Stack Walking Issues

Marty Itzkowitz
Sun Microsystems

Petascale Performance Tools
Snowbird, UT
July 18, 2007

Outline

- Machine-level unwind
- OpenMP unwind
- Interpreted languages
  - Java
  - Python

Machine-level Unwind

- Recognition of frames
  - Frame pointers or not
- Asynchronous-signal safety
  - Profile Trigger: signal from either clock- or HWC-profiling
- Handling corner-cases
  - From signal handlers
  - From syscall stubs
  - From hand-written assembly
  - From tricky compiler constructs
    - Non-ABI-compliant code

OpenMP Stack Walk: User-model

- Fork/Join Model
  - Master thread is always there
  - On entry to parallel computations, slave threads appear
    - During parallel computation, all threads equivalent
  - On exit from parallel region, slave threads disappear
- Actual Implementation
  - Slave threads created at first parallel region
    - Execution in so-called “mfunction”
    - Called from OpenMP support library
      - Master behavior is different from slave

OpenMP Unwind

- Reconcile master and slave callstacks
  - Callstacks seem natural, matching user model
- OpenMP Profiling API
  - Track forks; capture master stack
  - Exclude runtime frames (not interesting to users)
  - Stitch together slave stack with master at fork
- Scalability issue
  - Data volume grows with fork count, not time
  - Especially problematic with nested parallelism
    - We're working on a solution

Interpreted Languages

- Java
  - Two stacks: machine and user-Java
    - JVM gives us user-Java stack
  - Complexity of HotSpot compilation
- Python
  - Same two stacks (I presume)
    - How to get user-python stacks?