

Tom Cwik: Welcome  
<http://hpc.jpl.nasa.gov>

Thomas Sterling: Introduction to Beowulf-Cluster Computing  
<http://www.cacr.caltech.edu/~tron/>

Don Becker: Networking for Cluster Computing  
<http://cesdis.gsfc.nasa.gov/people/becker/whoiam.html>

Jan Lindheim: Middleware Systems  
<http://www.cacr.caltech.edu/~lindheim/>

Miron Livny: Condor - High Throughput Computing  
<http://www.cs.wisc.edu/~miron/>

James Jones: the Portable Batch System  
<http://www.nas.nasa.gov/~jjones/home.html>

David Jackson: The Maui Scheduler  
<http://www.mhpcc.edu/maui/>

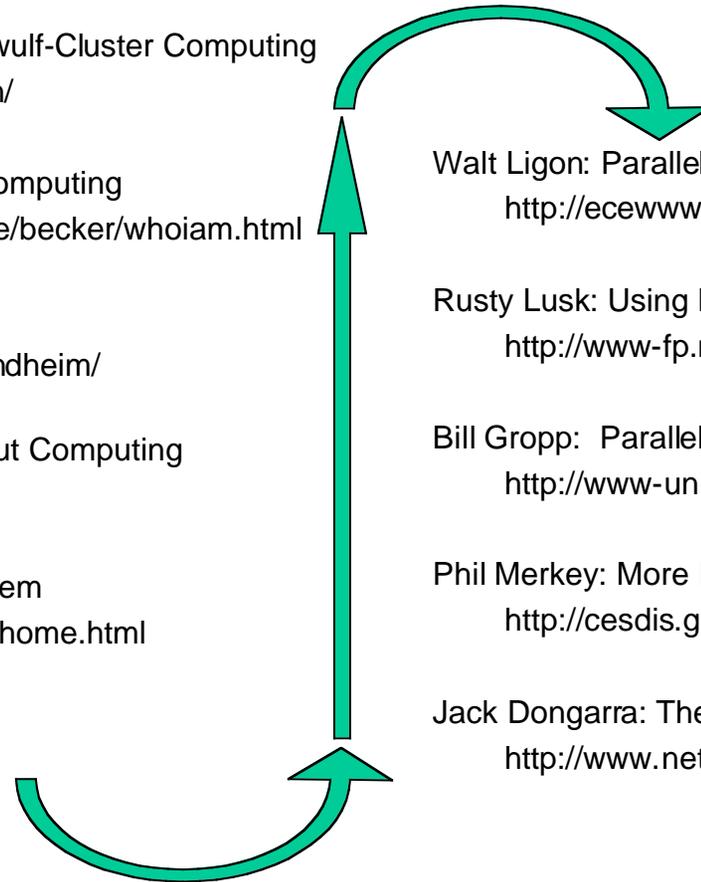
Walt Ligon: Parallel Virtual File Systems  
<http://ecewww.eng.clemson.edu/parl/walt/>

Rusty Lusk: Using MPI  
<http://www-fp.mcs.anl.gov/~lusk/>

Bill Gropp: Parallel Applications for Cluster Computing  
<http://www-unix.mcs.anl.gov/~gropp/>

Phil Merkey: More Parallel Applications for Cluster Computing  
<http://cesdis.gsfc.nasa.gov/people/merk/merk.html>

Jack Dongarra: The ATLAS Project  
<http://www.netlib.org/utk/people/JackDongarra>



UPDATES AT: <http://hpc.jpl.nasa.gov/PS/index.html>



Presented as a tutorial at  
Supercomputing'99:

*Introduction to:*  
**How to Run a Beowulf**

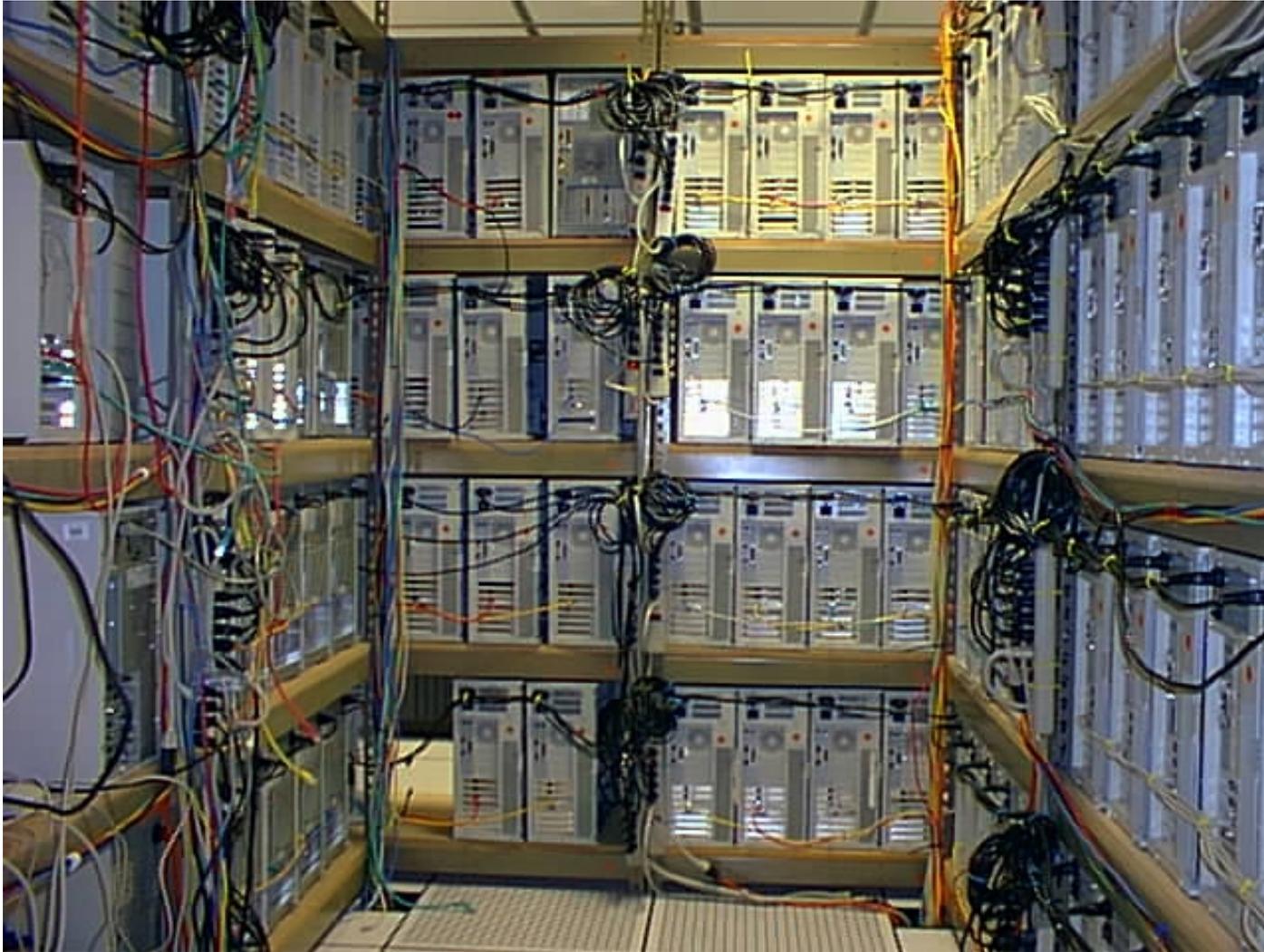
**Dr. Thomas Sterling**

*(and many many friends)*

California Institute of Technology  
and  
NASA Jet Propulsion Laboratory

November 16, 1999

And this is the good news.



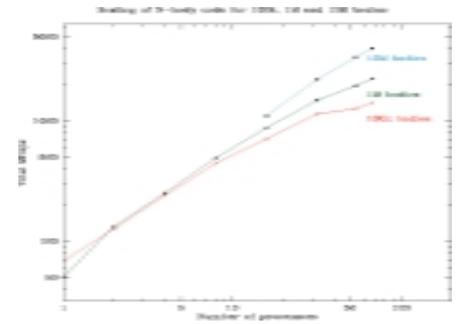
November 16, 1999

How to Run a Beowulf - Sterling et al.

3

# BEOWULF-CLASS SYSTEMS

- Cluster of PCs
  - Intel x86
  - DEC Alpha
  - Mac Power PC
- Pure M<sup>2</sup>COTS
- Linux or other open source Unix-like O/S
  - e.g. BSD, Solaris
- Message passing programming model
  - PVM, MPI, BSP, homebrew remedies
- No longer just single user environments
- No longer just for science and engineering applications

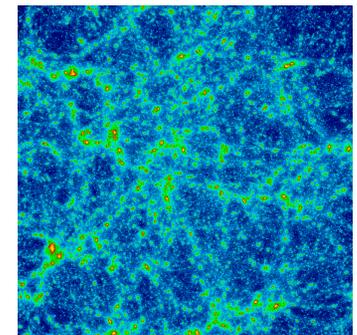


BEOWULF CLASS SYSTEMS

ITEM	DESCRIPTION	QTY	UNIT PRICE	TOTAL PRICE
17	STORAGE-PATCH CABLE 14' LVL 8 TWP/PCB	17	12.00	204.00
44	HELMED3-100 8x30 30MB 40MB	44	150.00	6,600.00
17	CMRTR11-GENERIC VSP-15 180A/758	17	53.00	901.00
1	UNDR140-ACR100 10" 11" MATRYNC	1	1,099.00	1,099.00
1	COMB110-110000 1000 100-24	1	200.00	200.00
1	150000-08 1000 100-24	1	210.00	210.00
1	150000-08 1000 100-24	1	75.00	75.00
1	150000-08 1000 100-24	1	24.00	24.00

SEE BACK FOR TERMS AND CONDITIONS OF SALE.

SUBTOTAL AMOUNT: \$4,514.00  
 SALES TAX: \$1,792.75  
 CREDIT: \$0.00  
 SHIPPING CHARGES: \$0.00  
 OTHER CHARGES: \$0.00  
 GRAND TOTAL: \$6,306.75



November 16, 1999

How to Run a Beowulf - Sterling et al.

# Beowulf-class Systems

## A New Paradigm for the Business of Computing

- Brings high end computing to broad ranged problems
  - new markets
- Order of magnitude Price-Performance advantage
- Commodity enabled
  - no long development lead times
- Low vulnerability to vendor-specific decisions
  - companies are ephemeral; Beowulfs are forever
- Rapid response technology tracking
- Just-in-place user-driven configuration
  - requirement responsive
- Industry-wide, non-proprietary software environment

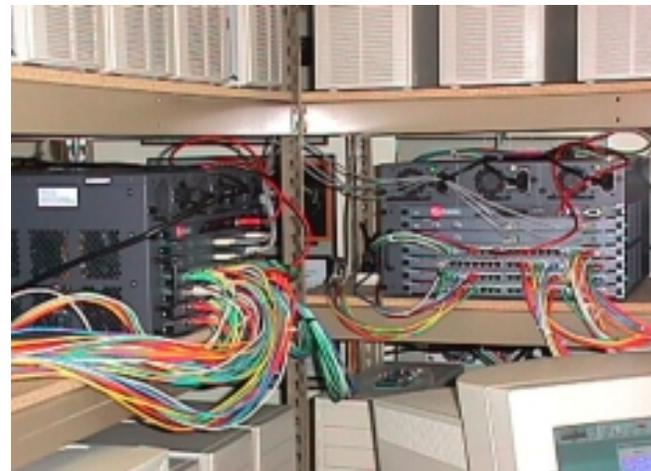
# Multiplicity of Roles for Beowulf

- Single user moderate-scale system
- Dedicated single-application system
- Educational laboratory platform
- Web servers
- Data archiving and retrieval
- Visualization, image processing, rendering, special effects
- Genetic programming
- Computer computer simulators
- Shared multi-user multitasking
- Throughput turbocharging
- Teraflops supercomputer



# Beowulf Hardware on a Slide

- Requirements and priorities
  - performance, size, cost, working environment, applications
- Processor
- Motherboard and chip set
- Memory capacity
- Disk
  - type, capacity, layout
- Packaging
- Network
  - class, NIC, switch, topology
- External interface
- *alternatively: choose a*

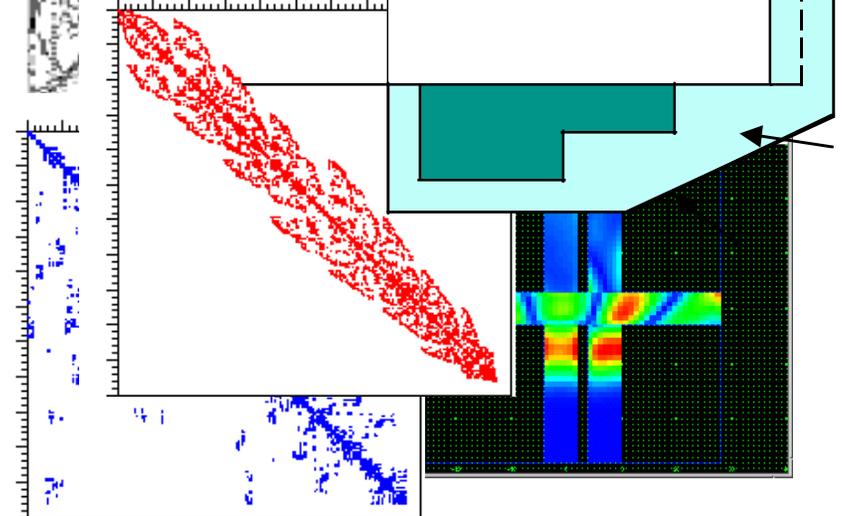
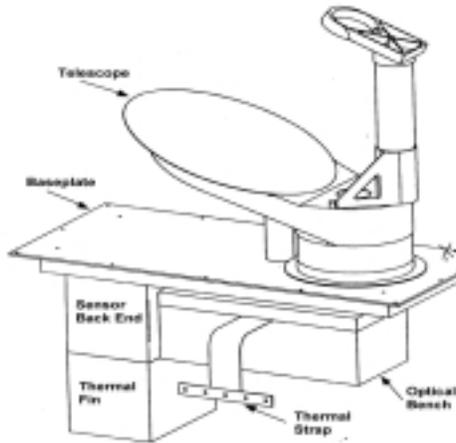
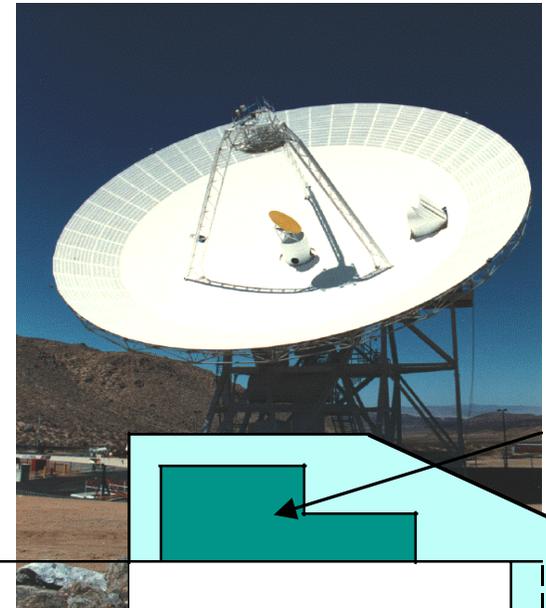
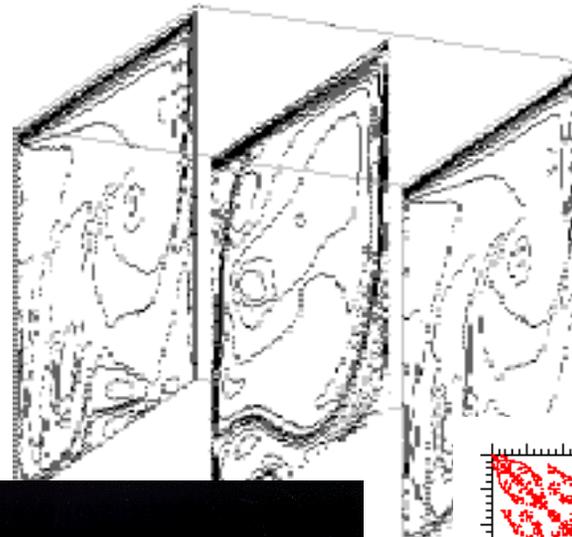
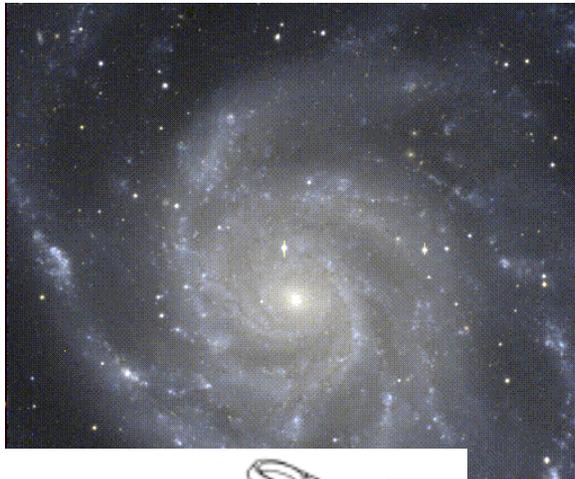


# The Beowulf Delta looking back

- 6 years
- Clock rate: X 6
- flops: X 100 per processor
- #processors: X 32
- Networking: X 100
- Memory: X 10
- Disk: X 30
- price-performance: X 140
- system performance:  
>3000



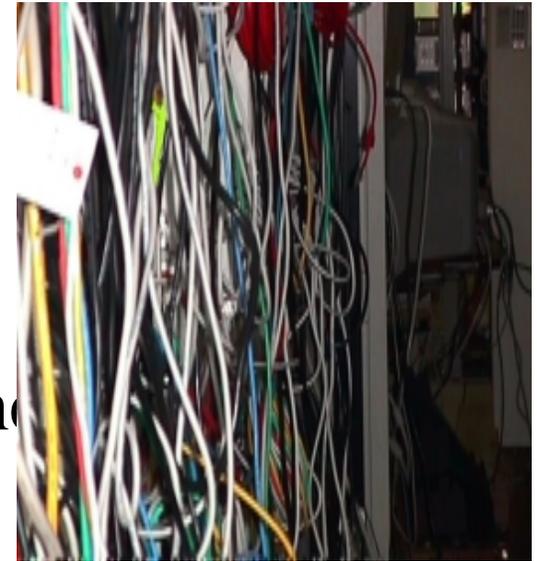
# Beowulf at Work



# Wire We Here?

## What makes Beowulf Run

- Operating system: Linux
- Network drivers: Fast /Gigabit Ethernet
- System initialization
- System management
- Distributed job scheduling: Condor, Maui
- System administration: PBS
- Parallel file system
- Distributed programming library: MPI
- Programming methodology
- Libraries: Atlas



— SCIENTIFIC  
— AND  
— ENGINEERING  
— COMPUTATION  
— SERIES

***How to Build a Beowulf***  
*A Guide to the Implementation and  
Application of PC Clusters*

*Thomas L. Sterling*

*John Salmon*

*Donald J. Becker*

*Daniel F. Savarese*

1st printing: May, 1999  
2nd printing: Aug. 1999  
MIT Press

# ***Building a Beowulf System***

Jan Lindheim

`lindheim@cacr.caltech.edu`

CACR, California Institute of Technology

and

MCS, Argonne National Laboratory

# *Overview*

- Software Installation
  - Tools
  - Customization
  - Sample Setup
- Cloning Process
  - Creating a Node Image
  - Configuring bootpd

# *Tools for making a Beowulf*

- From your favorite Linux or BSD distribution, you need:
  - Basic system software including networking software
  - Kernel sources: <http://kernelnotes.org>
  - Compilers and libraries
  - xntp or similar time synchronizer
  - autofs for mounting network file systems
  - rsync or similar tool for file synchronization

# *Additional Software Needed*

- Message Passing Interface (MPI)  
<http://www-unix.mcs.anl.gov/mpi>
- Parallel Virtual Machine (PVM)  
<http://www.epm.ornl.gov/pvm>
- home grown scripts to maintain system (ex. brsh)
- system monitoring software (ex. bwatch or CACR's bmon)
- <http://beowulf-underground.org> (brsh, bwatch)

# *Classes of Services (Node Types)*

- In a typical Beowulf Cluster we will have at least two classes of services:
  - Front-end server
  - Compute node
- Optionally we may have:
  - Development nodes
  - Storage nodes
  - Router nodes

# *Install and Customize OS*

- For each class of service in your cluster, install the OS of your choice and configure it to do its special task(s).
- In the rest of this talk we base the examples on using RedHat Linux, setting up a Front-End Server with Compute nodes

# *Sample Front-End Server: Configuration*

- Holds home directories
- Holds communication packages (MPI, PVM)
- Performs time synchronization to the outside world
- Controls access to compute nodes
- Provides a full development environment

# *Sample Front-End Server: Setup*

- Install the OS with network services enabled. Make sure to include the packages: knfsd, knfsd-clients, autofs and xntp
- Configure NFS client and server software:

```
# /etc/exports
```

```
/home 192.168.0.0/255.255.255.0(rw,no_root_squash)
```

- Install and configure Automounter:

```
# /etc/auto.beowulf
```

```
n000 -fstype=nfs n000:/scratch
```

# *Sample Front-End Server: Setup Cont.*

- Add an entry to `/etc/auto.master`:

```
# /etc/auto.master
```

```
/data /etc/auto.beowulf -timeout=1200
```

- Configure `xntp` get time from the outside and provide time service to the compute nodes:

```
# /etc/ntp.conf
```

```
server time.cacr.caltech.edu
```

```
broadcast 192.168.0.255
```

# *Sample Front-End Server: Setup Cont.*

- Configure a root NFS for the nodes.

<http://www.cacr.caltech.edu/beowulf/tutorial/NodeCloner.tar.gz>

will help you in this process.

- Install bootpd

<ftp://tsx-11.mit.edu/pub/linux/packages/net/netboot>

or dhcpd (included with RedHat Linux)

# *Sample Compute Node: Configuration*

- Mounts home directories from front-end server
- Mounts communication packages from front-end server
- Gets time synchronization from front-end server

# *Sample Compute Node: Setup*

- Install basic system with network services enabled. Make sure to include knfsd, knfsd-clients, xntp and autofs.
- Install all run-time libraries used on server
- Configure autofs to mount home directories from front-end server:

```
# /etc/auto.beowulf
```

```
home -fstype=nfs front-end:/home
```

# *Sample Compute Node: Setup Cont.*

- Add an entry for auto.beowulf in /etc/auto.master

```
# /etc/auto.master
```

```
/data /etc/auto.beowulf -timeout=1200
```

- Let /home on the compute node be a soft link to the autofs managed file system:

```
rm -fr /home; ln -s /data/home /home
```

- Configure time synchronization:

```
#/etc/ntp.conf
```

```
broadcastclient
```

# *Why use bootp or dhcp for Cloning Environment?*

- Various methods for installing a cluster include:
  - Step-by-step installation on each node
  - Install system on one node. Connect hard disks from the rest of the nodes to the prototype node and perform a disk-to-disk copy (dd)
  - Install system on one node: boot the nodes diskless to partition the hard disk and copy system image onto its hard disk

# ***Why use bootp or dhcp for Cloning Environment?***

- The first two methods are obviously very simple, but very time consuming.
- Configuring bootp or dhcp is more difficult, but the time savings are great.

# *Preparing the system for Cloning (Node Image)*

- Make tar–balls of all system partitions from your prototype compute node. Store these under the NFS root which nodes will mount during cloning:

```
tar cvflz /scratch/root.tar.gz /
```

```
tar cvflz /scratch/usr.tar.gz /usr
```

- Make a linux kernel with built–in support for ramdisk, bootp, root NFS, and drivers for your particular hardware (SCSI and network adapters)
- Register each compute node's MAC address

# *Making the Boot Floppy*

- On the server, insert a floppy and perform the following commands:
  - `mknod /dev/nfsroot b 0 255`
  - `rdev bzImage /dev/nfsroot`
  - `dd if=bzImage of=/dev/fd0 bs=512`

where `bzImage` is a new kernel built from the kernel source, located under `/usr/src/linux`

# Configuring Bootpd

- Sample bootptab file:

```
# /etc/bootptab
```

```
.pentium:sa=192.168.0.250:gw=192.168.0.254:ra=192.168.0.255: \  
:sm=255.255.255.0:ht=ether:hn:dn=cacr.caltech.edu: \  
:rp=/home/rootnfs/pentium
```

```
.alpha:sa=192.168.0.250:gw=192.168.0.254:ra=192.168.0.255: \  
:sm=255.255.255.0:ht=ether:hn:dn=cacr.caltech.edu: \  
:rp=/home/rootnfs/alpha
```

```
p000:ha=0080c83e2a11:tc=.pentium
```

```
a000:ha=0080c86386a8:tc=.alpha
```

# *The Cloning Process*

- Boot from the previously created floppy
- Through bootp or DHCP the node receives an IP address and path to its root NFS file system
- The node mounts root from server over NFS
- An init script will partition hard disk and extract tar archives to the proper partitions and customize network name and address and reboot the node

# *Benefits of this Cloning Environment*

- Stores sets of partition tar-balls as backups
- Can quickly rebuild a node after hard disk failure or file system corruption
- By keeping a cloning kernel on each node's disk, we can also tell lilo to use this to rebuild the node on next reboot. No floppy is needed!

# *Useful Linux & Beowulf Web Pages*

- <http://linuxlinks.com>
- <http://kernelnotes.org>
- <http://www.beowulf.org>
- <http://www.cacr.caltech.edu/beowulf>
- <http://metalab.unc.edu/mdw/HOWTO>
- <http://kt.linuxcare.com>

# *Conclusion*

- Initial time investment in server configuration pays off in future efficiency and expansibility
- Most tools needed are freely available on the net
- Documentation is plentiful
- Custom solutions are easily implemented
- Anybody who has done some unix administration can easily set up and maintain a simple cluster

# *The Portable Batch System (PBS): A Technical Overview*

James Patton Jones

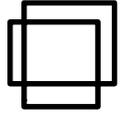
*jjones@pbs.mrj.com*

MRJ Technology Solutions

SC99 Booth #871

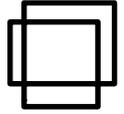
15 November 1999





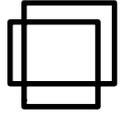
## ***PBS: The Leading OpenSource Batch System:***

- Multiple User Interfaces (CLI, GUI)
- Job-interdependency
- Cross System Scheduling
- Security and Access Control Lists
- Job Accounting
- Dynamic Distribution of Workload
- Automatic Load-leveling
- Parallel Job Support
- Automatic File Staging
- Comprehensive (and Documented) API



## ***PBS: System Overview***

- A client-server TCP-based system
- Can run on a standalone system, or in a distributed, heterogenous UNIX environment
- Utilizes UNIX security augmented by Access Control Lists (per-user, -group, -host, -domain)
- Completely modular (e.g. Security model can be replaced; username-mapping hooks provided, etc.)
- Kerberization of PBS in progress
- Supported on: Solaris, SunOS, IRIX, AIX, Digital Unix, UNICOS, Linux, FreeBSD, NetBSD, ...

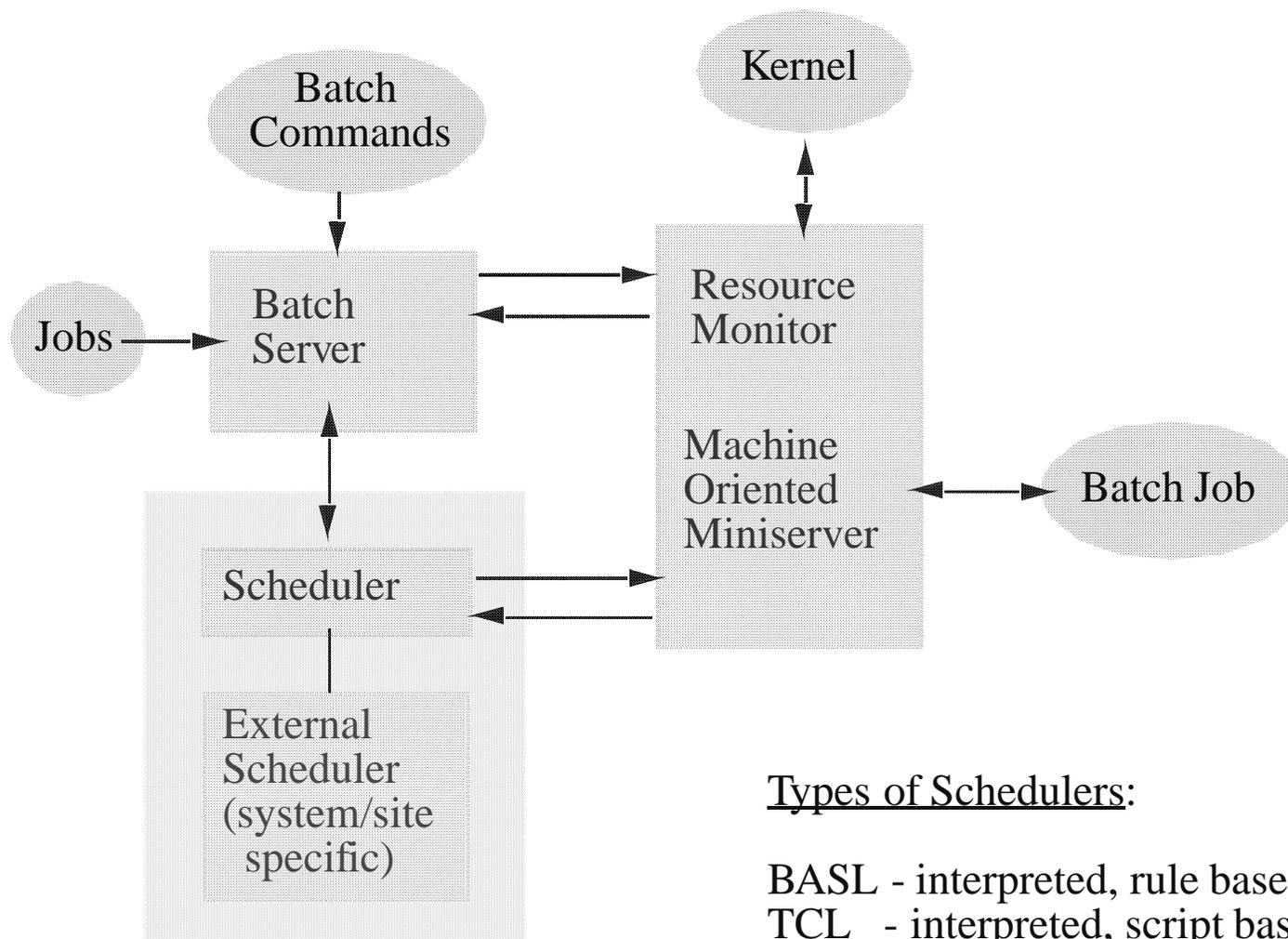


## ***PBS Components***

- **PBS Daemons**
  - batch server (maintains queues, jobs, ACLs)
  - resource monitor + mini-server (executes/monitors jobs)
  - scheduler (runs jobs according to local policy)
- **PBS User Commands**
  - qsub, qstat, qdel, qalter, qhold, qrels, ...
- **PBS Administrator Commands**
  - qmgr, pbsnodes, qrerun, qselect, qmove, ...

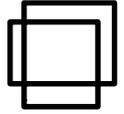


# How the PBS Components Interact



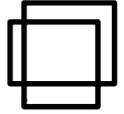
## Types of Schedulers:

- BASL - interpreted, rule based
- TCL - interpreted, script based
- C - compiled



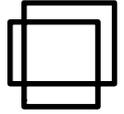
## *The PBS Scheduler Module*

- Default scheduler is general purpose, offering a variety of (selectable) scheduling algorithms, including:
  - first-in, first-out, round robin, load-balancing, load-stacking, pseudo fair share
- Several system-specific schedulers are included:
  - Cray UNICOS C90 and J90, IBM SP (with dynamic backfilling), SGI PowerChallenge, SGI Origin2000
- The scheduler is designed to be easily extended. Three interfaces are provided to enable any scheduling policy:
  - TCL/tk, BASL (a compiled scheduling language), and
  - C (most PBS schedulers are written in C using the PBS API)



## *Acquiring PBS*

- Official distribution site: *pbs.mrj.com*
- PBS is distributed in source code form
  - also available in Redhat Linux RPM format
- No-cost registration and download
- Documentation is included:
  - Administrator's Guide, External Reference Spec., Internal Design Spec.
- *pbs-users* mailing list archived on website
- FAQ, Tips For Users, Advice for Administrators...  
also on website



## ***Optional PBS Support Services From MRJ***

- **Systems Analysis**
  - Requirements analysis of computational needs
  - Recommendations of scheduling policies
  - PBS-Site integration
- **PBS Customization**
  - Customization of scheduling policies
  - Optimization of scheduling and system configuration
  - Custom installations
- **PBS Support and Maintenance**
  - Hourly, Annual, and Extended (24x7 coverage) support contracts
- **Customer Education**
  - PBS User Training, PBS Administrator Training
  - PBS Internals/Developer Training

# Cluster Resource Management

- Resource management is more than FIFO scheduling

# Cluster Resource Management

David B Jackson

Cluster Tutorial-SC99

Nov 14, 1999

# Maui Scheduler Features

- Scheduling Optimization
- Allocation Management
- Advanced Reservations
- Multiple Resource Management Interfaces
- SMP Enabled
- Extensive Policy Control Facilities

# Initial Clusters

- Homogenous
- Autonomously Managed
- Single User Workload
- FIFO Scheduling
- Excess Capacity
- Local Usage

# Cluster Evolution

- Multiple sources of management policies
- Multi-User workload
- Backlog of workload
- Heterogeneous resources
- Fragmented resources
- Requirements for advanced functionality
- Remote users/collaborations

Cluster Resource Management Issues- Optimizing Scheduling and Allocation Policies for Multiple Local and Remote Users Competing for Limited, Fragmented, Heterogeneous Resources under Multiple Political Policy Constraints

- Fairness (throttling/fairshare)
- Minimizing queue time/maximizing utilization
- Prioritizing workload
- Guaranteeing service

# Cluster Political Policies

- Resource access (where can they run)
- Prioritization (who runs first)
- Allocation (how much can they use)
- Functionality Access (who uses what function)

# Cluster Scheduling Issues

- Minimizing resource fragmentation (heterogeneous resources)
- Maximizing locality
- Queue Optimizations (backfill)
- Active Job Optimizations (preemption, gang-scheduling, etc)
- Utilization/turnaround optimizing prioritizations.

# Cluster Functionality

- Preemption policies
- Advance Reservations
  - deadline scheduling
  - special project handling
- Metascheduling (Remote/Collaborative workload)
- Co-Allocation of compute resources
- Co-Allocation of non-compute resources

# Maui Scheduler

- Open Source
- Fairness Policies
  - Throttling
  - Fairshare
- Prioritization Policies
  - Resources
  - Utilization
  - QOS
  - Credential
- Optimizations
  - Backfill
  - Node Allocation

# Maui Scheduler - Cont'd

- Allocation Control
  - Historical utilization based prioritization
  - Allocation bank interface (QBank)
  - Multiple accounts per user/Multiple users per account. (w/ Kitty and Reserved subaccounts)
  - Local account administration
  - Allocation Expiration/Real Time Allocation Checking/Debiting
- Advanced Reservations
  - Standing Reservations
  - Interactive Reservations
- Internal Diagnostics
- Extensive Statistics
- PBS/Loadleveler support

# Conclusions

- Clusters start easy but quickly evolve
- Must look to the future when designing clusters
- Tools are available to assist in this effort

# **The Parallel Virtual File System: Past, Present, and Future**

Walt Ligon and Rob Ross  
Parallel Architecture Research Laboratory (PARL)  
Clemson University, Clemson, South Carolina  
<http://ece.clemson.edu/parl/>

# What is PVFS?

---

- Parallel file system
  - Global name space
  - Physical distribution of data
- User-level implementation
- No message passing library dependencies
- Accessible with existing binaries

## A Short History of PVFS

---

- 1993-1994 – PVFS atop PVM implemented by Aric Blumer
- 1994-1995 – PVFS re-implemented with TCP on Alphas
- 1995-1996 – PVFS ported to Linux, Goddard Beowulf (486's)
- 1996-present – Major reorganization, C Library support added

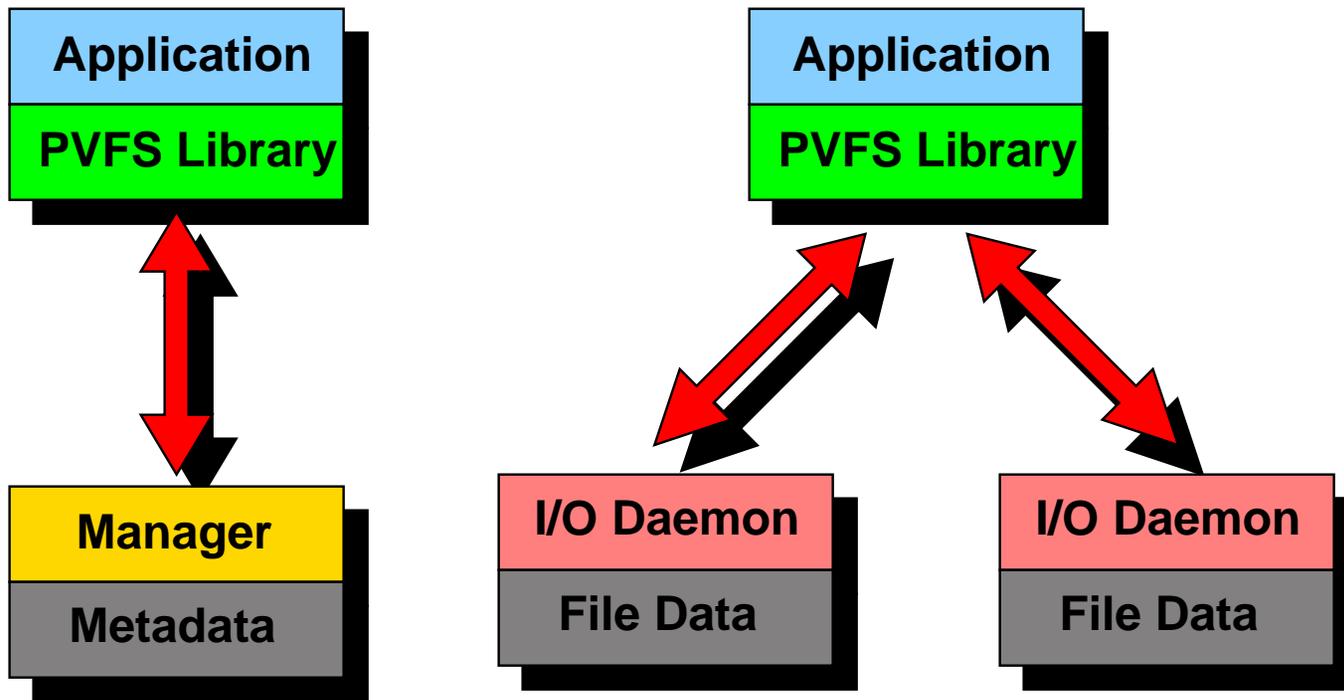
# PVFS Architecture

---

- Set of cooperating daemons handle file accesses
  - Manager Daemon performs permission checking for file accesses, manages opening, closing, creating, and removing files
  - I/O Daemons perform file I/O on I/O nodes and transfer data to application tasks
- Library of function calls provides link between applications and PVFS daemons
- Wrappers allow existing binaries to operate on PVFS files

# PVFS Architecture

---



# Storing File Data

---

- Decisions:
  - Use existing file systems on disks local to I/O nodes
  - Use a separate physical file for each part of a PVFS file
  - Give files unique identifiers within scope of file system
- Ramifications:
  - UNIX `mmap()`, `read()`, `write()`, etc. can be used for I/O
  - PVFS has no direct control over caching or block allocation

# Storing Metadata

---

- Metadata is information describing a file:
  - Permissions, modification times
  - Physical distribution of data
- Originally stored in a single file held by manager
- Now stored in a shared NFS file system:
  - Gives us unique name space
  - Provides directory structure for applications to see
  - Lightens load on manager

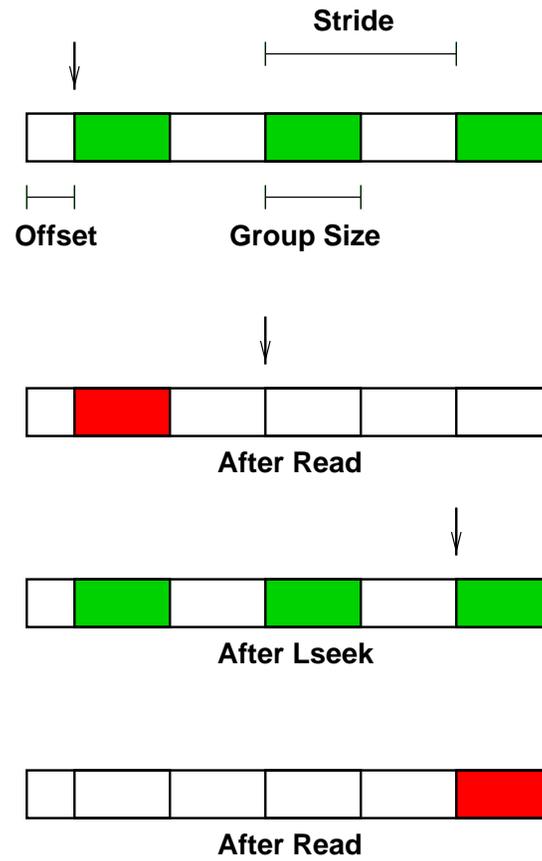
# Applications Interface

---

- Influences:
  - Charisma Project – common access patterns
  - Vesta Project – logical partitioning of files
- Partitioned file interface:
  - Simple partitioning, independent of physical distribution
  - Direct support for simple strided accesses
  - Fits 2D matrix applications well

# Partitioning in PVFS

---



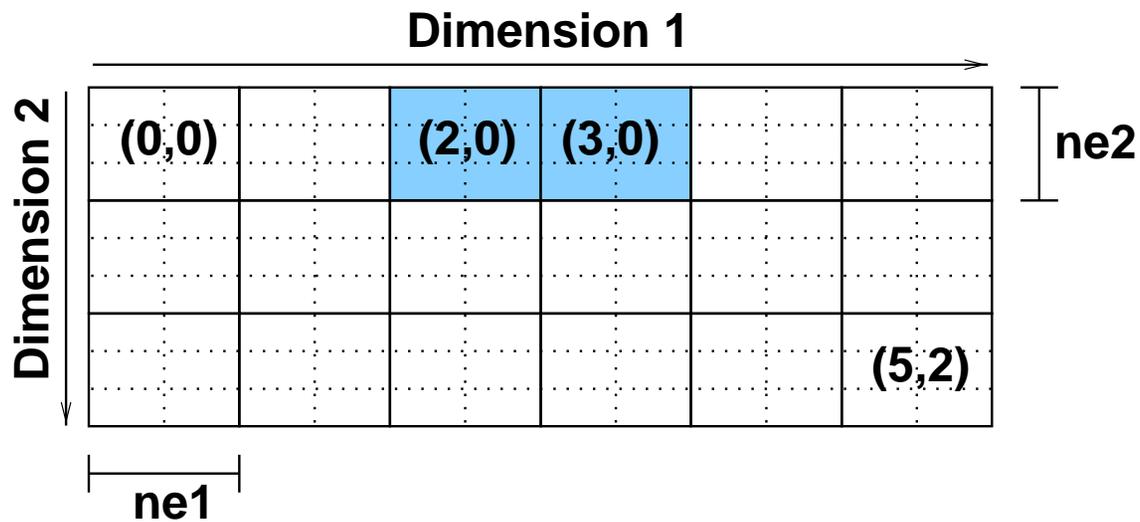
# Out-of-Core Interface

---

- Multi-Dimensional Block Interface (MDBI)
  - Allows a file to be viewed as a multidimensional array of records
  - Provides easy access to non-contiguous data without directly partitioning the file
  - Allows user specified buffering of file data

# Blocking with MDBI

---



 = 1 record  = 1 block  = 1 superblock

(D = 2, rs = 500, ne1 = 2, nb1 = 6, ne2 = 3, nb2 = 3, bf1 = 2, bf2 = 1)

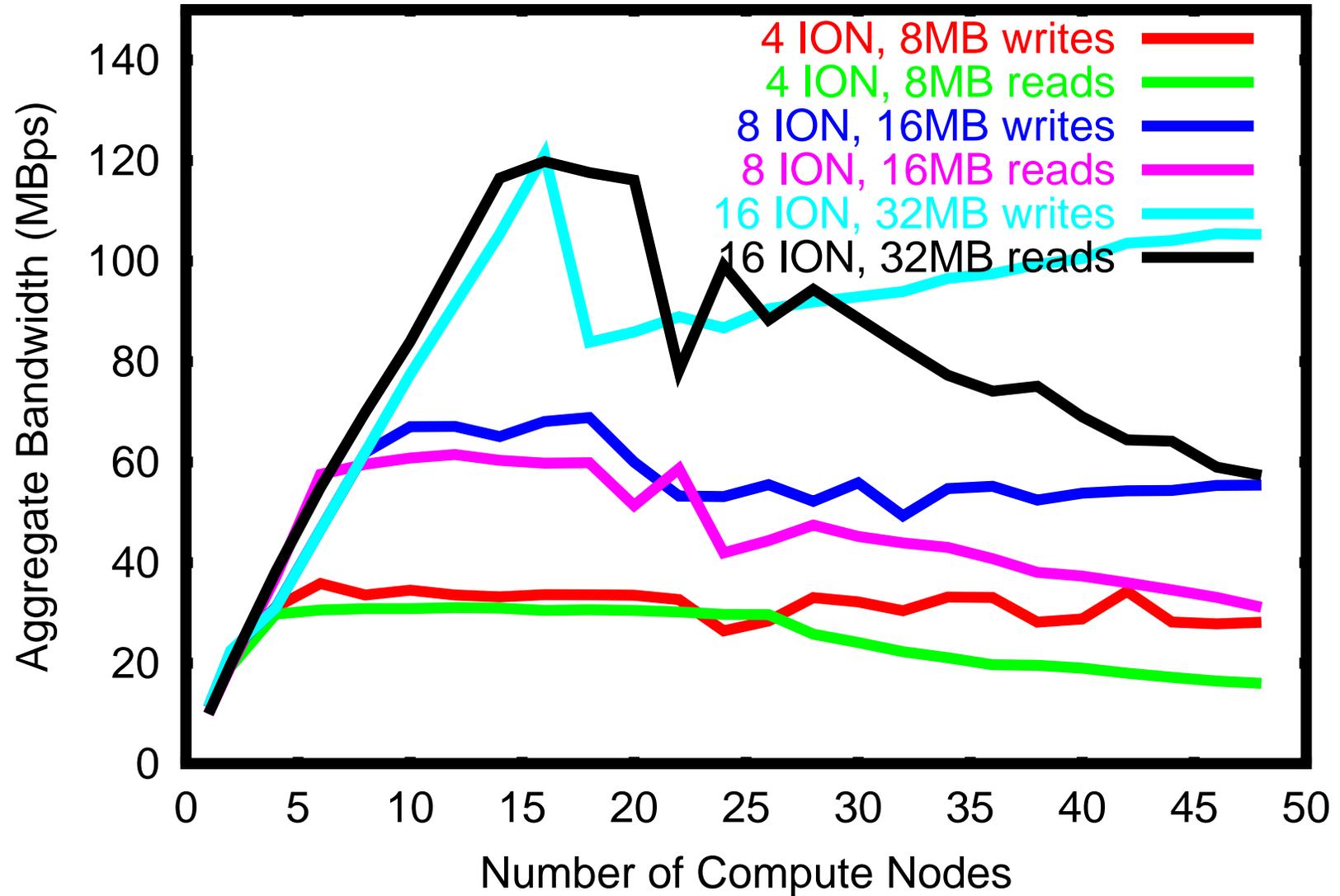
# Accessing With Existing Binaries

---

- Normally when an application performs I/O:
  - The application uses a system call to perform I/O
  - Wrappers in the C library catch this and perform the actual call
- Our modifications:
  - We rewrote the wrappers to let us catch I/O calls
  - We keep the state of files in user space
  - Pass non-PVFS I/O on to the system calls, handle PVFS I/O transparently

# Scaling and I/O Nodes

---



# MPI and Beowulf

Rusty Lusk

Argonne National Laboratory

November 1999

# Outline

- About MPI itself
- MPI implementations on Beowulf systems
- An introduction to MPI via examples
- MPI-based libraries
- Sources for MPI information
- Demo

# What is MPI?

- *A message-passing library specification*
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for
  - end users
  - library writers
  - tool developers

# Why Does Beowulf Have MPI?

- MPI enables portable parallel libraries
  - provides leverage
  - can use same libraries as on more expensive parallel supercomputers
- You can write your own parallel programs using the powerful and general message-passing model of parallel computation.
- Portability:
  - Develop on heterogeneous workstation networks; run on Beowulf
  - Develop on Beowulf; run on teraflop machines

# MPICH on Beowulf

- MPICH is a freely available, high-performance, portable implementation of MPI for nearly all parallel machines.
- On Beowulf, MPICH communicates using TCP/IP.
- Includes all source, user manual, man pages, many example programs
- Build and run on Beowulf (as on any machine) with:
  - configure
  - make
  - make install prefix=/usr/local/mpi (or wherever)
  - mpicc -o myprog myprog.c
  - mpirun -np 32 myprog

# LAM - An Alternate Implementation of MPI

- LAM is another implementation of MPI on Beowulf systems
- Available from Notre Dame at <http://www.cs.nd.edu/lam>

# Why Use MPI?

- MPI provides a powerful, efficient, and *portable* way to express parallel programs
- MPI was explicitly designed to enable libraries...
- ... which may eliminate the need for many users to learn (much of) MPI

# A Minimal MPI Program (C)

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

# A Minimal MPI Program (Fortran)

```
program main
use MPI
integer ierr

call MPI_INIT( ierr )
print *, 'Hello, world!'
call MPI_FINALIZE( ierr )
end
```

# Notes on C and Fortran

- C and Fortran bindings correspond closely
- In C:
  - `mpi.h` must be `#included`
  - MPI functions return error codes or **`MPI_SUCCESS`**
- In Fortran:
  - `mpif.h` must be included, or use MPI module (MPI-2)
  - All MPI calls are to subroutines, with a place for the return code in the last argument.
- C++ bindings, and Fortran-90 issues, are part of MPI-2.

# Running MPI Programs

- The MPI-1 Standard does not specify how to run an MPI program, just as the Fortran standard does not specify how to run a Fortran program.
- In general, starting an MPI program is dependent on the implementation of MPI you are using, and might require various scripts, program arguments, and/or environment variables.
- **mpirexec <args>** is part of MPI-2, as a recommendation, but not a requirement
  - write your MPI implementor

# Finding Out About the Environment

- Two important questions that arise early in a parallel program are:
  - How many processes are participating in this computation?
  - Which one am I?
- MPI provides functions to answer these questions:
  - **MPI\_Comm\_size** reports the number of processes.
  - **MPI\_Comm\_rank** reports the *rank*, a number between 0 and size-1, identifying the calling process

# Better Hello (C)

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

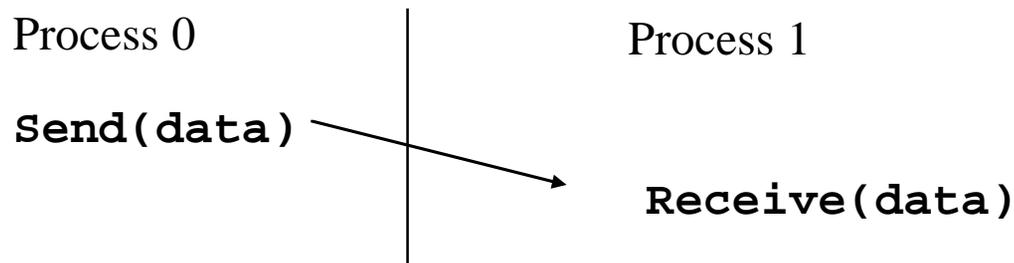
# Better Hello (Fortran)

```
program main
use MPI
integer ierr, rank, size

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierr )
print *, 'I am ', rank, ' of ', size
call MPI_FINALIZE( ierr )
end
```

# MPI Basic Send/Receive

- We need to fill in the details in



- Things that need specifying:
  - How will “data” be described?
  - How will processes be identified?
  - How will the receiver recognize/screen messages?
  - What will it mean for these operations to complete?

# MPI is Simple

- Many parallel programs can be written using just these six functions, only two of which are non-trivial:
  - **MPI\_INIT**
  - **MPI\_FINALIZE**
  - **MPI\_COMM\_SIZE**
  - **MPI\_COMM\_RANK**
  - **MPI\_SEND**
  - **MPI\_RECV**
- Point-to-point (send/recv) isn't the only way...

# Introduction to Collective Operations in MPI

- Collective operations are called by all processes in a communicator.
- **MPI\_BCAST** distributes data from one process (the root) to all others in a communicator.
- **MPI\_REDUCE** combines data from all processes in communicator and returns it to one process.
- In many numerical algorithms, **SEND/RECEIVE** can be replaced by **BCAST/REDUCE**, improving both simplicity and efficiency.

# Example: PI in Fortran - 1

```
program main
use MPI
double precision  PI25DT
parameter (PI25DT = 3.141592653589793238462643d0)
double precision  mypi, pi, h, sum, x, f, a
integer n, myid, numprocs, i, ierr
c                                     function to integrate
f(a) = 4.d0 / (1.d0 + a*a)
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
10  if ( myid .eq. 0 ) then
      write(6,98)
98   format('Enter the number of intervals: (0 quits)')
      read(5,99) n
99   format(i10)
endif
```

# Example: PI in Fortran - 2

```
      call MPI_BCAST( n, 1, MPI_INTEGER, 0,
+                  MPI_COMM_WORLD, ierr)
c                               check for quit signal
      if ( n .le. 0 ) goto 30
c                               calculate the interval size
      h = 1.0d0/n
      sum = 0.0d0
      do 20 i = myid+1, n, numprocs
        x = h * (dble(i) - 0.5d0)
        sum = sum + f(x)
20    continue
      mypi = h * sum
c                               collect all the partial sums
      call MPI_REDUCE( mypi, pi, 1, MPI_DOUBLE_PRECISION,
+                  MPI_SUM, 0, MPI_COMM_WORLD,ierr)
```

# Example: PI in Fortran - 3

```
c          node 0 prints the answer
      if (myid .eq. 0) then
          write(6, 97) pi, abs(pi - PI25DT)
97      format(' pi is approximately: ', F18.16,
+          ' Error is: ', F18.16)
      endif
      goto 10
30     call MPI_FINALIZE(ierr)
      end
```

# Example: PI in C -1

```
#include "mpi.h"
#include <math.h>
int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i, rc;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x, a;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    while (!done) {
        if (myid == 0) {
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0) break;
```

# Example: PI in C - 2

```
h    = 1.0 / (double) n;
sum  = 0.0;
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += 4.0 / (1.0 + x*x);
}
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
           MPI_COMM_WORLD);
if (myid == 0)
    printf("pi is approximately %.16f, Error is %.16f\n",
           pi, fabs(pi - PI25DT));
}
MPI_Finalize();
return 0;
}
```

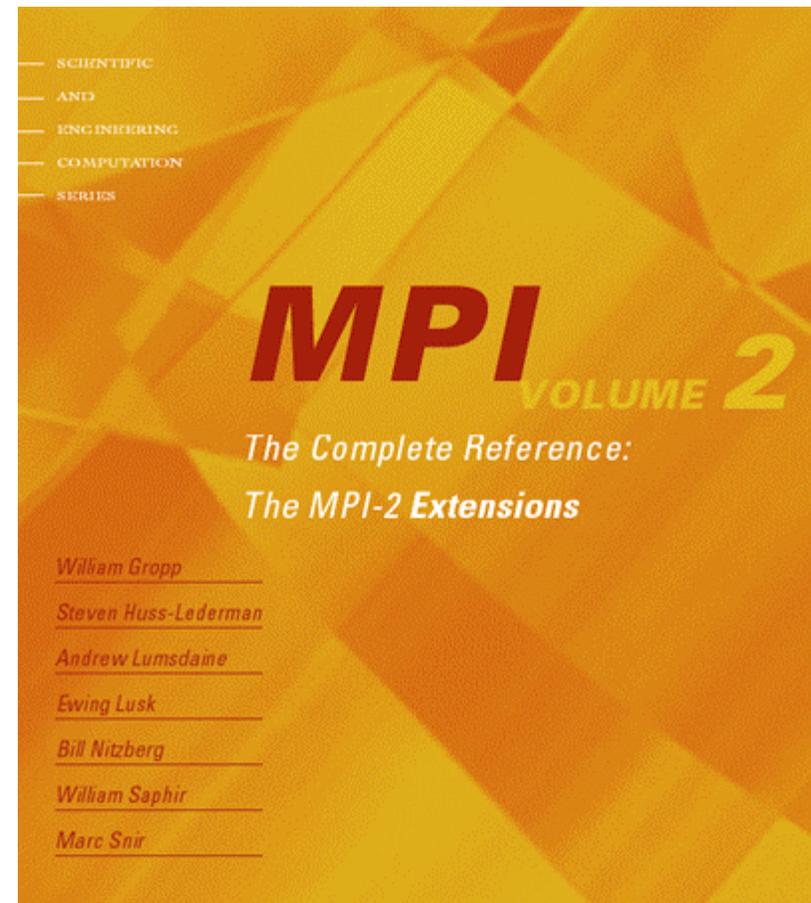
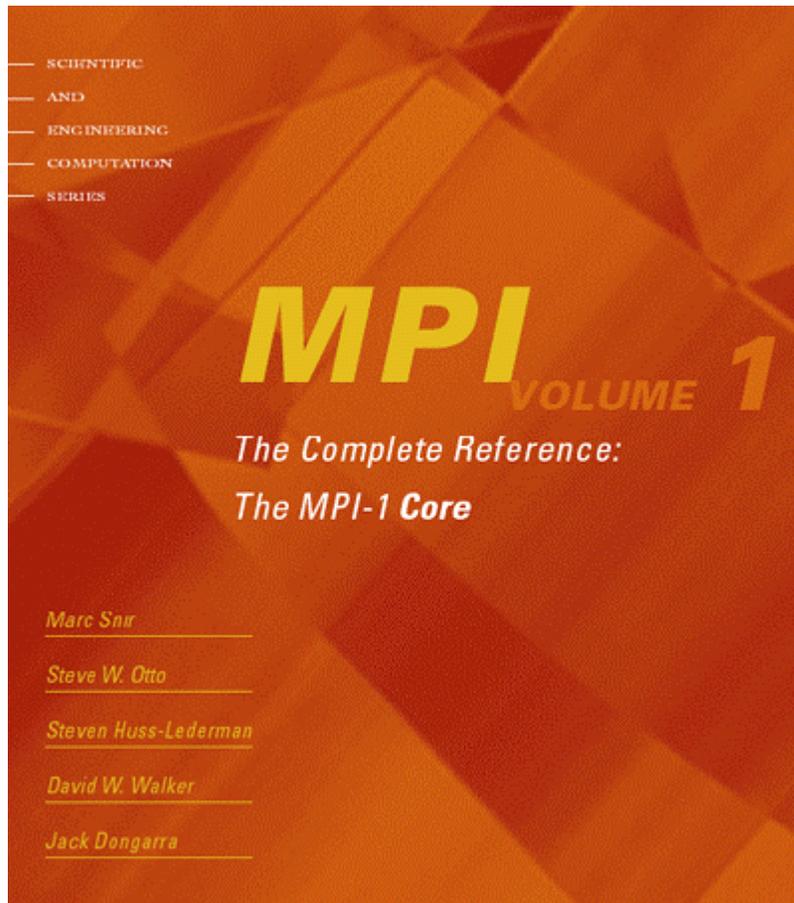
# Alternative set of 6 Functions for Simplified MPI

- **MPI\_INIT**
  - **MPI\_FINALIZE**
  - **MPI\_COMM\_SIZE**
  - **MPI\_COMM\_RANK**
  - **MPI\_BCAST**
  - **MPI\_REDUCE**
- What else is needed (and why)?

# MPI Information Sources

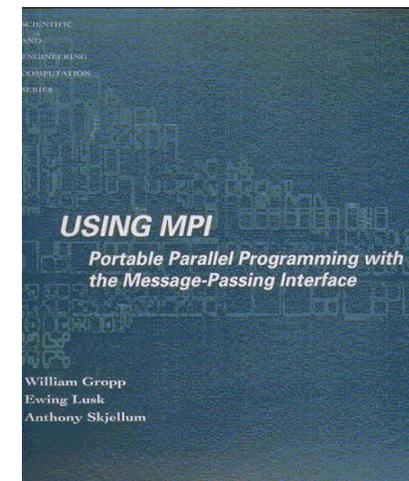
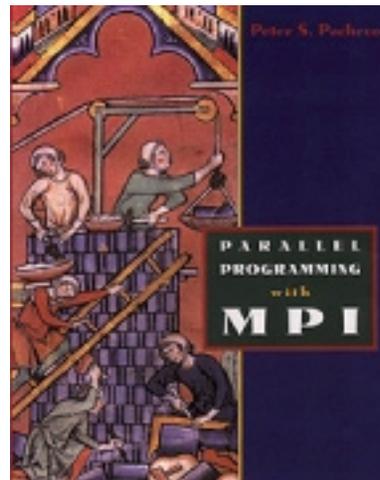
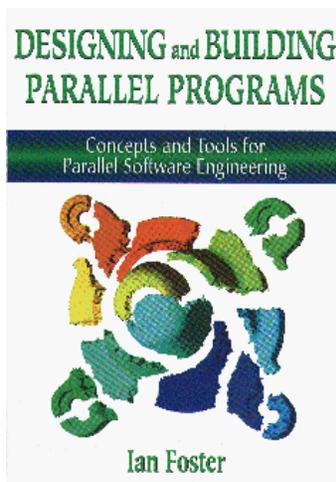
- Official Documents
  - <http://www.mpi-forum.org>
- Unofficial, but useful general MPI sites
  - <http://www.mcs.anl.gov/mpi>
  - <http://www.erc.msstate.edu/mpi>

# Book Versions of the Standard

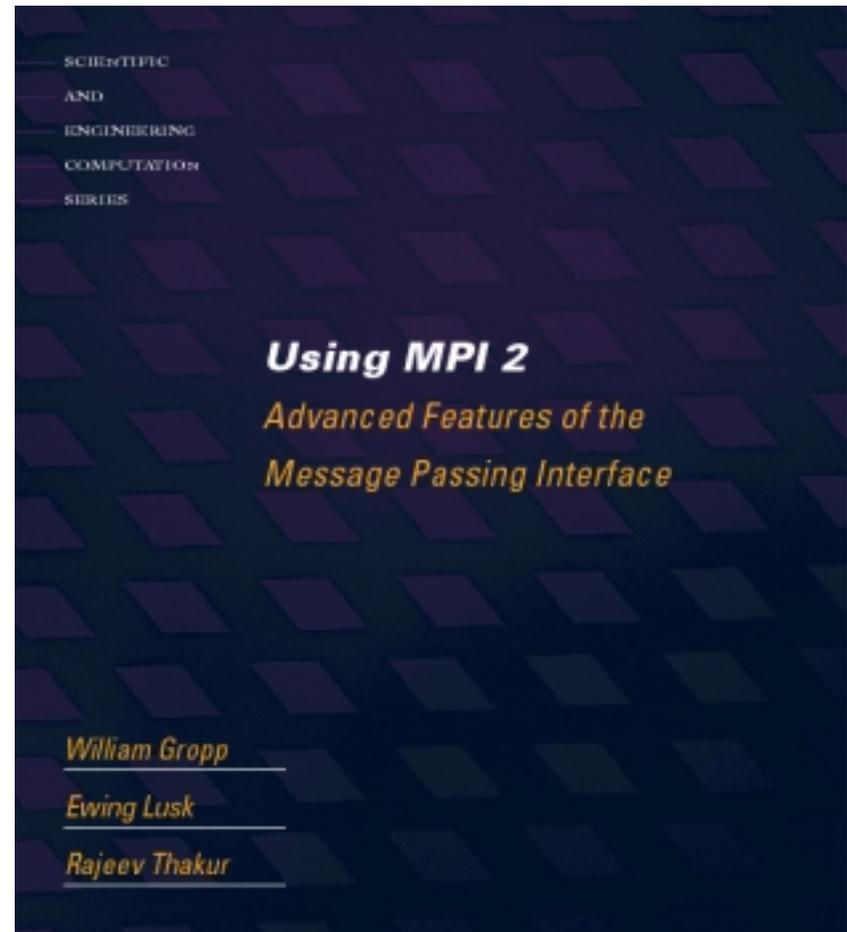
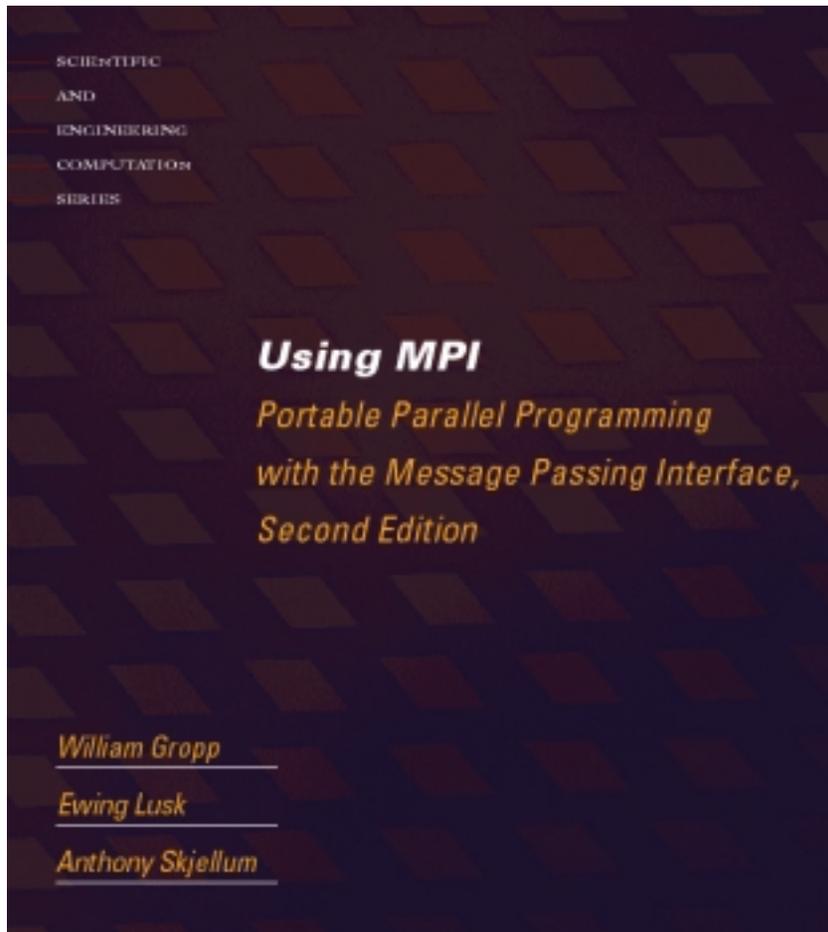


# Books on Programming with MPI

- *Designing and Building Parallel Programs*, by Ian Foster
- *Parallel Programming with MPI*, by Peter Pacheco
- *Using MPI*, by William Gropp, Ewing Lusk, Anthony Skjellum
- *Practical MPI Programming*, by Yukiya Aoyama and Jun Nakano (<http://www.redbooks.com>)



# Just Out!



# MPI Library Sampler

(From <http://www.mcs.anl.gov/mpi/libraries.html>)

- PETSc
  - sparse linear systems, nonlinear equations from PDE's, unconstrained optimization
- PGAPack
  - general-purpose genetic algorithm library
- ScaLAPack
  - parallel dense linear algebra
- MSG
  - structured grids in Fortran
- MPI-RGL
  - More regular grids

# More MPI Libraries

- Parallel Level 3 BLAS
  - parallel basic linear algebra subroutines
- GCL/MPI
  - Lisp interface for master/slave algorithms
- Aztec
  - solving linear systems with Newton-Krylov methods
- DOUG
  - Domain decomposition On Unstructured Grids
- FFTW
  - C FFT library
- MatheMatrix (commercial)
  - solving linear systems

# Sources of MPI Research Papers

- This conference
  - Krakow
  - Liverpool
  - Barcelona
- The MPI Developers Conference
  - Notre Dame
  - Atlanta
- Web sites
  - <http://www.mcs.anl.gov/mpi/papers/paperlist.html>
    - (mainly papers on using mpi, a few on research)
    - please send us pointers to mpi research papers

# Manager-Worker Algorithms

- One process coordinates the work of others.
- The manager divides up the total work into units for workers to work on and sends units to idle workers.
- When a worker completes a unit of work, it sends the result to the manager and requests more work to do.
- There is no communication among the workers.

# Demo

- Running the simple examples
- A sophisticated manager-worker example

## Programming Models for Clusters

- Parallelizing compilers do not exist!  
It is up to the user to handle this coarse grain, multi-computer with (sufficiently) high bandwidth and with high latency.
- HPF is available
- Message Passing Libraries: (The Standards)
  - MPI (M<sub>essage</sub> P<sub>assing</sub> I<sub>nterface</sub>)
  - PVM (P<sub>arallel</sub> V<sub>irtual</sub> M<sub>achine</sub>)
- Other systems
  - BNM — Beowulf Nominal Messages
  - BSP — Bulk Synchronous Protocol
  - DIPC — Distributed IPC
  - NVM-DSM — Network Virtual Memory
  - Threaded-C — Multithreaded Programming Model
  - RES, UPC, ...
- Use whatever your friends are using.

## Communication Takes Time

- A reasonable approximation to the total time for a message is

$$t_{\text{total}} = t_l + L/r$$

where  $t_l$  is the latency,

$L$  is the length of the message and

$r$  is the bandwidth.

- Define  $L_{1/2}$  be the length such that  $t_{\text{total}} = 2t_l$
- $L_{1/2} = rt_l$
- Use messages longer than  $L_{1/2}$  whenever possible.

## ESS Application Survey

- PPM (Piece-Wise Parabolic Method)
- N-body Tree Code
- PIC (Particle in Cell)
- SAR (Synthetic Aperture Radar)
- Spectral Methods: FFT's
- PARAMESH (Adaptive Mesh Refinement)
- HSEG (Hierarchical SEGmentation)
- MM5

## It is Easy to Get Started

Lots of programs only need the following:

- MSG\_INIT – join the party
- MSG\_FINALIZE – shut everything down
- MSG\_NUM\_PROCS – number of processors
- MSG\_MY\_NAME – the individuals name
- MSG\_SEND – point-to-point send
- MSG\_RECV – point-to-point recv

## Next Step: Collective Operations

Collective Operations involve all the processors:

- `MSG_BCAST` – one process sends data to all others
- `MSG_MANY_TO_MANY` – variation on broadcast
- `MSG_REDUCE` – data from all the processors is combined into a single value on one processor.
- Don't write your own
- Consider replacing `SEND/RECV` pairs with `BCAST/REDUCE`

## Next Step: Optimization

- Try not to use short messages
- Try not to use long messages
- Try to trade computation for communication
- Try to overlap communication with computation
  - `MPI_Isend` – send and immediately return
  - `MPI_Irecv` – receive and immediately return
  - `MPI_Wait` – wait on one of the above
  - `MPI_Test` – test one of the above

## Things to Consider

- No such thing as an embarrassingly parallel problem.
- Application Level parallelism sometimes yields the greatest scientific impact
- **Cycle stealing** is the most cost effective computing, but a cluster is probably easier to use.
- Balanced tightly coupled MPP's are probably easier to use, but a cluster may give you more compute power.
- Asymptotic results can be misleading—beware your intuition

# Automatically Tuned Linear Algebra Software - ATLAS

---

Clint Whaley

Antoine Petitet

Jack Dongarra

University of Tennessee

and

Oak Ridge National Laboratory

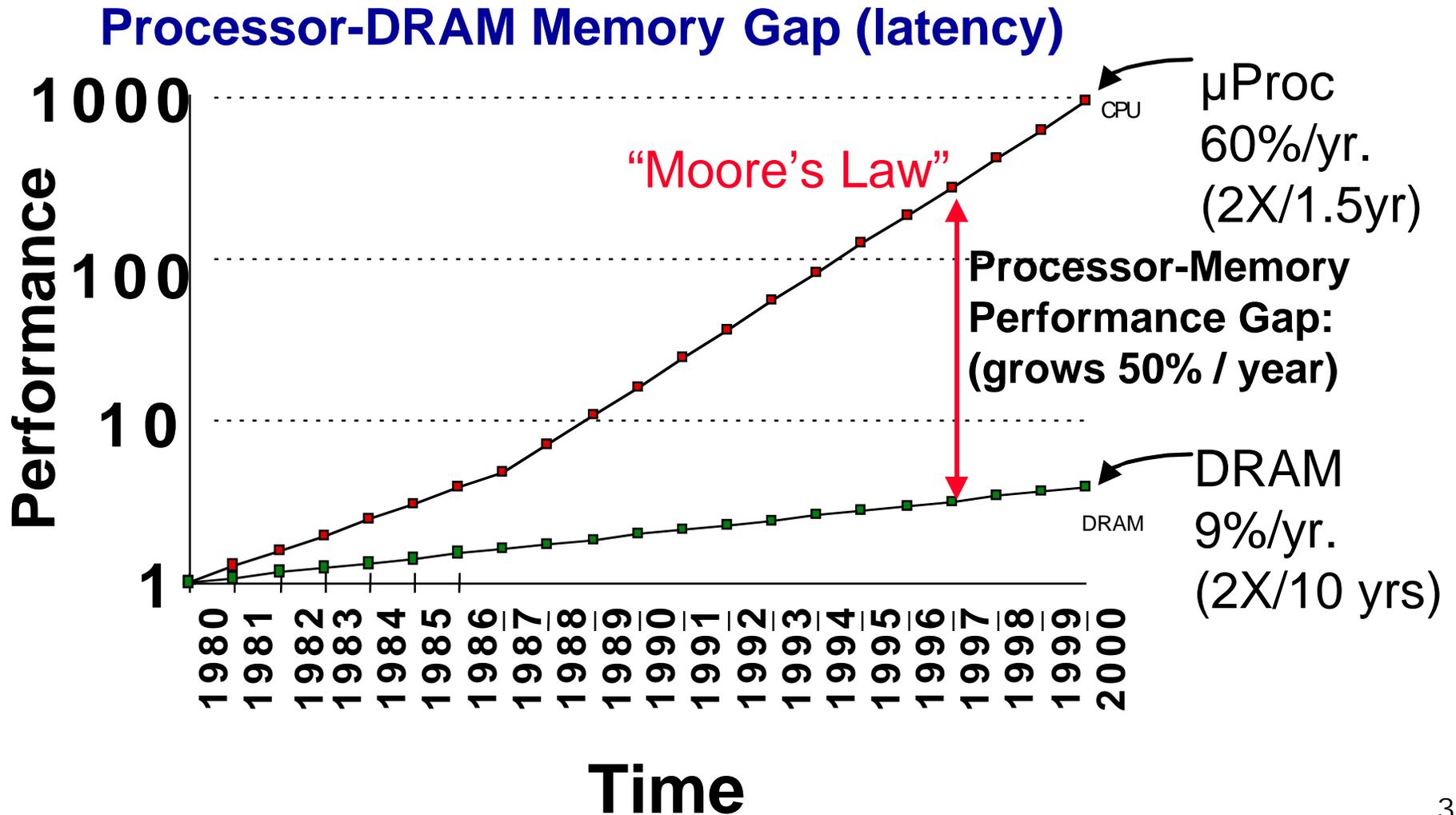
<http://www.netlib.org/utk/people/JackDongarra/>

# High-Performance Computing Directions

- ◆ Clustering of shared memory machines for scalability
  - „ Emergence of PC commodity systems
    - » Pentium/Alpha based, Linux or NT driven
    - » “Supercomputer performance at mail-order prices”
  - „ Beowulf-Class Systems (Linux+PC)
  - „ Distributed Shared Memory (clusters of processors connected)
  - „ Shared address space w/deep memory hierarchy
- ◆ Efficiency of message passing and data parallel programming
  - „ Helped by standards efforts such as PVM, MPI, Open-MP and HPF
- ◆ Many of the machines as a single user environments
- ◆ Pure COTS

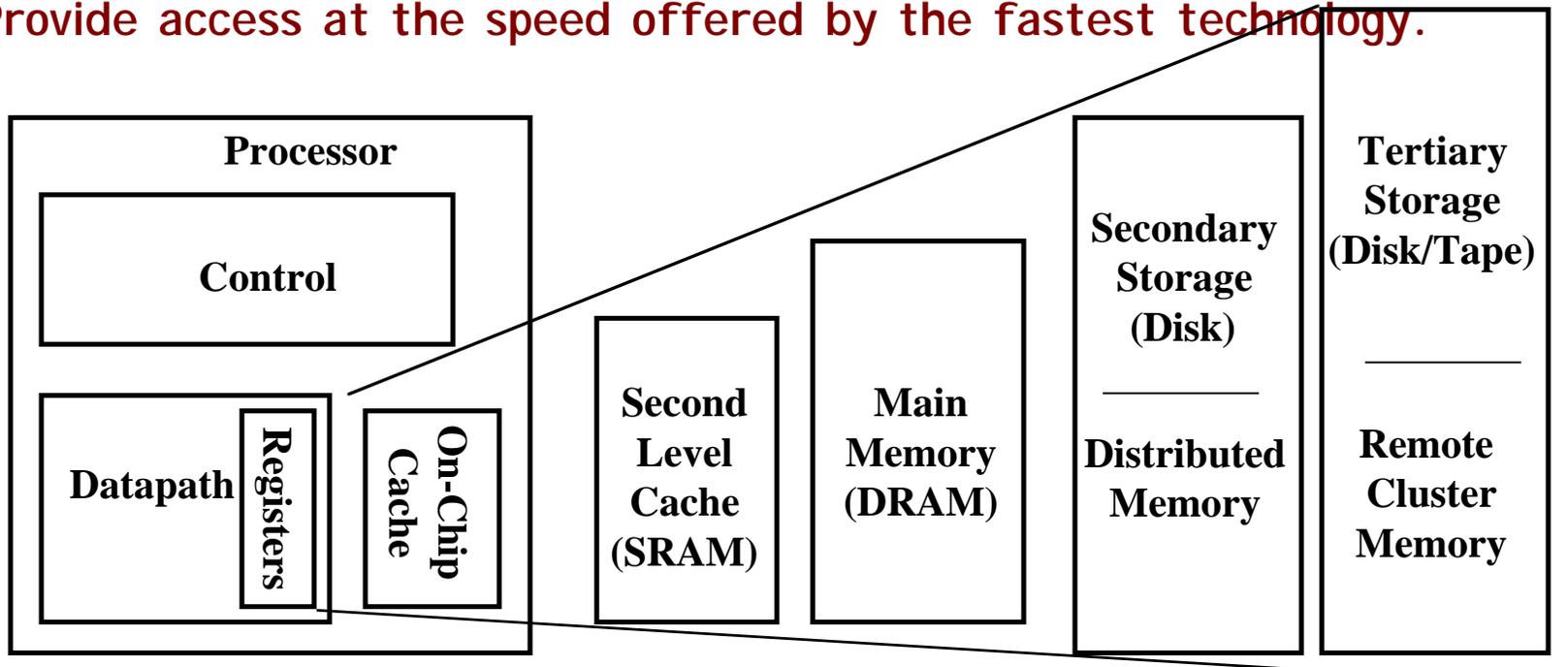


# Where Does the Performance Go? or Why Should I Care About the Memory Hierarchy?



# Memory Hierarchy

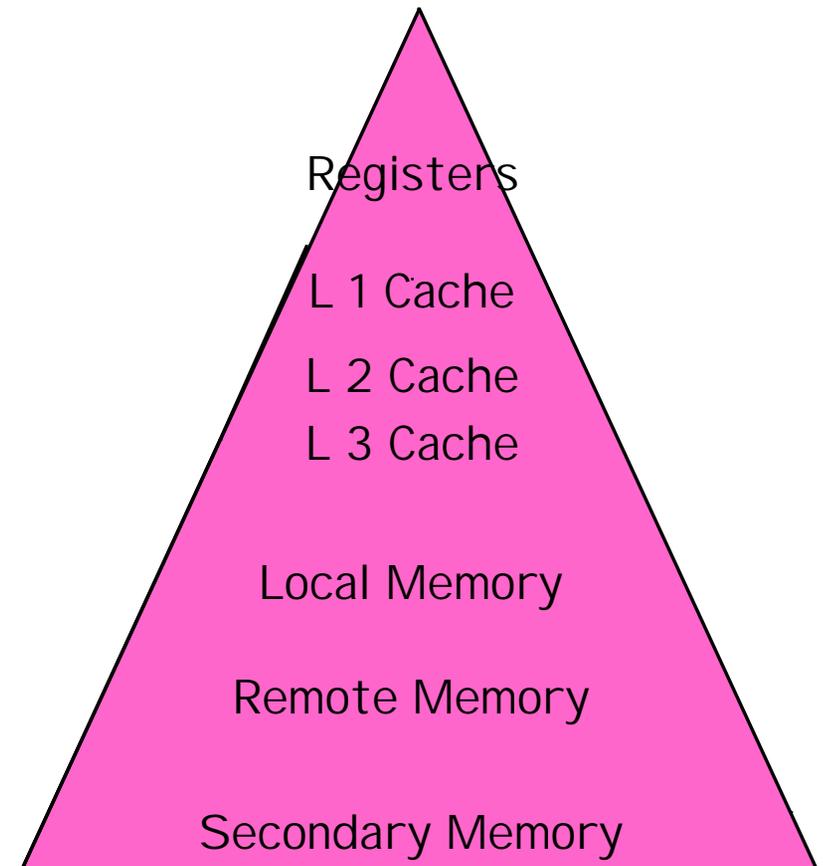
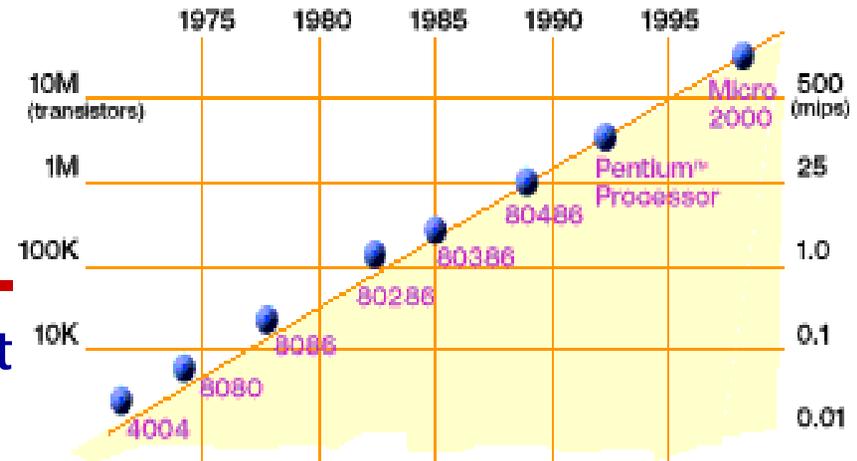
- ◆ By taking advantage of the principle of locality:
  - „ Present the user with as much memory as is available in the cheapest technology.
  - „ Provide access at the speed offered by the fastest technology.



<b>Speed (ns):</b> 1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
<b>Size (bytes):</b> 100s	Ks	Ms	100,000 s (.1s ms)	10,000,000 s (10s ms)
			Gs	Ts

# Performance Software

- ◆ Computing hardware doubles its speed every eighteen months.
- ◆ Yet it often takes more than a year for software to be optimized or "tuned" for performance on a newly released CPU.
- ◆ The job of optimizing software to exploit the special features of a given CPU has historically been an exercise in hand customization.



# Performance Issues - Cache & Bandwidth

---

- ◆ Performance instability
  - „ Small changes in the architecture may cause dramatic changes in delivered performance.
- ◆ Latency tolerant and bandwidth parsimonious algorithms and software are critical
  - „ Sometimes this means recompute rather than store/load
- ◆ Need to help the compiler
- ◆ Have a hard time today getting performance
  - „ Only going to get harder
- ◆ Level 3 BLAS as a starting point

# How To Get Performance From Commodity Processors?

---

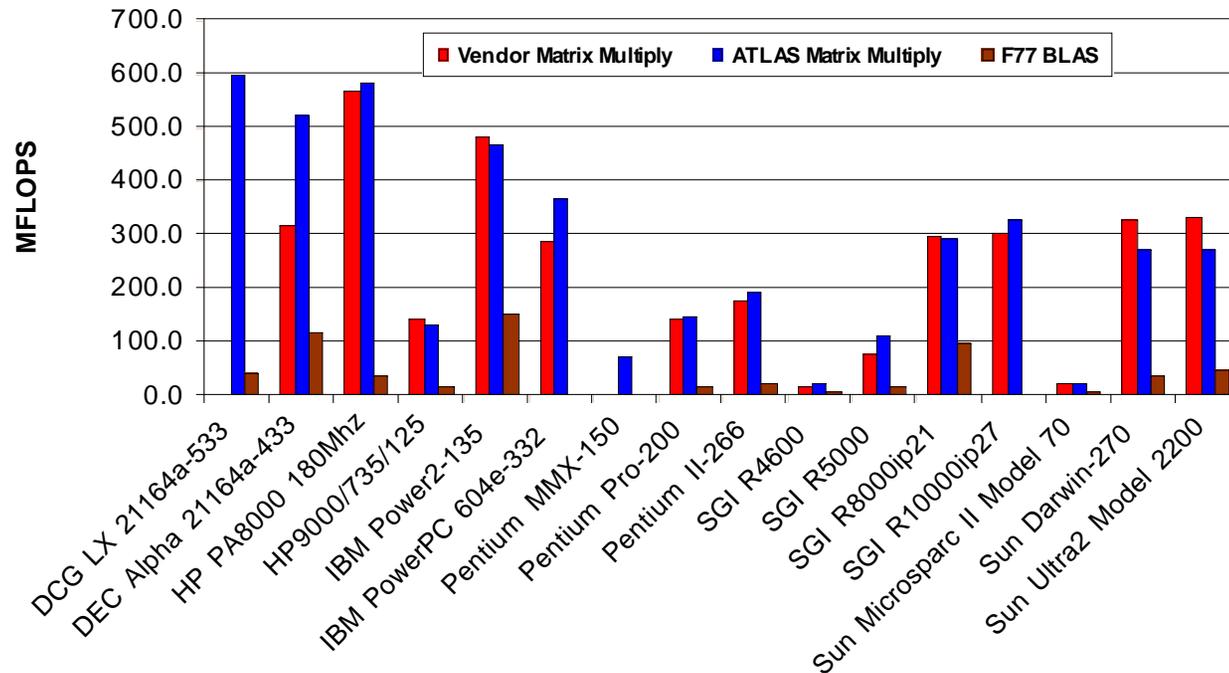
- ◆ Today's processors can achieve high-performance, but this requires extensive machine-specific hand tuning.
- ◆ Hardware and software have a large design space w/many parameters
  - „ Blocking sizes, loop nesting permutations, loop unrolling depths, software pipelining strategies, register allocations, and instruction schedules.
  - „ Complicated interactions with the increasingly sophisticated micro-architectures of new microprocessors.
- ◆ About a year ago no tuned BLAS for Pentium for Linux.
- ◆ Need for quick/dynamic deployment of optimized routines.
- ◆ ATLAS - Automatic Tuned Linear Algebra Software
  - „ PhiPac from Berkeley
  - „ FFTW from MIT (<http://www.fftw.org>)

# ATLAS

---

- ◆ An adaptive software architecture
  - „ High-performance
  - „ Portability
  - „ Elegance
  
- ◆ ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.

# ATLAS



- ◆ **ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.**

# Why ATLAS Is Fast?

---

- ◆ ATLAS does not implement a single fixed algorithm.
- ◆ The code is generated by a program that tests, probes, and runs 100's of experiments on the target sw/hw architecture.
- ◆ During installation the program generator determines an efficient implementation
  - „ measures the speed of different code strategies and chooses the best using an adaptive procedure.
- ◆ This leads to a new model of high performance programming in which performance critical code is machine generated using parameter optimization.

# Why Adaptive Programs?

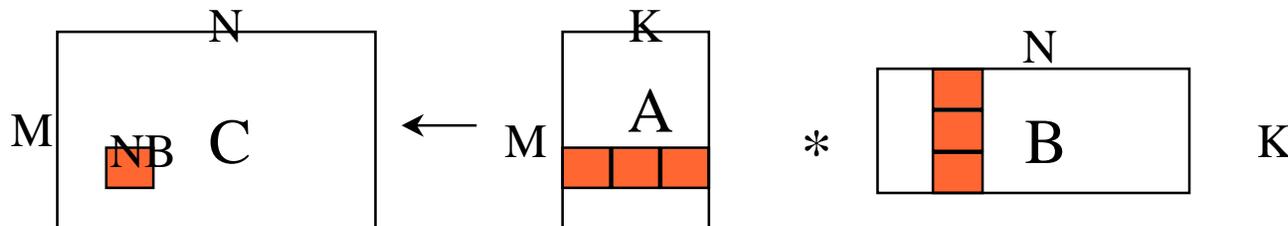
---

- ◆ Critical software that realizes near peak performance requires detail knowledge of a host of interlocking and competing factors.
- ◆ Performance can differ by factors of 10, even 100.
- ◆ Tuning even the simplest operation generally requires an intense and sustained effort by highly technically advanced programmers.
- ◆ Present day architectures are complicated, making predictability difficult, experimentation is a reliable way to achieve near peak performance.

# Adaptive Approach for Level 3 BLAS

---

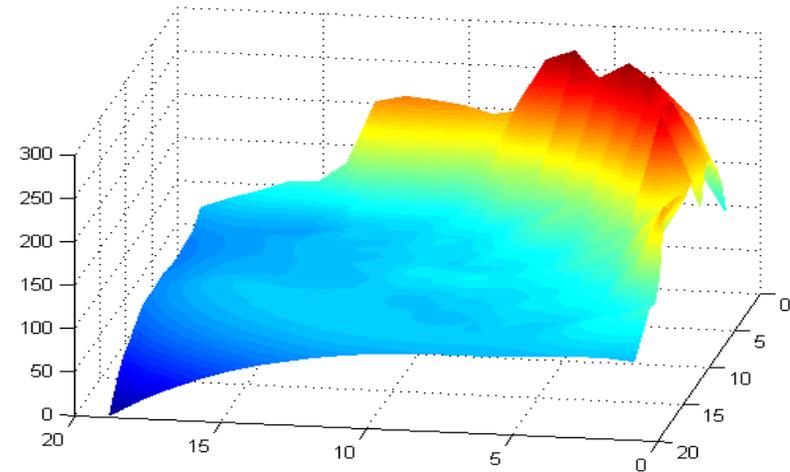
- ◆ Do a parameter study of the operation on the target machine, done once.
- ◆ Only generated code is on-chip multiply
- ◆ BLAS operation written in terms of generated on-chip multiply
- ◆ All transpose cases coerced through data copy to 1 case of on-chip multiply
  - „ Only 1 case generated per platform



# Code Generation Strategy

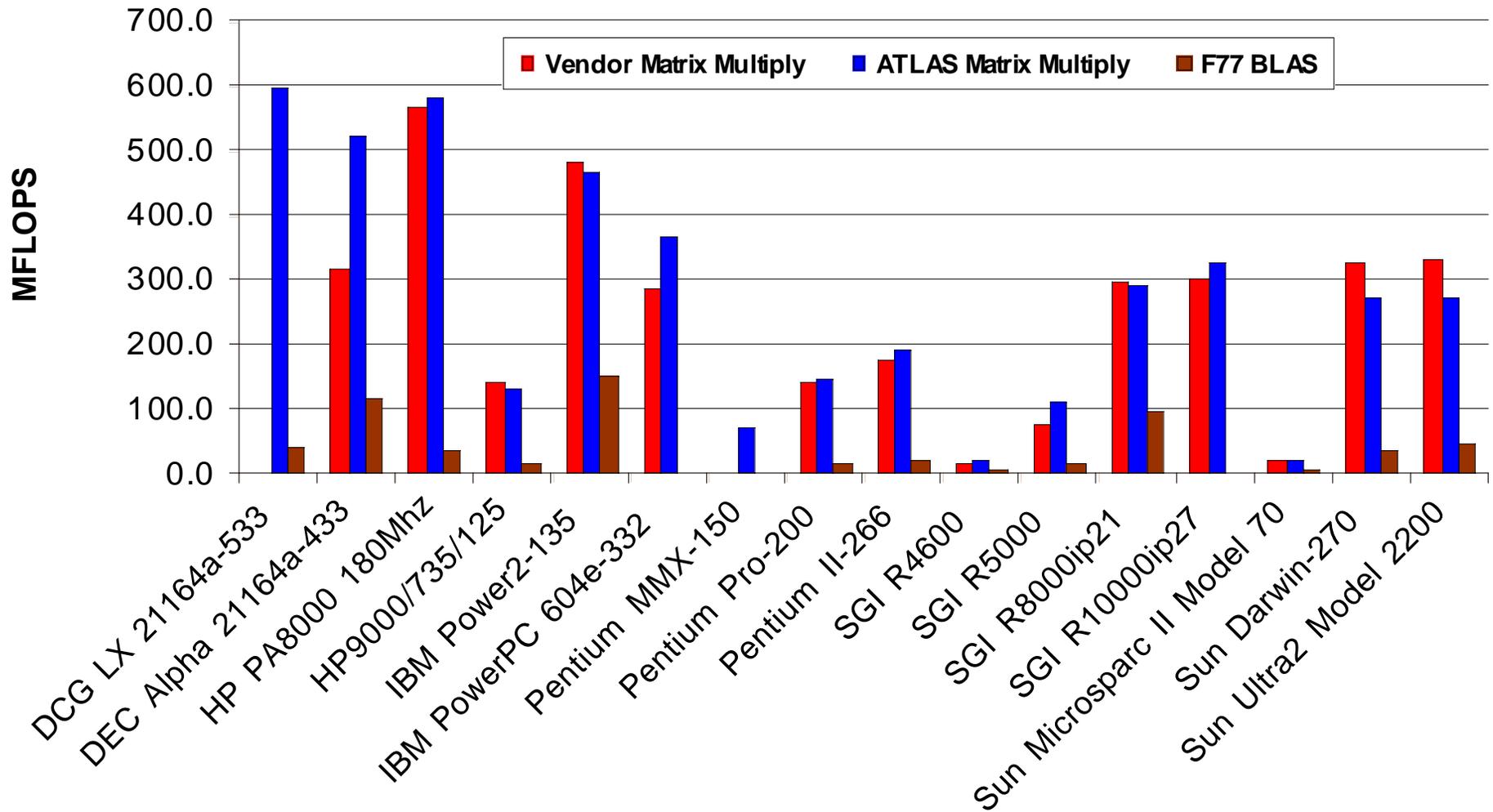
---

- ◆ On-chip multiply optimizes for:
  - „ TLB access
  - „ L1 cache reuse
  - „ FP unit usage
  - „ Memory fetch
  - „ Register reuse
  - „ Loop overhead minimization
- ◆ Takes a couple of hours to run.

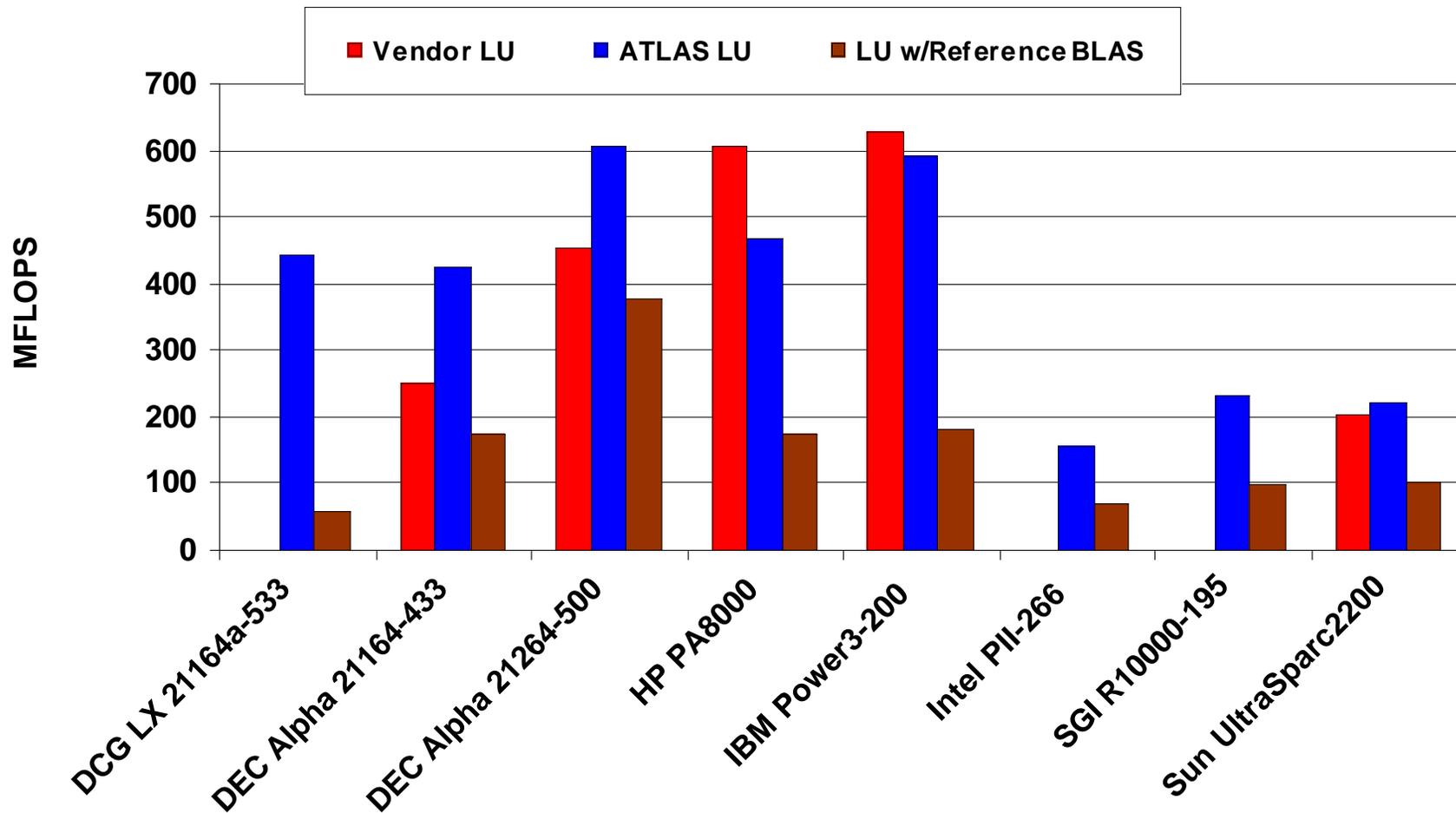


- ◆ Code is iteratively generated & timed until optimal case is found. We try:
  - „ Differing NBs
  - „ Breaking false dependencies
  - „ M, N and K loop unrolling

# ATLAS 500x500 DGEMM Across Various Architectures



# 500 x 500 LU Right-Looking

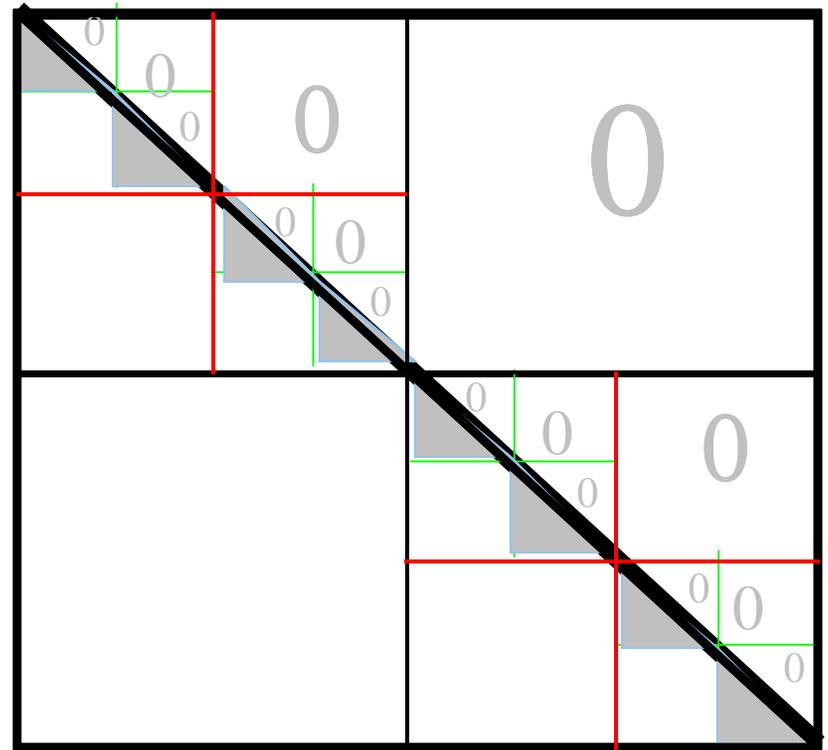


# Recursive Approach for Other Level 3 BLAS

---

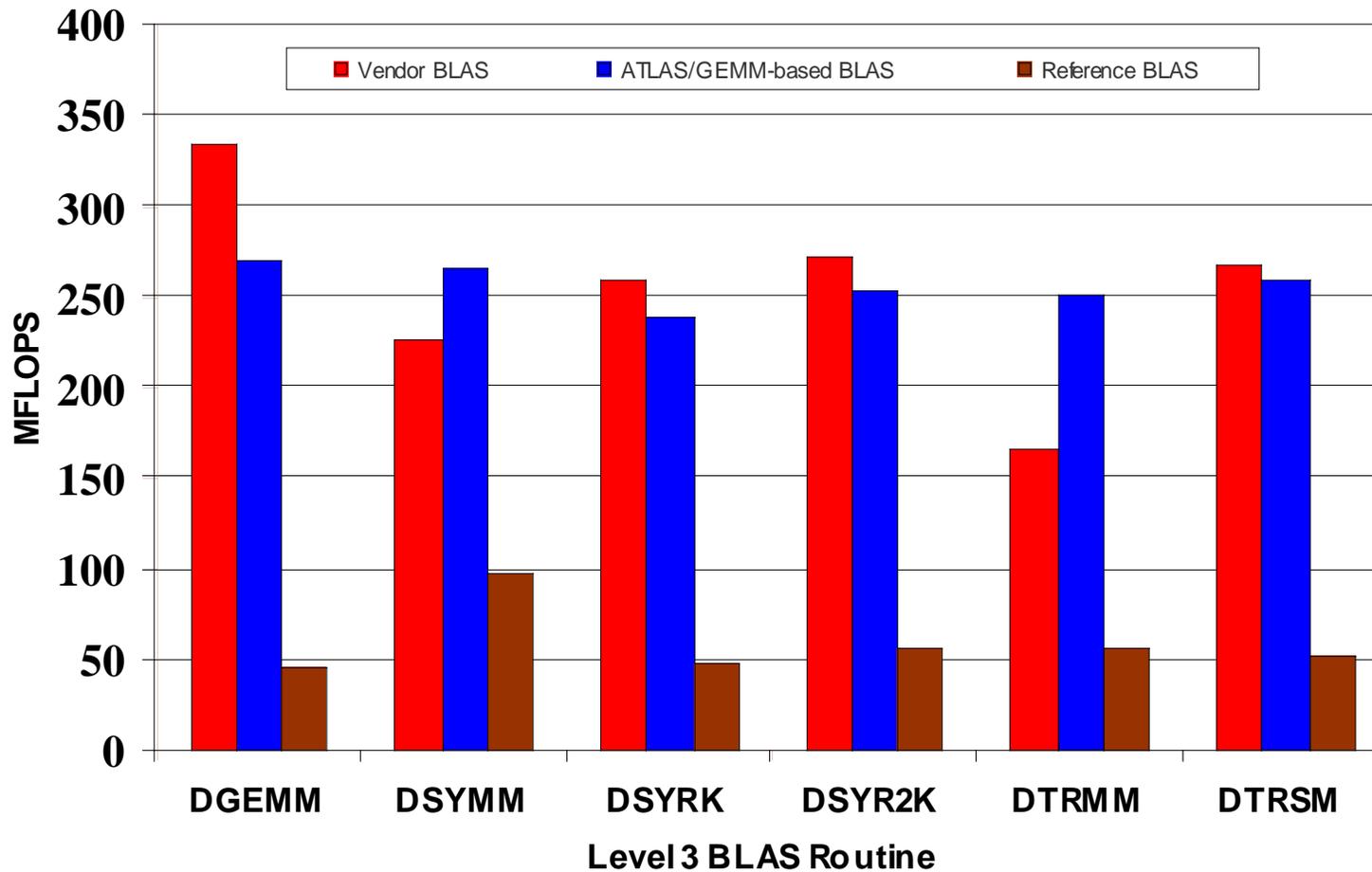
- ◆ Recur down to L1 cache block size
- ◆ Need kernel at bottom of recursion
  - „ Use gemm-based kernel for portability

Recursive TRMM



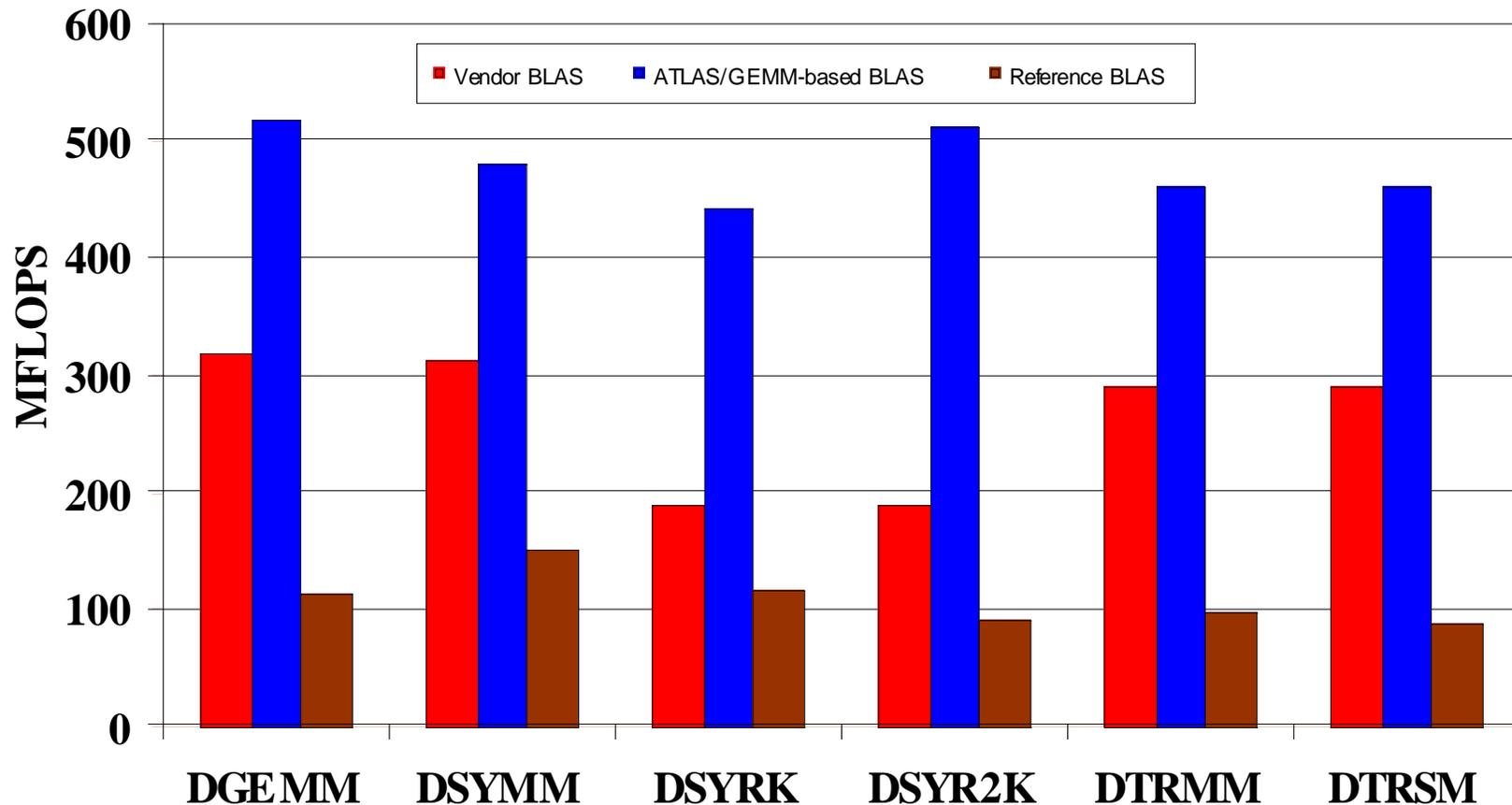
# 500x500 Recursive BLAS on UltraSparc 2200

---



# 500x500 Recursive BLAS on 433Mhz DEC 21164

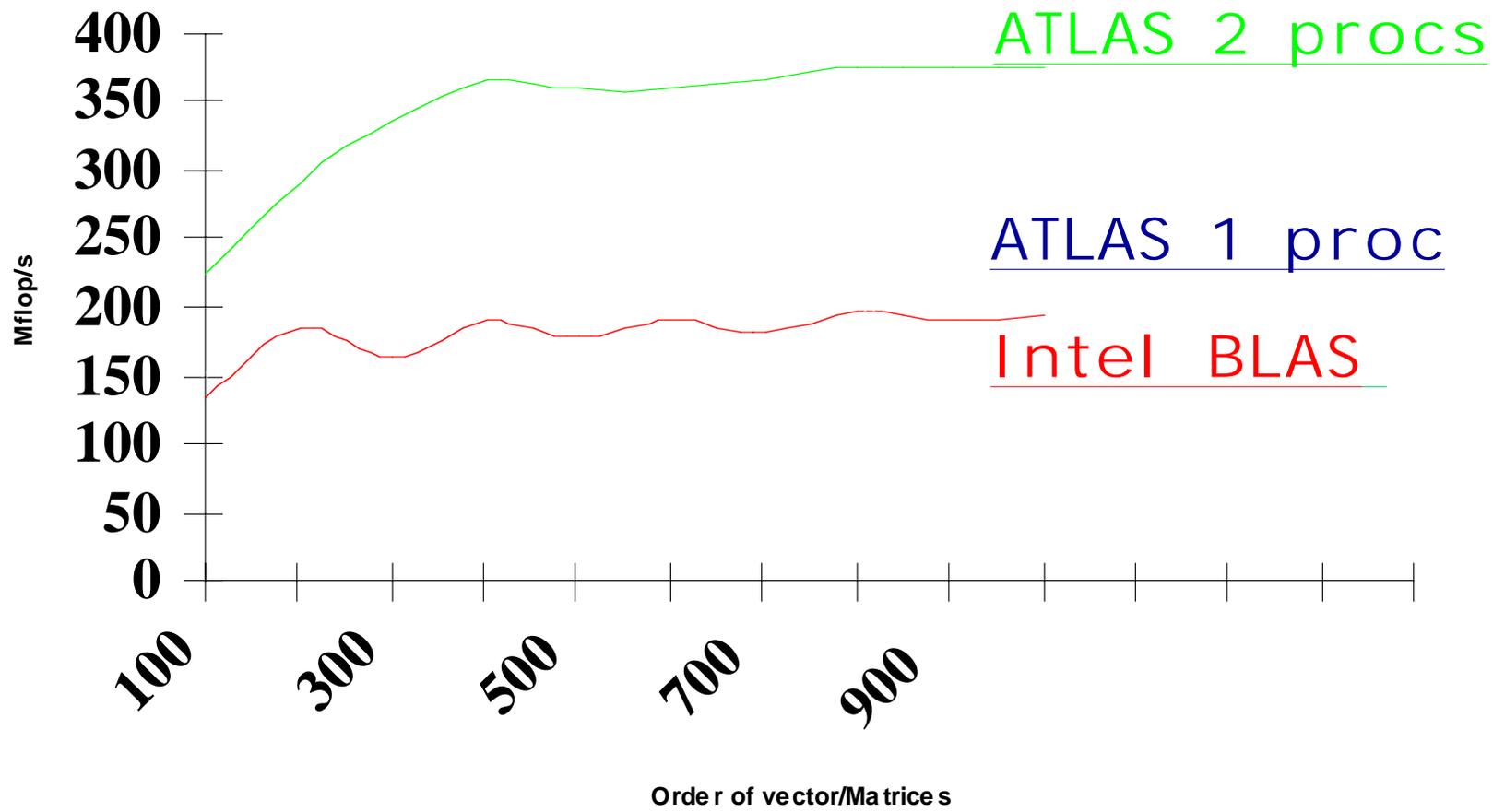
---



# Multithreaded BLAS for Performance

---

Intel Pentium II 300 MHz



# ATLAS

---

- ◆ Needs a reasonable C compiler and is focused on super scalar (RISC) architectures.
- ◆ Available today: [www.netlib.org/atlas/](http://www.netlib.org/atlas/)
- ◆ Keep a repository of kernels for specific machines.
- ◆ Extend work to allow sparse matrix operations
- ◆ Extend work to include arbitrary code segments

# Future Plans for ATLAS

---

- ◆ Level 1 and 2 implementations
- ◆ Threading
- ◆ Runtime adaptation
  - „ Sparsity analysis
  - „ Iterative code improvement
- ◆ Adaptive libraries
- ◆ Specialization for user applications
- ◆ Extend these ideas to Java directly
  - „ Java LAPACK

# Contributors to These Ideas

---

- ◆ Antoine Petitet, UTK
- ◆ Clint Whaley, UTK

- ◆ For additional information see...

[icl.cs.utk.edu/](http://icl.cs.utk.edu/)

[www.netlib.org/atlas/](http://www.netlib.org/atlas/)

[www.netlib.org/utk/people/JackDongarra/](http://www.netlib.org/utk/people/JackDongarra/)

**ATLAS received an R&D 100 Award this year**