Motivation

• Can we estimate the costs for a parallel code in order to
  - Evaluate quantitative and qualitative differences between different implementation alternatives
  - Understand the parameters effecting the performance of the application
  - Understanding relevant hardware characteristics

• Restrictions:
  - Any analytical model can not replace real measurements since parallel systems are too complex and unpredictable.
How to model collective operations?

- E.g. MPI_Bcast: strongly depending on the algorithm used to implement the operation
  - One process (root process) distributes the same data items to all members within a process group (communicator)
- Linear Algorithm:
  - the root process sends one message to each process in

```c
... if (rank == root) {
    for (i=0; i<size; i++)
        if (i != root)
            MPI_Send(buf, cnt, dat, i, TAG, comm);
} else
    MPI_Recv(buf, cnt, dat, root, TAG, comm, &stat);
```

Linear Algorithm (I)

- Hockney’s Model: \( t(s) = l + \frac{1}{b} s \)
  - \( s \): message size
  - \( l \): latency
  - \( b \): bandwidth
- Estimate of the execution time according to Hockney’s model for \( p \) processes:

\[
  t(s, p) = (l + \frac{1}{b} s)^*(p-1) \quad (4:1)
\]
Linear Algorithm (II)

- Using non-blocking operations:

```c
if (rank == root ) {
    for (i=0; i<size; i++)
        MPI_Isend (buf, cnt, dat, i, TAG, comm, &req[i]);
}

MPI_Recv (buf, cnt, dat, root, TAG, comm, &stat);
if (rank == root ) {
    MPI_Waitall ( size, req, statuses);
}
```

- Formula (4:1) is now arbitrarily wrong
  - Several communications simultaneously ongoing
  - Maximum (optimal) number of messages depending on message size and network parameters

How does communication really work (I)

- Two protocols usually used internally:
  - Eager protocol:
    - message is sent immediately to the receiver, without waiting for the according receive to be posted
    - Usually used for short messages (e.g. 1 KB in Open MPI)
  - Rendezvous protocol:
    - Send a header to receiver
    - Wait for an acknowledgment - receive has started
    - Send message data
    - Avoids having to buffer large messages on the receiver process (unexpected messages)
How communication really works (II)

- Three levels of buffering
  - Application level (e.g. MPI_Bsend)
  - MPI library level - unexpected message queues
  - System buffering
- System buffering works similarly to file systems
  - e.g. for sockets: data is copied into socket buffer before sending
  - MPI_Send returns as soon as data is in the socket buffer!
    - No way to alternate this data anymore, so it is safe to return control to the application

How communication really works (III)

- For a short message ( < socket buffer size (=sbsize) )
  - Data copied into socket buffer
  - write operation on the according socket called
  - MPI_Send returns control to the application in a time which is shorter than the network latency!
- For a long message
  - Large message is split into chunks of size sbsize
  - A chunk of the data is copied into socket buffer and sent
  - As soon as the receiving process acknowledges the receipt of the data chunk, the next chunk is copied into socket buffer etc.
How communication really works (IV)

- So transfer of a large message looks like
  - Sending a small chunk
  - Wait
  - Sending a small chunk
  - Wait
- This behavior is not modeled by Hockney, but e.g. by the LogGP model
- Based on LogGP, one should split a large message into smaller chunks and send them simultaneously for a bcast operation
  - Hide the gap by using a different channel

Multi-segmented linear algorithm

```c
nmsgs = cnt/scnt;
if (rank == root) {
    for (j=0; j<nmsgs; j++) {
        tbuf = buf + (j*scnt);
        for (i=0; i<size; i++)
            MPI_Isend (tbuf, scnt, dat, i, TAG, comm,&req[2*j+i]);
    }
    for (j=0; j<nmsgs; j++) {
        tbuf = buf + (j*scnt);
        MPI_Irecv (tbuf, scnt, dat, root, TAG, comm, &rreq[j]);
    }
    if (rank == root) {
        MPI_Waitall ( size*nmsgs, req, statuses);
    }
    MPI_Waitall ( nmsgs, rreq, rstatuses);
}
```
Binary and Binomial Trees

Number of messages increase with every iteration
- network saturated starting from a certain number of messages
- message segmenting can improve the performance as well

Chain Algorithms

- Segment a message and pass them from one process to another
- Performs very well for very large messages
k-Chain Algorithm

e.g. k=5

Hockney’s Model

\[ t(s) = l + s/b \]

- \( l \): latency of the network
- \( b \): bandwidth of the network
- How can we determine the latency and the bandwidth?
  - Ping-pong benchmark:
    - process A sends a message to process B, process B sends message back
    - Advantage: does not require synchronized clocks between A and B
    - Disadvantage: assumes symmetric communication performance (costs (A->B) == costs (B->A))
- To determine latency: execute ping-pong benchmark for cnt=0
Parallel Computation
Edgar Gabriel

Ping pong benchmark

```c
comm = MPI_COMM_WORLD;
for (i=1; i< MAX_MSG_LEN; i*=2 ) {
  t1 = MPI_Wtime();
  for ( j=0; j<MAX_MEASUREMENTS; j++ ) {
    if ( rank == 0 ) {
      MPI_Send (buf, i, MPI_INT, 1, 1, comm);
      MPI_Recv (buf, i, MPI_INT, 1, 1, comm, &status);
    }
    else if ( rank == 1 ) {
      MPI_Recv (buf, i, MPI_INT, 0, 1, comm, &status);
      MPI_Send (buf, i, MPI_INT, 0, 1, comm);
    }
  }
  t2 = MPI_Wtime();
  if ( rank == 0 ) {
    printf("Msg len: %d avg. exec.%lf bandw. %d 
",
       i, (t2-t1)/(2*MAX_MEASUREMENTS),
       i*sizeof(int)/((t2-t1)/(2*MAX_MEASUREMENTS));
  }
}
```

Ping-pong benchmark (II)
Ping-pong benchmark (II)

- To determine bandwidth: have to determine the saturation point
  - Required message length does depend on the network bandwidth

LogP

- Model published by Culler et al
- Parameters:
  - $L$: upper bound on the latency
  - $o$: overhead, defined as the length of the time that a process is engaged in the transmission or reception of a message. During this time, the process can not perform other operations
  - $g$: gap, defined as the minimum time interval between consecutive message transmissions or receptions. The reciprocal time of $g$ corresponds to the per-process communication bandwidth
  - $P$: number of processors
Parallel Computation
Edgar Gabriel

LogP (II)

Start sending
Message enters network
Sender

Message leaves network
Receiver

End receiving

Costs for sending a message:

\[ t = L + 2o \]  
(19:1)

Costs for sending two messages:

\[ t = L + g + 2o \]  
(20:1)
LogP(III)

- Please note:
  - Latency in the LogP model is different than the latency in the Hockney model.
    - Latency of Hockney includes the overhead \( o \)
  - In the formula (20:1), we assumed that \( o < g \)
    which is typically correct. The formulas should however be instead
    \[
    t = L + \max(g, o) + 2o
    \]

LogP(III)

- LogP assumes, that any large message can be decomposed to a series of short messages
  e.g. sending a message of \( k \) bytes takes
  \[
  t = o + (\lceil k / w \rceil - 1) \max(g, o) + L + o
  \]
  with \( w \) being the size of the network package in bytes for which LogP still holds
- LogP assumes, that the overhead is equal for the sender and the receiver side
  - More fine grained approaches use different values, e.g. \( o_s \) and \( o_r \)
LogGP

- Extension of LogP taking into account, that large message can often be transferred more efficiently than what LogP predicts, due to special hardware support
- Additional parameter:
  \( G \): Gap per bytes for long messages
- Sending a \( k \) byte message with LogGP:
  - \( o \) cycles until the first byte enters the network
  - \( G \) cycles for each subsequent byte
  - \( o \) cycles on the receiver side

\[
t = o + (k - 1)G + L + o
\]  

(23:1)

LogGP

Costs for sending two \( k \)-byte messages:

\[
t = o + (k - 1)G + g + (k - 1)G + L + o \\
= 2o + 2(k - 1)G + g + L
\]  

(24:1)
PLogP

- Extension of the PLogP model making the parameters $g$, $o_s$ and $o_r$ dependent on the message length $m$
  - $g(m)$, $o_s(m)$ and $o_r(m)$
- Latency $L$ is considered to be an end-to-end latency

<table>
<thead>
<tr>
<th>LogP/LogGP</th>
<th>PLogP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$L + g(1) - o_s(1) - o_r(1)$</td>
</tr>
<tr>
<td>$o$</td>
<td>$(o_s(1)+o_r(1))/2$</td>
</tr>
<tr>
<td>$g$</td>
<td>$g(1)$</td>
</tr>
<tr>
<td>$G$</td>
<td>$g(m)/m$, for sufficiently large $m$</td>
</tr>
<tr>
<td>$P$</td>
<td>$P$</td>
</tr>
</tbody>
</table>

PLogP(II)

Receiver spends $L + g(m)$ cycles in a recv operation.
PLogP(III)

- How can we determine the parameters of LogP, LogGP and PLogP
- Since we can determine the parameters of LogP/LogGP using the PLogP model, we will only focus on PLogP
- Idea: execute a series of measurements, whose performance you can model using PLogP, and which lead to a set of linearly independent equations
  - Determine the parameters from the equations

PLogP(IV)

- Test 1: Send \( n \) very small messages (\( m=0 \)) and wait for a single acknowledgement. Measure the
  - Time to send \( n \) messages of length 0: \( n^*g(0) \) (7)
  - RoundTripTime (RTT) = \( 2(L+g(0)) \) (8)
- Test 2: Send a message of length \( m \) and wait for an ack of length 0. Measure the
  - Time to send a message of length \( m \): \( o_s(m) \) (9)
  - RTT(m) = \( L*g(m)+L+g(0) \) (10)
- Test 3: send a message of length 0, wait for \( \Delta > \text{RTT}(m) \) and receive than a message of length \( m \)
  - Since \( \Delta > \text{RTT}(m) \) we know that the message is available, and thus we really measure \( o_r(m) \) (11)
Please note, that
• since $g$ is a network parameter (not software) $n$ has to be sufficiently large to saturate the network.
Parallel Computation
Edgar Gabriel

PLogP(VII) – Test 3

Sender

Receiver

Example: linear broadcast
Example: linear broadcast

- Execution time according to LogP:
  - First message takes $o$ cycles to push into the network
  - All subsequent messages take $g$ cycles
  - The last message takes $L+o$ cycles to be received
  $t(P) = o+(P-2)g+L+o$

- Execution time according to LogGP:
  - First message takes $o+(k-1)G$ cycles
  - Subsequent messages take $g+(k-1)G$ cycles
  - Last message takes $L+o$ cycles to be received
  $t(k,P) = o+(P-2)g+(P-1)(k-1)G+L+o$

Example: non-segmented chain broadcast
Example: non-segmented chain broadcast

- Execution time according to LogP:
  - Root process takes $o$ cycles to push the message into the network
  - A process takes $L+o$ cycles to receive the message and $o$ cycles to push the message into the network
  - Last process takes $L+o$ cycles to receive the message
  
  $t(P) = o + (P-2)(L+2o) + L + o = (P-1)(L + 2o)$

- Similarly for LogGP:
  
  $t(k,P) = o + (k-1)G + (P-2)(L+2o+(k-1)G) + L + o$
  
  $= (P-1)(L+2o+(k-1)G)$