Preprocessor Based Implementation of the Versatile Advection Code for Workstations, Vector and Parallel Computers

Gábor Tóth

Department of Atomic Physics, Eötvös University,
Puskin u. 5-7, Budapest 1088, Hungary,
gtoth@hercules.elte.hu, http://www.fys.ruu.nl/~toth

Abstract. The Versatile Advection Code is a single scientific software package designed and implemented to solve various hydrodynamic and magnetohydrodynamic problems typical of astrophysical research. It runs on workstations, and on vector and parallel supercomputers as well. The versatility for applications is ensured by the Loop Annotation Syntax preprocessor and the modular design of the software, while portability to different hardware platforms is achieved by the preprocessors that can translate the code from Fortran 90 both to High Performance Fortran and Fortran 77. Performance results are presented for several platforms.

1 Introduction

The Versatile Advection Code (VAC) [1, 2] has been developed since 1994 as a general purpose tool for hydrodynamic and magnetohydrodynamic astrophysical applications. VAC uses various shock capturing numerical methods [3], explicit, semi-implicit, or fully implicit time stepping [4, 5] on 1, 2, or 3 dimensional finite volume grids. The software package is complete with 120 pages of manual written in hypertext, a user interface based on web browsers, and visualization macros for the most popular visualization softwares. The ever growing number of users and applications proves that the concept of a single well designed general purpose scientific software package is a good alternative to the typical specialized scientific codes.

The most original software solution in VAC is the Loop Annotation Syntax (LASY) [6], which was developed to provide a compact notation for expressions occuring in a multidimensional hydrodynamic code independent of the number of represented spatial dimensions. The other important feature is the modular design, which allows VAC to solve different equations with different methods, and lets the user add extra terms in the equations, define special initial and boundary conditions, or specify non-default input/output data format by writing a few well specified subroutines.

VAC is designed from the beginning to run on workstations, where most scientists do their simulations, and on vector and parallel supercomputers, required for big 2D and 3D simulations, as well. The source code, after it is translated from the LASY notation, uses Fortran 90 (F90) array syntax and High Performance Fortran (HPF) style `FORALL` statements for all the expressions that operate on the whole computational grid. Thus it is easy to add HPF compiler directives in an automated fashion and run the code on a parallel machine under HPF. It is also trivial to translate the `FORALL` statements back to ordinary `DO` loops for a Fortran 90 compiler on a non-parallel machine.

Although Fortran 90 is becoming available on most scientific computing facilities, it is still necessary to be able to translate the source code to Fortran 77 (F77). A simple translator program is implemented to carry out this task for the limited number of language constructs that are used from the rich Fortran 90 language. Not using all the features of F90 is a restriction for the developer, but it is beneficial for the users, who are more familiar with the simpler F77 language, and for the compilers, which usually do a better job on simpler program constructs.

2 Preprocessors

The use of the preprocessors can be best demonstrated on a small piece of code. The purpose of the `gradient` subroutine is simple: calculate the gradient `gradq` of the quantity `q` in direction `idir` within a rectangle, defined by `ix^L` indices. From the actual, more general, subroutine used in VAC, I extracted the part which is valid for Cartesian grids and uses central differences. The subroutine is shown in Figure 1.

```
subroutine gradient(q,ix^L,idir,gradq)

include 'vacdef.f90'
double precision:: q(ixG^T),gradq(ixG^T)
integer:: ix^L,idir,jx^L,hx^L

!SHIFT
jx^L=ix^L+kr(idir,^D);
!SHIFT MORE
hx^L=ix^L-kr(idir,^D);
!SHIFT BEGIN
gradq(ix^S)=0.5D0*(q(jx^S)-q(hx^S))/dx(ix^S,idir)
!SHIFT END

return
end
```

Fig. 1. Example source code with LASY.

The included `vacdef.f90` file declares the global parameters and variables. The array dimensions `ixG^T`, the grid spacing `dx(ixG^T,ndim)`, and the Kronecker delta array `kr(3,3)`, which is used to shift indices in a certain direction, are all declared and initialized before this subroutine is called. The meaning of the LASY patterns starting with the special character ^ is briefly the following: ^D stands for dimensions, ^L for limits, ^S for array segments, and ^T for the total size of arrays. The VAC Preprocessor (VACPP) substitutes the patterns with substitute strings, whose number depends on the number of spatial dimensions, which is a parameter for VACPP. The preprocessor not only replaces the patterns with their substitute strings, but it also repeats the source code attached to the pattern, and the repetitions are separated appropriately. The detailed rules of LASY are described in [6], here I simply show the code translated to 2 dimensions in Figure 2.

```
subroutine gradient(q,ixmin1,ixmin2,ixmax1,ixmax2,idir,gradq)

include 'vacdef.f90'
integer:: ixmin1,ixmin2,ixmax1,ixmax2,idir,&
   jxmin1,jxmin2,jxmax1,jxmax2,hxmin1,hxmin2,hxmax1,hxmax2
double precision:: q(ixGlo1:ixGhi1,ixGlo2:ixGhi2),&
   gradq(ixGlo1:ixGhi1,ixGlo2:ixGhi2)

!SHIFT
jxmin1=ixmin1+kr(idir,1);jxmin2=ixmin2+kr(idir,2);
jxmax1=ixmax1+kr(idir,1);jxmax2=ixmax2+kr(idir,2);
!SHIFT MORE
hxmin1=ixmin1-kr(idir,1);hxmin2=ixmin2-kr(idir,2);
hxmax1=ixmax1-kr(idir,1);hxmax2=ixmax2-kr(idir,2);
!SHIFT BEGIN
gradq(ixmin1:ixmax1,ixmin2:ixmax2)=0.5D0*&
   (q(jxmin1:jxmax1,jxmin2:jxmax2)&
   -q(hxmin1:hxmax1,hxmin2:hxmax2))&
   /dx(ixmin1:ixmax1,ixmin2:ixmax2,idir)
!SHIFT END

return
end
```

Fig. 2. Source code translated to Fortran 90 for 2 spatial dimensions.

It is quite easy to imagine what the 1 or 3 dimensional versions would look like. Clearly, the LASY notation is not only more general, but also more compact than the translated F90 source code. The VACPP preprocessor is implemented as the *vacpp.pl* Perl script.

In case the user has no F90 compiler available, the Fortran 90 source is further translated to Fortran 77 by the *f90tof77* Perl script. The translation changes the free source format to fixed one, and replaces the array syntax by do loops. The *f90tof77* script can also deal with the differences between F90

```
      subroutine gradient(q,ixmin1,ixmin2,ixmax1,ixmax2,idir,gradq)

      include 'vacdef.f'
      integer ixmin1,ixmin2,ixmax1,ixmax2,idir,
     &   jxmin1,jxmin2,jxmax1,jxmax2,hxmin1,hxmin2,hxmax1,hxmax2
      double precision q(ixGlo1:ixGhi1,ixGlo2:ixGhi2),
     &   gradq(ixGlo1:ixGhi1,ixGlo2:ixGhi2)

*SHIFT
      jxmin1=ixmin1+kr(idir,1)
      jxmin2=ixmin2+kr(idir,2)
      jxmax1=ixmax1+kr(idir,1)
      jxmax2=ixmax2+kr(idir,2)
*SHIFT MORE
      hxmin1=ixmin1-kr(idir,1)
      hxmin2=ixmin2-kr(idir,2)
      hxmax1=ixmax1-kr(idir,1)
      hxmax2=ixmax2-kr(idir,2)
*SHIFT BEGIN
      do ix_2=ixmin2,ixmax2
      do ix_1=ixmin1,ixmax1
         gradq(ix_1,ix_2)=0.5D0*
     &      (q(ix_1+(jxmin1-ixmin1),ix_2+(jxmin2-ixmin2))
     &      -q(ix_1+(hxmin1-ixmin1),ix_2+(hxmin2-ixmin2)))
     &      /dx(ix_1,ix_2,idir)
      enddo
      enddo
*SHIFT END

      return
      end
```

Fig. 3. Source code further translated to Fortran 77.

and F77 regarding the variable declaration, and it can translate some functions like `sum, product, maxval, minval, any, all`, which operate on arrays and return scalars. The `where, forall, case` constructs can also be translated. Other features of Fortran 90, like dynamic allocation, modules, array valued functions, pointers, structures, etc. are not used in VAC, and cannot be translated by *f90tof77*, which is a short and simple program. The `gradient` subroutine in 2 dimensions and in F77 is shown in Figure 3. The loop variables `ix_1, ix_2` are declared in the included file.

```
subroutine gradient(q,ixmin1,ixmin2,ixmax1,ixmax2,idir,gradq)

include 'vacdef.hpf'
integer:: ixmin1,ixmin2,ixmax1,ixmax2,idir,&
   jxmin1,jxmin2,jxmax1,jxmax2,hxmin1,hxmin2,hxmax1,hxmax2
double precision:: q(ixGlo1:ixGhi1,ixGlo2:ixGhi2),&
   gradq(ixGlo1:ixGhi1,ixGlo2:ixGhi2)
!HPF$ DISTRIBUTE q(BLOCK,*) ONTO PP
!HPF$ DISTRIBUTE gradq(BLOCK,*) ONTO PP

!SHIFT
jxmin1=ixmin1+kr(idir,1);jxmin2=ixmin2+kr(idir,2);
jxmax1=ixmax1+kr(idir,1);jxmax2=ixmax2+kr(idir,2);
!SHIFT MORE
hxmin1=ixmin1-kr(idir,1);hxmin2=ixmin2-kr(idir,2);
hxmax1=ixmax1-kr(idir,1);hxmax2=ixmax2-kr(idir,2);
!SHIFT BEGIN
IF (hxmin1==ixmin1-1.and.hxmin2==ixmin2.and.&
    jxmin1==ixmin1+1.and.jxmin2==ixmin2) THEN
   gradq(ixmin1:ixmax1,ixmin2:ixmax2)=0.5D0*&
      (q(ixmin1+1:ixmax1+1,ixmin2:ixmax2)&
      -q(ixmin1-1:ixmax1-1,ixmin2:ixmax2))&
      /dx(ixmin1:ixmax1,ixmin2:ixmax2,idir)
ELSE IF(hxmin1==ixmin1.and.hxmin2==ixmin2-1.and.&
        jxmin1==ixmin1.and.jxmin2==ixmin2+1) THEN
   gradq(ixmin1:ixmax1,ixmin2:ixmax2)=0.5D0*&
      (q(ixmin1:ixmax1,ixmin2+1:ixmax2+1)&
      -q(ixmin1:ixmax1,ixmin2-1:ixmax2-1))&
      /dx(ixmin1:ixmax1,ixmin2:ixmax2,idir)
ELSE
   stop 'SHIFT did not optimize!'
ENDIF
!SHIFT END

return
end
```

Fig. 4. Source code with HPF directives and optimized index shifts.

The *f90tohpf* script inserts the HPF directives into the Fortran 90 source code automatically. All arrays defined on the full grid are declared with the `ixGlo1:ixGhi1,...` index limits, and they can be distributed among the processors according to the parameters given to *f90tohpf*. On different parallel architectures and/or for different problem sizes, different distributions may be optimal. The automatic insertion of the directives makes it extremely simple to, e.g., change a (BLOCK,BLOCK) distribution to (BLOCK,*) or (*,BLOCK).

Unfortunately, HPF compilers are not as mature as F77 or F90 compilers. Several HPF compiler bugs were found while VAC was tested on parallel computers. Due to the simplicity of the source code, there were relatively few problems, and they could be avoided relatively easily. Even if the code compiles and runs correctly, the performance can be very poor if the HPF compiler does not recognize the simple shift operations in the `gradient` subroutine and elsewhere in the source. The general global communication is much slower than the fast specialized shifts, which are supported by the hardware and the communication libraries of most parallel computers. To help the compiler, the VAC preprocessor can replace the general shift statement marked with the `!SHIFT` comments, with shifts in specific directions placed in the appropriate branches of an `if, else if` construct. The resulting code, shown in Figure 4, is longer and more difficult to read, but it usually compiles to a faster code under HPF. The physical layout of processors `PP` is defined in the include file. When only one spatial dimension is distributed, one can use the HPF directive `!HPF$ PROCESSORS PP(NUMBER_OF_PROCESSORS())`.

The code can also be translated to Connection Machine Fortran (CMFortran) with the `f90tocmf` script. Unfortunately the CM Fortran compiler recognizes index shifts for a very limited type of syntax, thus communication is not optimal without rewriting the critical shifts by hand. In principle, one could automate this optimization, but, since CM Fortran is disappearing from the scene, there is little motivation to write the necessary Perl script.

3 Results and Conclusions

VAC is being used by approximately 25 researchers, mostly astrophysicists. Most applications are hydrodynamic and magnetohydrodynamic simulations, but VAC is also used as a test suite for different numerical methods. Most users have access to powerful workstations, thus the code has been tested and used on DEC, SUN, IBM, SGI, HP workstations, and even on Pentium PC-s under LINUX.

Due to the simplicity of the loops, which is implied by the F90 array syntax, the code vectorizes very well. On a single node of the traditional vector supercomputer Cray C90, VAC runs about 23 times faster than on a DEC Alpha/400 workstation, while the ratio is 4.2 for the J90. These measurements were done for a specific problem [7], but the speed ratios are typical for all timings tried so far.

VAC has also been tested on the IBM SP, Cray T3E, Cray T3D, and Connection Machine 5 (CM5) parallel machines, and on a cluster of workstations, under different HPF compilers [8, 7]. The scaling is close to linear up to 8 processors on the IBM SP and on the Cray T3E for a rather moderate and *fixed* problem size, which proves that good scaling is possible under HPF even for a code as complex as VAC. The single node performance is a factor of 5.2 and 1.7 improvement relative to the DEC Alpha/400 workstation for the SP and the T3E machines, respectively. On a 16-node CM5, after optimizing the array shift operations by hand, the code runs about 15 times faster than on the DEC Alpha. VAC was tested on a cluster of workstations as well. The code compiled and ran successfully, but the multiuser environment did not allow for meaningful timing.

The Versatile Advection Code proves that it is possible to write one source code for several different applications and computer platforms with the aid of simple but powerful preprocessor and translator programs. All the preprocessor programs, `vacpp.pl, f90tof77, f90tohpf, f90tocmf, forall2do`, are implemented in Perl, which is a free software, and is installed on almost all scientific computers. Actually, the preprocessing step and the final compilation can be done on different computers if necessary.

Currently we are working on the HPF compatible implementation of the implicit time stepping module. As a first step the Poisson solver using Conjugate Gradient type iterative schemes (CG and BiCGSTAB), originally implemented in F77, has been rewritten to the LASY notation and now it runs successfully on parallel machines with HPF. The next step involves rewriting and testing the preconditioner [9] for the block penta- and heptadiagonal Jacobian matrices that arise in implicit time stepping schemes.

Acknowledgement. This work was performed as part of the project on 'Parallel Computational Magneto-Fluid Dynamics', funded by the Dutch Scientific Research Foundation (NWO) Priority Program on Massively Parallel Computing. It was sponsored by the Dutch National Computing Facilities Foundation (NCF) for the use of supercomputer facilities. The author receives a postdoctoral fellowship (D 25519) from the Hungarian Science Foundation (OTKA), and is supported by the OTKA grant F 017313. Collaboration on testing the code with R. Keppens, P. Meyer, and E. van der Zalm is gratefully acknowledged. The author also thanks the Astronomical Institute at Utrecht for its hospitality.

References

1. Tóth, G.: A general code for modeling MHD flows on parallel computers: Versatile Advection Code, Astrophys. Lett. & Comm. **34** (1996) 245–250
2. Tóth G.: Versatile Advection Code, in *Proceedings of High Performance Computing and Networking Europe 1997*, Lecture Notes in Computer Science, **1225**, edited by B. Hertzberger and P. Sloot (Springer-Verlag, 1997), p. 253–262
3. Tóth, G., Odstrčil, D.: Comparison of some Flux Corrected Transport and Total Variation Diminishing Numerical Schemes for Hydrodynamic and Magnetohydrodynamic Problems. J. Comput. Phys. **128** (1996) 82–100
4. Keppens, R., Tóth, G., Botchev, M. A., van der Ploeg, A.: Implicit and Semi-Implicit Schemes in the Versatile Advection Code: algorithms, submitted for publication to the Int. J. Num. Meth. in Fluids (1997)

5. Tóth, G., Keppens, R., Botchev, M. A.: Implicit and semi-implicit schemes in the Versatile Advection Code: numerical tests, Astron. & Astroph. **332** (1998) 1159-1170
6. Tóth, G.: The LASY Preprocessor and its Application to General Multi-Dimensional Codes, J. Comput. Phys. **138** (1997) 981-990
7. Tóth G., Keppens R.: Comparison of Different Computer Platforms for Running the Versatile Advection Code, in *Proceedings of High Performance Computing and Networking Europe 1998*, Lecture Notes in Computer Science, **1401**, edited by P. Sloot, M. Bubak, and B. Hertzberger (Springer-Verlag, 1998), p. 368-376
8. Keppens, R. and Tóth, G.: Simulating Magnetized Plasma with the Versatile Advection Code, in this volume of proceedings (1998)
9. van der Ploeg, A., Keppens, R., Tóth, G.: Block Incomplete LU-preconditioners for Implicit Solution of Advection Dominated Problems, in *Proceedings of High Performance Computing and Networking Europe 1997*, Lecture Notes in Computer Science, **1225**, edited by B. Hertzberger and P. Sloot (Springer-Verlag, 1997), p. 421–430