Flexible Optimization Control
E-mail on Function Evaluation

by Craig T. Dedo
November 11, 1998

Following is a complete copy of e-mail that I sent to Mr. James Giles last June about the current Fortran rules for optimizing function evaluations. Since this issue is part of the flexible optimization control issue, I believe that it is relevant to consideration of this issue.

[Begin e-mail]

Date:	Fri, 19 Jun 1998 15:39:55 -0500
From:	Craig T. Dedo <Craig.Dedo@mixcom.com>
Reply-To:	Craig T. Dedo <Craig.Dedo@mixcom.com>
Organization:	Elmbrook Computer Services
X-Mailer:	Mozilla 4.04 [en] (WinNT; U)
MIME-Version:	1.0
To:	James Giles <jamesgiles@worldnet.att.net>
CC:	Tony Warnock <ttw@lanl.gov>
Subject:	Function Evaluation - My Idea
Content-Type:	text/plain; charset=us-ascii

Dear Jim:

Following is my outline of my idea on solving the problem of non-deterministic evaluation of functions in statements. As I expected, my solution is different from any other that I know of.

I agree with you that this is a serious problem and needs a sensible solution. I.e., in your words, "this constitutes a lurking trap for the naive user" and "it may constitute a challenge even for considerably less naive users".

You should be aware that the members of J3 have been aware of this problem for many years. I did some research using my records from previous meetings. At Meeting 129, held May 3-7, 1994 at Incline Village, NV, Walt Brainerd gave a presentation on this very issue on Tuesday, May 4, 1994. His presentation notes were recorded as paper 94-183. At the end of his presentation, he conducted a 7-way straw vote on how we should resolve this issue. Here are the results:

0	Completely deterministic evaluation of expressions
5	Allow no "optimizations" except possibly within PURE functions
5	Restrict some of these "optimizations" to those that are resonable (perhaps only within one statement)
5	Alow all side effects and the results are processor dependent
10	Clarigy existing statements as in FIB
0	Add some more cases where side effects are not allowed
0	Prohibit side effects in functions

The reason nothing has been done to date is not that people do not consider this problem to be important. Rather, it is not obvious what it is that we need to do. I expect that it will be difficult to develop a solution that most members are willing to live with.

You need to be aware that the Fortran programming language has many and varied constituencies. Two of the most important are:

1. High-performance numeric computation (i.e., execution speed is everything).
2. Reliable results at any cost.

On this issue, these two constituencies want opposite solutions. The speed demons want to optimize away any function references that they can, so that their code will run faster. The robust results people want the opposite; they ALWAYS want to execute any and all function references that are in the code and therefore have completely deterministic code.

Considering this fact of differing constituencies, I consider all of the proposed solutions to date to be flawed in one major respect. All of them are one-size-fits-all solutions. Therefore, one or more of the major Fortran constituencies must sacrifice a major interest that they have. This is a recipe for continued stalemate.

I propose a solution based on some principles that I believe to be good language design.

1. Fortran should be user-friendly. The syntax should be easy and straightforward. There should be a minimum number of traps and gotchas.

2. Fortran should be a language which maximizes human productivity, i.e., function points (or some similar measure) per direct labor hour. It should be easy to convert an idea for a program into useable code with a minimum of hassles or fighting the quirks of the language.

3. Fortran should provide the tools that application developers need in order to get their jobs done effectively. Toward that goal, I believe that J3 should consider application developers to be mature, responsible adults. Therefore, we do not need to make extraordinary efforts to protect users from themselves.

Tony Warnock knows all of this. We have discussed this and related issues at previous J3 meetings.

Considering all of the above, I ask a simple question: Why not give everyone what they want? Why not develop some solution that allows the application developer to specify how to evaluate functions when there are situations when they could possibly be optimized away?

Let's get rid of the idea of forcing a single solution on everyone and instead put the decision making authority into the hands of the application developer. I trust the average programmer. Joe Sixpack knows better than I do what is good for Joe Sixpack.

A related point is that the same programmer may need different function evaluation rules in different programs. Or even in different parts of the SAME program.

Therefore what I propose is that we should develop a statement that gives the use the abilty to specify the evaluation rules for functions. I propose a syntax of:

EVALUATE evaluation-rule

The statement object, evaluation-rule, should have at least the values of: ALWAYS, NEVER, and SOMETIMES. Other values could be allowed as an extension to the standard.

ALWAYS would mean that the functions that could be optimized away are always executed no matter what. NEVER would mean that if a function reference can be optimized away, it will be. SOMETIMES would correspond to the current situation; a processor can decide when it will and when it will not optimize away a particular function reference. SOMETIMES is provided for backwards compatibility.

Alternatively, evaluation-rule could be an INTEGER value within a prescribed range, say 0 to 1000. One end of the range (e.g., 0) would correspond to NEVER and the other (e.g., 1000) would correspond to ALWAYS. Other INTEGER values would correspond to other function evaluation rules.

I propose that this statement would be modal, i.e., once a function evaluation rule is set, it remains in effect until another EVALUATE statement is executed. Thsi would give the application programmer maximum flexibility.

Alternatively, we may wish to restrict the statement to one evaluation rule per procedure. This could simplify implementation but at the cost of less flexibility for the programmer. It could also motivate application programmers to artificially split their procedures in order to implement different evaluation rules, with consequent inter-procedure overhead and less human-readable code.

Please let me know what you think of my idea, positive or negative.
[End of e-mail]

[End of J3 / 98-23a]