Reports for Conversion of MPJ Benchmark Programs

(Written by Vivian Chan - June 6, 2004)

Introduction

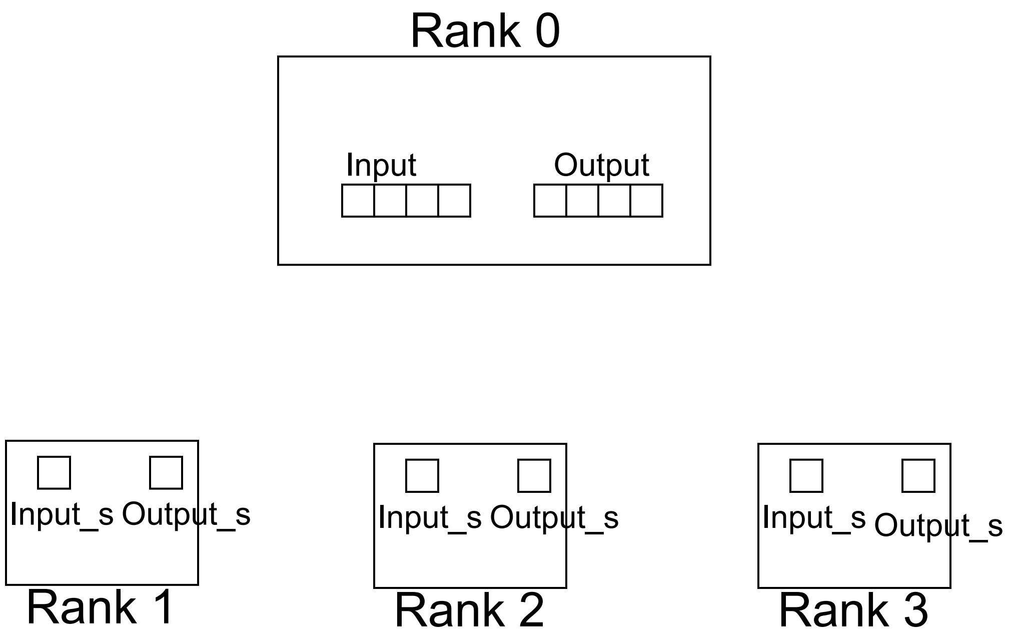
This research classes requires conversion of three MPJ benchmark programs. They are
1. JGFSeriesBench,
2. JGFLuFactBench, and
3. JGFRayTracerBench.

The original requirement was to convert four programs, which includes the conversion of JGFMonteCarloBench. Since JGFMonteCarloBench requires serialization of class objects, the conversion was not successful at this point. Additional functionalities are needed in the GridTcp programs to serialize class objects. The other three programs are converted successfully in terms of the user program level.

The conversion of each program involves a two step process. The first process is to convert the JGF bench that is using MPI to use Java Socket. The second process is to convert the Java Socket program to use the Grid Tcp Socket program. The rest of the programs that involves no communication process have no need to convert.

Implementation

The implementation of the conversion is based on the sample of an already converted program, JGFCryptBench. All three converted programs are based on the same implementation as follows.



Rank 0 node will have two arrays of sockets. The input[i] will get the input stream of rank i node. The output[i] will get the output stream of rank i node. In other words, if rank 0 wants to read data that was sent from rank 1, rank 0 would read the data from input[1]. If rank 0 wants to write some data to other ranks, it sends the data through output[i] where i is the rank of the destination node.

On the other hand, if the nodes other than rank 0 want to read data from rank 0, it reads the data through input_s. If they want to write data to rank 0, they write the data through output_s.

<u>From JGF to Java Socket</u>

When converting to Java Socket version, all of the above java sockets are initialized. The sockets of rank 0 are java sockets while the sockets of the other nodes are java server sockets. Then the effort is focused on how to convert the MPI calls to java socket communications. During reads, the nodes will read a stream of bytes from the corresponding sockets and the bytes will be reconstructed into the expected object. When writes, the nodes will convert the sending objects into to a stream of byte and send it through the corresponding sockets.

<u>From Java Socket to GridTcp Socket</u>

When converting to GridTcp Sockets, the sockets are initialized to use GripTcp sockets and server sockets. Furthermore, declarations of the four variables are required. The declarations are as follows.

```
int myRank;
GridTcp tcp;
GridIpEntry ipTable[];
int funcId;
```

These variables are automatically initialized the User Program Wrapper. Then, the names of the functions in the program will be converted into some function numbers. The following shows an example.

```
int main(){
    x = x + 1;

    someFunctionA();

    anotherFunctionB();

    lastFunctionC();
}

someFunctionA(){
}

anotherFunctionB(){

}

lastFunctionC(){

}
```

Assume above is the Java Socket version, the GridTcp version should look like the following.

```
func_0(){       //correspond to main
    x = x + 1;

    return -1;
}

func_1(){       //correspond to someFunctionA

    return -1;
}

func_2(){
    return -1;
}

func_3(){
    tcp.disconnect();
    return -2;
}
```

When a function returns -1, it tells the User Program Wrapper to continue to the next function by increasing the function id. If it returns -2, it signals the User Program Wrapper to terminate the user program. Before terminating the program tcp.disconnect needs to be called.

Since the GridTcp Socket inherited from the java socket, the code for communicating among nodes will remain the same as the java socket version.

<u>Series Implementation</u>

The communications among the nodes are taken place at the kernel() function. Rank 0 is receiving double data from other nodes. The JGF version uses Ssend and Recv to transfer the double data to rank 0 node from other nodes.

The sending nodes convert the double data into bytes and write the bytes to the output_s socket. Since the original implementation is using non-blocking send, the receiving node, rank 0, will use a buffered input stream to read the data through input[i] where i is the rank of a sending node.

LuFact Implementation

The communications among the nodes are taken place at the dgefa(). In that function, each node will have a chance to broadcast some double and integer data to the other nodes. Then rank 0 receives a vector data from the other nodes.

Based on the design, the communications are running from rank 0 to other ranks and vice versa. Communication cannot take place from rank 1 to rank 2 etc. Therefore, in implementing broadcast, if the source node is not rank 0, the source node will need to send the data to rank 0 and then rank 0 would send the data to all the other nodes.

For sending and receiving vector data, the same implementation of Series is used. The sending nodes converts the vector data into byte stream and send it through the output_s socket. Since blocking send is used this time, therefore, rank 0 is not using the buffer input stream instead it is only using the object input stream.

Ray Tracer Implementation

The communications among the nodes are taken place at the render() function in RayTracer.java file. The original implementation uses MPI reduce function and send/recv functions.

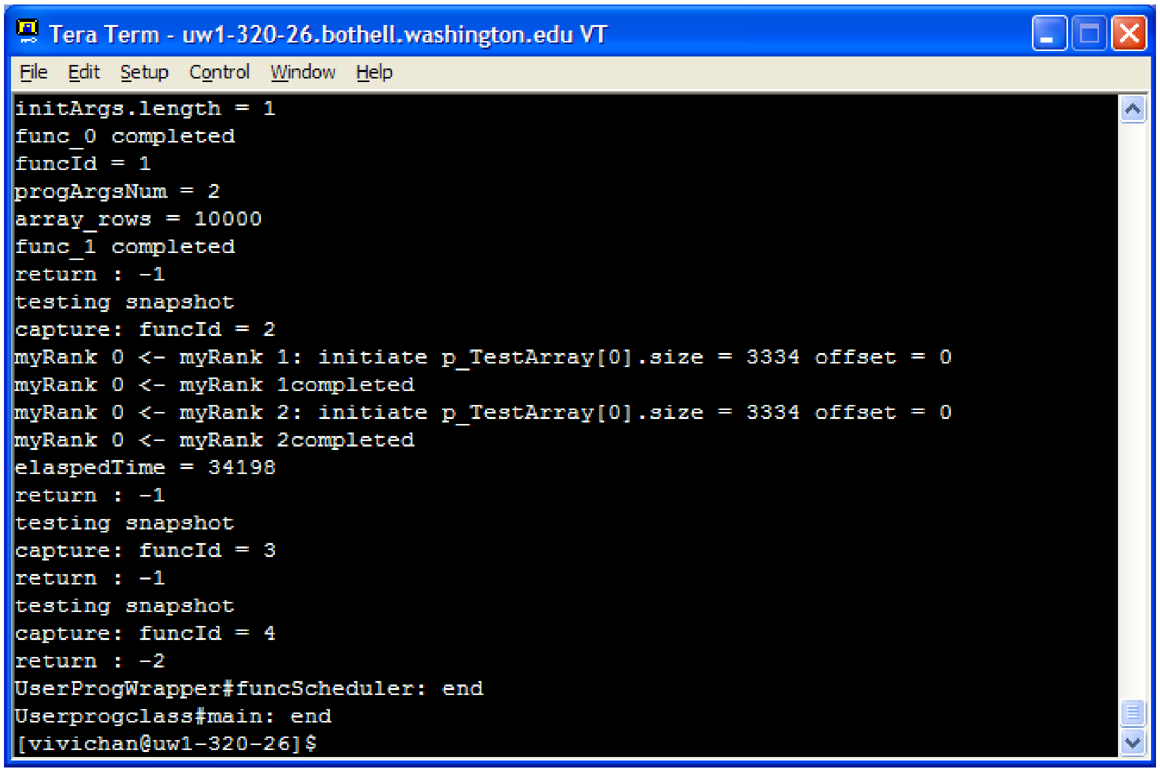
Nodes other than rank 0 are sending a double number to rank 0 to calculate the sum of those numbers. When the calculation is done, the sum is sent from rank 0 to the other nodes.

Basically, the MPI reduce function works the same way as rank 0 making individual connection to the other nodes to get the data. Therefore, the implementation is the same as implementing send and recv functions, i.e. the sending nodes converts the data into byte stream and write it to output_s. Then, rank 0 reads the object input stream through input[i] where i is the rank of a sending node.

Execution Output

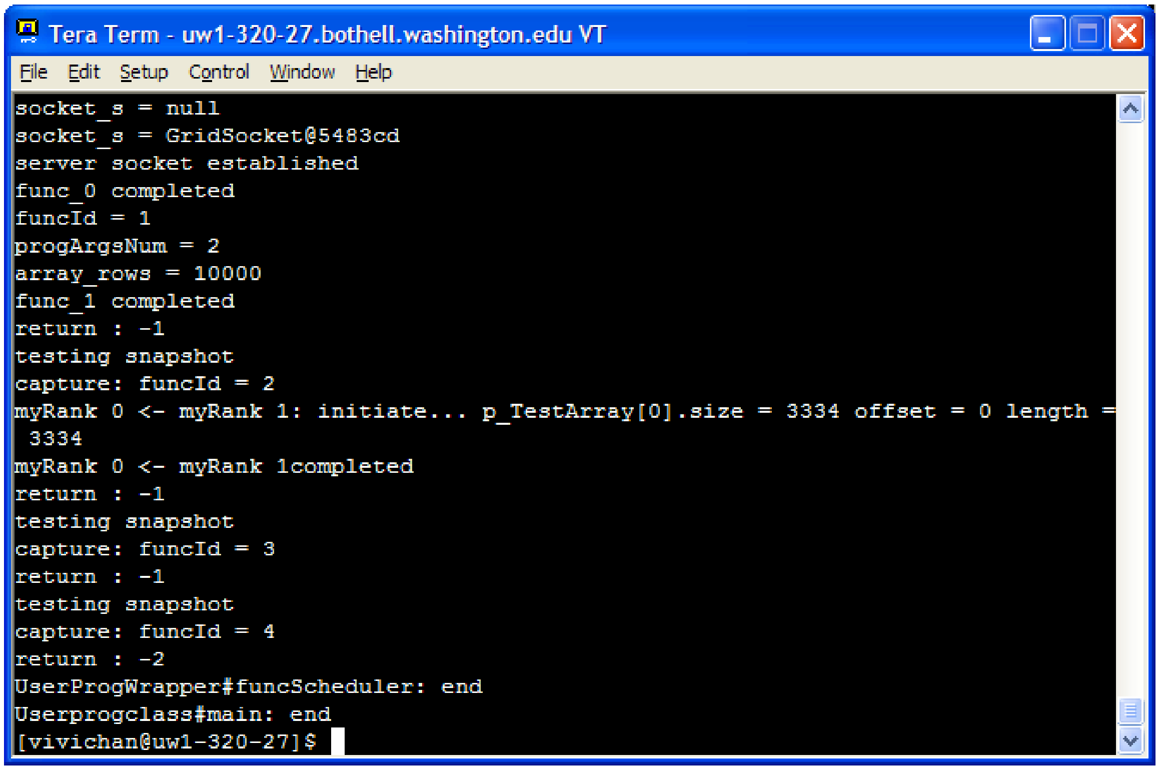
Series

Rank 0:


Tera Term - uw1-320-26.bothell.washington.edu VT

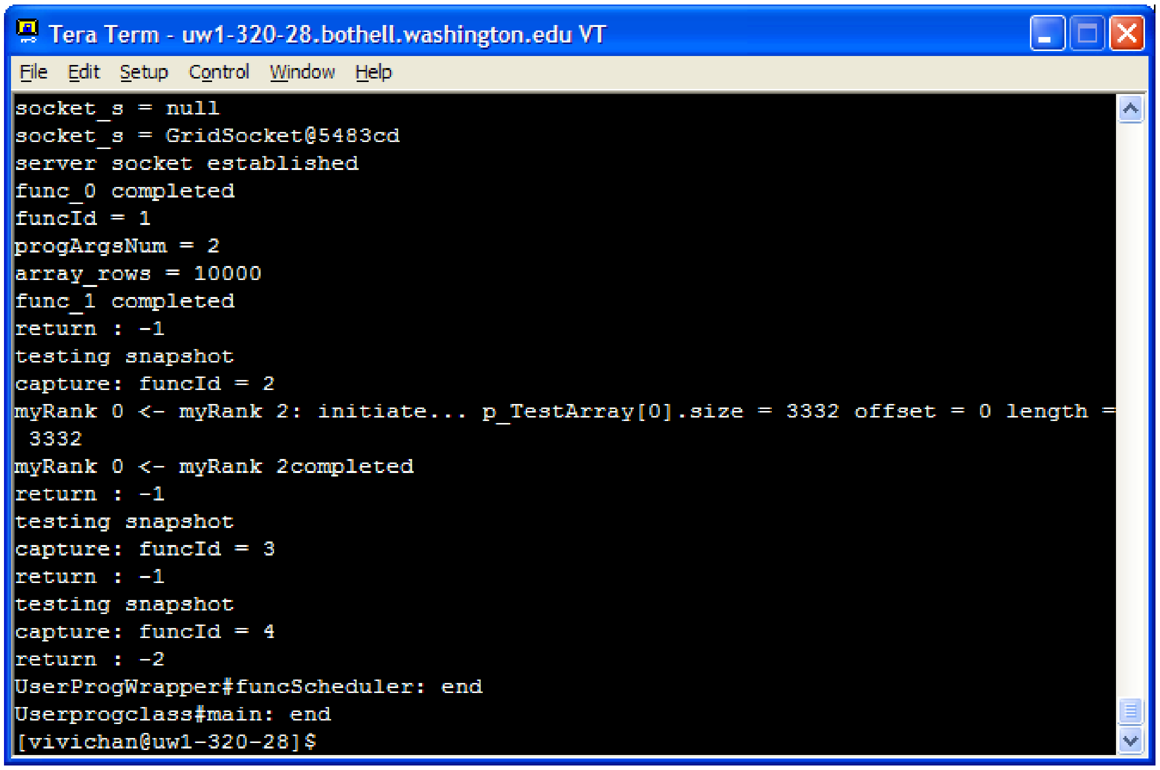
```
initArgs.length = 1
func_0 completed
funcId = 1
progArgsNum = 2
array_rows = 10000
func_1 completed
return : -1
testing snapshot
capture: funcId = 2
myRank 0 <- myRank 1: initiate p_TestArray[0].size = 3334 offset = 0
myRank 0 <- myRank 1completed
myRank 0 <- myRank 2: initiate p_TestArray[0].size = 3334 offset = 0
myRank 0 <- myRank 2completed
elaspedTime = 34198
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-26]$
```

Rank 1:


Tera Term - uw1-320-27.bothell.washington.edu VT

```
socket_s = null
socket_s = GridSocket@5483cd
server socket established
func_0 completed
funcId = 1
progArgsNum = 2
array_rows = 10000
func_1 completed
return : -1
testing snapshot
capture: funcId = 2
myRank 0 <- myRank 1: initiate... p_TestArray[0].size = 3334 offset = 0 length =
 3334
myRank 0 <- myRank 1completed
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-27]$
```

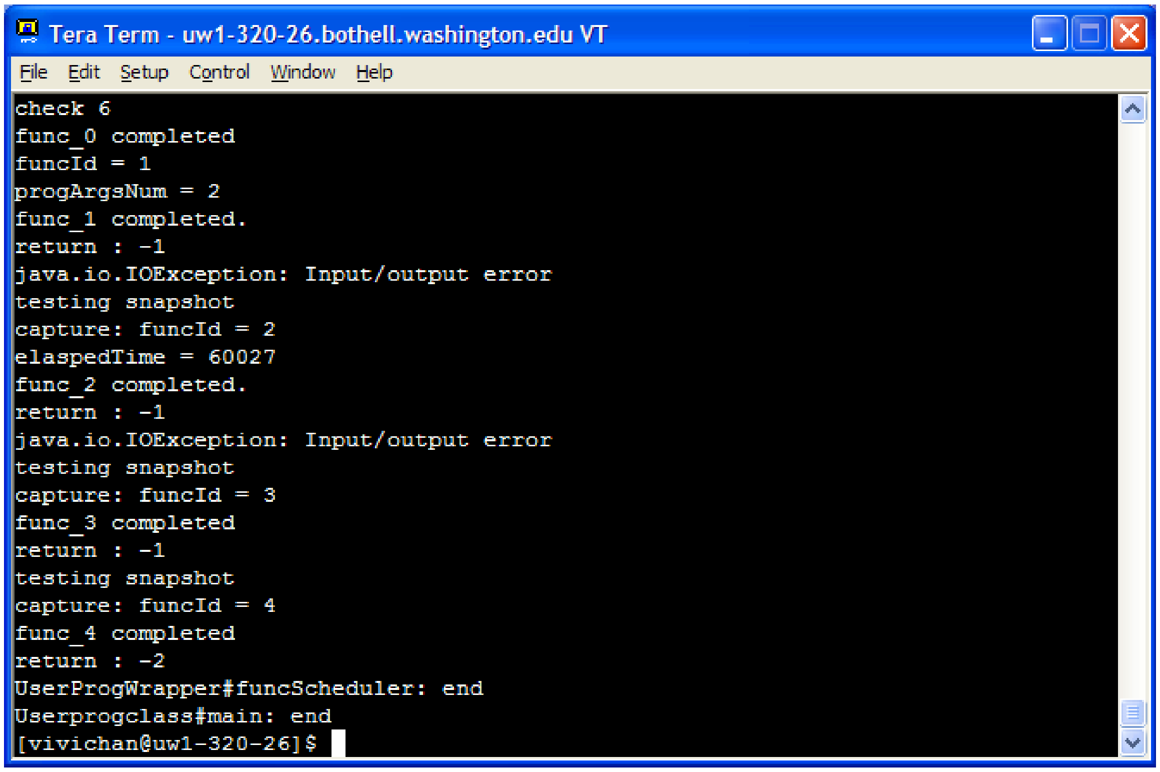
Rank 2:


Tera Term - uw1-320-28.bothell.washington.edu VT
File Edit Setup Control Window Help

```
socket_s = null
socket_s = GridSocket@5483cd
server socket established
func_0 completed
funcId = 1
progArgsNum = 2
array_rows = 10000
func_1 completed
return : -1
testing snapshot
capture: funcId = 2
myRank 0 <- myRank 2: initiate... p_TestArray[0].size = 3332 offset = 0 length =
 3332
myRank 0 <- myRank 2completed
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-28]$
```

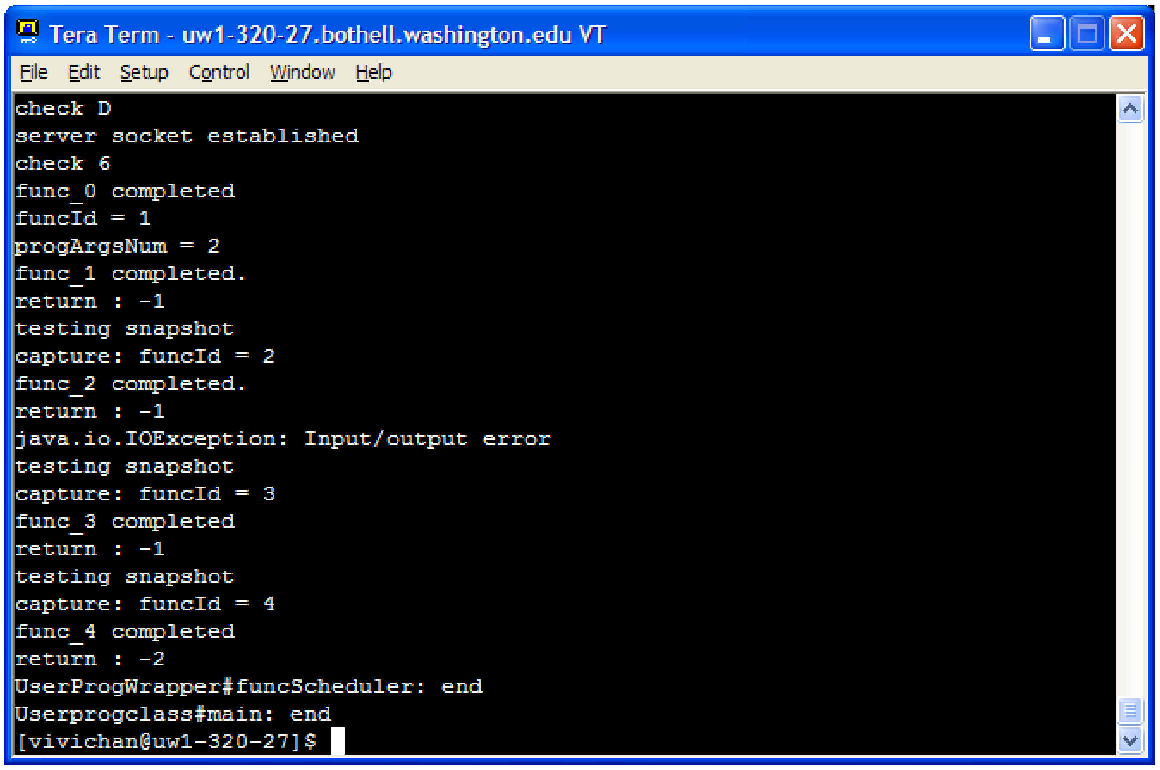
<u>Lufact</u>

Rank 0:


Tera Term - uw1-320-26.bothell.washington.edu VT
File Edit Setup Control Window Help

```
check 6
func_0 completed
funcId = 1
progArgsNum = 2
func_1 completed.
return : -1
java.io.IOException: Input/output error
testing snapshot
capture: funcId = 2
elaspedTime = 60027
func_2 completed.
return : -1
java.io.IOException: Input/output error
testing snapshot
capture: funcId = 3
func_3 completed
return : -1
testing snapshot
capture: funcId = 4
func_4 completed
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-26]$
```

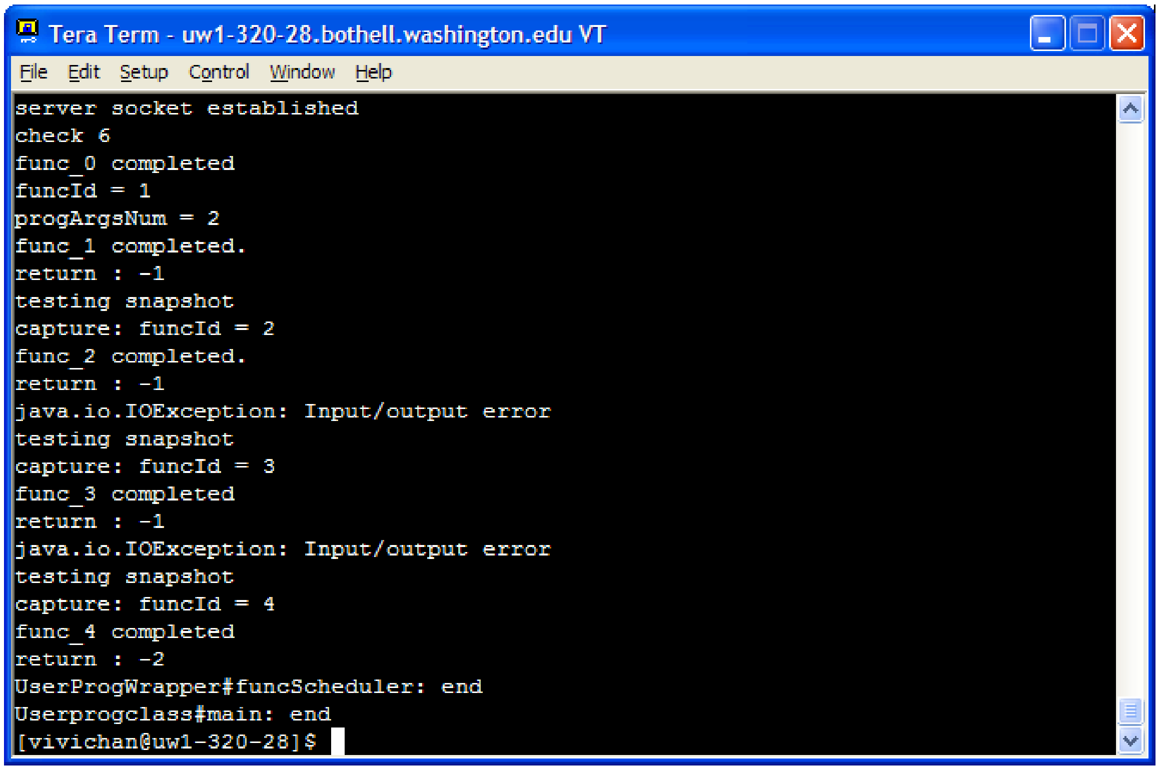
Rank 1:



Tera Term - uw1-320-27.bothell.washington.edu VT

```
check D
server socket established
check 6
func_0 completed
funcId = 1
progArgsNum = 2
func_1 completed.
return : -1
testing snapshot
capture: funcId = 2
func_2 completed.
return : -1
java.io.IOException: Input/output error
testing snapshot
capture: funcId = 3
func_3 completed
return : -1
testing snapshot
capture: funcId = 4
func_4 completed
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-27]$
```

Rank 2:

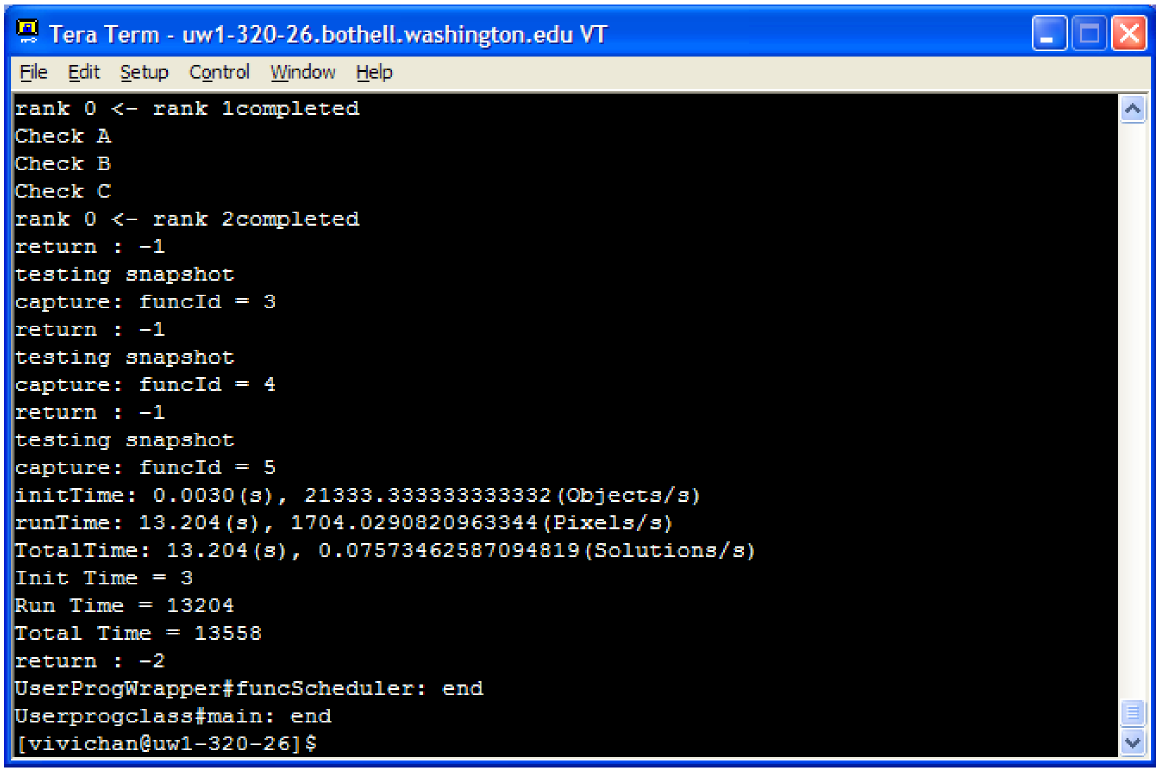


Tera Term - uw1-320-28.bothell.washington.edu VT

```
server socket established
check 6
func_0 completed
funcId = 1
progArgsNum = 2
func_1 completed.
return : -1
testing snapshot
capture: funcId = 2
func_2 completed.
return : -1
java.io.IOException: Input/output error
testing snapshot
capture: funcId = 3
func_3 completed
return : -1
java.io.IOException: Input/output error
testing snapshot
capture: funcId = 4
func_4 completed
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-28]$
```

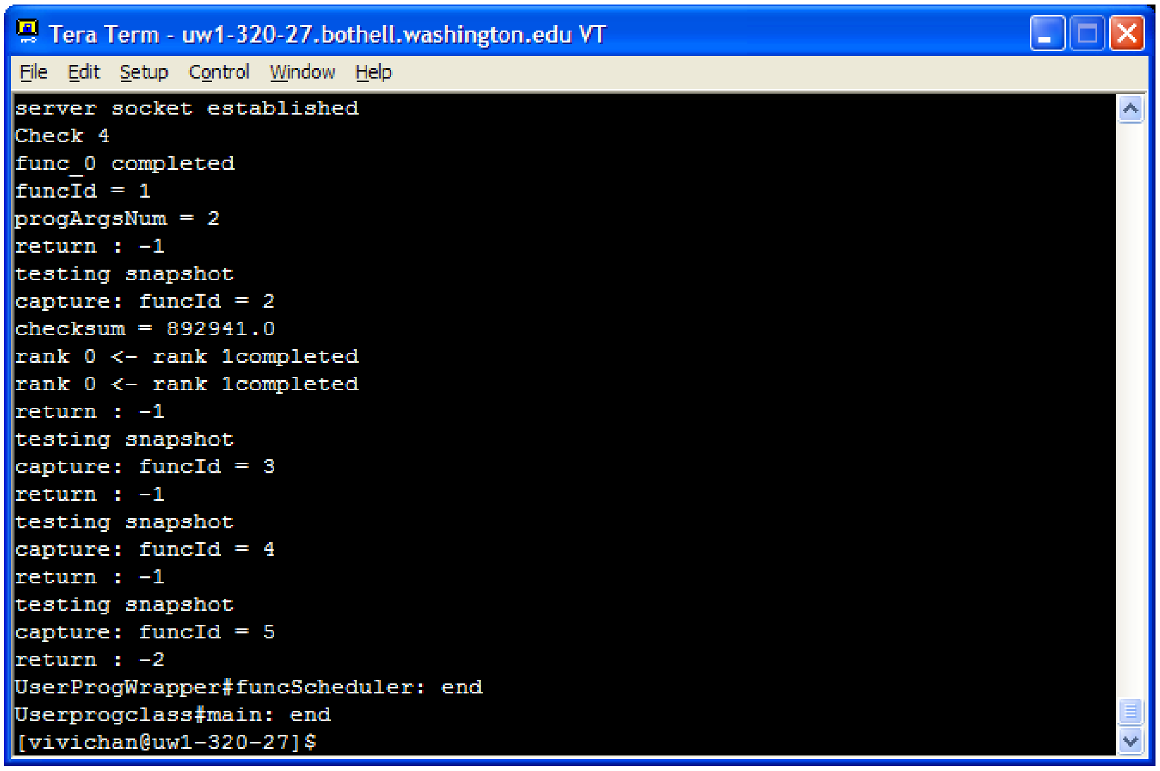
Ray Tracer

Rank 0:


Tera Term - uw1-320-26.bothell.washington.edu VT
File Edit Setup Control Window Help

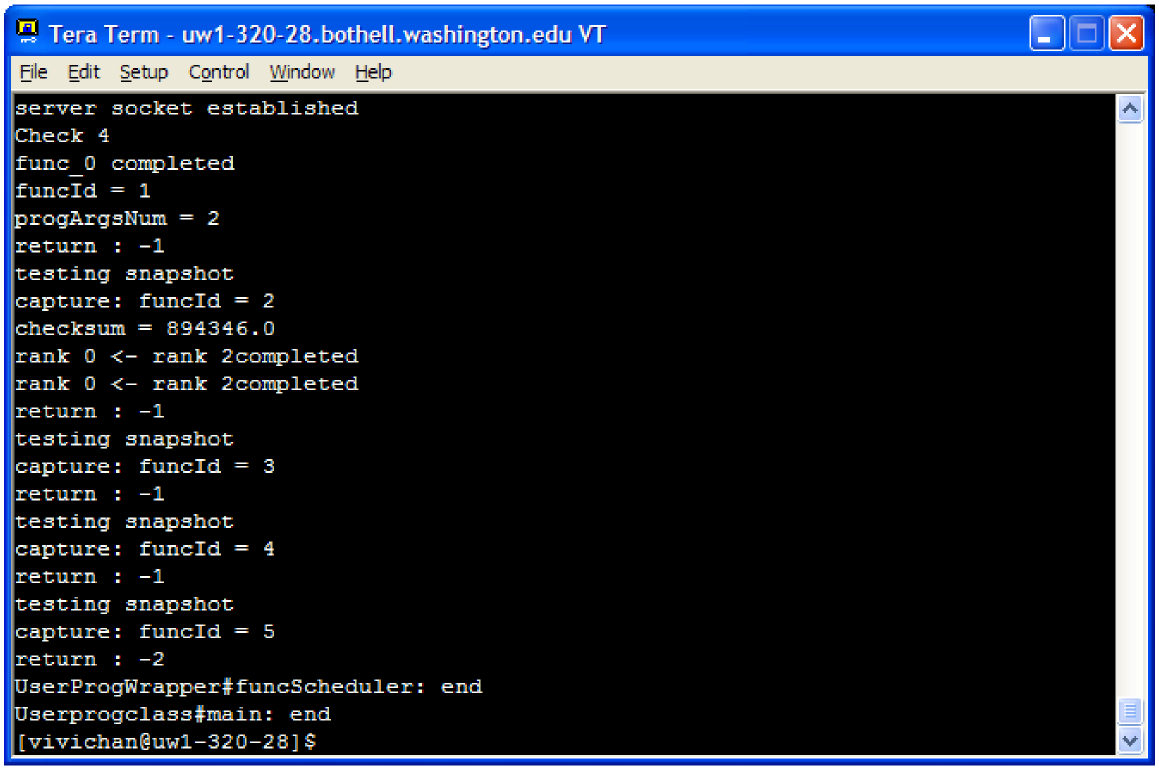
```
rank 0 <- rank 1completed
Check A
Check B
Check C
rank 0 <- rank 2completed
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -1
testing snapshot
capture: funcId = 5
initTime: 0.0030(s), 21333.333333333332(Objects/s)
runTime: 13.204(s), 1704.0290820963344(Pixels/s)
TotalTime: 13.204(s), 0.07573462587094819(Solutions/s)
Init Time = 3
Run Time = 13204
Total Time = 13558
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-26]$
```

Rank 1:


Tera Term - uw1-320-27.bothell.washington.edu VT
File Edit Setup Control Window Help

```
server socket established
Check 4
func_0 completed
funcId = 1
progArgsNum = 2
return : -1
testing snapshot
capture: funcId = 2
checksum = 892941.0
rank 0 <- rank 1completed
rank 0 <- rank 1completed
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -1
testing snapshot
capture: funcId = 5
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-27]$
```

Rank 2:


Tera Term - uw1-320-28.bothell.washington.edu VT
File Edit Setup Control Window Help

```
server socket established
Check 4
func_0 completed
funcId = 1
progArgsNum = 2
return : -1
testing snapshot
capture: funcId = 2
checksum = 894346.0
rank 0 <- rank 2completed
rank 0 <- rank 2completed
return : -1
testing snapshot
capture: funcId = 3
return : -1
testing snapshot
capture: funcId = 4
return : -1
testing snapshot
capture: funcId = 5
return : -2
UserProgWrapper#funcScheduler: end
Userprogclass#main: end
[vivichan@uw1-320-28]$
```

Results Evaluation

Both Series and Ray Tracer are completed with no exceptions thrown. Lufact test has exceptions thrown. The exception is not thrown from the user program. However, when Koichi Kashiwagi ran the Lufact test, no exceptions are thrown. Therefore, this could be an inconsistent error. The exception is stated as “java.io.IOException: Input/ output error”.

Instructions to Run the Test

Java Socket programs

To test the version of java socket programs, the rank 0 node should run last. For example, if the test will use three machines, rank 1 and rank 2 nodes should run the program before rank 0.

To run the java socket version, enter the following command.

```
java yourProgName #triplets rank programArgs
```

Series

The valid program arguments of series are a, b or c, which specifies the size of the data.

Rank 0:

java Series 2 0 a

Rank 1:
java Series 2 1 a

Lufact

The valid program arguments of Lufact are a b or c, which specifies the size of the data.

Rank 0:
java Lufact 2 1 b

Rank 1:
java Lufact 2 1 b

RayTracer
Ray Tracer requires no program argument.

Rank 0:
java RayTracerBenchSizeA 2 0

Rank 1:
java RayTracerBenchSizeA 2 1

GridTcp Socket programs

To run test on GridTcp Socket benchmark programs, there is no need to run the program in the order of the rank. The program can be started in different rank order. However, in order to run the test, the GridTcp.jar and UserProgWrapper.class files are needed to be present in the current directory.

In general, the command to run the tests is as follows.

java –cp GridTcp.jar:. UserProgWrapper – #triplets myrank ip – [yourrank ip -] yourProgName [args]

Assume that the following examples are using uw1-320-20 and uw1-320-21 nodes to run the test.

Series

The valid arguments are a, b or c.

Rank 0:
java –cp GridTcp.jar:. UserProgWrapper – 2 0 uw1-320-20 - 1 uw1-320-21 – GridTcpSeries 2 a

Rank 1:
java –cp GridTcp.jar:. UserProgWrapper – 2 1 uw1-320-21 - 0 uw1-320-20 – GridTcpSeries 2 a

<u>Lufact</u>
The valid arguments are a, b or c.

Rank 0:
java –cp GridTcp.jar:. UserProgWrapper – 2 0 uw1-320-20 - 1 uw1-320-21 – GridTcpLufact 2 b

Rank 1:
java –cp GridTcp.jar:. UserProgWrapper – 2 1 uw1-320-21 - 0 uw1-320-20 – GridTcpLufact 2 b

<u>Ray Tracer</u>
The valid arguments can be any number. However, the recommended arguments are 0 or 1.

Rank 0:
java –cp GridTcp.jar:. UserProgWrapper – 2 0 uw1-320-20 - 1 uw1-320-21 – GridTcpRayTracer 2 0

Rank 1:
java –cp GridTcp.jar:. UserProgWrapper – 2 1 uw1-320-21 - 0 uw1-320-20 – GridTcpRayTracer 2 0

File Storage

The converted programs are saved in medusa machine at the path of /home/uwagent/MA/ChanV. Each benchmark program has its own directory. The java versions are saved under the java directory while the GridTcp versions are saved under GridTcp directory.