Large-Scale Distributed Memory Design: Shared Physical Address Space

© Culler 1995

Direct Interpretation of the Net. Transactions

- None: Physical bit stream
  - blind, physical DMA — nCUBE, iPSC, . . .
- User/System
  - User-level port — CM-5, *T
  - User-level handler — J-Machine, Monsoon, . . .
- Remote virtual address
  - Processing, translation — Paragon, Meiko CS-2

⇒ *Global physical address*

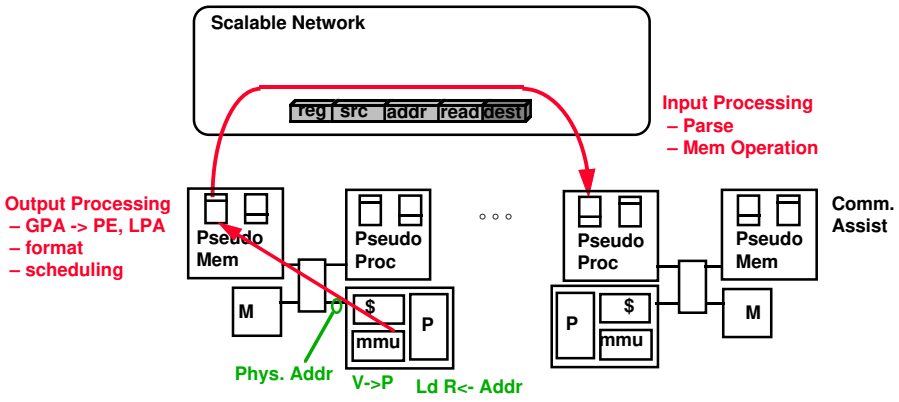
  - *Proc + Memory controller* — *RP3, BBN, T3D*

- Cache-to-cache
  - Cache controller — Dash, KSR, Flash

Increasing HW Support, Specialization, Intrusiveness, Performance (???)

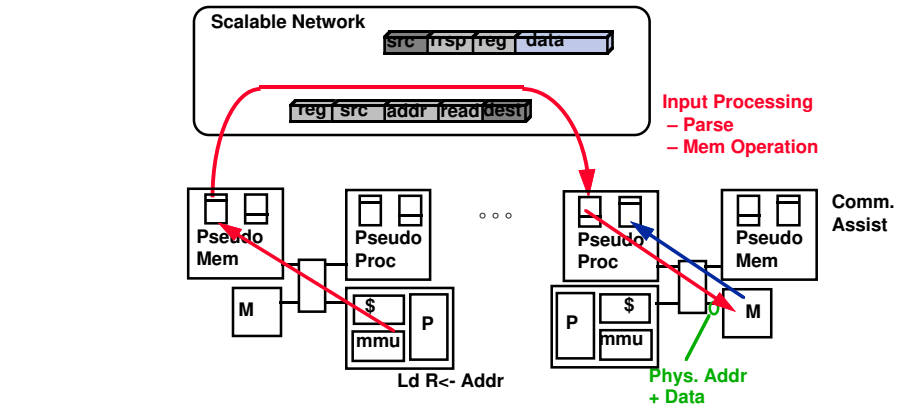
© Culler 1995

Shared Physical Address



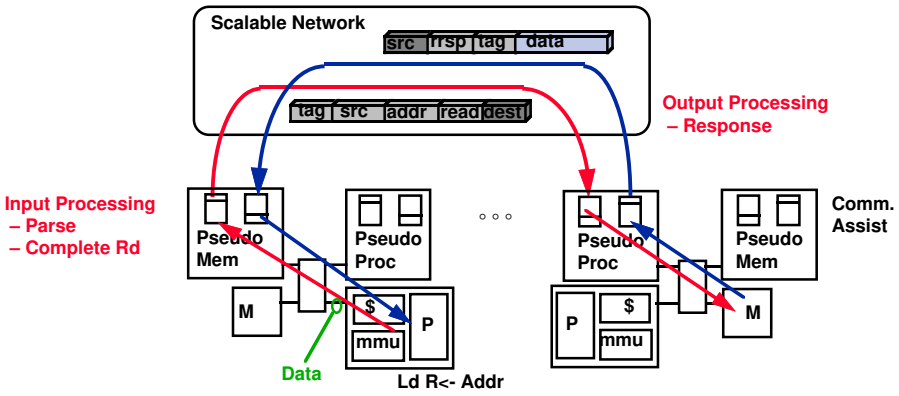
© Culler 1995

Shared Physical Address



© Culler 1995

Shared Physical Address



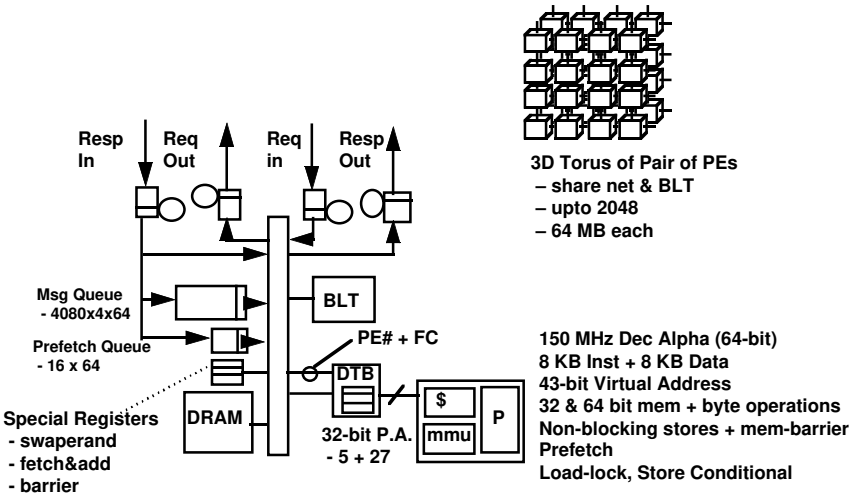
© Culler 1995

Key Distinction

- Network Transactions are initiated as a result of conventional memory operations
- Fixed set of message formats
  - interpreted directly in comm. assist hardware
- Request / Response
- Requires “dual ported” memory access
- Source bus transaction open till network transactions complete
-
- Examples: C.mmp, IBM RP3, BBN Butterfly, Cray T3D

© Culler 1995

Cray T3D



© Culler 1995

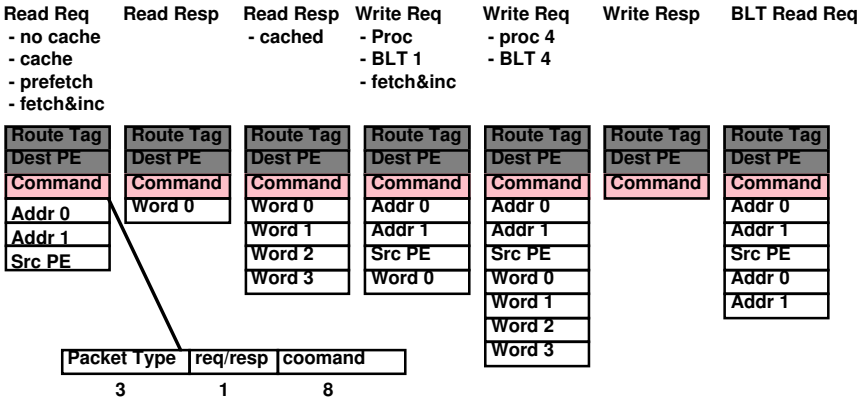
The Cray T3D

- ≤ 2048 Alphas (150 MHz, 16 or 64 MB each) + fast network
  - 43-bit virtual address space, 32-bit physical
  - 32-bit and 64-bit load/store + byte manipulation on regs.
  - no L2 cache
  - non-blocking stores, load/store re-ordering, memory fence
  - load-lock / store-conditional
- Direct global memory access via external segment regs
  - DTB annex, 32 entries, remote processor number and mode
  - atomic swap between special local reg and memory
  - special fetch&inc register
  - global-OR, global-AND barriers
- Prefetch Queue
- Block Transfer Engine
- User-level Message Queue

© Culler 1995

Packet Formats

- Many Types interpreted by input controller



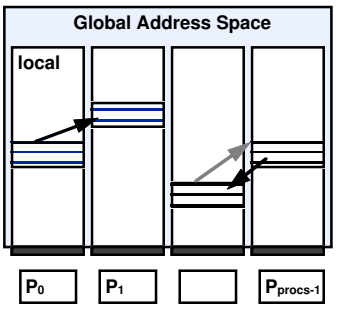
© Culler 1995

Methodology

- Shell provides mechanisms that support each facet of language
  - many mechanisms for single function in some case
- What is the performance?
- What should you use in compiling a parallel language?
-
-
- 1. Micro-benchmark machine primitives to identify realizable performance
- 2. Select code-generation strategy
- 3. Evaluate implementation of language primitive

© Culler 1995

Split-C in a Nutshell



- Explicitly parallel C
- 2D global address space
  - linear ordering on local spaces
- Local and Global pointers
- Read/Write
- Get/Put - to hide communication
  - x := G; . . .
  - sync();
- Bulk transfer
- Global ops (barrier)
- Signaling store
  - G :– x; . . .
  - store_sync(); or all_store_sync();
- Distributed (spread) arrays

© Culler 1995

Microbenchmarking Memory Access Time

Microbenchmark Template

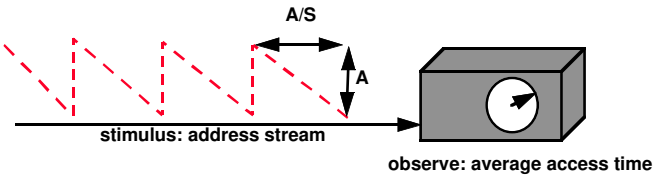
repeat to obtain confidence

repeat to mitigate timer resolution

for array size A from 4 KB to 8 MB doubling each time

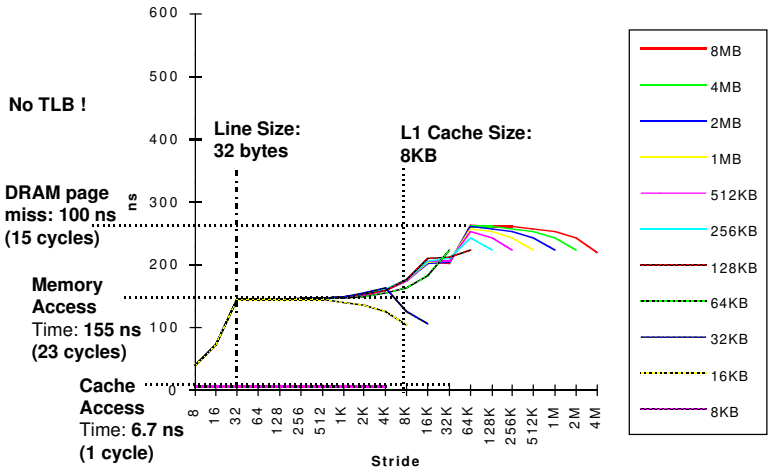
for stride S from 1 to A/2 doubling each time

memory operation on A[i]



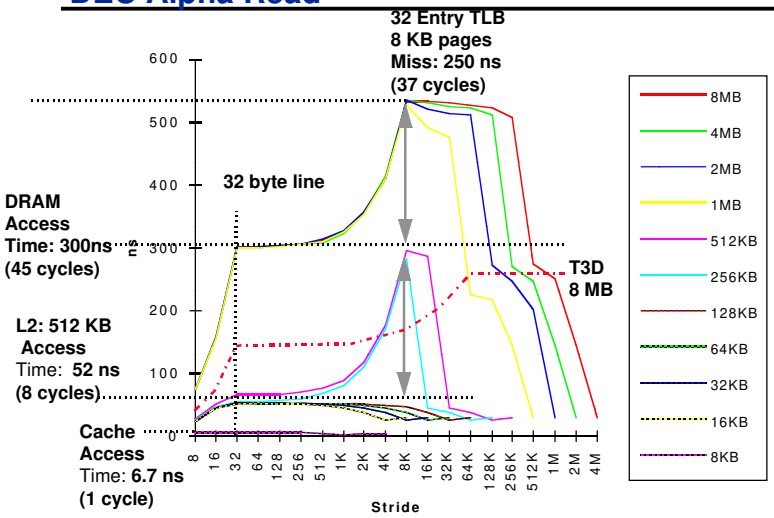
© Culler 1995

T3D Local Read (average latency)



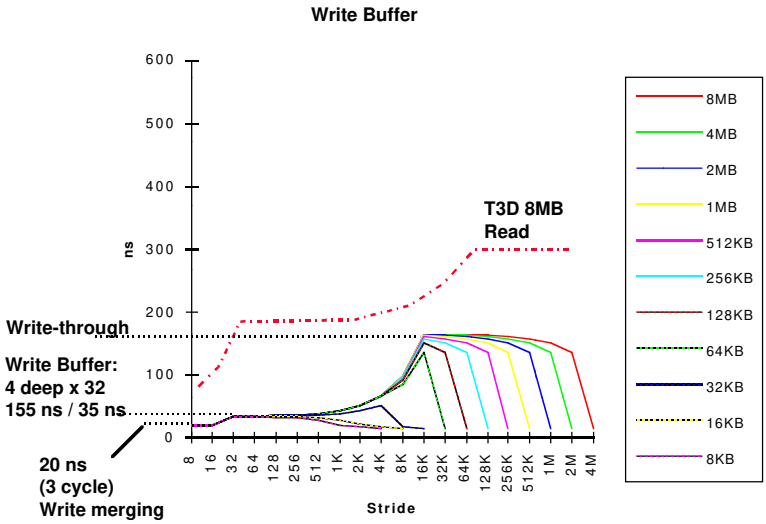
© Culler 1995

DEC Alpha Read



© Culler 1995

T3D Local Write Cost



© Culler 1995

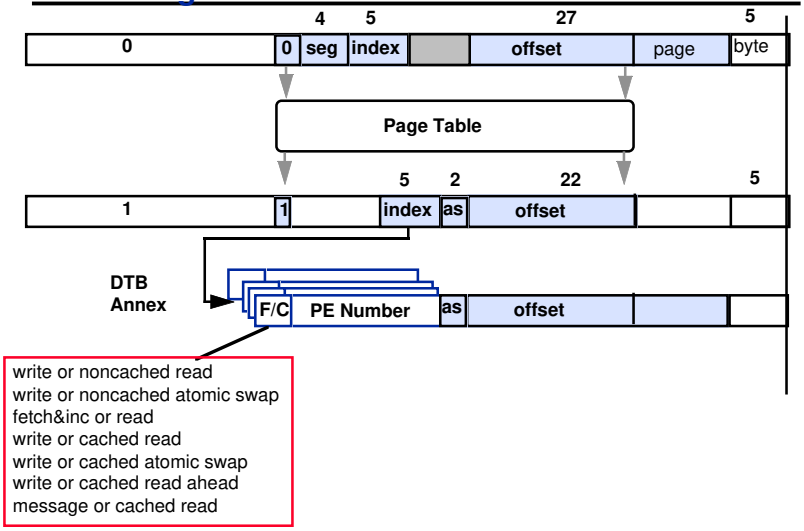
Global Pointers

- Language requirements
  - dereference: read / write global location
  - transfer: read / write / pass the pointer itself
  - arithmetic
    - » local addressing: acts on offset portion only
    - » spread addressing: acts on processor portion with wrap into offset
  - extraction and construction
    - » processor and offset components
  - null test
- T3D Design Constraints
  - 64 MB x 2048 Proc => 37 bits of address
  - Alpha provides 43 bits virtual address => 32 bit physical
  - T3D introduces external registers to expand physical address
    - » 23 cycle update using store-conditional instructions

  => Compiler must manage these registers

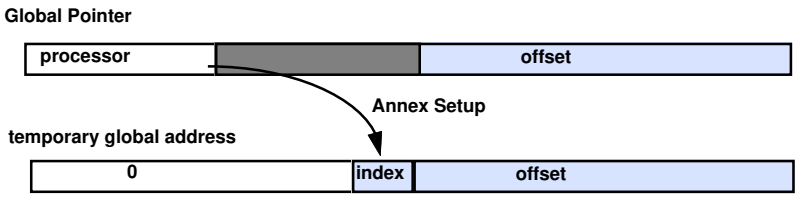
© Culler 1995

T3D Design Constraints



© Culler 1995

Global Pointer Representation

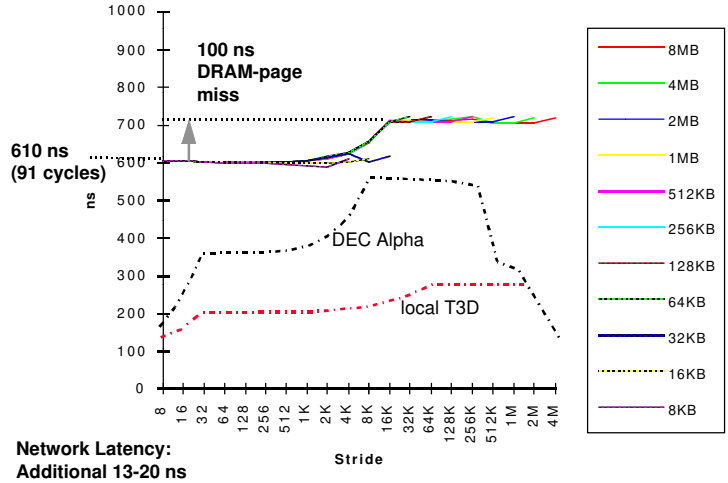


- How to manage Annex registers?
  - use one and update on each access
  - use several and avoid updates (compiler mgmt or RT map)
    - » consumes TLB entries (no problem)
    - » physical synonyms !!! – two annex entries for same processor
    - » direct map cache => fall into same line
    - » write-buffer => write one and then read stale value from the other.

© Culler 1995

Remote Read Uncached

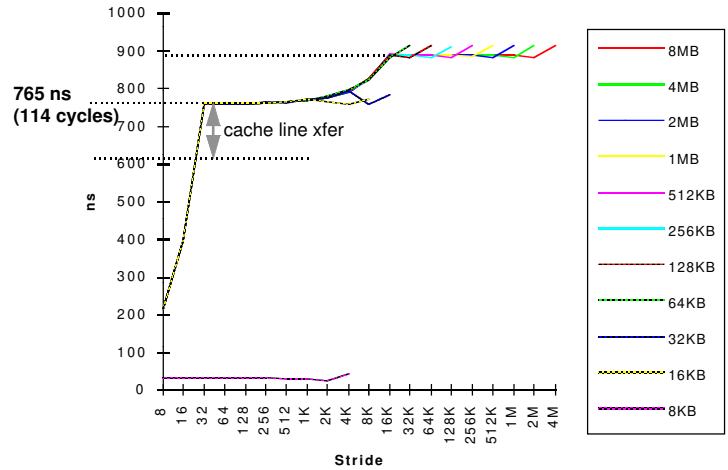
3 - 4x Local Memory Read !



Network Latency:
Additional 13-20 ns
(2-3 cycles) per hop

© Culler 1995

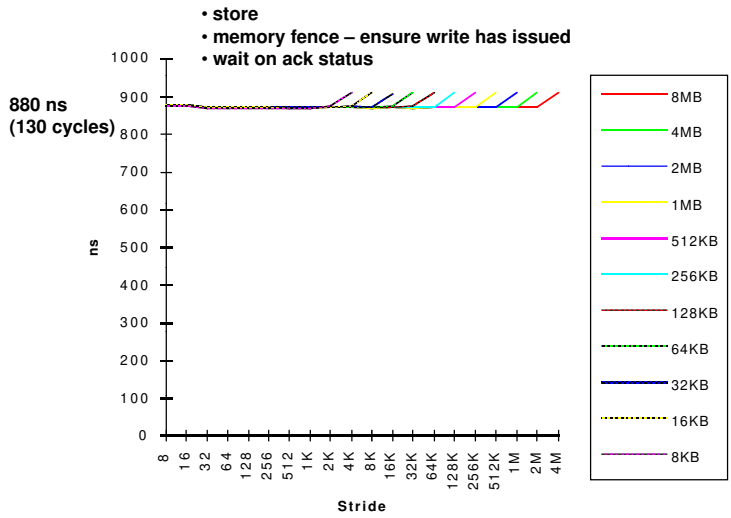
Remote Read – Cached



3 μs for Dash, 7.5 μs for KSR, 12.5 μs for CM-5, 22 μs Paragon

© Culler 1995

Blocking Remote Write



© Culler 1995

Split-C Implementation

- Read: x = *G
  - annex setup + uncached load: 785 ns (117 cycles)
  - cannot ensure that cache line would be flushed
    - » HPF might be able to use cached load
- Write: *G = x
  - blocking write in “remote invalidate” mode
  - 1050 ns (156 cycles)
- Byte writes
  - Alpha assumes read word, manipulate byte, store words
  - This may clobber surrounding bytes in MP !!!
  - Would use load-locked, modify, store-conditional, but these were usurped for Annex
  - Must enclose in explict lock (yechhhhh)
- Global / Local consistency
  - local ops allowed to bypass globals

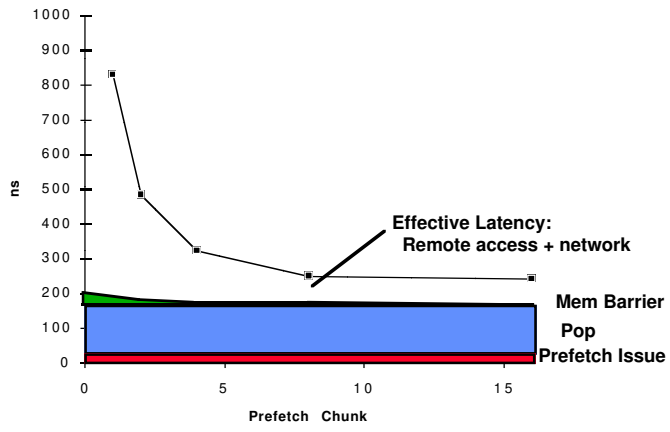
© Culler 1995

Split-Phase Access

- Split-C
  - get: x := *G; ... ; sync();
  - put: *G := x; ... ; sync();
- T3D
  - Uses fetch to provide non-blocking read into prefetch queue
    - » 16 deep;
    - » requires memory fence to ensure prefetch issues
    - » Pop entries from prefetch queue via load to special address
  - Non-blocking writes

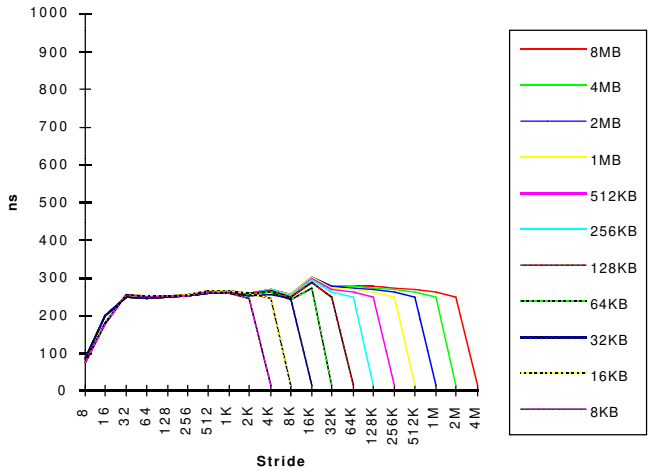
© Culler 1995

Prefetch Effective Latency



© Culler 1995

Non-blocking write



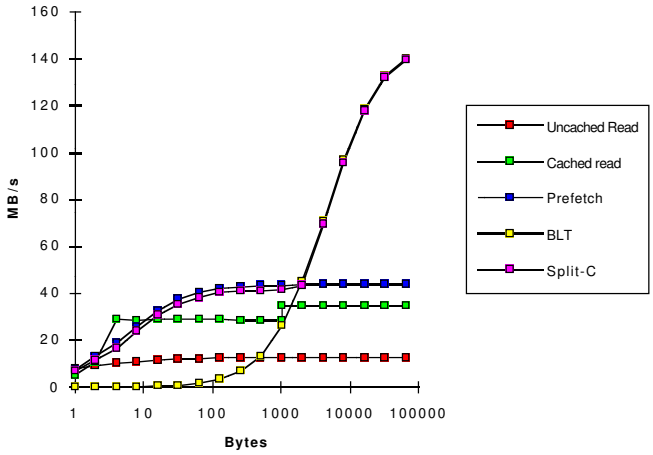
© Culler 1995

Bulk Operations

- Split-C
  - bulk_get, bulk_put
- T3D
  - block transfer engine
  - cached reads
  - prefetch
  - non-blocking writes

© Culler 1995

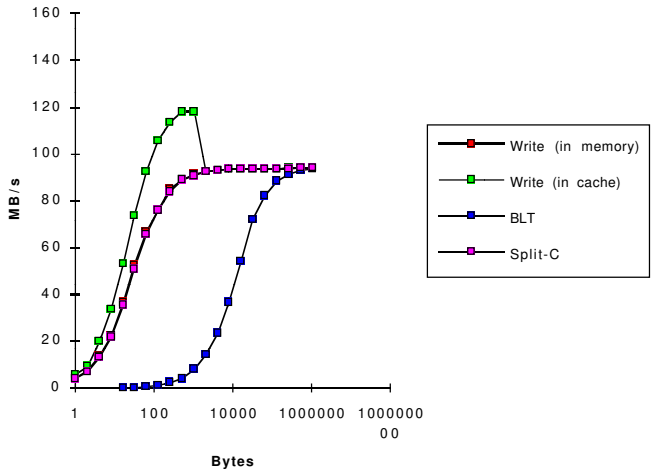
Bulk Read Trade-offs



cs258 16.27

© Culler 1995

Bulk Write Trade-offs



cs258 16.28

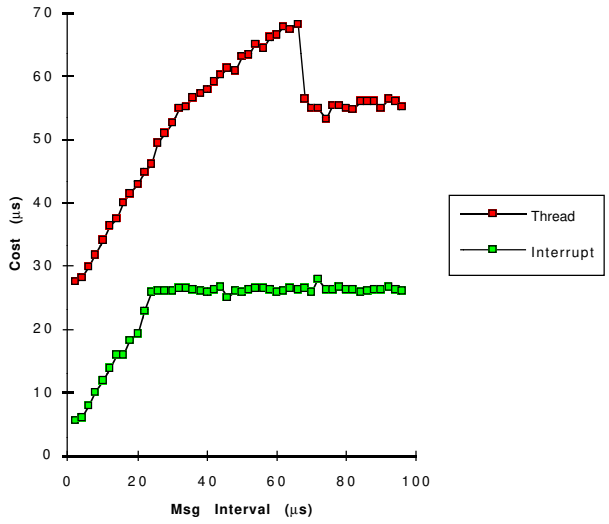
© Culler 1995

Bulk-Synchronous and Msg Driven Computation

- Split-C
  - signalling stores: *G :– x;
  - all_store_sync();
  - store_sync(N);
- T3D
  - bulk synchronous only
    - » non-blocking stores, wait net quiet, global barrier
  - msg driven or combination
    - » user-level message queue
    - » PAL call 813 ns (122 cycles)
    - » Interrupt cost: 20µs + 5µs per message received
    - » User-level Thread: 58 µs
    - » Build msg queue with fetch&inc
      - 2.9 µs send (4 words + handler), 1.5µs to pop and dispatch

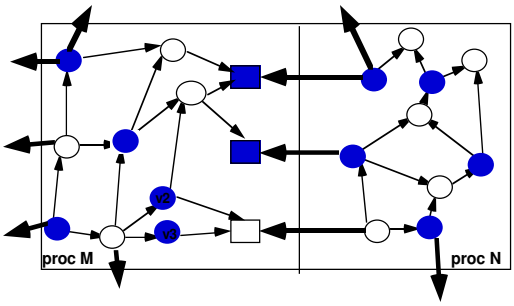
© Culler 1995

Active Message Cost



© Culler 1995

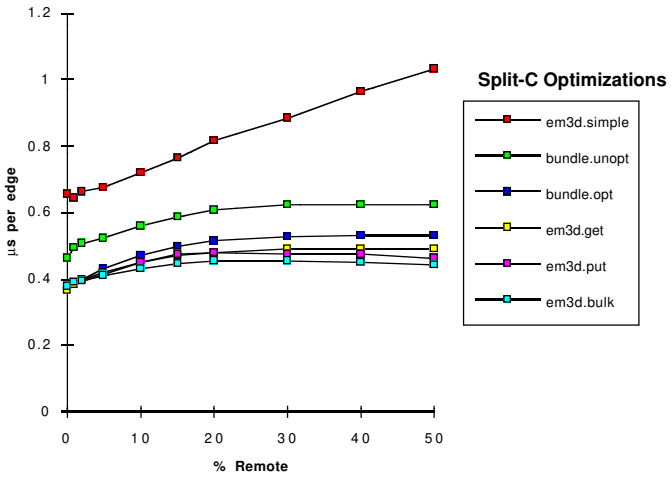
EM3D Example




```
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();
```

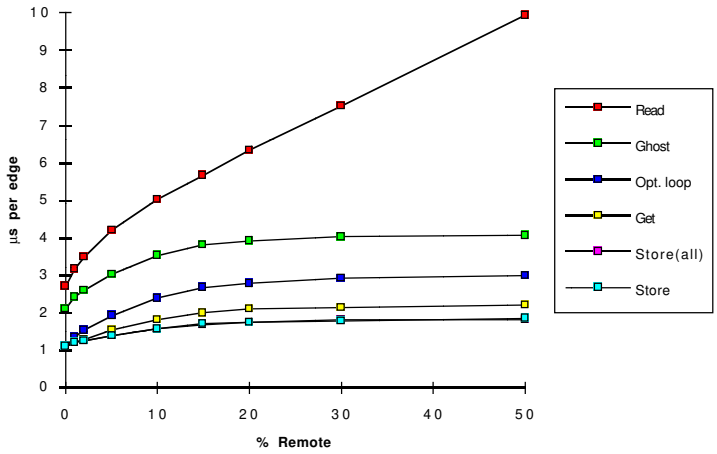

© Culler 1995

EM3D Performance on T3D



© Culler 1995

EM3D Performance on CM-5



© Culler 1995

Conclusions

- "Kitchen sink" of global communication / sync operations
  - requires careful assessment of performance trade-offs
- Raw hardware performance is very impressive
  - remote access is 3 - 4x local cache miss (still 100 cycles)
- Prefetch is very effective
  - effective latency can be as low as 250 ns per word
- Performance characteristics or subtle language/hardware interactions eliminated many potential options
  - simple Annex management strategy
  - bulk transfer almost too slow to be useful
  - message queue much slower than fetch&inc
  - prefetch + non-blocking write + fetch&inc are most valuable
- A few nasty oversights
  - partial word store

© Culler 1995