Abstract and Goals: The goal of the proposed research is to enable parallelization and optimization of applications for execution on contemporary multi-core platforms. Via code annotations, the user will provide hints for the presence of parallelism and the usage of static and dynamic data structures. These hints will enable exploitation of potential parallelism in these applications with high levels of efficiency via speculative execution. The runtime system will ensure safe execution in the presence of speculation. Finally, code optimizations will be developed for improving the efficiency of parallel execution on multi-cores.

Keywords: Software and hardware systems infrastructure.

1. Execution Model for Speculative Parallel Execution

Recently we proposed the Copy or Discard (CorD) execution model that provides support for speculative execution [MICRO08]. A key feature of this execution model is the CorD mechanism that enables speculative execution as follows. The state of a running program is divided into two parts, non-speculative state and speculative state. The non-speculative state is maintained separately from the speculative state, and when a speculative sub-computation has finished, the CorD mechanism is used to handle the speculatively computed results. If speculation is successful, the results of the speculative computation are committed by copying them into the non-speculative state. If miss-speculation is detected, no costly state recovery mechanisms are needed as the speculative state can be simply discarded. For the above approach to work, separate storage is allocated for a speculative computation. The values of shared-memory variables that are referenced by the speculative computation are copied-in at the start of the speculative computation and, once the speculative computation has completed without miss-speculation, the results are copied-out into the non-speculative state.

Under the above execution model, a parallel application consists of the main thread that maintains the non-speculative state of the computation and multiple parallel threads that execute parts of the computation using speculatively-read operand values from non-speculative state, thereby producing the speculative state of the computation. The main thread commits the speculative state generated by parallel threads in order, i.e., a parallel thread that is assigned an earlier portion of a computation from a sequential program must commit its results before a parallel thread that is assigned a later portion of a computation from the sequential program. Before committing results, the main thread confirms that the speculatively read values conform to the sequential program semantics. Version numbers of variable values are maintained to detect miss-speculation. Upon miss-speculation, the code must be re-executed by the parallel thread.

We identified several sequential programs and parallelized them to take advantage of the above execution model. The above approach was highly effective for these applications because of the following reasons: the computations that can be speculatively executed are coarse-grained; the miss-speculation rate is extremely low; and the shared data that can cause miss-speculation consists of a few variables and thus miss-speculation detection can be performed at a very small cost even in software. As shown below, our experiments with these applications resulted in speedups ranging from 3.7 to 7.8 on a Dell PowerEdge 1900 server with 8 cores (two Intel Xeon quad-core processors) at 3.00 GHz and 16 GB of RAM.

2. Proposed Research

Encouraged by the above results, we propose to explore the use of CorD execution model for applications where parallelism is more fine-grained and greater data sharing exists. In order to achieve this goal we will design annotations through which the programmer will be able provide information that will enable highly optimized implementation of CorD. We also propose to develop architectural support that exposes occurrence of dynamic inter-processor dependences to an application. Inter-processor dependences involving shared data being read or written during a speculative computation is an indicator of miss-speculation. The application will be augmented with a handler that responds to inter-processor dependences by discarding speculative state and restarting the speculative computation.

2.1. Fine-grained Parallelism: Programmer Assisted

A number of issues must be addressed for the above approach to succeed in extracting and exploiting parallelism present in applications with complex control flow. While fine-grained parallelism is frequently observed at runtime, it cannot be detected statically due to presence of infrequently exercised and/or harmless dependences. While the combination of CorD and inter-processor dependence detection mechanism is capable of exploiting parallelism via speculative execution, programmer must provide critical information in the form of hints to the compiler. This information will be expressed via code annotations that fall into two major categories:

Annotations for Parallelism. The programmer will annotate loops and functions such that iterations of the loops and calls to functions represent potential candidates for parallel execution. In other words, these annotated loops and functions represent likely sources of parallelism. However, the burden of correctness of parallelization is not placed on the programmer as parallelism is safely exploited via speculative execution. Miss-speculation detection will be achieved by developing a highly optimized mechanism for inter-processor dependence detection. If miss-speculation is detected, this mechanism will trigger the execution of a handler that will be responsible for taking recovery actions, i.e. discarding speculative state and initiating re-execution of speculative computation.

Annotations for Efficiency. The state separation between the non-speculative computation and parallel speculative computation threads entails inevitable memory copy operations. Specifically, the data referenced by a speculative computation has to be copied into the corresponding memory state so that the speculative computation can be performed. Also, new data values produced by a speculative computation need to be copied back to non-speculative memory state when the speculative computation succeeds. However, performing copy operation for all variables appearing in the speculative body may be unnecessary (e.g., some of the data are only locally used by parallel threads). A mechanism for annotating the data will be developed and provided to the programmer that he/she can use to communicate such hints to the compiler so that copying operations can be optimized. Finally, a mechanism for on-the-fly copying of data from non-speculative state to speculative state will be provided to handle situations when speculative parallel thread unexpectedly access data that has not already been copied.

Profiling will play a critical role in guiding the introduction of annotations by the programmer. In particular, profiling can be used to identify frequently executed program paths where not only most of the execution time is spent but also parallelism is observed. In addition, the observed usage patterns of data will be reported to the programmer. The programmer can then spend most of effort in providing annotations for relevant sections of code and relevant subset of data structures that have been identified via profiling. Finally, we will also develop adaptive techniques for controlling the degree of speculation. This is because speculation places additional strain on system resources (processing and memory) and its overall benefit is determined by the miss-speculation rate.

2.2. Fine-Grained Optimizations: Compiler Assisted

One of the impediments to generating highly optimized code for parallel applications is the interaction between parallel threads via synchronization operations. We have observed that synchronization points limit the scope over which optimizations can be performed thus limiting the efficiency of generated code. Moreover, when one thread reaches a synchronization point early, it has to wait for the arrival of the corresponding thread. Often these synchronizations are introduced to enforce a small number of inter-processor dependences. However, their impact on performance is significant because they limit the scope of optimization. We will develop an optimization framework that will enable us to speculatively optimize the code and then use light-weight dynamic analysis to detect violation of inter-processor dependences. A combination of static analysis and user provided information will be employed to identify a small set of memory locations that must be considered by the dynamic analysis for potential inter-processor dependency violations. We have already identified two optimizations:
Optimizing Register Usage. We will extend the idea of load speculation (data speculation) such that it can be applied in context of parallel programs running on multi-cores. Data speculation support will be used to expand the scope of register allocation to shared variables that are accessed across synchronization operations. This will allow us to eliminate significant amount of redundant stores and loads introduced preceding and following synchronization points. Dynamic analysis will be designed that will enable invalidation of speculatively loaded register values if they are modified by computations being executed on other cores.

Reducing Synchronization Delays. We will address this problem by allowing the thread that arrives at a synchronization point (e.g., a barrier) early to speculatively execute past the synchronization point. This form of speculative execution will also be achieved using the CorD mechanism of our speculative execution model. Consider a scenario in which loop iterations of an inner parallel loop are scheduled across multiple cores for execution and each invocation of this inner loop is preceded by synchronization of the cores via barrier synchronization. If a core completes its work before other cores, instead of waiting at the barrier it can speculatively execute the computation assigned to it during the next invocation of the inner loop. This will require that two versions of the computations be created: one corresponding to non-speculative execution and another for speculative computation. Each core can select the appropriate version at runtime. The speculative version will be created according to the CorD model.

The above optimizations are examples of situations where compiler-support will be developed to improve the efficiency with which fine-grained parallelism is exploited on multi-cores. We will also look for additional opportunities for speculatively optimizing code being executed by interacting threads.

3. Expected Outcomes and Results

The goal of this research is to develop a framework that will allow safe speculative parallel execution of sequential applications with complex control flow that contain parallelism but are extremely hard to parallelize using static analysis alone. In particular, we would like to focus on applications where presence of coarse-grained parallelism can be often observed via profiling techniques. The results produced by this project will be in the form of two components:

- **Programmer Assisted Speculative Parallelization.** A framework will be developed that will enable the programmer to annotate the application code with information that will allow speculative parallelization of the application. The annotations will enable the compiler and runtime system to perform speculative execution with high level of efficiency. Profiling will be used to identify segments of code as well as data structures that are critical for parallelization. The programmer will then focus all of his/her efforts on annotating the identified code and data structures.

- **Infrastructure for Speculative Execution & Optimization.** The compiler and runtime infrastructure will be developed that will encapsulate the implementation of the CorD execution model. In addition, wide range of optimizations will be developed to reduce the runtime overhead associated with copying of data between non-speculative state and speculative state as well as performing miss-speculation checks. Light-weight dynamic analysis, coupled with static analysis and user provided information, will result in the development of a speculative optimization framework. While our preliminary work was carried out using LLVM, we will work with Robert Hundt, our Google sponsor, to access the practicality of implementing our framework using gcc.

- **Evaluation using Relevant Applications.** A number of avenues will be considered for identifying applications for evaluation. First, publicly available benchmark suites will be used. Second, UCR has a strong group in data mining and applications that operate on large data sets will be obtained from this group for this work. Finally, with the assistance of our sponsor at Google, Robert Hundt and his compiler team, we will identify challenging and useful applications. Applications identified using all of the above efforts will be used to evaluate the effectiveness of the above system.

4. Budget Justification and Google Contacts

Total amount of funds being requested is $70,000. These funds will be used to support one PhD student for a period of one year ($50K per year for 1 PhD student) and to provide one month of summer salary for the PI, Rajiv Gupta ($20,000).

Robert Hundt will serve as the sponsor of this project. The results of this research work will be regularly shared with Robert Hundt and his compiler team.