The Analysis and Optimization of Collective Communications on a Beowulf Cluster

Wi Bing Tan and
Peter Strazdins

Department of Computer Science,
Australian National University,


20 December 2002
1 Talk Outline

1. talk outline
2. motivations for optimizing collective communications
3. the Beowulf cluster ‘Bunyip’
4. types of collective communications: all-gather, all-reduce and reduce-scatter
5. algorithms for collective communications: traditional
6. algorithms for collective communications: from repeated sub-operations
7. results: comparison of algorithms
8. results: comparison with performance models
9. a simulator for understanding collective communication performance
10. conclusions
Motivations for Optimizing Collective Communications

- clusters made from commercial-off-the-shelf (COTS) networks are increasingly popular
  - eg. Beowulf clusters built from switch-based networks
- such networks typically slow; also may be different from vendor-designed networks (eg. contention-free)
  - optimization of collective communications is thus particularly important
  - may require different techniques to the ‘traditional algorithms’ built for networks on custom-made vendor-designed parallel computers
- our goal: to evaluate and understand collective communication performance for COTS network clusters
3 The Beowulf cluster ‘Bunyip’

- 550 MHz dual Pentium III nodes, in 4 groups of 24
- each node has 3 100 Mb NICs
  - can communicate with 3 other nodes simultaneously
- contention-free switches
- ‘Bunyip’ is an monster in Australian mythology
- won Gordon-Bell Award for Price/Performance in 2000
- (MPI) communication startup cost is \( \alpha = \alpha_s + \alpha_r = 24 \mu s + 180 \mu s \);
  bandwidth is 8 MB/s \( = 8/\beta \),
  ie. communication cost per double is \( \beta = \beta_s + \beta_r = 0.082 \mu s + 1.063 \mu s \)
4 Types of Collective Communications

- **All-Gather:**
  start: node \( i, 1 \leq i \leq p \), has \( n \) words of data \((x_{1:n}^i)\)
  end: all nodes have all of the \( pn \) data \((x_{1:n}^1, \ldots, x_{1:n}^p)\)

- **Reduce-Scatter:**
  start: node \( k, 1 \leq k \leq p \), has \( np \) words of data \((y_{1:p, 1:n}^k)\)
  end: node \( i \) has \( n \) words of summed data \((x_{1:n}^i, \ x_{1:n}^j = \sum_{k=1}^{p} y_{i,j}^k)\)

- **All-Reduce**
  start: node \( i, 1 \leq i \leq p \), has \( n \) words of data \((y_{1:n}^i)\)
  end: all nodes have \( n \) words of summed data \((x_{1:n}, \ x_j = \sum_{k=1}^{p} y_{j}^k)\)

- these operations are widely-used (e.g. in dense linear algebra) and are in the MPI standard

- as we can model the time to send a message: \( t = \alpha + \beta n = (\alpha_s + \alpha_r) + (\beta_s + \beta_r) n \),
  we similarly can have performance models for collective communications, e.g. \( t_{ag} = f(n, p, \alpha_s, \alpha_r, \beta_r, \beta_s) \)
5 Algorithms: traditional

- **widely-used; good performance on traditional parallel computers**

- **bi-directional exchange:**

  \[
  t_{ag} = \alpha \log_2 p + (p - 1)(\beta n) \\
  t_{ar} = \log_2 p(\alpha + \beta n)
  \]

- **fan-in/fan-out: slowest, but often used**

- **recursive-halving/doubling:** (v. complex for non-power-of-2 \( p \! \) !)

  \[
  t_{ag} = t_{rs} + \log_2 p(\alpha + \beta \frac{n}{2}) \\
  t_{rs} = \alpha \log_2 p + (p - 1)(\beta n) \\
  t_{ar} = 2(\alpha \log_2 p + (p - 1)(\beta \frac{n}{p}))
  \]

- **ring (rotation) – no contention:**

  \[
  t_{ag} = t_{rs} = t_{ar} = (p - 1)(\alpha + \beta n)
  \]

- **above models assume an exchange is as fast as a single message**
6 Algorithms: based on repeated sub-operations

- tree-like and ring-like patterns occur frequently in dense linear algebra
- in the case of All-Gather:
  - **tree**: binary-tree broadcast from each node \( i, 1 \leq i \leq p \)
    \[ t^{rs} = p \log_2 p (\alpha + \beta n) \text{ (no overlap)} \]
    \[ t^{ag} \approx \min[\log_2 p, 2] p (\alpha + \beta n) \]
  - **pipeline**: pipelined broadcast from each node \( i, 1 \leq i \leq p \)
    \[ t^{ag} = t^{rs} = 3(p - 1)(\alpha + \beta n) \]
  - **fan-in**: gather from other nodes into node \( i, 1 \leq i \leq p \)
    \[ t^{ag} = t^{rs} \approx 2(p - 1)(\alpha + \beta n) \]
  - **full fan-in**: each node \( i \) in parallel:
    - for each \( k = 1 : n \), send data to node \( i + k \);
    - for each \( k = 1 : n \), receive data from node \( i - k \);
    \[ t^{ag} = t^{rs} = \frac{p - 1}{o} (\alpha + \beta n) \]
    \( o \) is degree of overlap on simultaneous receives, \( 1 \leq o \leq 3 \) on Bunyip
- simple to implement; not contention-free;
  but may be fast if there is overlap between the sub-operations
- exact performance models are in terms of \( \alpha_s, \alpha_r, \beta_r, \beta_s \)
7 Results: comparison of algorithms

- results for All-Gather (left) & Reduce-Scatter (right), for $p = 8$ (single Bunyip group)
- for All-Reduce, bi-directional exchange was best (as expected), and also significantly faster than MPI
- for $1000 \leq n \leq 10,000$, results were similar except some degradation at $n \geq 8,000$ for ring, fan-in and full fan-in
8 Results: compare with performance models & scalability

- results for All-Gather: compare with performance models for $p = 8$ (left), and performance at $n = 1000$ (right)
  - close match for all ops, with full-fan-in’s overlap factor $o^{ag} \approx 1$, $o^{rs} \approx 1.2$
  - larger $p$ requires the operations to be ‘inter-group’ on the Bunyip:
    - a hierarchical algorithm (based on ring or bi-directional exchange worked $\approx 20\%$ better for $n \geq 1000$
      (can avoid large messages between groups)
9 A Simulator for Collective Communications

- A message simulator and diagramming tool was developed to understand performance overlap
  - generated timing diagrams based on both performance model predictions and actual timestamps for MPI send & receive calls
  - was useful in understanding message overlap effects and deriving the performance models for tree and fan-in
    (predicted diagram for fan in, \( p = 8 \), on right)
  - is generic; source code is available from
    http://cs.anu.edu.au/
    ~Peter.Strazdins/
    projects/ClusterComm
10 Conclusions

- LAM MPI performance was sub-optimal for these operations
- for clusters like Bunyip
  - bi-directional exchange worked well for small messages, ring slightly better for large
  - significant overlap can occur on algorithms based on repeated sub-operations:
    - required more complex performance models (with separated send and receive components)
    - full fan-in (believed novel) modestly faster for Reduce-Scatter
      - would be even better if overlap factor $o \rightarrow 3$
    - these are very simple and reliable to implement
  - close match of actual results with performance models indicate a good understanding of performance is achieved
  - hierarchical algorithms slightly better for ‘inter-group’ communications
- message simulator was a useful tool in understanding performance