Tool Support in HPC++

Lars T. Hansen
University of Oregon

February 28, 1996

Overview of the Talk

- Position statement: a *standard* tool interface.
- Issues in constructing a tool-friendly HPC++ definition.
- Methodology.
- "Strawman" proposal.
- Action items for a plan.
- Current work in our group.
- Related work.

Position Statement

- A machine-independent tool interface to the parallel program enables construction of portable tools.
- A tool interface which is part of the language definition means every implementation must support it.
- The interface should support these broad classes of tools:
  - State accessors (e.g. visualizers).
  - Tracking tools (e.g. profilers, trace visualizers, monitors).
  - Breakpoint tools (e.g. state- and event-based debuggers).
  - Steering and control tools (e.g. load balancing, process migration, checkpointing).
- A client-server model allows abstraction of tool location w.r.t. program and its structure.
  - The "server" is the parallel program.
  - The "clients" are the tools.
  - Lightweight "server", heavyweight "clients" and standard libraries.
  - Compatible with CORBA but does not require it.

Issues of Event Visibility and Notification

How do the tools discover:

- Parallelism generated by the compiler?
- Parallelism hidden in libraries?
- Synchronization points in the program?
- Distributed data structures and their attributes?
- Structure of computation (contexts and threads)?

Even if the tools can discover these, how are the tools notified of events at program points or in data structures?

$\Rightarrow$ We need an instrumentation model.

Methodology

"Open Implementation:"

- Selected aspects of the implementation are revealed:
  - Creation and destruction of distributed data structures.
  - Attributes of distributed data structures (e.g. descriptors).
  - Program points like barriers, reductions, parallel loops.

  Revelation can be partial, read-only, or on-demand.
- Tools or even the program itself can manipulate these aspects of the implementation at run-time.

We have created a simple "strawman" proposal for HPC++ which follows this methodology.

Strawman Proposal: Overview

The current proposal is simplistic:

- Supports some, but not all, types of tools.
- Known deficiencies w.r.t. granularity.
- Ignores issues of startup/shutdown.

Proposal is based on work in our group on system for pC++.

Three parts:

- Event model.
- Command set.
- Event set.

Strawman Proposal: The event model

Client/server model.

Events occur at **locations** in **threads**.

At an event point:

1. Event+parameters+control are passed to client.
   Other threads continue computing.
2. Thread blocks in command loop.
3. Client sends commands, thread executes them.

Strawman Proposal: Commands

Commands are executed by the suspended thread.

The command set currently consists of:

- `continue`: continue execution of thread.
- `enable-event`: enable an event type globally in the thread.
- `disable-event`: disable an event type globally in the thread.
- `read-data`: read data from memory location.
- `write-data`: write data from memory location.

Strawman Proposal: Event Types

A subset of the event types:

- `begin-barrier`: thread enters barrier
- `end-barrier`
- `begin-parallel-loop`: thread enters parallel loop
- `end-parallel-loop`
- `begin-reduction`: reduction point inside parallel loop
- `end-reduction`
- `enter-function`: thread enters function
- `exit-function`
- `create-array`: distributed array is created
- `delete-array`
- `change-array`: array is changed, e.g. redistribute

Where do we go from here?

We'd like to form a **tools working group**, consisting of

- Compiler/library writers.
- Tool developers.

The goal: a tool interface specification for HPC++.

The working group should:

- Investigate possible models for the tool interface.
- Construct some tools in the context of existing languages and libraries (pC++, P++, HPF).
- Develop and maintain a spec in parallel with rest of HPC++ WG.

Current Work in Our Group

The TAU tools group at the University of Oregon has a prototype system:

- pC++/tulip augmented with aspects of the HPC++ computational model.
- Simple, fairly portable implementation.
- Working on prototype tools: visualizer, state-based debugger, event-based debugger, profiler.
- Port to Nexus likely before the summer.

Related Work

- Performance instrumentation server (Douglas Pase, IBM)
  `http://www.cs.wisc.edu/~pase/server_proposal.html`.
- Dynascope (Griffith University, Australia)
  `http://www.cit.gu.edu.au/~sosic/dynascope.html`.
- OMIS (Technische Universität München)
  `http://wwwbode.informatik.tu-muenchen.de/~omis/`.
- The `hpd` debugger (LANL)
  `http://www-c8.lanl.gov/dist_comp2/devel_tools.html`.
- Breezy (University of Oregon)
  `http://www.cs.uoregon.edu/paracomp/tau/breezy/index.html`.
- Super Symbol Table (University of Wisconsin)
  `http://www.cs.wisc.edu/~paradyn/CSTOtalk`.
- Parasight (Encore Computer Corp)
  Paper by Aral and Gertner in 1989 Workshop on Parallel and Distributed Debugging.
- The `p2d2` debugger (NASA)
  See `http://www.nas.nasa.gov/NAS/Tools/Projects/P2D2/`.
- Paradyn (University of Wisconsin)
  `http://www.cs.wisc.edu/~paradyn`.