Notes from the Instrumentation Specification Working Group
CScADS Workshop on Performance Tools for Petascale systems
August, 2010

The working group on "Instrumentation Specification" discussed a number of different topics. First, we reviewed the differences between the two configurable instrumenter components introduced by the talk of M.Geimer from Jülich Supercomputing Centre during the presentation sessions, a source-code instrumenter based on the TAU instrumenter and a binary instrumenter based on the Dyninst framework. Here, an immediate goal we identified was the unification of the keywords used for inserting instrumenter knowledge (e.g., the name of the function being processed) into the instrumentation code. Furthermore, adding a new keyword to provide unique identifiers (e.g., for local variable names used by the instrumentation code) was proposed.

With respect to the specification of filters and instrumentation code, the XML-based language used by the binary instrumenter was deemed more appropriate than the specification language used by the source-code instrumenter, however, a full specification of the language is not yet available. Such a specification is an essential requirement for further discussions. Moreover, we discussed whether it would be feasible to specify the instrumentation code snippets which are to be inserted in a unified, abstract programming language, for instance, similar to DynC. Although we consider this technically possible, it would require corresponding code generators for each target language in case of the source-code instrumenter.

Besides the more conceptual issues mentioned above, we also discussed a number of technical details. In this context, the question arose whether the implementation of the binary instrumenter could be simplified by providing extended processing and filtering capabilities through the Dyninst framework. Here, we discussed two different approaches: A filter interface which would iterate over all potential instrumentation points and query a user-defined predicate whether it should be added to the returned set of points, and a transform interface which would iterate over a given set of instrumentation points and insert the user-defined code snippet, potentially as an outlined function parameterized by the substitution keywords used to improve efficiency. Both options were considered useful and will be investigated further.

Finally, we discussed whether both existing instrumenters could provide better support for the Extrae measurement system used by the Paraver performance visualization and analysis tool. The main issue is that Extrae requires a mapping from function names to unique numbers. Although it would be rather trivial for the binary instrumenter to provide such a mapping (since it has a global view), it would be much more complicated for the source-code instrumenter as it would need to preserve state between multiple invocations. The use of a "state file" was proposed as a simple solution, however, this prohibits parallel builds.