

Transactional Memory Support in the GCC Open-Source Compiler Framework

Transactional memory (TM) has emerged as a promising technology to simplify parallel programming for multi-core and larger-scale parallel systems. TM provides an intuitive, high-level abstraction (atomic memory transactions) that allows programmers to simply declare the correct behavior of parallel programs. Low-level concurrency management as multiple threads access shared memory becomes the responsibility of the system.

Numerous universities and several companies (e.g., Intel, Microsoft, Sun, and IBM) are actively investigating TM technology. There has been an increasing number of papers that showcase the potential of transactions in parallel programs, introduce TM implementation techniques, and propose hardware mechanisms that accelerate common case behavior. Nevertheless, TM is not widely used by application developers as the technology has not been integrated with any of the widely used software development frameworks. It is unlikely that programmers will abandon the many advantages of mature frameworks in order to try this new technology. As a result, there are few large applications that use transactions and there is clear assessment on whether TM actually improves programmability. In addition, there is little convergence or sharing of technology between TM researchers. Each group uses a different programming model, code generation, and TM system approach. This leads in incompatible and immature modules that are unlikely to attract significant attention.

This proposal suggests that the HiPEAC network supports an engineer to introduce basic transactional memory support in the GCC open-source compiler framework. Integrating TM in the most widely-used software development platform will provide application developers and researchers with the following benefits:

- Programmers will be able to use TM techniques for automatic concurrency management in large applications.
- It will create a large volume of portable applications that use TM. Programmers will benefit as there will be examples and libraries of modules to use in their work. TM researcher will have the applications for benchmarking, analysis, and comparison purposes. The applications will also guide further research on TM programming, implementations, and hardware support. Finally, they will also allow us to evaluate in practice the actual advantages of TM in terms of programmability.
- The open-source framework will provide the basis for further research in concurrency that goes beyond TM. Researchers will be able to investigate ideas in programming models, compilation techniques, and runtime system by making incremental changes over a well-understood base. For example, a group may investigate how data-flow techniques can be combined with TM without spending a significant amount of time reproducing well-known results in TM technology.
- It will also motivate uses of TM beyond concurrency control. Transactions provide a mechanism for speculation or undo of software execution. This mechanism can be particularly useful for research in security, debugging, fault-

tolerance, thread-level speculation, and compiler-based aggressive optimizations.

Our philosophy is to provide support for the basic TM features that are widely recognized as important. As TM research advances, additional features can be added into the open-source framework as needed. At this point, the basic set of features includes flat transactions at word-granularity that are well-integrated into the programming language. The basic criteria for selecting features are simplicity, orthogonality to other features, predictability, and simplicity of implementation. Hence, the statement of work will include:

- Introduction of the `atomic {}` keyword in the GCC front-end for C/C++.
- Define a simple memory model for transactions. We currently favor the weak consistency model of OpenMP. Transactions will be orthogonal to try/catch mechanisms in C++.
- Extensions to the intermediate representation to capture the attributes of atomic blocks.
- Specification of a simple interface that allows GCC-generated code to be linked with a range of TM implementations (software, hardware, and hybrid).
- A compiler pass that introduces the TM read and write barriers necessary for management of transactional state and performs automatic function cloning for functions called within transactions.
- Compiler passes that perform basic optimizations for transactional code such as elimination of redundant barriers, partial inlining of TM barriers, code size reduction by eliminating redundant cloning, etc.

We should stress that the goal is to develop a portable compiler infrastructure that works with a range of TM implementations. By defining a flexible interface, users will be able to link their preferred TM system in order to facilitate their performance needs (e.g., scale of system, application characteristics) or research objectives. However, as part of this effort we will ensure that that GCC framework works well some of the popular, open-source STMs (e.g., RSTM from Rochester, TL2 from Sun, etc).

We believe that this work will require support for two to three development engineers for a period of one year. The tasks discussed above are well understood in the TM research literature. We expect that several research groups in Europe and the US will be able to contribute modules directly, participate in the discussion of interfaces, and help with testing and benchmarking. Overall, this work involves compiler development, not compiler research.

The outcome of this effort will be released as an experimental branch of the GCC environment. Other groups are likely to release further extensions that build upon this work. For example, the TM research group at Stanford University will extend its OpenTM framework to use the experiment compiler as its base. The result will be an environment that combines OpenMP, transactional memory, and GCC. We will also create links with application groups in order to receive early feedback from programmers on the usability of the framework.