HPC Exercice 1: MPI ping

Marko Grönroos

21. lokakuuta 1999

1 ping.cc

My implementation of ping uses a home-made base class library (Strings, etc), and a quickly made C++ coating for MPI calls. The coating is just a test, and would probably do better if implemented on top of streams.

```
#include "mpi++.h"
#include <stdio.h>
#include <applic.h>
#include <Math.h>

#define MAX_SIZE 1000000

Main () {
        MPIInstance mpi (mArgc, mArgv);

        // Reread application parameters and put the into variables
        int msgMinlen    = mParams[0];
        int msgIncrement = mParams[1];
        int msgMaxlen    = mParams[2];
        int loops = mParamMap["loops"];

        Vector record ((msgMaxlen-msgMinlen)/msgIncrement+1);
        for (int i=0; i<record.size; i++)
                record[i] = 0.0;

        // Create a buffer, and allocate enough space for it.
        String buffer;
        buffer.reserve (msgMaxlen);

        if (mpi.getRank() == 0) {

                // Sender process

                printf ("Sender iterating for %d...%d +=%d (%d values)\n",
                                msgMinlen, msgMaxlen, msgIncrement, record.size);

                for (int l=0; l<loops; l++) {
                        fprintf (stderr, "\r% 5d\r", l);
                        // Send packets of growing sizes
                        for (int s=msgMinlen; s<=msgMaxlen; s+=msgIncrement) {
                                buffer.len = s; // Force the buffer size; don't care about contents

                                // Bounce the buffer from Echo, and measure time
                                double t1 = mpi.time();
                                mpi.send (buffer, 1);
                                mpi.recv (buffer, buffer.len, 1);
                                double t2 = mpi.time();

                                // Record time difference in milliseconds
                                record[(s-msgMinlen)/msgIncrement] += (t2-t1)*1000/2;
                        }
```

```
                }

                // Display statistics
                for (int i=0; i<record.size; i++)
                        printf ("%d\t%1.6f\n", msgMinlen+i*msgIncrement, record[i]/double(loops));

                // Send stop message to the Echo
                mpi.send ("stop", 1);
        } else {

                // Echo process

                while (true) {
                        mpi.recv (buffer, MAX_SIZE, 0);

                        if (buffer == "stop")
                                break;

                        mpi.send (buffer, 0);
                }
        }
 end:;
}
```

2 Tests

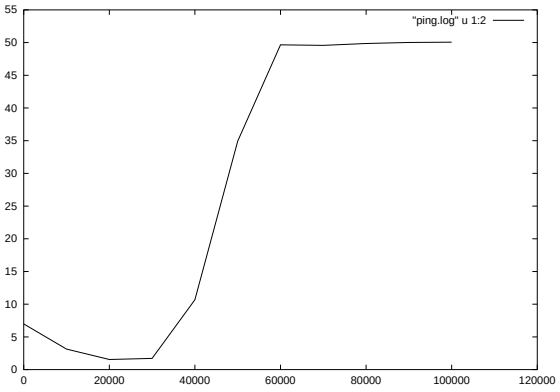
I wasn't able to compile the needed libraries on ÅA machines, so I used just a single machine, with a single processor, but with two virtual processors.

Test run:

```
> /usr/local/src/mpich/bin/mpirun -np 2 ping 1 10000 100001 -loops=1000
Sender iterating for 1...100001 +=10000 (11 values)
1       2.125779
10001   2.338599
20001   2.418336
30001   2.717577
40001   18.308221
50001   63.814653
60001   81.402558
70001   81.351042
80001   81.497070
90001   81.534450
100001  81.260095
```

Results plotted with gnuplot:

`plot "ping.log" u 1:2 w l`



3 C++ MPI coating

This coating is just a test.

3.1 mpi++.h

```
#include <object.h>
#include <cstring.h>

#include <mpi.h>

class MPIInstance : public Object {
        MPI_Status mMPIStatus;
  public:
                                        MPIInstance             (int& argc, char**& argv);
                                        ~MPIInstance    ();

        /** Returns the MPI process rank of the current process. */
        int                             getRank                 () const;

        /** Sends a message. */
        void                    send                    (const String& buffer, int receiver);

        /** Receives a message. Blocking. */
        void                    recv                    (String& buffer, int maxlen, int sender);

        /** Coating for the other recv. */
        String                  recv                    (int maxlen, int sender);

        /** Returns the number of processor in the given comm channel */
        int                             size                    (int communicator) const;

        /** Returns a time stamp. */
        double                  time                    () const;
};
```

3.2 mpi++.cc

```
#include "mpi++.h"

MPIInstance::MPIInstance (int& argc, char**& argv) {
        MPI_Init (&argc, &argv);
}
```

```
MPIInstance::~MPIInstance () {
    MPI_Finalize();
}

int MPIInstance::getRank () const {
        int myrank;
        MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
        return myrank;
}

void MPIInstance::send (const String& buffer, int target) {
        MPI_Send (buffer.getbuffer(), buffer.len, MPI_CHAR, target, 99, MPI_COMM_WORLD);
}

void MPIInstance::recv (String& buffer, int maxlen, int source) {
        buffer.ensure (maxlen+1);
        MPI_Recv(buffer.getbuffer(), maxlen, MPI_CHAR, source, 99,
                        MPI_COMM_WORLD, &mMPIStatus);
        int len;
        MPI_Get_count (&mMPIStatus, MPI_CHAR, &len);

        // Ensure string termination (there is always room for one 0 in
        // MagiClib Strings).
        buffer.getbuffer()[buffer.len=len] = 0;
}

String MPIInstance::recv (int maxlen, int sender) return result {
        recv (result, maxlen, sender);
}

int MPIInstance::size (int communicator) const {
        int size;
    MPI_Comm_size(communicator,&size);
        return size;
}

double MPIInstance::time () const {
        return MPI_Wtime();
}
```