課題

各 PE が自分のランクを出力する。

- 最も良い throughput を出すデータサイズを調べる
- クロスバースイッチの性能 (0->1 && 2->3) を同時に行う
- latency を調べる
- グラフを描いてみる

クロスバースイッチの性能を調べるには、2x1 と 1x2 というオプションを与えてやればよい（?)。同時に行うという意味が不明。理解が足りないのかもしれない。

(追記）講究の時間に聞いて意味を理解した。つまり、複数台のマシンで同時に実行させる事によって、回線 (myrinet,ethernet) がどうなるのかを調べればよい。

プログラムについて

講義用ページに上がっていたサンプルプログラムをベースに、スループットが出せるようにプログラムを直した。ただし、オーバーヘッド、キャッシュなどを考えていない。

プログラムで若干分からない事があったので、ここで補足しておく。資料、`MPI_send` 内の、送るデータの個数とは「送信バッファ内の要素数」のことである。このプログラムでは送るデータの型を `MPI_BYTE` にしてあるので、データを任意の量だけ送ることが出来る。

プログラムの実行

folon3 では-np 1x2 というオプションを使うと

```
FEP: ERROR SCore-D Login failed: Resource unavailable.
```

というエラー出てしまい、実行できなかった。なので、folon4 を使った。

folon4 の pcch1 での実行結果

膨大な量になったので、データを割愛してある。データの見方は、左から、rank、`MPI_Send` でやりとりするデータ量 (KB)、時間、スループット (MB/s) となっている。

```
[tobita@comte ~]$ scout -g pcch1
SCOUT: Spawning done.
SCOUT: session started.
[tobita@comte ~]$ scrun -nodes=1x2 ./throughput
SCore-D 5.6.1 connected.
<0:0> SCORE: 2 nodes (1x2) ready.
Process 0 of 2 on comte00.clusters.ueda.info.waseda.ac.jp
## Datasize(KB) Throughput(MB/s)
```

```
Process 1 of 2 on comte00.clusters.ueda.info.waseda.ac.jp
## Datasize(KB) Throughput(MB/s)

0           0       0.000006       0.000000
0           4       0.000031       126.088964
0           8       0.000039       198.648116
0          12       0.000048       242.166849
0          16       0.000064       245.003552
0          20       0.000074       264.490560
0          24       0.000086       274.039125
0          28       0.000096       285.721760
0          32       0.000108       289.010407
0          36       0.000117       299.675340
0          40       0.000128       304.727895
0          44       0.000139       310.124154
0          48       0.000151       310.939780
0          52       0.000160       317.462589
0          56       0.000171       320.120581
0          60       0.000182       322.521801
0          64       0.000195       319.821389
0          68       0.000207       320.438099
0          72       0.000218       322.208717
0          76       0.000229       323.930926
0          80       0.000245       319.313974
0          84       0.000254       323.568370
0          88       0.000269       319.403812
0          92       0.000279       321.507064
0          96       0.000291       322.238202
0         100       0.000302       322.869248
0         140       0.000431       316.893839
0         180       0.000573       306.844252
0         220       0.000738       291.257931
0         260       0.000900       281.982557
0         300       0.001072       273.337727
0         340       0.001236       268.690955
0         380       0.001421       261.081098
0         420       0.001615       253.925238
0         460       0.001839       244.276868
0         500       0.002157       226.336357
0         540       0.002516       209.624866
0         580       0.002880       196.695265
0         620       0.003268       185.263616
0         660       0.003649       176.654086
```

```
0         700       0.004021       169.993044
0         740       0.004401       164.185855
0         780       0.004769       159.726490
0         820       0.005135       155.955444
0         860       0.005497       152.776793
0         900       0.005853       150.160450
0         940       0.006211       147.798388
0         980       0.006558       145.929758
0        1020       0.006903       144.302527
```

2x1 で実行

```
[tobita@comte ~]$ scrun -nodes=2x1 ./throughput
SCore-D 5.6.1 connected.
<0:0> SCORE: 2 nodes (2x1) ready.
Process 0 of 2 on comte00.clusters.ueda.info.waseda.ac.jp
## Datasize(KB) Throughput(MB/s)
Process 1 of 2 on comte01.clusters.ueda.info.waseda.ac.jp
## Datasize(KB) Throughput(MB/s)
0           0       0.000014       0.000000
0           4       0.000070       55.444539
0           8       0.000115       68.120845
0          12       0.000121       97.173388
0          16       0.000175       89.174029
0          20       0.000186       105.147302
0          24       0.000221       106.252770
0          28       0.000239       114.294256
0          32       0.000267       116.998277
0          36       0.000282       124.597630
0          40       0.000314       124.321115
0          44       0.000330       130.267656
0          48       0.000358       130.781230
0          52       0.000373       136.100168
0          56       0.000396       138.217239
0          60       0.000419       139.889516
0          64       0.000449       139.063086
0          68       0.000464       143.055507
0          72       0.000493       142.696018
0          76       0.000511       145.201818
0          80       0.000540       144.665812
0          84       0.000552       148.530245
0          88       0.000581       148.034613
0          92       0.000603       148.895725
```

```
0          96       0.000628       149.398034
0         100       0.000645       151.291295
0         140       0.000883       154.803250
0         180       0.001112       158.024353
0         220       0.001364       157.548515
0         260       0.001609       157.807388
0         300       0.001856       157.889766
0         340       0.002107       157.585612
0         380       0.002339       158.626773
0         420       0.002581       158.927549
0         460       0.002831       158.662798
0         500       0.003069       159.118694
0         540       0.003332       158.286121
0         580       0.003605       157.116056
0         620       0.003892       155.578720
0         660       0.004103       157.089073
0         700       0.004370       156.426971
0         740       0.004644       155.615350
0         780       0.004889       155.787352
0         820       0.005151       155.473137
0         860       0.005422       154.893172
0         900       0.005691       154.425388
0         940       0.005951       154.246801
0         980       0.006256       152.975770
0        1020       0.006486       153.570923
0        1024       0.006507       153.679870
```

これ以外のデータは Excel で。

グラフ

別ファイル参照（eps 化はまだ)

プログラム

最終的に MPI_Send でやりとりするデータ量 (KB) と時間、スループット (MB/s) を表示するようにした。通信は 1 回のピンポン通信である。

MPI_Send でやりとりするデータ量 (KB) が大きくなるということは、パケットのサイズが大きくなるということ。0〜1024KB の間を 4KB づつずらして観察した。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>

#define DEFAULT_SIZE 1024
#define KB 1024
#define MB 1048576

int main(int argc, char *argv[]){
  int numprocs, namelen, myid;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  double wtime, timer, sum, zero_timer, zero_wtime, s;

  int try;
  int i, j, k;
  char *packet;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Get_processor_name(processor_name, &namelen);

  /*
  if(numprocs != 2){
    if(myid == 0){
      printf("Sorry, This program runs only in the case of using 2 PEs.\n");
    }
    MPI_Finalize();
    exit(0);
  }
  */

  fprintf(stdout, "Process %d of %d on %s\n", myid, numprocs, processor_name);
   fprintf(stdout, "## Datasize(KB)\tThroughput(MB/s)\n");
     fflush(stdout);


  packet = (char *) malloc(KB * KB);       // 1024KB
  //  try = 100;
  try = 100;

  /* warm-up */
  if(myid == 0){
    MPI_Send(packet, KB * KB, MPI_BYTE, 1, 4040, MPI_COMM_WORLD);
```

```
  MPI_Recv(packet, KB * KB, MPI_BYTE, 1, 4050, MPI_COMM_WORLD, &status);
}
else if(myid == 1){
  MPI_Recv(packet, KB * KB, MPI_BYTE, 0, 4040, MPI_COMM_WORLD, &status);
  MPI_Send(packet, KB * KB, MPI_BYTE, 0, 4050, MPI_COMM_WORLD);
}



/* i byte ping-pong */


for(i = 0; i <= 1024 * KB; i += 4 * KB){
  sum = 0.0;
 wtime = 0.0;

 for(j = 0; j < try; j++){
   if(myid % 2 == 0){
      timer = MPI_Wtime();

      MPI_Send(packet, i, MPI_BYTE, myid+1, 4040, MPI_COMM_WORLD);
      MPI_Recv(packet, i, MPI_BYTE, myid+1, 4050, MPI_COMM_WORLD, &status);

      MPI_Barrier(MPI_COMM_WORLD);
      wtime = MPI_Wtime() - timer;
      sum += wtime;
    }
    else if(myid % 2 == 1){
      MPI_Recv(packet, i, MPI_BYTE, myid-1, 4040, MPI_COMM_WORLD, &status);
      MPI_Send(packet, i, MPI_BYTE, myid-1, 4050, MPI_COMM_WORLD);
      MPI_Barrier(MPI_COMM_WORLD);
    }
  }
  /* output results */
 for(j=0;j<numprocs/2;j++){
   MPI_Barrier(MPI_COMM_WORLD);
  if(myid == 2 * j){
    s = sum / 2.0 / try;
    fprintf(stdout, "%d\t%4d\t%10.6f\t%f ",myid, i / KB,
            s,i / s / MB);
    fflush(stdout);
    if(myid + 2 >= numprocs){
      fprintf(stdout, "\n");
```

```
        fflush(stdout);
      }
    }

        }


    MPI_Barrier(MPI_COMM_WORLD);
  }

  /* Finallize */
  MPI_Finalize();
  return 0;
}
```

表示するときに同期を取りたかったため MPI_Barrier が多く使ってある。このため動作が少し遅くなった。

考察

まず、注目したいのは他のマシンとは同期を取るためだけに通信する、つまり Xx2 通信と、他のマシンとの通信がメインである、Xx1 通信の違いである。

これはグラフから容易に読み取れる。Xx2 通信は、最初一気にスループットが上昇するが、そのあとは一気に下がっていく。Xx1 通信は、最初スループットが上昇したら、その後はほとんど変化がない。両者とも最終的には 150MB/s ぐらいに収束する（ただしすべて Ethernet でやった場合は除く)。

> 補足：講究で myrinet は理論的には 180MB/s ぐらい出るということを聞いた。これを考えてみると、ちょっと性能が悪そうである。また、両者が 150MB/s に収束するのは、どうやら偶然らしい。

次に見たいのが Myrinet と Ethernet では何が違うかである。Myrinet と Ethernet をそれぞれグラフにしてみた。このグラフで一番特徴的なのは、Ethernet の Xx1 の部分である。スループットを見てみるとほとんど Myrinet と変わらないのに、振幅がもの凄く大きい事が見て取れる。これはおそらく同期を取るための通信 (MPI_Barrier) に Ethernet を使っているためであろう。つまり、Ethernet の性能が悪くなる（衝突などによって）とスループットが一気に悪くなるのだろう。

最後に、見たいのが Ethernet の Xx1 通信のグラフである。このグラフで特徴的な部分は、スループットが一気に悪くなる点があるというところだ。これは分割する事によって、また、グラフの 2x1,4x1,8x1 はそれぞれ収束する値が違う。これは、12MB/s という論理的な限界の値を share しているためだろう。16x1 になったら 12/8=1.5MB/s に収束するようになるはずだ。それを踏まえて Myrinet のグラフを見てみると、全て一定の値にスループットが収束している事が分かる。つまり folon4 の Myrinet は、何台で通信しても最悪のスループットが同じである。