HPC Best Practices

Ontario Summer School on
High Performance Computing

Scott Northrup
SciNet HPC Consortium
Compute Canada

June 13th, 2013

Outline

Acknowledgments

Contributing Material

- HPC Best Practices - G. Baolai, SHARCNET
- The Parallel File System and I/O - R. van Zon, SciNet
- Monitoring Job Efficiently - R. van Zon, SciNet
- Profiling and Tuning - L. J. Dursi, SciNet
- Tuning MPI - L. J. Dursi, SciNet

Outline

Workflow

Typical Simulation/Analysis Work-flow

- pre-process (grid creation, partitioning)
- solve/analysis
- postprocessing (data-mining, generate plots)

Workflow

Typical Simulation/Analysis Work-flow

- pre-process (grid creation, partitioning)
- solve/analysis
- postprocessing (data-mining, generate plots)

Automate

- learn and use script languages (bash, python)
- use scheduler efficiently (job size, dependencies)
- add data management into workflow from beginning

Batch computing

SciNet systems are batch compute clusters

- Computing by submitting batch jobs to the scheduler.
- When you submit a job, it gets placed in a queue.
- Job priority is based on allocation and fairshare.
- When sufficient nodes are free to execute a job, it starts the job on the appropriate compute nodes.
- Jobs remain 'idle' until resources become available.
- Jobs can be temporarily 'blocked' if you submit too much.

Batch computing

Components

Torque: Resource manager providing control over batch jobs and distributed compute nodes.

Moab: A policy-based job scheduler and event engine that enables utility-based computing for clusters.

Fairshare: Mechanism using past utilization for prioritization.

Job cycle

Preparation

- Compile
- Test on devel node
- Determine resources
- Write job script

llsubmit
qsub

Job cycle

Preparation

- Compile
- Test on devel node
- Determine resources
- Write job script

llsubmit
qsub

Monitor

- Job queued?
- When will it run?
- What else is queued?
- Efficiency?

qstat -f
checkjob
showstart
showbf
showq

Job cycle

Preparation

- Compile
- Test on devel node
- Determine resources
- Write job script

llsubmit
qsub

Monitor

- Job queued?
- When will it run?
- What else is queued?
- Efficiency?

qstat -f
checkjob
showstart
showbf
showq

Control

- Cancel job
- Ssh to nodes
- Interactive jobs
- Debug queue

canceljob
top
qsub -I
qsub -q debug

Job cycle

Preparation

- Compile
- Test on devel node
- Determine resources
- Write job script

llsubmit
qsub

Monitor

- Job queued?
- When will it run?
- What else is queued?
- Efficiency?

qstat -f
checkjob
showstart
showbf
showq

Control

- Cancel job
- Ssh to nodes
- Interactive jobs
- Debug queue

canceljob
top
qsub -I
qsub -q debug

Reports

- Check .o/.e *jobname*.{o,e}
- usage stats on ccdb webpage

showstats -u

Monitoring not-yet-running jobs

qstat and checkjob

- Show torque status right away on GPC: qstat
- Show moab status (better): checkjob *jobid*
- See more details of the job: checkjob -v *jobid*
  (e.g., why is my job blocked?)

showq

- See all the jobs in the queue: showq (from gpc or tcs)
- See your jobs in the queue: showq -u *user*

showstart and showbf

- Estimate when a job may start: showbf
- Estimate when a queued job may start: showstart *jobid*
- Estimates only!

Monitoring running jobs

checkjob

- checkjob *jobid*

showq

- showq -r -u $USER

ssh

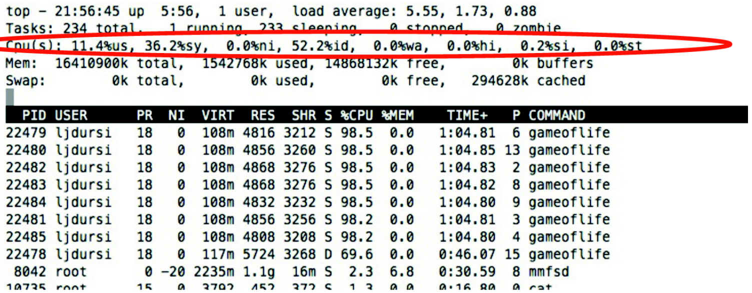
- ssh *node* (node name from checkjob)
- top: shows process state, memory and cpu usage

Job stdout/stderr files

- {jobname}.o{jobid}
- {jobname}.e{jobid}

Top example

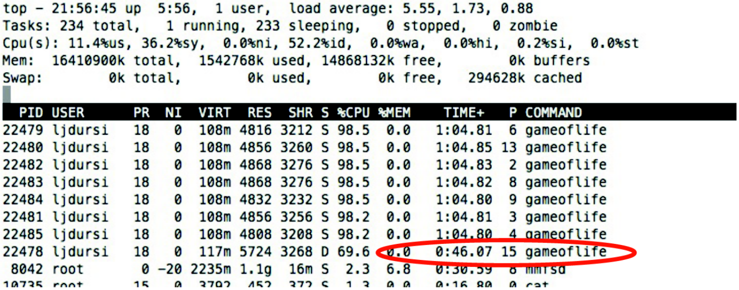
```
gpc-f103n084-$ ssh gpc-f109n001
gpc-f109n001-$ top
```



```
top - 21:56:45 up  5:56,  1 user,  load average: 5.55, 1.73, 0.88
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.4%us, 36.2%sy,  0.0%ni, 52.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  16410900k total,  1542768k used, 14868132k free,        0k buffers
Swap:        0k total,        0k used,        0k free,   294628k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
22479 ljdursi   18   0  108m 4816 3212 S 98.5  0.0   1:04.81  6 gameoflife
22480 ljdursi   18   0  108m 4856 3260 S 98.5  0.0   1:04.85 13 gameoflife
22482 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.83  2 gameoflife
22483 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.82  8 gameoflife
22484 ljdursi   18   0  108m 4832 3232 S 98.5  0.0   1:04.80  9 gameoflife
22481 ljdursi   18   0  108m 4856 3256 S 98.2  0.0   1:04.81  3 gameoflife
22485 ljdursi   18   0  108m 4808 3208 S 98.2  0.0   1:04.80  4 gameoflife
22478 ljdursi   18   0  117m 5724 3268 D 69.6  0.0   0:46.07 15 gameoflife
 8042 root       0 -20 2235m 1.1g  16m S  2.3  6.8   0:30.59  8 mmfsd
10735 root      15   0  3792  452  372 S  1.3  0.0   0:16.80  0 cat
```

Top example

```
gpc-f103n084-$ ssh gpc-f109n001
gpc-f109n001-$ top
```



```
top - 21:56:45 up  5:56,  1 user,  load average: 5.55, 1.73, 0.88
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.4%us, 36.2%sy,  0.0%ni, 52.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  16410900k total,  1542768k used, 14868132k free,       0k buffers
Swap:        0k total,        0k used,        0k free,   294628k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
22479 ljdursi   18   0  108m 4816 3212 S 98.5  0.0   1:04.81  6 gameoflife
22480 ljdursi   18   0  108m 4856 3260 S 98.5  0.0   1:04.85 13 gameoflife
22482 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.83  2 gameoflife
22483 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.82  8 gameoflife
22484 ljdursi   18   0  108m 4832 3232 S 98.5  0.0   1:04.80  9 gameoflife
22481 ljdursi   18   0  108m 4856 3256 S 98.2  0.0   1:04.81  3 gameoflife
22485 ljdursi   18   0  108m 4808 3208 S 98.2  0.0   1:04.80  4 gameoflife
22478 ljdursi   18   0  117m 5724 3268 D 69.6  0.0   0:46.07 15 gameoflife
 8042 root       0 -20 2235m 1.1g  16m S  2.3  6.8   0:30.59  8 mmfsd
10735 root      15   0  3792  452  372 S  1.3  0.0   0:16.80  0 cat
```

Control

canceljob

- If you spot a mistake: canceljob *jobid*

qsub for interactive and debug jobs

- `-I`:
  - Interactive
  - After qsub, waits for jobs to start.
  - Usually combined with:
- -q debug:
  - Debug queue has 10 nodes reserved for short jobs.
  - You can get 1 node for 2 hours, but also
  - 8 nodes, for half an hour.

Job output/error files (*.e / *.o)

```
----------------------------------------
Begin PBS Prologue Tue Sep 14 17:14:48 EDT 2010 1284498888
Job ID:      3053514.gpc-sched
Username:    ljdursi
Group:       scinet
Nodes:       gpc-f134n009 gpc-f134n010 gpc-f134n011 gpc-f134n012
gpc-f134n043 gpc-f134n044 gpc-f134n045 gpc-f134n046 gpc-f134n047 gpc-f134n048
[...]
End PBS Prologue Tue Sep 14 17:14:50 EDT 2010 1284498890
----------------------------------------
[ Your job's output here... ]
----------------------------------------
Begin PBS Epilogue Tue Sep 14 17:36:07 EDT 2010 1284500167
Job ID:      3053514.gpc-sched
Username:    ljdursi
Group:       scinet
Job Name:    fft_8192_procs_2048
Session:     18758
Limits:      neednodes=256:ib:ppn=8,nodes=256:ib:ppn=8,walltime=01:00:00
Resources:   cput=713:42:30,mem=3463854672kb,vmem=3759656372kb,walltime=00:21:07
Queue:       batch_ib
Account:
Nodes:   gpc-f134n009 gpc-f134n010 gpc-f134n011 gpc-f134n012 gpc-f134n043
[...]
Killing leftovers...
gpc-f141n054:    killing gpc-f141n054 12412

End PBS Epilogue Tue Sep 14 17:36:09 EDT 2010 1284500169
----------------------------------------
```

Outline

Data Management

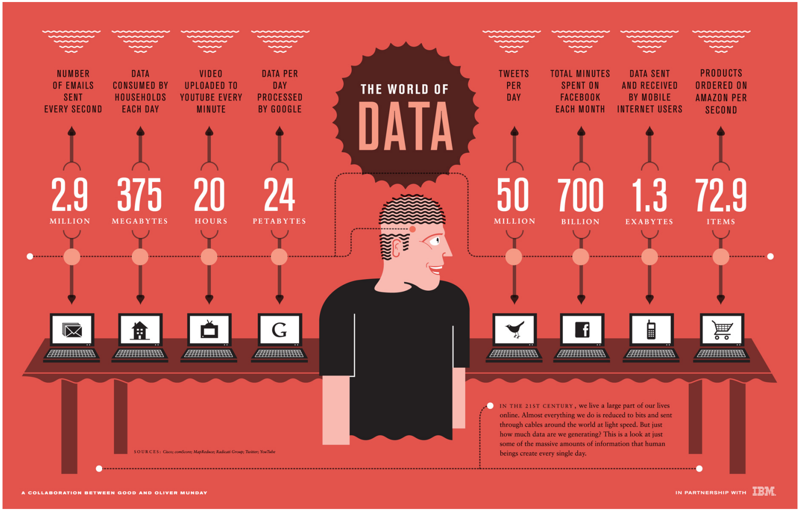
To much of a good thing?

- Increase in computing power makes simulations larger/more frequent
- Increase in sensor technology makes experiments/observations larger
  - Large Hadron: $\sim$ 50-100 PB to date (4 years)
  - Square Kilometer Array: $\sim$ 1 EB /day !
- Data sizes that used to be measured in MB/GB now measured in TB/PB.
- Easier to make big data than to do something useful with it!
- Data access is the now the bottleneck.

Big Data

Ben Chams - Fotolia

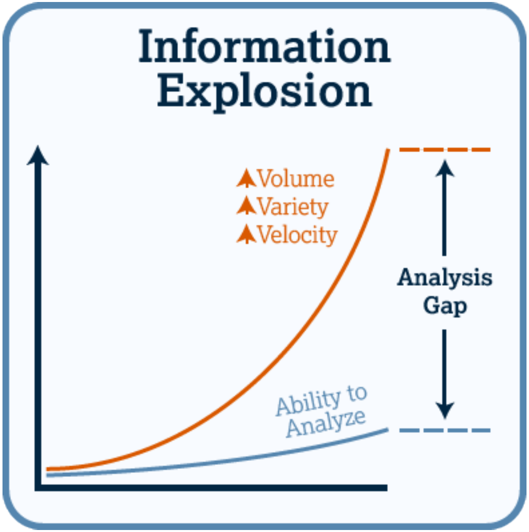
Big Data


THE WORLD OF DATA
NUMBER OF EMAILS SENT EVERY SECOND
2.9 MILLION
DATA CONSUMED BY HOUSEHOLDS EACH DAY
375 MEGABYTES
VIDEO UPLOADED TO YOUTUBE EVERY MINUTE
20 HOURS
DATA PER DAY PROCESSED BY GOOGLE
24 PETABYTES
TWEETS PER DAY
50 MILLION
TOTAL MINUTES SPENT ON FACEBOOK EACH MONTH
700 BILLION
DATA SENT AND RECEIVED BY MOBILE INTERNET USERS
1.3 EXABYTES
PRODUCTS ORDERED ON AMAZON PER SECOND
72.9 ITEMS
G
f
A COLLABORATION BETWEEN GOOD AND OLIVER MUNDAY
IN PARTNERSHIP WITH IBM

IN THE 21ST CENTURY, we live a large part of our lives online. Almost everything we do is reduced to bits and sent through cables around the world at light speed. But just how much data are we generating? This is a look at just some of the massive amounts of information that human beings create every single day.

SOURCES: *Cisco; comScore; MapReduce; Radicati Group; Twitter; YouTube*

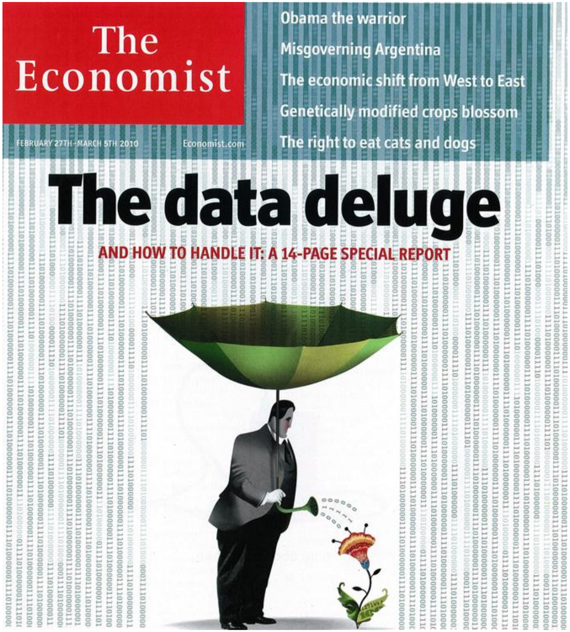
Big Data


Information Explosion
Volume
Variety
Velocity
Analysis Gap
Ability to Analyze

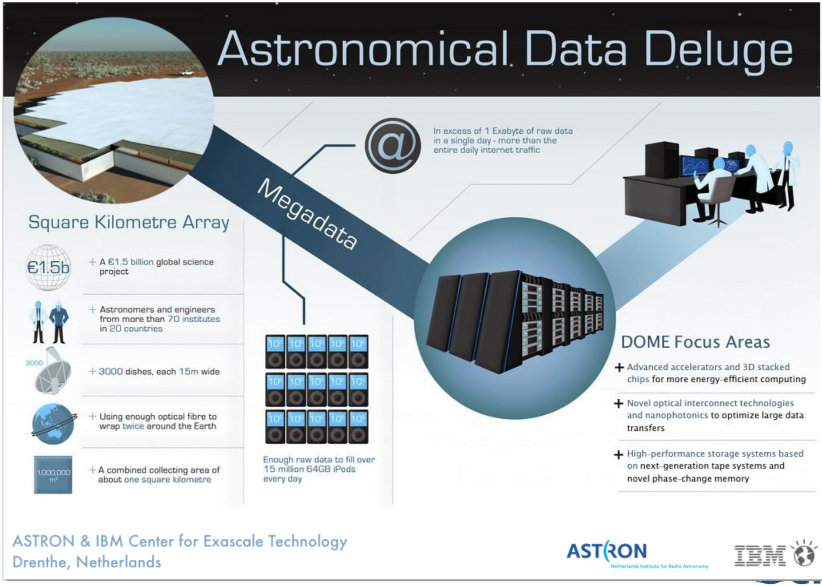
Big Data

COMMUNICATIONS OF THE ACM
CACM.ACM.ORG
12/08 VOL.51 NO.12
Surviving the Data Deluge
Open Information Extraction from the Web
CTOs on Virtualization
Living Machines
High-Performance Web Sites

Big Data


The Economist
FEBRUARY 27TH–MARCH 5TH 2010
Economist.com
Obama the warrior
Misgoverning Argentina
The economic shift from West to East
Genetically modified crops blossom
The right to eat cats and dogs
The data deluge
AND HOW TO HANDLE IT: A 14-PAGE SPECIAL REPORT

Big Data



Astronomical Data Deluge

In excess of 1 Exabyte of raw data in a single day - more than the entire daily internet traffic

Square Kilometre Array

+ A €1.5 billion global science project
+ Astronomers and engineers from more than 70 institutes in 20 countries
+ 3000 dishes, each 15m wide
+ Using enough optical fibre to wrap twice around the Earth
+ A combined collecting area of about one square kilometre

Enough raw data to fill over 15 million 64GB iPods every day

DOME Focus Areas

+ Advanced accelerators and 3D stacked chips for more energy-efficient computing
+ Novel optical interconnect technologies and nanophotonics to optimize large data transfers
+ High-performance storage systems based on next-generation tape systems and novel phase-change memory

ASTRON & IBM Center for Exascale Technology
Drenthe, Netherlands

ASTRON Netherlands Institute for Radio Astronomy

IBM

Data

Things to think about

- Big is Relative
    - Too Big to Fit in Memory (16-256 GB today)
    - Too Big to Fit on Disk (1-100 TB today)
- Plan for Data Analysis
    - Don't just save everything.
    - On the fly analysis, post-processing automation.
    - Is it worth storing or just recomputing?

Disk I/O

Common Uses

- Checkpoint/Restart Files
- Data Analysis
- Data Organization
- Time accurate and/or Optimization Runs
- Batch and Data processing
- Database

Disk I/O

Common Bottlenecks

- Mechanical disks are slow!
- System call overhead (open, close, read, write)
- Shared file system (nfs, lustre, gpfs, etc)
- HPC systems typically designed for high bandwidth (GB/s) not IOPs
- Uncoordinated independent accesses

Disk Access Rates over Time

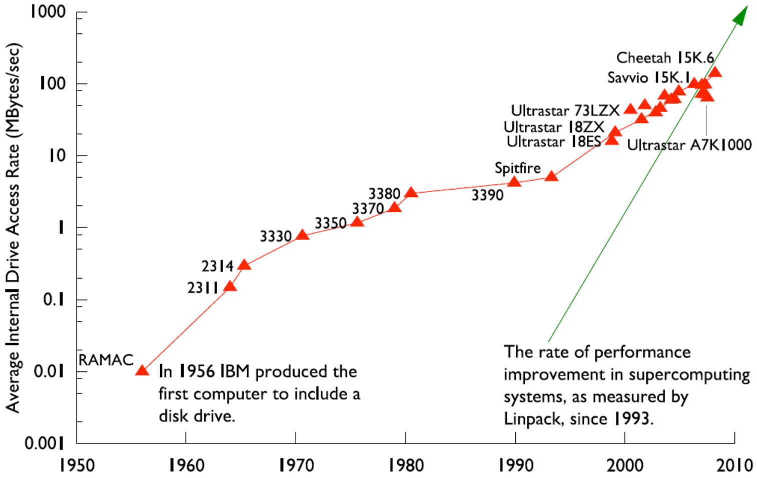


Figure by R. Ross, Argonne National Laboratory, CScADS09

Memory/Storage Latency

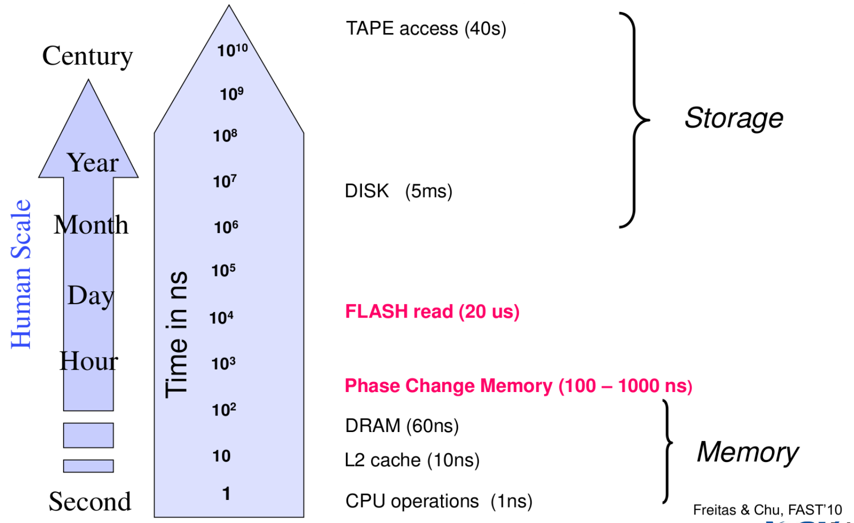


Figure by R. Freitas and L Chiu, IBM Almaden Labs, FAST'10

Definitions

IOPs

Input/Output Operations Per Second (read,write,open,close,seek)

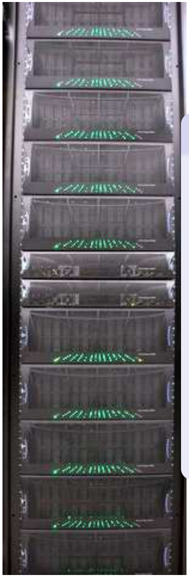
I/O Bandwidth

Quantity you read/write (think network bandwidth)

Comparisons

Device	Bandwidth (MB/s)	per-node	IOPs	per-node
SATA HDD	100	100	100	100
SSD HDD	250	250	4000	4000
SciNet	5000	1.25	30000	7.5

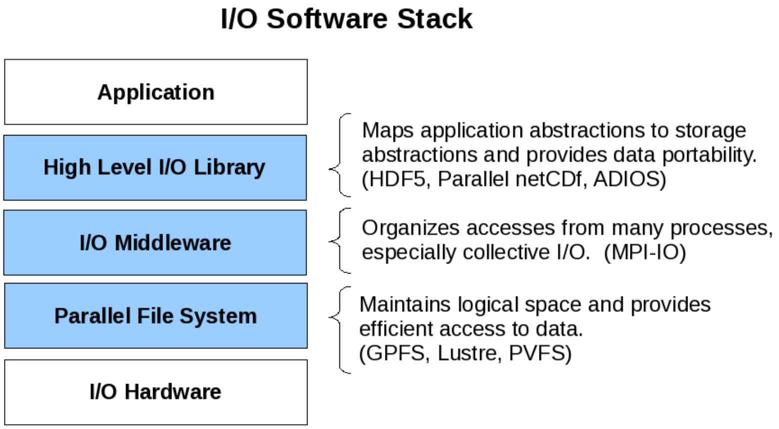
SciNet Filesystem



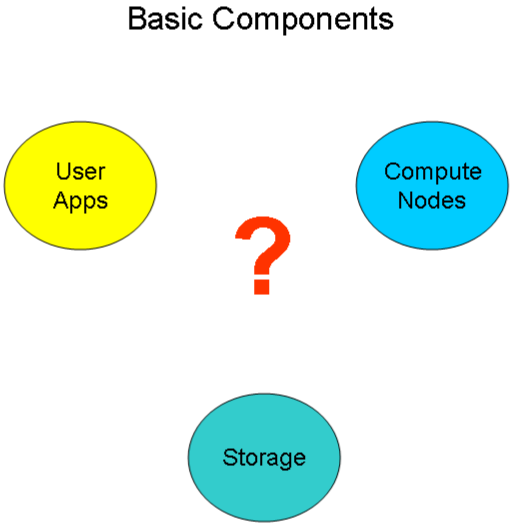
File System

- 1,790 1TB SATA disk drives, for a total of 1.4PB
- Two DCS9900 couplets, each delivering:
  - 4-5 GB/s read/write access (bandwidth)
  - 30,000 IOPs max (open, close, seek, ...)
- Single *GPFS* file system on TCS and GPC
- I/O goes over infiniband (as of April 2012)
- File system is parallel!

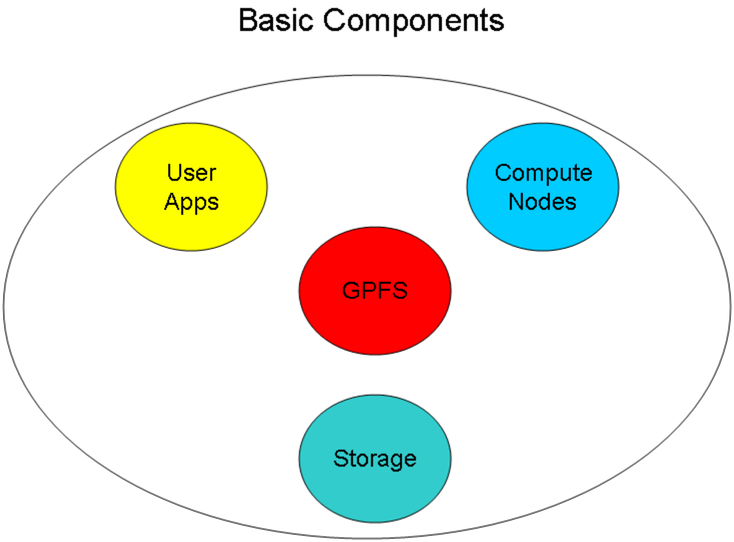
I/O Software Stack


I/O Software Stack
Application
High Level I/O Library
I/O Middleware
Parallel File System
I/O Hardware
Maps application abstractions to storage abstractions and provides data portability. (HDF5, Parallel netCDf, ADIOS)
Organizes accesses from many processes, especially collective I/O. (MPI-IO)
Maintains logical space and provides efficient access to data. (GPFS, Lustre, PVFS)

Parallel File System

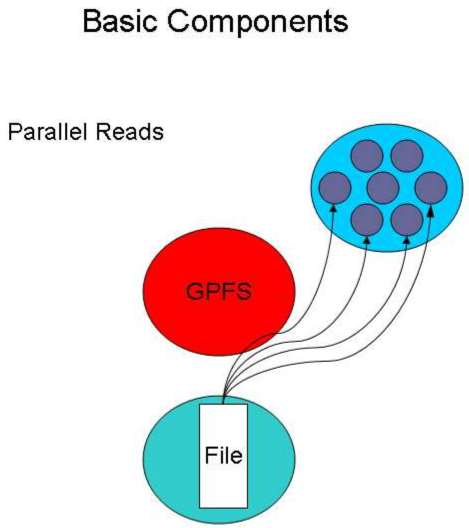


Parallel File System



General Parallel File System

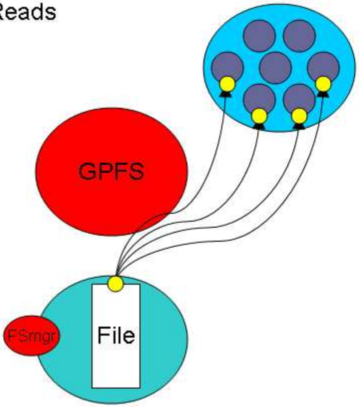
Parallel File System



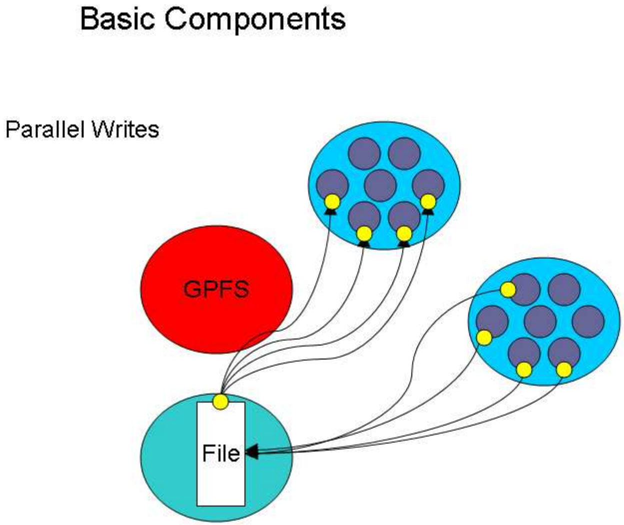
Parallel File System

Basic Components

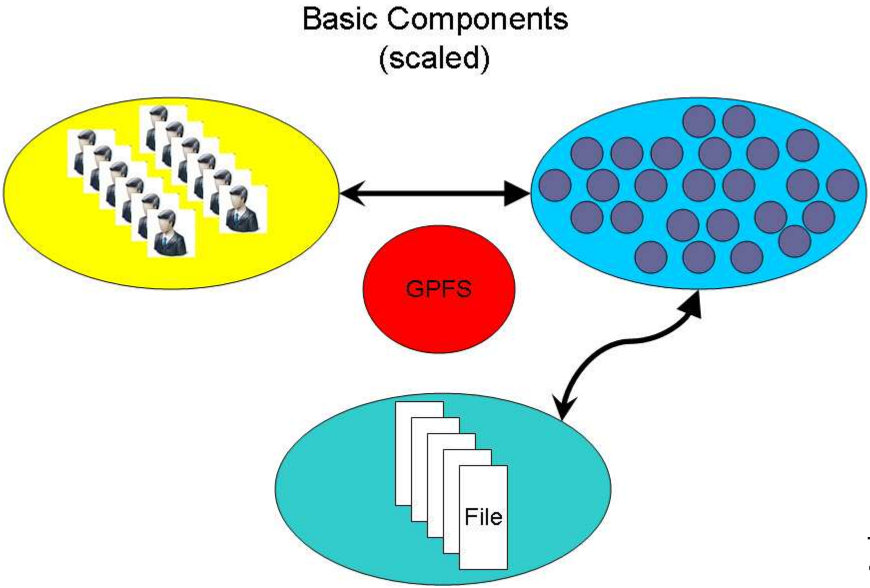
Parallel Reads



Parallel File System

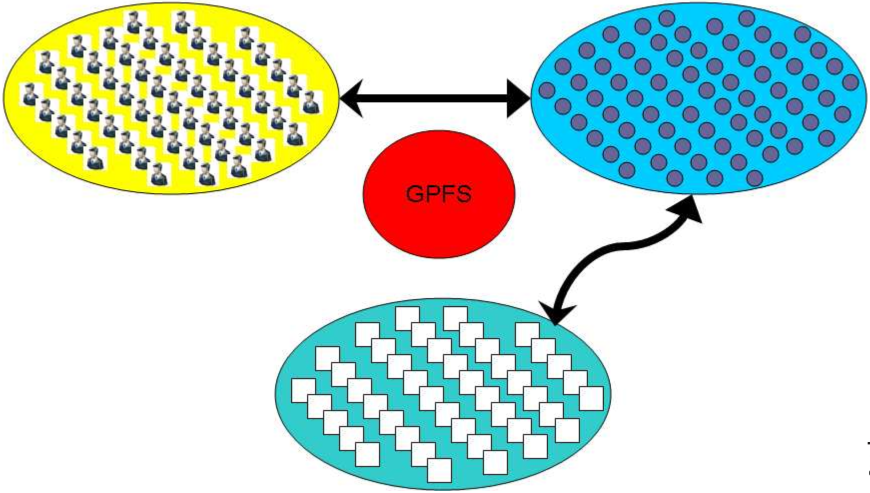


Parallel File System



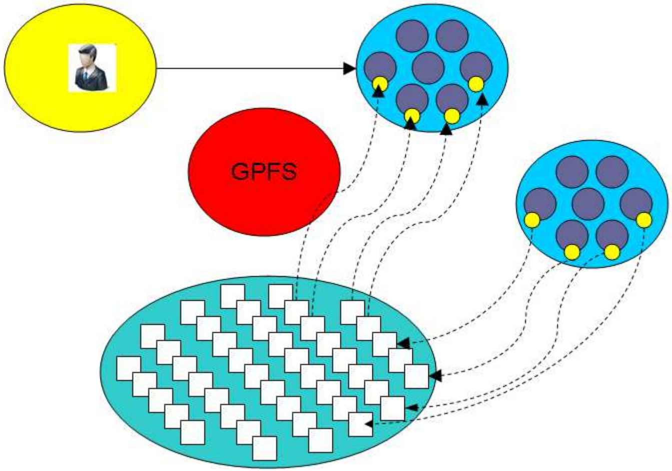
Parallel File System

How can we push the limit?



Parallel File System

How can we BREAK the limit?



Parallel File System

File Locks

Most parallel file systems use locks to manage concurrent file access

- Files are broken up into lock units
- Clients obtain locks on units that they will access before I/O occurs
- Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
- Locks are reclaimed from clients when others desire access

Parallel File System

- Optimal for large shared files.
- Behaves poorly under many small reads and writes, high IOPs
- Your use of it affects everybody!
  (Different from case with CPU and RAM which are not shared.)
- How you read and write, your file format, the number of files in a directory, and how often you `ls`, affects every user!
- The file system is shared over the network on GPC:
  Hammering the file system can hurt process communications.
- File systems are not infinite!
  Bandwidth, metadata, IOPs, number of files, space, . . .

Parallel File System

- 2 jobs doing simultaneous I/O can take much longer than twice a single job duration due to disk contention and directory locking.
- SciNet: 500+ users doing I/O from 4000 nodes.
  That's a lot of sharing and contention!

Data Management

Formats

- ASCII
- Binary
- MetaData (XML)
- Databases
- Standard Library's (HDF5,NetCDF)

ASCII

American Standard Code for Information Interchange

Pros

- Human Readable
- Portable (architecture independent)

Cons

- Inefficient Storage
- Expensive for Read/Write (conversions)

Native Binary

100100100

Pros

- Efficient Storage (256 x floats @4bytes takes 1024 bytes)
- Efficient Read/Write (native)

Cons

- Have to know the format to read
- Portability (Endianness)

ASCII vs. binary

Writing 128M doubles

Format	/scratch (GPCS)	/dev/shm (RAM)	/tmp (disk)
ASCII	173s	174s	260s
Binary	6s	1s	20s

Syntax

Format	C	FORTRAN

ASCII	`fprintf()`	open(6,file='test',form='formatted')
		write(6,*)

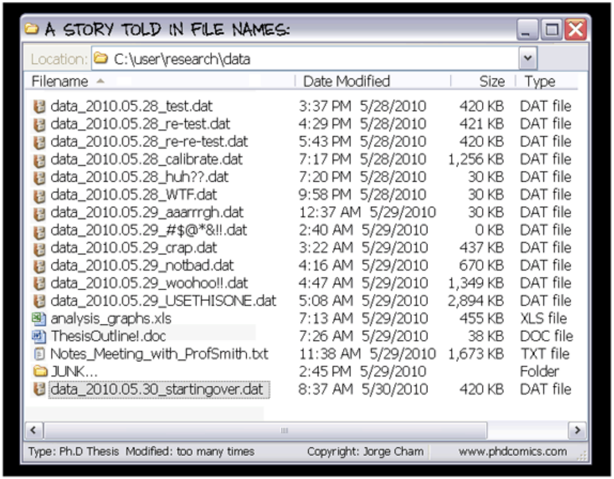
Binary	`fwrite()`	open(6,file='test',form='unformatted')
		write(6)

Data Management

File(s)

- Human-interpretable filenames lose their charm after few dozen files (or even after a few months pass)...
- Need to avoid thousands of files in a flat directory.
- A few big files are more efficient that many little ones.
- Keep parallel I/O in mind.
- Rigorously maintained metadata becomes essential.
- Possibly use a database or version control (i.e. git-annex).

Data Management


A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

http://www.phdcomics.com/comics/archive.php?comicid=1323

Metadata

What is Metadata?

Data about Data

- File System: size, location, date, owner, etc.
- App Data: File format, version, iteration, etc.

Example: XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<slice_data>
  <format>UTF1000</format>
  <verstion>6.8</version>
  <img src="slice1_2010.img" alt='Slice 1 of Data'/>
  <date> January 15th, 2010 </date>
  <loc> 47 23.516 -122 02.625 </loc>
</slice_data>
```

Databases

Beyond flat files

- Very powerful and flexible storage approach
- Data organization and analysis can be greatly simplified
- Enhanced performance over seek/sort depending on usage
- Open Source Software
  - SQLite (serverless)
  - PostgreSQL
  - mySQL

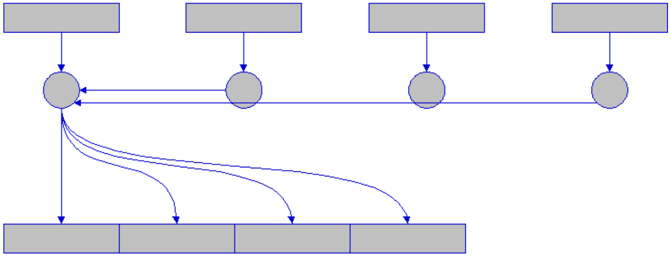
“Standard” Formats

- CGNS (CFD General Notation System)
- IGES/STEP (CAD Geometry)
- HDF5 (Hierarchical Data Format)
- NetCDF (Network Common Data Format)
- disciplineX version

Parallel I/O

Sequential I/O (only proc 0 Writes/Reads)

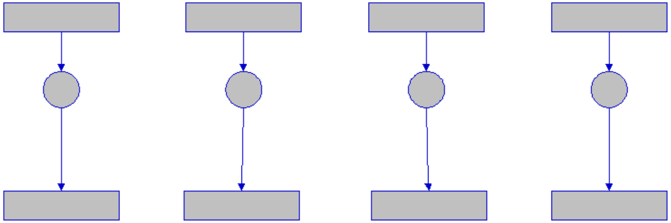
- Pro
  - Trivially simple for small I/O
  - Some I/O libraries not parallel
- Con
  - Bandwidth limited by rate one client can sustain
  - May not have enough memory on node to hold all data
  - Won't scale (built in bottleneck)



Common Ways of Doing Parallel I/O

N files for N Processes

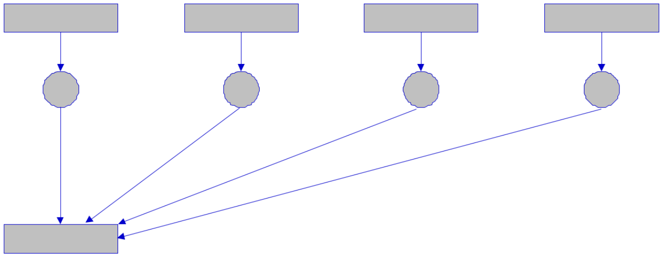
- Pro
  - No interprocess communication or coordination necessary
  - Possibly better scaling than single sequential I/O
- Con
  - As process counts increase, lots of (small) files, won't scale
  - Data often must be post-processed into one file
  - Uncoordinated I/O may swamp file system (File LOCKS!)



Common Ways of Doing Parallel I/O

All Processes Access One File

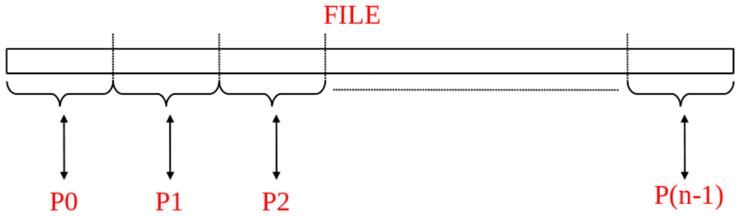
- Pro
  - Only one file
  - Data can be stored canonically, avoiding post-processing
  - Will scale if done correctly
- Con
  - Uncoordinated I/O WILL swamp file system (File LOCKS!)
  - Requires more design and thought



Parallel I/O

What is Parallel I/O?

Multiple processes of a parallel program accessing data (reading or writing) from a common file.



Parallel I/O

Why Parallel I/O?

- Non-parallel I/O is simple but:
  - Poor performance (single process writes to one file)
  - Awkward and not interoperable with other tools (each process writes a separate file)
- Parallel I/O
  - Higher performance through collective and contiguous I/O
  - Single file (visualization, data management, storage, etc)
  - Works with file system not against it

Parallel I/O

Available Approaches

- MPI-IO: MPI-2 Language Standard
- HDF (Hierarchical Data Format)
- NetCDF (Network Common Data Format)
- Adaptable IO System (ADIOS)
  - Actively developed (OLCF,SandiaNL,GeorgiaTech) and used on largest HPC systems (Jaguar,Blue Gene/P)
  - External to the code XML file describing the various elements
  - Uses MPI-IO, can work with HDF/NetCDF

I/O Best Practices

Make a plan

- Make a plan for your data needs:
  - How much will you generate,
  - How much do you need to save,
  - And where will you keep it?
- Note that /scratch is temporary storage for 3 months or less.

Options?

1. Save on your departmental/local server/workstation (it is possible to transfer TBs per day on a gigabit link);
2. Apply for a project space/HPSS allocation at next RAC call (but space is very limited);
3. Change storage format.

SciNet compute • calcul CANADA

I/O Best Practices

Monitor and control usage

- Minimize use of filesystem commands like `ls` and `du`.
- Regularly check your disk usage using /scinet/gpc/bin/diskUsage.
- Warning signs which should prompt careful consideration:
  - More than 100,000 files in your space
  - Average file size less than 100 MB
- Monitor disk actions with `top` and `strace`

- RAM is always faster than disk; think about using ramdisk.
- Use `gzip` and `tar` to compress files to bundle many files into one
- Try gziping your *data* files. 30% not atypical!
- Delete files that are no longer needed
- Do "housekeeping" (`gzip`, `tar`, delete) regularly.

I/O Best Practices

Do's

- Write binary format files
  Faster I/O and less space than ASCII files.
- Use parallel I/O if writing from many nodes
- Maximize size of files. Large block I/O optimal!
- Minimize number of files. Makes filesystem more responsive!

Don'ts

- Don't write lots of ASCII files. Lazy, slow, and wastes space!
- Don't write many hundreds of files in a 1 directory. (File Locks)
- Don't write many small files ($<$ 10MB).
  System is optimized for large-block I/O.

Outline

Software Development

Tools of the Trade

- Editors/IDE

Software Development

Tools of the Trade

- Editors/IDE
- Version Control

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers
- Libraries

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers
- Libraries
- Debuggers (gdb,idb, Allinea DDT)

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers
- Libraries
- Debuggers (gdb,idb, Allinea DDT)
- Performance (gprof,Scalasa,IPM)
- Memory (valgrind)

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers
- Libraries
- Debuggers (gdb,idb, Allinea DDT)
- Performance (gprof,Scalasa,IPM)
- Memory (valgrind)
- I/O (strace)

Software Development

Tools of the Trade

- Editors/IDE
- Version Control
- Build System (make)
- Compilers
- Libraries
- Debuggers (gdb,idb, Allinea DDT)
- Performance (gprof,Scalasa,IPM)
- Memory (valgrind)
- I/O (strace)

Outline

Version Control

What is it?

- A tool for managing changes in a set of files.

Version Control

What is it?

- A tool for managing changes in a set of files.
- Figuring out who broke what where and when.

Version Control

What is it?

- A tool for managing changes in a set of files.
- Figuring out who broke what where and when.

Why Do it?

- Collaboration
- Organization
- Track Changes
- Faster Development
- Reduce Errors

Collaboration
With others and yourself

Questions

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
    - But then only one person can be working at any time
    - And how do you enforce the rule?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost
- Option 3: **Version Control**

Organize and Track Changes

Question

Organize and Track Changes

Question

- Want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
    ...but keep the whole history of every file, forever

Organize and Track Changes

Question

- Want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
    ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

Organize and Track Changes

Question

- Want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
    ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

Answer

- **Version Control**

What Software to Use

Software

- Open Source
  - Subversion, CVS, RCS
  - Git, Mercurial, Bazaar
- Commercial
  - Perforce, ClearCase

available as modules on SciNet

Version Control Software

Subversion (svn)

- Centralized Version Control
- Replaces CVS
- Lots of web and GUI integration
- Users: GCC, KDE, FreeBSD

Git

- Distributed Version Control
- *nix command line driven design model
- advanced features `git-stash`, `git-rebase`, `git-cherry-pick`
- Users: Linux kernel, GNOME, Wine, X.org

Outline

GPC x86_64 Compilers

GNU Compiler Collection (v4.9.0)

- C (gcc)
- C++ (g++)
- FORTRAN (gfortran)

Intel Composer XE 2013 (v14.0) *recommended

- C (icc)
- C++ (icpc)
- FORTRAN (ifort)
- Threaded Building Blocks (TBB)
- Integrated Performance Primitives (IPP)
- Math Kernel Libraries (MKL)

Optimizations

Optimization Levels

- **-O0** disable optimization
- **-O1** optimizes for code size
- **-O2** optimizes for speed (default)
- **-O3** **-O2** plus more aggressive optimizations

Optimizations

Optimization Levels

- **-O0** disable optimization
- **-O1** optimizes for code size
- **-O2** optimizes for speed (default)
- **-O3** **-O2** plus more aggressive optimizations

From the Intel Manual

"The **-O3** option is particularly recommended for applications that have loops that do many floating-point calculations or process large data sets."

Optimizations

-O2 Optimizations

- intrinsic inlining
- inlining
- constant propagation
- forward substitution
- routine attribute propagation
- variable address-taken analysis
- dead static function elimination
- removal of unreferenced variables
- constant propagation
- copy propagation
- dead-code elimination
- global register allocation
- global instruction scheduling and control speculation
- loop unrolling
- optimized code selection
- partial redundancy elimination
- strength reduction/induction variable simplification
- variable renaming
- exception handling optimizations
- tail recursions
- peephole optimizations
- structure assignment lowering and optimizations
- dead store elimination

Optimization Terminology
Inlining

Inlining

Replaces the function call with the actual functions code.

Optimization Terminology
Inlining

Inlining
Replaces the function call with the actual functions code.

Original
```
int func(int &x,int &y) { return 4*x+3*y; }

int main(){
  int x=4, y=3;
  int b=fun(x,y)
}
```

Optimization Terminology

Inlining

Inlining

Replaces the function call with the actual functions code.

Original

```
int func(int &x,int &y) { return 4*x+3*y; }

int main(){
  int x=4, y=3;
  int b=fun(x,y)
}
```

Inlined

```
int main(){
  int x=4,y=3;
  int b= 4*x+3*y;
}
```

Optimization Terminology

Branch Elimination

Original

```
if ( x < x1 ) {
  a = a0 + a1;
} else if ( x < x2 ) {
  a = a0 - a1;
} else if ( x < x3 ) {
  a = a0 * a1;
} else if ( x < x4 ) {
  a = a0 / a1;
} else   {
  a = a0;
}
```

Optimizer Approaches

- static branch elimination
- compute all cases and conditions, then pick the correct one
- replace with switch statements, jump tables
- branch re-alignment

Optimizations

-O3 Additional Optimizations

- Loop Blocking for cache
- Loop Permutation or Interchange
- Loop Distribution
- Loop Fusion
- Loop Unrolling
- Unroll and Jam
- Loop Blocking or Tiling
- Loop Reversal
- Loop Peeling
- Loop Rerolling
- Profile-Guided Loop Unrolling
- Code Replication to eliminate branches
- Memory-access optimizations
- Data Prefetching
- Scalar Replacement
- Partial-Sum Optimization
- Predicate Optimization
- Data Transformation: Malloc Combining and Memset Combining
- Memset and Memcpy Recognition
- Statement Sinking for Creating Perfect Loopnests

Optimization Terminology

Loop Unrolling

Original

```
for (int x=0; x < 100; x++)
{
  func(x);
}
```

Optimization Terminology

Loop Unrolling

Original

```
for (int x=0; x < 100; x++)
{
  func(x);
}
```

Optimized

```
for (int x = 0; x < 100; x+=5)
{
  func(x);
  func(x+1);
  func(x+2);
  func(x+3);
  func(x+4);
}
```

Optimization Terminology
Loop Collapsing

Original

```
int a[100][300];
for (int i = 0; i < 300; i++)
  for (int j = 0; j < 100; j++)
     a[j][i] = 0;
```

Optimization Terminology

Loop Collapsing

Original

```
int a[100][300];
for (int i = 0; i < 300; i++)
  for (int j = 0; j < 100; j++)
     a[j][i] = 0;
```

Optimized

```
int a[100][300];
int *p = &a[0][0];

for (int i = 0; i < 30000; i++)
  *p++ = 0;
```

Optimization Terminology
Loop Fusion

Original

```
int x[100], y[100];
for (int i = 0; i < 100; i++)
   x[i] = 1;
for (int i = 0; i < 100; i++)
   y[i] = 2;
```

Optimization Terminology

Loop Fusion

Original

```
int x[100], y[100];
for (int i = 0; i < 100; i++)
   x[i] = 1;
for (int i = 0; i < 100; i++)
   y[i] = 2;
```

Optimized

```
int  x[100], y[100];
for (int i = 0; i < 100; i++)
{
  x[i] = 1;
  y[i] = 2;
}
```

Optimization Terminology

Loop Peeling

Original

```
int p = 10;
for (int i=0; i<10; ++i)
{
  y[i] = x[i] + x[p];
  p = i;
}
```

Optimization Terminology

Loop Peeling

Original

```
int p = 10;
for (int i=0; i<10; ++i)
{
  y[i] = x[i] + x[p];
  p = i;
}
```

Optimized

```
y[0] = x[0] + x[10];
for (int i=1; i<10; ++i)
{
  y[i] = x[i] + x[i-1];
}
```

Optimizations

System Specific

- **-march="cpu"** optimize for a specific cpu
- **-mtune="cpu"** produce code only for a specific cpu
- **-msse3,-msse4,-mavx, etc.** level of SIMD and vector instructions

Optimizations

System Specific

- **-march="cpu"** optimize for a specific cpu
- **-mtune="cpu"** produce code only for a specific cpu
- **-msse3,-msse4,-mavx, etc.** level of SIMD and vector instructions

Use this instead!

-xHost optimize and tune for the compiling CPU

Optimizations

System Specific

- **-march="cpu"** optimize for a specific cpu
- **-mtune="cpu"** produce code only for a specific cpu
- **-msse3,-msse4,-mavx, etc.** level of SIMD and vector instructions

Use this instead!

-xHost optimize and tune for the compiling CPU

GPC Recommendations

-xHost -O3

Optimization Terminology

Vector Extensions

Intel x86_64 extensions

- Streaming SIMD Extensions (SEE1 - SSE4.2)
- AVX, AVX2, AVX512

Original x86

Add two single precision vectors requires four floating-point addition instructions.

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

Optimization Terminology

Vector Extensions

Intel x86_64 extensions

- Streaming SIMD Extensions (SEE1 - SSE4.2)
- AVX, AVX2, AVX512

Original x86

Add two single precision vectors requires four floating-point addition instructions.

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

SSE

A single 128-bit 'packed-add' replaces four scalar addition instructions.

```
movaps xmm0, [v1]; xmm0 = v1.w | v1.z | v1.y | v1.x
addps xmm0, [v2];  xmm0 = v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x
movaps [vec_res], xmm0
```

Floating Point Math

-fpmodel

- **fast=1** default
- **fast=2** most aggressive
- **precise** value-safe optimizations on intermediate operations
- **except** strict floating point semantics
- **strict** disables all “fast-math” options

If Required

For floating point consistency and reproducibility use:
-fpmodel precise -fpmodel except

Memory Model

Seen this error?

relocation truncated to fit: R_X86_64_PC32

Memory Model

Seen this error?

relocation truncated to fit: R_X86_64_PC32

-mcmodel=

- **small** code and data restricted to the first 2GB of address space
- **medium** code restricted to the first 2GB of address space
- **large** no restrictions

Intel Math Kernel Library

MKL Components

- BLAS
- LAPACK
- ScaLAPACK
- FFT
- PBLAS
- BLACS
- plus others

Intel Math Kernel Library

Dynamic Link Line for MKL >10.3

- **-L mkl_rt**

Link Line - Composer XE 2013

- **-mkl=sequential** no-threaded versions (serial)
- **-mkl=parallel** threaded (openmp)
- **-mkl=cluster** for ScaLAPACK, FFT, BLACS

Link Line Advisor

http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/

Documentation

Intel Documentation

http://software.intel.com/en-us/articles/intel-parallel-studio-xe-for-linux-documentation/

Compiler Optimization flags

http://software.intel.com/sites/products/collateral/hpc/compilers/compiler_qrg12.pdf

White Paper on Floating Point

https://support.scinet.utoronto.ca/wiki/images/f/f2/FP_Consistency.pdf

Outline

Numerical Computing

Numerical Methods

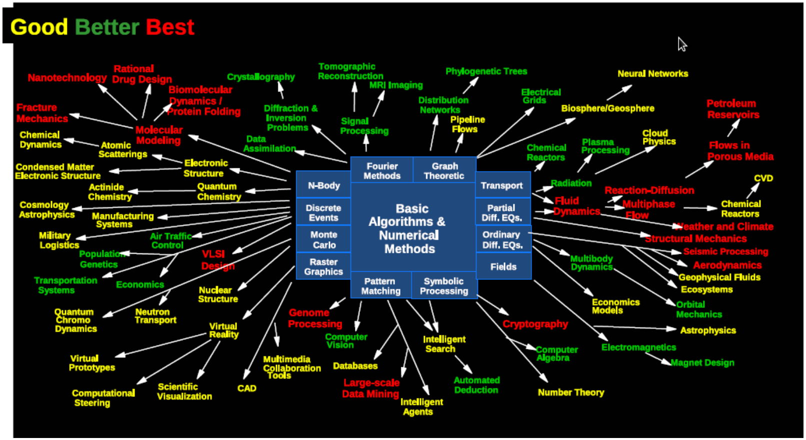
- Linear algebra
- Nonlinear equations
- Optimization
- Interpolation/Approximation
- Integration and differentiation
- Solving ODEs
- Solving PDEs
- FFT
- Random numbers and stochastic simulations
- Special functions

Numerical Algorithms

Top Ten Algorithms for Science (Jack Dongarra, 2000)

1. Metropolis Algorithm for Monte Carlo
2. Simplex Method for Linear Programming
3. Krylov Subspace Iteration Methods
4. The Decompositional Approach to Matrix Computations
5. The Fortran Optimizing Compiler
6. QR Algorithm for Computing Eigenvalues
7. Quicksort Algorithm for Sorting
8. Fast Fourier Transform
9. Integer Relation Detection
10. Fast Multipole Method

Numerical Algorithms



Argonne National Laboratory GBB

Libraries

Numerical Libraries

- BLAS (gotoblas, ATLAS)
- LAPACK (ESSL, MKL, ACML)
- ScaLAPACK
- GSL (GNU Scientific Library)
- FFTW
- PETSc
- TAO
- IMSL
- NAG

Libraries

Numerical Libraries

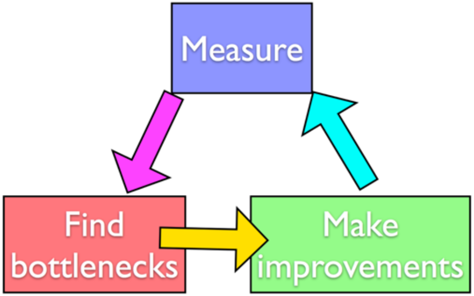
- BLAS (gotoblas, ATLAS)
- LAPACK (ESSL, MKL, ACML)
- ScaLAPACK
- GSL (GNU Scientific Library)
- FFTW
- PETSc
- TAO
- IMSL
- NAG

Don't re-invent the wheel!

Outline

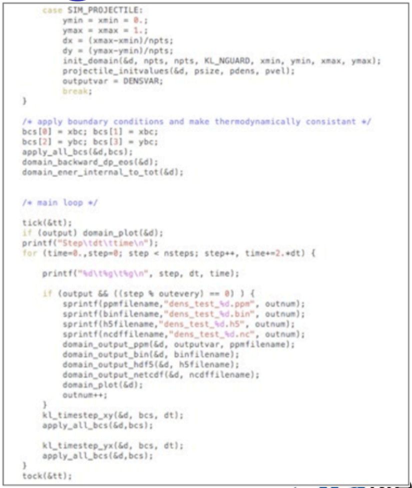
Profiling

- Like debuggers for debugging, profilers are evidence-based methods to find performance problems.
- Can't improve what you don't measure.



Profiling

- Where in your program is time being spent?
- Find the expensive parts
  - Don't waste time optimizing parts that don't matter
- Find bottlenecks.

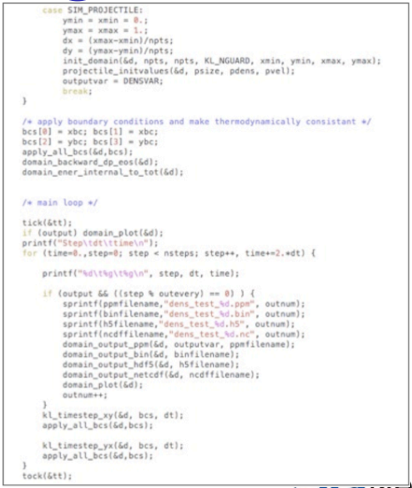


```
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```

Profiling

- Tracing vs. Sampling
- Instrumenting vs. instrumentation-free

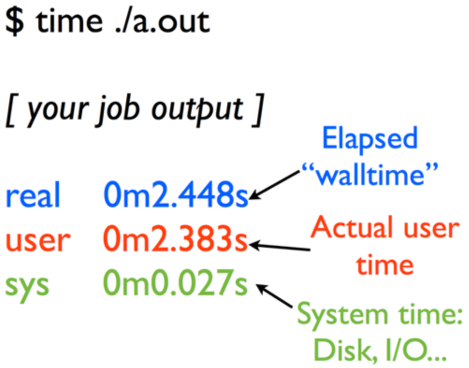
.



```
    case SIM_PROJECTILE:
        ymin = xmin = 0.;
        ymax = xmax = 1.;
        dx = (xmax-xmin)/npts;
        dy = (ymax-ymin)/npts;
        init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
        projectile_initvalues(&d, psize, pdens, pvel);
        outputvar = DENSVAR;
        break;
}

/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

    printf("%d\t%g\t%g\n", step, dt, time);

    if (output && ((step % outevery) == 0) ) {
        sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
        sprintf(binfilename,"dens_test_%d.bin", outnum);
        sprintf(h5filename,"dens_test_%d.h5", outnum);
        sprintf(ncdffilename,"dens_test_%d.nc", outnum);
        domain_output_ppm(&d, outputvar, ppmfilename);
        domain_output_bin(&d, binfilename);
        domain_output_hdf5(&d, h5filename);
        domain_output_netcdf(&d, ncdffilename);
        domain_plot(&d);
        outnum++;
    }
    kl_timestep_xy(&d, bcs, dt);
    apply_all_bcs(&d,bcs);

    kl_timestep_yx(&d, bcs, dt);
    apply_all_bcs(&d,bcs);
}
tock(&tt);
```

Timing whole program

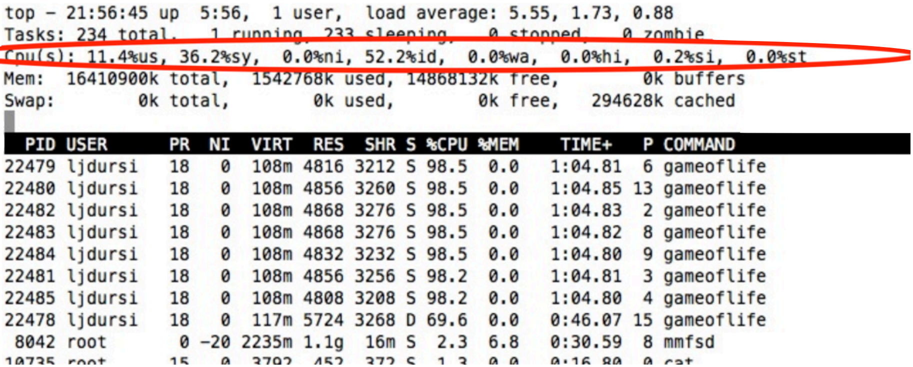
- Very simple; can run on any command.
- In serial, real = user + sys
- In parallel, ideally user = nprocs x real
- Can run on tests to identify *performance regressions*.



```
$ time ./a.out

[ your job output ]

real    0m2.448s
user    0m2.383s
sys     0m0.027s
```

Watching program run

```
$ top
```

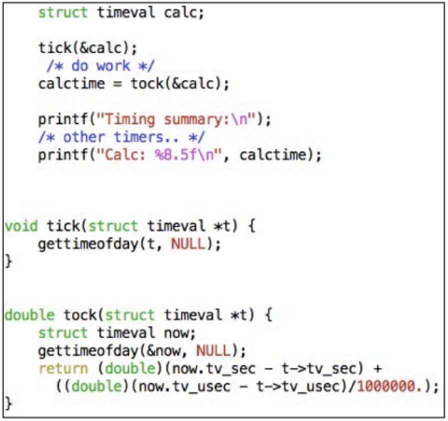


```
top - 21:56:45 up  5:56,  1 user,  load average: 5.55, 1.73, 0.88
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.4%us, 36.2%sy,  0.0%ni, 52.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  16410900k total,  1542768k used, 14868132k free,        0k buffers
Swap:        0k total,        0k used,        0k free,   294628k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
22479 ljdursi   18   0  108m 4816 3212 S 98.5  0.0   1:04.81  6 gameoflife
22480 ljdursi   18   0  108m 4856 3260 S 98.5  0.0   1:04.85 13 gameoflife
22482 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.83  2 gameoflife
22483 ljdursi   18   0  108m 4868 3276 S 98.5  0.0   1:04.82  8 gameoflife
22484 ljdursi   18   0  108m 4832 3232 S 98.5  0.0   1:04.80  9 gameoflife
22481 ljdursi   18   0  108m 4856 3256 S 98.2  0.0   1:04.81  3 gameoflife
22485 ljdursi   18   0  108m 4808 3208 S 98.2  0.0   1:04.80  4 gameoflife
22478 ljdursi   18   0  117m 5724 3268 D 69.6  0.0   0:46.07 15 gameoflife
 8042 root       0 -20 2235m 1.1g  16m S  2.3  6.8   0:30.59  8 mmfsd
10735 root      15   0  3792  452  372 S  1.3  0.0   0:16.80  0 cat
```

More system then user time - not very efficient

Instrumenting regions of code

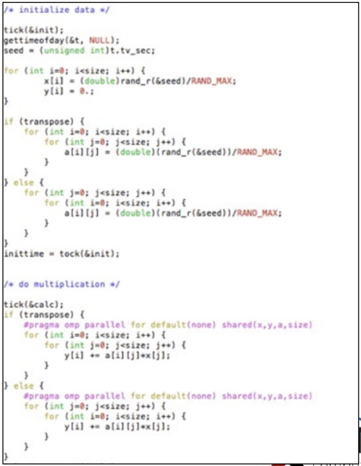
- *Instrumenting* the code
- Simple, but incrediby useful.
- Runs every time your code is run
- Can trivially see if changes make things better or worse



```
    struct timeval calc;

    tick(&calc);
     /* do work */
    calctime = tock(&calc);

    printf("Timing summary:\n");
    /* other timers.. */
    printf("Calc: %8.5f\n", calctime);


void tick(struct timeval *t) {
    gettimeofday(t, NULL);
}


double tock(struct timeval *t) {
    struct timeval now;
    gettimeofday(&now, NULL);
    return (double)(now.tv_sec - t->tv_sec) +
      ((double)(now.tv_usec - t->tv_usec)/1000000.);
}
```

Instrumenting regions of code

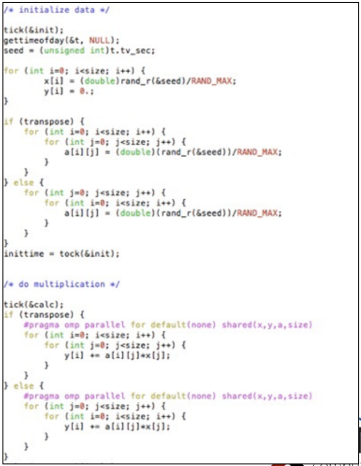
- Simple example - matrix-vector multiply
- Initializes data, does multiply, saves result
- Look to see where it spends its time, speed it up.
- Options for how to access data, output data.



```
/* initialize data */

tick(&init);
gettimeofday(&t, NULL);
seed = (unsigned int)t.tv_sec;

for (int i=0; i<size; i++) {
        x[i] = (double)rand_r(&seed)/RAND_MAX;
        y[i] = 0.;
}

if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
}
inittime = tock(&init);


/* do multiplication */

tick(&calc);
if (transpose) {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
```

Matrix-vector multiply

- Simple example - matrix-vector multiply
- Initializes data, does multiply, saves result
- Look to see where it spends its time, speed it up.
- Options for how to access data, output data.



```
/* initialize data */

tick(&init);
gettimeofday(&t, NULL);
seed = (unsigned int)t.tv_sec;

for (int i=0; i<size; i++) {
        x[i] = (double)rand_r(&seed)/RAND_MAX;
        y[i] = 0.;
}

if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
}
inittime = tock(&init);


/* do multiplication */

tick(&calc);
if (transpose) {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
```

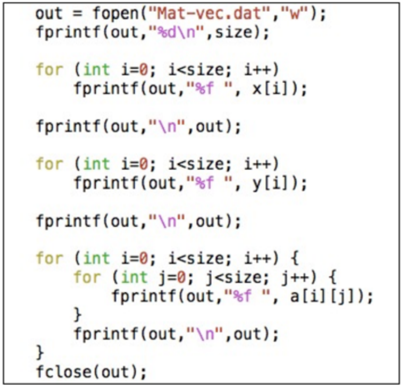
Matrix-vector multiply

- Can get an overview of the time spent easily, because we instrumented our code (~12 lines!)
- I/O huge bottleneck.

```
$ mvm --matsize=2500
Timing summary:
   Init:   0.00952 sec
   Calc:   0.06638 sec
   I/O :   5.07121 sec
```

Matrix-vector multiply

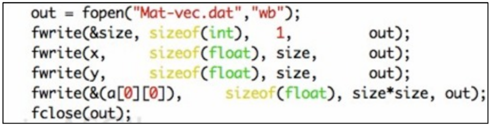
- I/O being done in ASCII
- having to loop over data, convert to string, write to output.
- 6,252,500 write operations!
- Let’s try a --binary option:



```
out = fopen("Mat-vec.dat","w");
fprintf(out,"%d\n",size);

for (int i=0; i<size; i++)
    fprintf(out,"%f ", x[i]);

fprintf(out,"\n",out);

for (int i=0; i<size; i++)
    fprintf(out,"%f ", y[i]);

fprintf(out,"\n",out);

for (int i=0; i<size; i++) {
    for (int j=0; j<size; j++) {
        fprintf(out,"%f ", a[i][j]);
    }
    fprintf(out,"\n",out);
}
fclose(out);
```

Matrix-vector multiply

- Let's try a --binary option:
- Shorter...



```
out = fopen("Mat-vec.dat","wb");
fwrite(&size, sizeof(int),   1,         out);
fwrite(x,     sizeof(float), size,      out);
fwrite(y,     sizeof(float), size,      out);
fwrite(&(a[0][0]),     sizeof(float), size*size, out);
fclose(out);
```

Matrix-vector multiply

- And much (36x!) faster
- File 4x smaller
- Still slow, but file I/O is always going to be slower than a multiplication.
- On to calculation...

```
$ mvm --matsize=2500
--binary
Timing summary:
   Init:   0.00976 sec
   Calc:   0.06695 sec
   I/O :   0.14218 sec
```

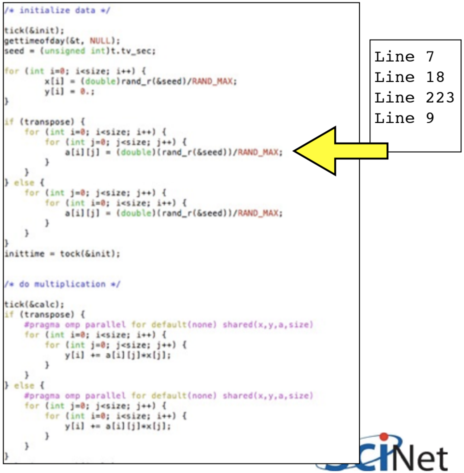
```
$ ./mvm --binary
$ du -h Mat-vec.dat
89M      Mat-vec.dat
$ ./mvm --binary
$ du -h Mat-vec.dat
20M      Mat-vec.dat
```

Sampling for Profiling

- How to get finer-grained information about where time is being spent?
- Can't instrument every single line.
- Compilers have tools for *sampling* execution paths.

Sampling for Profiling

- As program executes, every so often (~100ms) a timer goes off, and the current location of execution is recored
- Shows where time is being spent.

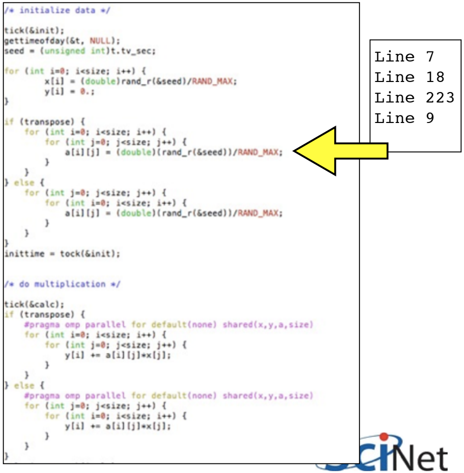


```
/* initialize data */

tick(&init);
gettimeofday(&t, NULL);
seed = (unsigned int)t.tv_sec;

for (int i=0; i<size; i++) {
        x[i] = (double)rand_r(&seed)/RAND_MAX;
        y[i] = 0.;
}

if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
}
inittime = tock(&init);


/* do multiplication */

tick(&calc);
if (transpose) {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
```

```
Line 7
Line 18
Line 223
Line 9
```

Sampling for Profiling

- Advantages:
  - Very low overhead
  - No extra instrumentation
- Disadvantages:
  - Don't know *why* code was there
  - Statistics - have to run long enough job



```
/* initialize data */

tick(&init);
gettimeofday(&t, NULL);
seed = (unsigned int)t.tv_sec;

for (int i=0; i<size; i++) {
        x[i] = (double)rand_r(&seed)/RAND_MAX;
        y[i] = 0.;
}

if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            a[i][j] = (double)(rand_r(&seed))/RAND_MAX;
        }
    }
}
inittime = tock(&init);


/* do multiplication */

tick(&calc);
if (transpose) {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    #pragma omp parallel for default(none) shared(x,y,a,size)
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
```

```
Line 7
Line 18
Line 223
Line 9
```

gprof for sampling

```
$ gcc -O3 -pg -g mat-vec-mult.c --std=c99
$ icc -O3 -pg -g mat-vec-mult.c -c99
```

turn on profiling

debugging symbols (optional, but more info)

```
$ ./mvm-profile --matsize=2500
[output]
$ ls
Makefile   Mat-vec.dat   gmon.out
mat-vec-mult.c   mvm-profile
```

gprof examines gmon.out

```
$ gprof mvm-profile gmon.out
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.24      0.41     0.41        3     0.00           main
  0.00      0.41     0.00        3     0.00     0.00  tick
  0.00      0.41     0.00        3     0.00     0.00  tock
  0.00      0.41     0.00        2     0.00     0.00  alloc1d
  0.00      0.41     0.00        2     0.00     0.00  free1d
  0.00      0.41     0.00        1     0.00     0.00  alloc2d
  0.00      0.41     0.00        1     0.00     0.00  free2d
  0.00      0.41     0.00        1     0.00     0.00  get_options
[...]
```

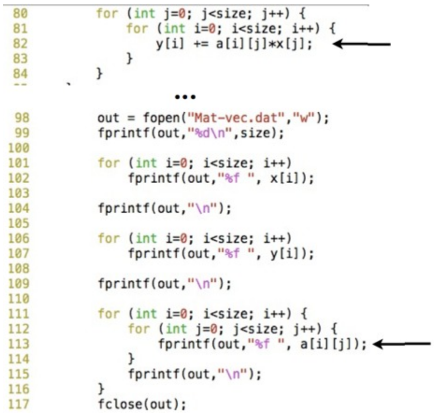
Gives data by function -- usually handy, not so useful in this toy problem

gprof --line

```
gpc-f103n084-$ gprof --line mvm-profile gmon.out | more
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 68.46      0.28     0.28                     main (mat-vec-mult.c:82 @ 401
 14.67      0.34     0.06                     main (mat-vec-mult.c:113 @ 40
  7.33      0.37     0.03                     main (mat-vec-mult.c:63 @ 401
  4.89      0.39     0.02                     main (mat-vec-mult.c:112 @ 40
  4.89      0.41     0.02                     main (mat-vec-mult.c:113 @ 40
  0.00      0.41     0.00    3     0.00   0.00  tick (mat-vec-mult.c:159 @ 40
  0.00      0.41     0.00    3     0.00   0.00  tock (mat-vec-mult.c:164 @ 40
  0.00      0.41     0.00    2     0.00   0.00  alloc1d (mat-vec-mult.c:152 @
  0.00      0.41     0.00    2     0.00   0.00  free1d (mat-vec-mult.c:171 @
  0.00      0.41     0.00    1     0.00   0.00  alloc2d (mat-vec-mult.c:130 @
  0.00      0.41     0.00    1     0.00   0.00  free2d (mat-vec-mult.c:144 @
  0.00      0.41     0.00    1     0.00   0.00  get_options (mat-vec-mult.c:1
```

Then can compare to source

- Code is spending most time deep in loops
- #1 - multiplication
- #2 - I/O (old way)



```
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }

                    ...

    out = fopen("Mat-vec.dat","w");
    fprintf(out,"%d\n",size);

    for (int i=0; i<size; i++)
        fprintf(out,"%f ", x[i]);

    fprintf(out,"\n");

    for (int i=0; i<size; i++)
        fprintf(out,"%f ", y[i]);

    fprintf(out,"\n");

    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            fprintf(out,"%f ", a[i][j]);
        }
        fprintf(out,"\n");
    }
    fclose(out);
```

gprof pros/cons

- Exists (almost) everywhere
- Easy to script, put in batch jobs
- Low overhead
- As with graphical debuggers, many nice graphical profilers exist as well

Memory Profiling

Most profilers use time as a the metric, but what about memory?

Valgrind

- Massif: Memory Heap Profiler
  - `valgrind --tool=massif ./mycode`
  - `ms_print massif.out`
- Cachegrind: Cache Profiler
  - `valgrind --tool=cachegrind ./mycode`
  - Kcachegrind (gui frontend for cachegrind)

http://valgrind.org/

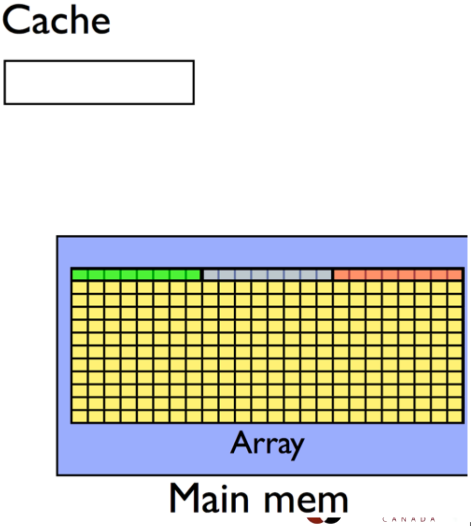
Memory Profiling: Valgrind Massif

Example of output from `ms_print`, showing heap memory usage.

```
--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
 11 17,558,376,865      108,721,536      108,079,702       641,834            0
 12 18,730,053,265      108,746,848      108,104,510       642,338            0
 13 19,748,755,982      108,742,200      108,099,974       642,226            0
 14 21,351,204,796      108,745,520      108,103,214       642,306            0
 15 22,575,905,502      108,742,200      108,099,974       642,226            0
 16 24,344,627,331      108,742,200      108,099,974       642,226            0
 17 25,780,057,465      108,742,200      108,099,974       642,226            0
 18 27,215,452,841      108,742,200      108,099,974       642,226            0
99.41% (108,099,974B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->55.61% (60,466,176B) 0x873A8A: BlockMat::setup() (in navierstokes3Dthermallyperfect.5)
| ->55.61% (60,466,176B) 0x47A0F5: Hexa_NKS_Solver<State>::allocate() (NKS.h:192)
|   ->55.61% (60,466,176B) 0x477796: int HexaSolver<State>(char*, int) (HexaSolver.h:150)
|     ->55.61% (60,466,176B) 0x476A9F: main (NavierStokes3DThermallyPerfect.cc:226)
|
->10.07% (10,948,608B) 0x47A3B2: Hexa_NKS_Solver<State>::allocate() (NKS.h:186)
| ->10.07% (10,948,608B) 0x477796: int HexaSolver<State>(char*, int) (HexaSolver.h:150)
|   ->10.07% (10,948,608B) 0x476A9F: main (NavierStokes3DThermallyPerfect.cc:226)
|
->09.15% (9,953,280B) 0x47A390: Hexa_NKS_Solver<Statee>::allocate() (NKS.h:186)
| ->09.15% (9,953,280B) 0x477796: int HexaSolver<State>(char*, int) (HexaSolver.h:150)
|   ->09.15% (9,953,280B) 0x476A9F: main (NavierStokes3DThermallyPerfect.cc:226)
```

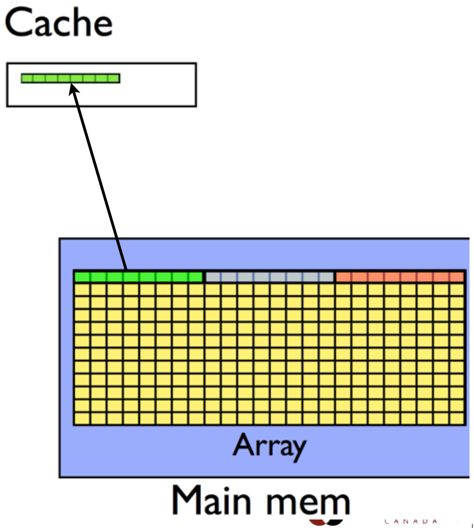
Cache Thrashing

- Memory bandwidth is key to getting good performance on modern systems
- Main Mem - big, slow
- Cache - small, fast
    - Saves recent accesses, a line of data at a time.



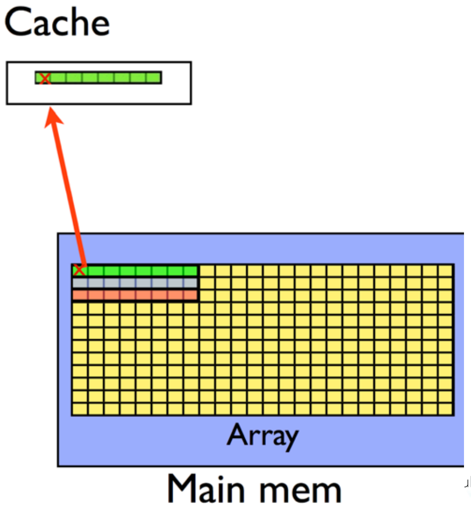
Cache Thrashing

- When accessing memory in order, only one access to slow main mem for many data points
- Much faster



Cache Thrashing

- When accessing memory out of order, much worse
- Each access is new cache line (cache miss)- slow access to main memory
- Can see ~10x slowdown



Cache Thrashing

- In C, cache-friendly order is to make last index most quickly varying

Good

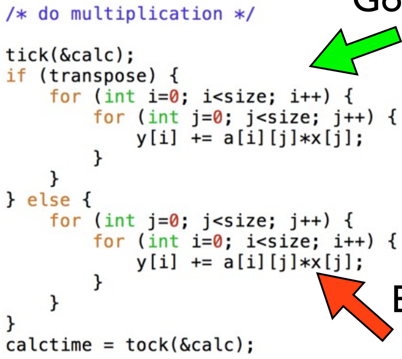
```
/* do multiplication */

tick(&calc);
if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
calctime = tock(&calc);
```

Bad

Cache Thrashing

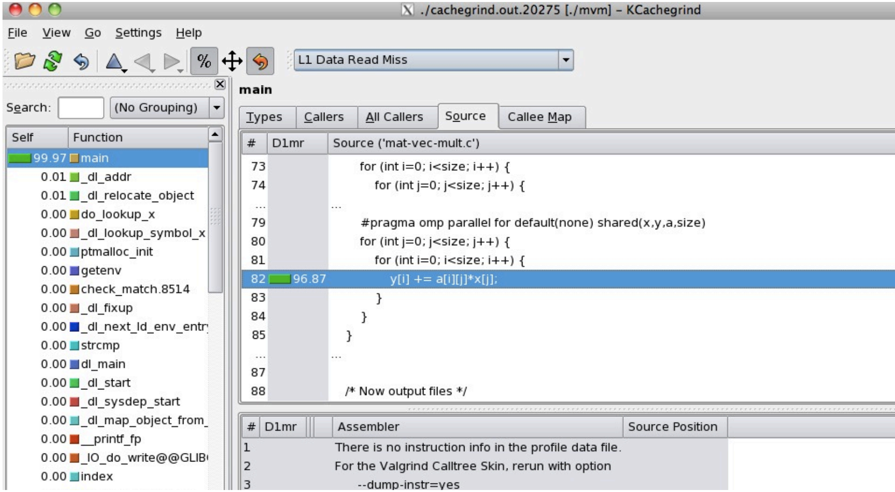
- Can see cache problems with valgrind + visualizer:
- valgrind --tool=cachegrind
- KDE tool kcachegrind available for window,s linux, mac os x.

Good



```
/* do multiplication */

tick(&calc);
if (transpose) {
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            y[i] += a[i][j]*x[j];
        }
    }
} else {
    for (int j=0; j<size; j++) {
        for (int i=0; i<size; i++) {
            y[i] += a[i][j]*x[j];
        }
    }
}
calctime = tock(&calc);
```

Bad



```
./cachegrind.out.20275 [./mvm] – KCachegrind
File  View  Go  Settings  Help
L1 Data Read Miss
Search:  (No Grouping)

Self    Function
99.97   main
 0.01   _dl_addr
 0.01   _dl_relocate_object
 0.00   do_lookup_x
 0.00   _dl_lookup_symbol_x
 0.00   ptmalloc_init
 0.00   getenv
 0.00   check_match.8514
 0.00   _dl_fixup
 0.00   _dl_next_ld_env_entr
 0.00   strcmp
 0.00   dl_main
 0.00   _dl_start
 0.00   _dl_sysdep_start
 0.00   _dl_map_object_from_
 0.00   __printf_fp
 0.00   _IO_do_write@@GLIB
 0.00   index

main
Types  Callers  All Callers  Source  Callee Map

#    D1mr    Source ('mat-vec-mult.c')
73              for (int i=0; i<size; i++) {
74                  for (int j=0; j<size; j++) {
...          ...
79              #pragma omp parallel for default(none) shared(x,y,a,size)
80              for (int j=0; j<size; j++) {
81                  for (int i=0; i<size; i++) {
82   96.87              y[i] += a[i][j]*x[j];
83                  }
84              }
85          }
...          ...
87
88          /* Now output files */

#  D1mr    Assembler                                        Source Position
1          There is no instruction info in the profile data file.
2          For the Valgrind Calltree Skin, rerun with option
3              --dump-instr=yes
```

kcachegrind viewing output of

```
$ module load valgrind
$ valgrind --tool=cachegrind ./mvm --matsize=250
$ kcachegrind cachegrind.out.20275
```

Cache Thrashing

- Once cache thrashing is fixed, and assuming I/O can't be improved, Init is now the bottleneck!
- So it goes...

```
$ ./mvm-omp --matsize=2500
   --transpose --binary
Timing summary:
  Init: 0.00947 sec
  Calc: 0.00811 sec
  I/O : 0.14881 sec
```

Other Profiling Tools

- Scalasca
- Open SpeedShop
- TAU Performance System
- HPC Tool Kit
- Allinea MAP
- Intel Tools (Vtune, ITAC)
- Xcode (OS X)

Profiling Summary

- Put your own timers in the code in/around important sections, find out where time is being spent.
    - if something changes, know in what section
- `gprof` is easy to use and excellent at finding where the time is spent.
- Know the 'expensive' parts of your code and spend your programming time accordingly.
- `valgrind` is good for all things memory; performance, cache, and usage.