Parallel Computing Platforms
Routing, Network Embedding

John Mellor-Crummey
Department of Computer Science
Rice University
johnmc@rice.edu
Topics for Today

- Routing
  - example network fabric: Infiniband
- Network embeddings
Communication Performance

• Depends on a variety of features
  — programming model semantics
    – synchronous or asynchronous communication
  — associated software protocols
    – get/put, send/receive
  — network topology
  — routing
Store-and-Forward Routing

• Definition
  — an intermediate hop completely receives a multi-hop message before forwarding it to the next hop

• Total communication cost for
  — message of size $m$ words
  — cost for header + message to traverse $l$ communication links
Packet Routing

- Store-and-forward makes poor use of communication resources
- Packet routing
  - breaks messages into packets
  - pipelines them through the network
- Packets may take different paths, thus each packet must carry
  - routing information
  - error checking
  - sequencing information
- Transmission time is much shorter, sum of
  - time for the first packet through the whole path
  - transmission time for the rest of the data
Cut-through Routing

- Packet routing in the extreme
  - divide messages into basic units called flits
  - start forwarding flit before it is received in its entirety
    - typically, after seeing the destination address in the first few bytes
    - pass along even bad packets: requires end-to-end error checking
- Flits are typically small → header information must be small
- To enable small headers
  - force all flits to take the same path, in sequence
    - tracer message first programs all intermediate routers
    - all flits then take same route
  - perform error checks on the entire message
    - not separately on flits
    - no sequence numbers needed
- Used in today’s Infiniband networks
Simplified Cost Model for Messages

- Valid for only uncongested networks
- If a link serves multiple messages
  - cost for transmission across the link must be scaled up by the number of messages traversing that link
- Network congestion varies by
  - communication pattern
  - match between pattern and network topology
- Communication models must account for congestion
Cost Models for Shared Memory Machines

Modeling shared memory communication cost is difficult

- Memory layout is typically determined by the system
- Finite cache sizes can cause cache thrashing
  — additional traffic because of failure to exploit close reuse
- Difficult to quantify overheads for coherence traffic
  — especially for multi-level caches with some shared levels
- Difficult to model
  — spatial locality
  — false sharing and contention
- Prefetching and multithreading can reduce exposed latency for data accesses
Routing Variants

• Method for choosing a path
  — oblivious: unique path between source and destination
    - determines route for each pair without considering traffic
    - appeal: can be computed in advance, even if they are computationally expensive
      optimal deterministic routing (with a minimal edge-forwarding index) in arbitrary networks is NP-hard [Saad ’93]
  — adaptive: use info about network state to determine path
    - must react quickly to changes in the global network
    - often constrained to fast suboptimal algorithms

• Path characteristics
  — minimal: shortest path between source and destination
  — non-minimal: may use longer path to avoid congestion
Routing a message from node $P_s (010)$ to node $P_d (111)$ in a 3D mesh using dimension-ordered routing.
Minimal vs. Non-minimal Routes

- (a) dimension ordered route: x then y
- (b) non-minimal route

WJ Dally, BP Towles. Principles and practices of interconnection networks. Elsevier. 2004
Routing Challenges and Approaches

- Routing must prevent deadlocks
  — use virtual channels with separate resources

- Routing must avoid hot-spots
  — can use two-step routing
    - message from $s \rightarrow d$
      first sent to a randomly chosen intermediate processor $i$
      then forward from $i$ to destination $d$

    - reduces a worst case permutation route to two randomized routing steps
      one with randomly picked source nodes
      a second with randomly picked destination nodes

Routing Costs: Theory vs. Practice

• Point-to-point metrics (bandwidth, latency) are not accurate predictors for average network bandwidth

• Widely-used aggregate theoretical metrics
  — edge-forwarding index: max # of paths through any edge
  — bisection bandwidth

• Practical techniques to increase performance
  — multiple virtual transmission channels
  — operating system bypass
  — adaptive routing

• Optimized routing offers substantial benefits
  — Infiniband: for some traffic patterns, congestion degrades bandwidth by 6.5x and increases latency 5x
  — can characterize routing algorithms by “effective bisection bandwidth”
Infiniband

- Infiniband: a switched, high-performance interconnect fabric
  - serial connection
  - high bandwidth, low latency
    - basic link rate: 2.5Gb/s (SDR)
    - multipliers: 4x, 12x
    - double data rate (DDR)
    - cut through routing (< 200ns switching*)
  - quality of service, failover, scalability

- Typical switching elements
  - crossbar with 24 full-duplex links
  - support virtual lanes to break routing cycles & avoid deadlock

- Does not mandate any topology
  - deployed networks typically use a fat tree, mesh, or torus

- Scalable to thousands of ports

*Mellanox Infiniscale III*
Infiniband Routing

- Oblivious destination-based distributed routing
- Each switch has a forwarding table (random or linear)
  - defines which port leads to which endpoint
  - on startup or after topology changes
    - Subnet Manager (SM) discovers the topology
    - computes forwarding table for each switch
    - uploads tables to the switches

- Credit-based flow control scheme in hardware
  - goal: avoid packet loss at congested switches
  - how: each output port can only send packets if it has credits at the destination input port
Infiniband OpenSM Routing Algorithms

- **MINHOP**
  - minimal paths among all endpoints; tries to balance the number of routes per link locally at each switch (under the constraint of minimal routes)
  - can be circular dependencies among switch buffers (admits deadlock)

- **UPDN**
  - uses the up*/down* routing algorithm to avoid circular buffer dependencies by restricting # of possible paths in the network
  - legal route must traverse zero or more links in “up” direction followed by zero or more links in “down” direction

- **FTREE**
  - a routing scheme optimized for fat trees; deadlock-free, but requires a fat tree network topology

- **DOR (Dimension Order Routing)**
  - routes along the dimensions of a k-ary n-cube to determine shortest paths and might create circular buffer dependencies

- **LASH (Layered Shortest Path)**
  - uses Virtual Lanes (VL) to break cyclic dependencies among channels of an underlying DOR scheme
P-SSSP Routing

• Single source shortest path based routing

• Algorithm iterates over all endpoints $u \in V_P$ and finds reverse shortest paths from $u$ to all other endpoints $v \in V_P$

\[
\begin{align*}
\text{Input: } & \text{Network } G = (V_P \cup V_c, E) \\
\text{Output: } & \text{Routes } R \\
1 & \text{foreach } u \in V_P \text{ do} \\
2 & \quad \text{comp. shortest paths from } u \text{ to all } v \in V_P \\
3 & \quad \text{add reverse paths to forwarding tables } (R) \\
4 & \quad \text{update edge weights along paths}
\end{align*}
\]

• Tree structure of the shortest path tree automatically generates valid destination-based routes

• After each endpoint, the algorithm updates edge weights with the number of routes that pass through each edge

• Difference between OpenSM’s MINHOP and P-SSSP
  — P-SSSP performs a global optimization of the edge loads
  — MINHOP does it only locally at each switch
Computing Routes with P-SSSP

Input: Network $G = (V_p \cup V_c, E)$
Output: Routes $R$

1. foreach $u \in V_p$ do
   2. compute shortest paths from $u$ to all $v \in V_p$
   3. add reverse paths to forwarding tables ($R$)
   4. update edge weights along paths

$V_p$ set of endpoints
$V_c$ set of crossbars

(b) From endpoint 0
(c) From endpoint 1
(d) From endpoint 2
P^2-SSSP

(a) The P^2-SSSP Algorithm

P-SSP: only updates weights P times
P^2-SSSP: more accurate greedy heuristic to minimize the edge-forwarding index:
perform the SSSP for each source-destination pair and updates the edge weights P(P – 1) times (one for each pair of endpoints)
Sample Infiniband Networks

- **Thunderbird** (Sandia National Laboratories)
  - 4390 endpoints, half-bisection bandwidth fat tree

- **Ranger** (Texas Advanced Computing Center)
  - 4080 endpoints, two Sun Magnum 3456 port switches, 5-stage full bisection bandwidth fat tree, 4 sockets (16 cores) / endpoint

- **Atlas** (LLNL)
  - 1142 endpoints, full bisection bandwidth fat tree

- **Deimos** (Technical University of Dresden)
  - 724 endpoints, three 288-port switches connected in a 30 link-wide chain

- **Odin** (Indiana University)
  - 128 endpoint cluster with a single switch (fat tree)
Routing Algorithm Performance Simulations

(a) $\Lambda(G, R)$ for different clusters
Routing vs. Infiniband B/W

- P-SSSP algorithm globally balances routes and thus improves the effective bandwidth of the network
- Dramatically improves effective bisection bandwidth
  - 25% on the 4080 endpoint Ranger cluster @ TACC (simulated)
  - 40% on Deimos 724-endpoint InfiniBand cluster (measured)
- Why is P²-SSSP not better?
  - edge forwarding index = theoretical lower bound to the minimal point-to-point bandwidth in the networks
    - not necessarily a good predictor for effective (avg) bisection BW
For fat trees, The bisection bandwidth is indicated by the oversubscription ratio, e.g., “1:2” means half bisection bandwidth.

Note: FTREE aborted with a segmentation fault on networks with more than two levels.
Global Adaptive Routing

- VAL gives optimal worst-case throughput
- MIN gives optimal benign traffic performance
- UGAL (Universal Globally Adaptive Load-balance)
  - [Singh '05]
  - Routes benign traffic minimally
  - Starts routing like VAL if load imbalance in channel queues
  - In the worst-case, degenerates into VAL, thus giving optimal worst-case throughput

B. Dally. From Hypercubes to Dragonflies. IAA Workshop, 2008.
UGAL

1. $H_m =$ shortest path (SP) length
2. $q_m =$ congestion of the outgoing channel for SP
3. Pick $i$, a random intermediate node
4. $H_{nm} =$ non-min path ($s \rightarrow i \rightarrow d$) length
5. $q_{nm} =$ congestion of the outgoing channel for $s \rightarrow i \rightarrow d$
6. Choose SP if $H_m q_m \leq H_{nm} q_{nm}$; else route via $i$, minimally in each phase
## UGAL report card

<table>
<thead>
<tr>
<th>Topology</th>
<th>Throughput (frac of capacity)</th>
<th>Algo</th>
<th>Θ benign</th>
<th>Θ adv</th>
<th>Θ avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 node topology</td>
<td></td>
<td>VAL</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIN</td>
<td>1.0</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UGAL</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$K_{64}$</td>
<td></td>
<td>VAL</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIN</td>
<td>1.0</td>
<td>0.33</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UGAL</td>
<td>1.0</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>8 x 8 torus</td>
<td></td>
<td>VAL</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIN</td>
<td>1.0</td>
<td>0.2</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UGAL</td>
<td>1.0</td>
<td>0.5</td>
<td>0.63</td>
</tr>
<tr>
<td>64 node CCC</td>
<td></td>
<td>VAL</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIN</td>
<td>1.0</td>
<td>0.2</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UGAL</td>
<td>1.0</td>
<td>0.5</td>
<td>0.63</td>
</tr>
</tbody>
</table>

B. Dally. From Hypercubes to Dragonflies. IAA Workshop, 2008.
adversarial traffic pattern: each node in a group sends to randomly selected node in another group.
Embeddings

Often need to embed a known communication pattern into a given interconnection topology

• Why?
  —may have an algorithm based on a particular logical topology
  —how should we use it on a machine with a different topology

• Thus, it is useful to understand mapping between graphs
Metrics for Graph Mappings

Mapping a graph $G(V,E)$ into $G'(V',E')$

- **Congestion** = maximum # edges in $E$ mapped onto edge in $E'$
- **Dilation** = maximum # edges in $E'$ mapped onto 1 edge in $E$
- **Expansion** = $(\# \text{ nodes in } V')/(\# \text{ nodes in } V)$
A Motivating Problem

Why FFT?

Often used in spectral methods - a class of techniques for numerically solving partial differential equations.
Binary Reflected Gray Code (RGC)

Adjacent entries $G(i, d)$, $G(i + 1, d)$ differ in only 1 bit

$G(i, x):$

- $G(0, 1) = 0$
- $G(1, 1) = 1$
- $G(i, x + 1) = \begin{cases} 
  G(i, x), & i < 2^x \\
  2^x + G(2^{x+1} - 1 - i, x), & i \geq 2^x 
\end{cases}$

1-bit Gray code  

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

2-bit Gray code

| 0 0 |
| 0 1 |
| 1 1 |
| 1 0 |

3-bit Gray code

| 0 0 0 |
| 0 0 1 |
| 0 1 1 |
| 0 1 0 |
| 1 1 0 |
| 1 1 1 |
| 1 0 1 |
| 1 0 0 |

$G(i, d)$

$d = \text{number of bits in code}$

Reflect along this line
Embedding a Ring into a Hypercube

• Given
  —ring of $2^d$ nodes
  —$d$-dimensional hypercube

• Map
  —node $i$ of the ring $\rightarrow$ node $G(i, d)$ of the hypercube
Embedding a Ring in a Hypercube

3-bit reflected Gray code ring

embedding into a 3D hypercube
Embedding a Mesh into a Hypercube

Map $2^r \times 2^s$ wraparound mesh into a $2^{r+s}$-node hypercube

- $G(k,d) = k^{th}$ element of Gray code of $d$ bits
- Let $||$ denote concatenation
- Map mesh node $(i, j) \rightarrow$ hypercube node $G(i, r - 1) || G(j, s - 1)$
Embedding a Mesh into a Hypercube

Mapping nodes in 4 x 4 mesh to nodes in a 4D hypercube

Congestion, dilation, and expansion of the mapping is 1
Embedding a Mesh into a Hypercube

Embedding a $2 \times 4$ mesh into a 3D hypercube

Congestion, dilation, and expansion of the mapping is 1
Embedding a Mesh into a Linear Array

• Mesh has more edges than a linear array
• No possible mapping with congestion = 1, dilation = 1
• Approach
  — first examine the mapping of a linear array into a mesh
  — invert above mapping
    – yields optimal mapping in terms of congestion
Embedding a Mesh into a Linear Array

Embedding a 16 node linear array into a 2-D mesh

Inverting the mapping - mapping a 2D mesh into a linear array (congestion 5)

Inverse of the mapping:
2D mesh to 16-node linear array

Key:
dark lines: links in the linear array
normal lines: links in the mesh.
Embedding a Hypercube into a 2-D

- How? invert the gray-code mesh to hypercube mapping
- Results in an optimal mapping
Embedding a Hypercube into a 2-D

$p = 16$

$p = 64$
References


