Some early performance results with MPI on the IBM SP1
EARLY DRAFT

William Gropp and Ewing Lusk

June 5, 1995

Abstract

This note provides some performance results for MPI, the new message-passing interface, on the IBM SP1. MPI is the product of over a years work by a large group of experts representing users, vendors, and software authors. MPI has been designed to be fast as well as powerful. We compare the performance of a portable MPI, and vendor-specific MPI, and the same vendors native message-passing on an MPP.

1 The test

The data shown in this note was generated with the program `twin`. These figures are for a round-trip using blocking sends and receives. The code is

```
if(myproc==proc1){
  MPI_Recv(rbuffer,len,MPI_BYTE,MPI_ANY_SOURCE,0,
           MPI_COMM_WORLD,&status);
  t0=MPI_Wtime();
  for(i=0;i<reps;i++){
    MPI_Send(sbuffer,len,MPI_BYTE,proc2,1,MPI_COMM_WORLD);
    MPI_Recv(rbuffer,len,MPI_BYTE,MPI_ANY_SOURCE,1,
             MPI_COMM_WORLD,&status);
  }
  t1=MPI_Wtime();
  elapsed_time = t1-t0;
}

if(myproc==proc2){
  MPI_Send(sbuffer,len,MPI_BYTE,proc1,0,MPI_COMM_WORLD);
  for(i=0;i<reps;i++){
```

```
    MPI_Send(sbuffer,len,MPI_BYTE,proc1,1,MPI_COMM_WORLD);
    MPI_Recv(rbuffer,len,MPI_BYTE,MPI_ANY_SOURCE,1,
             MPI_COMM_WORLD,&status);
  }
}
```

2 The results

Some of the results are shown in Figures 1, 2, and 3. The data labeled `mpich` is the public domain, portable MPI. The data labeled `mpif` is IBM's experimental MPI-F, version 1.12. The data labeled `euih` is IBM's propritary message-passing system. The `mpich` system is using `euih` to send and receive messages.

These show that (a) a vendor implementation of MPI can be as fast as the vendor's propritary system and (b) the public domain system is nearly as fast. For small messages, `mpich` has a latency that is about 12 μseconds greater than the underlying message-passing system and the transfer rate is about 1 megabyte per second slower. This is due to the fact that the `mpich` implementation, having received a message into an internal buffer, must copy it into the user's buffer. Looking at the data in detail, we see that the transfer rate for `euih` for short messages is about 7.28 MB/sec and for `mpich` about 6.18 MB/sec. If we ascribe the difference in rate solely to the additional copy, we can compute that the copy is performed at 40 MB/sec, which is quite reasonable. To see this, consider the times (in μseconds) to send a byte with `euih` + a copy and with `mpich`:

$$\frac{1}{7.28} + \frac{1}{copyrate} = \frac{1}{6.18}$$
$$copyrate = \frac{1}{\frac{1}{6.18} - \frac{1}{7.28}}$$

The implementation used in `mpich` avoids most of this copy for long messages by using a second message; this trades a relatively fixed cost (the latency of the second message) against the savings in not needing to copy the message from an internal buffer to the user's buffer. Further improvements can be made in the current implementation as well.

Additional data is available for nonblocking sends and receives.

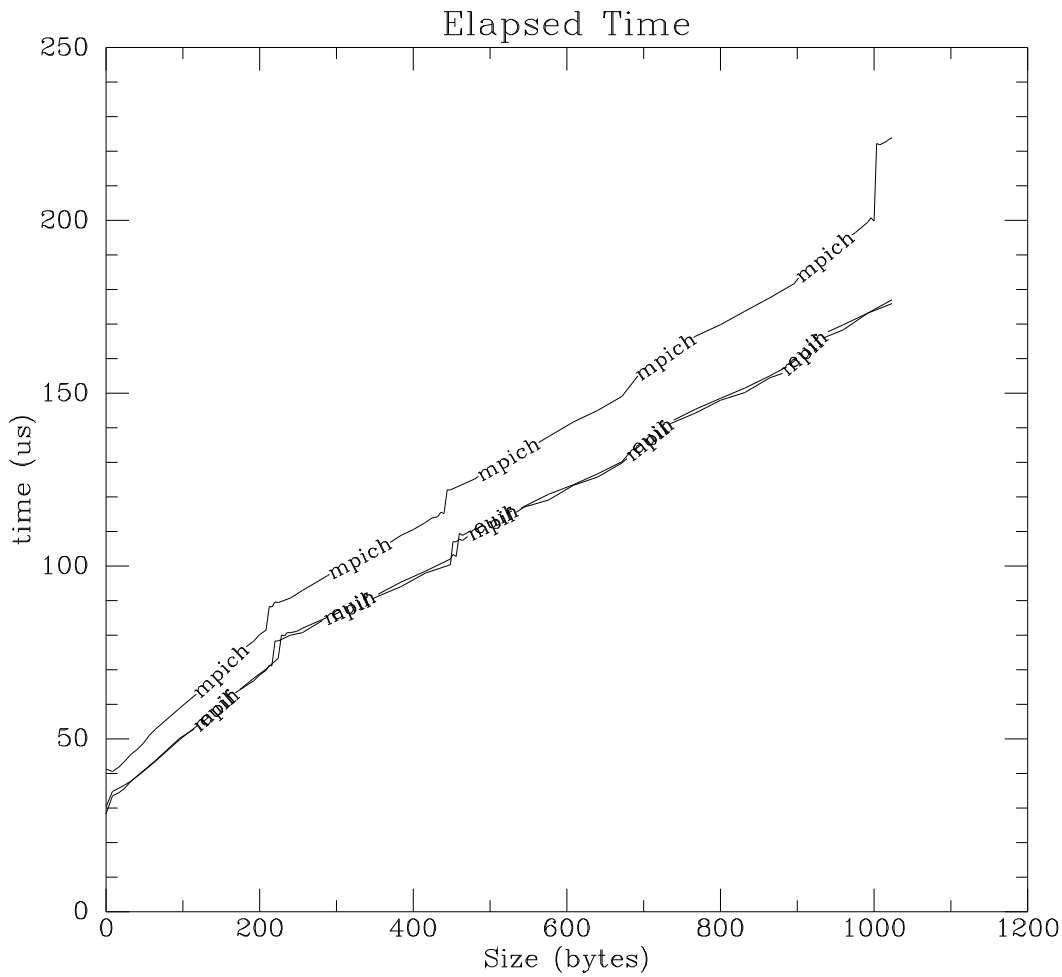

Figure 1: Time for sending short messages

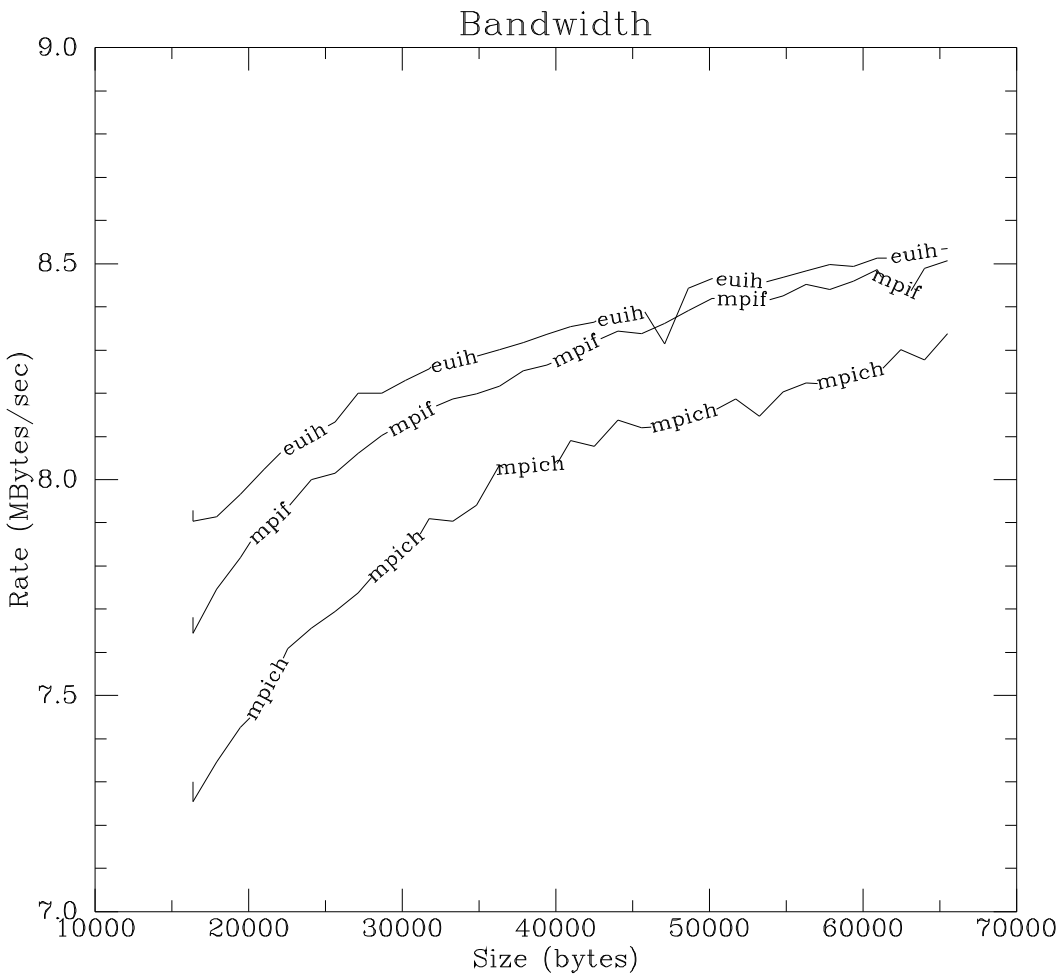

Figure 2: Bandwidth for messages of between 16384 and 65536 bytes

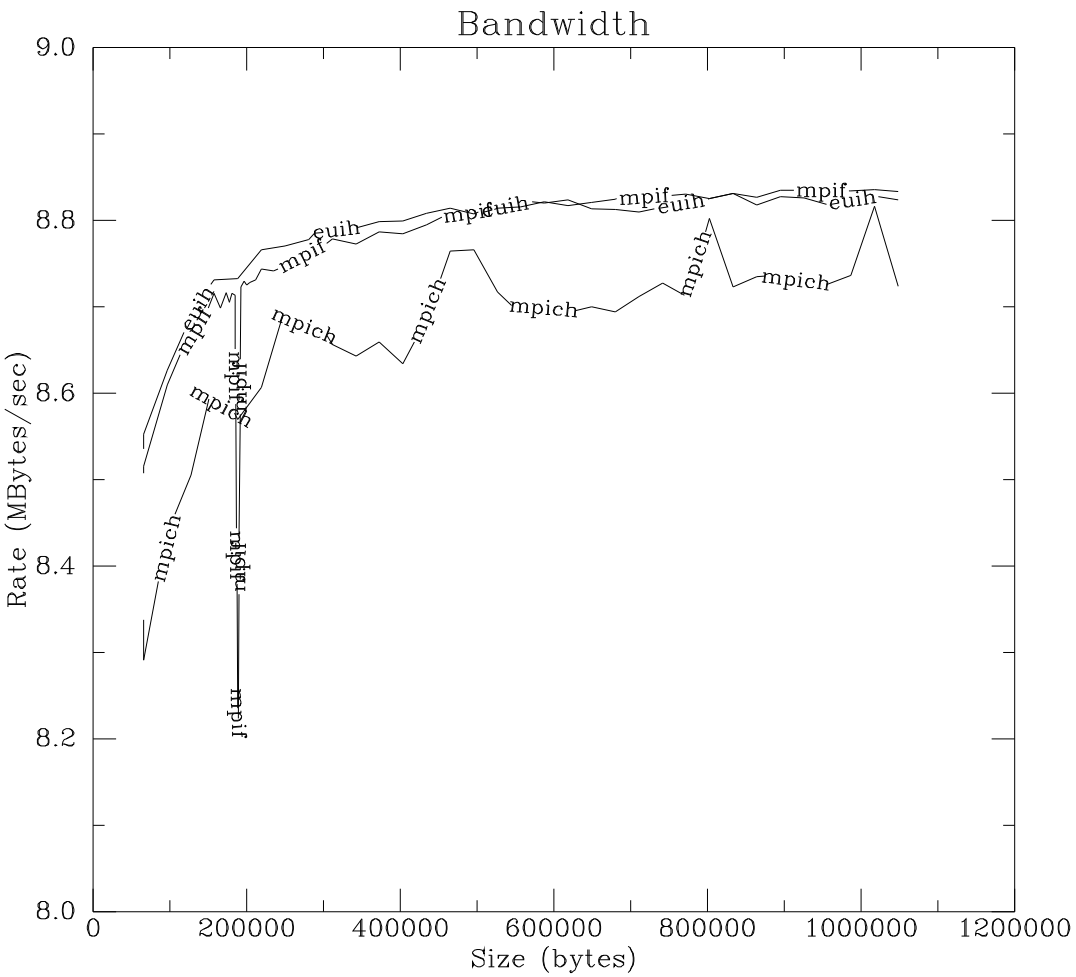

Figure 3: Bandwidth for messages of upto a megabyte in length