GFortran: A case study compiling a 1,000,000+ line Numerical Weather Forecasting System

Toon Moene
A GNU Fortran Maintainer
toon@moene.indiv.nluug.nl

May 8, 2005

Abstract

Gfortran is the Fortran (95) front end in the GNU Compiler Collection. It is new and untested in the Real World. This paper discusses how well the new entry to the Collection fares in terms of compiling HIRLAM, a Limited Area Numerical Weather Prediction system. HIRLAM is an old (started in 1985), continuously evolving project (see: http://hirlam.knmi.nl).

In addition, the effectiveness of optimizations implemented in the new middle end and the vectorization subclass of them (on a powerpc-unknown-linux-gnu target) with respect to this kind of code are investigated.

1 Introduction

HIRLAM (HIgh Resolution Limited Area Model) is a collaborative effort by the National Weather Institutes of Denmark, Finland, Iceland, Ireland, The Netherlands, Norway, Spain, and Sweden.

The project started in 1985 to provide local, short range (i.e., up to 48 hours) automatic weather forecasts for the Nordic countries. Others joined during the late 80's, early 90's.

In the 20 years since its inception, the code has grown to about 1.2 million lines of Fortran, tens of thousands of lines of C and an elaborate script system (both Bourne Shell and Perl) to run it in both experimental setup as well as operationally (i.e., 365 x 24).

The code runs on traditional vector supercomputers, shared memory multiprocessor machines and distributed memory architectures. Small (in area and/or grid) forecasts can be done on personal computers - even laptops.

2 "Weather Prediction by Numerical Process"

2.1 Physics

Physically, weather forecasting is the problem of tracking all processes on relevant scales in the atmosphere, trying to determine an "initial state" at a certain time and performing the forward extension of physical laws governing the atmospheric evolution, constrained by the boundary conditions (soil, top-of-atmosphere).

The physical laws involved are:

- Conservation of mass
- Conservation of energy
- Conservation of momentum
- Release and consumption of latent heat due to changes between the three phases of water in the atmosphere

For the purpose of weather forecasting, the atmosphere can be approximated by an ideal gas with atmospheric water being tracked in its three phases.

Note that the concept of the "initial state" of the atmosphere is an artifact one needs for weather forecasting - in reality, the atmosphere is in constant motion, where every "state" is a consequence of change from a previous "state".

2.2 Mathematics

Mathematically, the problem falls apart into two:

1. Determine the initial state of the atmosphere at a given time on the basis of a limited number of different observations of differing quality performed at that time.
2. Perform the integration of the coupled partial differential equations discribing the physical laws and constrained by the atmosphere's boundary conditions.

The mathematical problem is complicated by the fact that the physical laws are to be described on a rotating sphere.

2.3 Numerical Weather Forecasting

The mathematical problem posed above cannot be solved analytically.

2.3.1 Forecasting

As the integration of the coupled partial differential equations doesn't have an analytical solution, only numerical integration can be used.

To perform the numerical integration, the atmosphere is modeled using a three dimensional array of boxes, each of which represents a part of the atmosphere (we call this the "model atmosphere").

The integration of the coupled partial differential equations proceeds by accounting for air properties changing in a particular box due to advection of properties from neighbouring boxes (i.e., the temperature in a particular box can change due to inflow of air of a different temperature from an adjacent box). The differential equations are approximated by using finite differences.

A second effect to be accounted for is the processes occurring at subgrid scale. Radiation from the Sun can, or cannot reach the ground due to clouds; Radiation welling up from the Earth's surface can reach outer space (and be lost), or not, due to clouds.

2.3.2 Determining the Initial State

To start a weather forecast, we need a clear definition of the initial state of the (model) atmosphere at a certain time.

The deterimination of the initial state is complicated by the fact that the number and type of observations do not completely determine the model atmosphere - the problem of determining the initial state is "underdetermined".

The solution to this limitation is to find the best fit between a previously computed short range forecast (typically 3 or 6 hours in advance, called "first guess") and the observations.

This best fit is computed by minimizing a "cost function" describing the costs of deviating from the first guess and the observations, weighted by the "quality" of both (i.e., their error characteristics).

3 The Code

About three quarters of the code is dedicated to determining the initial state of the model atmosphere from which to start forecasting.

The rest is necessary for performing the actual forecast.

All of the code is organized in libraries that pertain to either the "initial state" or "forecasting" part of the problem. Some utility libraries are common to both problems.

An executable is formed by writing the following Fortran code:

```
      CALL SUBPROG
      END
```

where SUBPROG is the main program of that executable, written as a subroutine, and linked with the relevant libraries.

The "run schedule" is controlled by a Perl script that interprets a "schedule" file, spawning Fortran programs when needed.

4 The Compiler

To study the behaviour of gfortran compiling this code, the following version of the compiler was used:

```
$ /usr/rel/bin/gfortran -v
Using built-in specs.
Target: powerpc-unknown-linux-gnu
Configured with: ../gcc/configure
           --prefix=/usr/rel
           --disable-nls
           --disable-multilib
           --enable-languages=f95
Thread model: posix
gcc version 4.0.1 20050501
    (prerelease)
```

5 Errors Preventing HIRLAM From Being Compiled

5.1 Programmer Errors Caught

None anymore. The author presented three programmer errors detected by gfortran at the HIRLAM All Staff Meeting 14-16 March 2005, but they have been repaired.

Still, the compiler is being used to hunt down programmer errors. In the author's Institute it is installed for the express purpose of finding programmer errors where the "officially blessed", proprietary, compiler leaves much to be desired.

5.2 Problem Reports Encountered With This Effort

5.2.1 Strings

One of the Fortran codes drew the following ire:

```
ONETWO_pp.f: In function 'onetwo':
ONETWO_pp.f:89: internal compiler
  error:
  in gfc_conv_string_parameter,
  at fortran/trans-expr.c:2011
```

The offending code looks like this:

```
      do 7 j=1,2
         imatch(j)=1
 7    continue
```

where imatch is an integer array. Probably the line number is off in the error report. This is PR Fortran/18283.

5.2.2 Automatic Arrays

Here is another one:

```
getgrp_pp.f: In function 'getgrp':
getgrp_pp.f:17: internal compiler
  error: in
  gfc_trans_auto_array_allocation
  at fortran/trans-array.c:3036
```

The source looks as follows:

```
      integer cgroups(6,maxgrp)
```

where maxgrp is a dummy argument and cgroups is not. This is PR Fortran/21034.

5.3 Extensions to be supported

5.3.1 LOC

The following error:

```
 In file gc_com_pp.f:2149

      INTRINSIC LOC
                 1
Error: Intrinsic at (1) does not
       exist
 In file gc_com_pp.f:2153

      NBYTES = (LOC(LAST) -
                LOC(FIRST))
                 1
Error: Function 'loc' at (1)
       has no implicit type
```

is due to the fact that the Fortran code has to interface to the C-implemented M(essage) P(assing) I(nterface) definition, which is based on C concepts. LOC is the function that returns the adress of its argument. The GNU Fortran team probably has to implement this extension.

5.3.2 FLUSH

```
 In file PFLUSH_pp.f:25

      call flush(kunit, iostat)

Error: Too many arguments in
       call to 'flush' at (1)
```

Flush is an extension - the GNU Fortran team has to determine which "fashion" of the extension it wants to support.

6 What Works

What works is, well, compiling the other 1.2 million lines of code.

6.1 Typical Programming Constructs

Characteristic for the kind of code in HIRLAM is the SPEC mgrid routine RESID - here slightly altered for clarity:

```
      SUBROUTINE RESID(U,V,R,N,A)
      INTEGER N
      REAL*8 U(N,N,N),
     ,       V(N,N,N),
     ,       R(N,N,N), A(0:3)
      INTEGER I3, I2, I1
      DO 600 I3=2,N-1
      DO 600 I2=2,N-1
      DO 600 I1=2,N-1
 600  R(I1,I2,I3)=V(I1,I2,I3)
     - -A(0)*( U(I1,  I2,  I3  ))
     - -A(1)*( U(I1-1,I2,  I3  )
     +       + U(I1+1,I2,  I3  )
     +       + U(I1,  I2-1,I3  )
     +       + U(I1,  I2+1,I3  )
     +       + U(I1,  I2,  I3-1)
     +       + U(I1,  I2,  I3+1))
     - -A(2)*( U(I1-1,I2-1,I3  )
     +       + U(I1+1,I2-1,I3  )
     +       + U(I1-1,I2+1,I3  )
     +       + U(I1+1,I2+1,I3  )
     +       + U(I1,  I2-1,I3-1)
     +       + U(I1,  I2+1,I3-1)
     +       + U(I1,  I2-1,I3+1)
     +       + U(I1,  I2+1,I3+1)
     +       + U(I1-1,I2,  I3-1)
     +       + U(I1-1,I2,  I3+1)
     +       + U(I1+1,I2,  I3-1)
     +       + U(I1+1,I2,  I3+1))
     - -A(3)*( U(I1-1,I2-1,I3-1)
     +       + U(I1+1,I2-1,I3-1)
     +       + U(I1-1,I2+1,I3-1)
     +       + U(I1+1,I2+1,I3-1)
     +       + U(I1-1,I2-1,I3+1)
     +       + U(I1+1,I2-1,I3+1)
     +       + U(I1-1,I2+1,I3+1)
     +       + U(I1+1,I2+1,I3+1))
      END
```

Although one will not find this routine anywhere within the HIRLAM code verbatim, its use of arrays is very comparable to that of a finite difference approximation to continuous partial differential equations, where the difference of quantity A between neighbours I-1, I and I+1 is the measure of change.

By using RESID for the analysis of gfortran we achieve two goals:

1. RESID is a well studied routine by the compiler community due to the fact that it is part of SPEC2000.

2. It is much smaller than the smallest routine from HIRLAM (which contains about 350 lines - before formatting it in a column) without sacrificing on computational complexity.

6.2 Optimization

When optimizing this type of code, the number of integer/address registers necessary is important. In fact, it is far more important than the number of floating point registers; no shortage is apparent for them. If a processor doesn't have enough integer/address registers, the reload pass of the compiler has to generate code to reload them from (stack) memory.

How many integer/address registers do we need in routine RESID ?

If we assume a machine with "register+offset" addressing, we need the following registers in the inner loop:

- 1 for all A elements
- 1 for R
- 1 for V

- 1 for each of the U(..,I2[+/-1],I3[+/-1]), because N is not a constant, which means the offsets into the sub-arrays are not constant (totalling 9 registers)

- 1 for the loop count

Or 13 in total.

It is the function of the induction variable strength reduction and elimination pass to remove excess integer computation here.

In fact, it is rather amazing that resid.f.06.loop contains diagnostics like: "giv of insn XXX not worth while" (giv stands for "general induction variable"). In a routine like this, where all induction variables are based on the adresses of dummy arguments, reducing all givs leads to the minimum number of registers used (there may be combinations of reduced and non-reduced givs that lead to the same minimum, but reducing all is certainly correct and optimal).

6.3 Vectorization

The inner loop in RESID cannot be vectorized on PowerPC, because the operands of the computations are all 64-bit floating point variables (REAL*8).

If we change their declaration to REAL*4 (32-bit floating point variables), the inner loop is vectorized when using the following command line:

```
/usr/rel/bin/gfortran
    -O2 -ftree-vectorize
    -ftree-vectorizer-verbose=9
    -maltivec -da -S resid.f
```

This is significant to us, because most of the computations in the forecasting system can be performed with 32-bit floating point variables.

However, there still is a problem. The vectorizer has to assume it knows nothing about the alignment of the arrays A, R, U and V and hence has to perform a lot of code duplication for peeling unaligned loads and stores from the inner loop.

This is tragic, because in a normal Fortran program, the arrays would either be static or automatic arrays in a routine higher up in the call tree or the main program (and hence could be aligned by the compiler), or be ALLOCATEd there (and hence be aligned by the run time library).

Perhaps we need an "I know the arrays in this routine are suitably aligned" flag.

7 Conclusions

HIRLAM cannot be completely compiled by gfortran - yet. However, it is close, and some small amount of programming will get it there.