Nbench 2.0 Aug. 1995

Michael Cornelison

CompuServe	76450,2336
Internet	mico@ix.netcom.com
phone	(610) 712-6185

Overview

There are two types of PC benchmark utilities commonly available. One type measures the performance of an entire PC system, by measuring the elapsed time to complete a set of application tasks that represent a work profile. A second type measures more basic components of performance, namely processor, graphics IO, and disk IO speeds.

Application-based benchmarks are valid tools to measure the performance of a whole computer system, for the specific application set included in the benchmark. However, they may not be valid predictors of performance with other applications. Also, such benchmarks provide little insight into which parts of the system perform well or poorly. These benchmarks are primarily useful if you are purchasing a complete system, configured the same way as in the benchmark, for the purpose of running the same applications used in the benchmark.

The second type of benchmark, which makes separate measurements of processor, graphics, and disk performance, also has its limitations. The reported processor performance is likely some unknown mix of CPU, chipset, cache, and main memory performance. Thus, you may not know if a given benchmark is a good indicator of the performance you would see with your intended usage. There have been some notable problems in this area, particularly benchmarks that fit entirely within L1 (or L2) cache, and therefore provide no indication of L2 cache (or main memory) performance. Some benchmarks measure disk performance mixed-up with OS-based file caching, and may even be sensitive to OS parameters adjustable by the user. If small files are used, rotational latency may dominate the results, whereas if large files are used, disk and adapter throughput may dominate.

Nbench tries to address these problems. Each of the following performance components can be measured separately from other factors:

- CPU integer and floating speeds
- L1 and L2 cache speeds
- main memory speed
- disk read and write speeds

Nbench can report each of the above components of performance for 1-20 separate execution threads. Thus, it is possible to study the performance of multi-CPU hardware systems and multitasking operating systems. If a system has N CPUs, it should ideally be able to run N CPU test (or memory test) execution threads, in about the same amount of time as for one thread. Any slowdown in CPU test results are likely due to operating system task switching overhead, and any slowdown in memory test results are likely due to cache or memory bandwidth limitations, as well as operating system task switching overhead. If a system has one CPU, the throughput for each of N threads should ideally be 1/N of the throughput for one thread. Any slowdown is likely due to operating system task switching overhead.

When running memory tests, be careful not to place such demands on total memory that the system starts to page itself to death. This is not what you want to measure. To be valid, memory testing should be largely absent of disk activity. Pay attention to the thread count and your maximum memory region size: their product should not exceed the physical memory available for user applications. If an initial memory test causes paging activity, a repeat of the same test may show little or no paging activity, and will give a more valid result.

Nbench runs on the Windows NT 3.5 and Windows 95 operating system, and should also run OK with later versions of these systems.

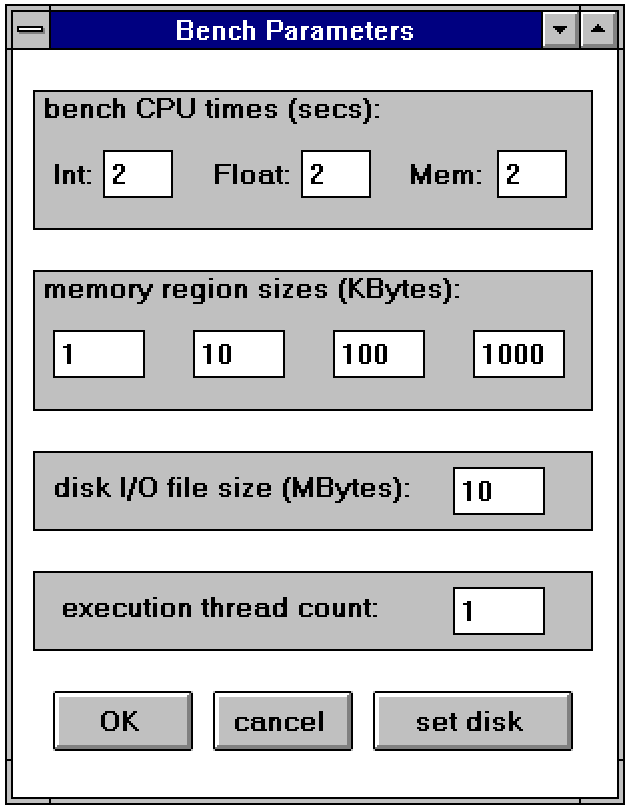
Main Menu

The following menu commands are available from the main window:

Quit	quit the application
Clear	erase report output from window
Setup	go to parameter **<u>setup</u>** dialog
Run	dropdown menu with following 3 entries:
Processor	perform CPU tests
Memory	perform memory tests
Disk I/O	perform disk I/O tests
Save	save report output to a file
Help	display help file

To run the benchmark tests, do the following: select **setup** and enter any desired changes in setup parameters, select **run** and then select one of the three benchmark tests (processor, memory, or disk). The test results are written into the main window within a few seconds. These reports will accumulate, until you select **clear** to erase them from the window. Use **save** to save the reports to a text file. Use **quit** to exit the application. If you select a menu command while a test is running, the current test will be interrupted and the new command executed. If you select a thread count greater than 1 in the **setup** dialog, the results are reported separately for each thread.

Setup Dialog



bench CPU times (secs):

Int: 2 Float: 2 Mem: 2

memory region sizes (KBytes):

1 10 100 1000

disk I/O file size (MBytes): 10

execution thread count: 1

Use this dialog to change benchmark parameters. Refer to the table below. Use **tab** or **shift+tab** to navigate forward or backwards through the parameters. Use the backspace key to erase, and the numeric keys to enter new values. Click **OK** when done, or **cancel** to abandon the updates and keep prior values. If the cursor jumps back to a prior field with a beep or bell, it means you have entered a bad value.

parameter	**range**	**description**
CPU times	1-99 secs.	minimum test time durations
memory region sizes	1-9999 KB	sizes of memory regions to be used for cache and memory tests
disk file size	1-999 MB	size of disk file for disk I/O test
thread count	1-20	no. of simultaneous execution threads

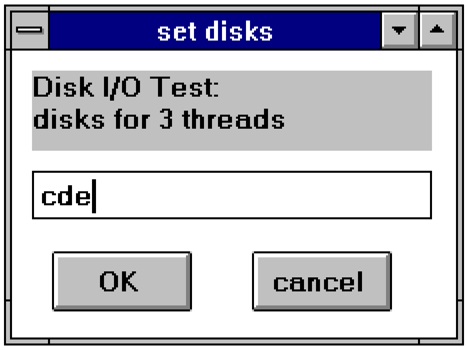
Use CPU time durations of a few seconds or more to insure consistent results. The actual test times may be extended for memory tests, if the required minimum byte counts have not been moved within the specified test times.

Use memory region sizes appropriate for the L1 and L2 cache sizes of your system. For example, if your system has a 4 KB L1 data cache, then a region size of 4 KB or less is effectively a test of L1 cache throughput. If you have a 256 KB L2 cache, then a region size of 256 KB or less is effectively a test of L2 cache throughput. The program loop is small enough to fit within the L1 cache, if your system has both L1 and L2 cache (true if your CPU is a 486 or later). To test main memory throughput, use a region size that is much greater than your L2 cache size (e.g. 2000 KB for an L2 cache of 256 KB).

Up to four memory region sizes can be specified, for running up to four memory tests in succession. If you want fewer tests, set one or more region sizes to zero.

Use a disk file size large enough to give consistent results. In order to make seek time and rotational latency become insignificant, this should be 2 MB or more.

If you select the **set disk** button, the following dialog box will appear:



Disk I/O Test:
disks for 3 threads

cde

Here you may input the disk drive letters to be used for disk I/O test files. Enter one letter for each execution thread. For example, if you have three threads, and you wish them to use drives c, d, and e respectively, then enter **cde** into the dialog text box.

Technical Notes

Here are examples of each type of report, with explanatory comments following.

Processor Performance Report

```
CPU Performance, MOPs/sec
 Integer Speed: 72.2
Floating Speed: 15.1
```

The integer performance test is a heapsort program which continuously re-sorts a set of 1000 random integers. Since both the program and the data fit within the L1 cache, it is a test of processor speed under the best conditions. The units are approximate millions of operations per second. An operation (from the viewpoint of C source code) is an assignment, comparison, arithmetic operation, array index calculation, or function call (with an additional operation counted per passed argument).

The floating performance test is a small loop of floating (double) calculations with the following ratios: add: 11 subtract: 2 multiply: 9 divide: 2
There is no array indexing in this test, so that floating arithmetic performance is the primary thing being measured. The code is impossible for the compiler to optimize by consolidating operations or moving them outside the loop.

Memory Performance Report

```
Memory Move Performance, MBytes/sec

memory  access    access method
region   width   random   serial
   1KB     1       45.7     57.9
   1KB     2       38.7     41.7
   1KB     4      172.7    222.7
   1KB     8      255.2    255.2
  10KB     1       20.1     29.0
  10KB     2       30.5     37.1
  10KB     4       79.0    112.8
  10KB     8      122.2    122.2
 100KB     1        9.6     16.1
 100KB     2       19.0     29.5
 100KB     4       38.3     54.6
 100KB     8       56.0     56.0
1000KB     1        4.6     10.4
1000KB     2        9.4     19.1
1000KB     4       18.6     32.1
1000KB     8       34.2     34.2
```

Each memory region size is tested for both random and serial moves, using aligned operands of 1, 2, 4, and 8 bytes. A little study of the above will reveal that the L1 cache throughput is about 255 MB/sec for 8 byte operands, the L2 cache is about 56 MB/sec, and the main memory is about 34 MB/sec . Sequential access is significantly faster than random for 1, 2, and 4 byte access, but 8 byte access shows no difference. This indicates that all memory accesses are 8 bytes, and that the upper 4 bytes are cached when the lower 4 bytes are accessed. Keep in mind that these numbers are for memory moves (read + write to new location), so 34 MB/sec actually represents a total memory bandwidth of about 68 MB/sec.

Disk Performance Report

```
Disk Performance, MBytes/sec
File size: 10.0 Mbytes
thread:          0          1          2
write:           1.91       1.66       1.79
read:            1.92       1.95       1.80
```

What is measured is the I/O throughput for a file written and read sequentially, from the beginning to the end. Note that disk seek and rotation times are not being measured, and normally should not influence the results. If your disk is badly fragmented, the measured speed will be reduced, since the file will be created in many discontiguous pieces. If this is happening, you should be able to hear the disk drive clattering as the file is being **read**. Do not consider the results to be valid unless the clattering during read is a low fraction of the total time. Some clattering will occur as the file is being written (created), since the OS must allocate the necessary storage space. The file is created with the attribute `FILE_NO_BUFFERING`, so that the OS will not cache the file, and all I/O will go directly between the application buffer and the disk. Thus, there should be no misleading speedup from OS caching, even if the file is quite small.

If more than one thread is used, be sure that each thread uses a different physical disk drive. Otherwise, the constant seeking between multiple files being accessed on one drive will cause lower performance to be reported. This may be OK if your intent is to measure drive performance under such conditions.

Known Problems

Under NT, if a test is repeatedly interrupted before completion (by selecting a new menu command), there is a small memory leak that causes the Nbench process page file allocation to grow. This is apparently a problem with the use of TerminateThread(). There is no usage of **new** or **malloc** in the thread functions that could cause this. You should not interrupt and restart a test hundreds of times. Instead, exit and restart Nbench.exe.

When many threads are used, the OS may not allocate resources evenly. I have seen instances where one thread would be completed while another one would be barely started. This is erratic and follows no obvious pattern. Sometimes there is no problem, and all threads run very evenly and finish at nearly the same time.

I was able to crash Windows 95 (pre-release 2) by running more than 10 threads in a disk test. NT had no problems up to the program limit of 20 threads.