Outline

Scalability
  • physical, bandwidth, latency and cost
  • level of integration

Realizing Programming Models
  • network transactions
  • protocols
  • safety
    – input buffer problem: N-1
    – fetch deadlock

Communication Architecture Design Space
  • how much hardware interpretation of the network transaction?

Limited Scaling of a Bus

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Length</td>
<td>~ 1 ft</td>
</tr>
<tr>
<td>Number of Connections</td>
<td>fixed</td>
</tr>
<tr>
<td>Maximum Bandwidth</td>
<td>fixed</td>
</tr>
<tr>
<td>Interface to Comm. medium</td>
<td>memory inf</td>
</tr>
<tr>
<td>Global Order</td>
<td>arbitration</td>
</tr>
<tr>
<td>Protection</td>
<td>Virt -&gt; physical</td>
</tr>
<tr>
<td>Trust</td>
<td>total</td>
</tr>
<tr>
<td>OS</td>
<td>single</td>
</tr>
<tr>
<td>comm. abstraction</td>
<td>HW</td>
</tr>
</tbody>
</table>

Bus: each level of the system design is grounded in the scaling limits at the layers below and assumptions of close coupling between components
Workstations in a LAN?

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Bus</th>
<th>LAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Length</td>
<td>~ 1 ft</td>
<td>KM</td>
</tr>
<tr>
<td>Number of Connections</td>
<td>fixed</td>
<td>many</td>
</tr>
<tr>
<td>Maximum Bandwidth</td>
<td>fixed</td>
<td>???</td>
</tr>
<tr>
<td>Interface to Comm. medium</td>
<td>memory inf</td>
<td>peripheral</td>
</tr>
<tr>
<td>Global Order</td>
<td>arbitration</td>
<td>???</td>
</tr>
<tr>
<td>Protection</td>
<td>Virt -&gt; physical</td>
<td>OS</td>
</tr>
<tr>
<td>Trust</td>
<td>total</td>
<td>none</td>
</tr>
<tr>
<td>OS</td>
<td>single</td>
<td>independent</td>
</tr>
<tr>
<td>comm. abstraction</td>
<td>HW</td>
<td>SW</td>
</tr>
</tbody>
</table>

No clear limit to physical scaling, little trust, no global order, consensus difficult to achieve.

Independent failure and restart

Scalable Machines

What are the design trade-offs for the spectrum of machines between?

- specialize or commodity nodes?
- capability of node-to-network interface
- supporting programming models?

What does scalability mean?

- avoids inherent design limits on resources
- bandwidth increases with P
- latency does not
- cost increases slowly with P
Bandwidth Scalability

What fundamentally limits bandwidth?
• single set of wires
Must have many independent wires
Connect modules through switches
Bus vs Network Switch?

Dancehall MP Organization

Network bandwidth?
Bandwidth demand?
• independent processes?
• communicating processes?
Latency?
Generic Distributed Memory Org.

Network bandwidth?
Bandwidth demand?
  • independent processes?
  • communicating processes?
Latency?

Key Property

Large number of independent communication paths between nodes
=> allow a large number of concurrent transactions using different wires
initiated independently
no global arbitration
effect of a transaction only visible to the nodes involved
  • effects propagated through additional transactions
Latency Scaling

\[ T(n) = \text{Overhead} + \text{Channel Time} + \text{Routing Delay} \]

Overhead?

Channel Time\( (n) = \frac{n}{B} \quad --- \quad \text{BW at bottleneck} \]

Routing Delay\((h,n)\)

**Typical example**

max distance: \( \log n \)

number of switches: \( \alpha n \log n \)

overhead = 1 us, BW = 64 MB/s, 200 ns per hop

**Pipelined**

\[
T_{64}(128) = 1.0 \text{us} + 2.0 \text{us} + 6 \text{hops} * 0.2 \text{us/hop} = 4.2 \text{us}
\]

\[
T_{1024}(128) = 1.0 \text{us} + 2.0 \text{us} + 10 \text{hops} * 0.2 \text{us/hop} = 5.0 \text{us}
\]

**Store and Forward**

\[
T_{64}^{sf}(128) = 1.0 \text{us} + 6 \text{hops} * (2.0 + 0.2) \text{us/hop} = 14.2 \text{us}
\]

\[
T_{1024}^{sf}(128) = 1.0 \text{us} + 10 \text{hops} * (2.0 + 0.2) \text{us/hop} = 23 \text{us}
\]
Cost Scaling

cost(p,m) = fixed cost + incremental cost (p,m)

Bus Based SMP?
Ratio of processors : memory : network : I/O ?

Parallel efficiency(p) = Speedup(P) / P

Costup(p) = Cost(p) / Cost(1)

Cost-effective: speedup(p) > costup(p)
Is super-linear speedup

Cost Effective?

2048 processors: 475 fold speedup at 206x cost
Physical Scaling

Chip-level integration
Board-level
System level

nCUBE/2 Machine Organization

Entire machine synchronous at 40 MHz
CM-5 Machine Organization

Board-Level Integration Example

System Level Integration

IBM SP-2
Network Transaction Primitive

- one-way transfer of information from a source output buffer to a dest. input buffer
  - causes some action at the destination
  - occurrence is not directly visible at source
- deposit data, state change, reply

Bus Transactions vs Net Transactions

<table>
<thead>
<tr>
<th>Issues</th>
<th>V-&gt;P</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>protection check</td>
<td>wires</td>
<td>flexible</td>
</tr>
<tr>
<td>format</td>
<td>reg, FIFO</td>
<td>??</td>
</tr>
<tr>
<td>output buffering</td>
<td>global</td>
<td>local</td>
</tr>
<tr>
<td>media arbitration</td>
<td>simple</td>
<td>complex</td>
</tr>
<tr>
<td>destination naming and routing</td>
<td>limited</td>
<td>many source</td>
</tr>
<tr>
<td>input buffering</td>
<td>action</td>
<td>completion detection</td>
</tr>
<tr>
<td>action</td>
<td>transaction ordering</td>
<td></td>
</tr>
<tr>
<td>completion detection</td>
<td>delivery guarantees</td>
<td></td>
</tr>
</tbody>
</table>


Shared Address Space Abstraction

Fundamentally a two-way request/response protocol
- writes have an acknowledgement

Issues
- fixed or variable length (bulk) transfers
- remote virtual or physical address, where is action performed?
- deadlock avoidance and input buffer full
  coherent? consistent?

The Fetch Deadlock Problem

Even if a node cannot issue a request, it must sink network transactions.

Incoming transaction may be a request, which will generate a response.

Closed system (finite buffering)
Consistency

write-atomicity violated without caching

Key Properties of SAS Abstraction

Source and destination data addresses are specified by the source of the request
• a degree of logical coupling and trust
no storage logically “outside the application address space(s)”
  – may employ temporary buffers for transport
Operations are fundamentally request response
Remote operation can be performed on remote memory
• logically does not require intervention of the remote processor
Message passing

Bulk transfers
Complex synchronization semantics
  • more complex protocols
  • More complex action

Synchronous
  • Send completes after matching recv and source data sent
  • Receive completes after data transfer complete from matching send

Asynchronous
  • Send completes after send buffer may be reused

Synchronous Message Passing

Constrained programming model.
Deterministic! What happens when threads added?
Destination contention very limited.
User/System boundary?
Asynch. Message Passing: Optimistic

More powerful programming model
Wildcard receive => non-deterministic
Storage required within msg layer?

Asynch. Msg Passing: Conservative

Where is the buffering?
Contention control? Receiver initiated protocol?
Short message optimizations
### Key Features of Message Passing Abstraction

Source knows send data address, dest. knows receive data address
- after handshake they both know both

Arbitrary storage “outside the local address spaces”
- may post many sends before any receives
- non-blocking asynchronous sends reduces the requirement to an arbitrary number of descriptors
  - fine print says these are limited too

Fundamentally a 3-phase transaction
- includes a request / response
- can use optimistic 1-phase in limited “Safe” cases
  - credit scheme

### Active Messages

User-level analog of network transaction
- transfer data packet and invoke handler to extract it from the network and integrate with on-going computation

Request/Reply
Event notification: interrupts, polling, events?
May also perform memory-to-memory transfer
Common Challenges

Input buffer overflow
- N-1 queue over-commitment => must slow sources
- reserve space per source (credit)
  - when available for reuse?
    - Ack or Higher level
- Refuse input when full
  - backpressure in reliable network
  - tree saturation
  - deadlock free
  - what happens to traffic not bound for congested dest?
- Reserve ack back channel
- drop packets
- Utilize higher-level semantics of programming model

Challenges (cont)

Fetch Deadlock
- *For network to remain deadlock free, nodes must continue accepting messages, even when cannot source msgs*
  - what if incoming transaction is a request?
    - Each may generate a response, which cannot be sent!
    - What happens when internal buffering is full?
logically independent request/reply networks
- physical networks
- virtual channels with separate input/output queues
bound requests and reserve input buffer space
- K(P-1) requests + K responses per node
- service discipline to avoid fetch deadlock?
NACK on input buffer full
- NACK delivery?
Challenges in Realizing Prog. Models in the Large

One-way transfer of information

No global knowledge, nor global control
  • barriers, scans, reduce, global-OR give fuzzy global state

Very large number of concurrent transactions

Management of input buffer resources
  • many sources can issue a request and over-commit destination before any see the effect

Latency is large enough that you are tempted to “take risks”
  • optimistic protocols
  • large transfers
  • dynamic allocation

Many many more degrees of freedom in design and engineering of these system

Summary

Scalability
  • physical, bandwidth, latency and cost
  • level of integration

Realizing Programming Models
  • network transactions
  • protocols
  • safety
    – input buffer problem: N-1
    – fetch deadlock

Communication Architecture Design Space
  • how much hardware interpretation of the network transaction?
**Network Transaction Processing**

Key Design Issue:
How much interpretation of the message?
How much dedicated processing in the Comm. Assist?

**Spectrum of Designs**

<table>
<thead>
<tr>
<th>Spectrum of Designs</th>
<th>None: Physical bit stream</th>
<th>User/System</th>
<th>Remote virtual address</th>
<th>Global physical address</th>
<th>Cache-to-cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Physical bit stream</td>
<td>User-level port</td>
<td>Processing, translation</td>
<td>Proc + Memory controller</td>
<td>Cache controller</td>
</tr>
<tr>
<td></td>
<td>• blind, physical DMA</td>
<td>User-level handler</td>
<td>Paragon, Meiko CS-2</td>
<td>RP3, BBN, T3D</td>
<td>Dash, KSR, Flash</td>
</tr>
<tr>
<td></td>
<td>nCUBE, iPSC, . . .</td>
<td>J-Machine, Monsoon, . . .</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CM-5, *T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Increasing HW Support, Specialization, Intrusiveness, Performance (??)
Net Transactions: Physical DMA

DMA controlled by regs, generates interrupts
Physical => OS initiates transfers
Send-side
• construct system “envelope” around user data in kernel area
Receive
• must receive into system buffer, since no interpretation in CA

nCUBE Network Interface

independent DMA channel per link direction
• leave input buffers always open
• segmented messages
routing interprets envelope
• dimension-order routing on hypercube
• bit-serial with 36 bit cut-through
**Conventional LAN Network Interface**

![Diagram of Conventional LAN Network Interface]

**User Level Ports**

- initiate transaction at user level
- deliver to user without OS intervention
- network port in user space
- User/system flag in envelope
  - protection check, translation, routing, media access in src CA
  - user/sys check in dest CA, interrupt on system
User Level Network ports

Appears to user as logical message queues plus status
What happens if no user pop?

Example: CM-5

Input and output FIFO for each network
2 data networks
  tag per message
    - index NI mapping table
  context switching?

*T integrated NI on chip
iWARP also

<table>
<thead>
<tr>
<th>Os</th>
<th>50 cy</th>
<th>1.5 us</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>53 cy</td>
<td>1.6 us</td>
</tr>
<tr>
<td>interrupt</td>
<td></td>
<td>10us</td>
</tr>
</tbody>
</table>
**User Level Handlers**

- Hardware support to vector to address specified in message
  - message ports in registers

**iWARP**

- Nodes integrate communication with computation on systolic basis
- Msg data direct to register
- Stream into memory