Enhanced Macrostep-based Debugging Methodology for Parallel Programs

Róbert Lovas and Péter Kacsuk

The macrostep-based debugging concept has been developed originally for message passing parallel programs designed in the frame of P-GRADE graphical programming environment. The macrostep is essentially the set of executed code regions between two consecutive collective breakpoints, which breakpoints are placed on communication operations automatically. The main advantage of the macrostep-based debugging is the handling of non-deterministic behaviour of point-to-point message passing applications that is inherited from wildcard receiving operations. By using the current version of macrostep-based debugger, program developers and testers can replay the entire parallel application and also apply the cyclic debugging technique that was introduced for sequential programs. Moreover, the macrostep debugger tool is able to generate all the possible timing conditions systematically therefore, the parallel application is forced to traverse its entire state-space.

In this paper we present a formal description of macrostep-by-macrostep execution of message passing parallel programs as well as the correctness of macrostep debugging methodology by showing the relation between the state-space of parallel applications without macrosteps and the reduced state-space when we apply macrosteps. Based on the formal description the macrostep-based debugging concept has been enhanced in order to support the different ways of collective communication (multicast, scatter, etc.) and the pre-defined scalable process communication templates, such as pipe, mesh or tree. Hereby, the new prototype of macrostep debugger tool can be used in the P-GRADE graphical programming environment without limitations and its usage is illustrated by a simple example.

Due to the combinatorial explosion the exhaustive traverse of each execution path might be impossible in case of real-size programs but some techniques can help the user traverse the different execution paths of parallel applications efficiently. Finally, we summarise some traditional and some novel methods to reduce further or cut the state-space of parallel applications (that must be traversed during the exhaustive testing), such as run-time temporal logic assertions, low-cost static and dynamic analysis of parallel behaviour based on coloured Petri-nets. The integration and evaluation of these methods are under development but some preliminary results are available and presented in the paper.