MIRIAD Benchmarks. I. Linux-Pentium and Solaris-Sparcs

"no single measurement can predict the performance of a computer system for a particular application, except for measuring it in that application"[1]

Peter Teuben

Astronomy Department, University of Maryland

Version: 1-aug-2000

This is an evolving document[2]

Abstract

We introduce the *MirStone*, a dimensionless measure of how well a standard set of `MIRIAD` programs perform on a given machine. It is simply computed from the elapsed time the benchmark script takes on an otherwise idle machine. It is normalized to be roughly unity on a typical late 20th century desktop (Ultra-300 or Pentium-400). A *MirStone* is defined as 5 divided by the elapsed time (in minutes) the benchmark takes to run.

The benchmark creates 3 artificial visibility datasets in the 9-element BIMA A, B and C arrays, then combines, maps, and deconvolves them. The total disk space required is about 350MB.

The current (Version 1) benchmark has been tested on a series of UltraSparcs running Solaris and Pentium PCs running Linux, in order to compare these configurations. We measured performance under a series of circumstances: varying CPU and memory, local and NFS disks, single and dual processor, single processor dual load etc.

Given enough memory, Linux makes efficient use of disk cashing, outperforming Solaris by 20-30% on otherwise similar CPU performance. Linux also scales a little better when two benchmarks compete for I/O on the same CPU, but competing I/O's remain terrible for performance on both architectures. One actually gains time (often as much as 50%) if they are run "serially", as opposed to "in parallel" on a single CPU. On a dual processor (SMP) machine that loss is not as large, but still noticeable. (i.e. buying SMPs just for `MIRIAD` is not the best idea, but could still be a useful thing to have if the second process is CPU bound).

On Pentium PCs the classic "SCSI is better than IDE" is no longer true for this benchmark, as long as correctly (DMA) tuned drives are used for EIDE. Both SCSI and EIDE drives performed equally well, even when in competing I/O mode. Linux laptops now also compete very well with mainstream desktops.

[1] Eric W. Greison, AIPS Memo 85

[2] BIMA Memo 81

1 Introduction

"Arguing the wonderfulness of benchmarks is like arguing how many angels can dance on the point of a pin."[1]

Performance benchmarks are by far the most pervasive of testing procedures. There are quite a few reasons to benchmark your own code. First of all, simple vendor specifications such as the CPU clock speed, disk I/O speed are all factors that can enter the performance of a specific application in complicated ways. Also, the mere publication of a linpack Mflop rating, or the commercial SpecInt95 or SpecFloat98 are just snapshots of how a particular piece of code performs on a particular piece of hardware. In this sense our benchmark is no different.

The design of which `MIRIAD` applications make up the benchmark is of course crucial. They should excersize a fair amount of the "most expensive" and "most common" code. For Version 1 of this benchmark we took a "simple mapping and de-convolution" [2] excersize. It is one of the more CPU intensive operations, and also fairly disk I/O intensive. Also, the benchmark should not be too quick, as it will (especially in the future) produce unreliable numbers. But it should also not be too long, as this will enable one to run a series of benchmarks with varying degrees of parameters to investigate current and future dependencies on things like amount of memory, diskspace, cpu and bus speed, operating system etc.etc.

2 Benchmarks

... there are lies, damn lies and benchmarks....[3]

Some common benchmarks which measure system performance in some known way are:

1. Bonnie: unix disk I/O benchmark
   URL: `http://www.textuality.com/bonnie/index.html`.
   Another popular disk I/O benchmark is iozone, and linux distributions come with the `hdparm` program that measures a very raw I/O (read) performance.

2. linpack: linear algebra FLOPS. Useful as a generic floating point performance measure for the CPU, but rather problem specific.

[1] unknown

[2] Radio synthesis telescopes are arrays of antennae that measure the components of the Fourier transform of the radio sky. The `MIRIAD` package is then used offline and serves as the "eyepiece" of the telescope, transforming those components into an image and correcting for instrument artifacts as well as instrument and atmospheric errors.

[3] John "maddog" Hall

3. unixbench/byte bench: overall operating system performance. BYTEmark v2:
   URL: `http://math.vanderbilt.edu/~ mayer/linux/bmark.html`
4. SPEC (SPECint95, SPECfp95): commercial benchmarks, useful for predicting numerical performances. There are several versions, 92, 95 and 98. These were needed as CPUs outsmarted the algorithms.
5. DDT ("*Classic AIPS*") produces an AipsMARK. *MirStone* is loosely modeled after this benchmark.

3 Scripts

Currently the benchmark is a simple C-shell script, and can be found in the binary linux distributions as `$MIR/local/install/mir.bench`. You can also find it in `astromake`[4] and on the MIRIAD web page, `http://bima.astro.umd.edu/miriad`. Here is an abbreviated version (without the command line parsing and other bells and whistles) of `mir.bench`:

```
#! /bin/csh -f

set tmp=bench$$
mkdir $tmp
cd $tmp


set nchan=32
set mapsize=1024
set cell=0.5

echo "MIRBENCH: (1999-02-12) $tmp : nchan=$nchan mapsize=$mapsize"
echo `hostname`
echo `uname -a`
echo `date`

echo "1,0,0,0,0,0,0,0,0" > point.source
#set corr=0,1,100.0,1000.0
set corr=$nchan,1,100.0,1000.0
set spectra=1.0,100.0,0.1
set harange=-6,6,0.01
set gnoise=0.1
set pnoise=10,0,0,0
```

[4] `astromake` is a (Linux binary) distribution of a few popular astronomical software packages, see `http://bima.astro.umd.edu/nemo/linuxastro/astromake`

```
set systemp=75,290,0.15
set tpower=0,0
set jyperk=150
set ant=($MIRCAT/bima9_a.ant $MIRCAT/bima9_b.ant $MIRCAT/bima9_c.ant.equ)


time uvgen out=vis1 harange=$harange source=$MIRCAT/point.source \
 corr=$corr spectra=$spectra ant=$ant[1] \
 gnoise=$gnoise pnoise=$pnoise systemp=$systemp tpower=$tpower jyperk=$jyperk \
 > uvgen1.log
time uvgen out=vis2 harange=$harange source=$MIRCAT/point.source \
 corr=$corr spectra=$spectra ant=$ant[2] \
 gnoise=$gnoise pnoise=$pnoise systemp=$systemp tpower=$tpower jyperk=$jyperk \
 > uvgen2.log
time uvgen out=vis3 harange=$harange source=$MIRCAT/point.source \
 corr=$corr spectra=$spectra ant=$ant[3] \
 gnoise=$gnoise pnoise=$pnoise systemp=$systemp tpower=$tpower jyperk=$jyperk \
 > uvgen3.log

time uvcat vis=vis1,vis2,vis3 out=vis > uvcat.log
# echo May need to add 'line=channel,$nchan' to invert for the old versions...
time invert vis=vis map=map1 beam=beam1 imsize=$mapsize cell=$cell > invert.log
time clean map=map1 beam=beam1 out=clean1 > clean.log
time restor model=clean1 beam=beam1 map=map1 out=cmap1 > restor.log
```

4 Running and Results

Before running the benchmark script, make sure that:

- The workstation has enough free memory so that swapping is not an issue. Although in default mode this benchmark only needs 12MB of memory to run, a minimum amount of 64M is recommended. Most modern machines like to use memory to cache I/O buffers which can significantly improve runtime. In particular Linux with 384MB or more memory performs very well.

- The machine is idle, no other active users, preferably there there is only one user. It is not necessary to be in single user mode. On low memory machines (64MB, possibly even 128MB or less) avoid using X windows.

- Enough disk space exists in the current working directory (350MB for the standard benchmark). There is anecdotal evidence that for disks that are close to filling up, the benchmark becomes unreliable (perhaps due to too much fragmenting).

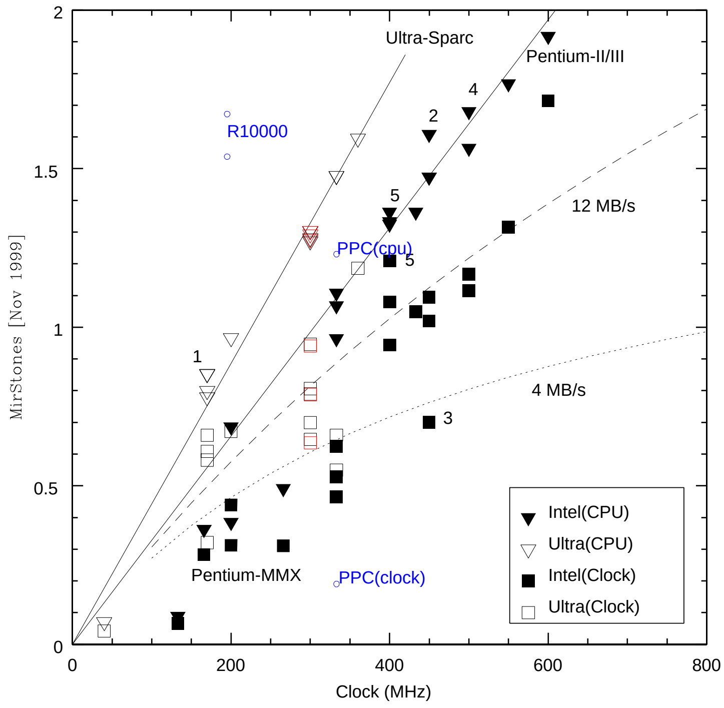

Figure 1: MirStones: square symbols represent the actual *MirStone*, an inverse wall-clock time based measurement (1 MirStone represents 5 inverse minutes elapsed time), whereas the triangles measure pure CPU ignoring any system and other I/O needed. The solid lines are linear estimates of the average Sparc and Pentium CPU performance, whereas the dashed and dotted curves are expected MirStone curves for effective (100%) disk I/O performance of 12 and 4 MB/s resp. Various deviations from the norm should be noted: 1) Sparc optimized compiler, 2) Pentium optimized compiler, 3) non-UDMA disk, 4) Athlon K7, 5) SCSI/Xeon w/ 384MB. 6a) 5400rpm EIDE 6b) 7200rpm EIDE 6c) 7200rpm SCSI-U2W

Run the benchmark as follows:

```
time ./mir.bench [nchan=32] [mapsize=1024] [cell=0.5]
```

which should produce an output like the following:

```
MIRBENCH: (1999-02-12) bench12185 : nchan=32 mapsize=1024
bop6.astro.umd.edu Linux bop6.astro.umd.edu 2.2.10 #2 SMP Mon Jul 19 11:12:31 ED
T 1999 i686 unknown Mon Oct 11 00:46:49 EDT 1999
18.130u 0.180s 0:18.61 98.3%    0+0k 0+0io 711pf+0w    (uvgen)
18.130u 0.160s 0:18.36 99.6%    0+0k 0+0io 398pf+0w    (uvgen)
18.190u 0.140s 0:18.83 97.3%    0+0k 0+0io 398pf+0w    (uvgen)
6.280u 0.650s 0:17.35 39.9%     0+0k 0+0io 8101pf+204w (uvcat)
### Informational:  The default for the "line" parameter is now all channels
### Warning:  Visibilities rejected: 10598
53.290u 13.300s 4:36.92 24.0%   0+0k 0+0io 40640pf+260w (invert)
60.470u 2.470s 1:15.18 83.7%    0+0k 0+0io 18799pf+0w   (clean)
28.620u 4.920s 1:26.20 38.9%    0+0k 0+0io 42441pf+0w   (restor)
203.130u 21.830s 8:31.69 43.9%  0+0k 0+0io 112234pf+521w  (mir.bench)
```

The first 7 timings are from the individual programs, the last and 8th one summarizes the script, and determines the *MirStone*. The elapsed time being `8:31.69` computes a MirStone value as `MS = 300/511.69 = 0.59`. Figure 1 shows a summary of all MirStones for a number of Pentium/Linux and Sparc/Solaris machines.

In the example above the 4th, 5th and 7th line (`uvcat`, `invert`, and `restor`) have an appreciable I/O overhead. In running this benchmark on many machines, it is primarely `invert` and `restor` which are responsible for I/O overheads on either low memory[5] Linux machines or Solaris machines in general.

5 Some mir.bench details

There are a few parameters in the `mir.bench` script that determine the size of the problem:

```
nvis=43200      (cannot be easily changed in the script, see harange=)
nchan=32        (is also a command line parameter to the script)
nsize=1024      (is also a command line parameter to the script)
```

these determine disk and memory usage, and could in principle be scaled as machines get more powerful. However, as we shall discuss, this is not good practice, since an operating system will never (linearly or quadratically) scale benchmark values as you would expect (see e.g. Figure 4).

The following `MIRIAD` programs (see Figure 2) are used by the benchmark:

[5] "low memory" in linux appears to be defined as where all I/O buffers of the problem will not fit in physical memory, about 384MB for this benchmark

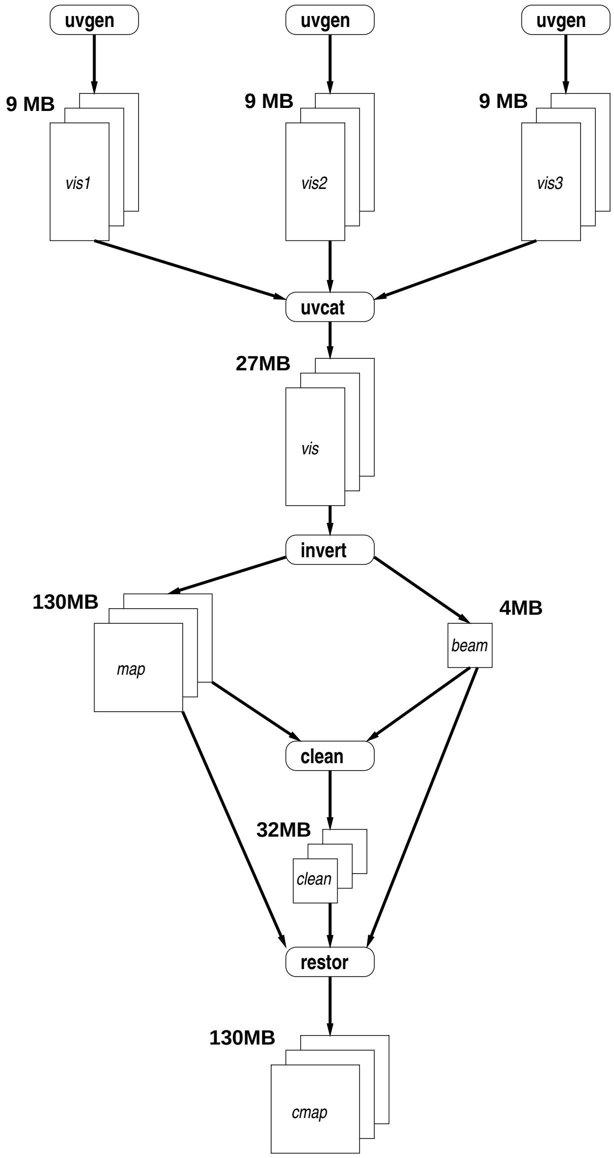

Figure 2: Flow diagram of the benchmark, as explained in Section 5

`uvgen` generate `nvis*nchan` complex numbers (stored as scaled 2 byte integers; should be a quick but mostly CPU bound, `MIRIAD` program). We create

3 datasets : VIS1, VIS2 and VIS3, one for each array configuration.

`uvcat` reads 3 of these datasets, combines them (meant to see how pure I/O works with `MIRIAD` datasets) to produce a bigger VIS dataset for `invert`.

`invert` reads VIS = `3*nvis*nchan`, sorts and grids and FFTs them into a MAP = `nsize**2*nchan` cube of 4-byte reals, and also a BEAM which is `(nsize/2)**2*nchan` [the point source response] `invert` is not only CPU intensive (sorting, gridding, FFT) but also I/O intensive.

`clean` reads MAP and BEAM cubes, and computes a CLEAN cube by decomposing the MAP into many BEAM responses. This program is mostly CPU intensive.

`restor` reads MAP, BEAM and CLEAN and computes the CMAP by taking the CLEAN components and adding in from MAP whatever was not taken out in CLEAN. Like `invert`, this program is pretty I/O and CPU intensive.

Note that `nchan` does not increase the memory usage of the programs, it merely scales the datasize. This is because the data is operated on one channel at a time (the channel size, `(nsize*nsize)` determines the memory usage). So, `nchan` should simply scale linearly with the computing time needed. Looking at Figure 3, where times are normalized to `nchan`, we see this is roughly the case for user time, but that system time slowly increases with the number of channels, with two slopes. For Linux (a 384MB machine) the breakdown occurs around 384 channels, where disk I/O buffers have filled memory and '"swapping" will occur. For Solaris (a 796MB machine) the degradation appears to be non-linear and grow. See also Figure 4 for a more detailed look at this dependance.

6 Remaining Issues

This paper will be an evolving paper, since hardware and software are forever changing closely following close to Moore's law. Here are some remaining issues, some of them where we can probably expect improvements over the next few years:

- 5400 rpm vs. 7200 rpm vs. 10000 rpm disks: current disks become faster (and hotter) as their rpm increases. The decrease in clock time on a 128M Piii-500 is about 8.3" on 226", or about 4%. This is at 87% CPU usage, where memory was not quite enough to optimally make use of memory I/O buffers in linux.
- UDMA-33 vs. UDMA-66: it is unclear if current 5400rpm, and maybe 7200rpm, are fast enough to warrent a UDMA-66 or -100 controller. For example, our current Pentium-600 w/ a 7200 rpm drive measures around

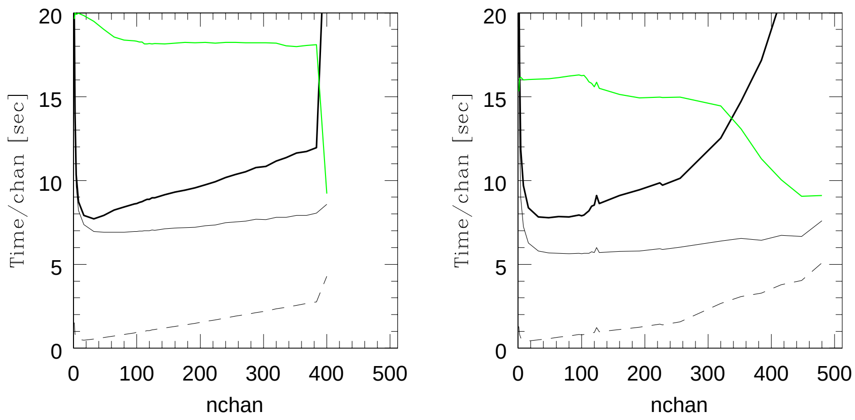

Figure 3: Performance curves for Linux (left, 384MB) and Solaris (right, 768M-B): cpu real (thick solid), user (solid) , and system (dashed) times per channel, as well as the normalized (user+system)/real CPU utilitization curve on top. Notice that the user cpu indeed is fairly independent of `nchan`, as one would expect, but that Solaris, despite the huge amount of memory has a bad and decaying CPU utilitization.

20 MB/sec on the standard UDMA-33 controller. Only if multiple drives in a RAID configuration are used will such controllers make sense.

- NFS: not too surprisingly, this makes a huge difference (from 7' to 14' and 18') on a 10Mbit ethernet; on a 100Mbit ethernet the difference is still measurable, but not as large. In addition Linux NFS performance is still below that of Solaris. The basic lesson here is that in the current configurations one should use CPU and DISK on the same machine. NAS in the future may change this one the wires get fast enough.

- SCSI vs. UDMA/EIDE: no difference on Linux; on Sparc Ultra UIDE performed rather poorly compared to SCSI. The difference between 5400rpm and 7200rpm UIDE drives was also quite noticeable in this case (see the points labeled 6a,b,c at 300MHz in Figure 1).

- EIDE, unlike SCSI, commands are single threaded and rely on the CPU to transfer data. Some manufacturers (e.g. Sun) claim that EIDE is adequate for single-users workstations, but for this kind of work there is a sufficient amount of multitasking to make this noticeable. However, the price difference is quite large, and EIDE based systems may still win in a price performance comparison.

- Filesystem; on Solaris the `tmpfs` filesystem offers some cashing advantages: /tmp being an integral part of their memory model (it appears that on /tmp disk buffers are placed in memory) makes a *big* difference. In Linux increasing the default blocksize from 1k to 4k (`mke2fs -b 4096`) improved the runtime by about 5%.

- IDE channel: master vs. slave. Occupying both channels is supposed to be bad, however on one machine the `hdparam` benchmark did not show any appreciable difference, neither did our own `mir.bench` show any difference. Even if two were run in parallel, one on the master and the other on the slave, it made no difference.

- PC66 vs. PC100 vs. PC133 (i.e. what is the effect of FBS)

- RDRAM : no data available yet, but memory running at higher clock speed will undoubtedly increase performance. Maybe show some 50 - 66 - 75 MHz experiments from that ASUS MB?

- Athlon vs. Intel. Athlons are about 10%-12% faster [SysMark98]. However, MirStones improved only about 7%. These experiments were done with a Redhat executable, not Mandrake (i.e. not pentium optimized).

- compiler: On Solaris the compile option `-fast`, and on Linux the PGCC (pentium enhanced) compiler with option `-march=pentiumpro -mcpu=pentiumpro-O3 -fno-globals` gives an additional 7% (check numbers for Solaris).

- Unix *buffer caching* can make for extremely efficient I/O. These buffers store data in a `write-behind` or `read-ahead` mode. The command `sar -b` will indicate how well this mechanism worked.

7 Tabular results

The following table summarizes various benchmarks and compares them with *MirStones*:

Table 1: UltraSparc and Pentium benchmarks

Machine OS	Freq MHz	SPEC fp95	SPEC int95	MDBNCH sec	A^{93}_{m} [1]	MirStone
P-mmx	166		5.6		3.5	0.28
P-mmx	266					0.31
P-ii	333			17.4	10.4	0.63
P-ii	400	12.4	15.3		14.5	
		12.8	16.9			
P-ii	450					1.02
P-iii	500	14.7	20.6		17.9	1.12
P-iii	550					1.32
P-iii	600	15.9	32.1			1.72
P-iii	800					2.15
AMD-K7	900					1.81
P-???[2]	1100	-	43.0	-	-	
Sparc-IPX	40				**1.0**	0.042
Sparc10/40	40	**1.00**	**1.00**	-	-	
		1.38	1.13	-	-	
Sparc20/51						0.16
Ultra-1	170	9.06	6.26	17.9	6.3	0.32
Ultra-2	200	-	-	14.60	-	0.67
Ultra-5	333	18.3	14.1			
Ultra-10	300	12.9	12.1		8.8	0.81
Ultra-10	360	19.9	15.2			1.19
Ultra-10	440	22.7	17.9	7.35	-	
R4400	4x200					0.34
R10000	175					0.58
R10000	32x195					1.49
Alpha		53.7	32.1			
Alpha 21264	750	75.4	38.3			
PPC	300					0.60

[1] No /tmp tricks used for Solaris

[2] Willamette, Q3 2000

Table 2: Disk performance

Machine OS	Freq MHz	disk	time	MirStone
U-5/10 (atlas)	300	local	234.18u 29.08s 6:12.44 70.6%	0.805
		tmpfs		
		nfs-100	234.19u 27.47s 7:28.78 58.3%	0.668
			233.40u 27.56s 7:44.09 56.2%	0.646
			232.11u 28.25s 8:17.11 52.3%	0.603
		nfs-10	232.34u 25.59s 19:41.70 21.8%	0.254
			272.96u 31.35s 19:43.95 25.7%	0.253

8 Performance

To understand some of the behavior of a benchmark, it is important to study its dependence as a function of the problem size. Our benchmark currently has two size parameters, `nchan` and `nsize`. `nchan` controls diskspace of the datasets and CPU usage in a more-or-less linear fashion, without affecting memory usage, whereas `nsize` controls memory as well as diskspace usage, more-or-less quadratically. Figure 4 plots the linear increase of CPU time with `nchan` and the degradation as swapping sets in.

Here's the formulae how much disk space this benchmark occupies:

$$DiskSize = N_c * (9 * (N_s/1024)^2 + 1.65) MB$$

for reasonable values of N_c and N_s this is summarized in Figure 5.

9 Some background on current disk storage technologies

There are basically two types of "drive controllers": **ATA** for those that use any of the various integrated drive electronics (IDE) protocols, and **SCSI** that use any of the many SCSI protocols. For ATA the current ones include enhanced IDE (EIDE), ATA-2, ATA-3, Ultra33-ATA and Ultra66-ATA. None of these devices use host-based controllers, instead they connect to the host's I/O bus via buffering logic. The operating systems' ATA drivers do most of the work. For SCSI the current ones include Fast SCSI (SCSI 2), Ultra SCSI (SCSI 3) or Ultra2 SCSI. All of these require host-based controllers, and the devices to most of the work.

- disk, drive
- bus controller

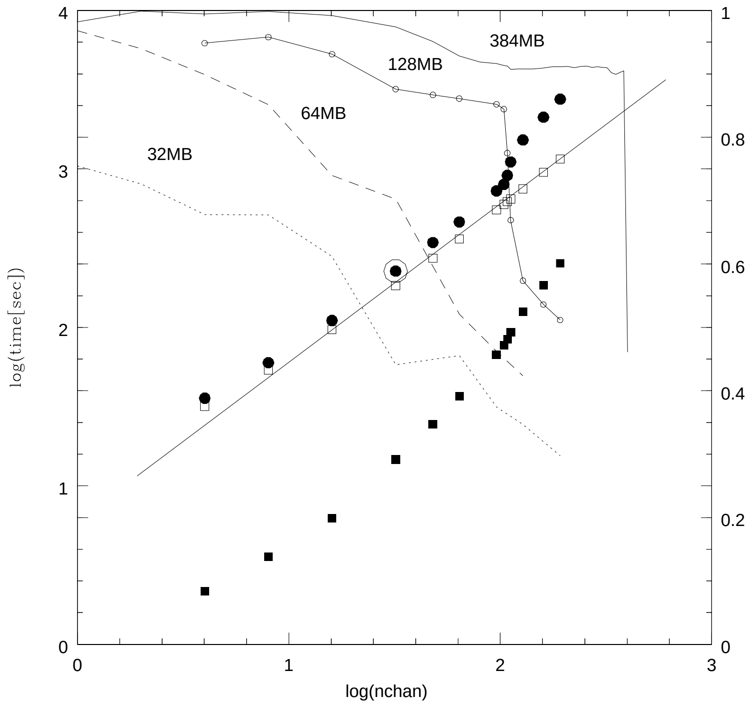

Figure 4: Run-time for the standard benchmark as a function of the number of channels for a 128MB Pentium-III 500 with a 7200 rpm UDMA EIDE disk. Both axes are plotted logarithmically. Closed circles are elapsed time, the standard benchmark at `nchan=32` is marked with a larger open circle. Open squares are the user CPU time, whereas closed quares measure the system (kernel) CPU time. The solid line is the percentage ((cpu+sys)/real) of CPU that has been utilized, and for a 128MB configuration the breakdown due to swapping at around 100 channels is quite dramatic. For 64MB (dashed) and 32MB (dotted) models the breakdown occurs earlier and is not so abrubt.

- drive controller
- ATA
- SCSI
- disk partition
- file system

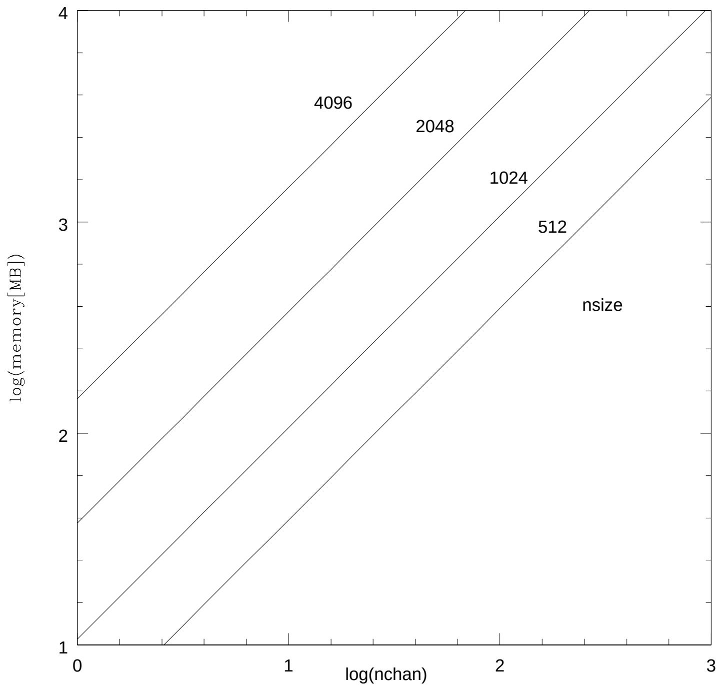

Figure 5: Total size of the dataset for various values of `nchan` (number of spectral channels) and `nsize` (mapsize). $Size(Mb) = Nc * (9 * (Ns/1024)^2 + 1.65)$

10 Future

It may be amusing to try and look into the future and see what kind of performance we can expect in the next 5-10 years. Traditionally CPU speeds have kept up with Moore's law (doubling every 18 months), although I/O speeds (such as networking and disk I/O) have lagged, thereby creating an ever increasing gap between how fast the processor can crunch and how fast it can get the data that it needs to crunch on, creating idle processor time. This suggests that operating in memory ("ramdisk") would be the most efficient use of this process.

11 Conclusions

- Linux EIDE drives need to be running in DMA mode if capable. Some (all?) distributions don't do this by default. Penalty is large, factor of 2.
- Linux with enough memory can run at 100% CPU, Solaris will only do this on a tmpfs (e.g. /tmp) filesystem, otherwise CPU utilization will be typically 50% - 80%.
- NFS on a 10Mbit line is very bad. NFS on a 100Mbit line is almost acceptable.

References

1. "A Retrospective View of Miriad" R.J. Sault, P.J.Teuben and M.C.H.Wright. in: Astronomical Data Analysis Software and Systems IV, ed. R.A. Shaw, H.E. Payne and J.J.E. Hayes. PASP Conf Series 77, 433 (1995). `http://bima.astro.umd.edu/miriad`
2. *Tests of Sparc and Disk Performance on Standard* `MIRIAD` *Tasks,* A. Rudolph and P. Teuben, 8-Jul-91 (BIMA memo 11)
3. DDT, (Dirty Dozen Test), Aips memo 85. See also `http://www.cv.nrao.edu/aips/ddt.html`
4. Usenet newsgroups: `news:comp.benchmarks`
5. **mdbnch**: `http://www.sissa.it/furio/Mdbnch/results.txt`
6. **Linux Benchmarks:** `http://www.silkroad.com/linux-bm.html` (now retired) Look at: `http://math.vanderbilt.edu/~ mayer/linux/bmark.html`

History

- 8-feb-2000: first release
- 1-aug-2000: minor changed, and released as BIMA memo 81