Tool Support in HPC++*

Position Statement

Lars T. Hansen, Janice Cuny, Allen D. Malony,
Sameer Shende, Kurt Windisch
University of Oregon

February 26, 1996

We believe that a machine-independent tool interface to HPC++ defined within the HPC++ language definition will (1) enable users to re-use tools with different language implementations, (2) enable tool developers to focus on functionality, avoiding dependencies on language implementation specifics, and (3) enable language implementers to spend more time on the language and less on end-user tools. We therefore advocate the specification of such a tool interface.

The tool interface should be based on a model that reveals particular aspects of the language implementation in a standard way, regardless of underlying mechanisms. For example, regardless of how reductions are implemented the tool interface specification may decree that two program points–the start of the reduction and the end of the reduction–be made known to the tools. Another manner in which the implementation can be revealed in a controlled way is by providing methods in the classes for standard language objects (e.g. arrays and iterators) which provide information about those objects.

The challenge of developing the tool interface specification consists in part of providing these revelations in a way which does not inhibit program optimization while allowing good precision from the tool's perspective.

We have developed and are now testing a prototype infrastructure for control of and interaction with parallel programs. The infrastructure, called *Sneezy*, provides platform-independent support for dynamic programming tools like debuggers, profilers, visualizers, and on-line monitoring systems. It is low-level, deferring many management and analysis tasks to the tools or more properly to libraries used by the tools. It supports multiple, concurrent tools.

In order to start a discussion about tool support for HPC++, we enclose a simple "strawman" proposal for HPC++ specific tool support, along with three design notes for Sneezy (implying that we believe that our design is a suitable basis for HPC++). The strawman proposal does not cover the full range of functionality provided by Sneezy and supports only some types of tools (simple monitors and state accessors). We believe that an elabortation of even this simple specification and the construction of sample tools based on the specification will provide an understanding of how tool support can become part of the language specification, allowing us to add additional functionality to support other types of tools in the future.

The rest of this document provides an overview of Sneezy; see the enclosed design notes for more details.

*This work has been funded by ARPA under U.S. Army contract No. DABT63-94-C-0029, "Programming Environments, Compiler Technology and Runtime Systems for Object Oriented Parallel Processing". Contact {lth,malony}@cs.uoregon.edu.

Sneezy is event-based, agent-based, and minimalist. It supports the inspection and control of multi-address-space, multi-threaded programs. The agent is a virtual entity which provides a control point for both the program and the tools. Individual threads of control in the parallel program pass control explicitly to a controlling client program (the tool) by calling event signalling procedures in the agent interface; the calls can also take place implicitly, as in traditional breakpoint debugging. Events are passed from the program to the agent and from there to the client. The client remains in control of the thread and may instrument it further and inspect or change its address space. Eventually the client passes control back to the thread via the agent, which continues its computation.

The agent allows the implementation of the communication between client and program to be abstract; the client and program can be on different machines, on the same message-passing machine, on the same node, or in the same address space without needing to know this. The Sneezy implementation can optimize communication.

Sneezy is minimalist because this makes it lightweight, tractable to implement by a small team, potentially simple to port and verify, and because a maximal interface which supports all desirable functionality seems extremely hard to define. Implementations of complex but common functionality will exist as libraries available to the clients; these libraries can be portable.

Multiple co-operating clients are supported; although specified, this support is not yet implemented. Multiple clients are convenient when the user wants multiple monitors (a visualizer and a debugger, say) or even multiple debuggers: we believe that Sneezy can support replay debugging under some computation models, where a replay control module and a state-based debugger will both be present, and where it is not natural for either to be part of the other.

The initial version of Sneezy works with the pC++ language augmented with some aspects of the HPC++ computational model. See the enclosed design notes for details.