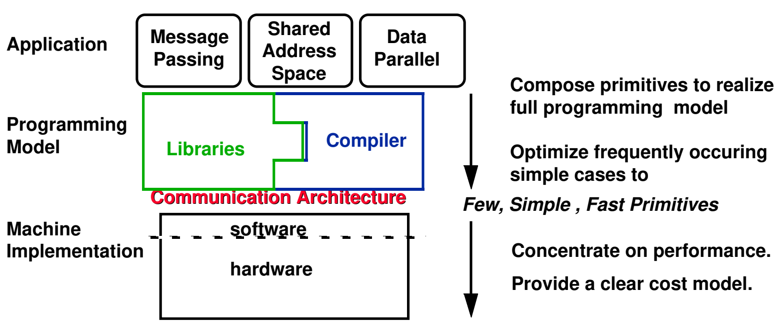
Efficient Communication in MPPs and NOWs

David E. Culler
Computer Science Division
University of California, Berkeley
culler@cs.berkeley.edu

A RISC view on Communication



Goals:

- expose the performance capabilities of underlying hardware in a simple, understandable form.
- support a variety of programming models.

Difference in concerns

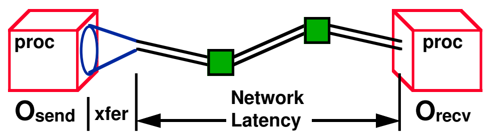
- MPI
  - process groups, task parallelism, . . .
  - communication contexts, matching communicators, . . .
  - application topologies, message selectivity, . . .
  - 12 combinations of send and receive
  - datatype definitions, general array sections, . . .
  - compatability with NX, Vertex, MPL, PVM, Express, . . .

- Active Messages
  - overhead, latency, bandwidth, deadlock, storage usage

Outline

- Active Message concepts and performance
- Relationship to traditional message passing
- Supporting higher level programming models
- Engineering an Active Message Layer
- Towards emerging Networks of Workstations

Components of Communication Performance



T(n) = Overhead_{send}+ Transfer + Latency + $\text{Overhead}_{receive}$

Overhead = Overhead_{send} + Overhead_{recv}

– “time processor spends interacting with communication substrate”

Transfer = n / bandwidth = n * **G**

– “time to squeeze msg onto skinniest wire”

Latency = **L** * Hops

– “time for a bit to cross the network”

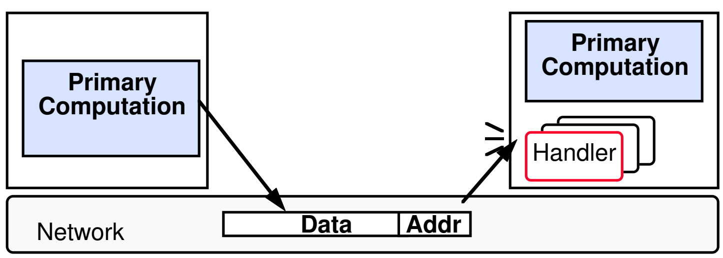
Communication Technology Trends

Machine	Year	*Overhead* µs	*L* µs/hop	*G* µs/wd	Overhead Cycles	FLOPs
nCUBE/10	87	400	–	10	4000	60
iPSC/2	88	700	11	1.4	10000	225
short		350	"	"	5000	112
nCUBE/2	90	150	2.3	1.8	3000	330
iPSC/860	91	160	11	1.4	6400	3200
Delta	91	55	0.5	0.4	2100	1100
CM5	92	95	0.2	0.4	3200	310
Paragon	93	250 (1.1)	0.1	0.1	12500	6500
SP-1	93	45(MPLp)	0.1	0.5	3000	1000
SS10 PVM	94	2,500	10-42	0.4	125000	25000

Gain with Active Messages

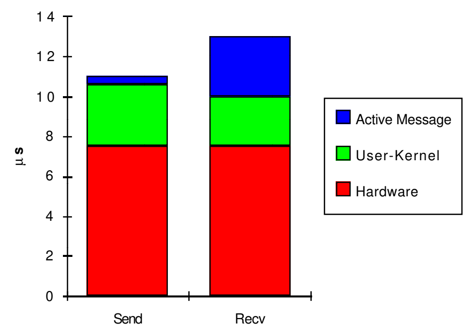
Machine	Year	*Overhead* µs	*L* µs/hop	*G* µs/wd	Overhead Cycles	FLOPs
nCUBE/10	87	400	–	10	4000	60
iPSC/2	88	700	11	1.4	10000	225
short		350	"	"	5000	112
nCUBE/2	90	150	2.3	1.8	3000	330
w/ A M.		**25**	"	"	**500**	**55**
iPSC/860	91	160	11	1.4	6400	3200
Delta	91	55	0.5	0.4	2100	1100
CM5	92	95	0.2	0.4	3200	310
w/ A.M.		**3**	"	"	**100**	**10**
Paragon	93	250 (1.1)	0.1	0.1	12500	6500
SP-1	93	45(MPLp)	0.1	0.5	3000	1000

Active Message Concept



- Sender injects the message directly into the network
- Associates a <u>small</u> user-level handlerwith each message
- Handler executes immediately upon arrival
  - pulls the message out of the network and integrates it into the ongoing computation, or replies (as determined by higher level programming model)
- No buffering, parsing, or allocation (beyond transport).

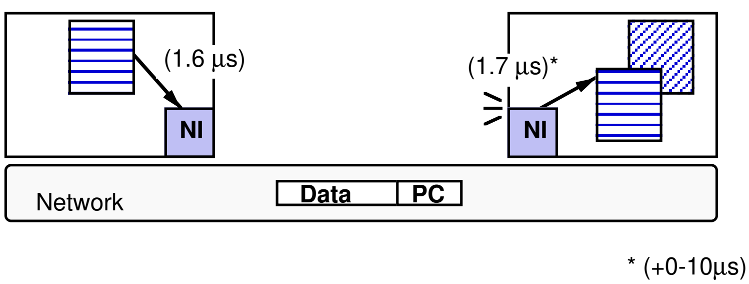
nCUBE/2 costs (von Eicken)



- system level physical DMA with interrupt on completion
- tight user/kernel interface
- simple memory-memory communication arch. exposes the full hardware potential

SHPCC.9

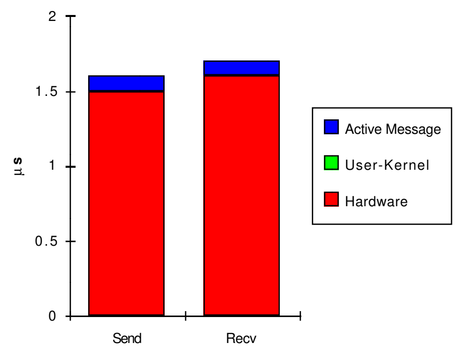
CM5 AM register – register communication arch



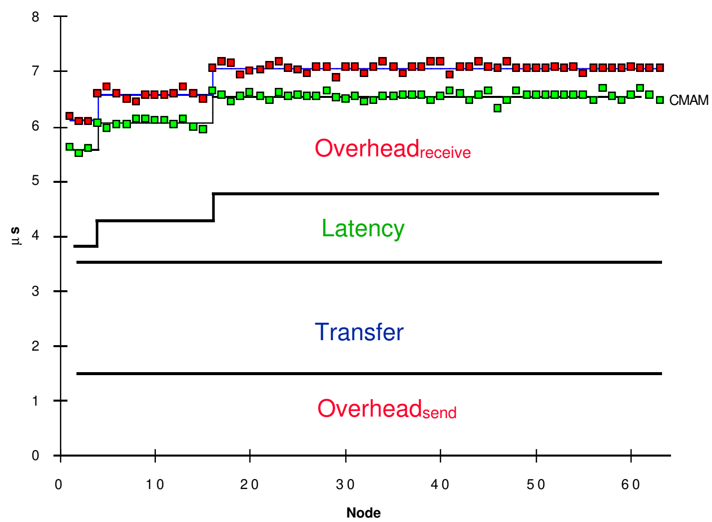
- user level memory mapped network FIFOs
- interrupting and polled arrival
- small 40 byte packets
- CMAM segments support bulk transfer and gather/scatter

SHPCC.10

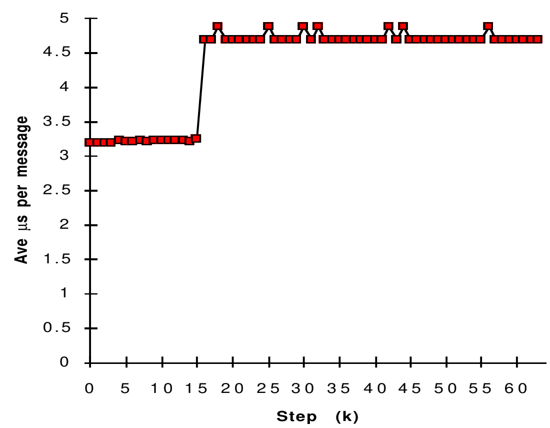
CMAM Costs (von Eicken)



CM-5 One-way Message Timing

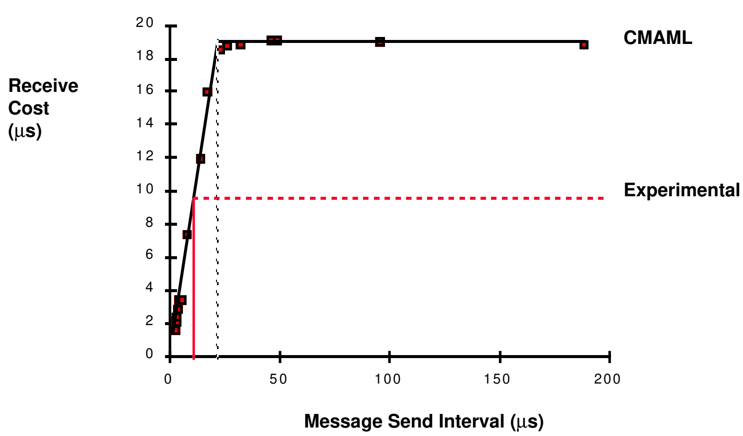


CMAM Performance under load



In step *k*, each processor *p* send several messages to processor *p* xor *k*.

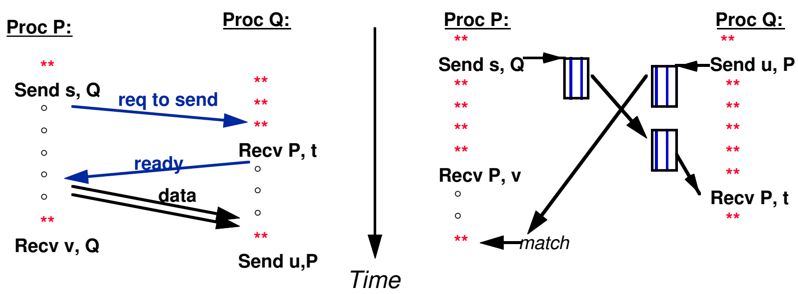
Cost of the interrupt



Where do all the cycles go?

Synchronous Send&Receive

Asynchronous Send&Receive

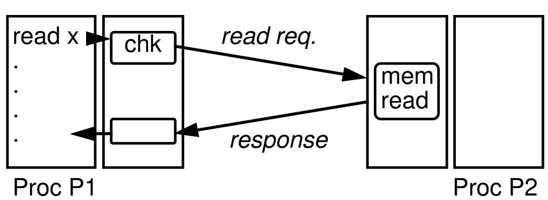


Fundamental Costs:
- 3 Active Messages
- user level protocol

Fundamental Costs:
- Copying
- Buffer management
- Matching

SHPCC.15

Shared Memory



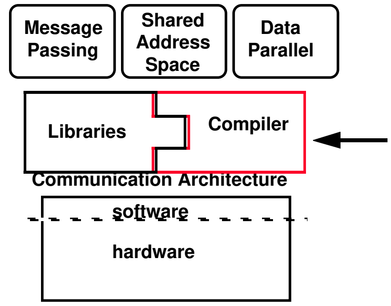
SHPCC.16

Active Message Summary

- "thin" communication abstraction
  - very close to what the hardware actually does
- simple primitives
  - highly optimized for specific hardware
- basis for an efficient implemention of many programming models
  - pay for what you actually use

SHPCC.17

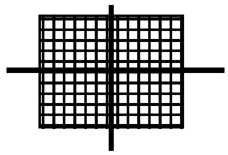
PART II: Supporting Programming Models

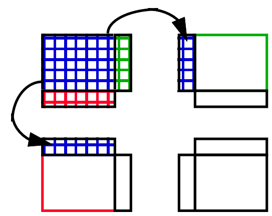


SHPCC.18

Global Address Space to Message Passing

HPF,
Fortran-D,





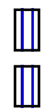
FORALL . . .

= A[f(i),g(i)]

Send, Send, . . .
Compute local

Recv, Recv, . . .
Compute

Barrier



?

° Compiler already allocated the buffer space!

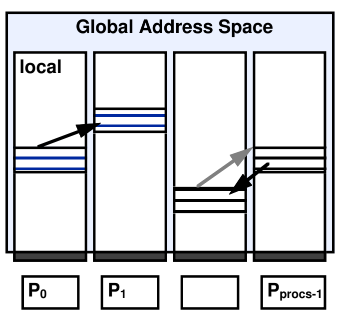
° Compiler knows which processor and what address to store into!

° Compiler knows if it is available!

=> Construct simple memory-to-memory transfer (i.e., write, put, or store)

° For user decomposition, use read or get!

Split-C: deliver efficient comm. to the application



- Explicitly parallel C
- 2D global address space
  - linear ordering on local spaces
- Local and Global pointers
  - spread arrays too
- Read/Write
- Get/Put

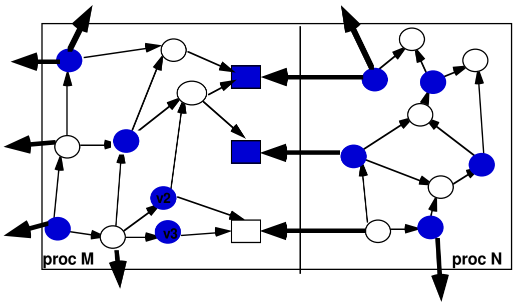
  x := G; . . .

  sync();
- Signaling store

  G :– x; . . .

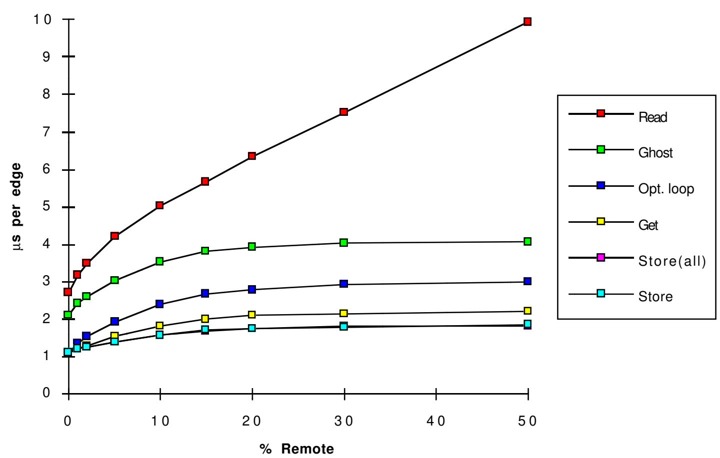
  store_sync(); or all_store_sync();
- Bulk transfer & global comm.

EM3D Example

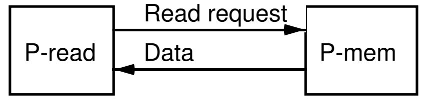



```
void all_compute_e()
{   node_t *n;
    int i;
    for (i = 0; i < boundary_nodes; i++) *ghost[i] := *gptr[i];
    sync();
    for (n = e_nodes; n; n = n->next) {
      for (i = 0; i < edge_count; i++)
         n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
```


EM3D Performance



Implementation of Read on CMAM



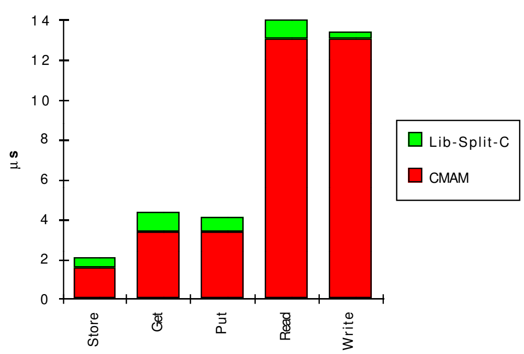

```
volatile int SM_msgs = 0;
static int read_buf;
void SM_read_i  ( int node,  int *addr )
{
    SM_msgs++;
    CMAM( node, SM_read_i_handler, ME, addr );
    wait (SM_msgs);
    return(read_buf);
}
```

```
void SM_read_i_handler( int rnode, int *addr )
{   CMAM_reply( rnode, SM_read_resp, *addr ); }
```

```
void SM_read_res(int data)
{   read_buf = data;
    SM_msgs--; }
```

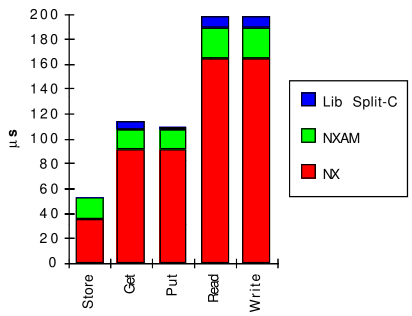
SHPCC.23

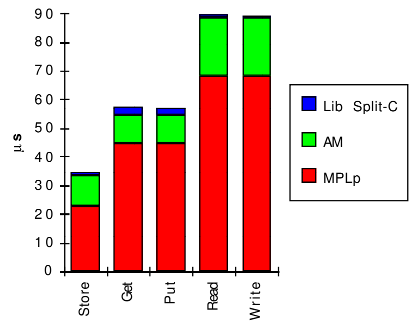
Performance of Split-C Primitives on CMAM



SHPCC.24

Split-C on Msg Passing

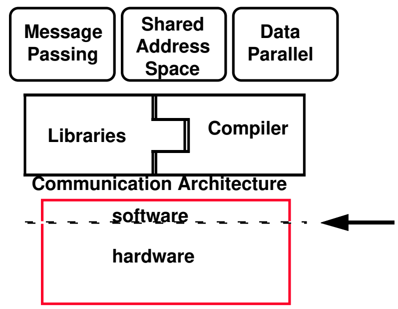




Summary of Split-C Study

- Simple global address space + *passive events* is powerful.
- Optimize communication against simple cost model.
  - perhaps the right target for higher level languages
- Implementation is straightforward
  - no heroic compiler
  - functionality of Msg Passing libs of little value
  - events are costly with pure read/write
- Short message ≡ Argument registers
  - keep the overhead and latency low for these!
- Bandwidth matters on bulk transfer, not form.
- Interesting variations in global communication ops.

PART III: Engineering an Active Message Layer



SHPCC.27

Key Issues for Reliable Networks

- Modern networks are deadlock free as long as processors continue to accept messages
- Sends (into the network) stall
  – destination contention will cause back-up
- Machine implementation layer should operate within fixed resources.
- Most higher level programming models require request/ response protocol.

- Access to network interface costs like a cache miss!

SHPCC.28

CMAM Solution

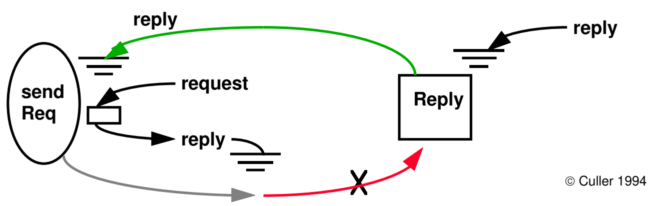
- Send (inject) tries till successful
  - continue to handle receives while trying to send (don't buffer)
  - handler cannot send (request handler can reply)
- Two networks viewed as request and reply nets
  - send-request accepts requests and replies
  - reply only accepts replies

=> semantics: send may block, but will receive

=> direct handler concept is valuable

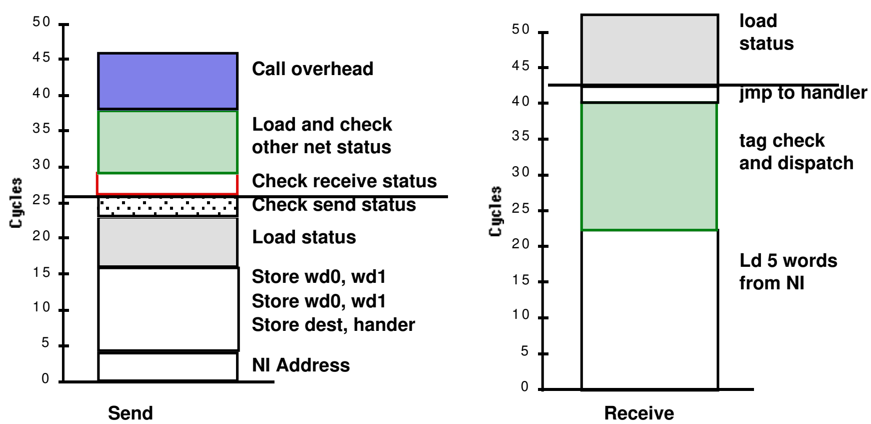
=> no software storage allocation

=> NI and network provide buffering for bursts



SHPCC.29

CMAM Cost Breakdown



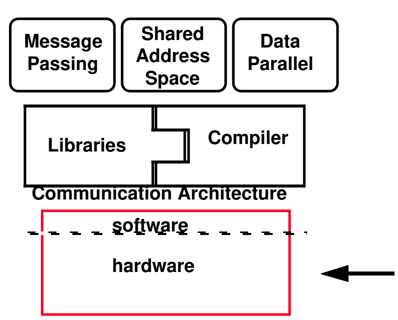
SHPCC.30

Engineering Summary

- the communication abstraction does matter
- concentrate on few simple primitives
- key communication issues can be addressed in a few instructions
  - about 20 for send and 20 for receive
  - most of the cycles are data access and HW status
- carefully control storage allocation in the machine implementation layer
  - there is room to enhance CMAM a little

SHPCC.31

PART IV: Towards SHPCC on a NOW



SHPCC.32

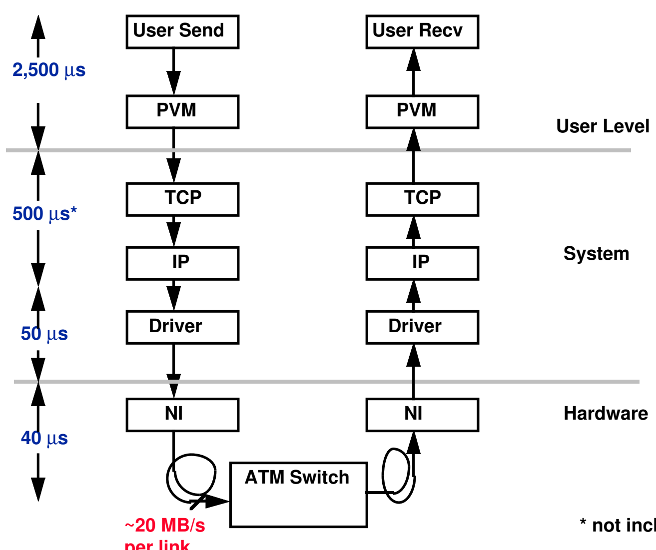
SHPCC cannot ignore Networks of Workstations

- Processor performance is better than MPPs at any time
  - 2 year lag with 1.5x improvement / year => 2.25x
- Cost
- Software and system support

- ATM switch bandwidth
  - per link BW almost competitive with last year's MPPs (20MB/s)
  - spec calls for 4x improvement soon.
  - scales with number of processors (in theory)
- Small cells and simple routing allow for low latency
  - G*n for 5+48 byte cell at 155mb/s is 2.8µs
  - switches have high degree (16x16 or 64x64)
  - few store-and-forward cell delays is competitive

SHPCC.33

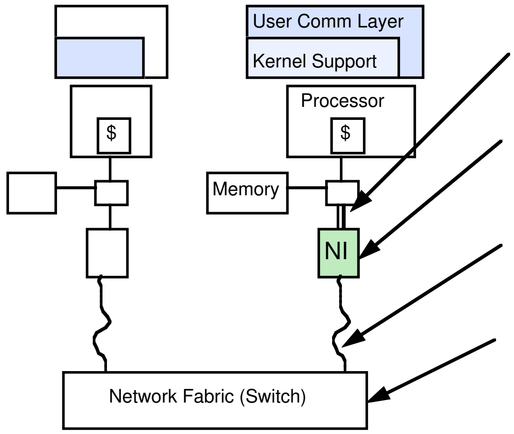
PVM over ATM



* not including set up

SHPCC.34

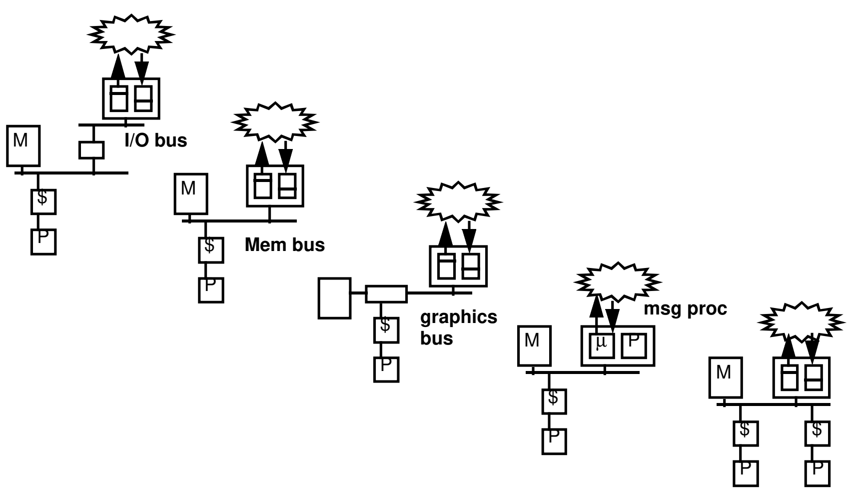
Current hardware limitations



- Standard I/O busses
  - high latency
  - low bandwidth
- Network Interface cards
  - slow
  - very complex
- Optical links
  - plenty of bandwidth
  - latency is high today
- Network switches
  - many cell delays
  - media is nearly reliable
  - drop packets on contention
  - flow control ?

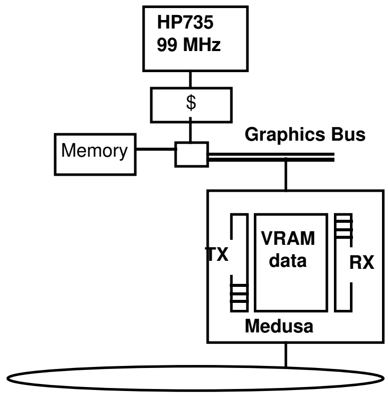
SHPCC.35

Rich Spectrum of Alternatives



SHPCC.36

A Prototype Design Study



Access costs:
- **2 cycle buffered store**
- **38 MB/s store BW**
- **42 cycle load latency**
- **18 MB/s load BW**

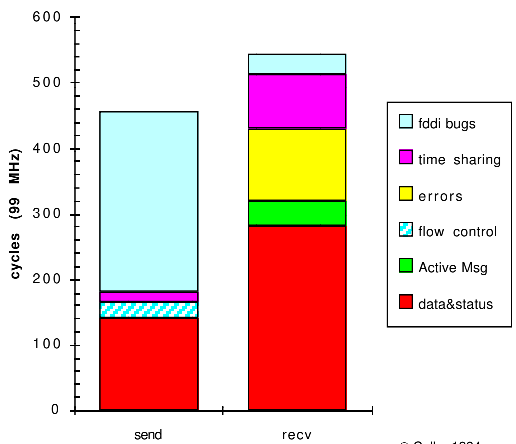
NIC:
- **MPP-like interface**
  - **load status (tx)**
  - **store data**
  - **go**
  - **load status**
  - **load data**
- **Lots of storage**
  - **retains send data**
  - **continuous receive**

SHPCC.37

HPAM Costs

- 1.7 + 3 µs send overhead
- 5 µs receive overhead
- 10µs transfer time

(5 wd msg)



SHPCC.38

Issues for nearly reliable networks

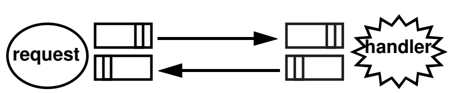
- Must accept messages at all times
  - contention (buffer full) causes loss
- Sends may fail
  - must retain data for retry
  - easier to bulk transfers than reg-reg messages

=> Exploit request/response nature and simple buffer management to minimize overhead and loss

- Access to network interface costs more than a cache miss!

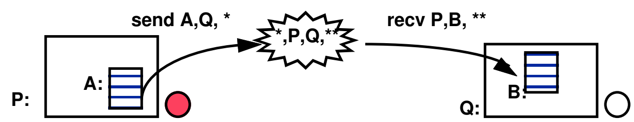
HPAM Highlights

- Data written to NIC will be delivered
- Simple point-to-point request/response buffer scheme with sequence numbers
  - requester: free buffer => prev. request/reply success and receive buffer available
    - » otherwise, engage time-out and retry from NIC buffer
  - reply: next request => prev. reply sucess
    - » same => resend reply w/o invoking handler
- PID in message



Summary

- Modern MPPs have high performance communication hardware
  - demand that the performance be delivered
  - it doesn’t have to be 1000s of cycles of overhead
  - Active Messages provide and existence proof.
- Few, simple, low level primitives is key
- Take a critical look at traditional message passing



  - an abstraction of what machines do?
  - a basis for higher level programming models (MPI?, HPF?)
- Keep your eye on the killer NOWs