

CS184b: Computer Architecture (Abstractions and Optimizations)

Day 16: May 6, 2005
Fast Messaging



Today

- Message Handling Requirements
- Active Messages
- Processor/Network Interface Integration

What does message handler have to do?

- Send
 - allocate buffer to compose outgoing message
 - figure out destination
 - address
 - routing?
 - Format header info
 - compose data for send
- put out on network
 - copy to privileged domain
 - check permissions
 - copy to network device
 - checksums?

What does message handler have to do?

- Receive
 - queue result
 - copy buffer from queue to privileged memory
 - check message intact (checksum)
 - message arrived right place?
 - Reorder messages?
 - Filter out duplicate messages?
- Figure out which process/task gets message
 - check privileges
 - allocate space for incoming data
 - copy data to buffer in task
 - hand off to task
 - decode message type
 - dispatch on message type
 - handle message

1990 Message Handling

- nCube/2 160 μ s (360ns/byte)
- CM-5 86 μ s (120ns/byte)
- (compare processors with 33ns cycle)
 - $86 \times 30 \approx 2600$ cycles

Functions: Destination Addressing/Routing

- Prefigure and put in pointer/object
 - hoist out of inner loop of computation
 - just a lookup
- TLB-like translation table
 - provide hardware support for common case

Functions: Allocation

- Why? In general,
 - messages may be arbitrarily long
 - many dynamic sized objects...
 - may not be expecting message ?
 - Remote procedure invocation
 - Remote memory request
- Hand off to OS
 - OS user/consumption asynchronous to user process (lifetime unclear)

Caltech CS184 Spring2005 -- DeHon

7

Functions: Allocation

- Pre-allocate outside of messaging
 - when expecting a message
 - ? Standard sizes for common cases?
- Avoid copying
 - use shared memory
 - issues with synchronization
 - direct from user space

Caltech CS184 Spring2005 -- DeHon

8

Functions: Ordering

- Network may reorder messages
 - multiple paths
 - with different lengths, congestion
- Not all tasks require ordering
 - (may be ordered at higher level in computation)
 - dataflow firing example
- What requires ordering?

Caltech CS184 Spring2005 -- DeHon

9

Functions: Idempotence

- Failed Acknowledgments
 - may lead to multiple delivery of same message
- Idempotent operations
 - bit-set, bit-unset,
- Non-idempotent operations
 - increment, exchange
- How make idempotent?
 - TCP example

Caltech CS184 Spring2005 -- DeHon

10

Functions: Protection

- Don't want messages from other processes/entities
 - give away information (get)
 - destroy state (put)
 - perform operation (transfer funds)

Caltech CS184 Spring2005 -- DeHon

11

Functions: Protection

- How manage?
 - Treat network as IO
 - OS mediates
 - (can) trust message stamps on network
 - Give network to user
 - messaging hardware tags with process id
 - filter messages on process tags
 - (can) trust message stamps because of hardware
 - Cryptographic packet encoding

Caltech CS184 Spring2005 -- DeHon

12

Functions: Checksum

- Message could be corrupted in transit
 - likely with high-bit rate, long interconnect
 - (multiple chips...multiple boxes...)
- Wrong bits
 - in address
 - in message id
 - in data
- Typically solve in hardware

Caltech CS184 Spring2005 -- DeHon

13

What does message handler have to do?

- Send
 - allocate buffer to compose outgoing message
 - figure out destination
 - address
 - routing?
 - Format header info
 - compose data for send
- put out on network
 - copy to privileged domain
 - check permissions
 - copy to network device
 - checksums

Not all messages require

Hardware support

Avoid (don't do it)

14

Caltech CS184 Spring2005 -- DeHon

Not all messages require
Hardware support
Avoid (don't do it)

What does message handler have to do?

- Receive
 - queue result
 - copy buffer from queue to privilege memory
 - check message intact (checksum)
 - message arrived right place?
 - Reorder messages?
 - Filter out duplicate messages?
- Figure out which process/task gets message
- check privileges
- allocate space for incoming data
- copy data to buffer in task
- hand off to task
- decode message type
- dispatch on message type
- handle message

Caltech CS184 Spring2005 -- DeHon

15

End-to-End

- Variant of the primitives argument
- Applications/tasks have different requirements/needs
- Attempt to provide in the network
 - mismatch
 - unnecessary
- Network should be minimal
 - let application do just what it needs

Caltech CS184 Spring2005 -- DeHon

16

Active Messages

- Message contains PC of code to run
 - destination
 - message handler PC
 - data
- Receiver pickups PC and runs
- [similar to J-Machine, conv. CPU]

Caltech CS184 Spring2005 -- DeHon

17

Active Message Dogma

- Integrate the data directly into the computation
- Short Runtime
 - get back to next message, allows to run directly
- Non-blocking
- No allocation
- Runs to completion
- ...Make fast case common

Caltech CS184 Spring2005 -- DeHon

18

User Level NI Access

- Avoids context switch
- Viable if hardware manages process filtering
 - Single process owns network?
 - Process tags used to filter into queues (compare process ID tags in virtually mapped cache...register renaming in SMT)

Caltech CS184 Spring2005 -- DeHon

19

Hardware Support I

- Checksums
- Routing
- ID and route mapping
- Process ID stamping/checking
- Low-level formatting

Caltech CS184 Spring2005 -- DeHon

20

What does AM handler do?

- | | |
|--|--|
| <ul style="list-style-type: none"> • Send <ul style="list-style-type: none"> – compose message <ul style="list-style-type: none"> • destination • receiving PC • data – copy/queue to NI | <ul style="list-style-type: none"> • Receive <ul style="list-style-type: none"> – pickup PC – dispatch to PC – handler dequeues data into place in computation – [maybe more depending on application] <ul style="list-style-type: none"> • idempotence • ordering • synchronization |
|--|--|

Caltech CS184 Spring2005 -- DeHon

21

Example: PUT Handler

- | | |
|--|--|
| <ul style="list-style-type: none"> • Message: <ul style="list-style-type: none"> – remote node id – put handler (PC) 0 – remote addr 1 – data length 2 – (flag addr) 3 – data 4 • No allocation • Idempotent | <ul style="list-style-type: none"> • Receiver: <pre>poll - r1 = packet_pres - beq r1 0 poll - r2=packet(0) // handler - branch r2 put_handler - r3=packet(1) // addr - r4=packet(2) // len - r6=packet+4 // first datum - r5=r6+r4 // end mdata - *r3=packet(r6) - r6++ - blt r6,r5 mdata - consume packet - goto poll</pre> |
|--|--|

Caltech CS184 Spring2005 -- DeHon

22

Example: GET Handler

- | | |
|---|---|
| <ul style="list-style-type: none"> • Message Request <ol style="list-style-type: none"> 1. remote node 2. get handler 3. local addr 4. data length 5. (flag addr) 6. local node 7. remote addr | <ul style="list-style-type: none"> • Message Reply can just be a PUT message <ul style="list-style-type: none"> – put into specified local address |
|---|---|

Caltech CS184 Spring2005 -- DeHon

23

Example: GET Handler

- | | |
|--|---|
| <pre>get_handler - out_packet(0)=packet(6) - out_packet(1)=put_handler - out_packet(2)=packet(3) - out_packet(3)=packet(4) - r6=4 - r7=packet(7) - r5=packet(4) - r5=r5+4 mdata - packet(r6)=*r7 - r6++ - r7++ - blt r6,r5 mdata - consume packet - send message - goto poll</pre> | <ul style="list-style-type: none"> • Message Request <ol style="list-style-type: none"> 1. remote node 2. get handler 3. local addr 4. data length 5. (flag addr) 6. local node 7. remote addr |
|--|---|

Caltech CS184 Spring2005 -- DeHon

24

Example: DF Inlet synch

- Consider 3 input node (e.g. add3)
 - “inlet handler” for each incoming data
 - set presence bit on arrival
 - compute node when all present

Example: DF Inlet Synch

- inlet message
 - node
 - inlet_handler
 - frame base
 - data_addr
 - flag_addr
 - data_pos
 - data
- Inlet
 - move data to addr
 - set appropriate flag
 - if all flags set
 - enable DF node computation
- ? Care not enable multiple times?

Interrupts vs. Polling

- What happens on message reception?
- Interrupts
 - cost context switch
 - interrupt to kernel
 - save state
 - force attention to the network
 - guarantee get messages out of input queue in a timely fashion

Interrupts vs. Polling

- Polling
 - if getting many messages to same process
 - message handlers short / bounded time
 - may be fine to just poll between handlers
 - requires:
 - user-level/fine-grained scheduling
 - guarantee will get back to
 - avoid context switch cost

Interrupts vs. Polling

- Can be used together to minimize cost
 - poll network interface during batch handling of messages
 - interrupt to draw attention back to network if messages sit around too long
 - polling works for same process
 - interrupt if different process
 - common case is work on same process for a while

AM vs. JM

- J-Machine handlers can fault/stall
 - touch futures...
- J-Machine fast context with small state
 - not get to exploit rich context/state
- AM exploits register locality by scheduling together larger block of data
 - processing related handlers together (same context)
 - more in two week (look at TAM)

Active Message Results

- CM5 (user-level messaging)
 - send 1.6 μ s [50 instructions]
 - receive/dispatch 1.7 μ s
- nCube/2 (OS must intervene)
 - send 11 μ s [21 instructions]
 - receive 15 μ s [34 instructions]
- Myrinet (GM)
 - 6.5 μ s end-to-end GMs
 - 1-2 μ s host processing time

Caltech CS184 Spring2005 -- DeHon

31

Hardware Support II

- Roll presence tests into dispatch
- compose message data from registers
- common case
 - reply support
 - message types
- Integrate network interface as functional unit

Caltech CS184 Spring2005 -- DeHon

32

Presence Dispatch

- Handler PC in common location
- Have hardware supply null handler PC when no messages current
- Poll:
 - read MsgPC into R1
 - branch R1
- Also use to handle cases and priorities
 - by modifying a few bits of dispatch address
 - e.g. queues full/empty

Caltech CS184 Spring2005 -- DeHon

33

Compose from Registers

- Put together message in registers
 - reuse data from message to message
 - compute results directly into target
 - user register renaming and scoreboarding to continue immediately while data being queued

Caltech CS184 Spring2005 -- DeHon

34

Common Case Msg/Replies

- Instructions to
 - fill in common data on replies
 - node address, handler?
 - Indicate message type
 - not have to copy

Caltech CS184 Spring2005 -- DeHon

35

Example: GET handler

- Get_handler
 - R1=i0 // address from message register
 - R2=*R1
 - o2=R2 // value into output data register
 - SEND -reply type=reply_mesg_id
 - NEXT

Caltech CS184 Spring2005 -- DeHon

36

AM as Primitive Model

- Value of Active Messages
 - articulates a model of what primitive messaging needs to be
 - identify key components
 - then can optimize against
 - how much hardware to support?
 - What should be in hardware/software?
 - What are common cases?
 - Should get special treatment?

Caltech CS184 Spring2005 -- DeHon

37

Big Ideas

- Primitives/Mechanisms
 - End-to-end/common case
 - don't burden everything with features needed by only some tasks
- Abstract Model
- Separate essential/useful work from overhead

Caltech CS184 Spring2005 -- DeHon

38

[MSB-1] Ideas

- Minimize Overhead
 - moving data around is not value added operation
 - minimize copying
- Overlap Compute and Communication
 - queue
 - don't force send/receive rendezvous
- Get the OS out of the way of **common** operations

Caltech CS184 Spring2005 -- DeHon

39