Application Performance Optimization

By Börje Lindh - Sun Microsystems AB, Sweden

Sun BluePrints™ OnLine - March 2002

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 USA
650 960-1300

Part No.: 816-4529-10
Revision 1.3, 02/25/02
Edition: March 2002

Application Performance Optimization

Many developers and users leave a lot of performance on the table due to lack of information. Several good documents on this subject are available, but they are all very extensive and might not be easy to find.

This document is a short introduction to performance on Sun systems. It tries to provide an overview of the subject as well as to supply links and pointers to tools and documentation. The intent is to provide information to help you improve the application performance of your system.

What Limits Application Performance?

This is not a trivial question. The usual system tools allow you to understand how the system as a whole is used, but these tools do not tell you if the application is using the CPU in an optimal way. To obtain optimal performance, you must look at both the system level and the application level.

For the system level, you have the usual tools such as `vmstat`, `iostat`, and so forth. Also, there is a set of performance tools developed by Adrian Cockcroft and Rich Pettit called SE Toolkit. We especially recommend the tools called `zoom.se` for interactive testing and `virtual_adrian` for non-interactive logging of system behavior. To understand what process is causing the load, the command `prstat` is very helpful. You can find a lot of information on system performance as well as a pointer to the SE Toolkit at `http://www.sun.com/sun-on-net/performance`. The actual tool is available from `http://www.setoolkit.com` and from `http://www.sunfreeware.com`.

The Performance Analyzer, which is part of the Forte™ Developer software, can be very helpful in understanding how well the application is optimized.

The Optimal Situation

When the application is running into the limits in every dimension, including fully utilized CPU pipelines and cache bandwidth, you have achieved fully balanced performance. You are not likely to do better until you get a new computer that is faster in all aspects—CPU speed, memory bandwidth, memory latency, and so forth.

I/O Limitations

Virtual memory (disk) is much, much slower than physical memory, actually on the order of 100,000 times slower. On many systems, disk I/O limits application performance. Therefore, having enough memory for both the application and caching the set of commonly read files to eliminate unnecessary disk I/O is very important.

A single disk can handle only a certain number of random access I/O requests. For a 10,000 RPM disk this number is on the order of 120 to 130 I/O operations per second. If your application must do more than that, you need multiple disks even if your storage needs are less than the size of one disk.

To evenly spread the load between multiple disks, you can use disk striping. This is also referred to as RAID-0. There are a few common RAID levels. RAID-1 is mirroring and RAID-5 is striping with parity so that a disk can be lost without losing data. When both high performance and protection against data loss is required, it is usual to use RAID-0+1 (both striping and mirroring).

Note that if your I/O needs are large, you may require a lot more storage capacity than you need for data storage to meet your performance needs.

If the system uses synchronous writes, that is, an application is waiting for the data to reach stable storage before it continues, a write cache can significantly improve application performance. Most high-end disk subsystems today have this.

If you want to learn more about system performance tuning, we recommend reading *Sun Performance and Tuning—Java and the Internet*, 2nd Ed" by Adrian Cockcroft and Richard Pettit (ISBN 0-13-095249-4). An advanced network storage solution today might be faster than the traditional locally attached disk. In some cases, the standard UNIX® file system itself may be a bottleneck, and you must look at alternate file systems such as Sun™ QFS."

Network performance also limits the performance of an application. The SE Toolkit is very useful in finding these kinds of problems for typical TCP/IP network traffic. If the application is running in parallel on multiple nodes that must synchronize,

finding out that the network is actually the bottleneck requires other tools. The Forte Developer analyzer and the Prism™ debugger software (which is part of the Sun HPC ClusterTools™ software) are the right tools to use.

Not only network capacity but also network latency, that is, how long it takes to transfer a message between two computers, may become a limiting factor. The different forms of Ethernet networks all have fairly high latency, but there are also other networks like Myrinet (`http://www.myri.com`) which have better characteristics for parallel computing. Another alternative is to run the application on a large SMP machine where the different processes can communicate using shared memory instead of a network.

Memory Hierarchy

Today's systems have a complex memory hierarchy. Multiple levels of fast buffer memory (often referred to as cache) have been inserted between the processor and main memory. The purpose of cache is to store frequently used data. Programs that can take advantage of these caches run significantly faster. Accessing data in a CPU register takes 1 to 2ns. Accessing memory takes, depending on system, between 180ns and 550ns. Accessing virtual memory (on disk) takes in the order of 10ms (10,000,000ns).

The CPU chip has both level 1 instruction and level 1 data caches. Accessing these caches is usually two to three times slower than accessing the registers. There is also a Translation Lookaside Buffer (TLB) on the chip, which is cache for translations between virtual page addresses and physical page addresses. As the CPU does not have room for large enough caches on the chip, there is also an external level 2 cache made up by fast SRAM memory. The speed of this is usually 4 to 10ns; sizes today vary between 0.5 megabytes and 8 megabytes.

The three common ways to replace data in caches are:

- Direct mapped—location equals memory location modulo cache size
- Fully associative—location equals oldest cache line (LRU)
- *X*-way associative—choose a set first; direct mapped within set (the value of *X* is often 2 or 4)

A higher associativity reduces the risk for cache conflicts and may thus improve performance. In the UltraSPARC™ I and II processors , the L1 data cache is direct mapped, the L1 instruction cache is two-way, and the L2 cache is direct mapped. In the UltraSPARC III processor both the L1 caches are four-way and the L2 is direct mapped. In the UltraSPARC III Cu processor both the L1 caches are four-way and the L2 is two-way.

For some systems, like the Sun Enterprise™ *X*500 series of machines, memory interleave can greatly affect the overall system performance. A Perl script called `memconf` is available from `http://www.sunfreeware.com` that can be helpful in understanding memory interleaving in your system.

For the Solaris™ 7 operating environment a tool called `memstat`, written by Richard McDougall, allows you to see what is in the system memory. This tool is available from `http://www.sun.com/sun-on-net/performance`.

What Is Tuning All About?

The main idea of tuning is to:

- Keep data as close to the CPU as possible.
- Use all data that have been loaded into the cache (spatial locality).
- Reuse data that has been cached as long as possible (temporal locality).
- Obtain an optimal instruction stream and make maximum use of the hardware resources in the CPU.

All of the preceding tuning might be realized by the compiler, but it is limited by lack of information. The coding style can also make a difference and cause the compiler to generate suboptimal code. We recommend that you write clear code and leave the low-level details to the compiler. Where possible, memory access should be streamlined.

By the latter we mean that care should be taken to set up the data structures such that main memory is accessed linearly without jumps (the so-called unit stride memory access) and that data elements are reused in a single loop as often as possible. It is outside the scope of this paper to go into more detail.

Manual algorithm changes can be very rewarding in performance, and simple modifications can improve performance on a variety of systems. The performance analyzer can help identify opportunities for algorithm performance enhancements.

Interested readers can find much more information on this topic in *Techniques For Optimizing Applications—High Performance Computing* by Rajat Garg and Ilya Sharapov (ISBN 0-13-093476-3). Also, tuning optimization seminars called Sun Tune seminar are given by Ruud van der Pas. If you are interested in attending one of these seminars, contact your local Sun office.

Application Code Best Practices

Compilers today are very powerful, but they lack the knowledge about what the programmer is trying to do. To make the job of the compiler easy, thus allowing good optimization, keep the code as clear and simple as possible.

Some advice from one of our best tuning experts, Ruud van der Pas is:

- Split the code in compute intensive parts.
- Write efficient, but clear code; leave the details to the compiler.
- Avoid very bulky loops.
- Be careful how you set up your data structures.
- Minimize global data structures.
- Simplify branches where possible.
- Put the most likely branch first.
- Push the branch part out of the loop

Optimized math routines are available. You can access these routines by adding the `-xlibmopt` flag in FORTRAN, and the `-lmopt` flag in C. The Sun Forte Developer software includes a highly tuned performance library that has routines like BLAS level 1–3, LAPACK, FFTPACK and VFFTPACK. These routines can be linked in by adding `-xlic_lib=sunperf`. To get optimal performance, it is important to use the right version by setting `-xarch`. Also, note that these routines are compiled with `-dalign`, so this option is required.

Why Enable Compiler Optimization?

It is important to understand that compiler optimization is disabled by default. The default mode is to generate correct code as fast as possible to increase the productivity of the software developer, as the optimization work takes time. By turning on optimization, the execution time for an application can sometimes be reduced to 50 percent or less of the unoptimized execution time. This means that once you have a correct program, you need to recompile with optimization enabled and verify that your program is still correct.

Compiling for the right instruction set, so that all registers in the CPU are used, is important. (If an application is compiled for v7, only half of the UltraSPARC processor floating point registers are used.) With the UltraSPARC III Cu processor chip and onwards, it is possible to move data to the on-chip prefetch cache, so that the data is available on chip when needed.

Also, be aware that the compiler is constantly being improved. Moving to the latest compiler version can often improve application performance.

Putting Your Effort in the Right Place

In many cases, one routine or perhaps a few routines take up most of the execution time. You should put most of your effort into these routines. The Forte Developer software has a very powerful tool called Performance Analyzer that you can use to find these routines.

When tuning, the most important thing is to verify that the program generates the correct results. Be sure to check that the result remains correct when the optimization level is increased. Some of the optimizations techniques used by the compiler make assumptions about the code, which in some cases may be incorrect.

Compiler flags are interpreted left to right, which means that you might not always get the result you expect. The best way to check what the compiler is doing is to turn on verbose mode, which is done with `-v` in FORTRAN and `-#` in C.

If you want to check the flags without doing any real compilation to verify that you really get what you want, you can add `-dryrun` (FORTRAN) or `-###` (C).

If you are debugging code, you must compile with the `-g` flag. The Forte Developer software supports source browsing, which needs `-xsb` to be added.

The command `f95 -flags`, will print the most common flags accepted by the FORTRAN 95 compiler. The current version of the Sun Forte program supports C, C++, FORTRAN and Java™software. Besides the compilers, the current version also includes a complete development environment with context sensitive editors, source browser, debugger, and so forth.

Besides the usual man pages, you can also find useful information in:

`/opt/SUNWspro/docs/index.html`

`/opt/SUNWspro/READMEs`

`http://docs.sun.com` (or locally installed AnswerBook collections)

Compiler Flag `-fast`

The most important flag is `-fast`, which is a macro consisting of many different flags. Note that `-fast` differs between different compiler versions.

In the Forte Developer 6.2 software, `-fast` expands a variety of flags. The flags common to FORTRAN and C are:

`-xtarget=native`

`-xO5`

`-fns`

`-fsimple=2`

`-dalign`

`-xlibmil`

FORTRAN only flags are:

`-xdepend`

`-xvector=yes`

`-xprefetch=yes`

`-ftrap=%none (f77)`

`-ftrap=common (f95)`

`-xlibmopt`

`-pad=local`

C only flags are:

`-ftrap=%none`

`-fsingle`

`-xbuiltin=%all`

Also note that `-fast` optimizes for the machine on which the compilation is done. If you don't compile on the target machine on which you will run the application, you might need to set things like `-xarch, -xcache` and `-xchip`.

Always verify that your results are correct, and check the program execution profile in the Performance Analyzer.

Also in some of the flags included in `-fast,` the basic optimization level is set with `-O`*n*, where *n* is a number between 1 and 5:

n **=** 1 – *Basic block optimization*

n *= 2 – Global optimization*

n **=** *3 – Loop unrolling and modulo scheduling*

n *= 4 – Intrafile inlining and pointer checking*

n *= 5 – Aggressive optimization*, including profiling feedback

The optimization level seldom causes any problems with program accuracy.

The Solaris operating environment is a 64-bit operating environment that allows execution of both 32-bit and 64-bit applications concurrently. If your application requires an address space larger than 4 gigabytes, you must compile for 64 bits. You do this by setting the `-xarch` flag to `v9`[`a`,`b`]. Using `-xarch=v9b` means that it should be compiled for 64 bits, using the full instruction set in the UltraSPARC III processor, including UltraSPARC III processor specific instructions. However, a program compiled this way will not run on a machine with an UltraSPARC II processor.

If you don't need 64 bits, 32-bit code is usually slightly faster. In this case, use `-xarch=v8plus[a,b]`. Note that if you compile with `-xarch=v8`, you will generate generic SPARC® V8 code that will run also on old SuperSPARC™ processors, but you will, for example, not make optimal use of all the floating point registers in the UltraSPARC processor.

The flag `-xtarget=native` means that you optimize for the machine you are compiling on. Optimizing the code for a specific CPU pipeline and cache size will only affect performance, not compatibility. If you, for example, optimize for the UltraSPARC III processor, the code will run on an UltraSPARC II machine, but not at optimal speed. The pipeline to optimize for is set with the flag `-xchip=[ultra2,ultra3, ...]` The cache to optimize for is defined with the flag `-xcache`. Here are two examples:

`-xcache=16/32/1:2048/64/1`—UltraSPARC II processor with 2 megabyte L2$ (direct mapped)

`-xcache=64/32/4:8192/64/1`—UltraSPARC III processor with 8 megabyte L2$ (direct mapped)

The Forte Developer software includes a program called `fpversion` that you can use to find the characteristics of the machine you are using.

The option `-fsimple` selects floating point optimization preferences. You set it with `-fsimple=`***n***, where ***n***, can be:

n = 0—Strict IEEE754 conformance (default)

n = 1—Allows conservative simplifications. Numeric values are not likely to change.

n = 2—More aggressive floating point optimizations. Numeric values are likely to be slightly different due to roundup differences.

Setting `-fsimple` can often have a significant impact (up to 20 percent) on performance, but you must verify the accuracy of the results.

The `-xdepend` option is an important option for performance, as loops are analyzed for dependencies and possible transformations such as loop interchange and loop fusion. It requires an optimization level of at least `-xO3`. This option is included in `-fast` for FORTRAN but not for C in the Forte Developer 6.2 software.

The `-xprefetch` option can hide memory latencies by issuing fetch instructions ahead of time, so that the data is available when it's needed for computing. This option does, however, take up an instruction slot, and might therefore make some code run slower. An UltraSPARC II processor CPU or later is required and at least `-xarch=v8`. You can add prefetch directives to the code, or just let the compiler do it is best.

Multifile Program Optimization

In many cases a big program is split into multiple files during the development. This method makes the development process easier. It is fast to just recompile the file you working on and relink (using make). However, from an application performance standpoint, this method is not optimal, as the compiler only compiles one file at the time and has no knowledge of the rest of the program. By adding the option `-xpio` to the compiler flags, you allow the compiler to optimize the whole program and, for example, inline routines from separate source files. Previous to version 6.2 of the Sun Forte Developer software, the way to get around this was to either concatenate the files before final compilation or compile all files in a single command and use the `-xcrossfile` option.

Parallelizing Your Application.

Essentially there are two ways to parallelize a code—shared memory parallelization and message passing. If you want to use shared memory parallelization, the Forte Developer software is the tool you need. It supports the OpenMP program development environment. To use this tool, you must add OpenMP compiler directives to your source code, compile with `-xexplicitpar -mp=openmp`, and then set the environment variable `OMP_NUM_THREADS` to the number of threads you want before running your code. You can also let the compiler do its best without adding any directive and, in some cases, even this can produce a respectable program speedup.

The other parallelization possibility is to use message passing. The Sun HPC ClusterTools 4.0 software is Sun’s current MPI implementation. It is based on the MPI development environment that Sun acquired from Thinking Machines. Besides the MPI libraries, it contains a Scientific Subroutine Library called S3L and a very powerful debugger / data visualizer called Prism debugger software. The Sun HPC ClusterTools 4.0 software supports jobs running on up to 2048 CPUs on up to 64 nodes.

Conclusion

In many real life systems, the application performance is far from optimal. A lot of improvement can often be achieved by measuring how the system behaves and configuring the system optimally (regarding memory, disk I/O, network, and so forth).

If you have access to the application source code, compiling with the latest compiler version and with compiler optimization enabled will make a significant improvement. In most cases, compiling with `-fast` is a very good start. Also, keeping the code as simple as possible and using optimized libraries when possible helps a lot.

Author's Biography

Börje Lindh

Börje has over 13 years of UNIX software experience. He joined Sun in 1994 as a Systems Engineer, and is now a Technical Computing Specialist at Sun Sweden.