CDF/DOC/COMP_UPG/PUBLIC/5743
B. Ashmanskas, S. Belforte, R. Hughes, P.T. Keener, A. Kreymer,
P. Murat, A. Reichold, M. Shimojima, D. Waters, S. Wolbers, F. Würthwein
October 10, 2001
Version 1.0

Report on Benchmarking CDF CAF Review Fall 2001

Abstract

We report on procedures and results of a variety of benchmarking efforts on a variety of systems, including fcdfsgi2, fcdfsun2, and cdfpca. This was done as part of the CAF computing review between 8/15 - 10/01/2001.

1 Introduction

The intention of the benchmarking effort described here is to provide a set of realistic performance numbers for running common CDF applications on SGI (fcdfsgi2), Sun (fcdf-sun2) and Linux (cdfpca, fcdflnx1, a desktop PC) platforms with and without optimization. Among the applications chosen are both user applications as well as typical code management procedures.

The results are meant to empower the physics groups to assess to what extent the central analysis facilities at CDF are likely to satisfy the physics analysis needs of the collaboration for summer conferences 2002, as well as the analysis of the full Run2a 2 fb^{-1} data set.

The review committee will use these numbers in connection with whatever "expected usage profile" the collaboration/committee comes up with in order to arrive at recommendations regarding the charges to the committee as detailed in Appendix A.

Given that the actual CDF analysis software, especially the I/O system and physics analysis data (PAD) format is still under heavy development we have tried hard to augment the obvious CDF applications with simpler benchmarks that measure pure CPU and pure I/O performance separately.

The structure of this document is such that we briefly summarize all the main findings and then refer to various appendices for details.

2 Overview of the platforms

The platforms tested are are fcdfsun2, a Sun with 24 750 MHz UltraSPARC III processors, and fcdfsgi2, an SGI with 64 300 MHz R12k processors. These two SMPs had 1GB and 0.5GB RAM per processor respectively. The operating systems on these two SMPs were SunOs 5.8 and IRIX64 6.5.

As a Linux/Intel platform we have mainly used cdfpca with 8 700MHz PIII Xeon processors, and fcdflnx1 with 4 processors of the same type. The two linux systems had 1/4 and 1 GB RAM per CPU respectively. Cdfpca had Linux 2.2.16-3smp installed.

We found that the disk systems didn't matter for most of our tests as almost all of them were CPU limited. Further details can be found in Appendix B.

We chose performance/MHz to compare different platforms. We point out that this metric favors SGI and Sun over Intel/Linux as is explained in more detail in Appendix C. The actual dimensions quoted are often "seconds$\times$ MHz" which is the inverse of performance/MHz.

More details on the systems tested can be found in Appendix B.

3 Summary Results

All benchmarks were run using 3.18.0 release of the CDF software unless otherwise noted.

In Table 1, times (sec) are real times. That is, you start your stopwatch when you launch the benchmarking task and stop it when it is complete. It is not a more difficult number to interpret such as CPU seconds. CPU numbers (in percent) are average CPU utilisations. These are actually calculated as (user time + system time)/(real time). Therefore numbers less than 100% indicate that the process was waiting some fraction of the time, usually for I/O. So, the CPU time (i.e. proportional to the number of CPU cycles needed to complete a task) is, to first order, just (real time)*(CPU).

In Table 2 we list all the CPU limited benchmarks again but this time in units of "seconds $\times$ MHz", i.e. the inverse of performance/MHz. In addition Table 2 indicates the relative performance of the different platforms in ().

A number of the benchmarks in Table 2 have only been run on some of the platforms. The missing information on some platforms is indicated by a dashed line.

The results of these tests prompted a variety of more detailed studies which were generally not repeated on all platforms. We summarize below the conclusions of these tests and refer the reader to various appendices for details.

TABLE 1:

Benchmarking Task : Result :	sun2	lnx1	cdfpca	sgi2
ROOT STRESS (ROOTMARKS)	175.50	174.20	146.60	89.00
AC++Dump (Test 1) (MB/s)	13.52	14.93	14.41	7.11
AC++Dump (Test 1) (CPU)	84.04	84.98	88.35	80.63
AC++Dump (Test 2) (MB/s)	2.82	3.65	3.56	1.25
AC++Dump (Test 2) (CPU)	99.58	96.02	97.09	98.84
cdfsim (100 ttbar events) (sec)	991.38	886.98	773.96	failed
cdfsim (100 ttbar events) (CPU)	98.72	87.80	99.94	failed
cdfsim details (seconds):				
Pythia clk time/event	0.11	0.065	0.056	failed
SimulationControl clk time/event	9.39	8.724	7.609	failed
dd (read 1GB sim file) (MB/s)	28.90	18.59	28.40	7.23*
dd (read 1GB sim file) (CPU)	35.97	33.91	26.28	5.06*
dd (read 2GB prod file) (MB/s)	29.02	20.23	31.79	3.77*
dd (read 2GB prod file) (CPU)	37.98	28.07	30.21	2.63*
Production (5000 ttbar evts) (sec)	22462.05	13572.24	13607.84	35453.03
Production (5000 ttbar evts) (CPU)	99.85	99.89	99.80	99.59
Stnmaker (5000 ttbar evts) (sec)	2994.41	2363.12	2447.80	6565.89
Stnmaker (5000 ttbar evts) (CPU)	99.52	99.73	98.51	99.56
Stnmaker details (1e-4 seconds):				
FileInput	751.0	548.4	578.5	1676.4
PuffModule	3672.7	3057.3	3063.8	7872.9
DHOutput	164.8	118.5	131.6	318.5
FillStntuple	55.1	59.6	59.2	148.9
InitStntuple	458.4	316.1	334.1	1215.7
StntupleFilter	6.4	3.8	4.4	10.6
StntupleMaker	0.7	0.3	0.1	0.7
Stntuple (5000 ttbar events) (sec)	5.29	5.34	5.56	13.64
Stntuple (5000 ttbar events) (CPU)	98.3	100.2	102.4	93.5
TrackTest (sec)	317.04	211.49	217.49	4050.95
TrackTest (CPU)	99.65	97.49	97.00	14.91
TrackTest details (1e-4 seconds):				
FileInput clk time/event	13.3	9.2	9.5	23.2
PhiKAnal clk time/event	77.9	49.3	50.4	153.0
TrackAnal clk time/event	21.6	14.7	14.9	40.3

* dd results on sgi2 should not be trusted. See additional remarks below.

A few additional remarks are in order here:

- The load average at start of the benchmarking was 26 as indicated by "uptime" on fcdfsgi2. We do believe that the load on the machine affected only the dd benchmark and possibly the FileInput timing for TrackTest. The reason for this is that we used /cdf/scratch for the benchmarking. This is used by pretty much every user in CDF. It's hard to imagine that it wasn't used while we were doing our tests.
- The dd benchmark on /cdf/data02/s5/ on fcdfsgi2 is 51.5MB/sec, i.e. the FC disks that are datadisks on fcdfsgi2 give reasonable performance.
- The total time for TrackTest on fcdfsgi2 is nonsense. Detailed inspection of the logfile indicates that the job was idly sitting around for 20 minutes after it had completed. It remains a mystery what caused this.
- TrackTest and StntupleMaker CPU times in Table 1 can not be compared with each other in a straighforward fashion as the former was run on full Geant $t\bar{t}$ MC while the latter was run on Quicksim $b\bar{b}$ MC. Running TrackTest on the $t\bar{t}$ sample results in the following for cdfpca:

```
FileInput       0.0596 seconds/event
PhiKAnal        0.347  seconds/event
TrackAnal       0.024  seconds/event
```

  We do not understand the difference in CPU time for the different datasets at this point. Taking it at face value indicates that Stntuple making (0.038 sec/event, dominated by FillStntuple and InitStntuple) is roughly 60% slower than the simple TrackAnal module (0.024 sec/event).
- The linux systems benchmarked here use CPUs that are kind of ancient. To get a sense of performance scaling with clock speed we also ran TrackTest on a 1GHz PIII:

```
                PIII 1GHz:    fcdfsun2:

FileInput        0.001694      0.001181  seconds/event
PhiKAnal         0.003603      0.007725  seconds/event
TrackAnal        0.001139      0.002155  seconds/event
```

  We conclude that a PIII 1GHz is roughly twice as fast for a typical CDF user module than fcdfsun2. This is quite surprising given that the PIII 1GHz and the Ultra III 750MHz are roughly comparable on both fp (U3 beats P3 by 10%) as well as int (P3 beats U3 by 20%) Specmarks, as is detailed in Appendix C.

  Comparing these numbers here with the ones for cdfpca in the summary above we furthermore conclude that P3 700MHz to P3 1GHz scales with clock speed in performance for the user analysis module. It is thus not unreasonable to expect a P4 1.8GHz to be roughly 3.5 times faster than fcdfsun2 .

TABLE 2:

	fcdfsun2	fcdflnx1	cdfpca	fcdfsgi2
cdfSim:				
ttbar	7425 (1.2)	6202 (1.0)	5418 (0.9)	---
ProductionExe:				
ttbar cdfSim	3369 (1.8)	1900 (1.0)	1905 (1.0)	2127 (1.1)
PuffModule (puffing up the event):				
ttbar cdfSim	275 (1.3)	214 (1.0)	214 (1.0)	236 (1.1)
FileInput (reading in an event):				
ttbar cdfSim	113 (1.5)	77 (1.0)	81 (1.1)	101 (1.3)
Stnmaker (making an Stntuple):				
on data	--	--	61	--
ttbar cdfSim	35 (1.3)	27 (1.0)	28 (1.0)	41 (1.5)
TrackAnal (D+ skim using CTVMFT):				
cdfsim ttbar	25 (1.5)	--	17 (1.0)	--
run 126903 jr	14 (1.5)		9.5 (1.0)	--
run 126903 jr no CTVMFT fits	--	--	3.4	--
quicksim bbbar	1.6	1.0	1.0	1.2
Stntuple analysis	0.79	0.75	0.78	0.82

Let us next summarize some of the additional studies that are detailed in the appendices.

3.1 Attempts at benchmarking maxopt

At present only a fraction of the CDF software compiles under maxopt on all platforms. In particular, we gave up trying to compile CDF software under maxopt on the sun. In addition, some packages, for example the COT tracking code, are reputed to lead to different results under maxopt than under noopt. It goes well beyond the scope of this committee to disentangle what works and what doesn't under maxopt.

The one program for which we might have more or less reliable maxopt vs noopt results on all platforms is RootStress. The ROOTMARKS performance maxopt/noopt ratios for sun/sgi/cdfpca are 1.3/1.8/1.4.

We also have results from Edm_ReadWriteSeqRoot_ByEvent indicating maxopt/noopt ratios of 2.3/1.8/1.8 for sun/sgi/cdfpca, but the sun result here was obtained on fcdfsun1 and not fcdfsun2. Additional details may be found in Appendix G.2.

Finally, we have CT_Reconstruction numbers but here the problem is that we were told that the maxopt and noopt output of CT_Reconstruction do not fully agree. While we have not verified this claim ourselves it remains unclear to what extent maxopt is at all useable here.

In summary, we were unable to clearly evaluate maxopt. From what we have done we see no clear indication that the gains that may exist are significantly larger on one platform than the other.

3.2 AC++ I/O modules

It is not exactly a secret that the total data bandwidth seen by a user module in AC++ pales in comparison to the bare disk I/O capacity.

This can be seen for example by looking at the time Stnmaker spends in the FileInput module given the 400kB event size for the $t\bar{t}$ Monte Carlo sample used in this test. For example on fcdfsun2 we find an effective maximum bandwith through the FileInput module of 5.3 MB/sec. This is a factor 5.5 smaller than the bare disk I/O bandwidth as measured using dd.

We have made various attempts at understanding this which are described in detail in Appendix G. The bottom line appears to be that the bare Edm (as measured with Edm_ReadWriteSeqRoot_ByTBuffer) is well capable of reaching 27MB/sec at only 45% CPU utilization. However, unpacking the objects presently requires significant CPU resources.

A particularly good example for "CPU limited I/O" is CdfTrack. Even after a factor 1.6 increase in speed from release 3.18.0 to release 4.0.0, CdfTrack still takes roughly 2.7 times more CPU per MB I/O than the rest of the event. A $t\bar{t}$ MC event on cdfpca after dropping CdfTracks still has only 12 MB/sec of I/O as compared to 28MB/sec as measured with dd for block sizes from 200Bytes to 1MB.

As an aside, we point out that CdfTrack makes up only ~12% of the event size in this sample. In fact no one object takes up more than ~20% of the event, and 18 objects each comprising more than 1% make up 93.3% of the total. We consider this very promising for multi-branch I/O, assuming no one object dominates CPU consumption.

We particularly recommend that CPU consumption and the advent of multi-branch I/O be taken into account when designing the PAD format. A PAD format that reduces event size at the cost of CPU power, or combines non-essential with essential information into the same storable objects may not be an optimal choice.

3.3 Run1 vs Run2 CPU/event

In order to compare Run1 and Run2 software CPU requirements per event we have dug out some logfiles for a user analysis skim on the "old" cdfsga in 1996, and compared the MHz $\times$ seconds/event as implied by these logfiles with the performance of TrackTest and Stnmaker on our $t\bar{t}$ sample as well as recent Run2 data. Given the issues already mentioned with I/O into AC++ we compare only the module timing provided with "show timer" in AC++ with total execution time as well as AC timing info in Run 1.

Run2 software appears to require roughly an order of magnitude more CPU power per event than Run1 software. The details can be found in Appendix H.

3.4 Code management Issues

We have attempted to at least document the CPU time spent during the nightly rebuild as well as the building of a user test release.

In a nutshell, noopt builds take roughly equal amounts of time on Sun and Linux, and four times longer on fcdfsgi2. However, the comparison may be somewhat unfair as we had no control over usage of fcdfsgi2 while the test ran while there were no other users on the other systems during the tests.

Apart from the time it takes to build we noticed the following "features":

- The partial maxopt build presently possible under sun takes 2-3 times longer than a noopt build. If one were to build maxopt as main release than roughly 16 of the 24 processors on fcdfsun2 would be required for reasonably efficient release building. A noopt build keeps 8 out of 24 processors busy for 2 hours. A 2 hour build time is required in order to provide timely feedback to developers when building our weekly integration releases.
- In 3.18.0 SunOS requires 1.5GB of real memory to build a single ProductionExe. A minimal exe like AC++Dump still needs 935MB of real memory to build. fcdfsun2 has 1GB RAM/CPU.
- ProductionExe size on sun/sgi/cdfpca is in the ratio 3.3/1.6/1.0 . Other exes follow

the same pattern. The actual size of an exe in 3.18.0 on sun ranges from 465MB to 300MB.

- We did not make any attempt at running a mix of exes and compilations simultaneously on the sun to investigate performance limitations due to real memory starvation because fcdfsun2 starts swapping long before its real memory is exhausted. We consider this something that can probably be resolved by careful "tuning" of the system.

Additional details can be found in Appendix I.

Note added: Memory requirements have decreased by about 30% since release 4.0.0, due to code restructuring.

4 Cost per MHz

Given the performance issues with run2 code as compared with run1 code we feel compelled to comment at least in passing on the costs per MHz of compute power for various platforms:

- Sun: 24 CPU 750 MHz for $400,000 or $22/MHz.
- Intel: 8 CPU 700 MHz for $50,000 or $9/MHz.
- Intel: 2 CPU 1000 MHz for $1,600 or $0.8/MHz.

It might be worthwhile to point out that the recent L3 farms purchase of 120 dual PIII 1GHz cost less than $1200 per node or $0.6/MHz. It is furthermore of some relevance that the cost for the sun does not include a 3 year maintenance contract of $100k.

To be fair, one should point out that these costs do not include the human resources to purchase, install, integrate, tune and manage these systems.

5 Conclusions

- There are still significant setup issues for fcdfsun2 to be resolved.
- We consider the porting of CDF software to the sun platform at best incomplete and inefficient. If we were to believe that the port is efficient then we would have to conclude that SunOS itself is rather inefficient. Whether or not this is an issue related to CDF's use of KAI instead of a native compiler is impossible for us to assess.

- Performance improvements with maxopt are presently more at the level of $\sim 30\%$ than a factor of 2. It is conceivable that this might improve in the future when the focus shifts from algorithm development to performance tuning.

- Reading of storable objects in CDF is severely CPU limited, and differs between different objects by at least a factor ~ 3. CdfTrack was shown to have particularly large CPU needs.

- Run2 analysis modules (i.e. ignoring I/O issues mentioned above) require roughly an order of magnitude more CPU power per event than similar modules in Run1. We do not understand the origin of this discrepancy.

- We recommend that the ongoing design of the PAD format take CPU consumption as well as multi-branch I/O into account. In particular, a PAD format that shrinks event size at the cost of CPU power, or combines essential with non-essential information into the same storable object is unlikely to be optimal.

- In light of the apparent difference in performance of run1 and run2 user analysis skim software we feel compelled to point out the cost differential in \$/MHz of processor power in an SMP ($\sim \$20/MHz$) versus a dual processor PC ($< \$1/MHz$). We expect this difference to increase in the near future as dual P4 become as common as dual P3 are at present.

A Charge of the Review

A.1 Charge 1:

CDF has purchased a Sun Fire 6800 computing system with 24 750 MHz SMP nodes capable of being seen as a single system image. The nodes will be upgraded to 900MHz in September. CDF's plan is to purchase a second identical Sun Fire 6800 system in FY01 or FY02, if the performance of the first system with the CDF software is adequate.

The most immediate charge to this committee is to review the performance of this Sun system, fcdfsun2, and recommend to the offline whether or not they should proceed with the purchase of a second sun system. This should be done in the context of specific performance requirements driven by expected Run II datasets that will be available (a) for summer 2002 analysis (300pb-1) and (b) for completed Run IIa analyses (2fb-1).

fcdfsun2 should be available to users near the end of August. The committee should begin by evaluating the performance of running the offline software on fcdfsun1, a four processor 400 MHZ machine, and extrapolate to the performance of fcdfsun2. Before the completion of the report, this extrapolation should be explicitly verified on fcdfsun2.

If the committee decides to recommend against the purchase of a second Sun machine, the committee should suggest alternative uses for the funds that were earmarked for the second Sun, approximately $$400K, within the central systems.

A.2 Charge 2:

The committee will review the CDF central analysis systems and recommend directions for FY02 and FY03. The committee is invited to review both the current technical choices within the CDF centralised computing model, and the model itself, and recommend directions either inside our outside the current model. The committee may consider some or all of the following:

1. The existing and planned central analysis systems: fcdfsgi2, fcdfsun1, fcdfsun2, fcdflnx1.
2. The storage and CPU resources available to these and future systems.
3. Recommended UNIX flavors in the central systems in the future, restricted to the three possibilities IRIX, Linux and Solaris.

```
   4.Possible future farm-like systems for batch driven central analysis.
   5.Possible future distributed computing environments.

The report on this "long term" charge will provide important input for
planning for RunIIb, which will commence late this fall.
```

B Details on the platforms

```
                 fcdfsgi2:  SGI Origin 2000
                 ==========================

 Processor/memory info:
 ----------------------
 * 64 300-MHz MIPS R12000 processors
 * 32 GB main memory

 Disk info:
 ----------
 * Local SCSI disks in chassis slots:  /, /tmp, /var/mail, etc.
         3   x    9 GB
         5   x   18
         9   x   36

 * Local SCSI-attached RAID (in external chassis with RAID controller etc.):
         /cdf/code       300 GB
         /cdf/scratch    500 GB

 * Local fibre-attached RAID (all areas /cdf/data*):
         21 arrays       330 GB per array
    Details:
         data01-06               assigned to physics, offline, detector groups
         data08                  look area, managed by DAQ group
         data07, data09-data18   managed by DIM
         data19-data21           to go into DIM; temporarily assigned elsewhere

 * NFS mounted from NetApp file servers:
         /cdf/home       270 GB          (mounted from fcdfhome)
         /cdf/spool      645 GB          (mounted from fcdfspool)



                 fcdfsun2:  Sun Fire 6800
```

```
                ========================

Processor/memory info:
----------------------
* 24 750-MHz UltraSPARC-III processors
* 24 GB main memory

Disk info:
----------
* Local SCSI disks:
        6   x   18 GB

* Local SCSI-attached RAID (in external chassis with RAID controller etc.):
        /cdf/code       300 GB

* NFS mounted from NetApp file servers:
        /cdf/home       270 GB          (mounted from fcdfhome)
        /cdf/spool      645 GB          (mounted from fcdfspool)



                fcdflnx1:  SGI Linux server
                ===========================

Processor/memory info:
----------------------
* 4 700-MHz Pentium III Xeon processors
* 4 GB main memory

Disk info:
----------
* Local SCSI disks:
        5   x    9 GB
        1   x   18
        1   x   50

* Local SCSI-attached RAID (in external chassis with RAID controller etc.):
        /cdf/scratch    500 GB
        (300 GB /cdf/code array planned but waiting for additional disks)

* NFS mounted from NetApp file servers:
        /cdf/home       270 GB          (mounted from fcdfhome)
        /cdf/spool      645 GB          (mounted from fcdfspool)
```

```
                cdfpca:  Dell server
                ====================

Processor/memory info:
----------------------
* 8 700-MHz Pentium III Xeon processors
* 2 GB main memory

Disk info:
----------
* Local SCSI disks:
        4       72 GB   (3 combined in software RAID set)

* NFS mounted from NetApp file servers:
        /cdf/home       270 GB          (mounted from fcdfhome)
        /cdf/spool      645 GB          (mounted from fcdfspool)
```

C Computing in terms of SPECint92

In the main body of the text we compare different CPU's solely by their clock speed. Here we describe an alternative metric, namely to compute publicly available SPEC2000 numbers into "equivalent" SPECint92. We choose SPECint92 here because that is one of the metrics mentioned in the original CAF specs [?]. To convert to slightly less ancient metric like SPECint95 just divide SPECint92 by roughly 45 [3].

For simplicity, we use the conversion factors [3]:

$$SPECint92/SPECint2000 \quad = \quad 4.5$$

In addition, we quote SPECfp2000 as well as fraction of SPECfp2000 performance of a P4 1800MHz.

System Chip/MHz		SPECint92	SPECint2000	SPECfp2000	rel. perf.	Like
R12K	300	1150	254	269	0.44	fcdfsgi2
R14K	500	1850	410	362-436	0.65	new sgi
USIII	750	1650	369	323	0.52	fcdfsun2
USIII	900	2000	439	369	0.60	fcdfsun2 Sept upgrade
P3	700	1400	307	205	0.33	cdfpca
P4	1800	2700	599	615	1.0	Dell 330

We point out that the ratio of specmarks/clock speed for sun and sgi is larger than for Intel. In the body of this report we talk about performance/MHz whenever we compare different architecture CPU's. Performance in those cases is of course measured for CDF applications.

If we were to use performance/specmark instead of performance/MHz than sgi and sun would look worse than they do now when compared to Intel. In the body of the report we are thus being charitable to both sun and sgi when comparing them to the Intel/Linux platform.

D The Benchmarking Programs

All CDF applications are based on release 3.18.0, the latest frozen release at the time the benchmarking started. Some patches had to be applied and are described in Appendix A.

All the scripts to run these benchmarks are stored in cvs in "Validation/benchmarking". This includes any tcl files to control the various AC++ jobs.

Apart from running the benchmarks listed below on the various platforms we also studied two issues in more detail:

- performance improvements due to maxopt. Results are presented in Appendix E.
- I/O performance of some of the presently existing storable objects. Results are presented in Appendix G.

D.1 CPU intensive benchmarks

1. **tiny**: Used only for simple CPU scaling behaviour. We find that CPU scales as expected. See section F.
2. **cdfSim**: geant based simulation of $t\bar{t}$ events in the CDF detector. No trigger cuts were applied at generator level. Average event size is 190kB.
3. **ProductionExe**: run full reconstruction on the $t\bar{t}$ sample above. Average event size is 400kB.
4. **AC++ user analysis module**: An AC++ executable with two user analysis modules is run on a MC sample of generic $b\bar{b}$. The sample was generated using Pythia followed by a fast simulation (GenTrig/QuickSim). The Edm objects accessed are thus only CdfTracks, CdfTrackColl, HEPG, MVTX. Average event size is 19kB.

- TrackAnal: look for $D^+ \to K^-\pi^+\pi^+$, kinematic cuts, d0 cuts, vertex fit using CTVMFT. Vertexing is only run after kinematic and d0 cuts, of course. There are 9762 candidate D^+ in 23982 events, all of which are fit using CTVMFT, i.e. 40% of all events require a vertex fit.
- PhiKAnal: look for $B^+ \to \phi K^+$, $\phi \to K^+K^-$, same as above but also fills a ton of histograms, ntuples, more complicated selection, in short, it's a realistic analysis module someone might use to do physics with. However, it would be silly to use something as involved as this for skimming on the CAF. TrackAnal would be more appropriate for that.

5. **CT_Reconstruction**: The drift chamber reconstruction code is one of only a few things we were able to benchmark with different levels of optimization. And even this didn't work on all platforms.

D.2 I/O intensive benchmarks

1. **dd**: Bare disk I/O speed measurement. We did this for 3 different block sizes, 5kB, 100kB, and 1MB. The rational here is to measure I/O for block sizes of a typical ntuple and PAD event, as well as the "asymptotic limit". We did not observe any difference in I/O and thus report only one of the three numbers.
2. **release copying via the rawhide script & cp -r**: see Appendix G.4.
3. **EdmBenchmarking**: Used to measure scaling behaviour on Sun SMP. We've worked hard to circumvent the problem of caching on the Sun SMP to measure real I/O to disk rather than I/O to memory. We believe that our efforts were successful.

D.3 Complex benchmarks

1. **Full CDF software release building**: This is what CDF code management does. We have timed release building, copying, and release packaging for remote sites.
2. **Limited test release building**: This is what a user would do. As an example of such an activity, we time the building of the benchmarking suite.
3. **stntupleMaker**: Running an exe on $t\bar{t}$ MC that makes the standard ntuple.
4. **analysis of stntuple**:
5. **AC++dump**: test 1 is reading the output of simulation. test 2 is reading the ouput of production. The key difference is therefore that the input to test 1 is just RAW data banks. The input to test 2 contains high level reconstruction objects with all their fancy streamers, in particular the results of silicon tracking.

6. **ROOT benchmark**: ROOT internal benchmark "RootStress" (see ROOT user's guide, section 16) is run on 1000 events (creates a 100MB file). This program spends a small fraction of its time testing various ROOT operations (graphics, cuts, histogram manipulation) while most of the time is spent on a simple example of user's analysis reading the large Tree generated and filling histograms (step 9). No CPU intensive analysis (e.g. combinatorial loops) is done. It is a moderately I/O intensive program as it read/writes a total of about 600MB in 100 secs on the fastest platforms. The stress program computes an overall performance index called ROOTMARKS, that is (real_time + cpu_time) relative to a reference platform. We found real_time $\simeq$ cpu_time, so ROOTMARKS scales with the amount of time it takes to do the job. We verified that ROOTMARKS ratios and elapsed time ratios across sgi2/sun2/lnx1 are the same to $1-2\%$, also these ratios are constant within 10% when changing the file size (number of events).

   We found that ROOTMARKS change up to a few percent when re-running the test on the same machine.

E Maxopt vs Noopt

E.1 CT_Reconstruction

CT_Reconstruction is the drift chamber reconstruction software. It should be pointed out that running this under maxopt does not result in the same output as under noopt. We have not investigated this further. All numbers are time (seconds) to reconstruct 100 events.

Optimization	Linux	Linux	IRIX	Sun	Sun
	fcwork2	cdfpca	fcdfsgi2	fcdfsun2	fcdfsun1
GHz	1.4	0.7	0.3	0.75	0.3
default	19.8	30.4	64.9	45.8	120.5
minopt	17.4	21.9	36.0	X	X
maxopt	8.0	12.5	13.8(A1)	X	X

E.2 ROOTMARKS

```
platform             sun2      lnx1     pca     sgi2
----------------------------------------------------
noopt                175.5     174.2    146.6    89.0
maxopt               220.0     229.0    219.8   156.3
----------------------------------------------------
ratio max/noopt        1.3       1.3      1.5     1.8
```

The root executable size is 168kB on sun and linux, 200 on sgi. For the unoptimised version in 3.18.0 it is 170 on linux, 205 on sun, and 223 on sgi. (sic, 200kB). It looks like optimisation does very little to the size.

It should be noted that the last test in the stress suite (CINT test) fails on all 4 machines with identical error message: "nlines in stress_lhcb.ps file = 1583". We did not investigate these crashes any further.

F CPU Scaling behaviour on SMP

In addition to the CPU benchmarks already discussed, it is interesting to test the scaling behaviour of CPU intensive tasks. That is, we would like to examine how the peformance of individual tasks depends on the number of such tasks running on the same machine at the same time. We have chosen for this purpose the Fermilab benchmarking program `tiny` [4].

Figure 1 shows the results of these scaling tests on cdfpca and fcdfsun2. Although cdfpca is an 8-way machine, the results shown here indicate that the benchmarking tasks only had between 4 and 5 processors available. fcdfsun2 had all its processors available, and the behaviour is exactly as would be expected from perfect scaling. That is, 48 simultaneous jobs each take twice as long to complete as 24 simultaneous jobs.

We have observed no departure from perfect scaling behaviour for `tiny` running on unloaded SMP machines. These CPU scaling tests are expected to be much more important when alternative architectures are considered in the future, in particular network cluster based architectures. In this case, network bandwidths and latencies can play an important role in the scaling behaviour of certain CPU intensive tasks.

G Additional Results on I/O tests

This section discusses a variety of I/O details that go beyond the summary numbers presented above. Issues involve I/O of actually existing CDF objects, I/O limits of the event data model (Edm) itself, and code management issues.

G.1 AC++Dump results for various input banks

```
Hi,

 I thought I'd pass along some strange results I've been seeing, while
```

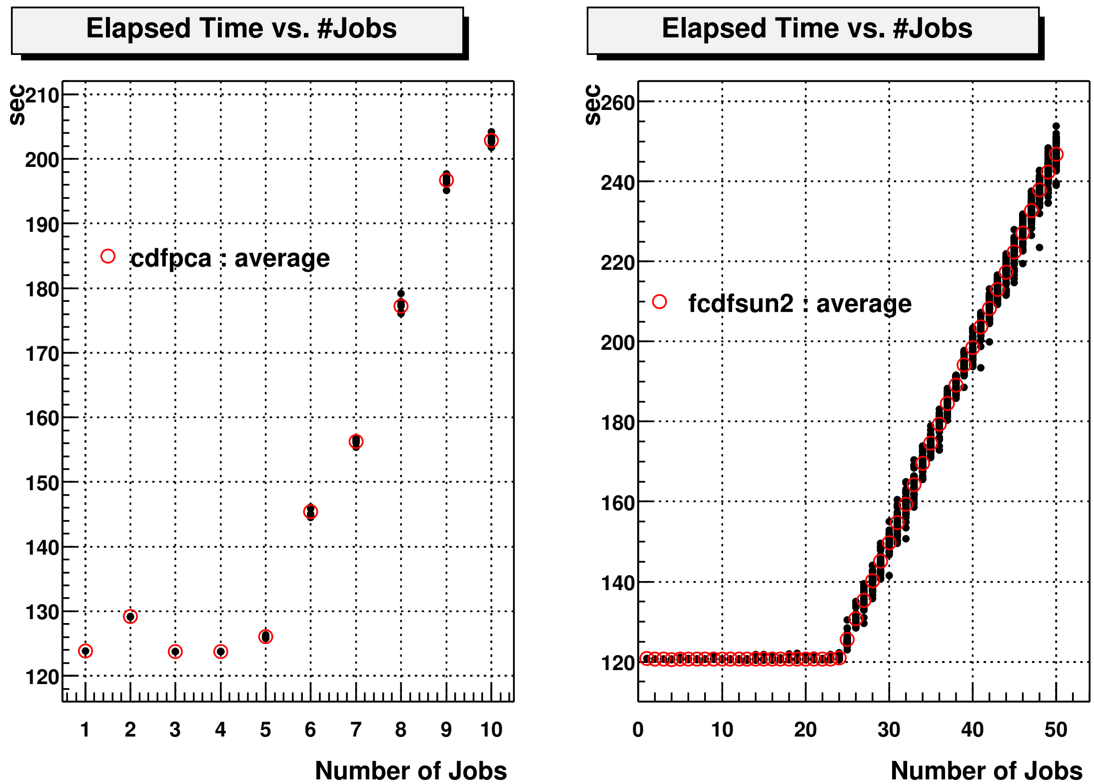

Figure 1: `tiny` scaling tests on cdfpca and fcdfsun2. In each case the total elapsed time is plotted versus the number of simultaneously running `tiny` benchmarking jobs. The averages are denoted by open circles. See text for more details.

```
trying to decide just what to AC++Dump.  It seems that the I/O rates with
AC++Dump vary dramatically with the contents of the input file.  Here are
a few examples (numbers are MB/sec):
```

	fcdfsgi2		nchi03 (866 MHz P3)	
	dd	AC++Dump	dd	AC++Dump
pyttbar_sim.rt	14	9	29	14
jr01d356.0001phys	15	2	26	3.5
jr01e476.0001phys	14	2	26	3
jr01e476.rewrite	15	3.5	25	8
jr01e476.notrack	16	8	26	12
ir01e4cc.0001test	--	--	28	0.4

```
*.notrack means I dropped CdfTrackHits CdfTrack CdfTrackColl; *.rewrite
just means I copied the input to the output with AC++Dump, to distinguish
between just copying the file and copying the file without the tracks.
The rates quoted are for reading the input file (either the original file
```

or the rewritten file).

The *.0001test file is a run I took with only the COT crates booked, and a crossing (zero-bias) trigger, so there are about 125K events, low occupancy, almost entirely COTD, in the 1GB file.

-Bill

Speed with which AC++Dump can read a certain file depends on the files content. In particular, raw data banks do not get unpacked into complex objects. The speed with which they can be read is thus similar to Edm_ReadWriteSeqRoot_byTBuffer . Objects like CdfTrack on the other hand require significant unpacking. We did not make any attempt of understanding this in any more detail.

G.2 EDM Results from Rob Kennedy

Hello,

Here FINALLY is a full table of EDM read(production output) benchmarks. Thanks to Art for his assistance in providing development builds on Sun, as well as maxopt with kai v4_0f on fcdfsgi2. In the future, this will be done with integration or frozen releases. It was done with development this time due to the ROOT v3.01 upgrade and the naive assumption that maxopt development would build on all OSs.

Units are MB of data read per CPU-seconds (user+sys) consumed

SRT_QUAL+root	fcdfsun1	fcdfsgi2	cdfpca	ncdf75
default+debug	2.5	3.2	6.6	7.8
default+optim	2.7	3.3	6.9	8.2
minopt +optim	3.8	3.8	9.2	10.4
maxopt +optim	5.7	5.8(1)	11.6	13.9(2)
CPU	UltraSparc2	R12000	PIII(Cas)	PIII(Cop)
CPU clock [MHz]	300	300	700	850

(1) Required an upgrade to kai v4_0f in order to avoid a crash in optimized code on IRIX only.

(2) Was 14.4+-0.2 using root v2_26_01a in development in late July/early August. So far, the difference does not appear to have a specific cause in CDF

```
code, as profiles from both periods appear quite similar. Early tests with ROOT
v3.01 did not show a noticeable change either.

        As stated in the first installment of results, this was measured using a
program from EdmModules/examples as follows:

time Edm_ReadWriteSeqRoot_ByEvent -g b4 ab01bb4d.0146phys

The data file in use contains production output, and can be found (without
using the usual tools) at fcdfsgi2:/cdf/data02/s13/edm. This process is
single-threaded, so at most one CPU is used to read data at a time. Data is in
SeqRoot format, so the entire event resides in a single branch; all objects in
the event are read from that one branch.

Thanks,
Rob K.
```

G.3 Closer look at I/O in AC++

In this section we take a closer look at what a cdfSim event is made up of, and how fast the major pieces can be read in. We do all of this with our standard $t\bar{t}$ Monte Carlo, generated with cdfSim 3.18.0 and run through ProductionExe 4.0.0. The reason for switching to this version of ProductionExe is that the streamer for CdfTrack has been improved significantly in 4.0.0. This resulted in a 33% decrease in size for CdfTrack, and factor 1.6 faster I/O.

G.3.1 CPU time in FileInput module

We first ran over 4950 $t\bar{t}$ events to establish the difference in time between release 3.18.0 and 4.0.0:

```
                          3.18.0              4.0.0
FileInput
in exe
TrackTest             0.0596 seconds/event   0.0363 seconds/event
on ttbar cdfSim
```

We then used only the first 92 events out of the release 4.0.0 file, and ran once on them as is, once after dropping everything but the objects LRIH_StorableBank, CdfTrack, CdfTrackIdManager, and CdfTrackColl, and once dropping only the above objects:

	standard output	only CdfTrack	excl. CdfTrack
average Event size	400kB	54kB	340kB
Time per event in FileInput	0.042 seconds	0.012 seconds	0.028 seconds
average I/O bandwidth	9.5 MB/sec	4.5 MB/sec	12 MB/sec

From this we conclude that even after the recent changes in the CdfTrack streamer CdfTrack's are read into AC++ a factor 2.7 less efficiently than the rest of the event.

We note that the bare disk I/O on cdfpca at the time these tests were made was measured to be 28 MB/sec for block sizes from 1MB to 200Bytes, i.e. we are by no means hardware I/O limited in these tests.

G.3.2 Dominant parts of an event:

Object:	per cent of event
CdfTrackHits	18.7
OBSP_StorableBank	12.4
CdfTrack	11.5
HEPG_StorableBank	9.9
MSVX_StorableBank	7.9
COTQ	7.1
SIXD_StorableBank	5.6
OBSV_StorableBank	3.3
ISLD_StorableBank	3.1
MISL_StorableBank	2.5
CesClusterColl	2.2
CalData	1.5
CdfJetColl	1.4
CESD_StorableBank	1.3
CprClusterColl	1.3
PESD_StorableBank	1.2
SiHitSet	1.2
PesClusterColl	1.1

All of the above add up to: 93.3 % of total. This tally was made for the first event in our standard $t\bar{t}$ sample using Evt_ObjectLister. It is not an average over the whole sample.

In light of CDF's plans of moving towards multi-branch I/O it is very encouraging that there is no single StorableObject type that is dominating the total event size.

G.4 Release creation via RAWHIDE & cp

```
Copying a release from local disk to the same disk,

    from ~cdfsoft/dist/releases/3.18.0
    to   ~cdfsoft/dist/releases/3.18.0def

  Distribution/create_rawhide
      is used to copy the base release, source code only.
    cp -r
      is used to copy the lib, bin and tmp areas.
         bin has a few large files
         tmp has many small files, about 53,000
         lib is somewhere in between.

  Times are in seconds, elapsed time

  SYSTEM        RAW   LIB    BIN   TMP

  cdfpca         91    61    494    471
  fcdfsun2      261   137   1000    382
  fcdfsgi2      461   146   1479   1801
```

G.5 Release removal

```
Commands used are

 /usr/bin/time -p $PROJECT_DIR/Distribution/create_rawhide \

   /dev/null xxx 3.18.0def |& tee maint/log/raw/xxx.log
 and

 /usr/bin/time -p $PROJECT_DIR/Distribution/remove_development \
    xxx | & tee maint/log/rem/xxx.log
```

```
create rawhide does :
   cp -r of source areas and libraries
   find | cpio of selected files in tmp:
      GNUmakefile
      ISOcxx/ISOcxx.h
      *.icc
      *.hh
   copy and strip of binaries (two steps on SunOS, where strip has no -o )
   declares and configures the new release in UPS

remove_development basicly does rm -r

    System     Create                Remove  (elapsed/user/system/percent)
              (seconds)             (seconds)
    CDFPCA      316    7   52  19%    45  0  2   4 %
    FCDFSGI2   1079    9  278  27%    94  1 56  60 %
    FCDSUN2    1928   24  388  21%    87  0  6   7 %

Notes:

SunOS does not have on option to strip binaries with redirected output.
      Therefore the binaries must be copied then stripped in place.

Note the large system CPU overhead removing the release on fcdfsgi2.
The other systems are I/O limited
```

G.6 I/O scaling behaviour using Edm tools

We have made scaling tests of two EDM (Event Data Model) benchmarking programs, Edm_ReadWriteSeqRoot_ByTBuffer ("Edm_Buffer") and Edm_ReadWriteSeqRoot_ByEvent ("Edm_Event"). Edm_Buffer performs block I/O without reconstructing objects in memory, while Edm_Event does event-oriented I/O in a similar manner to framework I/O modules [5]. We examine the scaling performance of these utilities on fcdfsun2, reading a sequential ROOT file generated by cdfSim that is approximatley 1 GB in size.

The Edm_Buffer results are shown in figure 2. The fact that even a single job runs with less than 50% CPU indicates that such a task is largely I/O bound. This is reflected in the aggregate read rate, which is already close to the maximum of approximately 30 MB/sec for a single task. This value is in good agreement with the dd results discussed earlier. Note that the average read rate per CPU second is much higher at almost 60 MB/CPU-sec, indicating that Edm_Buffer would be capable of reading a file at 60 MB/sec on fcdfsun2 before becoming CPU starved, in the absence of I/O system constraints.

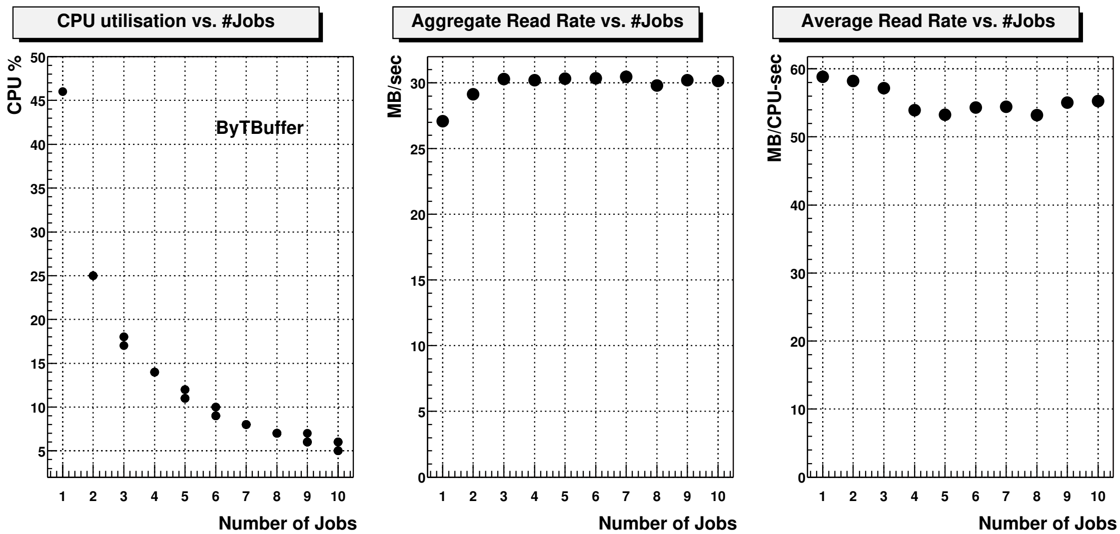

Figure 2: Scaling tests using Edm_ReadWriteSeqRoot_ByTBuffer on fcdfsun2. The plots show the CPU utilisation, aggregate read rate measured in MB/sec, and average read rate measured in MB/CPU-sec. See text for further details.

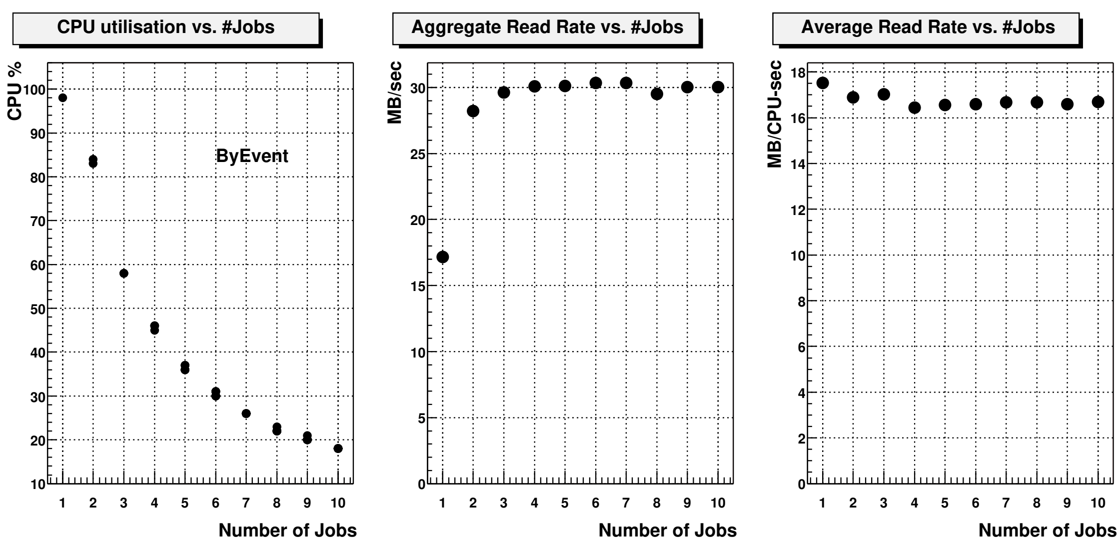

Figure 3: Scaling tests using Edm_ReadWriteSeqRoot_ByEvent on fcdfsun2. The plots show the CPU utilisation, aggregate read rate measured in MB/sec, and average read rate measured in MB/CPU-sec. See text for further details.

The Edm_Event results are shown in figure 3. It is clear that a single job is CPU bound, with close to 100% CPU utilisation and a read rate little more than half the available 30 MB/sec. Not until there are 3 jobs running in parallel does the aggregate read rate approach the 30 MB/sec observed in other tests. The average rate is almost constant at between 16 and 18 MB/CPU-sec, again indicating purely CPU constrained tasks.

H Run1 vs Run2 user analysis

B. Ashmanskas dug out three logfiles from 1996 that did a user skim on two different run1 data sets.

Dataset	# of evts	AC timing	''MIPS'' sec/event
ECLB-5B BPAD	5612569	47050 sec	1.25
ECLB-5B BPAD	5612569	32930 sec	0.88
BHMB-5B	1916089	14451 sec	1.13

Run 1 analysis skim thus needed ~ 1.1 MHz seconds per event on "old cdfsga".

The first two jobs were looking for electron + $D^0 \rightarrow K^-\pi^+$ and were smart about only recreating SVX fits for tracks whose CTC parameters looked promising, i.e. not every track was refit as part of the analysis skim.

In comparison, we have benchmarked a large variety of run2 applications on a variety of different input files with the following results:

	fcdfsun2	fcdflnx1	cdfpca	fcdfsgi2
cdfSim:				
ttbar	7425 (1.2)	6202 (1.0)	5418 (0.9)	---
ProductionExe:				
ttbar cdfSim	3369 (1.8)	1900 (1.0)	1905 (1.0)	2127 (1.1)
PuffModule (puffing up the event):				
ttbar cdfSim	275 (1.3)	214 (1.0)	214 (1.0)	236 (1.1)
FileInput (reading in an event):				
ttbar cdfSim	113 (1.5)	77 (1.0)	81 (1.1)	101 (1.3)
Stnmaker (making an Stntuple):				
on data	--	--	61	--
ttbar cdfSim	35 (1.3)	27 (1.0)	28 (1.0)	41 (1.5)

```
TrackAnal (D+ skim using CTVMFT):
cdfsim ttbar        25 (1.5)     --          17   (1.0)     --
run 126903 jr       14 (1.5)                  9.5 (1.0)     --
run 126903 jr
no CTVMFT fits      --           --           3.4           --
quicksim bbbar       1.6         1.0          1.0           1.2

Stntuple            0.79        0.75         0.78          0.82
analysis
```

The closest to Bill's example is probably TrackAnal on a recent data file. On fcdfsun2 this took 14 MHz seconds/event, a factor 13 more than Bill's example from run1.

I Results on building software releases

This section summarizes our findings with respect to release building. This includes code management issues as well as release building by users to develop their physics analysis modules. In a nutshell, sun and the linux system tested are equivalent while performance on fcdfsgi2 is four times worse.

The only exception to this general rule is the release packaging part of code management. Here sun and sgi are roughly equivalent and a factor of 2-3 worse than fcdflnx1.

In addition to this, there is serious concern for sun2 about the amount of memory needed to build even the most basic exes, as well as the size of the resulting exe.

I.1 Memory Usage and Exe size

We ran 'pshow' shapshots of 'ld' building the standard benchmark test release (noopt), for all binaries on fcdfsun2 with the following results:

```
PROGRAM          Real      SIZE
 Name           Memory    of Exe
                MBytes    MBytes
                 sun2   sun2 sgi2 pca
ProductionExe    1447   465  226  139
cdfSim           1418   433  228  150
TrackTest        1031   319  158   95
stnmaker.exe      973   303  154   92
AC++Dump          935   296  150   89
```

These numbers indicate:

- sun needs at least 1TB disk space for release maintenance. This is 3 times what's needed on Linux.
- There is concern that the memory consumption for building even the most modest AC++ executable is close to 1GB, and exe sizes are 300-500MB. The sun has 1GB of RAM per processor. This indicates that performance is likely to drop massively when there are significantly more jobs /active users on the machine than processors.

I.2 Building a full CDF software release

```
Build CPU times, release 3.18.0
 default optimization
 times given in minutes

 cdfpca     7 way

  PHASE    USER    SYS   TOTAL

 refresh    1.5     6.9     8.4
 codegen   11.5    24.     25.5
 include    1.5     5.8     7.3
 lib      325.     74.    399.
 shared    10.2     2.7    12.9
 bin       39.     12.     51.    NET   504.1

 fcdfsgi2   12 way

 refresh    2.1    10.3    12.4
 codegen   20.8   234.    244.8
 include    2.6     9.6    12.2
 lib      885.    611.   1496.
 shared    26.6    10.8    37.4
 bin      193.     40.3   233.3   NET  2036.1

 fcdfsun2    8 way

 refresh    1.7     2.2     3.9
 codegen   14.     16.5    30.5
 include    1.7     1.9     3.5
 lib      313.     87.8   400.8
```

```
shared    11.4     2.3    13.7
bin       78.     11.4    89.4   NET   541.8
```

Notes and Comments:

- large system overheads on IRIX for codegen phase is caused by character-mode I/O in rootcint.
- System overheads in lib increase on SunOS with more CPU's

```
                  USER    SYS    TOTAL

        lib       313.     87.8   400.8 ( 8  way build)
        lib       327.    265.    592.  ( 24 way build)
        lib       350.   1601.   1951.  ( 48, not fair, we have only 24 CPUs)
```

- SunOS need 1.5GB real memory to link ProductionExe.
- SunOS swaps heavily with only 12 of 24 GB realm memory in use.

Release packaging

```
Elapsed times making the distribution tarfiles.
These contain source code and libraries for release 3.18.0def.

Release 3.18.0def contains just the default libraries,
without minopt or maxopt.


System    Time    User    System  Percent  Minutes (for convenience)
         (seconds)
fcdflnx1  318.17  296.37  26.64   102 %     5
cdfpca    352.88  267.89  30.80    85 %     6
fcdfsun2  694.63  395.81  69.02    67 %    12
fcdfsgi2  859.458 626.947 94.466   84 %    14

Notes :

   Linux systems are cpu-bound, running gzip

    SunOS/IRIX systems are I/O bound, and much slower than Linux.
```

I.3 Building a user test release

Node Description		fcdflnx1	cdfpca	fcdfsun2	fcdfsgi2
gmake depend:	real	375.24 1.35	327.36 1.17	287.87 1.00	1356.673 4.86
	user	289.17 2.52	284.73 1.39	205.27 1.00	854.373 4.16
	sys	58.64	35.52	74.79	410.798
gmake codegen:	real	38.30 1.00	35.16 1.09	68.80 1.96	121.947 3.47
	user	25.88 1.00	25.72 1.00	38.96 1.51	51.255 1.99
	sys	10.36	10.01	27.10	61.002
gmake lib:	real	1409.37 1.24	1275.04 1.12	1135.94 1.00	4219.686 3.71
	user	988.52 1.21	988.25 1.21	817.42 1.00	2669.267 3.27
	sys	123.70	85.35	193.66	730.257
gmake (t)bin	real	529.14 1.00	528.82 1.00	1386.97 2.62	3493.444 6.61
	user	319.18 1.00	366.41 1.15	1110.32 3.48	2693.651 8.44
	sys	65.06	48.77	143.20	263.700
total	real	2352.05 1.09	2166.38 1.00	2879.59 1.34	9191.750 4.24
	user	1622.75 1.00	1665.21 1.03	2171.97 1.34	6268.540 3.86
	sys	257.76	197.65	438.75	4465.757
num 1.00's		5=*****	3=***	4=****	0

The last row notes the number of times a given system exceeds the performance of all others in one of these tests.

J Network Tests

network copy tests

Copying the bin and tmp directories from release 3.18.0def .

Copies were from cdfpcb, a 600 MHz PIII,
to the ~cdfsoft/trel/copy empty subdirectory on each system.
The local Fast Ethernet network was used.

bin contains 3.6 GBytes of binaries in 162 files
tmp contains 1.6 GBytes , over 50,000 files under 223 package directories

```
   SYSTEM   BIN/SCP BIN/RCP TMP/SCP TMP/RCP (seconds realtime)

   cdfpcb     670             1676          (initial copy from cdfpca)
   fcdflnx1   511     508     1893    1034
   fcdfsun2  1072     359     3287    2898
   fcdfsgi2   785     350     2099    1260

Commands used were

/usr/bin/time -p \
   scp -c blowfish -q -r cdfsoft@cdfpcb:dist/releases/3.18.0def/bin bin
/usr/bin/time -p \
   rcp -r cdfsoft@cdfpcb:dist/releases/3.18.0def/bin bin

/usr/bin/time -p \
   scp -c blowfish -q -r cdfsoft@cdfpcb:dist/releases/3.18.0def/tmp tmp
/usr/bin/time -p \
   rcp -r cdfsoft@cdfpcb:dist/releases/3.18.0def/tmp tmp

Detailed logs are in /cdf/home/kreymer/bench/copy.log
```

References

[1] http://www-cdf.fnal.gov/upgrades/computing/projects/central/

[2] CDF note 4100, March 1997. This note specifies the CAF design requirements. This note assumes: 1 MIPS = 1 Tiny = 1 SPECint92 = 1/40 SPECint95

[3] We verified Specmark conversion factor by comparing three systems (DEC Alpha 8200 5/350, Intel Alder 200MHz, Dell Dim. XPS 133MHz). We find a ratio SPECint92/SPECint95 = 43,46,45 for the three systems. Similarily, we find SPECint2000/SPECint95 = 9.1,13.2,9.4 for Dell Prec. Workstation P3 700MHz and SGI 2200 2x400MHz R12K, Sun Ultra 10 333MHz (UltraIIi CPU).
http://performance.netlib.org/performance/html/new.spec.cint92.col0.html
http://www.spec.org/osg/cpu95/results/cpu95.html
http://www.spec.org/osg/cpu2000/results/cpu2000.html

[4] ftp://linux-rep.fnal.gov/pub/benchmarks

[5] Robert Kennedy, private communication.