from *SIAM News*, Volume 31, Number 7

Porting Scientific Software to Intel SMPs Under Windows/NT

By Bob Kuhn and Eric Stahlberg

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS

Greg Astfalk, Editor

A user interested in porting scientific software to an Intel-based computer running Windows/NT will encounter challenges not necessarily associated with porting from one Unix system to another. While not intended as an exhaustive guide for such users, this article addresses many of the issues that come up in practice. As proof-of-concept, we present our experience in porting the Fortran computational chemistry code DGauss from a Unix-based system to a Windows/NT system.

Scientific software for PCs is of current interest because, with the floating-point performance of the Pentium II, Pentium Pro, and Pentium II Xeon processors approaching that of RISC workstations, Intel-based platforms are able to compete on a performance basis with scientific workstations. The Pentium Pro, the Pentium II, and the newest Pentium II Xeon processors (all referred to simply as Pentium systems in the remainder of this article) now have sufficient memory and cache to meet the demands of many scientific computations. The operating systems, compilers, and system software on the Intel-based systems are able to support efficient execution of scientific application software. Moreover, these systems now have SMP (symmetric multiprocessor) technology to enable parallel computation for compute-intensive applications.

The guidelines presented here follow the same process that should be applied in migrating any large piece of parallel software to a new platform. The stages are:

- Migrate the single-processor version;
- Enable parallel processing;
- Carry out performance tuning;
- Perform quality and assurance testing.

Overview of DGauss

The DGauss program, our sample code for porting to NT, is a component of the UniChem Molecular Modeling System, which is available from Oxford Molecular Group [4]. DGauss has more than 1100 subroutines and 160,000 lines of source code. It is a true supercomputer application, having been developed at Cray Research starting in 1989. It was one of the first chemistry applications to demonstrate sustained gigaflop performance on the Cray Y-MP. The vector, symmetric multiprocessor capabilities of the Y-MP were utilized via Cray-specific source code directives to achieve this performance.

The program's versatility was extended in 1993 with the availability of versions for workstation-class machines, beginning with MIPS R4000 processor-based Silicon Graphics workstations. More recently, a directive-based approach available from SGI was used to implement SMP parallelism on the SGI Challenge.

Advances in massively parallel processing (MPP) led to the porting of DGauss to the Cray T3D, where it has been run with more than 256 processors. This phase of development necessitated a generalization of the parallelization approach used in DGauss, which greatly contributed to the ease of porting it to a parallel Windows/NT platform.

Migrating from RISC Unix to an Intel Windows/NT System

It is advisable to start the first step of the porting process—migration of the scalar version from Unix to Windows/NT—with a stable, well-debugged version of the software. In most instances this should be a previously released and supported version. It is not advisable to make any feature changes to the software as it goes through the porting steps.

The major topics to be considered are:

- Selection of compilers and software tools;
- Data format translation;
- Interlanguage procedure calling;
- Support library conversion.

The preliminary work done to support the Windows/NT platform should focus on selection of the right system software tools. A poor choice at this point can hurt the product for years. There are several compilers to choose from on Windows/NT systems. The necessary capabilities should be identified before a compiler is selected. Important considerations include support for Fortran features to be used, interlanguage communication, limits on the complexity of source code, and performance of the generated code.

Initially, we found the Microsoft Powerstation Fortran and C compilers to be capable of fulfilling all the requirements for compiling the program. Since then, we have found Digital Visual Fortran [1], subsequently announced as a logical successor to Powerstation Fortran, to have Unix Fortran features not present in Powerstation Fortran. Users may find compilers they like for

other reasons, however.

Preprocessors are commonly available for C programs through the specification of options for compilation. Fortran source code preprocessing, however, is not as easily achieved. Digital Visual Fortran contains a preprocessor, **`fpp`**. For DGauss, we achieved better results by utilizing the C preprocessor available with the Microsoft C compiler (invoked as **`cl -EP`**). In some cases we have had to use the GNU C preprocessor to migrate Unix applications to Windows/NT.

Compiler options must be reconsidered each time an application is ported. An option comparable to that used on one platform is commonly found on other platforms. We recommend, however, that compiler optimization options be turned off until the tuning phase. One option likely to be needed forces the compiler to allocate locals on the stack instead of statically. This is necessary to make the code thread-safe for parallel execution. Data format translation, discussed later, can sometimes be handled by compiler options.

Makefiles and run-scripts require conversion for use under NT. A Unix **`make`** file can be converted easily to NT's **`nmake`**. A user may wish to write a simple compiler wrapper script to convert the file types Unix expects, e.g., "**`.o`**," to the types NT expects, "**`.obj`**." In this manner the makefile remains portable. A number of helpful tools make it possible to convert the development process in stages. For example, we have used the MKS Toolkit [3] to provide Unix tools under NT. Use of such tools allows us to focus on the more critical issues of the software. A number of GNU software tools are also available for NT [5].

The second step in the migration process for the sequential code is to consider data-formatting issues. This involves selecting the basic datatypes used by the code and developing a plan for migrating existing data from the current platform to the new platform. In selecting basic datatypes, both floating-point and integer representations need to be considered. The interlanguage communication requirements must also be considered. In our case, maintaining sufficient precision for computational chemistry applications is critical. Consequently, all real variables and constants were required to have 8-byte (i.e., 64-bit) representations, **`real*8`** in Fortran and **`double`** in C. Selection of integer size was based on the number of bytes in address words on the target platform, in this case 4 bytes.

For correct specification, some language constructs have to be made more explicit under Windows/NT. Some specific examples we encountered in porting DGauss were:

- All subroutine references must have arguments consistent with the definitions; otherwise, unresolved references can occur in loading.
- Integers used for dimensioning arrays must be declared before the arrays when **`implicit none`** is used.
- Because the compiler does not automatically promote parameters to the correct floating-point type, this must be done explicitly with **`real*8`** or **`double precision`** declarations.

An important consideration is that many of the initial numerical differences between the Pentium and RISC workstation results were traced to the representation of **`double precision`** constants and parameters employed by the compiler. Applications can also include nonportable assumptions about the stored value of **`.true.`** and **`.false.`**.

The third step in the migration process is to resolve the interlanguage communication between Fortran and C. An initial load pass indicates where it is necessary to reconcile differences between Fortran and C language external symbols, as shown in the following example:

```
void _stdcall UNISCFMON( int *SCF_IterationNumber,
                         int *SCF_Spin,
                         double *SCF_Fock_Error,
                         double *SCF_DensityFit )
```

This specification allows the Fortran routine to reference UNISCFMON with the appropriate two integer and two 8-byte real arguments. The Windows/NT compilers allow a number of calling conventions to be specified. A check for compatibility of conventions should be done on a test program before an entire application is built.

The final step in the single-processor migration requires building the interface to all the system utilities needed for the application. Migrating from Unix to Windows/NT sometimes requires a search for a comparable function and some creativity in finding a workable solution.

Most Unix platform vendors support tuned BLAS operations. Intel provides the Math Kernel Library, MKL [7], for tuned BLAS operations. Vendors have extended tuned libraries to include higher-level routines for matrix operations, including diagonalization, FFTs, and solution of linear systems. On Windows/NT, high-performance numerical libraries, e.g., the NAG library [6], are also available.

Enabling Parallel Processing on Intel SMPs

The process of invoking parallelism in software that has been migrated to a Pentium NT system involves three steps:

- Selecting the best method for implementing SMP parallel processing;
- Writing code to implement parallel processing for that method;
- Debugging the chosen parallel implementation.

In order to realize the full potential of the new Intel multiprocessor servers, it is necessary either to leverage the SMP parallelism expressed in the existing code or to find tools that will help in the process. With DGauss, we explored a number of mechanisms for accomplishing this. Our requirements in this search were to minimize source code modifications, to be compatible with existing supported platforms, and to use software already available on the Pentium platform.

The four commonly used mechanisms for parallel processing are automatic parallelization, threads, directives, and message passing. For directive-based parallelism on an SMP, the new OpenMP [8] standard meets all these criteria and was particularly appropriate for DGauss. DGauss used a subset of the Cray multitasking and SGI **doacross** directives. Because SGI and Cray support OpenMP, conversion to OpenMP would improve the Unix version as well.

The following code fragment illustrates Cray and SGI versions of a parallel region (with ". . ." indicating that a list of variables has been removed from the example for simplicity):

```
c
c   CRAY directives
c
cmic$  do parallel shared(FOCK, ATOMS, ...)
cmic$*             private(PROC,NWRK_L, ...)
c
c   SGI directives
c
c$doacross  share(FOCK, ATOMS, ...)
c$&         local(PROC,NWRK_L, ...)
c
       do PROC = 0, Ncpus_get() - 1
```

In OpenMP, this is replaced by one specification for both systems:

```
c
c   OpenMP directives
c
c$omp    parallel do
c$omp&           shared(FOCK, ATOMS, ...)
c$omp&           private(PROC,NWRK_L, ...)
         do PROC = 0, Ncpus_get() - 1
```

The KAP/Pro Toolset (KPTS) [2] supports OpenMP on Windows/NT systems and includes a translator for converting Cray or SGI directives to OpenMP directives.

It is notoriously hard to find bugs in parallel programs, which is a primary reason for starting the migration to parallel processing with stable uniprocessor source code. Bugs do appear, however, and there are three distinctly different approaches for finding them. The first, straightforward approach is to use print statements as probes at the appropriate points in the program. A second approach is to use a multithreaded debugger. These are difficult to find on Windows/NT systems at this time. The third is to use bug-finding tools, e.g., the unique KPTS Assure tool, which will check the program for flaws in the parallel directives much more rigorously than possible for a human using either of the first two approaches.

Assure looks for potential problems while compiling and executing the application. It produces a database of suspicious parallel processing events. The types of events considered suspicious by Assure are:

- References to shared variables that have not been synchronized, which is the single most frequent error in parallel programs;
- Thread-private variables that have not been properly initialized;
- Thread-private variables referenced outside the parallel region.

The Assure tool was used with the folllowing code, which is assumed to be within a parallel loop:

```
     If (IandJ) then
        . . .
        do ll = 1, Nvec
          density_value = dd(nij,ll)
          do k = 1, nfit
132         Vrhs(k,ll) = Vrhs(k,ll) + buff(k,npp)*density_value
          enddo
        enddo
     else
        . . .
        do ll = 1, Nvec
          density_value = dd(nij,ll)
          do k = 1, nfit
163         Vrhs(k,ll) = Vrhs(k,ll) + buff(k,npp)*density_value
          enddo
        enddo
     endif
```

Here is what Assure reports for this case:

```
  Conflict       Source   Source                Source
  Type           Symbol   Routine/Line Number   Filename
-------------------------------------------------------------------
Write --> Read   VRHS     J0RHS/132 - 163       C:\j0int\j0rhs.for
Write --> Read   VRHS     J0RHS/163 - 132       C:\j0int\j0rhs.for
```

VRHS is a shared variable that can be written by one thread at line 132 when **IANDJ = .true.**. However, **VRHS** can be read by a different thread at line 163 when the value of **IANDJ = .false.** on a different iteration of the parallel loop. This is a parallelism bug called a "data race." Frequently, data races are fixed by the addition of a critical section that sequentializes access to the shared variable.

Although powerful, Assure cannot be run on all parallel programs. The OpenMP directives make structured portable parallelism easy. Older parallel programs can describe parallelism in ways that are unstructured, unportable, and unverifiable. Converting to OpenMP for better debugging made sense for DGauss.

Performance Tuning on Intel SMPs

Overall, we found the performance of DGauss on an SMP Pentium system to be very good. After a few days of tuning, the application speedup was about 10% less on the Pentium SMP than on a RISC SMP. The difference is most likely related to the operating system and could change in the near future. The amounts of performance tuning necessary for a particular application can vary significantly. It is harder to migrate software to a Pentium system from a Cray than from an SGI or other RISC SMP because of the greater similarity of the RISC and NT architectures.

We found our target platform more sensitive to subtle changes in the code than most Unix systems. Which options were used in compiling the code, how the code was linked, and whether the cache was warm or cold had significant performance impacts. At several points, we made incorrect assumptions that led to wrong conclusions. Therefore, we believe that it is important to take a systematic approach to performance tuning on Intel SMPs. The general process of performance tuning consists of isolating and then fixing performance problems.

The performance-tuning cycle is repeated several times, with bottlenecks exposed and removed until the software is running with good performance in a balanced way. In some instances, a particular bottleneck can require significant reengineering; the tradeoff between performance payback and work involved must then be evaluated. Only relatively small changes (compile options, directive options, and math subroutines) were used to achieve very good performance on DGauss—once we figured out what the performance bottleneck really was.

Several performance analysis tools are available on NT, although as yet not many are oriented toward parallelism. For the simple performance tuning we needed for DGauss, GuideView [2], which focuses exclusively on SMP directive-based performance, was a useful tool. It gives information on system performance bottlenecks, such as the time spent waiting to enter critical sections, the time spent in critical sections doing productive but single-threaded work, and the time spent scheduling parallel work. GuideView isolates performance problems by analyzing performance for each parallel directive used in the program. Figure 1 shows the GuideView performance analysis for a two-electron integral computation that is part of DGauss.

As shown in the figure, all the sequential code combined, called R0, takes more time than all but one parallel region, R22. This is the performance bottleneck. Parallel region R20 is also interesting because the parallel performance is masked by parallel overhead.

Once performance problems have been isolated, the changes should be divided into two types: (1) those that improve the parallelism of the methods and algorithm being used and (2) those that improve the performance on a particular target machine. Both can provide significant payback, given Amdahl's law, but changes of the latter type frequently require additional modifications when the code is migrated to another platform. For example, reducing the serial time in R0 will help on all platforms, but improving R20 may be less valuable because of the variation in overheads from system to system.

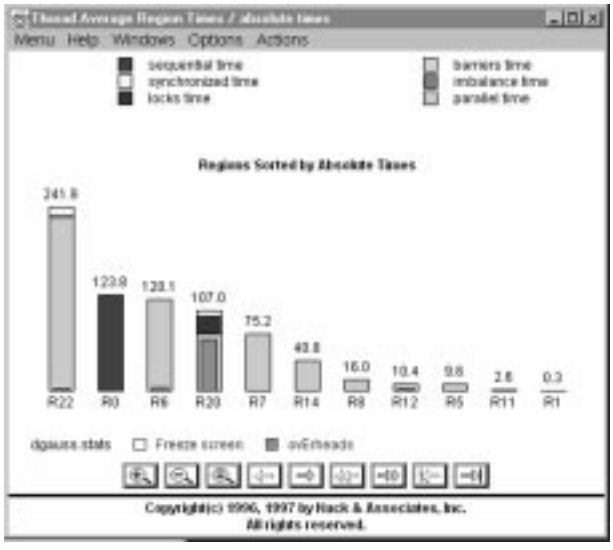

Figure 1. *Output window from GuideView.*

DGauss Performance

After a limited amount of tuning of DGauss on a Pentium SMP, we ran two analyses on the four benchmark problems shown in Table 1 on several systems. The examples are from a series of datasets generally used to evaluate the performance of DGauss across a range of computer architectures. Table 1 gives the parameters that determine the size of the overall calculation. The computational requirements scale between the second and third power of the number of basis functions.

The data in Table 2 show that the average speedup with two processors was 1.56. Performance of the new Intel systems with the two processors was competitive with that of an SGI/Cray J90. In

earlier tests, systems with four 200-MHz processors had a speedup of 2.5, showing a respectable scalability.

Summary

With the popularity of Intel SMPs growing rapidly, it is important for application users to be able to achieve the best possible performance. We have described a process for migrating an application parallelized on a Unix system to an Intel platform running Windows/NT.

The process, based on our experience with DGauss, was reasonably easy to execute with tools now available. The performance achieved on this new platform was striking. With hardware expenditures in the range of tens of thousands of dollars, performance rivaling that of a Cray can be achieved on this and many other applications.

	Atoms	Basis Functions	Grid points
Si_2OH_6	9	65	9K
$Si_2O_7H_6$	15	155	18K
$Si_8O_7H_{18}$	33	293	36K
$Si_8O_{25}H_{18}$	51	563	63K

Table 1. *Four benchmark problems.*

	400 MHz Pentium II Xeon			Cray J90
	1 CPU	2 CPU	Speedup	1 CPU
Si_2OH_6	28.6	18.3	1.57	25
$Si_2O_7H_6$	267.5	171.8	1.56	180
$Si_8O_7H_{18}$	1744	1106	1.58	912
$Si_8O_{25}H_{18}$	7222	4731	1.53	3840

Table 2. *Performance of the computationally intensive portion of DGauss (seconds).*

References

[1] *Dinosaurs Do Dance* (a review of DIGITAL Visual Fortran), BackOffice Magazine, August 1997 (http://www.digital.com/fortran/index.html).

[2] B. Kuhn and E. Stahlberg, *Porting Scientific Software to Intel SMPs Under Windows/NT*, Scientific Computing & Automation, November 1997, Part I, and January 1998, Part II (http://www.kai.com/).

[3] P. Coffee, *MKS Toolkit a boon for power users*, PCWeek, January 1997 (http://www.mks.com/).

[4] DGauss v. 4.1, Oxford Molecular, Medawar Centre, Oxford Science Park, Oxford, OX4 4GA, England (http://www.oxmol.com/).

[5] Descriptions of GNU software packages (http://www.gnu.org/ or http://www.cygnus.com/).

[6] NAG Fortran SMP Library, NAG Ltd., July 1998 (http://www.nag.co.uk/).

[7] B. Greer, *The Most Important Technical Library in the World*, Fortran Futures 98, May 1998 (http://developer.intel.com/design/perftool/perflibst/mkl/index.htm).

[8] *OpenMP Fortran Application Program Interface*, October 1997 (http://www.openmp.org/).

Bob Kuhn (kuhn@kai.com) is with Kuck & Associates, Inc., in Champaign, Illinois. Eric Stahlberg (eric_stahlberg@ibm.net) formerly worked at Oxford Molecular Group, Inc., and is now with Vital Images, Inc., in Minneapolis, Minnesota.