

CSCI-4320/6340 Assignment 1: Message Passing Interface

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590

January 23, 2009

DUE DATE: 12 p.m/Noon, Friday, February 6th

1 Description

For this assignment you will be creating a timing test program for MPI. In particular, you will write a series of functions where each function will determine the **MAX average**, **MIN average** and **AVERAGE average** execution time in microseconds of a particular MPI routine across all the MPI tasks, where each MPI task would have exercised a particular MPI call in a loop, L times. The following routines (or pairs of routines) are the ones you must measure the performance of for this assignment.

- **MPI_Send** and **MPI_recv**: Here, you will pair-up the MPI tasks according to the following pattern, Rank 0 pairs with Rank $N - 1$, Rank 1 pairs with Rank $N - 2$, Rank 2 pairs with Rank $N - 3$ and so on until Rank $N/2 - 1$ pairs with Rank $N/2$. In a loop, each pair of tasks will first send a message of size M , where M is an input parameter and then receive the message that the other task just sent. The loop will iterate L times (where L is also an input parameter). You will time the whole loop, compute the local average time and then determine across all processors the MAX average, MIN average and AVERAGE average cycle times. I will give you the cycle timer code for an x86 system. The MAX, MIN and AVERAGE can be computed by doing a **MPI_Allreduce**.
- **MPI_Isend** and **MPI_Irecv**: Do the same as you did for the previous **MPI_Send** and **MPI_Recv**.

- **MPI_Scan:** Compute the prefix sum over the ranks of all MPI tasks. Do this L times in a loop like before and then use the allreduce to compute the MAX average, MIN average and AVERAGE average cycle times across all MPI tasks.
- **MPI_Gather:** Here, each task will send it rank number, which is then gather/received by the target MPI task 0 (ZERO) into an array of ranks. Do this L times and compute the statistics as before.
- **MPI_Scatter:** Here, the source MPI task 0 (ZERO) will re-distribute the previously gathered rank values to the nodes that sent them. As before, do this L time and compute the statistics.
- **MPI_Barrier:** Here, have each task enter the barrier and do it L times in the loop, then compute the usual statistics.
- **MPI_Allreduce:** Here, generate a random floating point value (double) using `drand48()`. Use the Allreduce routine to determine what was the MIN random value among all MPI tasks. Do this L times and compute the statistics.

2 Timing Code

The following is a routine that you would include as “rdtsc.h”. RDTSC stands for the “read time-stamp counter” and it is an x86 or PowerPC assembly language instruction that returns a 64 bit number that is the number of cycles this machine is processed since the last boot-time. The C code for this is:

```
#ifndef __RDTSC_H_DEFINED__
#define __RDTSC_H_DEFINED__

#if defined(__i386__)

static __inline__ unsigned long long rdtsc(void)
{
    unsigned long long int x;
    __asm__ volatile (".byte 0x0f, 0x31" : "=A" (x));
    return x;
}

#elif defined(__x86_64__)
```

```

// typedef unsigned long long int unsigned long long;

static __inline__ unsigned long long rdtsc(void)
{
    unsigned hi, lo;
    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
    return ( (unsigned long long)lo)|(( (unsigned long long)hi)<<32 );
}

#elif defined(__powerpc__)

// typedef unsigned long long int unsigned long long;

static __inline__ unsigned long long rdtsc(void)
{
    unsigned long long int result=0;
    unsigned long int upper, lower,tmp;
    __asm__ volatile(
        "0:                                \n"
        "\tmftbu    %0                    \n"
        "\tmftb     %1                    \n"
        "\tmftbu    %2                    \n"
        "\tcmpw     %2,%0                  \n"
        "\tbn     0b                        \n"
        : "=r"(upper), "=r"(lower), "=r"(tmp)
        );
    result = upper;
    result = result<<32;
    result = result|lower;

    return(result);
}

#else

#error "No tick counter is available!"

#endif

```

```
#endif
```

Use the above macro `rdtsc`. To do things like:

```
unsigned long long start_time = 0;
unsigned long long finish_time = 0;
unsigned long long total_time = 0;

rdtsc( start_time );

for( i; i < MAX_WHATEVER; i++ )
{
    DO TEST
}

rdtsc( finish_time );

total_time = finish_time - start_time;
```

Note, I'll place a copy of of this in `rdtsc.h` on the Class website for you to download.

3 HAND-IN INSTRUCTIONS

Using the CS cluster, you will need to run your tests over the following configurations. Note, the CPU speed of the CS cluster Opteron processors is 2.0 GHz. To translate the cycle counts into microseconds you will need to divide the number of cycles by 2000.0. However, don't convert to microseconds until the performance test is complete as you will perturb your results with the overhead of the floating point division operations.

- **Number of Processors:** 2, 4, 8 processors. If the Blue Gene accounts are available, then run your test up to 128 processors.
- ***L* Loop Iterations:** 1024, 8192, 65536
- ***M* message sizes:** 4, 16, 64, 256, 1024, 4096 bytes for the MPI send/recv and isend/irecv routines only.

Tabulate this data into a table using LaTeX or MSWord. Attach a printed version of you code and place a copy in your account on my office machine and let me know where I can find it.

Note, you'll need modify your path to get MPI to work correctly with the following bash command line:

```
export PATH=/cs/chris/MPI/mpich-1.2.7p1/bin:$PATH
```