Legacy Fortran Software: Applying Syntactic Metrics to Global Climate Models

Mariano Méndez[1], Jeffrey Overbey[2], Fernando G. Tinetti[1⋆]

[1] III-LIDI, Fac. de Informática, Universidad Nacional de La Plata
50 y 120, La Plata, Buenos Aires, Argentina
[2] National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
1205 W. Clark St. Room 1008 Urbana, Illinois, USA

Abstract. It is difficult to maintain legacy Fortran programs that use outdated programming constructs, especially when this maintenance requires a detailed understanding of the code (e.g., for parallelization). Initially, we want to gauge the prevalence of such constructs by applying straightforward syntactic metrics to some well-known global climate models. Detailed information regarding files, subroutines, and loops has been collected from each model by applying a lightweight source code static analysis based on ASTs (Abstract Syntax Tree) for a posterior analysis. Modernizing Fortran Legacy programs is still a challenge. Our objective has been to collect relevant information on these programs to help us approach parallelizing legacy scientific programs in a shared memory environment (e.g. using multi-core processors). The data we collected indicate that old Fortran features are still being used on these models in these days. Furthermore, we propose some metrics to be used as a guide to determine how many changes a program needs in order to be modernized, optimized, and eventually, parallelized.

Key words: Source Code Modernization, HPC, Legacy Systems, Fortran Legacy Systems

1 Introduction

Over the last 50 years, Fortran has been a very popular programming language among scientists for building scientific software. Another particular aspect of this programming language that makes it a very interesting subject for a study is its long lived history. The Mathematical **For**mula **Tran**slating System (Fortran) was born in 1952. Contrary to popular belief among software engineers, this language is still widely used. The area of scientific computing, or rather, the area of numerical processing has been producing Fortran software over many decades. Throughout this period of time, most of the effort in the context of software engineering has been focused on the development process of new software rather than on other stages of the software life cycle. It is widely known that the

⋆ Comisión de Investigaciones Científicas de la Prov. de Bs. As.

maintenance stage takes up most of the cost involved in developing software, however this stage is not widely studied. By 1979, some studies had shown that the software maintenance costs was about 67% of the total project costs [20], in 2000 there was a report showing that this cost had grown to the 90% of the entire project cost [5]. Also, by 1983 studies conducted on software maintenance had revealed that 50% of the time was devoted to understanding the program [6]. In the year 2000, the number of legacy source code lines was calculated to be 250.000 billion in [17].

1.1 Legacy: Modernizing & Parallelization

There is not a formal definition of what a Legacy System is. But we can summarize the Legacy System concept as follows: "Legacy Software is critical software that cannot be modified efficiently" [7]. All the definitions have various things in common. One of them is resistance to change and another one is how this kind of software has become critical to the organization. An important concept that goes hand in hand with these definitions is the inherent complexity of legacy software. Legacy software poses a particular challenge in the maintenance stage. In this stage, software that has been running in production for 20 or 30 years is hard to manage because software gradually deteriorates. During maintenance, a program may need different types of changes. Enhancements, corrections, adaptations and preventions to a system may be needed. All of these tasks require knowledge and comprehension of the system.

From an "Unknown" to a "Manageable" Project. If we are determined to modernize or parallelize an old Fortran source code and if it can be considered a legacy program, we should, as a first step, break through from the "Unknown" to a "Manageable" process. There is a set of issues that are generally found in Fortran Legacy Source Code [19], such as the following: software seems to have been built with old design styles, documentation is rarely found, different programming styles are found among source code lines, and the wide usage of Common Blocks, defined as mechanisms for sharing memory among subroutines.

Enhancing different aspects. We have measured some source code characteristics in order to determine: how many changes on source code should be applied, which kind of source transformation should be applied on it, and to obtain information to help us quantify the changes needed. In order to gather these data we have focused our analysis on a small set of scientific applications in the field of climatology. We have worked on Global Climate Models (GCM) by performing a detailed measurement on their source code. "Our purpose has been purely scientific in an attempt to find how things are, without moralizing or judging people's competence" [9] or their work. We have the following goals for legacy software: modernization, upgrading and parallelization, turning it into a manageable program, and leaving it ready for further changes in the future.

Modernization. Fortran has faced a particular evolution throughout its own life. It has been the first high level programming language to have its own ASA (American Standard Association) standard (now ANSI) [2]. It was known as FORTRAN 66; during the last 46 years this standard has been revised five times (1978, 1991, 1997, 2004, 2010) [13]. This evolutionary process brought about changes in the language such as: new language features, the deletion of some language constructions, the obsolescence of language features. Even when obsolete features were deleted, compatibility remained: "Unlike Fortran 90, Fortran 95 was not a superset; it deleted a small number of so-called obsolescent features. This incompatibility is more theoretical than real however, as all existing Fortran 95 compilers include the deleted features as extensions" [10]. Modernizing and updating old FORTRAN source code has become an important objective because a large set of programs written years ago are still operative these days. A good example can be found in scientific software.

Parallelization. A major problem to deal with is the need for better performance. One reliable option is taking advantage of parallel processing by using shared memory parallel hardware. Even though multi-core processors are widely spread even in small scale computers, their facilities are not fully exploited. A good starting point when it comes to taking advantage of shared memory parallel processing is to use libraries or facilities like those provided by threading. But if we venture into the parallelization task, we will find some obstacles closely related with the aforementioned modernization issues. There are Fortran programs which present a large set of old language features such as: fixed format without any indentation, obsolete language features still in use, deleted language features which are being used, and so forth. These features make the parallelization process a very hard and error prone task.

2 Several Metrics: The Rational

The information gathering process has been divided into 4 categories, each one related to: files, subroutines, Modules, and Do Loops. The information has been gathered and exported into comma-separated value (CSV) files to be analysed. For each file, subroutine or module found in a program we have measured data as shown in Table 1. In addition, we have taken the following measurements for each file:

- Number of Fortran Include Lines: Fortran has its own INCLUDE statement.
- Number of Preprocessor directives: a set of distinct preprocessor directives has been quantified such as: Conditional Directives, Conditional Constructs, Inclusion, Macro Definitions, Pragmas and Controls.
- Two metrics per KLOC: number of Go To statement per file KLOC and number of obsolete features (listed in the Appendix B of the Fortran 2008 standard) per file KLOC.

Also, for each Do Loop found in a program source code, a detailed analysis was conducted in order to collect: number of source lines in Do Loops, I/O operations in the loop, subroutine calls performed, assignments, IF statements, Go To statements and OpenMP directives.

Feature	F	M	S	D
Sloc	X	X	X	X
NbNcSloc	X	X	X	
Logical Executable SLOC	X	X	X	
Subroutines	X	X		
Functions	X	X		
Procedures	X	X		
Pubblic Subroutine		X		
Public Function		X		
Use Satement		X		
Use Stmt With Only		X		
Public Procedures		X		
Private Procedures		X		
Variables		X		
PublicVar		X		
PrivateVar		X		
percentage Of Parameter		X		
Files Use		X		
Files Declare		X		
Goto	X		X	X
Assigned Goto	X		X	
Arithmetic If	X		X	
Stmt Functions	X		X	
Entry	X		X	
ComputedGoto	X		X	
Pause	X		X	
Do	X		X	X
Common Blocks	X		X	

Feature	F	M	S	D
Parameter			X	
Intent In			X	
Intent Out			X	
Intent In/Out			X	
If				X
Else	X			
End If	X			
End Do	X			
Continue	X			
Stop	X			
Return	X			
Call	X			X
Dimension	X			
Select Case Stmt	X			
Print	X			X
Write	X			X
Read	X			X
Format	X			
Continue Stmt	X			
Obsolete Operator	X			
Assignement	X			X
OpenMP Directives			X	X
New Style Do Loops	X		X	
Do While	X		X	
Old Style Do Loop	X		X	
Shared Do Loop	X		X	
Do Loop Nested Levels	X		X	

Table 1. F:file, M:Module, S:Subroutine, D:DO statement

Implementation. As a first step, we have used a tool to determine in which programming language these programs have been coded. To perform this analysis a tool called CLoC [22] has been used. We have conducted further analysis of the source code by using Abstract Syntax Trees (AST), i.e. a large number of Fortran source code lines have been transformed into an AST structure. These metrics have been implemented using Photran [21].

3 Global Climate Models

Among scientific software, Global Climate Models have some distinctive features that make them an attractive case study. The first one is the preponderant impact that they have in a world where climate change is a highly controversial phenomenon. Results emerging from these models have helped climatologist to shed light on the reasons behind global warming. The second interesting feature is related to the fact that GCMs are focused on one specific problem in its entirety. This stands in direct contraposition to synthetic programs or compute-intensive benchmark software used to evaluate certain features from a high performance system. Another good reason to select this kind of software, lies on the fact that they are running in a production environment. Hence, they are being satisfactorily evaluated from the perspective of the specific problem to be solved.

Finally, these programs are founded on mathematical and numerical models that represent the physical phenomena involved (climate in this case).

A set of six models has been selected from CMIP5 [18], including models from all over the world. Some of these models are licensed only for lawful scientific purposes in research and education, while others are completely free:

- **GISS-AOM**:(C4x3) from GISS, the NASA Goddard Institute for Space Studies [15].
- **GISS-ER:** ModelE20/Russell 4x5xL20 from GISS, the NASA Goddard Institute for Space Studies [16].
- **CSIRO Mk3**: from Commonwealth Scientific and Industrial Research Organization in collaboration with Queensland Climate Change Centre of Excellence [8].
- **IPSL**: from Institut Pierre-Simon Laplace [11].
- **COSMOS**: The COSMOS-1.2.1.1 package includes model codes of ECHAM5, ECHAM5J, MPIOM, and HAMOCC, and the OASIS coupler from The Max Planck Institute for Meteorology [14].
- **BCC-CSM1.1** from Beijing Climate Center, China Meteorological Administration [4]

4 Results

An example of the results gathered from CLoC analysis can be seen in Table 2.

	COSMOS			
Lang.	Files	blank	comment	code
F 90	551	34040	63827	163353
F 77	436	2149	62077	61731
C/C++ H	728	3914	275	25673
K Shell	39	811	1618	6575
C	11	1724	1826	6280
V. Basic	6	343	0	4942
bash	2	410	355	2309
B Shell	11	398	212	2265
make	20	356	131	629
C++	16	0	0	267
Pascal	3	4	0	255
Python	1	101	126	221
T def	3	100	191	112
awk	2	8	14	27
Assembly	1	31	0	8
XML	1	2	0	5
SUM:	1831	44391	130652	274652

	CSIRO-Mk3L			
Lang.	Files	blank	comment	code
F 77	329	7670	6612	48242
F 90	47	924	1635	6947
IDL	59	1110	1883	5721
C Shell	12	127	405	255
make	8	90	131	208
SUM:	455	9921	0666	61373

	GISS-AOM-4x3			
Lang.				
F 77	14	10	6547	16707
K Shell	6	36	66	232
B Shell	1	1	0	8
SUM:	21	47	6613	16947

	GISS-ER-ModelE			
Lang.	Files	blank	comment	code
Fortran 77	428	28900	76211	252999
Fortran 90	118	10844	11646	56911
HTML	8	605	33	5250
Perl	18	568	628	3037
B Shell	23	544	377	2131
make	17	317	90	915
K Shell	10	122	117	509
C/C++ H	8	36	0	307
Python	2	47	54	202
m4	3	56	294	145
Pascal	2	1	0	117
C	1	22	23	42
SUM:	638	42062	89473	322565

Table 2. Programming Language composition of COSMOS, CSIRO-Mk3L, GISS-ER-ModelE and GISS AOM 4x3 models

From the data collected it is undeniable and obvious that most of these models have been built mostly with Fortran. Another remarkable aspect that can be observed is the fact that there is a set of these models built solely with Fortran and shell script (bash or csh). The source code analysis has been conducted over 1.085.663 lines from 3840 Fortran source code files, resulting in 1.133.430 Fortran lines of code after the inclusion files were processed. Some interesting points resulting from the source code analysis are:

1. **A contrast** has been made with a previous work (hand made) performed by Donald Knuth in 1971 regarding 250.000 FORTRAN source code lines. There are some remarkable aspects to take into account. The first one is the reduction of Go To statements from 13% to 1,05%, the decrease of Format statements from 4% to 0,73% and Equivalence statements decreased from 0,7% to 0,04%. But surprisingly these features are found in modern programs: most of these models were built between 1990 and 2010. The second important aspect is the fact that new language features such as Functions, End If, End Do and Select Case statements represent approximately only 14% of the total number of statements. The third one is related to the finding of most of the language features labelled as obsolete in the Appendix B in the Fortran 2008 standard, from the Fortran 90 standard [1]. Some of these features that have been found were Arithmetic IF statements (91), Entry statements (206), Computed Go To statements (39), and others.

	GCMs		Lockheed	
Statements	Number	Percentage	Number	Percentage
Assignement	212167	43,63	78435	41
If	50556	10,40	27967	14,5
Gotos	5129	1,05	24942	13,0
Call	42564	8,75	15125	8,0
Continue	9623	1,98	9165	5,0
Write	17748	3,65	7795	4.0
Format	1775	0,37	7685	4,0
Do	39322	8,09	7476	4,0
Data	2257	0,46	4468	2,0
Return	6277	1,29	3639	2,0
Dimension	6059	1,25	3492	2,0
Common	818	0,17	2908	1,5
Subroutine	8154	1,68	2001	1,0
Rewind	100	0,02	1724	1.0
Equivalence	187	0,04	1382	0,7
EndFile	2	0,00	765	0,4
Read	2573	0,53	586	0,3
Print	2467	0,51	345	0,2
Entry	206	0,04	279	0,1

	GCMs		Lockheed	
Statements	Number	Percentage	Number	Percentage
Stop	2150	0,44	190	0,1
Pause	6	0,00	57	
Assign	13	0,00	57	
External	738	0,15	23	
Implicit	5663	1,16	0	
NameList	102	0,02	5	
BlockData	25	0,01	1	
Other			7286	3,2
Computed Got	39	0,00		
Assigned Goto	4	0,00	-	
Arithmetic If	91	0,02	-	
Function	1370	0,28	-	
Stmt Function	0	0,00	-	
End If	30516	6,28	-	
End Do	26884	5,53	-	
Else Stmt	9892	2,03	-	
Select Case	841	0,17	-	
TOTAL	486318	100	207941	100

Table 3. Distribution of Fortran statements types in GCMs and in [9]

2. **Go To statements:** We also measured the number of Go To statements per KLOC, since this can substantially hinder understandability. Programmers have been trying to avoid the inclusion of Go To statements in source code [3], and a high Go To density could prohibit modernizing or parallelizing the code. Spaghetti code is very complex to understand, making the modernizing process more difficult to achieve. Table 4 shows the density of Go To statements for each model. Some of these values are very descriptive of how many Go To statements should be removed from the source code. Go To statement is one of the most undesirable features to find in modern programs.The total number of Go To statements in all of the models is 5129, resulting in 4,53 Go To statements per thousand lines of code overall.
3. **Obsolete Features**: The more obsolete features found in the code, the more difficult the modernization tasks become. Counting the number of occurrences of obsolete features helps programmers to get a quick idea of how old the source code is. This metric has been applied to the models, see Table 5. Source code can be improved by applying some automatic source

code transformations in order to enhance code readability. Several transformations are currently available in Photran. After the transformations were applied, the obsolete features per KLOC (OF/KLOC) were reduced from 72,53 OF/KLOC to 53,60 OF/KLOC. In other words, automated transformations were able to remove 20 uses of obsolete features for every 1000 lines of code. Therefore, the use of automated source code transformation tools have a preponderant impact in the source code modernization tasks.

Model	Num. of Stmts Go To	/ KLOC
All Model		4,52
GissAOM	1395	34,91
GissER	1766	5,59
CSIRO	368	3,86
ipsl	251	0,14
Cosmos	1045	2,55
BCC-CSM	304	3,16

Table 4. number of Go To statements per KLOC

Model	Number OF	/ KLOC
all GCMs	4810	4,82
Giss AOM	1636	72,53
Giss ER	2685	9,58
Csiro	1788	23,29
ipsl	467	2,75
COSMOS	1194	3,15
BCC-CSM	6261	3,80

Table 5. number of Fortran language obsolete features listed in the standard per KLOC

4. **Do Loops**: In a previous work [12] we have studied different types of DO loop styles, such as old style Do Loops; (e.g. Do Loops that end in a labelled statement). In this work we have measured which types of DO were found in the code. From this analysis we have obtained some remarkable results, see Table 6 and Table 7:
   (a) No Do While statements have been found in the source code.
   (b) The 8% of the total DO loops have been recognized as Shared DO loops which have been marked as an obsolete feature since the advent of Fortran 90 standard.
   (c) The 30,89% of the total DO loops in the source code have been labelled as old style do loops.

	Number
DO Loops	40039
New Style DO Loops	27667
Old Style DO Loops	12372
Shared DO Loops	3407

Table 6. Type of Do loops Analysed

Model	NSDL/KLOC	OSDL/ KLOC
AOM	0	40,50
CSIRO	21,35	30,46
modelE	25,58	8,59
ipsl	6,55	0,85
Cosmos	6,23	0,37
BCC-CSM	3,69	0,08

Table 7. number of DO Types per KLOC

Table 7 in particular, shows that some models are updated in terms of Do Loops (e.g. BCC-CSM) while others should be updated before more elaborate transformations are applied (e.g. AOM).

5. **Number of Nested Do Loops**: Do Loops size and depth level have been analysed. Table 8 shows the depth of Do Loop statements in levels and their percentage. Figure 1 shows the distribution of the DO loop nested levels

through each model. Taking into account that most of the runtime is used

Depth	1	2	3	4	5	6	7
Number	19467	13520	5245	1456	285	58	8
Per cent	48,62	33,76	13,09	3,63	0,71	0,14	0,01

Table 8. Depth of nested levels from 40039 Do Loop statements analysed

in Do Loops, more nesting provides more opportunities and more flexibility for parallelization. For example, inner loops may be vectorized, while outer loops may provide opportunities for more coarse-grained parallelism.

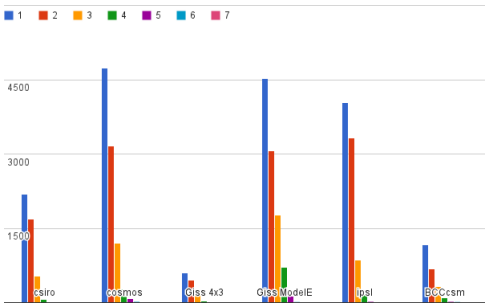

Fig. 1. Depth of nested levels on each Global Climate Model analysed

6. **Percentage of parallelized source code**: We also measured the number of Do loops that have already been parallelized using OpenMP. The size in lines of code (LOC) has been collected and finally compared with the total LOC. From this analysis, we can conclude that 7% of the total lines of source code are enclosed within OpenMP directives. This information will support some further analysis once the code is run in a parallel environment.

5 Conclusions and Further Work

GCMs have been gaining momentum over the last years because of their impact in climate research on global warming. There are many scientific institutions working and spending resources on these programs. In this article, we have analysed how these programs have been built with a view on where and how they can be improved, updated or modernized. Since the ultimate goal is parallelization, we have included some metrics specifically focused on the code most likely to be parallelized: Do Loops. Other specific metrics have shown that old and obsolete features are still in use. Some of them are very complex to eliminate or update, such as Go To statements. On the other hand, there are some features

that can be easily modernized with a good source code transformation tool, like those found in modern IDEs. In addition, these improvements can be measured by using the same metrics.

Future work will attempt to analyse a larger set of Fortran source code gathered from the internet in order to try to determine how Fortran source code is structured. More metrics could be added to the analysis. Combining these with the ones introduced in this work may lead to more useful information or insight for optimization and/or parallelization. We still have to address two main issues regarding metrics and their usefulness: 1) combining static with dynamic measurements (such as runtime profiles and traces), and 2) quantitative/objective measurement on likelihood and/or complexity and feasibility for parallelization in shared as well as distributed memory parallel hardware.

References

1. American National Standards Institute. *American National Standard for programming language, FORTRAN — extended: ANSI X3.198-1992: ISO/IEC 1539: 1991 (E)*. American National Standards Institute, September 1992.
2. B. Bates. C# as a first language: a comparison with c++. *Journal of Computing Sciences in Colleges*, 19(3):89–95, 2004.
3. E.W. Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.
4. Y. Ding, Y. Xu, ZC Zhao, Y. Luo, and X. Gao. Climate change scenarios over east asia and china in the future 100 years. *Climate Change Newsletter*, pages 2–4, 2004.
5. L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, 2000.
6. R.K. Fjeldstad and W.T. Hamlen. Application program maintenance study: Report to our respondents. *Proceedings Guide*, 48, 1983.
7. N.E. Gold. *The meaning of legacy systems*. Univ. of Durham, Dept. of Computer Science, 1998.
8. H. Gordon, S. O'Farrell, M. Collier, M. Dix, L. Rotstayn, E. Kowalczyk, T. Hirst, and I. Watterson. *The CSIRO Mk3. 5 Climate Model*. Centre for Australian Weather and Climate Research, 2010.
9. D.E. Knuth. An empirical study of fortran programs. *Software: Practice and Experience*, 1(2):105–133, 1971.
10. Cohen Malcom. Fortran: A Few Historical Details. http://www.nag.co.uk/nagware/NP/doc/fhistory.asp, October 2004.
11. O. Marti, P. Braconnot, J. Bellier, R. Benshila, S. Bony, P. Brockmann, P. Cadule, A. Caubel, S. Denvil, JL Dufresne, et al. The new ipsl climate system model: Ipsl-cm4. 2005.
12. M. Méndez, J. Overbey, A. Garrido, F.G. Tinetti, and R. Johnson. A catalog and classification of fortran refactorings. In *11th Argentine Symposium on Software Engineering (ASSE 2010)*, pages 500–505, 2010.
13. M. Metcalf. The seven ages of fortran. *Journal of Computer Science and Technology*, 11(1):1–8, 2011.
14. E. Roeckner and Max-Planck-Institut für Meteorologie. The atmospheric general circulation model echam5: Part 1: Model description, 2003.

15. G.L. Russell, J.R. Miller, and D. Rind. A coupled atmosphere-ocean model for transient climate change studies. *Atmosphere-ocean*, 33(4):683–730, 1995.
16. G.A. Schmidt, R. Ruedy, J.E. Hansen, I. Aleinov, N. Bell, M. Bauer, S. Bauer, B. Cairns, V. Canuto, Y. Cheng, et al. Present-day atmospheric simulations using giss modele: Comparison to in situ, satellite, and reanalysis data. *Journal of Climate*, 19(2):153–192, 2006.
17. I. Sommerville. *Software engineering*. Addison-Wesley New York, 2000.
18. K.E. Taylor, R.J. Stouffer, and G.A. Meehl. An overview of cmip5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485, 2012.
19. F.G. Tinetti and M. Méndez. Fortran legacy software: source code update and possible parallelisation issues. In *ACM SIGPLAN Fortran Forum*, volume 31, pages 5–22. ACM, 2012.
20. M.V. Zelkowitz, A.C. Shaw, and J.D. Gannon. *Principles of software engineering and design*. Prentice Hall Professional Technical Reference, 1979.
21. Photran, an Integrated Development Environment and Refactoring Tool for Fortran. http://www.eclipse.org/photran/.
22. Count lines of code: Cloc. http://cloc.sourceforge.net/.