



MPT Automatic Code Reuse

The Core Concepts of Blue Cheetah CUB

Author

Kevin D. Howard

Original Date: 22 September 2010

Update: 29 September 2010

This document assumes that the Massively Parallel Technologies, Inc., (MPT) paper entitled “MPT Kernels – Blue Cheetah CUB’s Core Concepts” has been read and understood. The formal concept of code reuse dates back to 1968 when Douglas McIlroy of Bell Laboratories proposed basing the software industry on reusable components. Since then, a number of related concepts have been developed: cut and paste, software libraries, and object-oriented programming are examples. Other techniques, such as generic frames and component-based software engineering, attempt to reuse components by automatically coding. However, David Parnas, a key developer of object-oriented programming concepts, concluded that automatic programming has always been a euphemism for programming in a higher-level language than was available at the time to the programmer. The primary problem with code-reuse techniques is that they still require the programmer to select the proper reusable code to use, forcing a manual activity on what is attempting to be an automatic process. In the MPT Kernels paper, it was shown that the only code that must be written is the code required for the MPT Process Kernels (MPKs) and that an MPK represents the linearly independent code. Further, the description for the required MPK is a function of the MPT design, that is, the creation of MPT Kernels from the MPT Algorithm Decomposition (MAD) model.

Each MPK generates a Cyclomatic Complexity (CC) M value of -1. MPKs are depicted below¹:

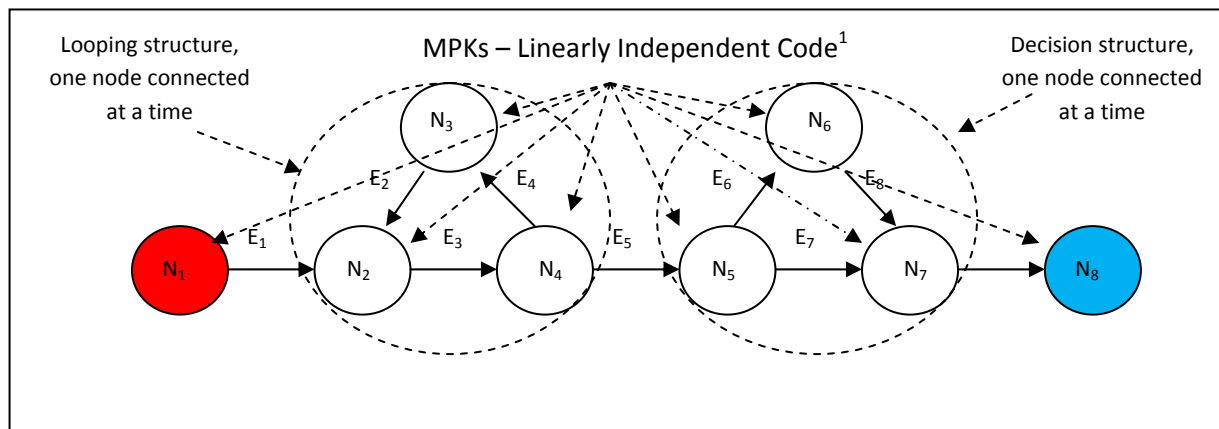


Figure 1 Algorithm with 8 Linearly Independent Components (N₁ - N₈)

Any of the linearly independent components, or MPKs, consists of one or more linearly dependent components as shown below:

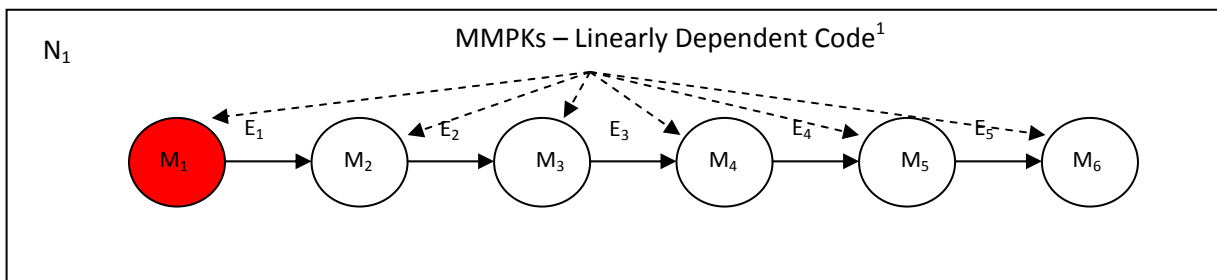


Figure 2 N₁ with 6 Linearly Dependent Components (M₁ - M₆)

Calculating the CC value, any MPK without considering any internal, linearly dependent components would generate an M value of -1. Calculating the CC value for an MPK with the inclusion of any internal linearly dependent components may generate either the value 1 (if there is more than one internal component) or the value -1 (if there is only one internal component). This secondary CC analysis is called the MPT Algorithm Decomposition of an MPT Algorithm Decomposition or MAD². MAD² produces only MADs of MPKs (MMPKs). MMPKs represent the smallest functional linearly dependent code blocks used within an algorithm. These small functional code blocks can now be annotated as part of the design process, analogously to the way MPKs are annotated. Five prerequisites at the functional-code-block level must be met to enable a system to automatically find and properly use the MMPK:

- 1) The executable version of each MMPK must be accessible by the system.
- 2) A separate keyword list must be associated with each MMPK so that the system can find possible matches.
- 3) An input and output parameter list with parameter type information must be associated with each MMPK.
- 4) A way to save new MMPKs with associated keyword lists and parameter information must be available.
- 5) A functional decomposition design must continue to the MMPK level and must contain:
 - a. a keyword list per functional block,
 - b. a description of the input and output parameters with type information,
 - c. an input dataset per functional block, and
 - d. an expected output dataset per functional block.

When the design is complete, the system can perform a keyword search for all of the MMPKs within the design. The found MMPKs can process the input dataset and compare the results to the expected output dataset. If there is a match in keyword lists, input/output parameters, and the data transformation for an area of the design, that design area is replaced with the matched MMPK. Since the design already contains the automatically programmed control aspects of an algorithm (see “MPT Kernels – Blue Cheetah CUB’s Core Concepts”), the added MMPKs will perform the required work under the proper conditions; that is, code from completely different problems can, thus, be reused automatically. Any MMPKs within the design with no equivalent pre-coded MMPK can now be coded, annotated with a keyword list and with any input/output parameter definitions. Coding and annotating additional MMPKs increases the available list of automatically reusable code.

¹ Description of Cyclomatic Complexity substantially from wikipedia: http://en.wikipedia.org/wiki/Cyclomatic_complexity