The Distributed Application Debugger (DAD)
Introduction

- The Distributed Application Debugger is a debugging tool for parallel programs
- Targets the MPI platform
- Runs remotely even on private networks
- Has record and replay features.
- Integrates GDB
Results from survey of students learning parallel programming concluded 3 things:

1. Sequential errors are still frequent
2. Message errors are time consuming
3. Print statements are still used for debugging
Survey results categorized according to the domains of multilevel debugging

- Sequential errors
- Message errors
- Protocol errors

In addition to

- Data decomposition errors
- Functional decomposition errors
Survey Results

[Bar chart showing various categories such as Data Decomposition, Functional Decomposition, Sequential Error, Message Error, Protocol Error, and Other Error with different percentages represented by bars for different tasks like Mandelbrot, Differential Equation Solver, Pipeline Computation, Partial Sum, Equation Solver, and Matrix Multiplication.]
## Survey Results

<table>
<thead>
<tr>
<th></th>
<th>Data Decomp.</th>
<th>Functional Decomp.</th>
<th>Sequential Error</th>
<th>Message Error</th>
<th>Protocol Error</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Time</td>
<td>19.9</td>
<td>68.1</td>
<td>24.3</td>
<td>61.4</td>
<td>50.0</td>
<td>30.6</td>
</tr>
<tr>
<td>Total Time Spent</td>
<td>278</td>
<td>545</td>
<td>1,846</td>
<td>1,536</td>
<td>451</td>
<td>545</td>
</tr>
<tr>
<td># Errors</td>
<td>14</td>
<td>8</td>
<td>76</td>
<td>25</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Total time Spent in %</td>
<td>5.67%</td>
<td>11.12%</td>
<td>37.67%</td>
<td>31.34%</td>
<td>9.20%</td>
<td>5.0%</td>
</tr>
</tbody>
</table>
The Components

- **The Client**
  - The GUI interacting with the programmer

- **The Call Center**
  - A central messaging hub (running on the cluster) for
    - Routing messages from the MPI program to The Client
    - Routing commands from The Client to the MPI program

- **Bridges**
  - A relay application for passing data between The Client and The Call Center, when The Call Center is not directly accessible (cluster behind firewall)

- **The Runtime**
  - A libraries with wrapper code for the MPI functions (talks to The Call Center)
Login from Home to Cluster not directly possible.
The Setup

- Client runs at home
- Bridges on the servers in between home and the cluster
- Call Center on the cluster
- MPI processes on the cluster
The Distributed Application Debugger

![Image of the Distributed Application Debugger interface]

- *Toolbar*
- *Node Panels*
- *SSH Configuration*
  - **Configurations**
    - Remote Computer: New Connection
    - User Name:
    - Password:
    - Transfer Directory:
    - Connection Port:
The user provides a connection path and credentials on all machines.
The user provides a connection path and credentials on all machines.
The system initiates SSH connections to each configured computer and launches a Bridge or The Call Center.
Each component then connects to each other via TCP.
Connection Initiated
Include a special mpi.h header file
- MPI calls are caught by wrapper functions
- Upon start up, each node creates a callback connection to The Call Center
- Data passed to MPI functions is sent back.
MPI Code Redirected

```c
#include "mpi.h"

int main(int argc, char *argv[]){
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myId);

    if(myId == 0){
        //This is the master node.
        //Send the whole buffer to the middle index
        MPI_Send(transferBuffer, numSlaves, MPI_INT, startingNode, numSlaves, MPI_COMM_WORLD);

        //Sync and then distribute the initial values
        MPI_Barrier(MPI_COMM_WORLD);

        //Wait for the result
        MPI_Recv(&finalResult, 1, MPI_INT, numSlaves, TAG, MPI_COMM_WORLD, &status);

        //Send result and wait for everyone to get it
        MPI_Send(&finalResult, 1, MPI_INT, startingNode, TAG, MPI_COMM_WORLD);
    }

    else{
        //Distribute and process the partial sums
    }
}
```

```c
#include <mpi.h>
#include "debug.h"
#include "mpidebug.h"

int main(int argc, char *argv[]){
    _MPI_Init(&argc,&argv);
    _MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
    _MPI_Comm_rank(MPI_COMM_WORLD,&myId);

    if(myId == 0){
        //This is the master node.
        //Send the whole buffer to the middle index
        _MPI_Send(transferBuffer, numSlaves, MPI_INT, startingNode, numSlaves, MPI_COMM_WORLD);

        //Sync and then distribute the intial values
        _MPI_Barrier(MPI_COMM_WORLD);

        //Wait for the result
        _MPI_Recv(&finalResult, 1, MPI_INT, numSlaves, TAG, MPI_COMM_WORLD, &status);

        //Send result and wait for everyone to get it
        _MPI_Send(&finalResult, 1, MPI_INT, startingNode, TAG, MPI_COMM_WORLD);
    }

    else{
        //Distribute and process the partial sums
    }
}
```
The Connected System
The Connected System
An MPI session can be run in 3 modes:

- **Play**
  - Just run like regular MPI

- **Record (Record all messages)**
  - Record all messages

- **Replay**
  - Use recorded messages to play back
The Runtime behaves like regular MPI
- Nothing is saved to disk
- Nothing is read from disk
- Messages and parameters ARE sent back to The Client
Session Modes (Record)

- The Runtime
  - Saves messages and parameters to a log file
  - Executes the actual MPI call
  - Saves the result
Session Modes (Replay)

- The Runtime does not execute any real MPI calls.
  - All data is supplied from log files.
  - No actual communication takes place
  - Guarantees the same run as when the log file was recorded
Mixed mode is special

- Some processes execute real MPI calls
- Some replay from log file
  - Sometimes its necessary to execute MPI calls if communicating with someone who is executing real MPI calls; E.g. to avoid buffer overflow
  - Validation is done on real values and log file values
Debugging Data

- The Runtime sends back 2 debugging messages per MPI command
  - A *PRE* message indicating that an MPI command is about to be executed
  - A *POST* message indicating that an MPI command completed
- Console messages are routed per node to the appropriate window.
Analyzing Data

- Debugging data gets displayed within the Console, Messages, or MPI tabs.
The Console Tab displays anything that the user’s code wrote to stdout.
The Messages Tab displays messages as they come. Matches Send/Receive pairs between nodes. Messages without a corresponding Send or Receive message get highlighted in red.
The MPI tab displays all MPI commands
  • in the order they were executed
  • along with their parameters.

Commands statuses (success, fail, or blocked) are displayed with icons in the Status Column.
Analyzing Data
Buffer values can be requested and inspected.
Analyzing Data
Attaching GDB

- GDB can be attached to any node and controlled with the GDB Control Panel.
Attaching GDB
Attaching GDB
The source code to The Distributed Application Debugger can be found on GitHub at:

https://github.com/mjones112000/DistributedApplicationDebugger
Questions??