GANDF: A GCC-based ANDF Translator

Richard L. Ford

Open Software Foundation Research Institute

January 26, 1993

GANDF implements a family of translators for ANDF by interfacing ANDF to the gcc back-end.

1. Introduction

ANDF is an architecture- and language- neutral distribution format being developed by OSF and other collaborators around the world. It is based on the TDF technology provided by the Defence Research Agency (DRA) of the UK Ministry of Defense.

GANDF is an experimental ANDF translator[1] being implemented at the Open Software Foundation (OSF) Research Institute (RI), based on the back-end technology of the Gnu C Compiler(gcc), produced by the Free Software Foundation, along with some support routines from the DRA technology.

This distinguishes GANDF from the other existing ANDF translators, which are all either written by DRA, or directly derived from the DRA installers. Each of the existing translators is for a single target, whereas GANDF will be targeted to any of the targets supported by gcc.

1. This paper follows the following terminology used by DRA. A *producer* is all of the software that is used to produce the ANDF form of an application. The primary component of a producer is the *compiler*, which does the actual translation of source code into ANDF. An *installer* is all of the software used to install an application on a target. The main component of an installer is the *translator*, which does the actual translation of target-dependent ANDF into machine code. When not used in the ANDF context, the word *compiler* will have the more usual meaning.

2. Objectives

Preserve Investment in System Vendor Compilers

Most computer system vendors have a large investment in compilers. Much of this is in the middle and back-ends of the compilers where sophisticated techniques are employed to get highly optimized code. This makes the system vendor’s machine look good. If ANDF is to be successful, there must be ANDF installers which are competitive with such system vendor compilers in reliability and code quality. In addition, many customers of ANDF applications would feel more confident if they used an installer that was supported by their system vendor.

One way to achieve these goals would be to interface ANDF to the system vendor’s compilation system. One objective of the GANDF project is to test the feasibility of interfacing ANDF to system compilers, by using the Gnu C Compiler (gcc) as a readily accessible test case.

Initially we only address the problem of interfacing ANDF to the back-end. ANDF is more powerful than many existing intermediate languages. We point out some problems that arise when trying to interface ANDF to a compiler back-end, and ways that these problems can be solved.

Another aspect of preserving the system vendor’s investment in compiler technology has to do with the front-ends. Many vendors have their own dialects of programming languages that they would want to maintain, even if they are moving to the ANDF technology. Their front-ends may provide special information to their back-ends that allow for special optimizations. So another question to ask is, how easy would it be to change a compiler front-end to emit ANDF? That topic is not currently being addressed by the GANDF project, but is a possible follow-on project. We could take the gcc front-ends (for C, C++, and F77) and try to make them produce ANDF instead of the usual gcc IL.

Increased Availability

In addition to providing a case study on interfacing of ANDF to a compiler back-end, GANDF has the potential for being useful as a means of quickly producing new installers for ANDF. GCC currently supports[2] (in some cases experimentally) the following cpus:

alpha, a29k, arm, cN, hppa1.0, hppa1.1, i386, i860, i960, m68000, m68k, m88k, mips, ns32k, romp, rs6000, sparc, vax, we32k.

and the following operating system types:

bsd, sysv, mach, minix, genix, ultrix, vms, sco, isc, aix, sunos, hpux, unos, luna, dgux, newsos, osfrose, osf, dynix, aos, ctix, msdos (with dos extender), and windows nt.

In addition, gcc is designed to make it relatively easy to add a new target. The GANDF code will be target-independent in the same way that the gcc code is target-independent, namely, where target dependence is needed, it is parameterized and the target-dependent information provided as part of the machine description. The target parameterization already available with gcc will satisfy most of the needs of GANDF. Thus once one GANDF port is finished, there will be very little extra work needed to port to additional platforms.

The gcc compiler is a widely used and respected compiler. It is hoped that having ANDF installers based on the same technology will help to increase the ANDF momentum.

Educational/Technology Transfer

A further objective of GANDF is educational. Having a freely available installer for ANDF will help other installer writer's learn of possible techniques for implementing ANDF. In addition, some system vendors might choose to use GANDF to provide installers for their systems, rather than using the DRA installer technology (some vendors use gcc as their supported compiler). Other companies may choose to provide commercial support for GANDF.

3. Role of GANDF

To understand the role of GANDF, one must understand the stages in production of software using ANDF. There are the following steps (assuming C is the source language):

2. according to documentation in gcc release 2.2.2. I've added the entries for msdos and nt. Support for these is not currently completely supplied as part of the FSF distribution.

- Driver program (tcc or gcc) is invoked to drive and coordinate the following steps, as needed.
- Source code (.c) file is preprocessed to .i file.
- Preprocessed file (.i) is compiled to target independent ANDF file (.j)
- Target Independent ANDF files (.j) are linked together with token libraries (.tl) by the token linker (tld) to produce a target-dependent ANDF file (.t)
- A *translator* (e.g. trans386) reads a target dependent ANDF file (.t) and produces machine code, either assembly code (.s) or to direct object code (.o), as might be done if one were concerned with compile-time performance.
- Assembly language files (.s) are assembled into object files (.o)
- Object files (.o) are linked with system libraries, using the system linker, to produce an executable.

There are other possible scenarios. The ANDF producer actually combines the preprocessing and production of .j files into one step. There could be ANDF to ANDF optimizers. One could imagine *translator*s that combine the functions of the token binder, translator, assembler and system linker into a single program (e.g. to do really global optimization).

The role of GANDF in this scheme is that *gandfc*[3] is a translator from target dependent ANDF files (.t) to assembly language, analogous to trans386 in the DRA technology, or cc1 in the gcc technology (cc1 is the program that take preprocessor output (.i) and compiles it into assembly code(.s))

There is one complication in the role of gandfc. The DRA translators expect one style of command line options from the DRA driver, tcc. On the other hand, gandfc, being based on gcc technology, expects the kind of options that cc1 would get from the gcc driver, gcc. For that reason gandfc would usually be invoked via gcc[4]. In order to make it possible to invoke gandfc from tcc, an additional shell script is used which translates command line options supplied by tcc to the equivalent option of gcc.

3. I use GANDF as a generic name for the project to develope ANDF tools based on the gcc technology. Currently gandfc is the only tool in the GANDF project.A gcc-based front-end targeted to ANDF would be another possibility.

4. Because of the way that the driver program, gcc, is table driven, using a file called *specs*, it was possible to make gcc drive gandfc without changing gcc itself. All that was required was to edit the *specs* file.

Note that gandfc is only one component of an installer. Assemblers and linkers are needed, but these would usually already be available, or the gnu assembler and linker could be used. The other required piece is the token linker. In addition, each installer needs a way of creating target-dependent token libraries.

4. Organization of GANDF

Currently GANDF (or more specifically, gandfc) is composed of three components.

GCC Component

The gcc component consists of all of the language-independent code of gcc, as well as some of the C-specific code. A few modifications have had to be made to this code for GANDF. This component includes the overall compiler executive, as well as support for the expression tree and RTL (register transfer list) data structures and optimization and code generation routines.

DRA Component

This consists of routines to read and decode the ANDF into an internal format, expand tokens, and perform certain other ANDF transformations (e.g. constant folding). Some of this processing is probably unnecessary for GANDF, since the equivalent transformations will be done by the gcc back-end. On the other hand, it may be that the combination of optimizations done by the DRA code and gcc code will be better than either one individually. That is one of the things that the GANDF experiment will determine. This component is currently available only under license. Some of this code also has had to be customized for GANDF.

OSF Component

This is the heart of GANDF. It consists of routines which translate the internal ANDF form into appropriate gcc structures. Further details about this component are given in the following section. This component is being written from scratch at the OSF RI.

5. ANDF Translation Notes

In this section we will discuss some of the more interesting issues involved in translating ANDF to a compiler intermediate form.

ANDF Types(Shapes)

ANDF types (shapes) are translated into the target-dependent data types used by gcc. Currently only shapes easily representable by the target architecture are supported. However, it would not be hard to extend support to arbitrarily large integers, for example, using the GNU Multiple-Precision arithmetic package.

Arithmetic and Logical Operators

Translating typical unary or binary operations, e.g. add, multiply, is straightforward as long as no special error treatment is needed.

Error treatments

ANDF provides the capability to recover from errors by taking some default action, or by jumping to a specified label. Although this concept could cover general kinds of errors, the current ANDF only uses this facility for integer and floating point overflow.

Currently gcc does not provide any capability for such error recovery, though there is some work in progress to support C++ exception handling. Note that the ANDF C producer does not use error treatments, but a C++ producer would need them.

GANDF implementation of overflow handling will have to be target dependent. A new component of the machine description will be needed to specify how to handle overflows.

For machines that trap on overflow, one strategy is to set up trap handlers as part of the ANDF run-time environment. Global variables can be used to specify what should be done if an overflow occurs.

For machines that do not trap on overflow, explicit checks for overflow must be done following each operation that is required to be checked. Note that whether a machine traps or not may depend on the type of operation. For example, some machines trap on floating point overflow, but not on integer overflow. Machines with IEEE floating point may just return infinity on overflow.

Multiple Return Types

One difficult feature of ANDF is the ability of a single procedure to return results of more than one type. This feature is similar to the Fortran feature of having functions with multiple entry-points each of which may be of a different type, but is different in that Fortran has different entry-points to distinguish between the return types. The problem is that gcc, and I expect most compiler back-ends, expect a function to only return a single data type, at least for a given entry-point. This feature is not yet implemented in GANDF, and I expect that its implementation will require some relatively low-level modifications.

One problem with this feature of returning more than one type of result is that for some data types, returning a value of that type requires cooperation from the caller, e.g., passing in the address where the result is to be placed, possibly changing the position of other parameters. There would be no problem implementing multiple return types if one did not have to conform to native calling conventions, but with that constraint I think it is difficult, and perhaps impossible on some platforms. Also, the ANDF token mechanism does not provide a way of specifying a procedure with more than one return type. My opinion is that this is one feature of ANDF which is not needed. It would be better to allow a function to return at most one type of value.

Discovering the need for changes to the ANDF specification, such as that suggested above, is one of the purposes of the GANDF experiment.

Non-local Variable Access

ANDF allows the current environment of a procedure (roughly equivalent to its stack frame pointer) to be captured and used elsewhere, to access those local variables of the procedure that have been marked as “visible”. GANDF will implement this by creating a structure to hold the visible variables. Those with non-overlapping lifetimes will be overlayed as if in a union (or optionally they could be allocated distinct space, perhaps for debugging purposes, or to ease garbage collection).

Identify Operator

The identify operator is used by ANDF to give a name to an expression which is computed only once, but used possibly repeatedly. gcc has a similar operator, save_expr. This will be used by GANDF unless the identified value is marked as “visible”, in which case it will have to have storage allocated for it as described above.

Local Label Variables

ANDF allows local label values to be stored in a variable. Though this is not allowed in C, gcc does in fact allow such label variables.

Long Jumps.

The ANDF *long_jump* operation takes as arguments an environment expression (stack frame) and local label expression. It unwinds the stack until the specified stack frame is reached and then transfers control to the associated procedure at the location specified by the local label value. gcc has built-in functions that perform the C *longjump*. These will be used to implement the ANDF *long_jump*. Further study is needed to ensure that these have the same semantics, e.g. with respect to preserving values of local variables.

Local Allocation.

The ANDF *local_alloc* operator is analogous to C’s *alloca*, which allocates space by extending the stack. gcc also has built-in support for *alloca*, so support of *local_alloc* should be straightforward. ANDF operators related to freeing up such space may be harder.

Global Scoping

In ANDF, all the top-level variables and procedures (represented by TAGs in ANDF) are visible globally. This supports mutual recursion, and allows the initial values of some variables to contain the addresses of procedures or other variables. The support routines in gcc expect variables and procedures to be declared before use. As a result, GANDF must perform an initial pass, which only looks at top level definitions and declares the top level entities. Then a complete pass actually produces code.

ANDF run-time environment,

For the most part, ANDF constructs translate straightforwardly into machine code. However some constructs, e.g. catching of overflow traps, could require a run-time environment. Also, if GANDF supports arbitrary precision integer arithmetic it will need run-time routines for that.

6. Status and Availability

Status

GANDF is still under development. There is currently support for about half of the ANDF operators, but some of the basic features, such as procedure parameters (on the callee side), are still unimplemented. So far GANDF has been successfully configured and executed for three targets:

- HP PA-RISC 1.1/HPUX
- IBM RS-6000/AIX
- DEC ALPHA/OSF1.

Most of the development is being done on an RS-6000.

Statistics

To give some idea of the size of the components of GANDF, here are the number of lines of source code, including comments, for them:

- GCC component: 432k lines, of which 285k lines are target independent code and 147k lines are target dependent (machine descriptions or configuration files).
- DRA component: 24k lines. This is just the part used by GANDF.

- OSF component: 5k lines. When GANDF is completed this is likely to be more like 20k lines.

A complete installer will also require a token binder. The current token binder, tld, is about 7k lines of source.

Availability

GANDF is currently just an experiment, and whether it will become generally available depends on the outcome of the experiment. The most valuable outputs of the GANDF experiment will be its influence on the evolution of the ANDF specification and the knowledge gained on interfacing ANDF to compiler back-ends (and possibly front-ends).

Because GANDF makes use of gcc, if it ever is distributed, it will be subject to the terms of the Gnu General Public License, meaning it will be freely available in source form. The only potential problem with this is that the DRA component is currently only available under license from DRA. However there is a possibility that the part of the DRA ANDF technology used by GANDF will be made freely available (or freely licensed) by DRA. Otherwise, that component could be rewritten based on the public ANDF specification. [5]

7. Conclusions

Final conclusions of the GANDF experiment will have to await completion of the experiment. However here are our current guesses of what the conclusions will be:

- The GANDF will provided feedback on the ANDF specification which will result in an ANDF that is more compatible with existing compiler technologies, and which will thus make it easier for system vendors to adapt their compiler's to ANDF technology.

5. I have an idea for general table-driven tools (encoders, decoders, assemblers, and pretty-printers) for handling ANDF-like data. These would use data description files written in an ANDF meta-notation. Such tools could be used to describe and process user data, thus allowing user data to be distributed in a compact and target neutral way. Such tools could be used to replace some of the DRA code.

- GANDF will be a practical and useful way of producing ANDF installers with quality characteristics similar to those of gcc. For many applications these installers will be quite adequate, but user's who want the most sophisticated optimizations may want to use installers produced by commercial compiler vendors (either system vendors or compiler companies). Some commercial companies may choose to support GANDF for their customers.
- GANDF will provide a valuable educational tool for those wishing to learn more about ANDF technology.

For further information please contact:

Richard Ford
richford@osf.org
(617) 621-7392