Helping to choose the right commodity compilers for High Energy Physics

Nenad Buncic, [a] Sverre Jarp, [a] Christophe Meessen, [b] Hong Tang, [c] Refael Yaari [d]

[a] *IT Division, CERN, CH-1211 Geneva 23, Switzerland,*

[b] *CPPM, Marseille, France,*

[c] *NETIQ, Palo Alto, United States,*

[d] *Weizmann Institute, Israel*

We describe the compiler characterisation and testing that has been carried out in RD47 so far. The main emphasis has been put on native Windows/NT compilers, but references, especially concerning performance, have also been made to the gnu compiler running under Linux.

Key words: PC, x86, Compiler, Comparisons, Windows/NT, Linux, Benchmarks, IDE.

This paper summarises some of the compiler experience that has been collected so far inside the RD47 effort at CERN. Emphasis has been put on compilers running under Windows/NT since this was the chosen operating system environment of the R&D collaboration, but a direct comparison to the GNU gcc compiler for Linux has been made in the performance section.
The Windows/NT compilers are described in a sequence corresponding to how they were adopted and tested. Given that a new version of a compiler appears approximately every 12-18 months, we have tested several versions over the last couple of years [1] . If nothing else is specified, the description in this paper will always refer to the latest version which is normally a superset of previous versions. One exception to this "rule" is the recent FORTRAN compiler for Microsoft Developers' Studio (DevStudio), where the Digital FORTRAN compiler has replaced Microsoft Powerstation 4.0 as of March this year. In summary our compiler migrations have been as follows:

- Watcom C++/FORTRAN version 9.5 → 10.5 → 10.6 → 11.0

[1] Although RD47 was officially started in the spring of 1996, some of this work had started already in the previous year.

Preprint submitted to Elsevier Preprint *3 June 1997*

- Microsoft Visual C++ v. 2.x → 4.0 → 4.2 → 5.0
- Microsoft FORTRAN Powerstation v. 2.x → 4.0 → Digital FORTRAN 5.0
- Intel C/FORTRAN Reference Compilers v. 2.1 → 2.3 → 2.4 (beta)
- GNU C/C++ compiler v. 2.7.2 (under Linux)

Neither the Borland C++ compiler nor the Symantec C++ compiler have been deployed until now [2]. One reason was simply lack of time; there were too many compilers (and compiler versions) to test. Another reason was that, at least initially, we were very much focused on compiler suites that offered both a FORTRAN and a C/C++ version. In recent testing, the experience with f2c (under Linux) has shown us that the lack of a native FORTRAN compiler is not so dramatic, and it is hoped that at least one of these two remaining compilers will be covered before RD47 ends.
The evaluation strategy has concentrated on the porting of classical physics (batch) applications but in the future experience will also be gathered for graphical applications, in particular in the use of OpenGL and Open Inventor for visualisation.

1 Watcom C/FORTRAN Compilers

1.1 Overview

When we first began porting the CERN library and physics applications to Windows/NT in the first half of 1995, we chose the Watcom/Powersoft compilers (version 9.5), for the following main reasons:

- Relative ease of porting (i.e. acceptance of a syntax similar to that of UNIX compilers)
- Ease of mixed language use
- Relatively good performance
- Good set of integrated tools (Editor, Make facility, etc.)

1.2 Relative ease of porting:

With a serious physics application, such as DICE (plus CERNLIB), requiring in the order of half a million lines of code to be ported, this was clearly a top priority. When the CERN benchmarks [1] were ported as a preamble, there

[2] Two additional compiler suites are about to enter the Windows market: The popular Macintosh product, Codewarrior, by Metrowerks and the ABSOFT family of compilers.

were almost no changes required to make them run. The porting issues that occurred were already reported in a presentation given at CHEP-95 [2], and the detailed issues have also been summarised in [3].

1.3 Ease of mixed language use

In order to produce a working version of the CERN library, this feature became highly relevant due to the inclusion of C code in KERNLIB and HBOOK(?). With the Watcom compilers, this issue is completely painless when the "stack convention" is used for passing parameters. It is a bit more awkward when the "register" convention is used.

1.4 Relatively good performance:

As already reported in the CHEP95 presentation, we were first able to get 17.7 CERN Units (CUs) out of a Pentium 133 MHz system using these compilers. This was annoyingly close to 20 CUs and shortly after the conference, we were able to boost this number to 20.2 [3]. Since such PCs quickly fell below 2000 USD, we claimed around January 1996 that the magic boundary of 100 USD/CU has been crossed. At the same time Pentium Pro systems started to appear on the market and these were measured to be just over 40 CUs at 200 MHz. Given the fact that these systems were more expensive at first (in the range of 5-8000 USD), they did not improve on the CERN Unit cost. Even dual processor configurations did not help, since such "flagships" only appeared with the high price already mentioned. Today we reckon that a "low-end" CERN Unit can be obtained for less than 50 USD.

1.5 Good Set of Integrated Tools:

The Watcom compilers come with a good set of tools, which although they may not be regarded as "flashy", are very appropriate for porting and code development. The main tools (Editor, the Make facility (WMAKE), the Library management tool (WLIB), and the Linker (WLINK) can be used independently of the Integrated Development Environment (IDE), and this was

[3] This increase was made possible by a new release of the compilers (10.5), and better use of compiler options, such as -NOSC (to pass parameters in registers, rather than on the stack). Many of these results were reported in an internal CERN/CN document: "PC Benchmarking" CN/DPG/96/05), also accessible on the Web as http://wwwcn.cern.ch/pdp/wise/pc_bench.htm.

the way we started. The profiling tool is outstanding. It generates a graphical frequency distribution of the modules where the execution time is spent. By clicking on a routine, one gets the individual methods (subroutines); by clicking in one of these, the source lines are displayed; by clicking again, the assembly lines appear; all with illustration of the percentage of time spent in each location.

1.6 Integrated Development Environment

The IDE, offered by Watcom, is certainly less "fancy" than that of the competition, but it has sufficient functionality to assist greatly in the efforts of firstly writing code, compiling and linking, debugging the code and finally sampling the execution for performance enhancements. It is, nevertheless, understandable that some people, may find this IDE somewhat rudimentary, especially when all of the options or all the ways of configuring the environment are not yet fully mastered.
The IDE allows a separate set of options to be used for a development version and a release version of the code (See Target Options in Target Menu), but lacks the capability of using separate sub-directories for the two object files created. After having set the options for the Development version, it is possible to copy settings to the release version. This avoids, for instance, the problem of losing pre-processor definitions or include-library settings. This copy capability exists for all the components such as the compilers themselves, the Make utility, the WLINK or WLIB switches and works in both directions (CopyDev or CopyRel). It is also possible to set different switches for one individual routine (a typical case is to lower the optimisation for such a routine), but we could not find a way of pre-setting these switches (See Source Options in the Source Menu) to those already in use for the rest of the project. The options come straight from the configuration file which is global for all projects on a given system.

1.7 IDE Settings/Compiler options

A lot of experience has been gathered with the large number of settings that are available in the Watcom IDE. In the following list we try to highlight some of the options that developers ought to be aware of (either because they are particularly useful, or particularly harmful):

- Useful:
  - TR - Allows a full trace-back to be generated in case of a crash in a FORTRAN executable

    * NODESC - Allows character strings to be passed correctly between FORTRAN and C.
    * WIL - Allows wild branches (which may still be found in HEP programs, unfortunately)
    * K - Allows WMAKE to continue after an error (not the default)
- Harmful (or annoying):
    * Or - Allows code to be relocated (with the consequence that program behaviour may change)
    * 3r - Allows parameter passing in registers (this is useful !), but it also suppresses generation by the C compiler of a trailing underscore which is expected by FORTRAN.

Another annoying behaviour is the fact that when the application is run from inside the IDE, the window simply disappears at the end of execution rather than halting at the end with a "Please press any key to continue" message. This is circumvented, however, by opening a permanent DOS window for such testing.

2 Microsoft C/FORTRAN Compilers

2.1 Overview

The Microsoft compilers and Development Environment (DevStudio) have taken an increasingly important role in RD47. The reasons are the following:

- It is the leading compiler on the market and was chosen by the IT/ASD group for the first official implementation of CERNLIB/NT.
- The compiler environments are identical between hardware platforms. This feature covers x86 and Alpha, but no longer covers PowerPC and MIPS due to the discontinuation of the support for these platforms.
- DevStudio offers a rich environment with enormous amounts of information and a powerful search capability.
- Through successive releases the compilers have changed from generating "sluggish" execution speeds to top-notch.

2.2 The Market Leader

The Microsoft compilers must be seen as the market leader (especially by those who want both FORTRAN and C/C++ versions). Although we are not advocating that the 'market leader' argument should be followed blindly, one

has to accept that this is often caused by good technical arguments (see the following points), and by the fact that these compilers enjoy the best integration with the underlying operating system.
They are also the compilers that young students from a university background are most likely to be familiar with (Microsoft has agreements with many European Education Ministries).

2.3 Multi-platform availability

Windows/NT is a portable operating system, and although only x86 and Alpha remain as viable alternatives, it is important to have a compiler environment that is identical in these cases. In the UNIX world, this can be achieved by using the gnu compilers across different platforms, and this is indeed a strategy that has been adopted by certain HEP experiments, including BaBar.
In RD47 we demonstrated this feature by preparing a Microsoft project under x86 using the source of Geant4. When the project was debugged we asked C.W.Hobbs from the Digital Joint Project to move the project to Alpha to see what the problems would be. With the exception of generating Rogue Wave's tools.h++ (which is a pre-requisite for Geant4) the port was completely painless.
In DevStudio version 4 the project control files were binary files, so that it could be cumbersome to edit them to change a disk letter (or something similar), in version 5 they have become ASCII files.

2.4 Rich Development Environment

DevStudio is a very rich development environment, some people will even argue that it is overwhelming for the novice user. Combined with the help files and "books" from the distribution CD or the knowledge base from the Microsoft Developer's Network (MSDN), the developer gets into an environment that provides all necessary information about WIN32 programming interfaces, usage examples, language syntax rules, and development-related topics such as MFC (Microsoft Foundation Classes), OLE (Object Linking and Embedding), and ODBC (Open Database Connectivity) to mention just a few.
The development environment contains all the tools for editing, compiling, linking, debugging, library maintenance, sampling and profiling. A find-in-files utility is also included and all of this is woven together inside one environment that can be controlled by pull-down menus or control buttons very much like a word processing environment, such as MS Word.
The profiler is much more primitive than the one from Watcom but this is one of the few areas where Watcom gets the upper hand. Another Microsoft

problem occurred in 4.0 (or 4.2) when many files were added to a project. DevStudio would chop the list at a fixed position, ignore the rest, and inform the user that the source file that had had its name chopped in half looked suspicious ! Fortunately this is fixed in 5.0 and the equivalent of Watcom's "Add All" works correctly. The principle gripe remaining is now the fact that DevStudio does not allow a target to be redefined. With Watcom the developer can change an executable into a static library (.lib) or a dynamic link library (.dll) and this can be very useful.

2.5 Performance

When we chose the Watcom compiler, some quick comparative benchmarks indicated that the Microsoft compiler generated binaries with inferior execution speeds. Through successive improvements in version 4 and 4.2, the situation is now reversed and without revealing all the details in chapter 4 of this paper, Microsoft is now the leader in both compilation speeds and execution speeds (with the sole exception of PBO-based FORTRAN execution; see chapter 3).

2.6 Code porting

The main problem with the Microsoft compilers, is (or was ?) the fact that the Powerstation FORTRAN compiler required a syntax different from what is used under UNIX (and therefore reflected in the sources we used). The changes can be summarised as follows:

- A string in a FORTRAN subroutine call expands into a pointer followed by the string length. This extra parameter does now have to be taken into account by explicit modifications in a called C routine.
- C/FORTRAN mixing requires the use of "__stdcall" and the called routines (from FORTRAN) must be in uppercase.
- The subroutine names are suffixed by @nn where nn is the number of bytes of parameters, e.g. @24 in the case of six parameters. This enforces exact signature matching and porting (of sloppy code) can become quite a nightmare. Note that the automatically-generated string length always counts as an explicit parameter in this case.

2.7 Digital FORTRAN comes to the rescue.

With Digital FORTRAN these annoying features become options and the user can choose both the string-passing convention and the signature suffix. Since

Visual FORTRAN is incompatible with Powerstation libraries (as soon as any I/O is performed), this is probably a reason to deviate from the Microsoft options and embrace the ones used by both the UNIX compilers and Watcom.

3 Intel C/FORTRAN Compilers

The Intel Reference Compilers (IRC) serve one main purpose; namely the generation of executables that have the highest possible performance on x86 platforms. Intel has undoubtedly produced these compilers to help its own SPEC numbers, but the compilers are fortunately good enough to be of general interest. They appear in the form of cuckoo eggs; i.e. they are intended to be used only inside the environment created by a Microsoft compiler. In this fashion, they can either be used from the command line or the C-compiler can be transparently exchanged with the Microsoft one inside the DevStudio environment. The IRC compilers understand at least 90% of the options provided for the MS compiler, so such a switch is completely painless.
We did get consistently better performance with the Intel FORTRAN compiler, especially when using Profiler-Based Optimisation (PBO), although the difference was usually no more than 10-20%. Details can be found in section 4.
With the 2.4 beta version of the C++ compiler, we are able to compile all of Geant4, but so far the performance has not been better than the others. One reason for this is the fact that PBO does not yet work correctly. It is hoped that this will be fixed in the near future.

4 Performance Comparisons

4.1 Overview

As a criterion for evaluating these compilers, we performed a certain set of timing tests. These tests reflect not only the execution time of an optimised executable, but also compilation and linking times. The latter two are meant as an indication of the speed obtained in a development environment (where execution times are not so important), whereas the former reflect more the production environment where the time to create the executable is negligible compared to days or weeks of running time.
The tests consisted of the following Geant3 and Geant4-prototype cases:

- FORTRAN Gexam1/3.15 - Stand-alone.
- FORTRAN Gexam1 w/CERNLIB/3.21.

- C++ Geant404 Calorimeter (prototype) - Stand-alone executable.
- C++ Geant404 Calorimeter (prototype) - Executable built from library.

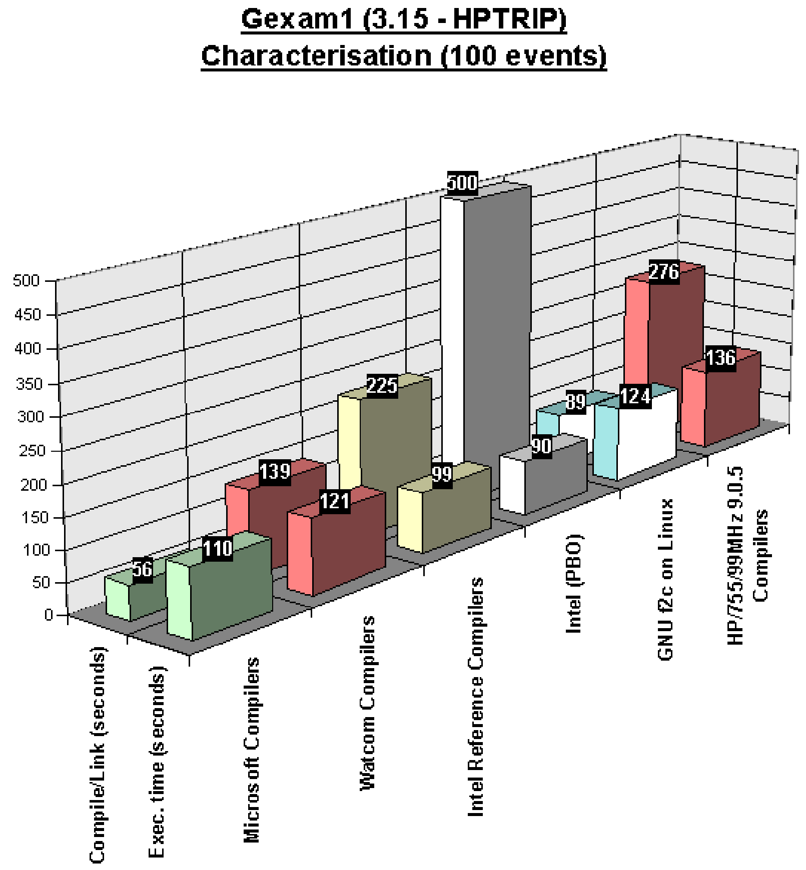
All the measurements were made on HP Vectra 6/200 MHz PCs with dual processors, 64 or 96 MB memory and two local disks. This may sound like an "expensive" PC configuration, but compared to traditional workstations, such a PC (for about 5,000 USD) offer an interactive development environment that is extremely attractive with the two processors totalling over 80 CUs.
The deployed compilers were the compilers from Microsoft, Watcom, and Intel under Windows/NT and the GNU f2c/gcc compiler under Linux. The GNU testing is in no way a complete set of measurements, but should give an indication of Linux performance compared to NT. It has been suggested that the tests be expanded to cover the native FORTRAN compiler and also the GNU compilers under NT to check that the resulting performance is the same. In one example the HP-UX 9.0.5 compilers were also used but there has so far been no intention of carrying out a systematic comparison with all the "native" UNIX compilers..

4.2 Gexam1 stand-alone

This first example, which has become a 'classic' benchmark test, consists of about 55,000 lines of FORTRAN extracted from the then current CERN library [4] , so that the Gexam Example 1 could be compiled without any need for pre-generated libraries. It also contains about 5,000 lines of C code (the necessary KERNLIB routines), which means that language mixing had to work in all the tested language environments.
In this two-dimensional chart the compilation/link time is depicted in the back row, with execution time for 100 events in shown in the front row. For compiling/linking the Microsoft compilers are the fastest, the Watcom compiler is twice as slow and the Intel compilers about twice as slow again. In order to get the best execution speed, Profile-Based Optimisation (PBO) was used with the Intel compiler. In this case the total time reflects two invocations of the compiler with a little training example run in between. This creates a binary which runs about 20% faster that the Microsoft executable. The GNU/Linux compiler gets the second best compile/link speed and the execution time is quite respectable. The HP/9000/755 (99MHz), which used to be the reference platforms for simulation jobs a few years ago, now provide results that are rather unimpressive.

[4] This extraction was done in 1992 in connection with a benchmark trip to HP carried out by R.Brun. For this reason the test is often referred to as "HPTRIP".

Figure 1



4.3 Gexam1 with library

In the second example the execution times from the first example are compared to the same run when Gexam1 is built from a library.

Table 1

	Stand-alone 3.15	Library 3.21
Linux/gcc	124 s	144 s
Watcom compilers	121 s	154 s
Microsoft compilers	110 s	383 s

It is normally admitted that Gexam1 timings increase by about 15-20% when moving from version 3.15 to 3.21 because of the modified physics behaviour. In these tests, The Linux library increased execution time by only 16% whereas

Watcom increased by 27%. The reason Microsoft increases by about 350% is the fact that the 96A version of CERNLIB is unoptimised. This makes the library unusable for large production, but probably adequate for development. In any case work is now underway to release CERNLIB 97A for NT as a fully optimised library.

4.4 Geant4 Calorimeter

In the third example we switched entirely to C++ code by adopting one of the "tagged" versions (4.0.4) of the Geant4-prototype. In the first variant, a calorimeter was generated by compiling all of the 161 routines that were needed to build an executable.

Table 2

	Compiling/Linking time	Execution Time
MS DevStudio (Alpha)	9.33 m	13.4 s
MS DevStudio (x86)	6 m	31.1 s
Watcom IDE	12.8 m	36.7 s
GNU gcc/Linux Compiler	8.9 m	33.7 s

These measurements show that, once more, Microsoft compile time on the x86 architecture is unparalleled. The same compiler on Alpha, which uses the Digital GEMM back-end requires about 50% longer, in spite of the fact that the machine, a 400 MHz Alpha-station is in fact much faster, i.e. the execution performance is 2.3x that of the Pentium Pro. The Watcom compiler behaves similarly to the FORTRAN case; compilation time is about a factor 2 slower and execution time is 18% worse than Microsoft.
The GNU results come from a straight-forward compiler invocation, i.e. without an integrated development environment. The compile/link time is about 50% worse that Microsoft, but execution time is only 8% worse.

4.5 Geant4 Calorimeter with Library

In this example we pre-generated a G4LIB which contained all the 160 routines required for building the calorimeter. The task was therefore reduced to a compilation of "calorimeter.cpp" and a consequent linking operation against the library.

Table 3

	Compilation time	Linking Time
MS Compiler	5 s	1.25 s
MS DevStudio	10 s	1.5 s
Watcom IDE	28 s	3.5 s
GNU gcc/Linux Compiler	10 s	1.1 s

First of all, there is good news to developers who used to think that linking was a time-consuming operation. For both Microsoft and GNU, the linking time is amazingly fast with GNU even a little bit ahead of Microsoft. For compilations, Microsoft DevStudio and GNU perform equally well, but the Microsoft speed can be improved by a factor 2 by doing the compilation outside of the development environment. This illustrates the fact that these development environments come with an associated cost. The IDE will keep track of dependencies and invoke the right "make" and build commands behind the scenes, but will function more slowly than a straightforward "make" from the command line. Fortunately both Watcom and Microsoft DevStudio (version 5, but not version 4) allow the project make files to be used outside of the IDE. In a typical development cycle, the IDE may be used heavily in the beginning, but as the developer gets a more and more mature product, he/she may be tempted to work outside the IDE to get the maximum speed.

5 Future work

Fortunately (or maybe unfortunately) there is no absolute termination point for the evaluations we are doing inside RD47. The following options are high on our list of items to be included before RD47 terminates:

1) Ensure generation of a properly optimised version of CERNLIB/97A for Windows/NT together with the CERN/IT/ASD group and produce a proper characterisation in terms of execution speeds. In this activity, one could even envisage more than one version of the library being made available.
2) Bring the Atlas DICE simulation program up to the latest level. Today, we have a version of DICE that corresponds to level 96_12 of the Atlas code and the source level 96A for the library. The executable has been generated with both the Watcom and the Microsoft/Digital compilers. The natural termination point for this work would be to ensure that Atlas has a reliable Window/NT version of DICE that can be run on the RD47 PC farm which is about to be established by the CERN/IT/PDP group.

3) Repeat the Geant4 tests based on the alpha version which is about to be released. The purpose would be both to generate a more realistic benchmark than the very early version of the calorimeter described above, but also to include characterisations of the visualisation environment of Windows-based PCs, especially by focusing on OpenGL and Open Inventor. Simulated X-servers should not be excluded provided the can be offered at "commodity" cost levels.
4) Include tests of other compilers. It might be interesting to test either the Borland or the Symantec compiler, or the Metrowerks newcomer in the NT market. Furthermore one should not exclude the gcc compiler for Windows/NT, especially given the encouraging results with the same compiler under Linux. It must, however, be understood that this NT compiler relies on the availability of the Microsoft include-libraries, which come with the purchase of the MS C/C++ compiler, so it cannot be seen as a "free replacement" for the latter.

6 Conclusions

There is an abundant choice of compilers for Windows/NT. This reflects the opportunities in a gigantic market. We are not the only community to evaluate the compiler market and there was recently an excellent article about C++ compilers in Dr. Dobb's Journal [Ref. 4]. PCs are not the absolute speed demons in terms of execution speed, but we have tried to argue why they will increasingly penetrate both the desktop and the cost-conscious farming environments of High Energy Physics. In this context it is comforting to see that PCs with either Linux or Windows/NT are able to appeal to the users who will (be forced to) choose a working environment based on their existing skills. In particular we believe that the productivity gains of a good Integrated Development Environment on a modern (but still cheap) PC is a must for the large software development tasks required for LHC.
Windows/NT has the advantage over a UNIX-based system that it can be seen as an All-In-One platform for HEP, covering everything from desktops for secretaries and engineers, to development stations or farm engines for physicists. It has the disadvantage, however, of not benefiting from a large community of HEP developers that are familiar with the WIN32 APIs. This fact will inevitably lead to the search for emulation or high-level compatibility libraries that can ease the transition of code between UNIX and NT. Over time, this problem should go away and it can even be envisaged that HEP development will first be done under Windows/NT with subsequent porting to UNIX.
In either case, HEP must take out insurance against changes in the computing market by having all programs capable of running on multiple platforms with multiple operating systems.

7 References

[1] "Benchmarking Computers for HEP",CERN/CN report 92/13, E.McIntosh,.

[2] "PC as Physics Computer for LHC ?", CERN/CN report 95/13, S.Jarp et al.

[3] "Porting the Zeus Reconstruction code to PC" Private communications, R.Yaari, 1996.

[4] "Examining WIN32 C++ compilers" R.van der Wal, DDJ, March 97.