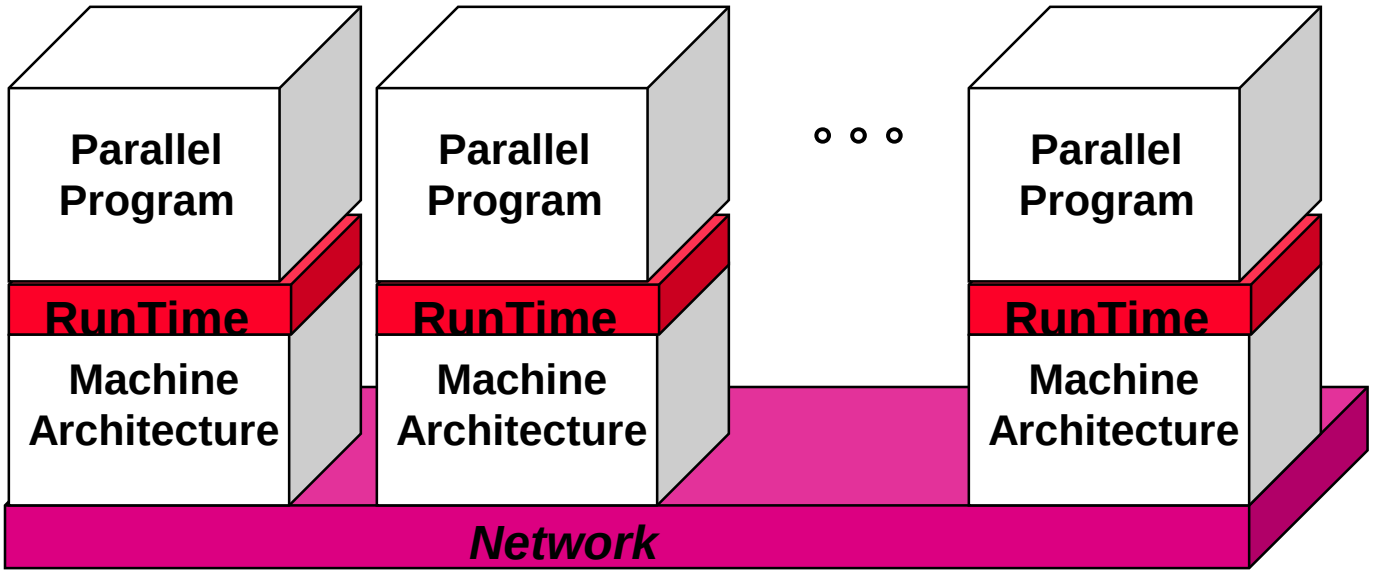
# Architectural Interactions in High Performance Clusters

**RTPP 98**

**David E. Culler**
**Computer Science Division**
**University of California, Berkeley**

# Run-Time Framework

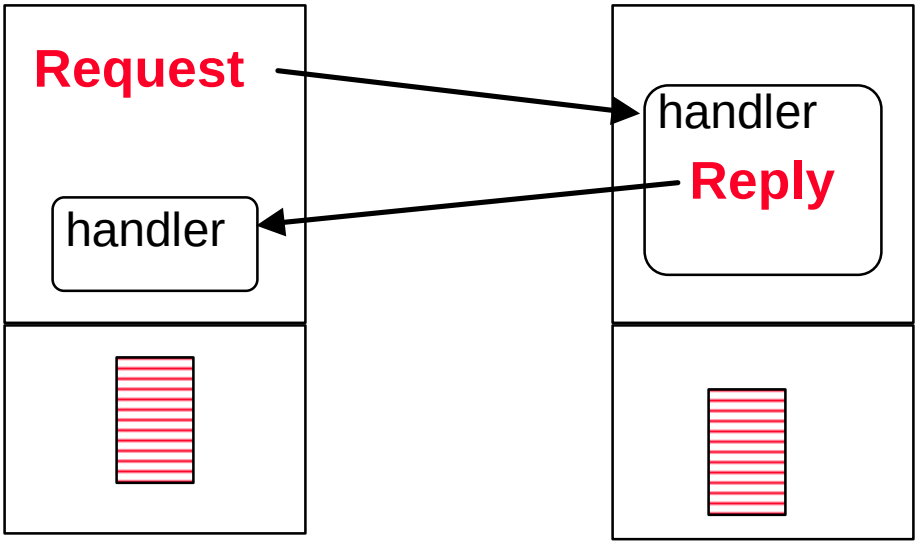


# Two Example RunTime Layers

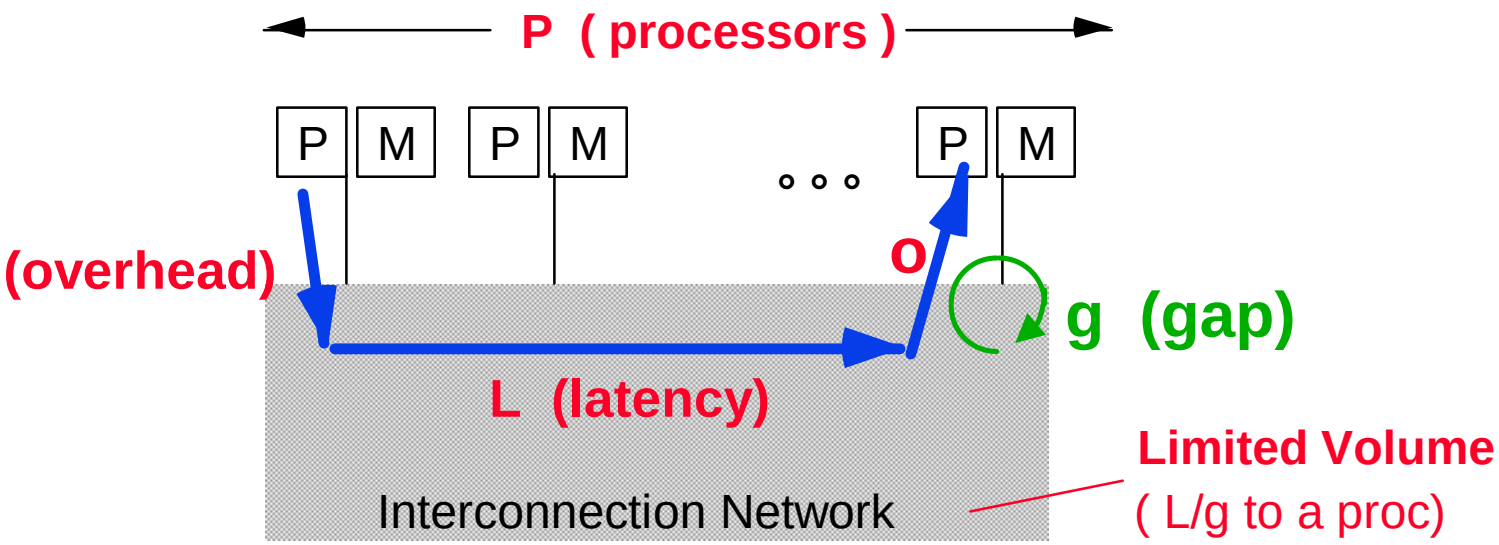
- **Split-C**
  - **thin global address space abstraction over Active Messages**
  - **get, put, read, write**
- **MPI**
  - **thicker message passing abstraction over Active Messages**
  - **send, receive**

# Split-C over Active Messages

- **Read, Write, Get, Put built on small Active Message request / reply (RPC)**
- **Bulk-Transfer (store & get)**



# Model Framework: LogP




**Latency in sending a (small) message between modules**

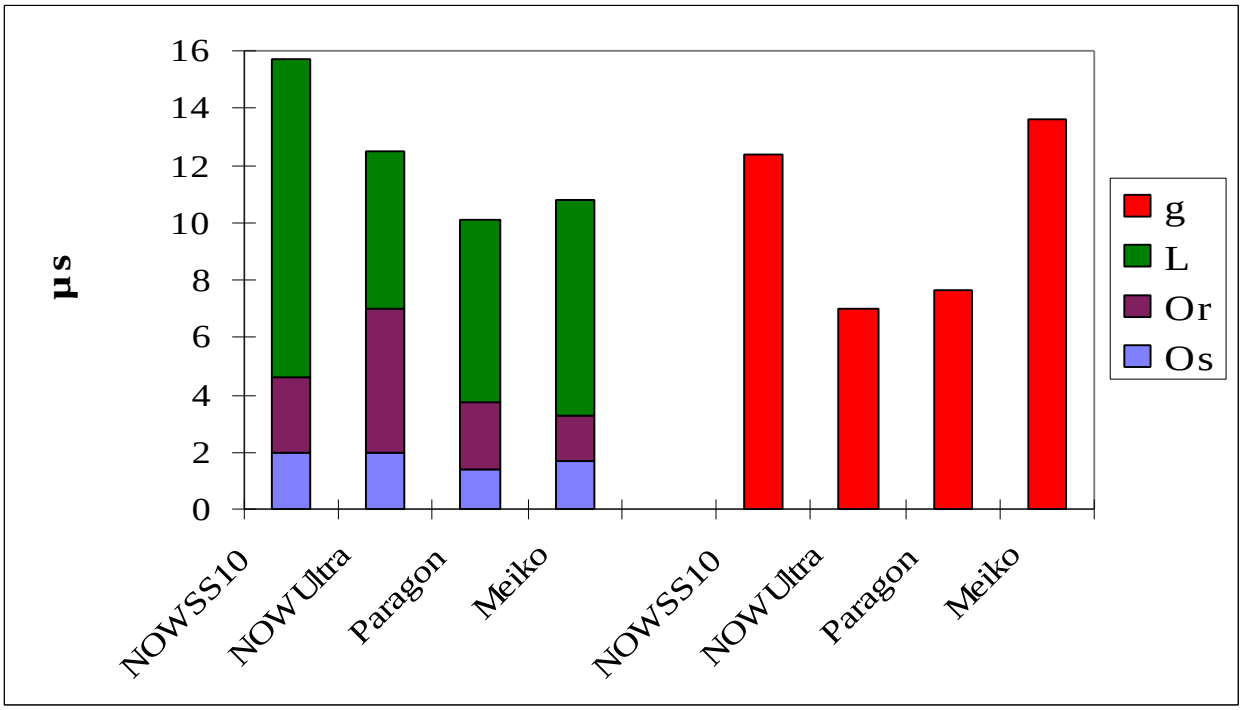
**overhead felt by the processor on sending or receiving msg**

**gap between successive sends or receives (1/rate**

**Processors**

Round Trip time: 2 x ( 2o + L)

# LogP Summary of Current Machines

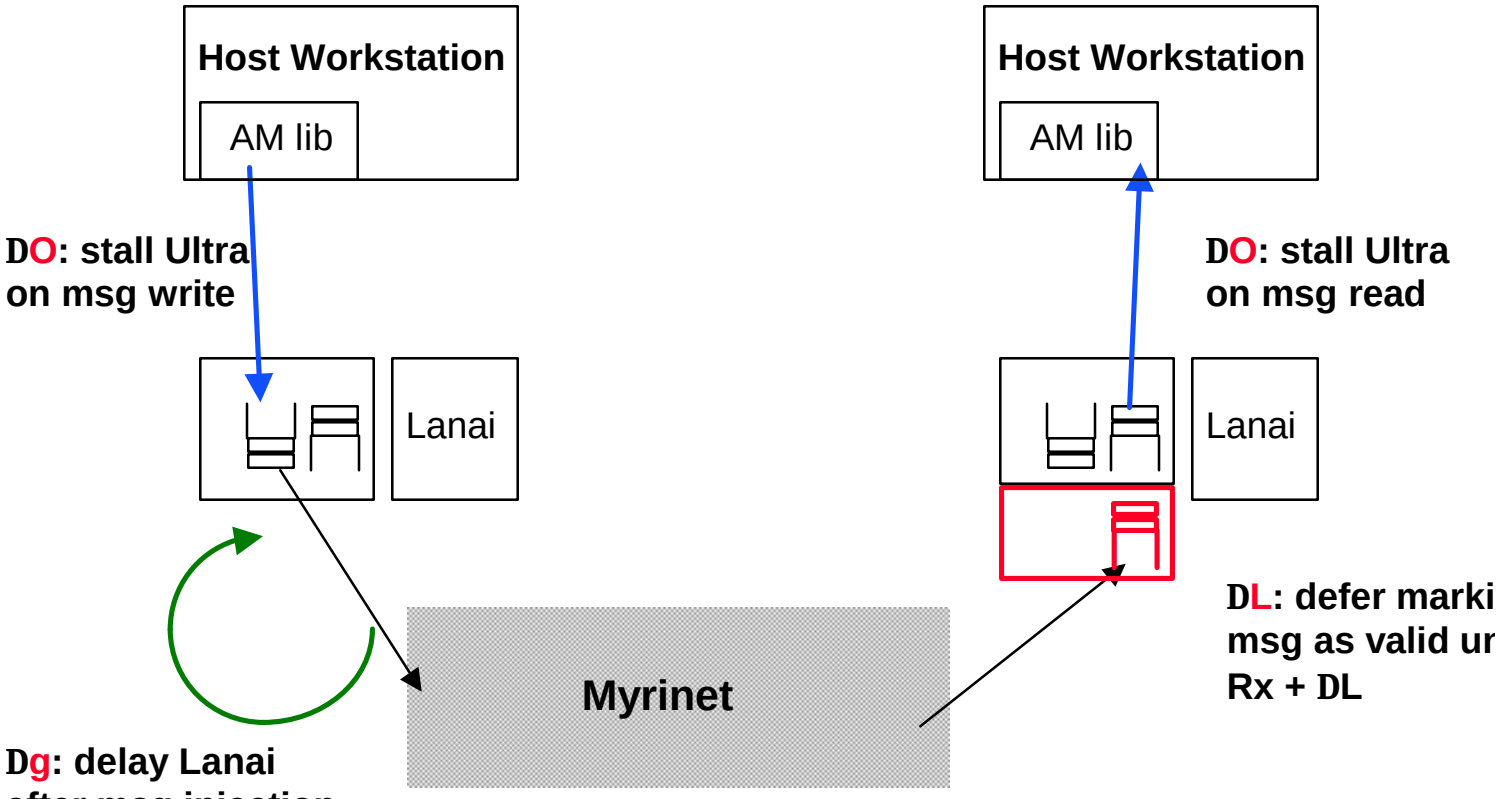



Max MB/s: 38 141 47

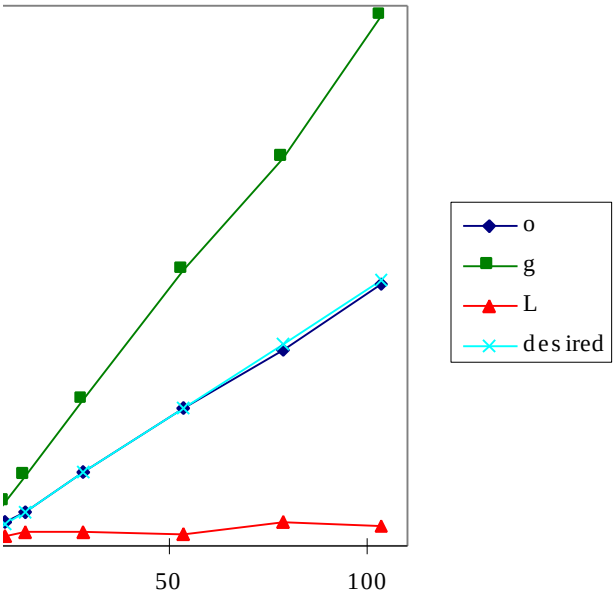
# Methodology

- **Apparatus:**
  - **35 Ultra 170s (64 MB, .5 MB L2, Solaris 2.5)**
  - **M2F Lanai + Myricom Net in Fat Tree variant**
  - **GAM + Split-C**
- **Modify the Active Message layer to inflate L, o, g, or G independently**
- **Execute a diverse suite of applications and observe effect**
- **Evaluate against natural performance models**

# Adjusting L, o, and g (and G) *in situ*

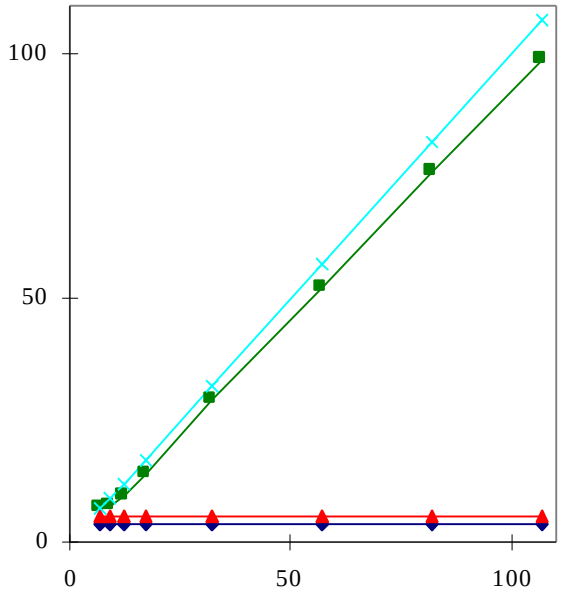


# Calibration

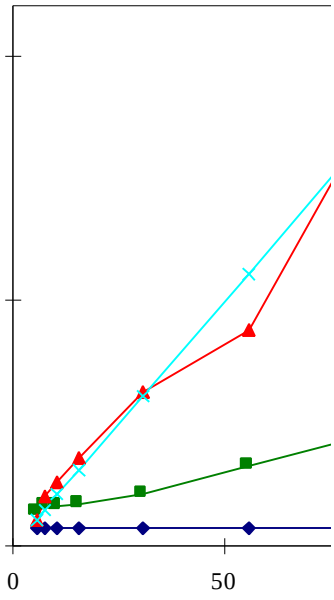



O (desired)




g (desired)




L (desired

# Applications Characteristics

- **Message Frequency**
- **Write-based vs. Read-based**
- **Short vs. Bulk Messages**
- **Synchronization**
- **Communication Balance**

# Applications used in the Study

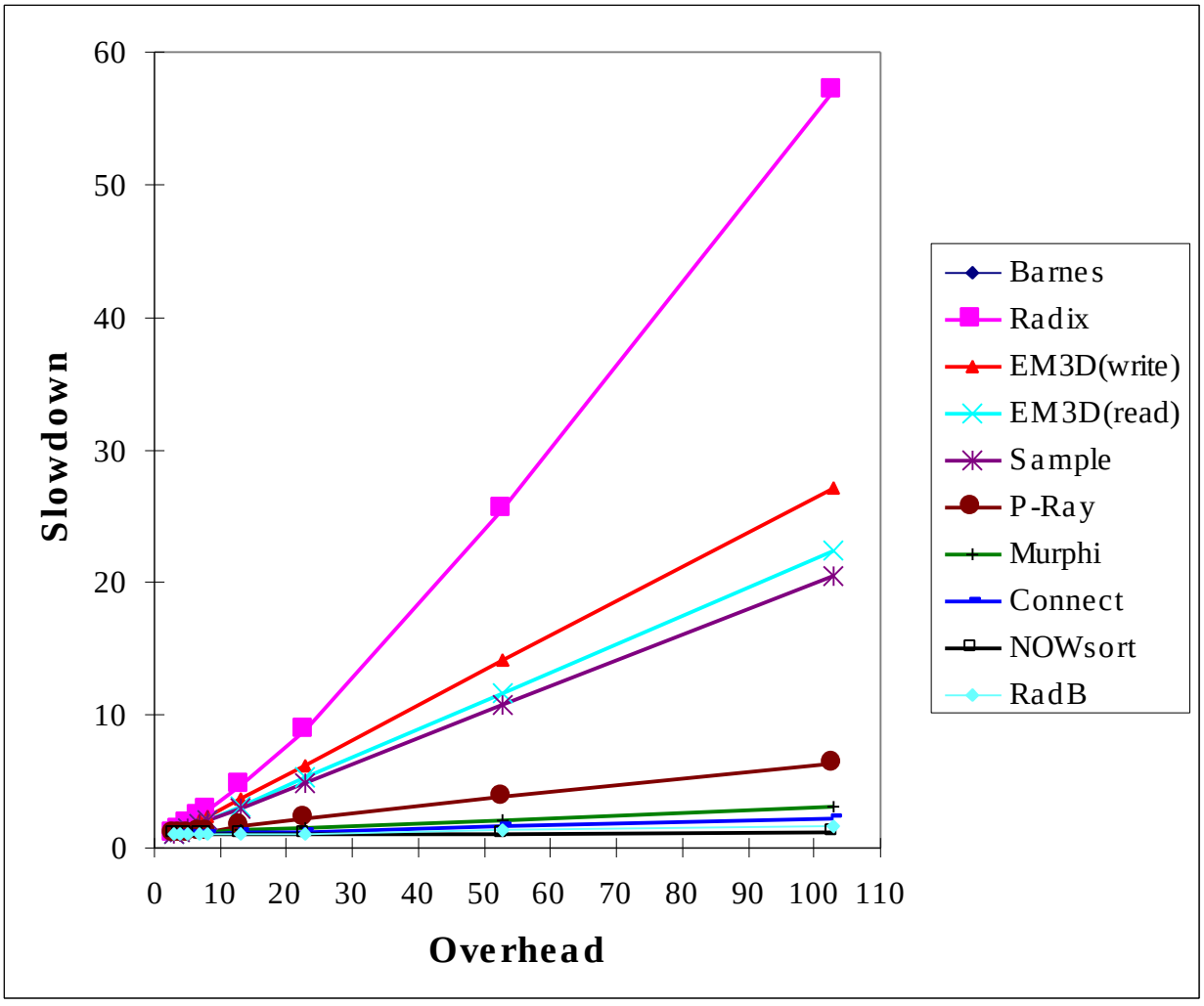
| | Description | Input | Time(16) | Time(32) | Ms |
|---|---|---|---|---|---|
| | Int Radix Sort | 16 M 32-bit keys | 17.0 s | 9.8 s | |
| | Int Sample Sort | 32 M 32-bit keys | 101 s | 44.3 s | |
| | EM Wave Prop. | 8 K nodes, deg. 10, 100 steps | 24.3 s | 15.9 s | |
| | Hierarchical N-body | 1 M bodies | 81.2 s | 46.6 s | |
| | Ray Tracer | 1 M pixel image, 16 k objs | 25.8 s | 17.8 s | |
| | Protocol Verifier | SCI, 2 procs, 1 line, 1 mem | 71.3 s | 37.9 s | |
| | Connected Components | 4 M nodes, 2D, 30 % | 2.5 s | 1.45 s | |
| | Bulk Radix | 16 M 32-bit keys | 6.5 s | 3.95 s | |

# Baseline Communication

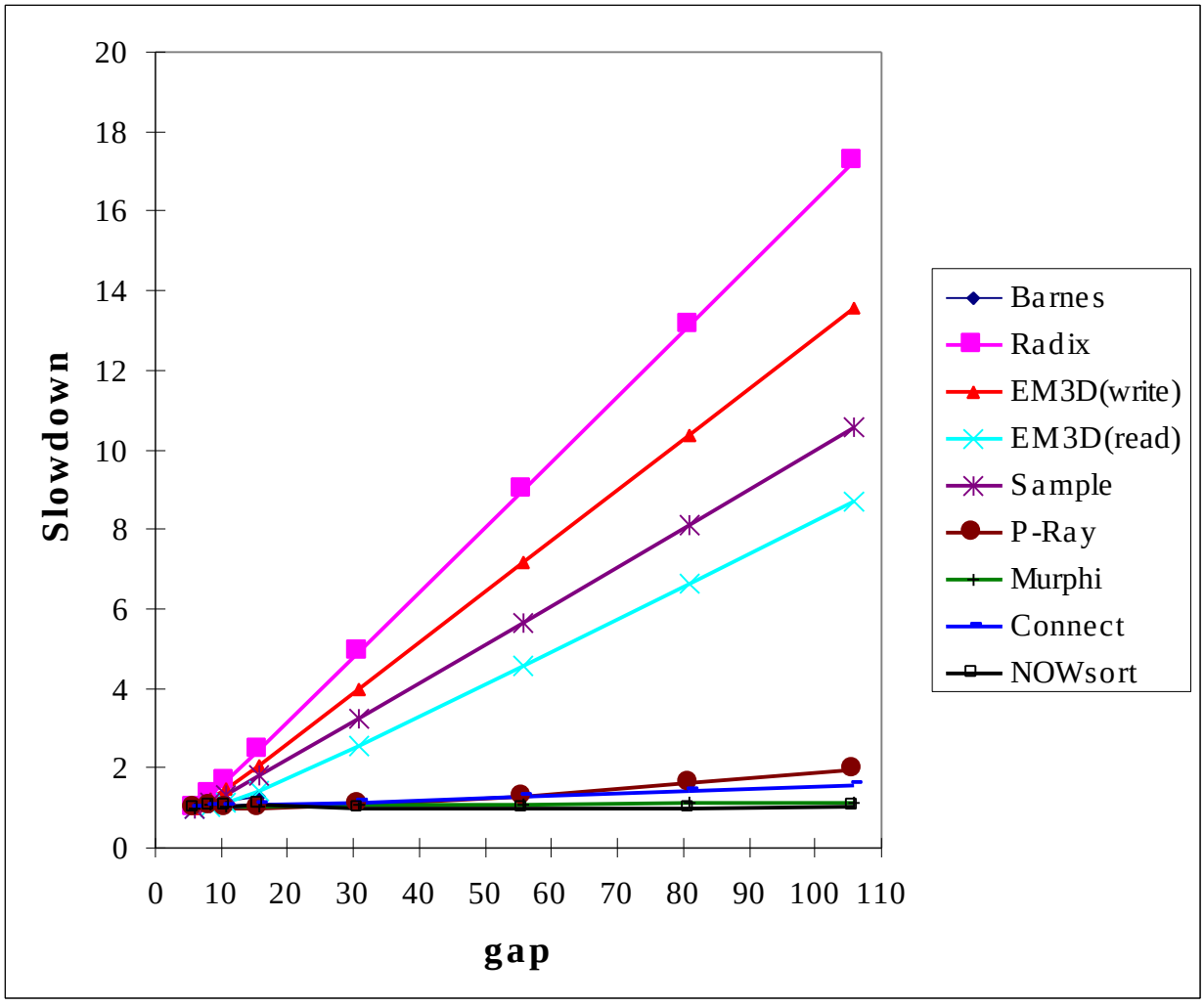
| Program | µs/msg | ms/Barrier | Avg Msg/Proc | Max Msg./Proc | Reads | Bulk Msg |
|---|---|---|---|---|---|---|
| radix | 7.6 | 895 | 2,228,364 | 2,229,106 | 0.0% | 0.0% |
| em3d | 10.2 | 324 | 9,953,384 | 9,974,265 | 0.0% | 0.0% |
| sample | 14.0 | 2499 | 1,966,199 | 2,319,362 | 0.0% | 0.0% |
| ebarnes | 60.1 | 233 | 1,351,194 | 1,400,601 | 9.6% | 23.5% |
| p-ray | 108.1 | 1465 | 216,869 | 353,640 | 47.7% | 47.9% |
| murphi | 219.4 | 36054 | 328,699 | 332,054 | 0.0% | 51.2% |
| connect | 282.9 | 101 | 8,912 | 9,221 | 34.1% | 0.1% |
| radb | 1260.0 | 80 | 5,520 | 5,927 | 0.0% | 43.7% |

# Application Sensitivity to Communication Performance

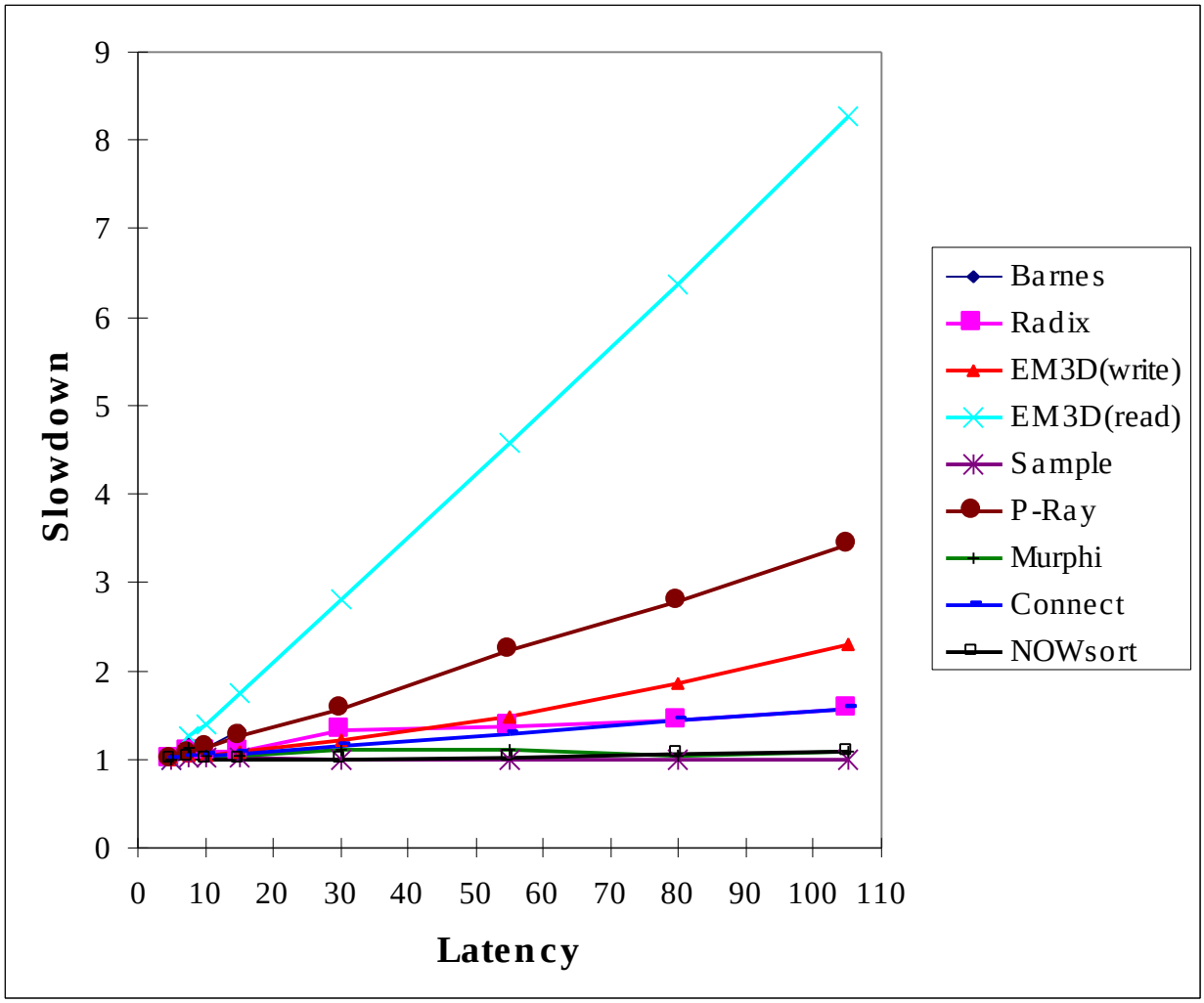
# Sensitivity to Overhead



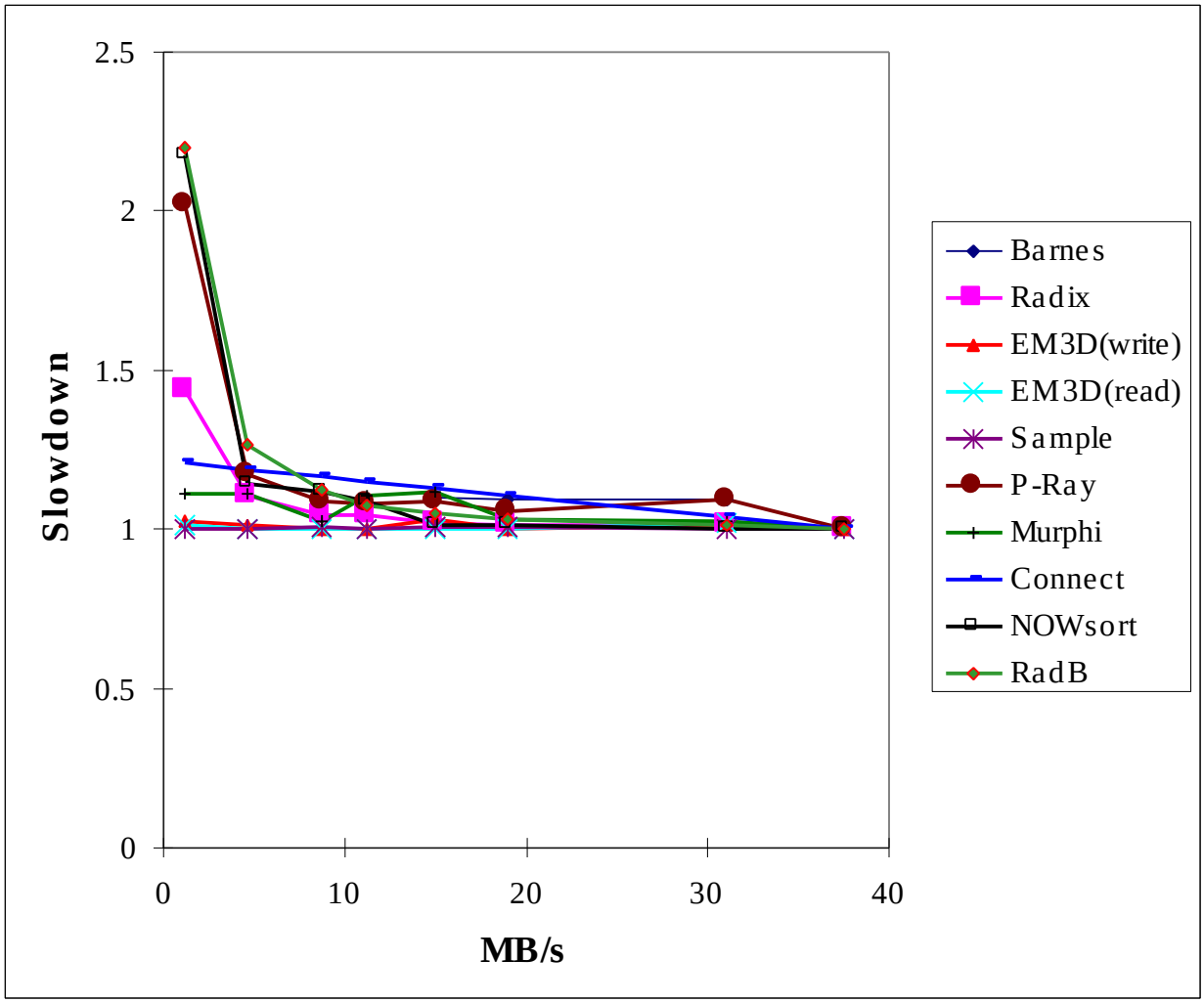
# Sensitivity to gap (1/msg rate)



# Sensitivity to Latency



# Sensitivity to bulk BW (1/G)



# Modeling Effects of Overhead

- $T_{pred} = T_{orig} + 2 \times \max \#msgs \times Do$
  - request / response
  - proc with most msgs limits overall time

| verhead (µs) | radix | sample | em3d | ebarnes | p-ray | connect | murphi | radb |
|---|---|---|---|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1 |
| 1 | 1.03 | 1.00 | 1.14 | 1.02 | 1.03 | 1.02 | 1.00 | 1 |
| 2 | 1.02 | 1.00 | 1.26 | 1.00 | 1.04 | 1.04 | 1.02 | 1 |
| 4 | 1.02 | 1.00 | 1.50 | 0.90 | 1.07 | 1.07 | 1.04 | 1 |
| 5 | 1.01 | 0.99 | 1.59 | 0.88 | 1.06 | 1.12 | 1.03 | 1 |
| 10 | 1.01 | 0.99 | 2.01 | 4.83 | 1.21 | 1.30 | 1.09 | ( |
| 20 | 1.02 | 0.98 | 2.53 | N/A | 1.31 | 1.32 | 1.16 | ( |
| 50 | 1.03 | 0.98 | 3.22 | N/A | 1.50 | 1.60 | 1.30 | 1 |
| 100 | 1.04 | 0.98 | 3.61 | N/A | 1.58 | 1.85 | 1.46 | 1 |

- Why does this model under-predict?

# Modeling Effects of gap

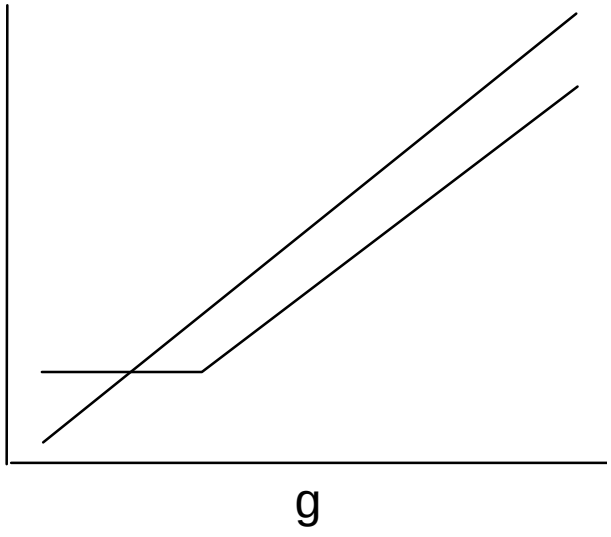
- **Uniform communication model**

$T_{pred} = T_{orig}$ , if g < *l*, average msg interv

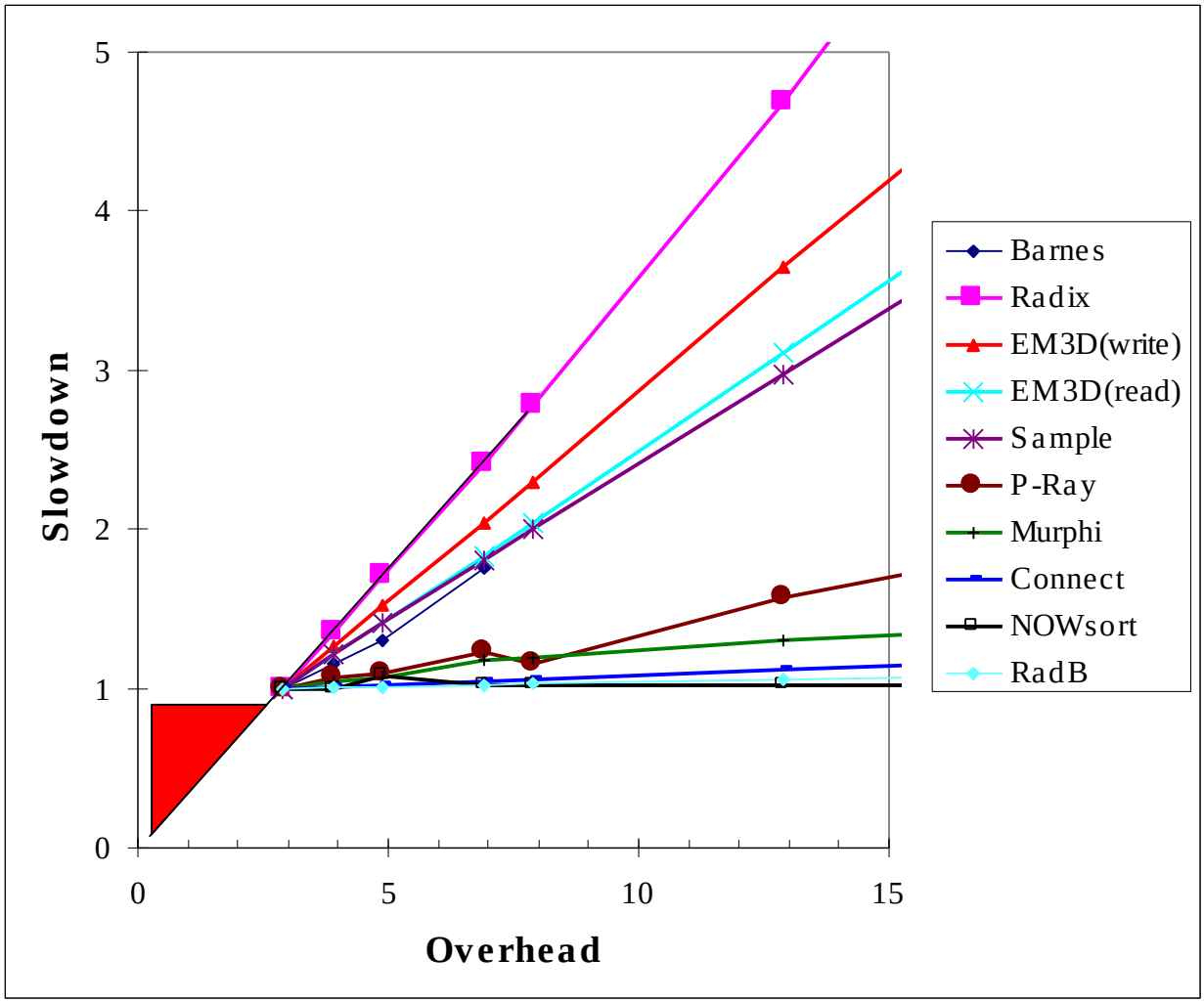
$= T_{orig} + m (g - l)$, otherwise

- **Bursty Communication**

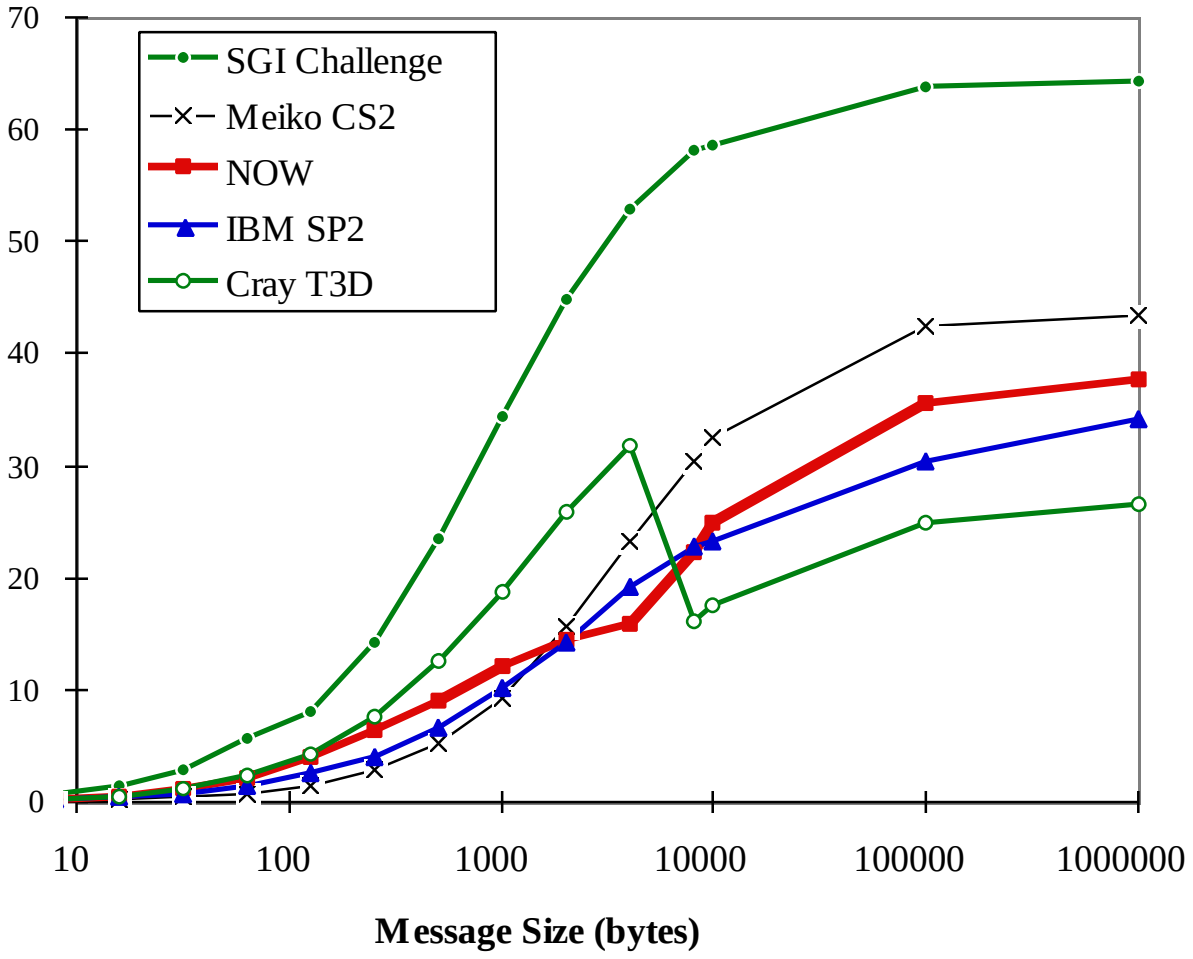
$T_{pred} = T_{orig} + m \, Dg$



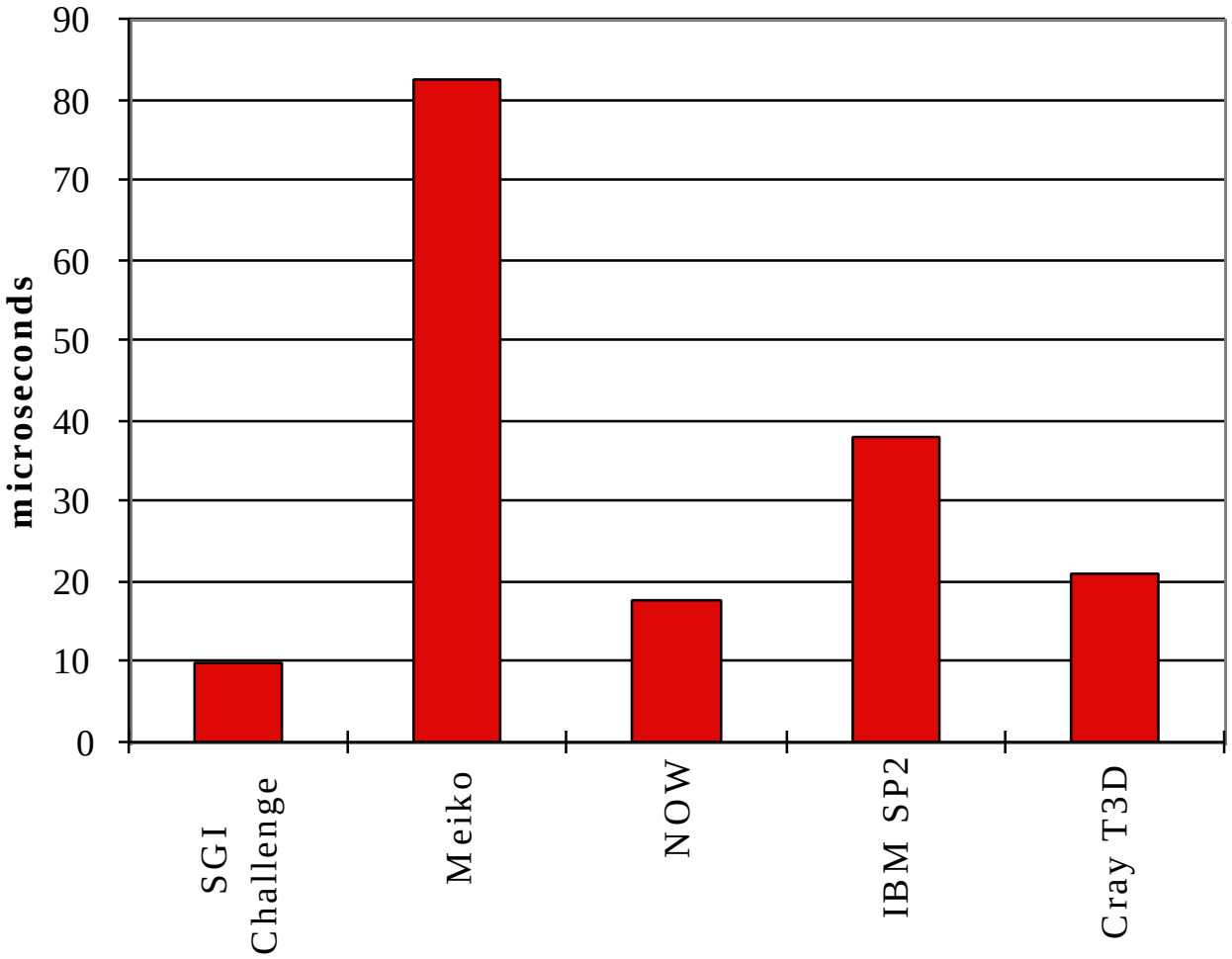
# Extrapolating to Low Overhead



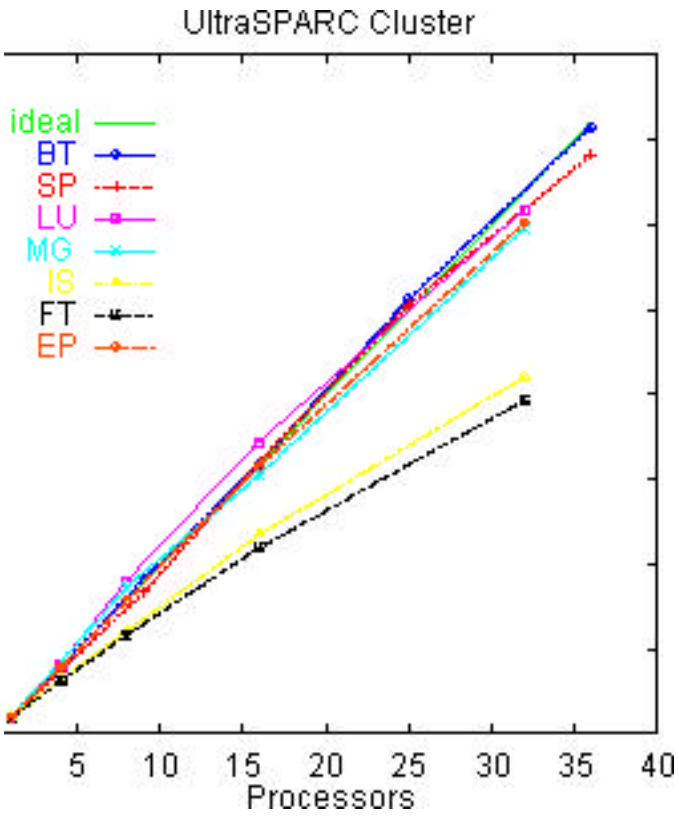
# MPI over AM: ping-pong bandwidth

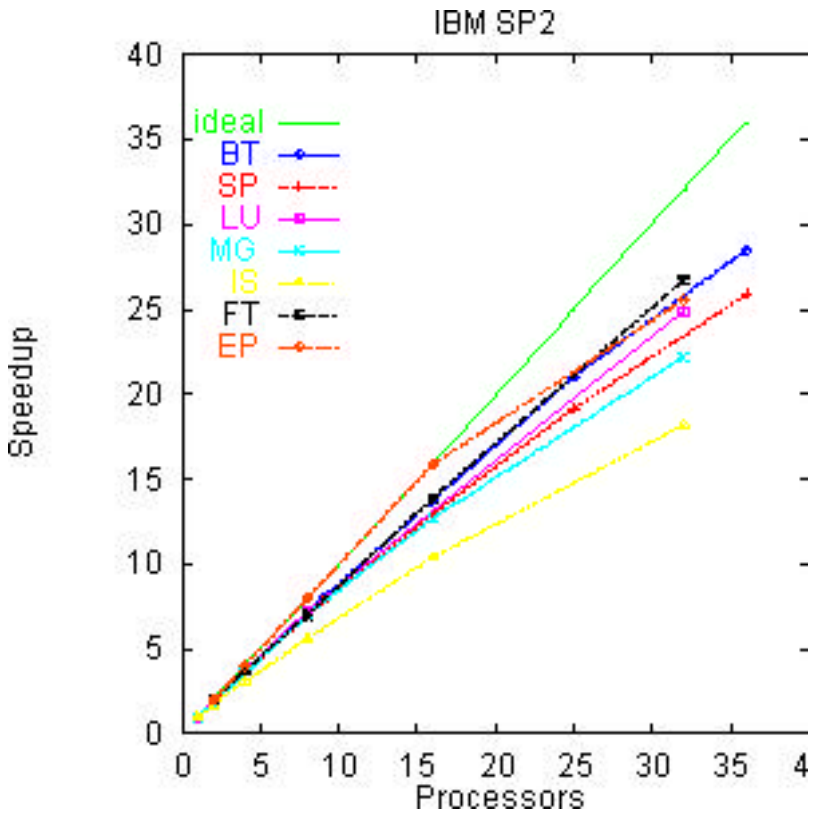


# MPI over AM: start-up

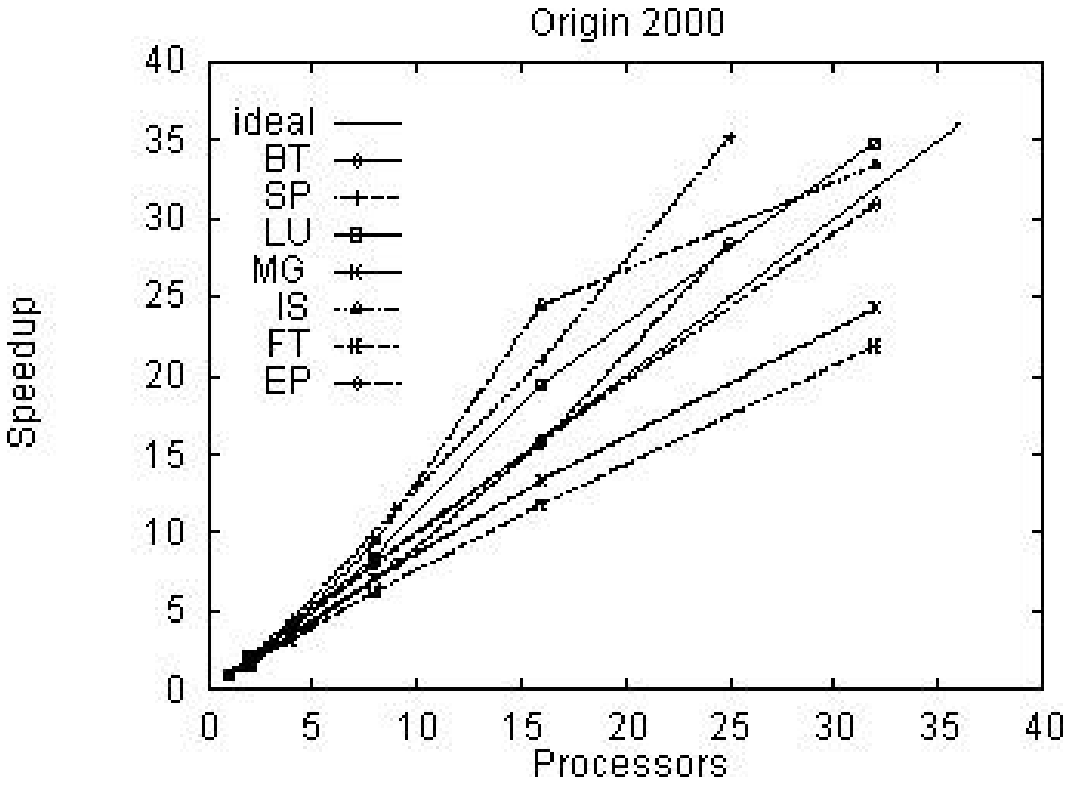


# NPB2 Speedup: NOW vs SP2



# NOW vs. Origin



# Single Processor Performance

| | Origin | Ultra 170 | SP2 |
|---|---|---|---|
| BT | 2488 | 4178 | 2574 |
| SP | 1652 | 2897 | 1817 |
| LU | 1373 | 2470 | 1871 |
| MG | 53 | 90 | 53 |
| IS | 37 | 41 | 29 |
| FT | 133 | 131 | 139 |
| SPECfp95 | 19 | 9.4 | 9.7 |
| SPECint95 | 9.5 | 5.6 | 3.2 |
| Triad MB/s | 317 | 254 | 655 |

# Understanding Speedup

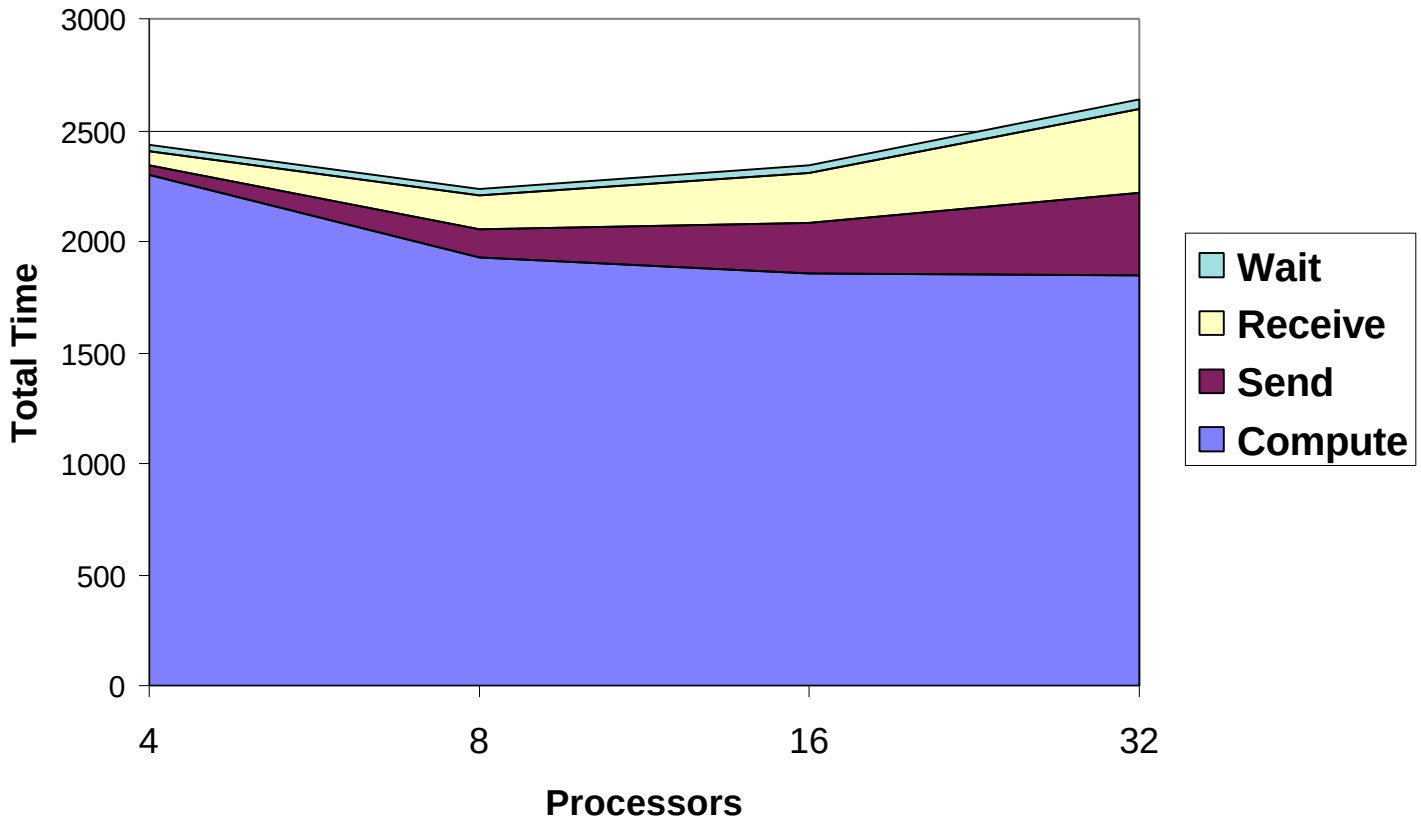
$$\text{SpeedUp}(p) = \frac{T_1}{\text{MAX}_p\,(T_{\text{compute}} + T_{\text{comm.}} + T_{\text{wait}})}$$

$$T_{\text{compute}} = (\text{work}/p + \text{extra}) \times \text{efficiency}$$

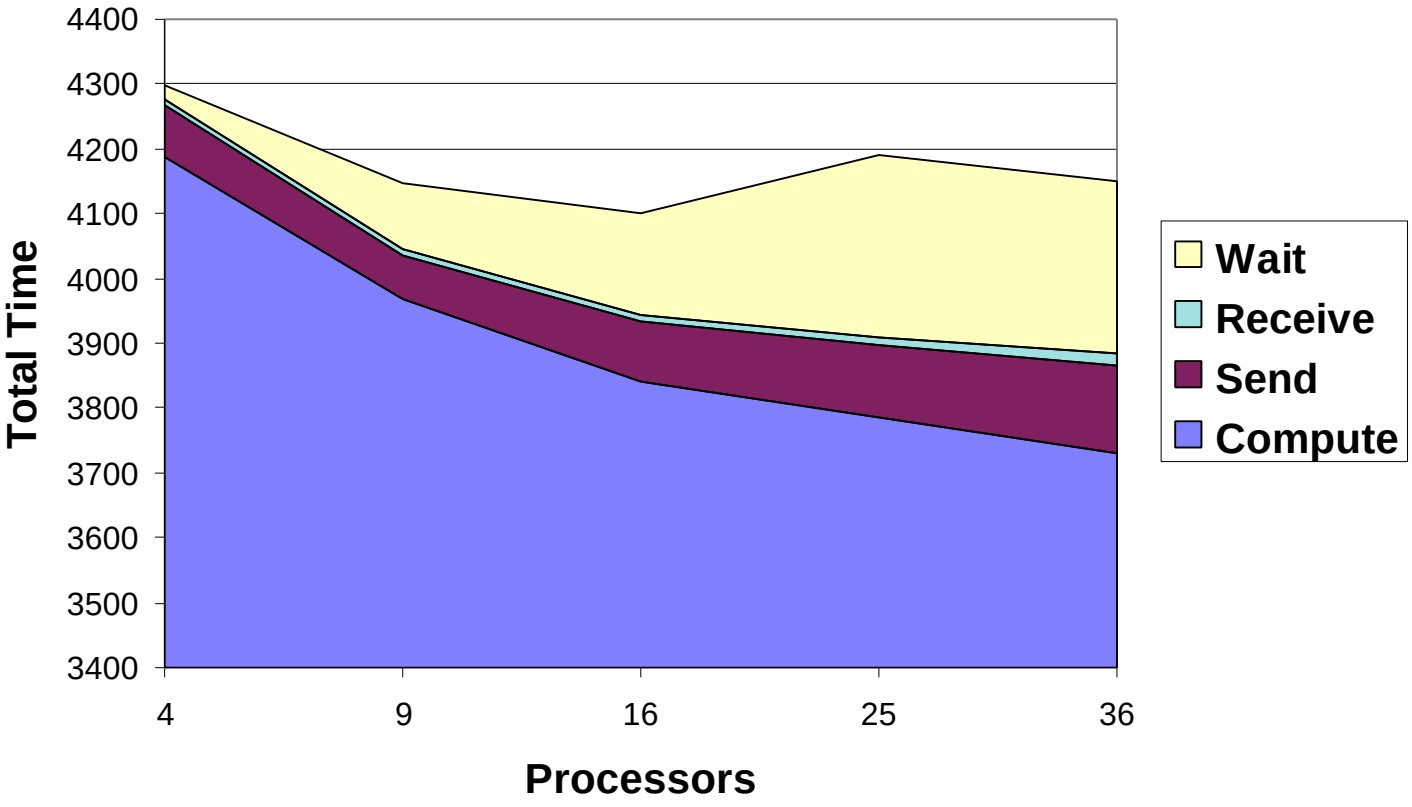
# Performance Tools for Clusters

- **Independent data collection on every node**
  - **Timing**
  - **Sampling**
  - **Tracing**
- **Little perturbation of global effects**

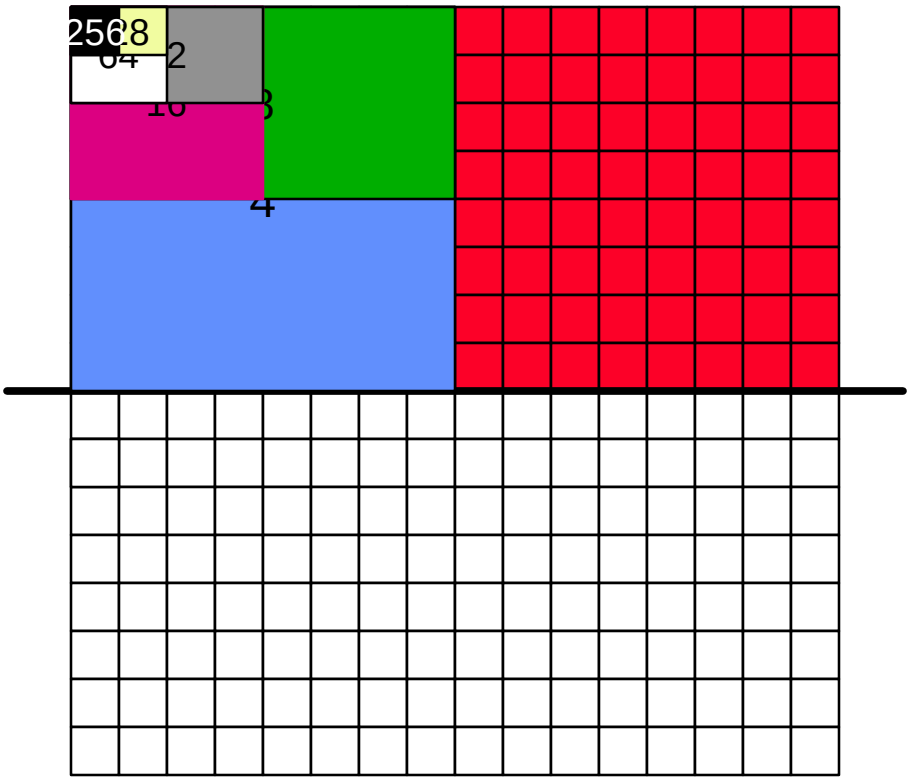
# Where the Time Goes: LU-a



# Where the Time Goes: BT-a

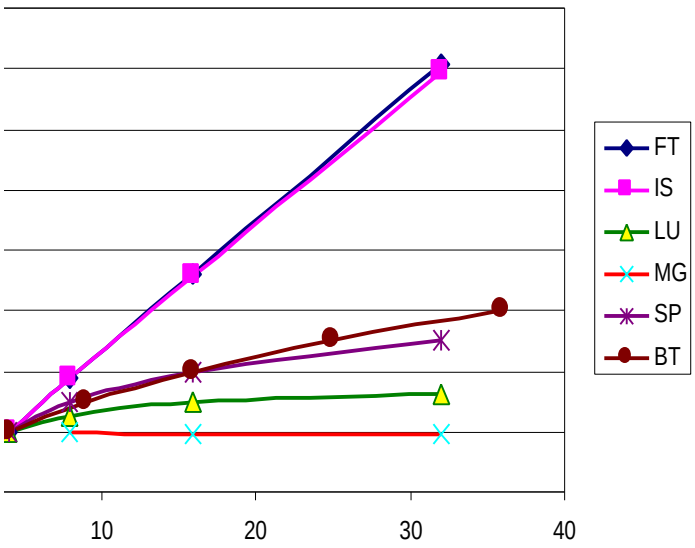


# Constant Problem Size Scaling

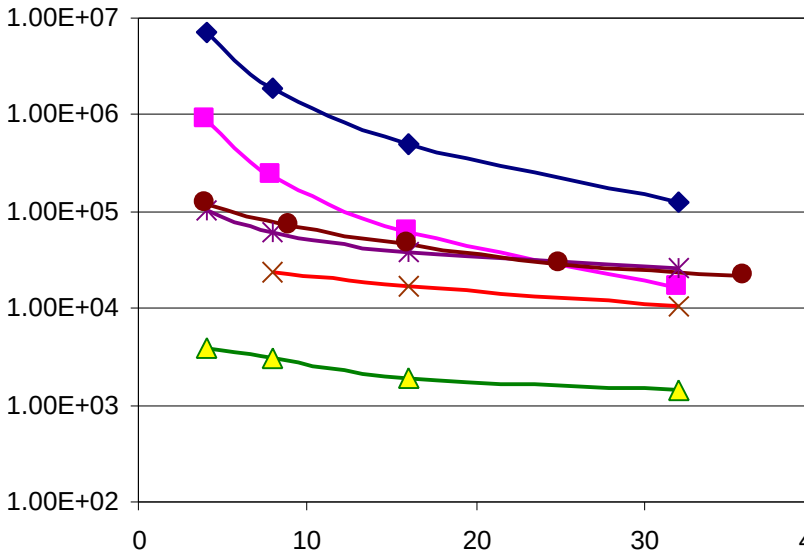


# Communication Scaling

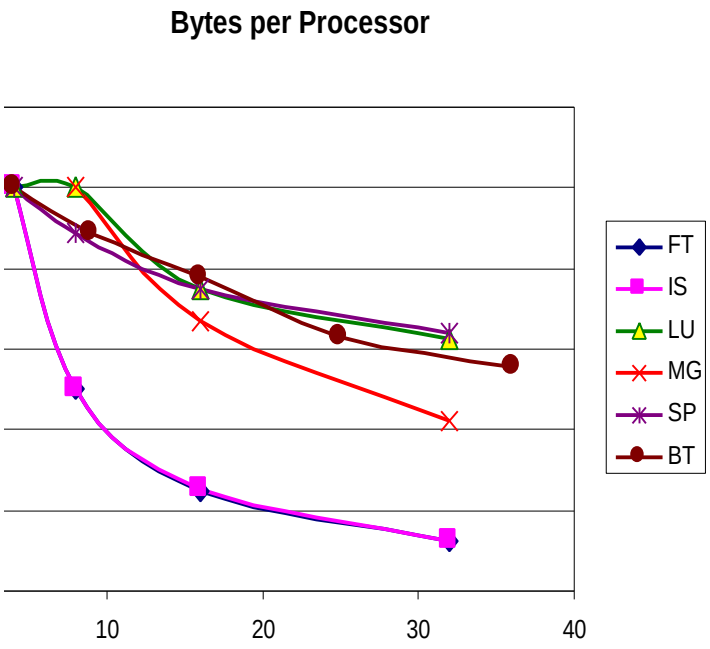
Normalized Msgs per Proc

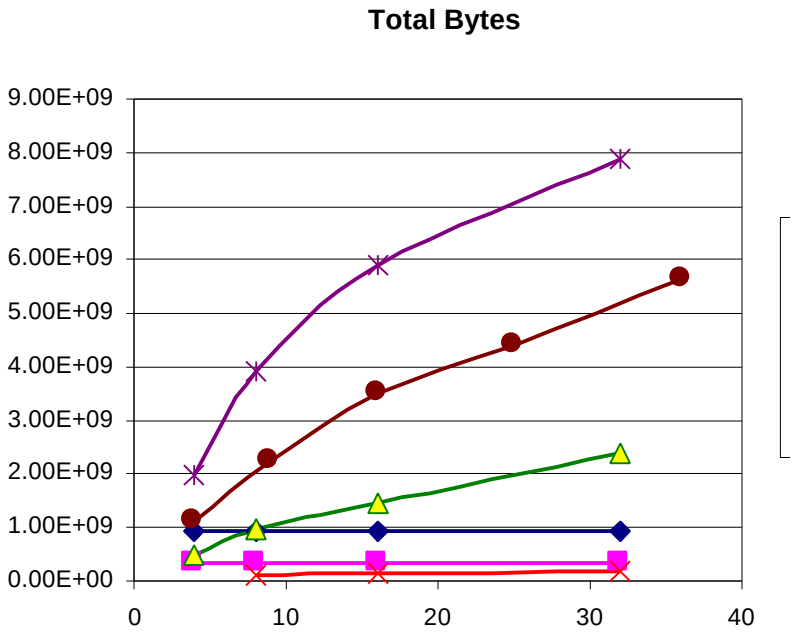



Average Message Size

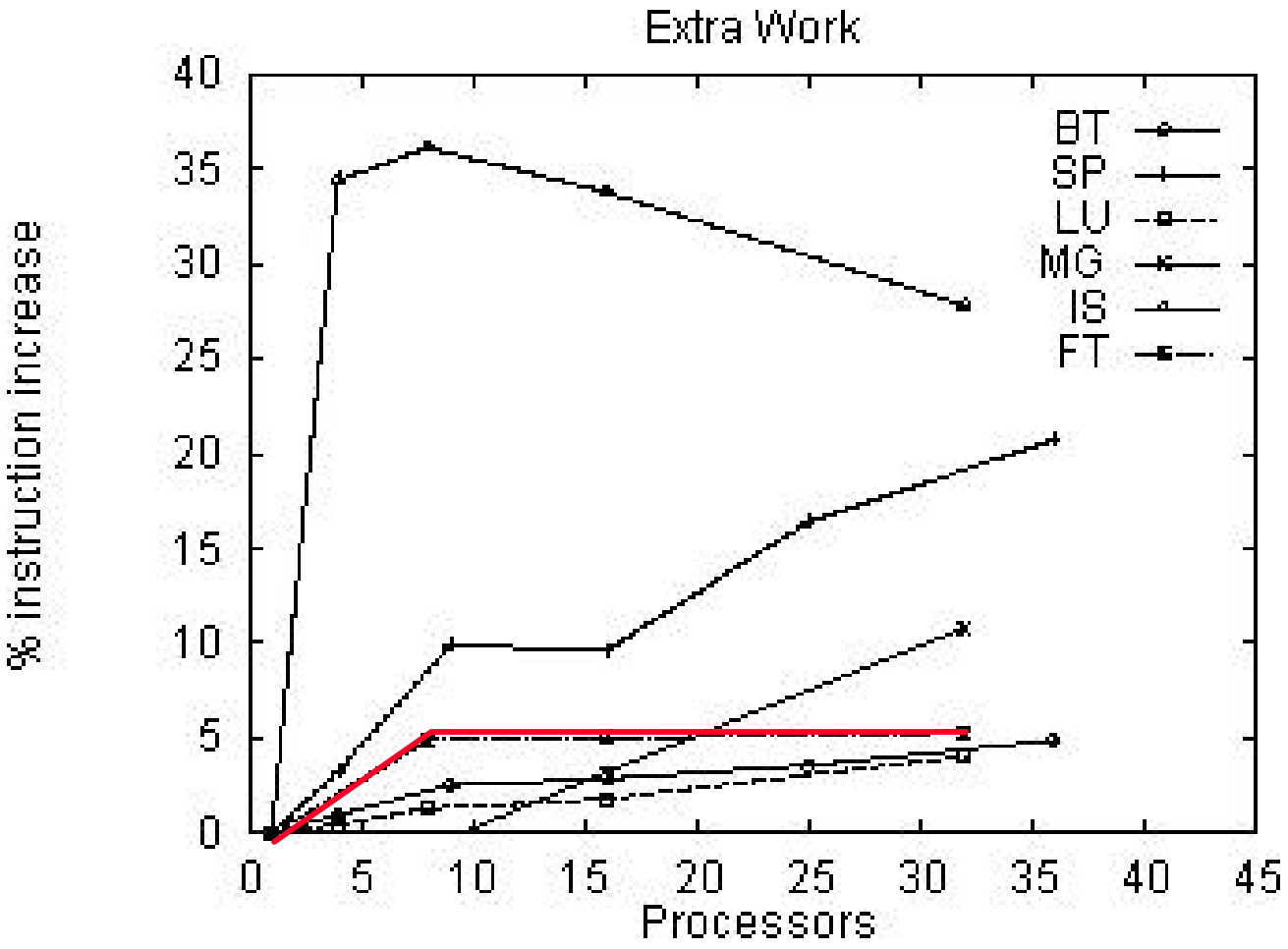


# Communication Scaling: Volume

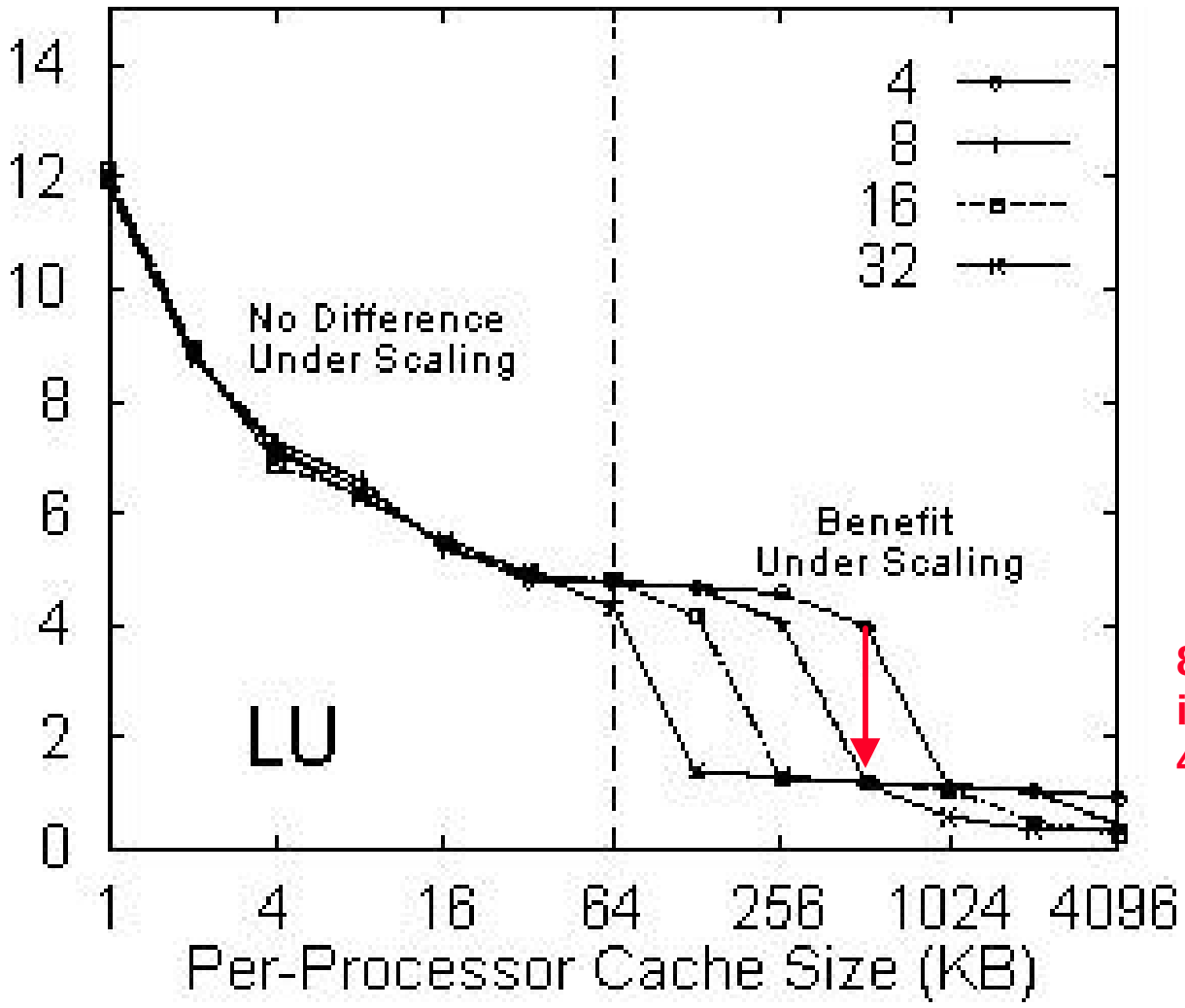


# Extra Work

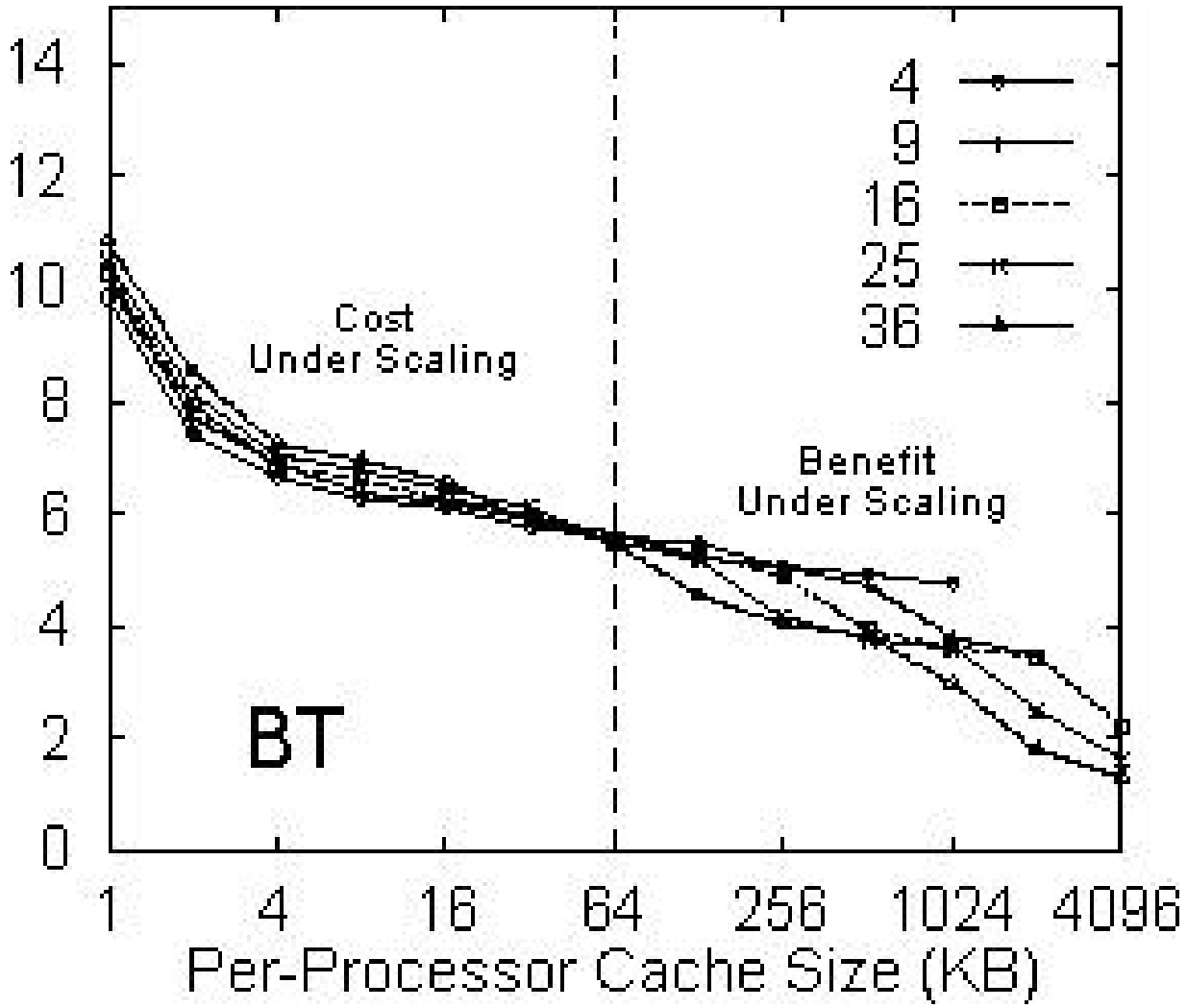

Extra Work
% instruction increase
Processors
0
5
10
15
20
25
30
35
40
45
BT
SP
LU
MG
IS
FT

# Cache Working Sets: LU

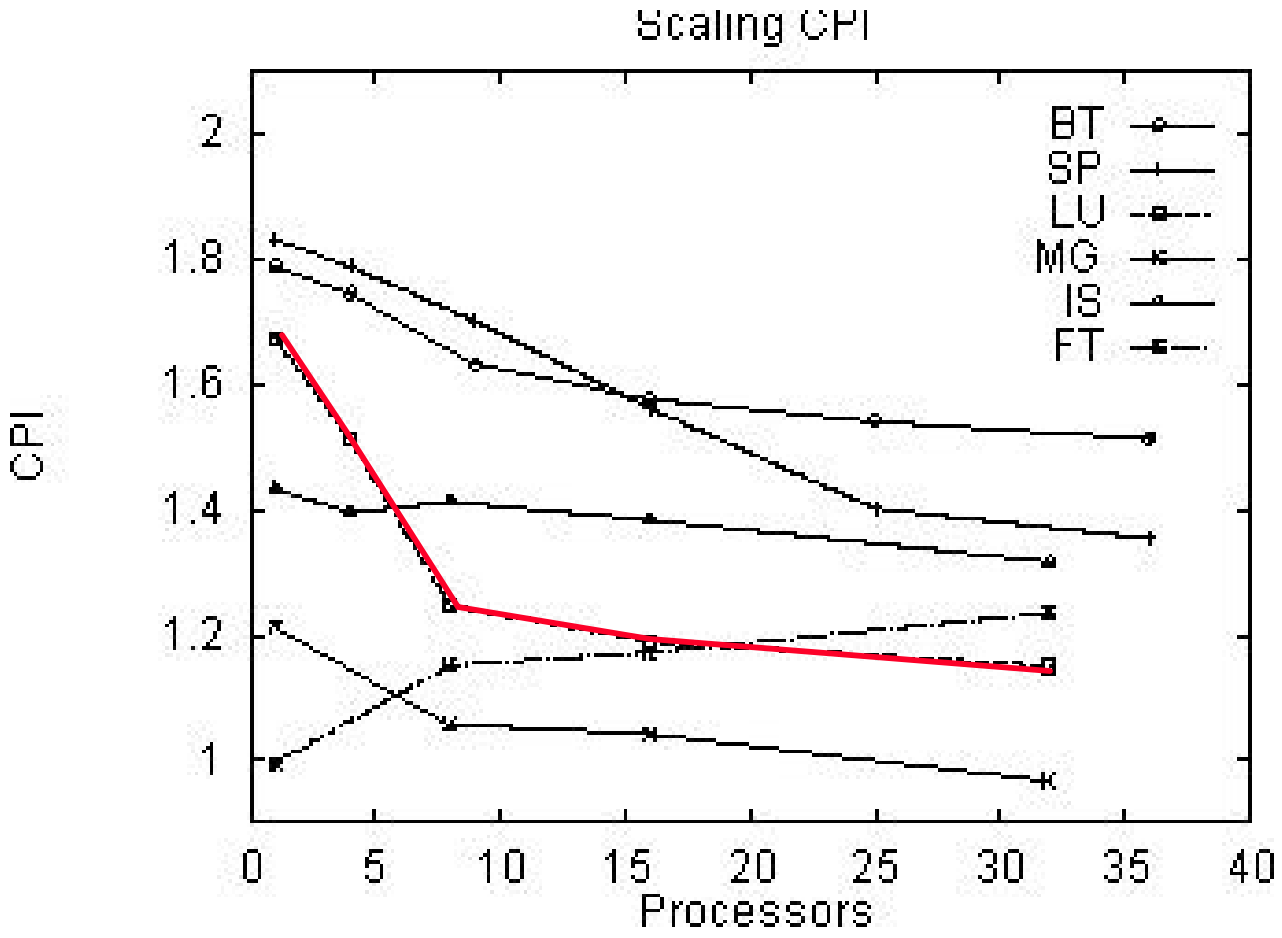



**8-fold reduc in miss rate 4 to 8 proc**

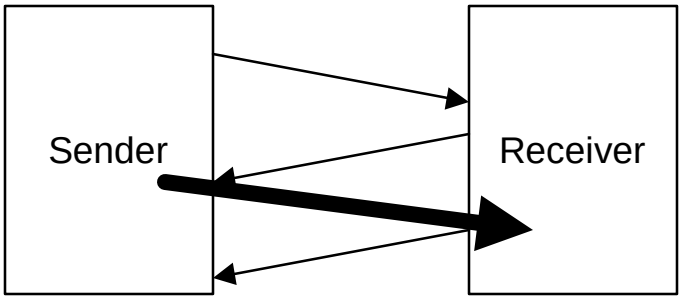
# Cache Working Sets: BT

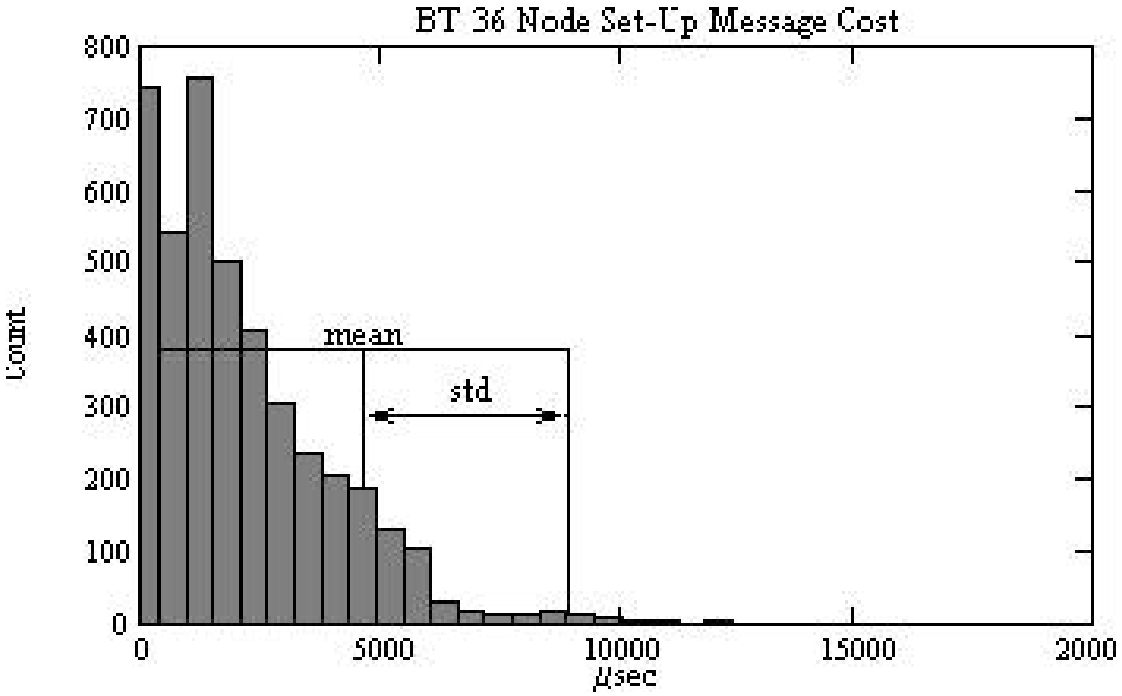


# Cycles per Instruction

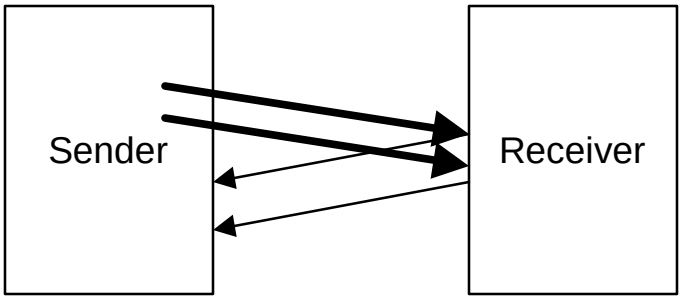


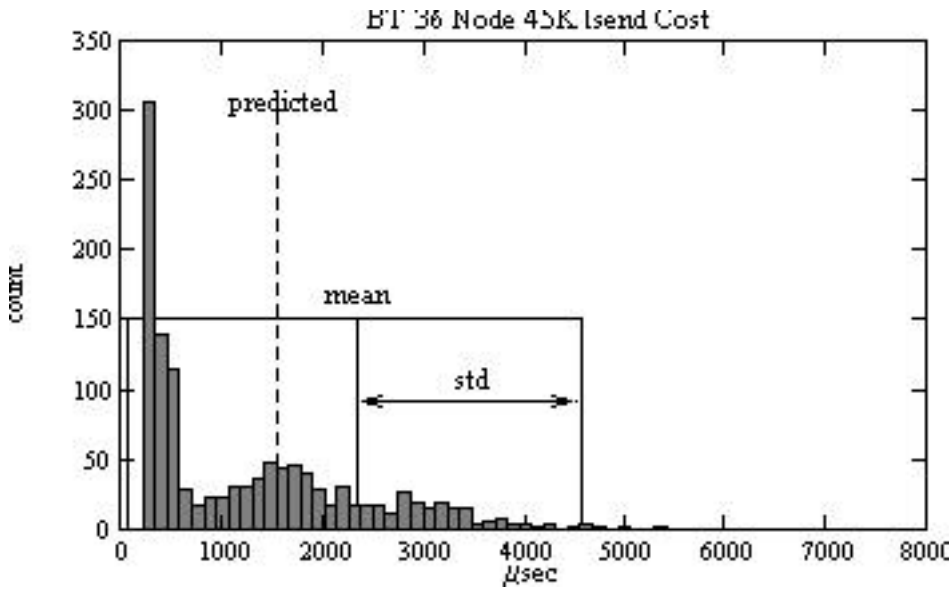
# MPI Internal Protocol



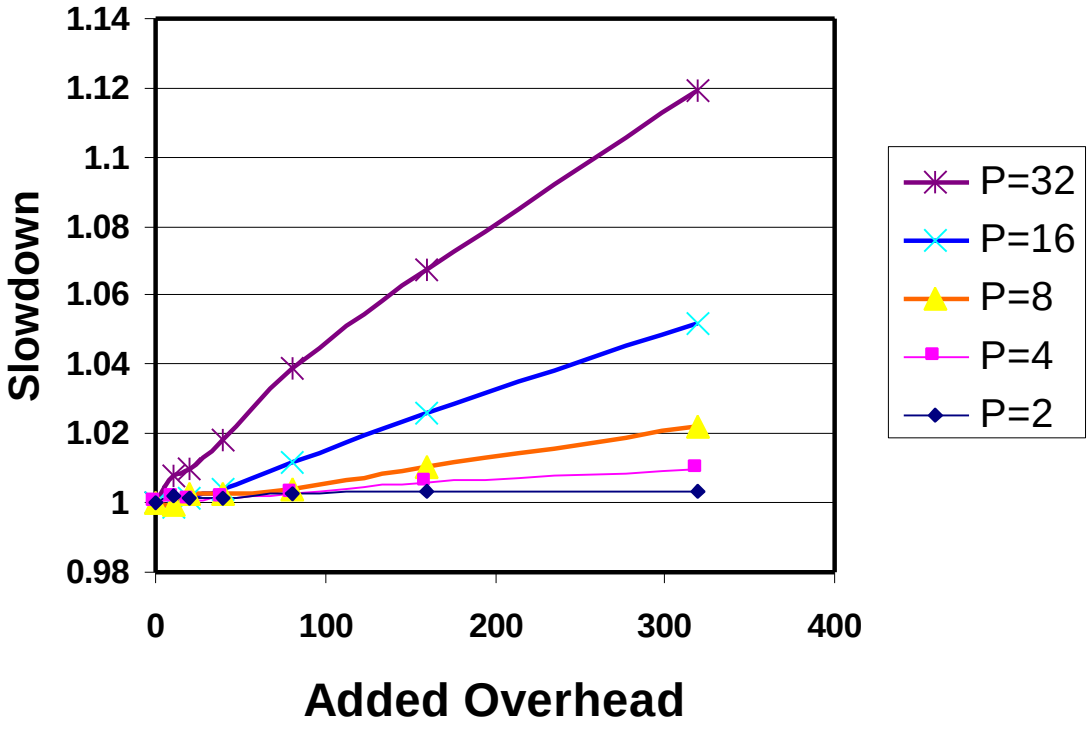



# Revised Protocol



# Sensitivity to Overhead



# Conclusions

- **Run Time systems for Parallel Programs must deal with a host of architectural interactions**
  - **communication**
  - **computation**
  - **memory system**
- **Build a performance model of you RTPP**
  - **only way to recognize anomalies**
- **Build tools along with the RT to reflect characteristics and sensitivity back to PP**
- **Much can lurk beneath a perfect speedup curve**