The National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign opened in 1986 as one of the five original National Science Foundation (NSF) Supercomputer Centers. During the decade covered by that program, the center earned an international reputation for innovative applications in high-performance computing, visualization, and desktop software. NCSA greatly broadened the user base of remote supercomputing and the Internet by developing the cross-platform software tool NCSA Telnet in 1987. In 1992, the center developed NCSA Mosaic, the first easily-available graphical Web browser that helped launch the explosive growth of the Web in the 1990s.

Since 1997, NCSA has been the leading-edge site for the National Computational Science Alliance (Alliance), one of two partnerships of the NSF's Partnerships for Advanced Computational Infrastructure (PACI) program. As the leader of the Alliance, NCSA is working to connect computing systems, research teams, scientific applications and toolkits, and large databases through a high-performance information infrastructure called the Grid. NCSA's wide range of education and outreach programs illustrate its commitment to helping communities, education, and business employ cutting-edge technologies and computing tools. The center also collaborates with Fortune 500 corporations through its Private Sector Program. This program gives businesses the opportunity to explore new technologies and tools that can help them maintain a competitive edge in a global economy.

As it enters the new century, NCSA continues to be at the forefront of supercomputing technology and to work to apply the power of supercomputers, grid technologies, and advanced visualization environments to the needs of society, including scientists, our schools, our communities, and populations that have been underrepresented in the technology revolution.

About Us
NCSA
Alliance
NSF PACI
Community Partnerships
Private Sector Program
Technology Strategy
Roadmaps
Software Deployment
NCSA Projects
Science
Chemical Engineering
Cosmology
Nanomaterials
Environmental Hydrology
Scientific Instrumentation
Success Stories
User Information
Getting Started
Hardware & Software
Consulting
Training
Allocations & Accounts
Alliance Resources
News
Access Online
data link Newsletter
Press Room
Archive

Archived Content

FAQ List

Frequently Asked Performance Related Questions

Miscellaneous System Issues (Memory, I/O, etc.)

QUESTION: I was timing one of my codes, and noticed that a significant portion of the time was in "sys" (which I presume is system time). Can you tell me what types of operations take up system time, and if it is reasonable to add system time to the CPU time in order to comapre the speeds of programs? Or should we use only the CPU time in order to compare programs? (I guess that the "user" time returned by /usr/bin/time refers to the CPU time.)

ANSWER: System time is any CPU time the system spends doing tasks on behalf of your process. The most common source of system time is the cost of doing input/output for your process. Whether you should include system time in your comparisons depends a bit on why you are doing the comparison. System time is included in the charges for your jobs, so if you are comparing cost, it wise to include it. On the other hand, if you are trying to compare the CPU time consumed by competing computational algorithms, it is probably misleading to include the system time used to set up the problem and print its results. (The system times should be very similar and will tend to mask the speedup in the computation part of the program.) On the other hand, if the system times are significantly different for the two programs, then the algorithms may be having an effect on system time and it may be necessary to include system time after all.

QUESTION: Is there an option that I can give to get the total amount of

memory that my program used ?

ANSWER: If the major portion of the arrays in your code is allocated statically, i.e. at the compile time, then executing the size command on the executable will give you the size of memory, roughly, needed by your code. However, if you allocate your arrays dynamically, at run-time, then the size command will not show useful info. You can compile your code with the -static option and use the size command with that executable to get an idea on the memory size, then recompile without -static to generate the executable you will actually run.

QUESTION: Currently I am porting some codes from the C90 to the SGI-PCA. For an exact same batch job, the comparison of major CPU times by C90 and SGI-PCA are listed below (in seconds):

```
                                            C90       SGI-PCA
CPUTime for Filling Coef. Matrices = 343.49       1062.37
CPUTime for Solutions in BiCGSTAB  =  38.83       1539.52
CPUTime for Calc. BDR_p            =  52.82         92.90
CPUTime for Solutions in BiCGSTAB  =  38.23       1480.26
CPUTime for Calc. BDR_s            =  52.84         92.42
                         Total CPU = 526.90       4269.39
```

As reflected above, the total CPU on SGI-PCA is 8 times more than on C90. I used "-O3 -64 -mips4 -r8 -pfa" options of "f77" in SGI-PCA , while on C90, I just simply used "cf77" without any options. Is there an explanation for such big CPU differences between Cray C90 and SGI-PCA?

ANSWER: The peak performance on the Cray Y-MP and SGI PCA are similar, but NOT the C90. The peak speed on a single processor of the C90 is 1 Gflop compared to 390 Mflops on a R10000. Also, you indicated that you are running on a single processor. In this case, you should not use the -pfa option - this does automatic parallelization, and when you run on only one processor, you incur the parallel overhead in any case. In general, for getting accurate timing numbers, you should run in the dedicated queues.

QUESTION: I have a file generated to record the iteration process in my job, but the information is only written into the file AFTER the job has completed. Please let me know if there is an easy way to do I/O in the middle of a job.

ANSWER: I/O is buffered, so you need to flush the buffer to be able to see the output. Add the line

```
          call flush(iunit)
```

after the write.