

# Scalability and Scaling of Performance Tools

Rob Fowler  
[rjf@renci.org](mailto:rjf@renci.org)



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



LACSI



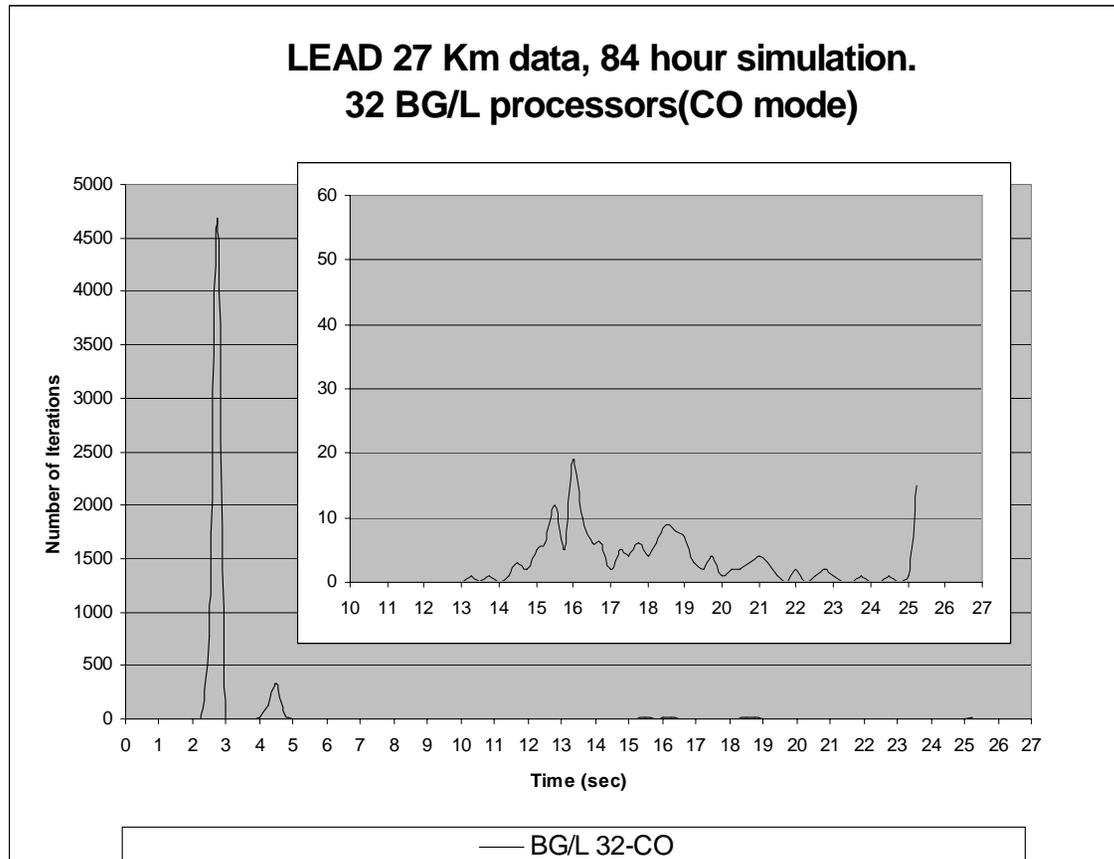
# Overview

- **Preliminaries**
- **Post Mortem Sampling (Rice)**
- **Adaptive Sampling (RENCI)**
- **Scalability Diagnosis Using Profiling (Rice)**

# The I/O Issue and Scaling.

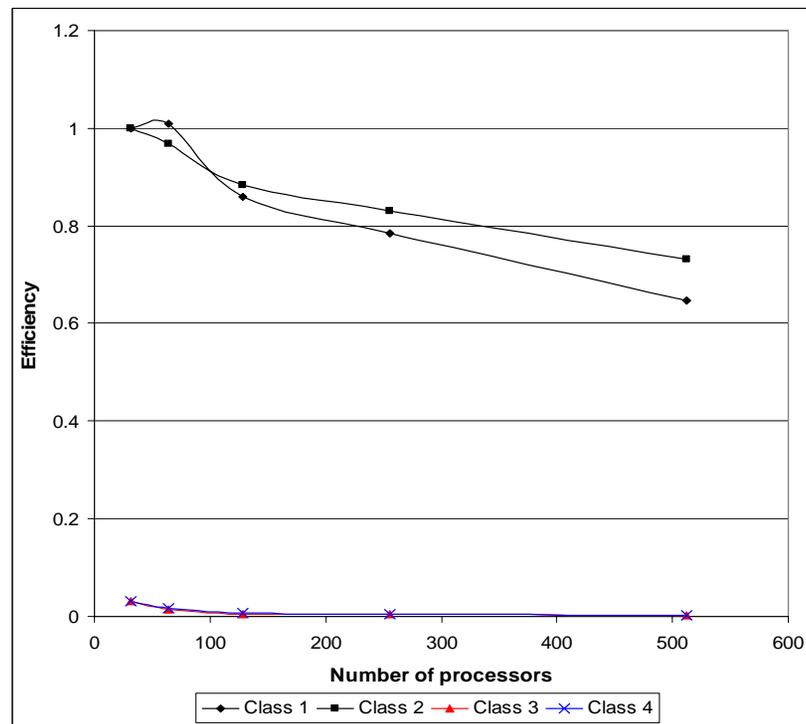
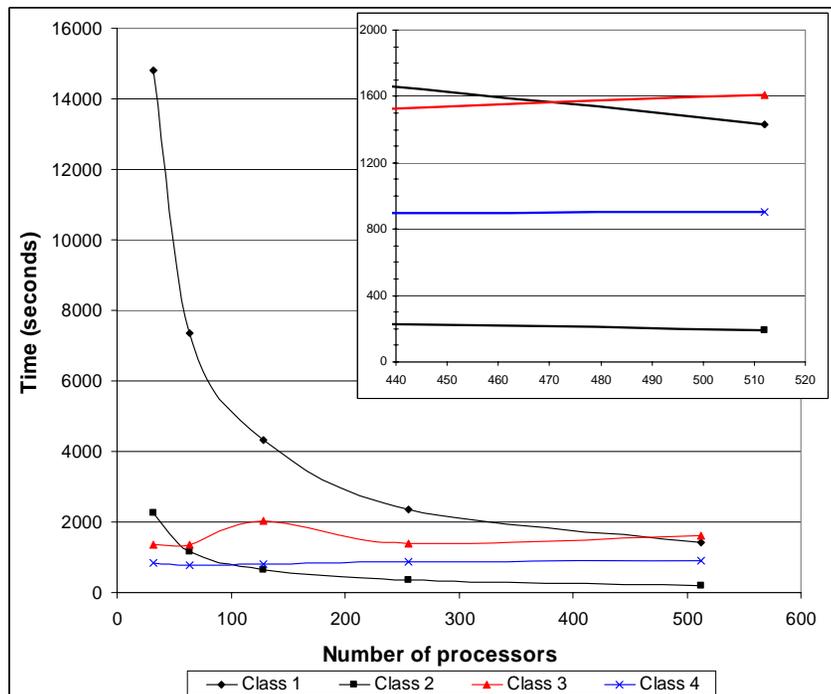
- **Seymour Cray (1976):**
  - I/O has certainly been lagging the last decade.
- **D. Kuck (1988):**
  - Also, I/O needs lots of work.
- **Dave Patterson (1994):**
  - Terabytes >> Teraflops or Why Work on Processors When I/O is Where the Action is?
- **Seymour Cray (maybe Ken Batchter):**
  - A supercomputer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

# A "Real" WRF problem on BG/L



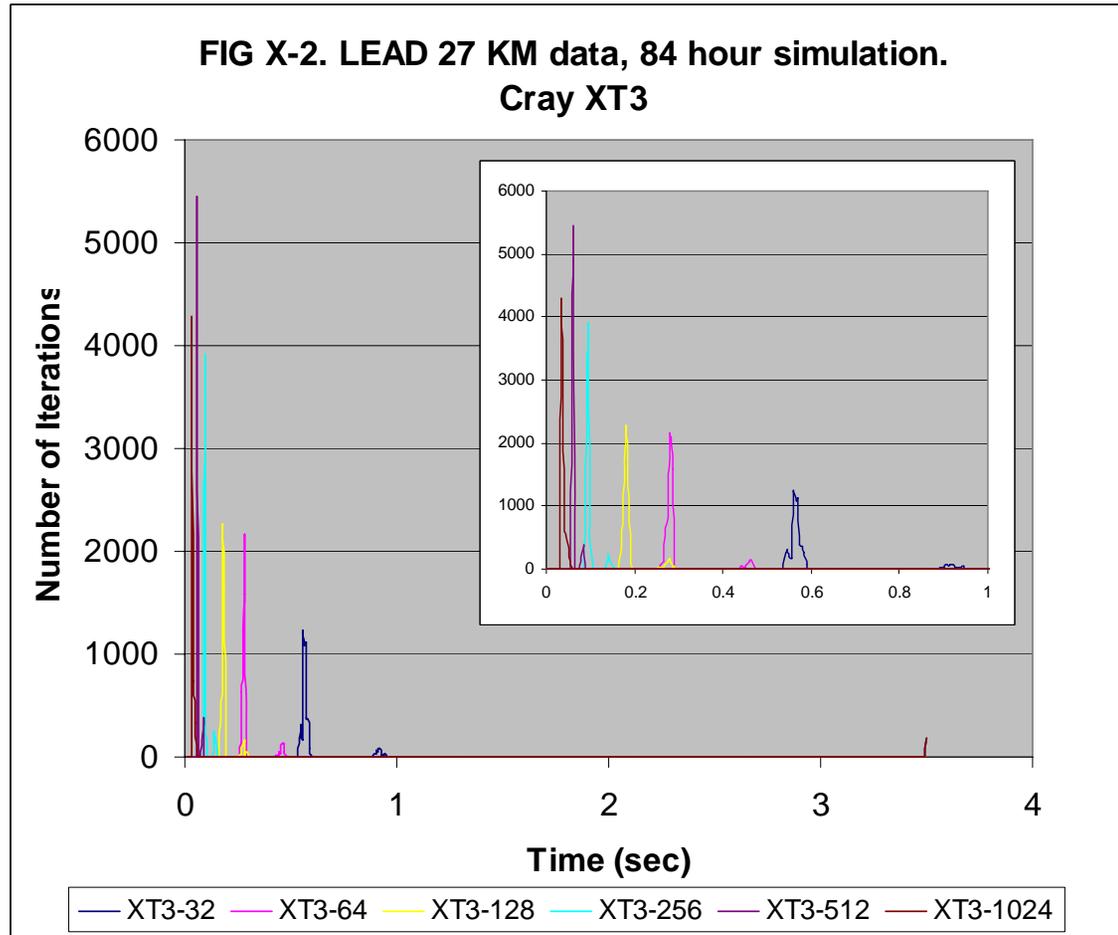
- WRF run for LEAD : CONUS 27km grid, 84 simulated hours, hourly simulation output, checkpoint every 12 simulated hours.
- NetCDF used for I/O.

# Speedup and efficiency of WRF on BG/L

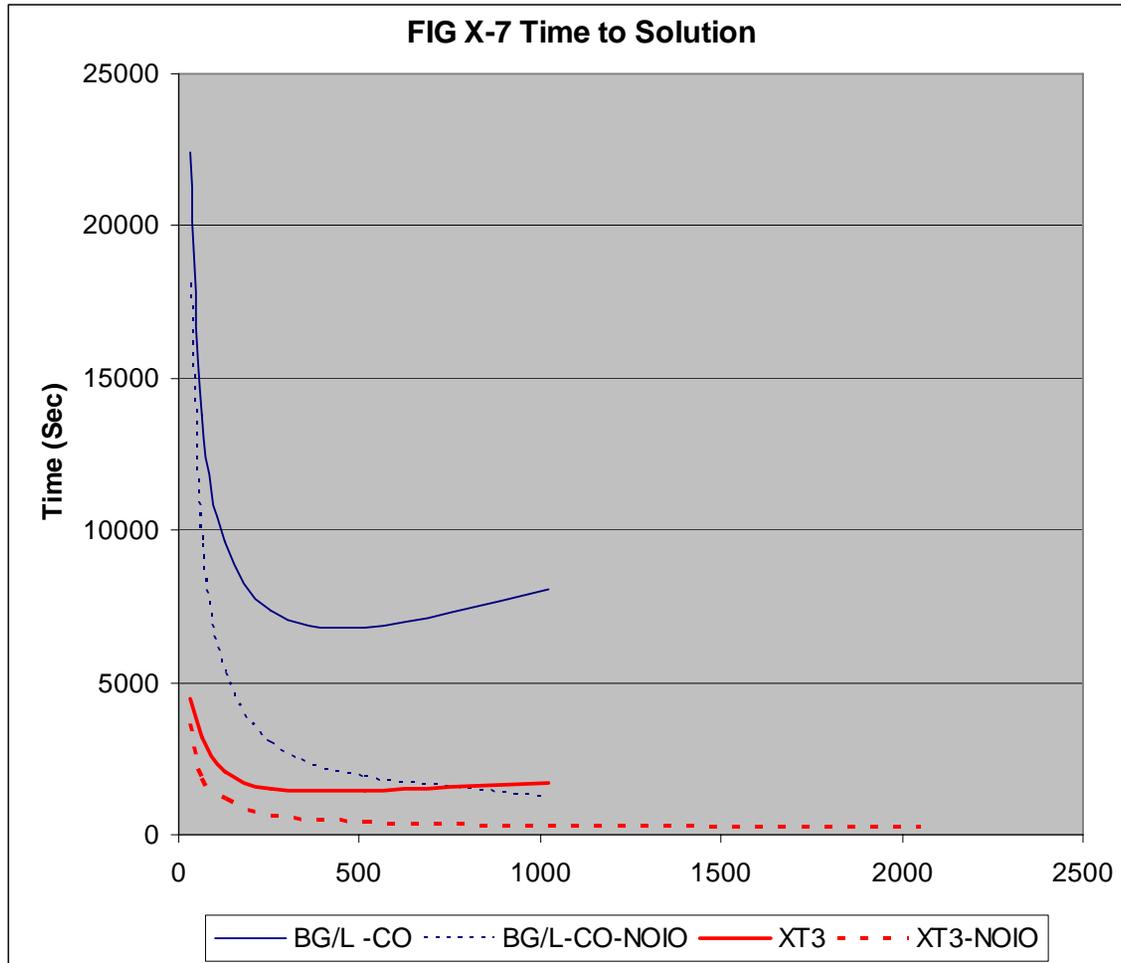


- Four classes of iterations: 1 & 2 just compute, 3 & 4 do file I/O using NetCDF.
- Computation scales reasonably well. I/O does not scale at all.
- What about weak scaling, i.e. run on a “petascale challenge input”:
  - With a bigger problem, the computation will scale better on more processors.
  -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

# WRF on XT3



# Summary of WRF Results.



# Takeaway messages

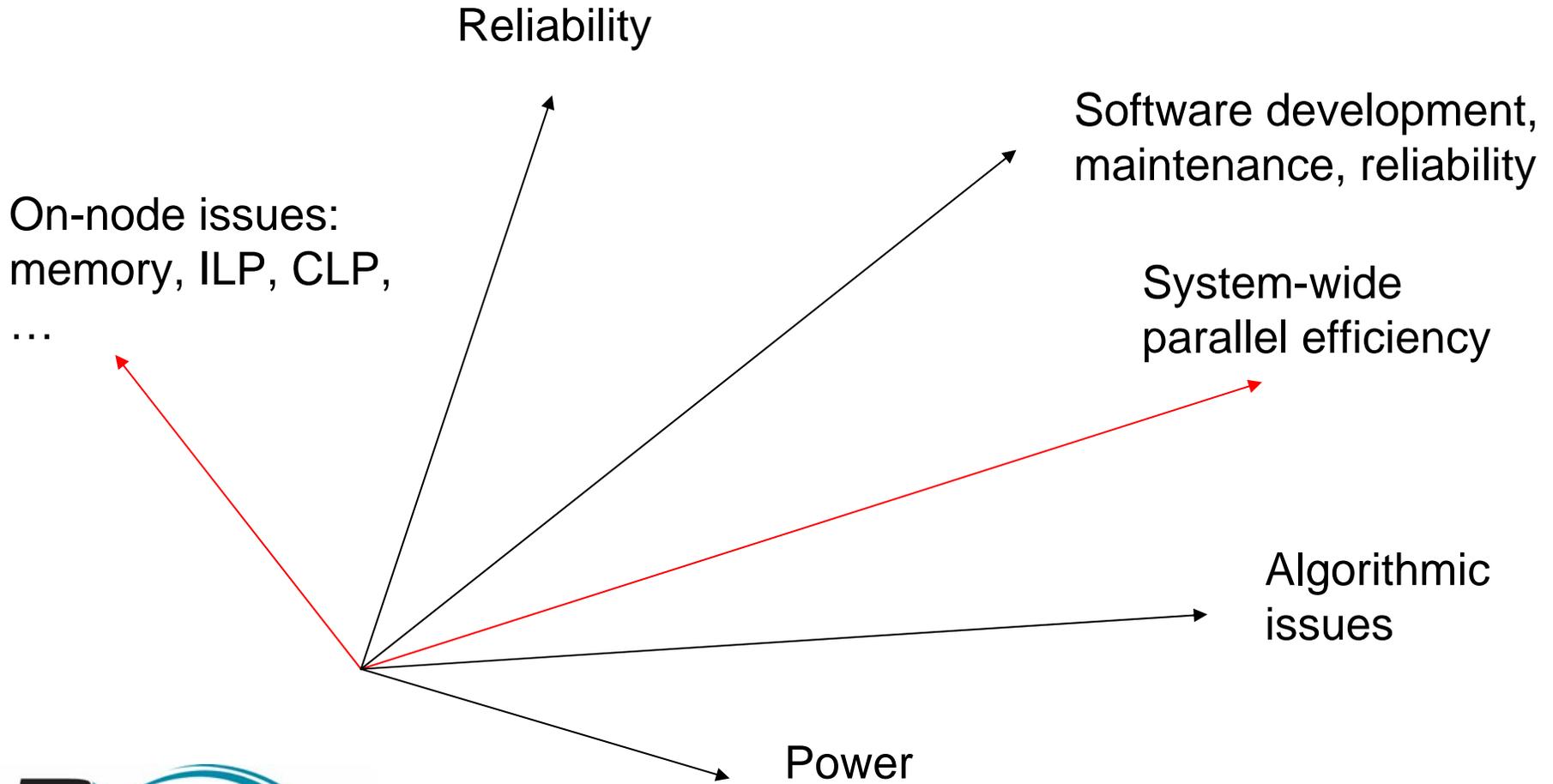
**I/O continues to be the elephant in the room.**

**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.**

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

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



# The HPCToolkit Manifesto

- Focus on real problems: diagnosis for tuning, ...
- Use Event Based Sampling (EBS)
  - Low, controllable overhead (2K-5K samples/sec for <2%)
  - No instrumentation calipers needed/wanted
    - Overhead issue, e.g., >10 M procedure calls/sec
    - Instrumentation adds dependencies, serialization.
  - Collect multiple metrics, compute others.
    - CPI, Bus Utilization, Miss Rates, Lost Cycles
- 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

# Management 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? Plan 9?
- Issue: Does event-based sampling introduce “jitter”?
  - Much less impact than using fine-grain calipers.
  - Not unless you have much worse performance problems.

# Overview

- Preliminaries
- **Post Mortem Sampling (Rice)**
  - **Adam Bordelon, Bradley Broom, R Fowler**
- Adaptive Sampling (RENCI)
- Scalability Diagnosis Using Profiling (Rice)

# Problem: Profiling Parallel Programs

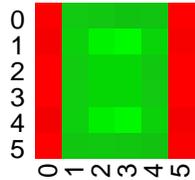
- Sampled profiles can be collected for about 1% overhead.
- How can one productively use profiling on large parallel systems?
  - Understand the performance characteristics of the application.
  - Explain node-to-node variation.
    - Model and understand systematic variation.
      - Characterize intrinsic, 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 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.
- **“Cancer Gene Expression Problem” (Data Mining)**
  - 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

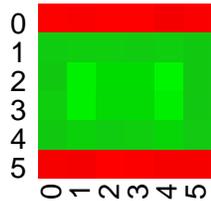
```

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
  
```

# 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

```

# Overview

- Preliminaries
- Post Mortem Sampling (Rice)
- **Adaptive Sampling (RENCI)**
  - Todd Gamblin, Dan Reed, Rob Fowler
- Scalability Diagnosis Using Profiling (Rice)

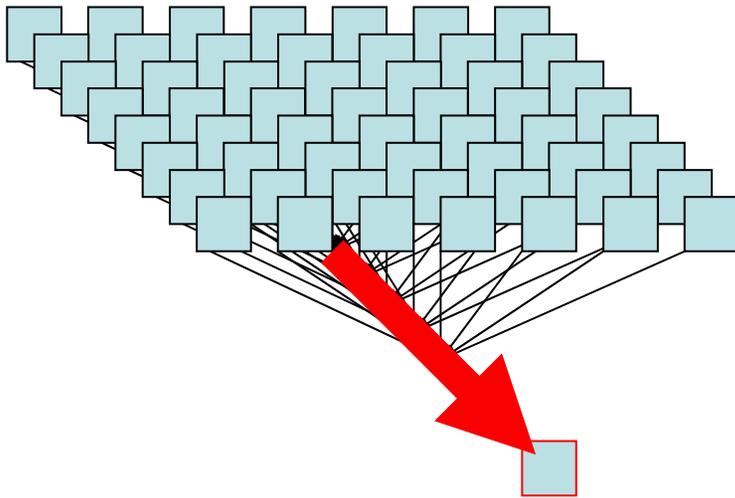
**Problem:** Low-overhead performance monitoring and characterization of large-scale scientific applications

**Recall the Parallel I/O problem**

**Approach:**

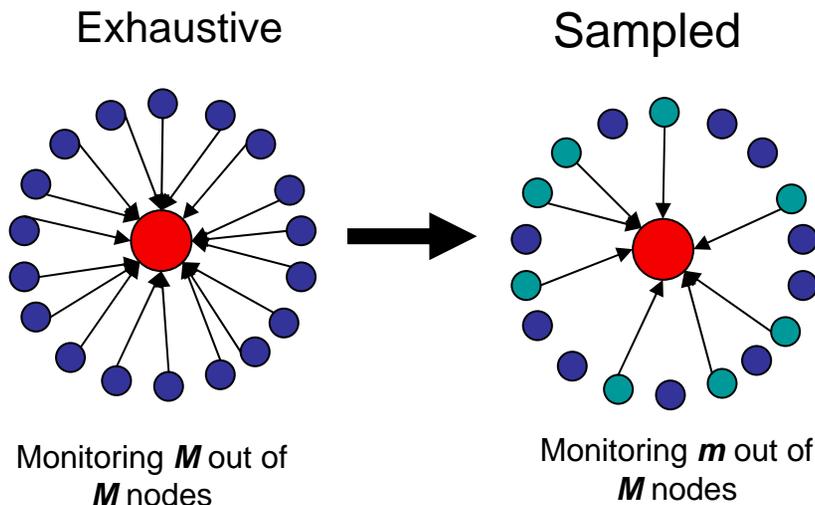
1. Use clustering algorithms to discover behavioral equivalence classes of nodes in scientific applications.
2. Discovered classes provide insight into application behavior across nodes.
3. Variance of monitored data within classes is also low, and we exploit this property to reduce monitoring overhead.

# Adaptive Sampling



- For large clusters, cannot examine performance data from all nodes
  - Centralized collection is impossible at scale!
- We've developed the Adaptive Monitoring and Profiling Library (AMPL)
  - Models parallel application as a population of runtime performance events
  - User specifies desired confidence and error in advance
- Uses population sampling to estimate performance metrics
  - We give a probabilistic guarantee that confidence and error will be met

# Adaptive Sampling

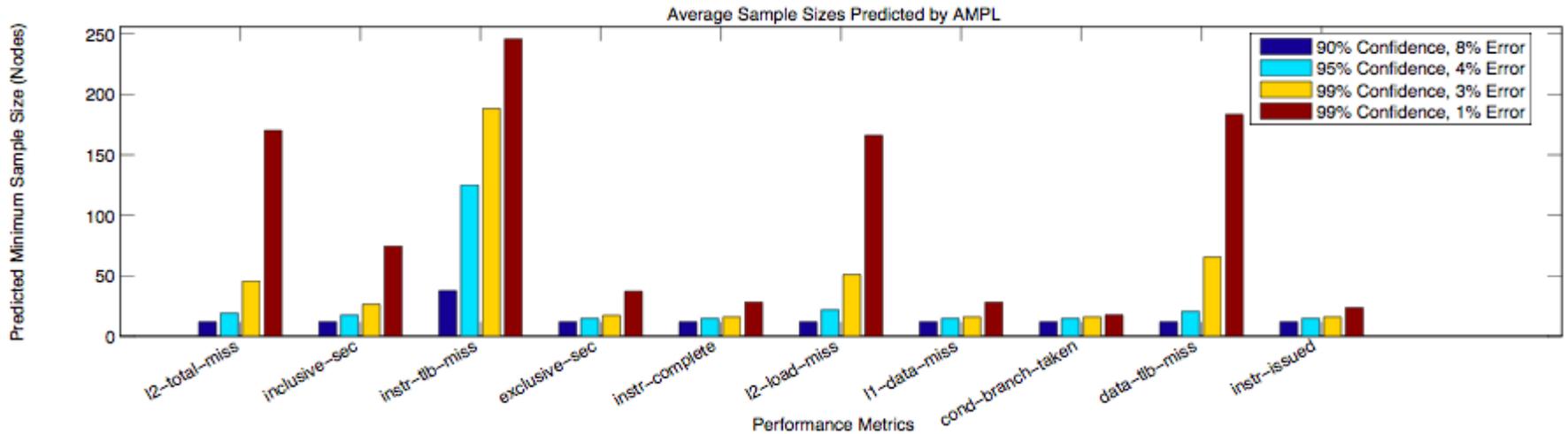


$$m = \frac{Ns_r^2}{N^2V^2 + s_r^2} \quad V = \left(\frac{d}{z_\alpha}\right)^2$$

For  $N$  performance events, error  $d$ , and confidence interval  $z_\alpha$

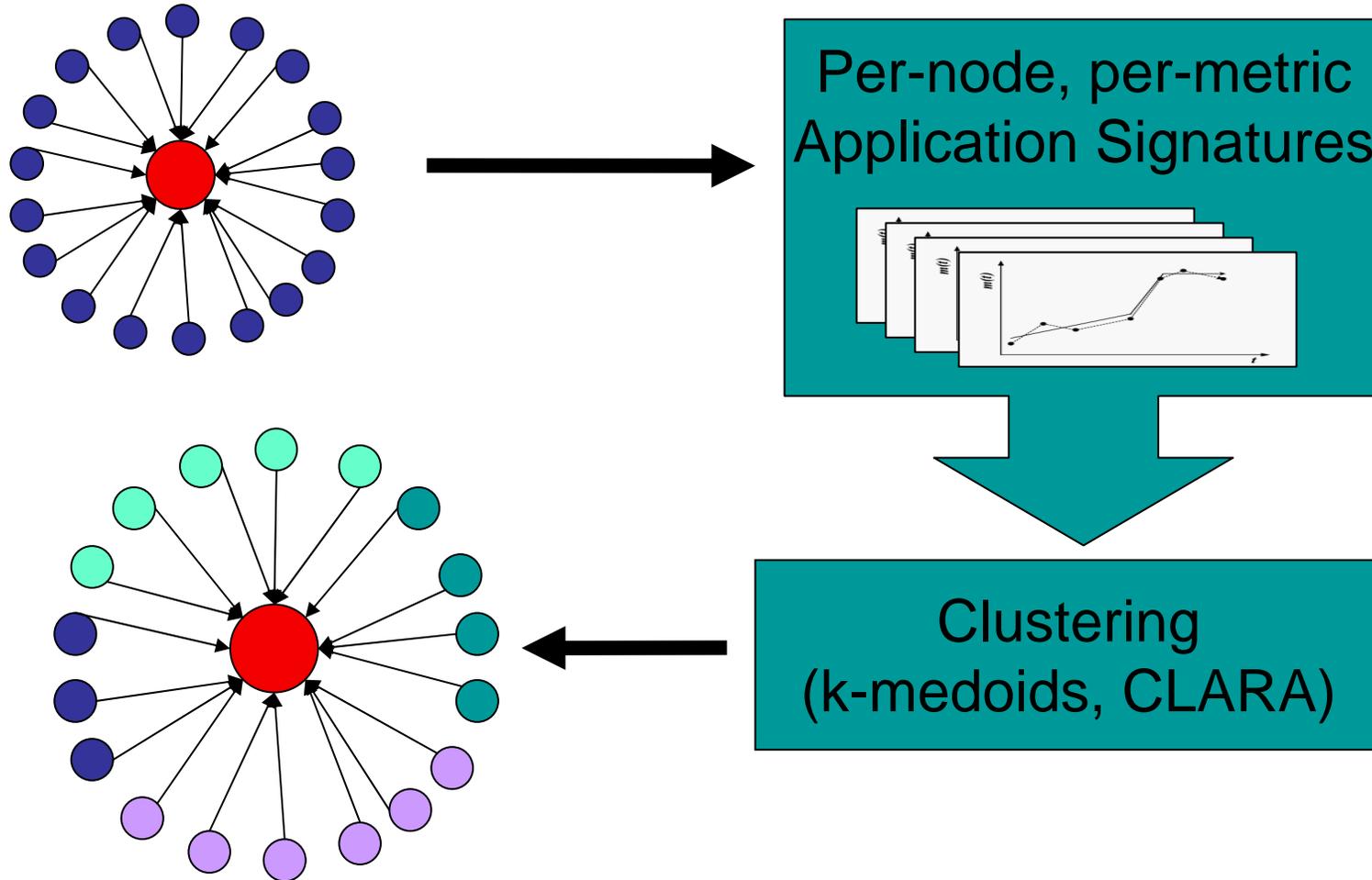
- Given:
  - Desired **confidence**
  - Desired **minimum error**
- Based on variance of collected data:
  - Determine the minimum number of events to sample
- Collect data from just enough nodes that minimum number of events are collected
- Update sample size as monitoring proceeds, depending on variance
  - Updates are done at the end of fixed time windows

# Results with sPPM



- 5-14% of data overhead with PAPI counter (caliper) measurement with sPPM at 90% confidence and 8% accuracy
- 7-14% of overhead at 99% confidence and 1% error for low-variance metrics

# Clustering



Different colored regions  
can be monitored separately

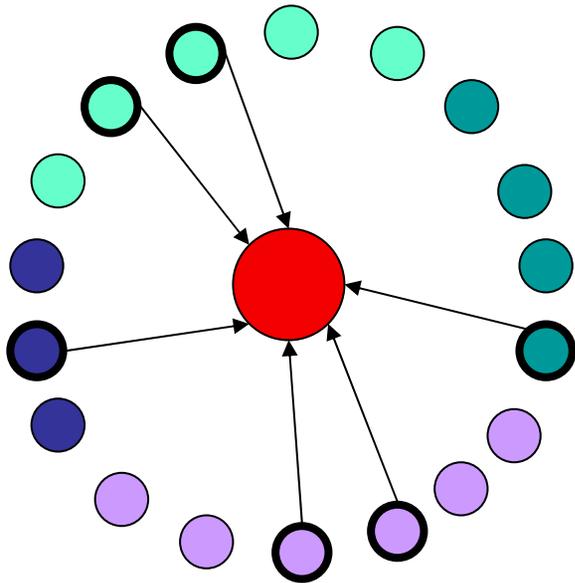
# Guided Stratification

- We know from above that number of nodes sampled depends on variance of monitored data
- Adaptive sampling can be improved by stratifying population
  - Total nodes sampled will be smaller if variance **within** monitored groups is less than variance **between** them:

$$\frac{1}{M} \sum_{i=1}^k (M - M_i) S_i^2 < \sum_{i=1}^k M_i (\bar{Y}_i - \bar{Y})^2$$

- Intuitively, sample size is lower when sum of intra-group variances is smaller than variance of population as a whole

# Guided Stratification

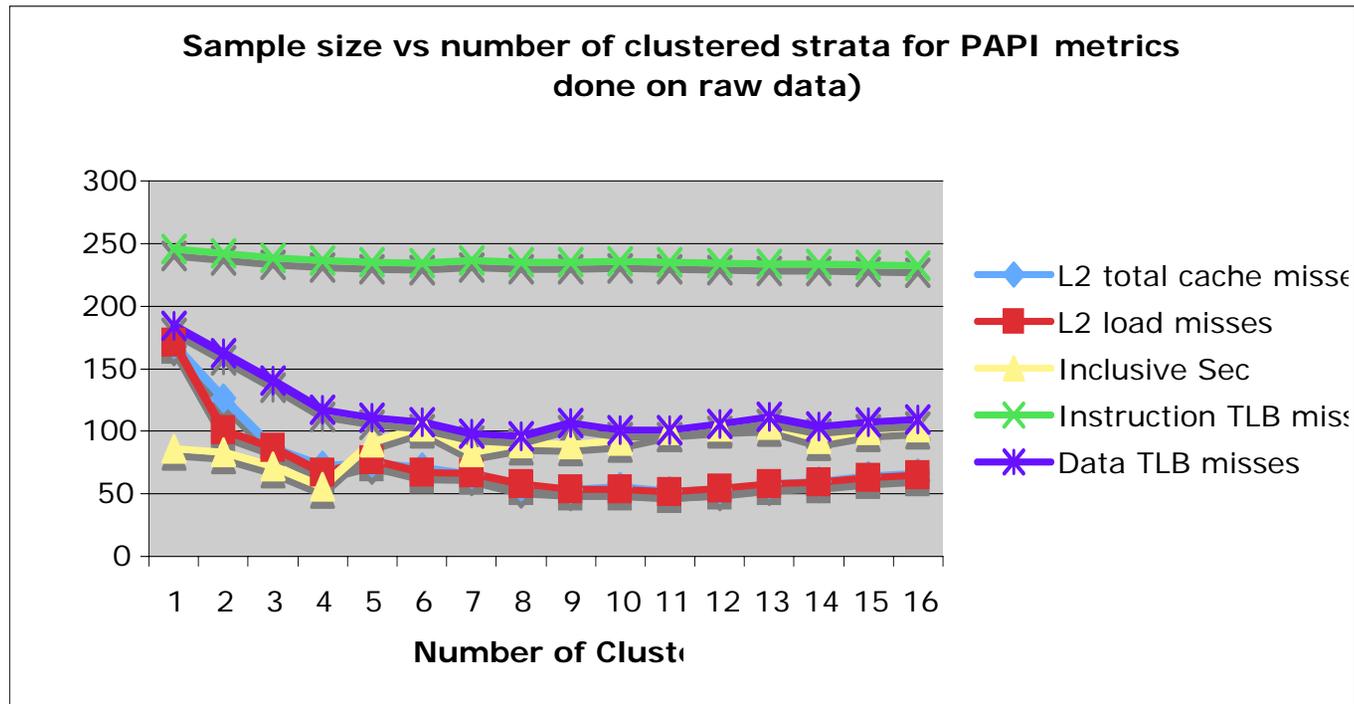


Each discovered equivalence class is monitored separately,  
Total number of monitored nodes is reduced further

Subgroups have lower variance than group as a whole,  
so total number of sampled nodes is decreased.

- **Procedure:**
  - Cluster per-node application signatures
  - Use output of clustering to separate nodes into behavioral equivalence classes
  - Re-run sampling with this new stratification
- **2 benefits:**
  - Adaptively sampling groups separately can *further reduce overhead* in number of nodes sampled
  - Provides a *longitudinal behavioral picture* of an application's performance that isn't possible with other tools

# Monitoring PAPI Metrics in sPPM: Monitored Sample Sizes for 1-16 clusters



- Clustering done on raw data here (exhaustive trace)
- 99% confidence, 1% error

# Work in Progress

- **Extensions of Existing Work:**
  - Precise measurement of overhead and perturbation of applications
  - Tests at larger scale (>1024 nodes)
  - Tests with more applications (ADCIRC, Chombo)
- **Additions:**
  - Clustering on sampled data rather than exhaustive data
  - Dynamic repartitioning of nodes in response to runtime phase changes
  - Learning methods for selecting most relevant metrics to partition on
  - Use a distributed model-checking approach to partition nodes into clusters, to find outliers in existing groups, and to redefine clusters.

# Overview

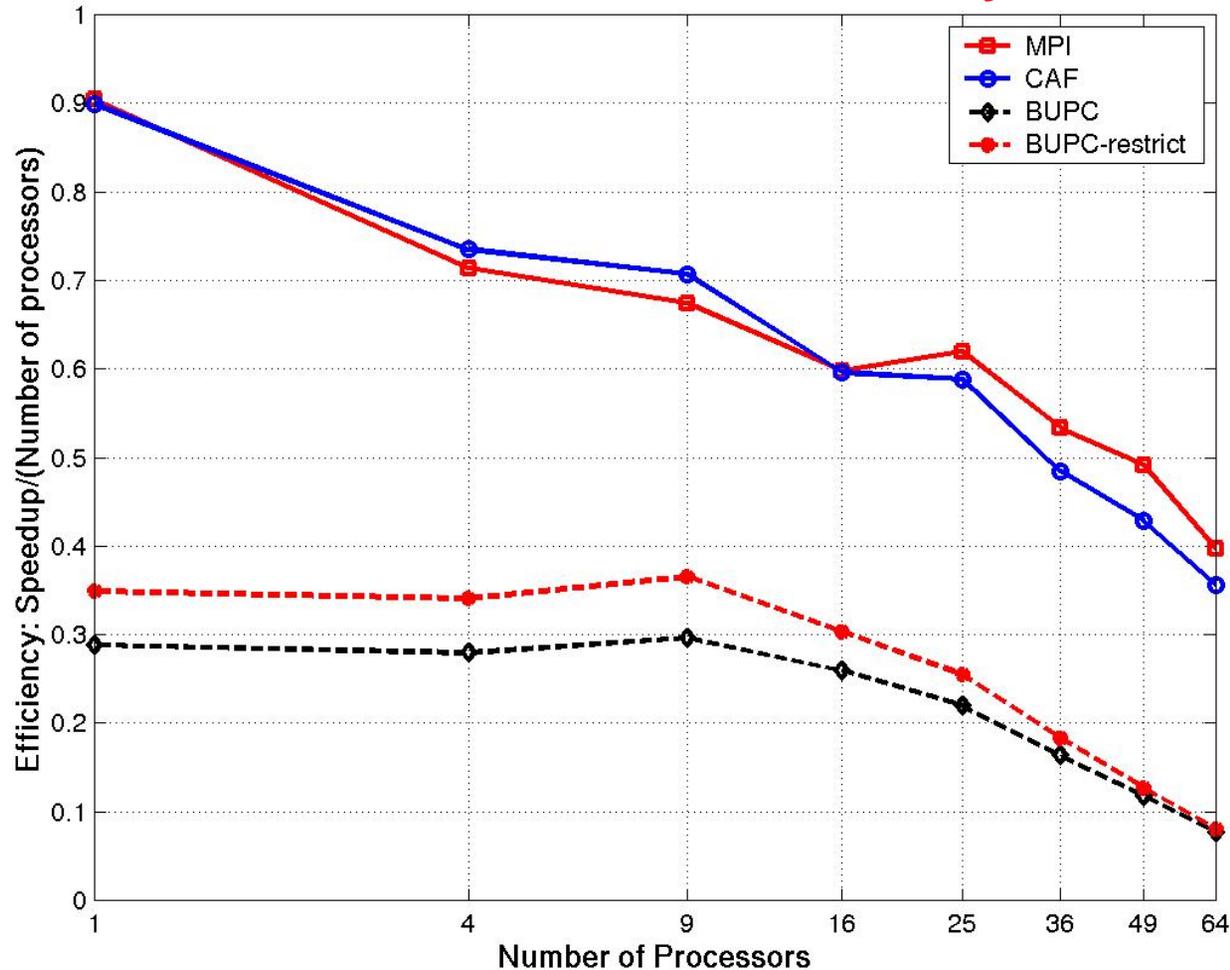
- Preliminaries
- Post Mortem Sampling (Rice)
- Adaptive Sampling (RENCI)
- **Scalability Diagnosis Using Profiling (Rice)**
  - Cristian Coarfa, Nathan Froyd, Fengmei Zhao, John Mellor-Crummey

# Impediments to Scalability

- **Communication overhead**
  - synchronization
  - data movement
- **Load imbalance**
- **Serialization**
- **Algorithmic scaling**
  - e.g. reductions: time increases as  $O(\log P)$
  - partially replicated computation
  - replicated initialization

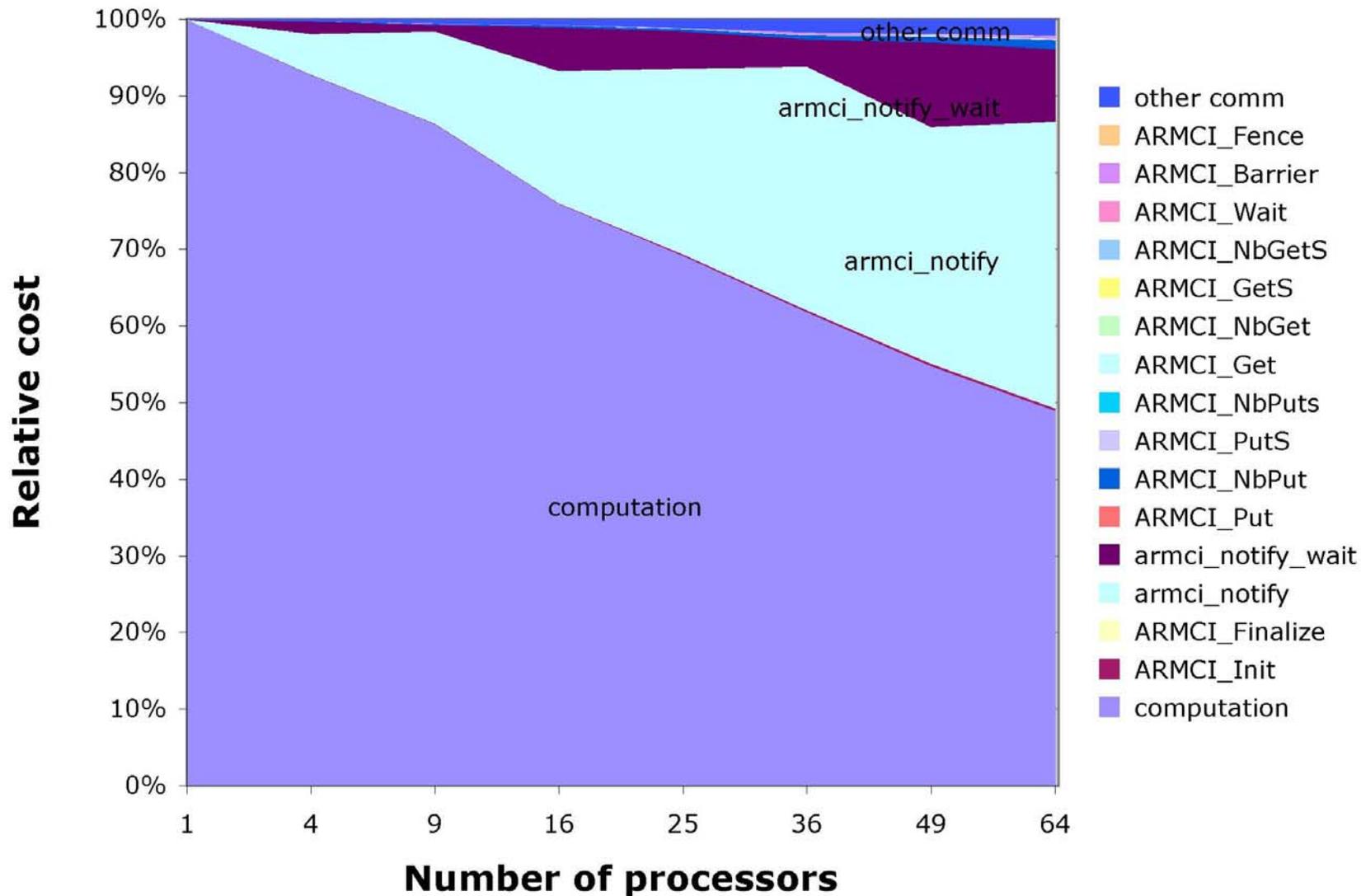
# Motivating Example

CAF NAS SP size  $64^3$  on an Itanium2+Myrinet cluster



# Motivation, Part 2

CAF NAS SP size 64<sup>3</sup> on an Itanium2+Myrinet cluster



# Goal

Automatic technique for pinpointing and quantifying scalability problems

- **What's important: overall impact on running time**
  - not absolute scaling
- **Want a tool that**
  - analyzes scalability
  - guides user to the problems

# Strategy

- **Use call path profiling to measure executions**
  - attribute costs to *full* calling context
  - **method: call stack sampling [Froyd et. al ICS 05]**
    - monitor unmodified fully optimized code
      - language independent C/C++, Fortran, assembly code, ...
    - supports parallel and serial codes
      - CAF, UPC, MPI, ...
    - efficient (1K samples per second ~ 3-5% overhead)
- **Use differential comparisons of profiles to compute a scalability metric for each calling context**

# Two Costs of Interest

- **Exclusive cost: local computation**
- **Inclusive cost: self cost + inclusive cost of callees**

```
subroutine foo
```

```
do i = 1, n      exclusive cost  
    c[i]=a[i]+alpha*b[i]
```

```
end do
```

```
call bar()
```

**inclusive cost**

```
call baz()
```

```
end subroutine
```

# How to compare profiles

## Performance Analysis with Expectations

- **Users have performance expectations for codes**
  - **strong scaling: linear speedup**
  - **weak scaling: constant execution time**
  - **sequential codes: scale w.r.t. inputs**
    - e.g. compiler analysis should scale  $\sim$ linearly w/ program size
- **Putting expectations to work**
  - **measure performance under different conditions**
    - e.g. different levels of parallelism or different inputs
  - **define our expectations**
  - **compute the deviation from expectations for each calling context**
    - for both inclusive and exclusive costs
  - **correlate the metrics with the source code**
  - **explore the annotated call tree interactively**

# Relative Scalability

- Parallel program P
- Two experiments:
  - $E_p$  on  $p$  CPUs
  - $E_q$  on  $q$  CPUs
- Total execution times  $T_p$  and  $T_q$
- Consider  $m$  a node in the call tree
- $C_p(m)$ ,  $C_q(m)$  costs of  $m$  on  $p$ ,  $q$  CPUs

- **Expectation:** 
$$C_q(m) = \frac{p}{q} C_p(m)$$

- **Fraction of excess work :** 
$$EW(m) = \frac{q \times C_q(m) - p \times C_p(m)}{q \times T_q}$$

# Ensemble of Parallel Experiments

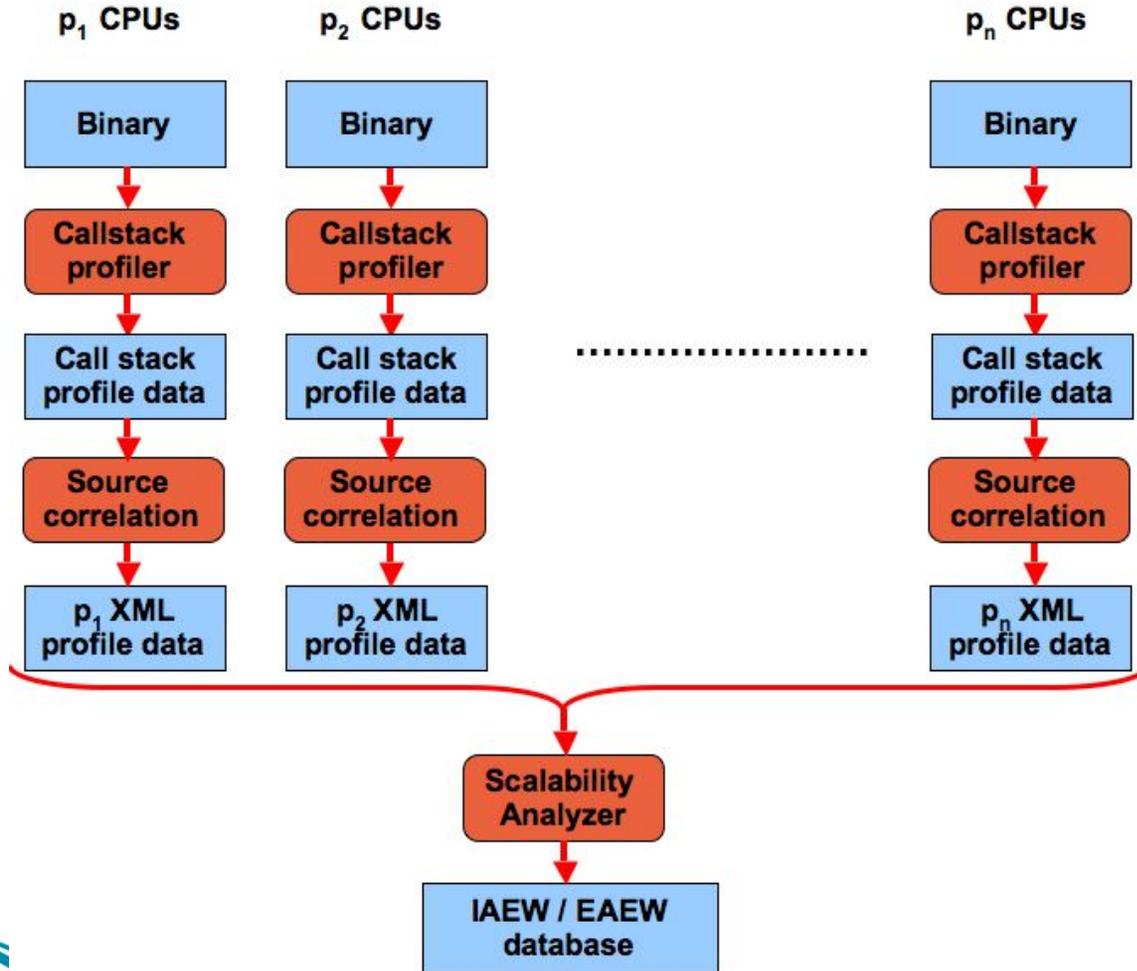
- Program P
- Ensemble of experiments  $E_i$  on  $p_i$  CPUs,  $i=1..n$
- $T_i$ - total execution time of experiment  $E_i$
- For a node  $m$  in the call tree
  - $C_{p_i}(m)$  cost of  $m$  on  $p_i$  processors

- **Expectation:** 
$$C_{p_i}(m) = \frac{p_1}{p_i} C_{p_1}(m)$$

- **Fraction of excess work:** 
$$EW(m) = \frac{\sum_{i=1}^n (p_i \times C_{p_i}(m) - p_1 \times C_{p_1}(m))}{\sum_{i=2}^n p_i \times T_i}$$

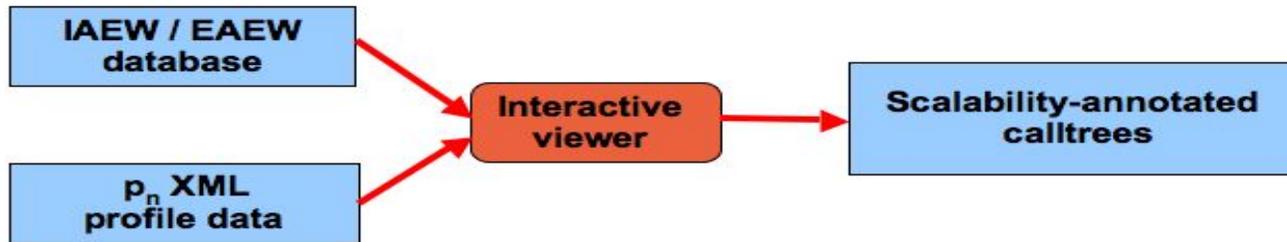
# Strong Scaling Analysis

## Measurement and analysis of calling contexts

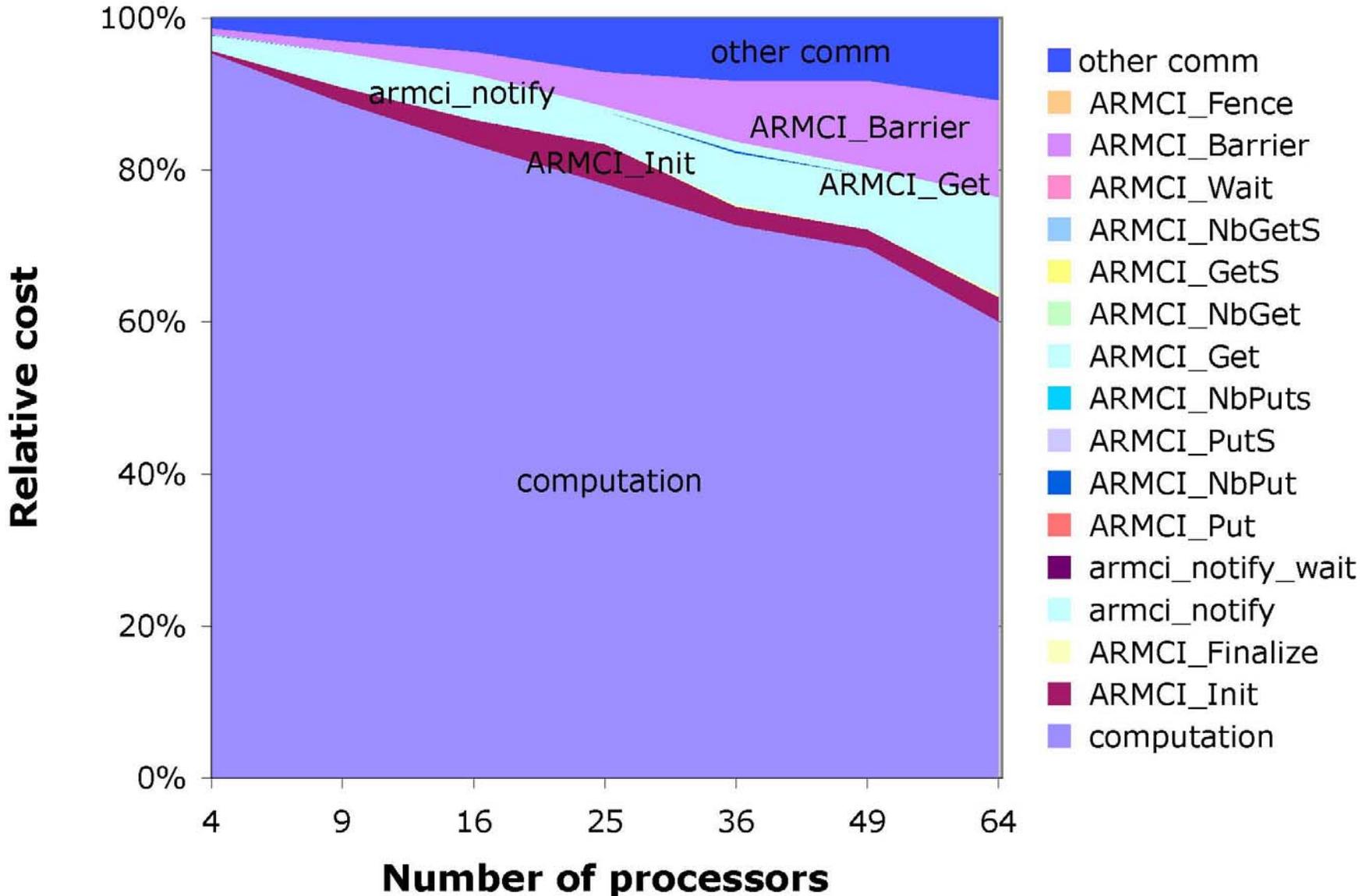


# Strong Scaling Analysis

## Exploration of annotated call tree



# LBMHD Size 1024<sup>2</sup>



users/ccristi/Research/cc-caf-experiments/bin/mhd-caf-0111

File

mhd.cafctmp.w2f.f

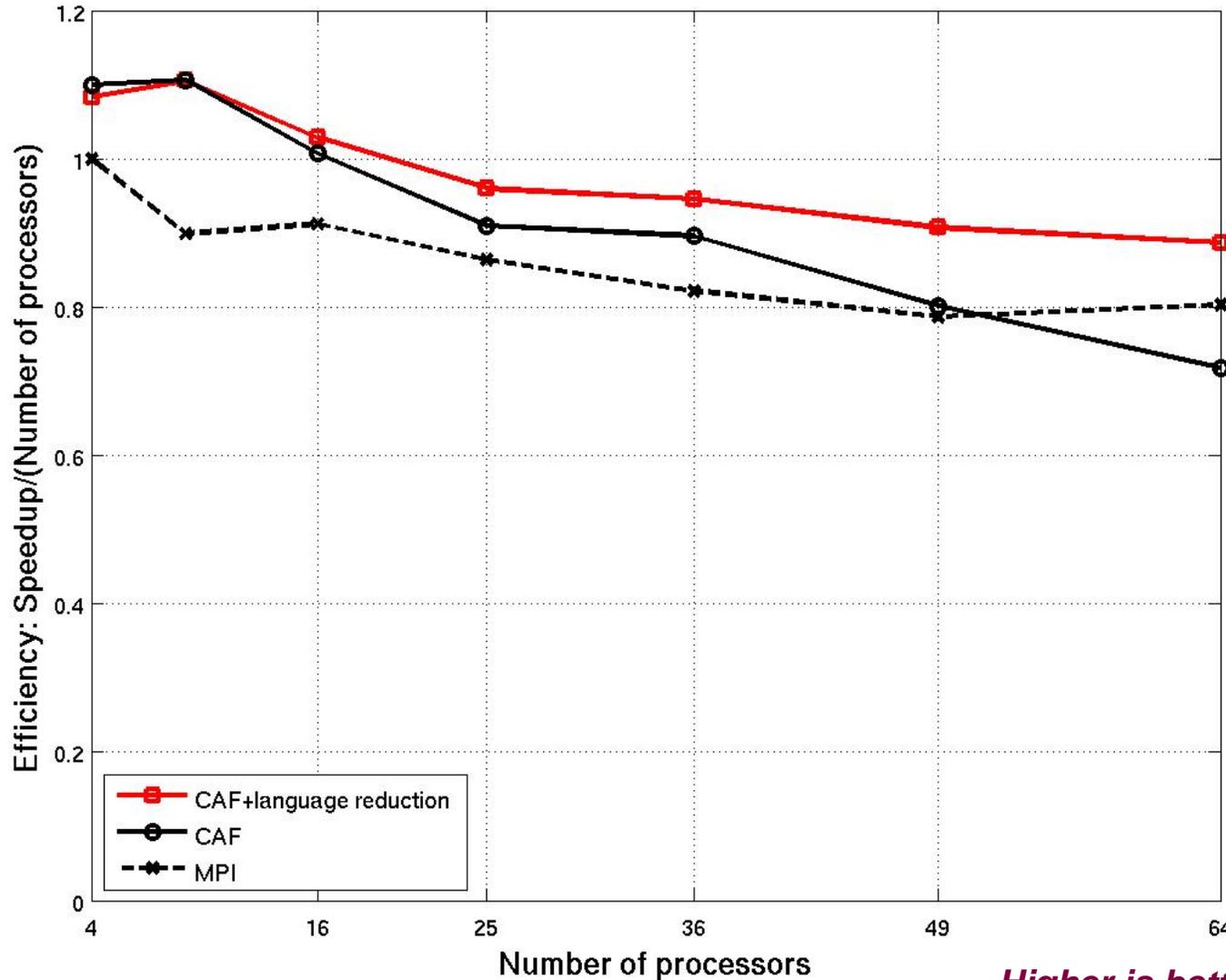
152

153 PROGRAM mhd

Scopes

	# samples	IAEW	EAEW
mhd	3.76e05 100.0	0.32e00	-0.00e00
mhd.cafctmp.w2f.f: 285	5.55e04 14.7%	0.08e00	0.00e00
decomp	5.55e04 14.7%	0.08e00	0.08e00
mhd.cafctmp.w2f.f: 319	7.65e04 20.3%	0.06e00	0.00e00
stream	7.65e04 20.3%	0.06e00	0.00e00
mhd.cafctmp.w2f.f: 242	3.90e04 10.4%	0.06e00	0.00e00
cafinit_	3.90e04 10.4%	0.06e00	0.00e00
mhd.cafctmp.w2f.f: 1599	2.25e04 6.0%	0.03e00	0.00e00
caf_allsum_dp	2.25e04 6.0%	0.03e00	0.00e00
mhd.cafctmp.w2f.f: 1719	1.20e04 3.2%	0.02e00	0.00e00
cafsynchall_	1.20e04 3.2%	0.02e00	0.00e00
CommunicationInterface.cc: 427	1.20e04 3.2%	0.02e00	0.00e00
ARMCICommunicationInterface::cafSynchAll()	1.20e04 3.2%	0.02e00	0.00e00
ARMCICommunicationInterface.cc: 1712	1.20e04 3.2%	0.02e00	0.00e00
ARMCI_Barrier	1.20e04 3.2%	0.02e00	0.00e00
mhd.cafctmp.w2f.f: 1730	4.50e03 1.2%	0.01e00	0.00e00
mhd.cafctmp.w2f.f: 1670	4.50e03 1.2%	0.00e00	0.00e00
mhd.cafctmp.w2f.f: 1727	1.50e03 0.4%	0.00e00	0.00e00
mhd.cafctmp.w2f.f: 244	7.50e03 2.0%	0.03e00	0.00e00
cafglobalstartupinit_	7.50e03 2.0%	0.03e00	0.00e00
mhd.cafctmp.w2f.f: 1601	2.10e04 5.6%	0.03e00	0.00e00
caf_allsum_dp	2.10e04 5.6%	0.03e00	0.00e00
mhd.cafctmp.w2f.f: 1719	1.05e04 2.8%	0.02e00	0.00e00
cafsynchall_	1.05e04 2.8%	0.02e00	0.00e00
mhd.cafctmp.w2f.f: 1730	7.50e03 2.0%	0.01e00	0.00e00
mhd.cafctmp.w2f.f: 1670	3.00e03 0.8%	0.00e00	0.00e00

# LBMHD size 1024<sup>2</sup>



efficient  
reductions  
yield  
25% gain  
on 64 CPUs

*Higher is better*



# Conclusion: Scalability Analysis through Differential Profiling

- Scalability analysis with expectations
  - automatically diagnose scalability impediments
  - independent of programming model
  - fraction of excess work scalability metric
    - dimensionless metric with ubiquitous applicability
  - attribution to calling context enables precise diagnosis of bottlenecks
- Future plans: use expectations to explore
  - scalability issues with MPI-2
  - weak scaling
  - algorithmic scaling w.r.t. various input parameters

# Summary

- **Multiple, related efforts at Rice and RENCI**
  - **Efficient on-node measurement with EBS**
  - **Exhaustive post mortem analyses**
    - Biclustering using HPM and derived data
    - Differential profiling for scalability analysis
  - **Incremental analyses**
    - Stratified sampling for data collection
    - Distributed model-checking for cluster identification

# Contact Information

Rob Fowler  
[rjf@renci.org](mailto:rjf@renci.org), [rjf@unc.edu](mailto:rjf@unc.edu)  
919 445 9670

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



THE UNIVERSITY  
of NORTH CAROLINA  
at CHAPEL HILL



LACSI

