Support for Debugging Automatically Parallelized Programs

Robert Hood
rhood@nas.nasa.gov

Gabriele Jost
gjost@nas.nasa.gov

CSC/MRJ Technology Solutions
NASA AMES Research Center
Background

- Computational Intensive Applications
- Fortran, C/C++
- Migration of codes to parallel computers
- Shared memory parallelization:
  - Multithreading
  - Compiler support via directives
- Distributed memory parallelization:
  - Requires explicit message passing, e.g. MPI
- Desire to generate message passing versions of existing sequential code
The CAPTools Parallelization Support Tool

- Developed at the University of Greenwich
- Transforms existing sequential Fortran code into parallel message passing code
  - Extensive dependence analysis across statements, loop iterations, and subroutine calls.
  - Partitioning of array data
  - Generation of necessary calls to communication routines

Possible sources for errors:
- Wrong user information
- Tool makes mistake
Relative Debugging

- P1: version of a program that produces correct results.
- P2: version of the same program that produces incorrect results.
- Relative Debugging:
  - Compare data between P1 and P2 to locate the error.
  - P1 and P2 can possibly run on different machines, e.g., a sequential and a parallel architecture.
    - Applies directly to our situation.
Questions

- What data values should be compared?
  - Variables that have been determined as being incorrect and variables that define them.

- When during execution should they be compared?
  - Places where suspicious variables are defined.

- Where should data residing in multiple address space be compared?
  - Suspicious values from both executables written to file.
  - Debugger collects data from both executables.
  - Executables establish communication and compare data.

- How do we decide whether the values are correct?
  - Array checksums, element-by-element comparison, etc.

- How do we handle distributed data?
  - Array distribution information is necessary.
Main Players in the Prototype:

The CAPTools Database

- The CAPTools Database:
  - Provides **variable definition information** across subroutines to determine which variables should be checked.
  - Provides **array distribution information** to determine how distributed data should be compared against undistributed data.

Block wise distributions

CAPTools Information:

sub1: var1: CAP1_LOW:CAP1_HIGH, 1:N
sub2: var2: 1:M, CAP1_LOW:CAP2_HIGH
Main Players in the Prototype: The Comparison Routines

- The comparison routines: inserted at entry and exit of suspicious routines to bracket error location.
- compit1: Inserted in sequential program S
  - Receives data from each processor from parallel program P1, P2, ...
  - Compares data to its own:
    - checksum, partial checksums, element-by-element
  - Calls special routine if discrepancy detected.
- compit2: Inserted in parallel program.
  - Sends local data to sequential process.

```
S
...
subroutine sub1(var1)
call compit1(var1)
...
call compit1(var1)
end
```
```
P1
...
subroutine sub1(var1)
call compit2(var1)
...
call compit2(var1)
end
```
```
P2
...
subroutine sub1(var1)
call compit2(var1)
...
call compit2(var1)
end
```
Main Players in the Prototype: Instrumentation Server and P2d2

- Instrumentation Server (IS):
  - Based on dyninstAPI which was developed at the University of Maryland
  - C++ library that provides API for runtime code patching,
  - Permits insertion of calls to comparison routines into a running program

- P2d2 debugger:
  - Developed at NASA Ames Research Center
  - Portable, scalable, parallel debugger
  - Client-Server architecture based on gdb
  - P2d2 coordinates the actions of the other players and provides user Interface
A Relative Debugging Session (1)

```fortran
40-     subroutine output (phi3, nptx, npty)
        implicit none
        integer nptx, npty, i, j
        double precision phi3(0:nptx+1, 0:npty+1)
        double precision phi7(0:nptx+1, 0:npty+1)
45-     do j = 0, nptx+1
            do i = 0, npty+1
                phi7(i,j) = phi3(i,j)
            end do
        end do
50-     do j = 0, nptx+1
            write (8,*) (phi7(i,j), i = 0, npty+1)
        end do
55-     return
    end

70-     subroutine update (phi4, oldphi4)
        implicit none
        integer nptx, npty, i, j
        double precision phi4(0:nptx+1, 0:npty+1)

file: testnew2.f
```

function:variable to insert in check list:

output:phi7"
Behind the Scenes

p2d2

gdb

Instrumentation

control, process id, machine name, port number

executable name

database

NASA

Ankles Research Center
Behind the Scenes (2)

p2d2 indicates to user that difference was detected

- Run outside of debugger control
- Communicate with $S_X$

breakpoint trap if error is detected
A difference was detected in variable 'phi4' when exiting from function 'update'.
The variable had tested equal when entering function 'update'.

OK
Related Work

- **GUARD**
  - Relative Debugger for Parallel Programs
  - Developed at the Griffith University in Brisbane, Australia.
  - The debugger collects data from both executables and performs comparison.
  - Does not aim particularly at automatically parallelized programs.
  - Provides user commands like “assert” and “compare” for comparison.
  - Provides means for the user to describe array distribution.
Project Status and Future Work

- We have built a prototype of a relative debugging system for comparing serial codes and their tool produced counterparts.
  - Prototype runs on SGI Origin IRIX 6.5
- We used dynamic instrumentation to minimize comparison overhead:
  - First timing experiments were inconclusive.
- We plan to modify the p2d2 user interface to support multiple computations executing simultaneously.
- Extend prototype to handle OpenMP programs.