Programmer-Guided Reliability in Chapel[1]

David E. Bernholdt,[2] Wael R. Elwasif, Christos Kartsaklis, Seyong Lee, and Tiffany M. Mintz
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831 USA

Extended Abstract

The "Programmer Guided Reliability" (PGR) project investigates the use of programmer-provided information (code and annotations) to enhance the detection and correction of silent data corruption (SDC) in technical applications, with the goal of characterizing the effectiveness and cost (in terms of performance and energy) of such an approach, and to allow the user (or system managers) to make trade-offs in the space of reliability, energy, and performance (R-E-P).

We have chosen to focus on SDC because (a) it is one of the least understood and least considered classes of errors that applications experience today, and (b) trends in computer architecture and power constraints suggest that future systems are likely to be significantly more vulnerable to SDC than current systems. Additionally, this class of errors may be more effectively addressed at higher levels of the software stack, than a one-size-fits-all solution in hardware or the lowest levels of the software stack.

At the heart of the PGR approach is the programmer providing additional code to detect, and possibly correct, data corruption errors within the application (generically referred to as "error detectors"). Error detectors may be derived from characteristics of the algorithms, data structures, or even the nature of the target problems, and other factors which are understood by the application scientist, but generally not susceptible to automatic analysis by the compiler (though we anticipate that some common error detection patterns may be automatable). The programmer annotates the code to indicate that it is reliability-related, and also to indicate parameters of the algorithm that can be used to control its application at run time.

All application code, including error detectors, impacts the cost of the application, both in terms of performance, and power consumption. Many types of error detectors naturally expose parameters which influence its efficacy and cost. For example, the frequency with which the checksum of a static data structure is verified, or the degree of redundancy in a redundant storage or execution approach will impact both the detector's efficacy, but also its cost. This gives us a means to first characterize, and later control the position of the application within a three-dimensional trade space of reliability, energy, and performance. Characterization will help to identify different types of error detectors which might be most effective for the needs of a given application, and as the community develops experience with an increasing variety of error detectors and applications, it will help determine "best practices" for application-level error detection. Control will allow users and system managers to manage the R-E-P trade-off according to their needs, either an application launch, or even dynamically during execution, for example, in response to a need to limit energy use or indications of reduced reliability within the system.

We have chosen to use Chapel as the vehicle for demonstrating our ideas and approach. In addition to being well-aligned with trends in HPC architectures, its "multiresolution" approach, with higher-level language features implemented (in Chapel) on more fundamental features provides an opportunity to explore the implications of error detectors in multiple layers of code, with different degrees of exposure to users. We believe that the ideas developed in this project, implemented in Chapel, will also be quite portable to other language environments.

[1]Sponsored by the DoD Advanced Computing Initiative. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the Department of Energy under contract DE-AC05-00OR22725.

[2]Presenting author and primary contact. Email: *bernholdtde@ornl.gov*