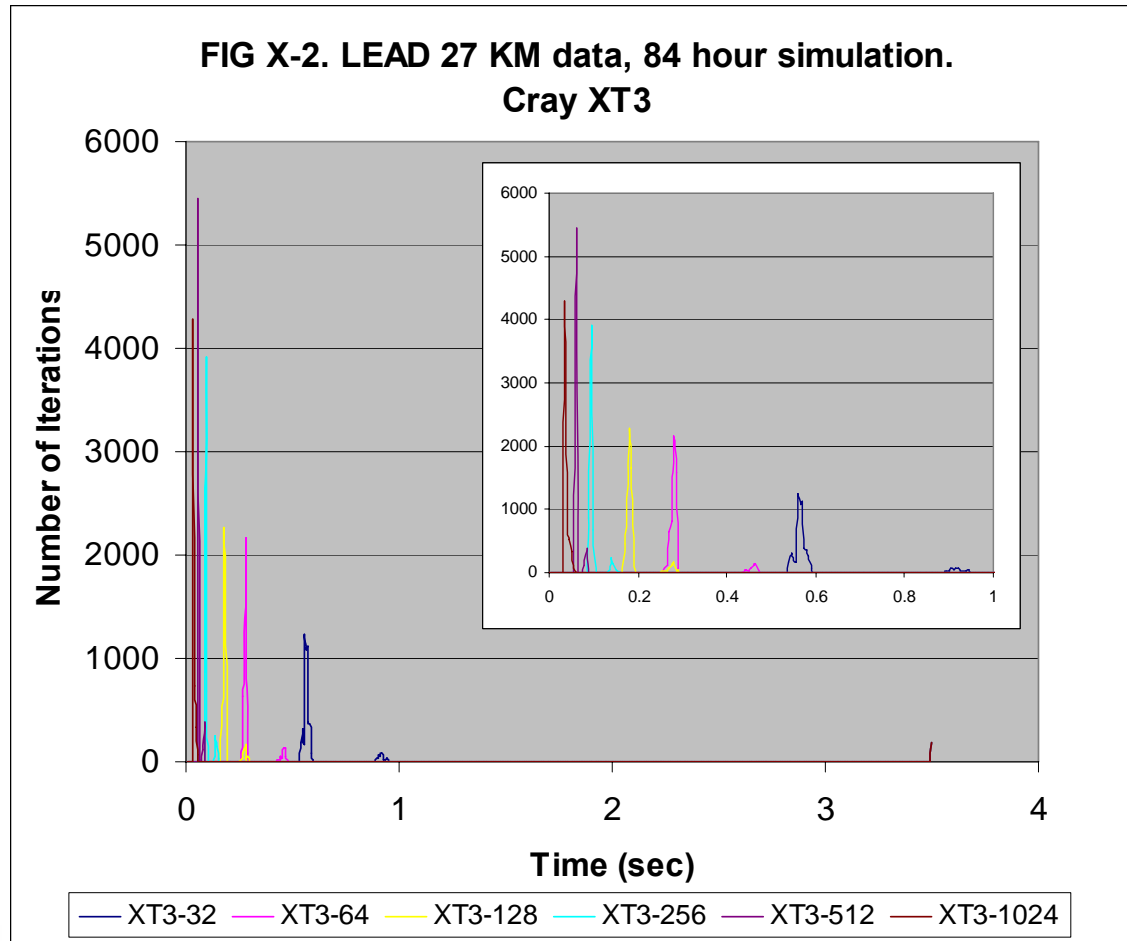
(use asymptote)

Class	$t_{parallel}$	$t_{sequential}$
1	457600	1400 sec
2	72280	200
3	0	1600
4	0	900

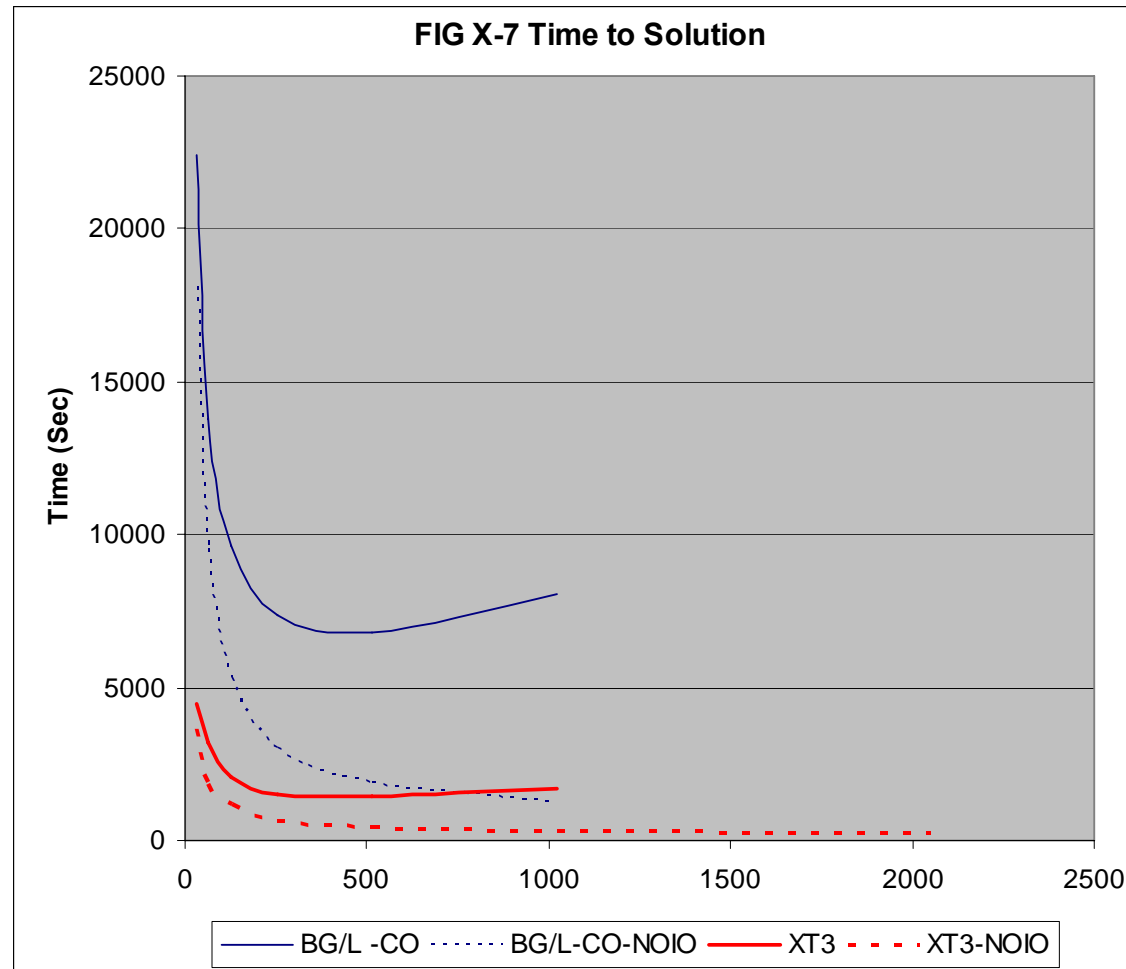
- Aggregate Model:

$$t(P) = 529900/P + 4100 \text{ (sec)}$$

WRF on XT3



Summary of WRF Results.



Takeaway messages


I/O continues to be the elephant in the room.

Use alternatives to NetCDF, but this doesn't make the problem go away.

"Good" news: An ensemble of 50 WRF runs is a lot more useful than a 50X bigger single run.

Petascale capacity vs. petascale capability?

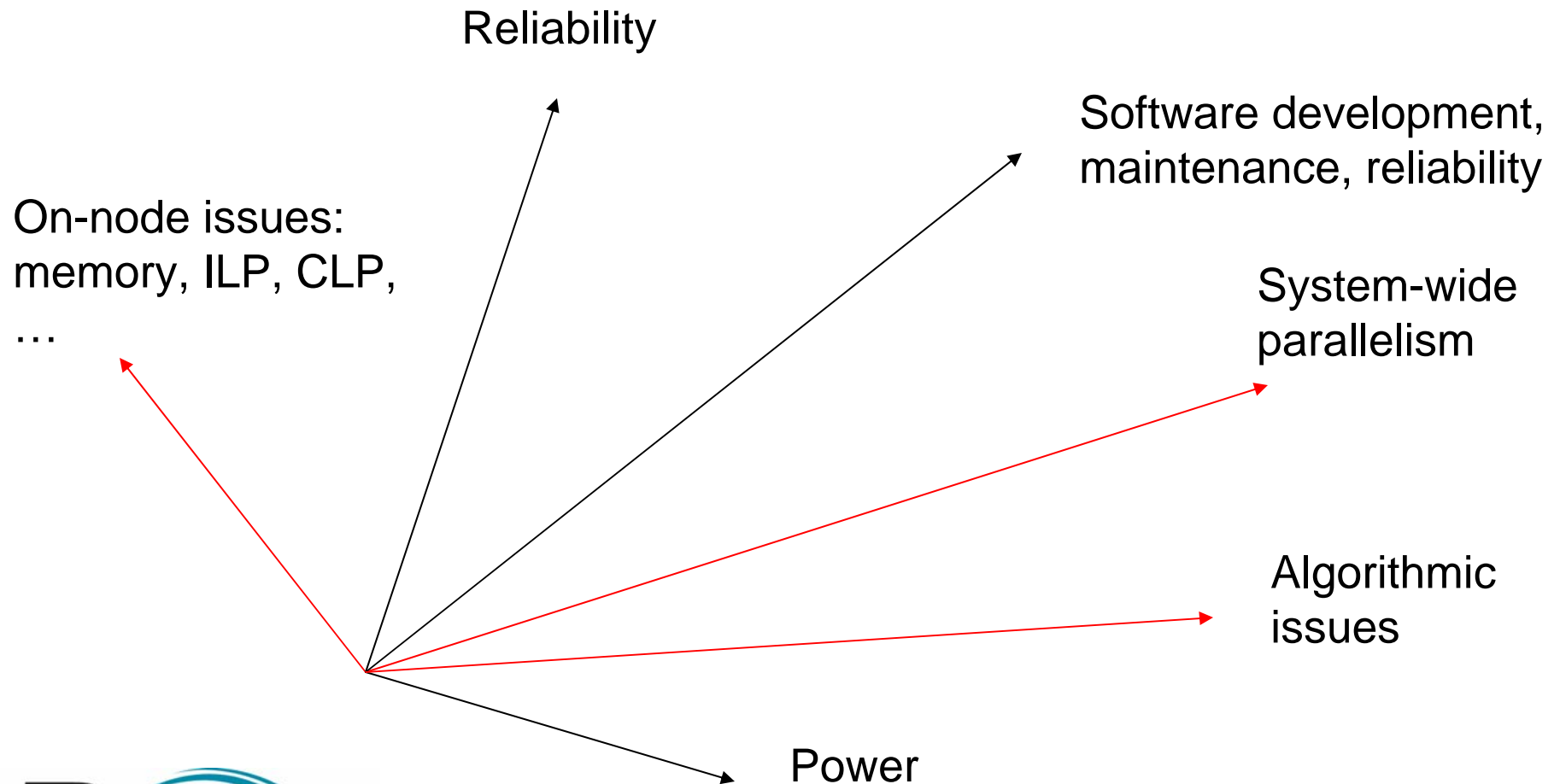
(SC06 BOF on Petascale Performance Eval. Wed 5PM)



Scalable (System Measurement and Performance Analysis): Recent Progress

**Rob Fowler
Workshop on
Performance & Productivity
of Extreme-Scale Parallel Computing
Oct. 16 2006.**

Performance/Productivity at the High End: Optimization in multiple dimensions



Moore's law

Circuit element count doubles every N months. ($N \sim 18$)

- **Why: Features shrink, semiconductor dies grow.**
- **Corollaries: Gate delays decrease. Wires are relatively longer.**
- **In the past the focus has been making "conventional" processors faster.**
 - **Faster clocks**
 - **Clever architecture and implementation → instruction-level parallelism.**
 - **Clever architecture, HW/SW Prefetching, and massive caches ease the "memory wall" problem.**
- **Problems:**
 - **Faster clocks --> more power ($P \sim V^2F$)**
 - **More power goes to overhead: cache, predictors, "Tomasulo", clock, ...**
 - **Big dies --> fewer dies/wafer, lower yields, higher costs**
 - **Together --> Power hog processors on which some signals take 6 cycles to cross.**

P4: Competing with charcoal?



The Sea Change: More On-Chip Parallelism

- **Single thread ILP**
 - **Instruction pipelining constraints, etc.**
 - **Memory operation scheduling for latency, BW.**
 - Memory arch. Becoming a point-to-point network.
- **Multi-threading CLP**
 - **Resource contention within a core**
 - Memory hierarchy
 - Functional units, ...
- **Multi-core CLP**
 - **Chip-wide resource contention**
 - Shared on-chip components of the memory system
 - Shared chip-edge interfaces

Challenge: Programmers (and optimizing compilers) will need to be able understand all this. Tools needed.

Why is performance not obvious?

Hardware complexity

- **Keeping up with Moore's law with one thread.**
- **Instruction-level parallelism.**
 - Deeply pipelined, out-of-order, superscalar, threads.
- **Memory-system parallelism**
 - Parallel processor-cache interface, limited resources.
 - Need at least k concurrent memory accesses in flight.

Software complexity

- **Competition/cooperation with other threads**
- **Dependence on (dynamic) libraries.**
- **Languages: "Object Orientation Considered Harmful"**
- **Compilers**
 - Aggressive (-O3+) optimization conflicts with manual transformations.
 - Incorrectly conservative analysis and optimization.

“Simple” recipe for performance.

- **Simultaneously achieve (or balance)**
 - High ILP,
 - Memory locality and parallelism,
 - Chip-level parallelism,
 - Concurrent I/O and communication.
- **Address this throughout lifecycle.**
 - Algorithm design and selection.
 - Implementation
 - Repeat
 - Translate to machine-specific code.
 - Maintain algorithms, implementation, compilers.

It gets worse: Scalable HEC

All the problems of on-node efficiency, plus

- **Scalable parallel algorithm design,**
- **Load balance,**
- **Communication performance,**
- **Competition of communication with applications,**
- **External perturbations,**
- **Reliability issues:**
 - **Recoverable errors → performance perturbation.**
 - **Non-recoverable error → You need a plan B**
 - Checkpoint/restart (expensive, poorly scaled I/O)
 - Robust applications

While you're at it ...

- **Promote programmer productivity.**
- **Lower costs.**
- **Decrease overall time to solution.**
- **And protect the huge investment in the existing code base.**

Performance Tuning in Practice



The Trend in Software



At the limit of usability?

Featuritis in extremis, a.k.a. feeping creaturism?



(Special order: \$1200
from Wegner.)

All you need to know about software engineering.

'The Hitchiker's Guide to the Galaxy, in a moment of reasoned lucidity which is almost unique among its current tally of five million, nine hundred and seventy-three thousand, five hundred and nine pages, says of the Sirius Cybernetics Corporation products that "it is very easy to be blinded to the essential uselessness of them by the sense of achievement you get from getting them to work at all. In other words - and this is the rock-solid principle on which the whole of the Corporation's galaxywide success is founded -- their fundamental design flaws are completely hidden by their superficial design flaws."

(Douglas Adams, "So Long, and Thanks for all the Fish")

Dealing with Featuritis

- **Use specialized, interoperating tools.**
 - Encourage fusion of data from multiple sources
 - E.g., performance measurement on your favorite accelerator, NIC, ...
- **Drive workflows with scripts.**
 - Integrate with cluster runtime management
- **Integrate with the development process.**
 - Eclipse/PTP

Rice HPCToolkit: A Review

- **Use Event Based Sampling (EBS)**
 - Low, controllable overhead (<2%)
 - No instrumentation needed/wanted
 - Collect multiple metrics, compute others.
- **Hierarchical correlation with source**
 - Attribution to source line granularity
- **Flat or call stack attribution**
- **Time-varying behavior - epochs**
- **Driven from scripts**
- **Top-down analysis encouraged**

Issue: EBS on Petascale Systems

- **Hardware on BG/L and XT3 both support event based sampling.**
- **Current OS kernels from vendors do not have EBS drivers.**
- **This needs to be fixed!**
 - **Linux/Zeptos?**
- **Issue: Does event-based sampling introduce “jitter”?**
 - **Much less impact than using fine-grain calipers.**
 - **Not unless you have much worse performance problems.**

Problem: Profiling Parallel Programs

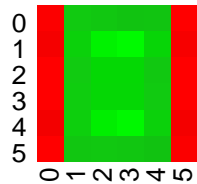
- **Sampled profiles can be collected for about 1% overhead.**
- **How can one productively use profiling on large parallel systems?**
 - **Understand the performance characteristics of the application.**
 - **Study node-to-node variation.**
 - Model and understand systematic variation.
 - Characterize intrinsic, systemic effects in app.
 - Identify anomalies: app. bugs, system effects.
 - **Automate everything.**
 - Do little “glorified manual labor” in front of a GUI.
 - Find/diagnose unexpected problems, not just the expected ones.
- **Avoid the “10,000 windows” problem.**

Statistical Analysis: Bi-clustering

- **Data Input:** an M by P dense matrix of non-negative values.
 - P columns, one for each process(or).
 - M rows, one for each measure at each source construct.
- **Problem: Identify bi-clusters.**
 - Identify a group of processors that are different from the others because they are “different” w.r.t. some set of metrics. Identify the set of metrics.
 - Identify multiple bi-clusters until satisfied.
- **The “Cancer Gene Expression Problem”**
 - **The columns represent patients/subjects**
 - Some are controls, others have different, but related cancers.
 - **The rows represent data from DNA micro-array chips.**
 - **Which (groups of) genes correlate (+ or -) with which diseases?**
 - **There’s a lot of published work on this problem.**
 - **So, use the bio-statisticians’ code as our starting point.**
 - “Gene shaving” algorithm by M.D. Anderson and Rice researchers applied to profiles collected using HPCToolkit.

Cluster 1: 62% of variance in Sweep3D

Cluster #1 (raw)



Weight	Clone ID
-6.39088	sweep.f,sweep:260
-7.43749	sweep.f,sweep:432
-7.88323	sweep.f,sweep:435
-7.97361	sweep.f,sweep:438
-8.03567	sweep.f,sweep:437
-8.46543	sweep.f,sweep:543
-10.08360	sweep.f,sweep:538
-10.11630	sweep.f,sweep:242
-12.53010	sweep.f,sweep:536
-13.15990	sweep.f,sweep:243
-15.10340	sweep.f,sweep:537
-17.26090	sweep.f,sweep:535

```

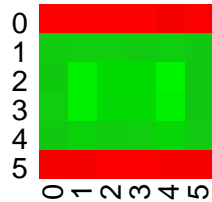
if (ew_snd .ne. 0) then
  call snd_real(ew_snd, phiib, nib, ew_tag, info)
c   nmess = nmess + 1
c   mess = mess + nib
else
  if (i2.lt.0 .and. ibc.ne.0) then
    leak = 0.0
    do mi = 1, mmi
      m = mi + mio
    do lk = 1, nk
      k = k0 + sign(lk-1,k2)
    do j = 1, jt
      phiibc(j,k,m,k3,j3) = phiib(j,lk,mi)
      leak = leak
      &      + wmu(m)*phiib(j,lk,mi)*dj(j)*dk(k)
    end do
    end do
    end do
    leakage(1+i3) = leakage(1+i3) + leak
  else
    leak = 0.0
    do mi = 1, mmi
      m = mi + mio
    do lk = 1, nk
      k = k0 + sign(lk-1,k2)
    do j = 1, jt
      leak =leak+ wmu(m)*phiib(j,lk,mi)*dj(j)*dk(k)
    end do
    end do
    end do
    leakage(1+i3) = leakage(1+i3) + leak
  endif
endif

if (ew_rcv .ne. 0) then
  call rcv_real(ew_rcv, phiib, nib, ew_tag, info)
else
  if (i2.lt.0 .or. ibc.eq.0) then
    do mi = 1, mmi
    do lk = 1, nk
    do j = 1, jt
      phiib(j,lk,mi) = 0.0d+0
    end do
    end do
    end do
  endif
endif

```

Cluster 2: 36% of variance

Cluster #2 (raw)



Weight	Clone ID
-6.31558	sweep.f,sweep:580
-7.68893	sweep.f,sweep:447
-7.79114	sweep.f,sweep:445
-7.91192	sweep.f,sweep:449
-8.04818	sweep.f,sweep:573
-10.45910	sweep.f,sweep:284
-10.74500	sweep.f,sweep:285
-12.49870	sweep.f,sweep:572
-13.55950	sweep.f,sweep:575
-13.66430	sweep.f,sweep:286
-14.79200	sweep.f,sweep:574

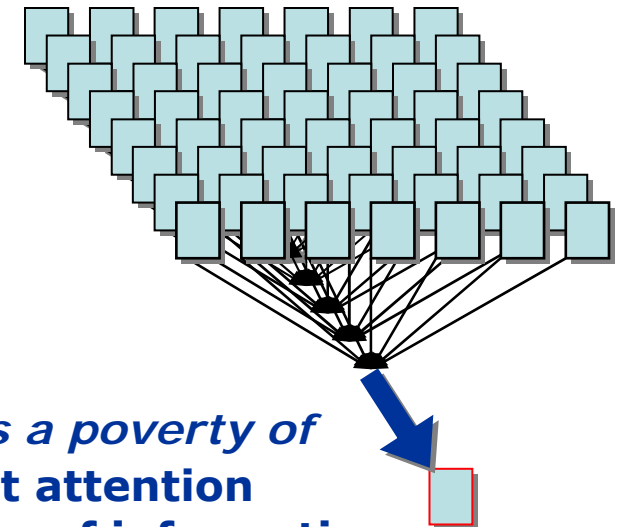
```

if (ns_snd .ne. 0) then
  call snd_real(ns_snd, phijb, njb, ns_tag, info)
c   nmess = nmess + 1
c   mess = mess + njb
else
  if (j2.lt.0 .and. jbc.ne.0) then
    leak = 0.0
    do mi = 1, mmi
      m = mi + mio
      do lk = 1, nk
        k = k0 + sign(lk-1,k2)
        do i = 1, it
          phijbc(i,k,m,k3) = phijb(i,lk,mi)
          leak = leak + weta(m)*phijb(i,lk,mi)*di(i)*dk(k)
        end do
      end do
    end do
    leakage(3+j3) = leakage(3+j3) + leak
  else
    leak = 0.0
    do mi = 1, mmi
      m = mi + mio
      do lk = 1, nk
        k = k0 + sign(lk-1,k2)
        do i = 1, it
          leak = leak + weta(m)*phijb(i,lk,mi)*di(i)*dk(k)
        end do
      end do
    end do
    leakage(3+j3) = leakage(3+j3) + leak
  endif
endif
c J-inflows for block (j=j0 boundary)
c
  if (ns_rcv .ne. 0) then
    call rcv_real(ns_rcv, phijb, njb, ns_tag, info)
  else
    if (j2.lt.0 .or. jbc.eq.0) then
      do mi = 1, mmi
        do lk = 1, nk
          do i = 1, it
            phijb(i,lk,mi) = 0.0d+0
          end do
        end do
      end do
    end do
  end do

```

Further Issues with Scalable Performance Monitoring

- **Scalable performance monitoring**
 - **Profiles/summaries: space efficient but lack temporal detail**
 - **event traces: temporal detail but space demanding**
- **Even collecting profiles/summaries is challenging**
 - **exorbitant data volume (100K nodes)**
 - **high extraction costs, with perturbation risk**
- **Tunable detail and data volume**
 - **application signatures (tasks)**
 - Dynamic filtering of time series.
 - 1st try: Polyline fit using least squares
 - In progress: Wavelet-based filtering
 - **stratified sampling (system)**
 - adaptive node subset



"... a wealth of information creates a poverty of attention, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it." *Herbert Simon*

Sampling Theory: Exploiting Software

- **SPMD models create behavioral equivalence classes**
 - domain and functional decomposition
- **By construction, ...**
 - most tasks perform similar functions
 - most tasks have similar performance
- **Sampling theory and measurement**
 - extract data from “representative” nodes
 - compute metrics across representatives
 - balance volume and statistical accuracy
- **Estimate mean with confidence $1-\alpha$ and error bound d**
 - select a random sample of size n from population of size N



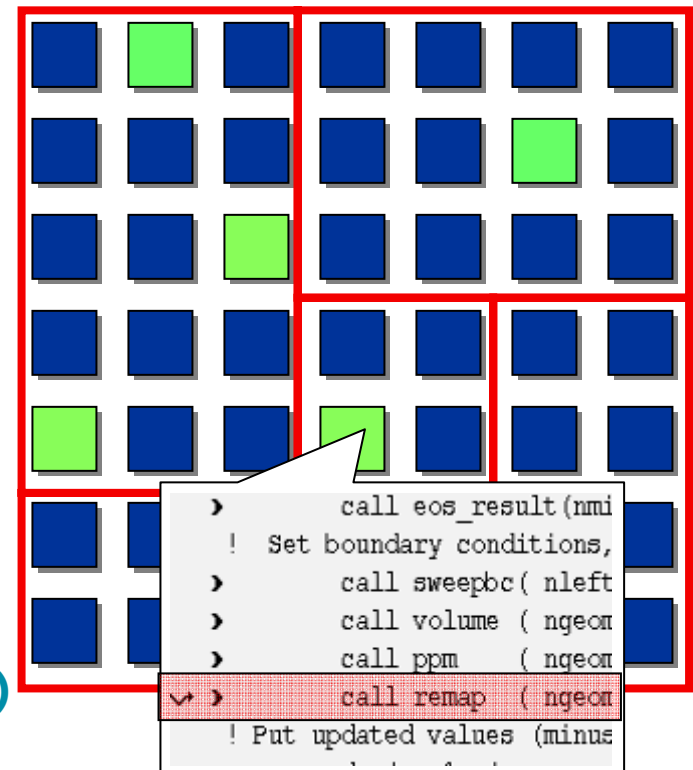
Sampling Must Be Unbiased!

$$n \geq N \left[1 + N \left(\frac{d}{z_\alpha S} \right)^2 \right]^{-1}$$

- approaches $\frac{z_\alpha S}{d}$ for large populations

Adaptive Performance Data Sampling

- **Simple case**
 - select subset n of N nodes
 - collect data from the n
- **Stratified sampling**
 - identify low variance subpopulations
 - sample subpopulations independently
 - reduced overhead for same confidence
- **Metrics vary over time**
 - samples must track changing variance
 - number of subpopulations also vary
- **Sampling options**
 - fixed subpopulations (time series)
 - random subpopulations (independence)



Contact Information

Rob Fowler
rjf@renci.org, rjf@unc.edu
919 445 9670

RENCI
<http://www.renci.org/>