

PLASMA Contributors' Guide

Parallel Linear Algebra Software for Multi-core Architectures

Version 2.0

Electrical Engineering and Computer Science
University of Tennessee

Electrical Engineering and Computer Science
University California, Berkeley

Mathematical & Statistical Sciences
University of Colorado, Denver

Jack Dongarra
Jakub Kurzak
Piotr Luszczyk

Contents

1	Introduction	1
2	Coding Style	2
2.1	FORTRAN Style	2
2.2	C Style	2
2.3	Coding Practices	4
2.4	Naming Convention	4
2.5	Boiler Plate text/code for Each File	5
2.6	Exceptions	5

CHAPTER 1

Introduction

PLASMA is a relatively young project. The rules for contributing code are expected to clarify some time before the SC'09 conference. By then please contact the PLASMA team before developing code intended to be included in PLASMA. For the time being this document serves mostly as an internal document.

CHAPTER 2

Coding Style

2.1 FORTRAN Style

FORTRAN means FORTRAN 77. Extensions from Fortran 90, Fortran 95, or Fortran 2003 are not allowed.

Currently PLASMA contains significant amount of FORTRAN code located in the `coreblas/` directory. This code implements single-core operations on matrix tiles. In the future this code might or might not be rewritten to C for portability reasons. The advantage of keeping this code in FORTRAN is its close resemblance of LAPACK code, from which, for the most part, the code is derived. By the same token, the main coding rule, applying to the development and maintenance of this code, is that it should follow LAPACK as closely as possible. This applies to the use of whitespaces, punctuation, indentation, line breaking, the use of lower and uppercase characters, comments, variable naming, etc.

2.2 C Style

Only code that conforms to the ANSI C standard is allowed. The standard is commonly referred to as C89 and was ratified by ISO. One way to check for compliance is to use the following command:

```
gcc -std=c89 -W -Wall -pedantic -c plasma.c
```

Since the C89 standard does not support complex data types the following command needs to be used to remove warnings about it:

```
gcc -std=c99 -W -Wall -pedantic -c plasma.c
```

PLASMA code needs to be portable and work on Windows where the most commonly used compiler is a C++ compiler. PLASMA code must then compile with a C++ compiler. The following command will compile a C source code using the GNU C++ compiler:

```
gcc -x c++ -W -Wall -pedantic -c plasma.c
```

No Trailing Whitespaces: There should be no trailing whitespace characters at the end of lines, no whitespace characters in empty lines and no whitespace characters at the end of files (The last closing curly bracket should be followed by a single newline). This is easy to accomplish by using an editor that shows whitespace characters, such as Kwrite, Kate, Emacs (just use M-x `delete-trailing-whitespace` command). Otherwise a `sed`, `awk`, or perl “one-liner” script can be used to clean up the file before committing to the repository (e.g., `tools/code_cleanup`).

Whitespace Separators: There should be a whitespace between a C language keyword and the left round bracket and a whitespace between the right round bracket and the left curly bracket. There should be no whitespace immediately after left round bracket and immediately before right round bracket. Comas separating arguments are followed by a single space and not preceded by a space.

End-of-line Management: Every file should have an end-of-line character at the end unless it’s a zero-length file. End-of-file character is `\n` (as it is on Unix including Linux; ASCII code 10). Other end-of-line schemes should not be used: Windows and DOS (`\n\r` – ASCII codes 10 and 13) and Mac (`\r` – ASCII code 13).

Indentation: The unit of indentation is four spaces. the left curly bracket follows the control flow statement in the same line. There is no newline between the control flow statement and the block enclosed by curly braces. The closing curly bracket is in a new line right after the end of the enclosed block.

There is no specific limit on the length of lines. Up to a 100 columns is fine. Clarity is paramount. For multi-line function calls it is recommended that new lines start in the column immediately following the left bracket.

Tabs: Tab characters should not be used. Tabs should always be emulated by four spaces, a feature available in almost any text editor. If that proves difficult, again, a `sed`, `awk`, or perl “one-liner” can be used to do the replacement before the commit.

Variable Declarations: For the most part all variables should be declared at the beginning of each function, unless doing otherwise significantly improves code clarity in a specific case.

Constants: Constants should have appropriate types. If a constant serves as a floating point constant, it should be written with the decimal point. If a constant is a bit mask, it is recommended that it is given in hexadecimal notation.

Printf Strings: ANSI C concatenates strings separated by whitespace. There is no need for multiple printf calls to print a multi-line message. One printf can be used with multiple strings.

F77 Trailing Underscore: When calling a FORTRAN function the trailing underscore should never be used. If the underscore is needed it should be added by an appropriate conditional preprocessor definition in an appropriate header file (e.g.: `core_blas.h`, `lapack.h`).

Special Characters: No special characters should be used in the code. The ASCII codes allowed in the file are between 32 and 127 and code 10 for new line.

2.3 Coding Practices

Dead Code: There should be no dead code: no code that is never executed, no including of header files that are not necessary, no unused variables. Dead code can be justified if it serves as a comment, e.g., canonical form of optimized code. In such case the code should be in comments.

OS Interactions: Error checks have to follow each interaction with the OS. The code should never be terminated by the OS. In particular each memory allocation should be checked. The code cannot produce a segmentation fault.

User Interactions: User input needs to be checked for correctness. The user should not be able to cause undefined behavior. In particular the user should not be able to cause termination of the code by the OS.

2.4 Naming Convention

Any externally visible C symbols should be prefixed with `PLASMA_`. Following the prefix, the name should be in lower case (this will create a mixed-case name and thus will guarantee the lack of name clashes with FORTRAN interfaces that are always either all lower-case or all upper-case). For example: `PLASMA_dgetrf`. This is in line with C interfaces for MPI (`MPI_Send`), PETSc, and BLAS (`BLAS_dgemm`).

2.5 Boiler Plate text/code for Each File

Copyright, License, year, . . .

2.6 Exceptions

As often is the case all rules have exceptions. Exceptions should only be used after consulting with the PLASMA team members.