INTRODUCTION

System Resilience is a key challenge on the road to advanced exascale capability computing systems due to:
• Large number of processor cores, memory modules projected to be deployed; System MTTF scales inversely to number of components
• Deep scaling of process technologies and near-threshold voltage operation leading to increased soft error induced failures, aging effects etc.

Proposed Approach: Introspective Resilience: HPC application programmers often have insights on application fault tolerance. But no convenient mechanisms in today’s programming models to convey such information
• Knowledge leveraged by compiler & introspective runtime framework to build resilience knowledge base
• Introspective runtime framework observes the rate and source of anomalous fault events and affects application execution and assignment of system resources to the application.

PROGRAMMING LANGUAGE EXTENSIONS

Simple Language Extensions based on familiar constructs to enable programmer to express their fault tolerance knowledge to the compiler & system runtime

RESULTS: APPLICATION SURVIVABILITY BASED ON LANGUAGE EXTENSIONS

RESULTS: PERFORMANCE EVALUATION OF REDUNDANT MULTITHREADING WITH INTROSPECTION

INTROSPECTION FRAMEWORK

The runtime framework leverages the tolerance knowledge conveyed by the programmer through the programming language extensions, those exposed by compiler transformations
• Dynamic Resilience Map (DRM) is maintained as an active knowledge base of the addresses and offsets in the logical address space of the application marked tolerant.
• Observation of system ’s anomalous events including corrected errors, thermal indicators etc. and trend analysis of rate and source of faults.

COMPILE TRANSFORMATIONS: OPPORTUNISTIC & LAZY FAULT DETECTION THROUGH REDUNDANT MULTITHREADING

The structured code blocks specified by programming directive are outlined by the compiler, but
• Executed by a single thread until the runtime system intervenes.
• Introspective runtime system makes inferences about the vulnerability of the system resources and signals the application to dynamically enable/disable the redundant multithreaded execution of the outlined code blocks (Figure (a))
• Optimization: Lazy detection through relaxed comparison of values generated by duplicate threads (Figure (b))

CONCLUSIONS

• Cross-layer approach based on introspection: programming model extensions capture programmer’s fault tolerance insights which are leveraged by compiler and runtime framework
• Error detection/correction capabilities provided through source-level code transformations by compiler framework
• Runtime inference allows for reasoning about the context and significance of faults to the application outcomes.