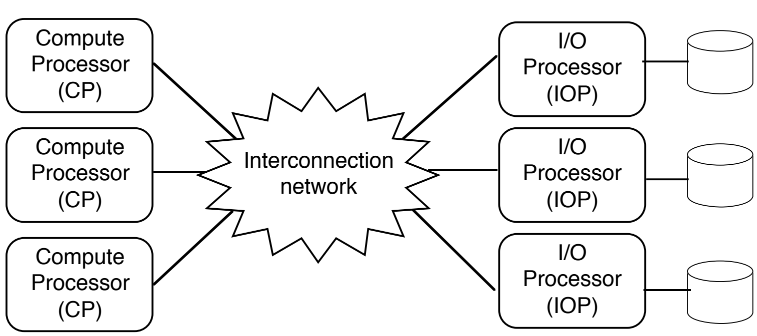
Disk-directed I/O for MIMD Multiprocessors

David Kotz

Dartmouth College

Typical MIMD Multiprocessor

- Compute processors (CP)
  - mostly application processing
- I/O processors (IOP) with disks
  - mostly file-system processing



Typical Parallel File System

- file blocks are striped across disks
- Unix-like semantics
  - open, read, write, seek, close
  - “file pointer” tracks current position
- some extensions
  - file-pointer “modes”: independent, shared, synchronized

Typical Workload

- The Dartmouth CHARISMA project
  - traced iPSC/860 at NASA Ames
  - traced CM-5 at NCSA
- Parallel scientific applications
  - large files
  - small request size: often < 200 bytes
  - sequential but not consecutive
  - complex, but regular, patterns

A Typical Program

- The situation:
  - programmer thinks: read a huge matrix
    - 2-dimensional, stored in row-major order
    - distribute the columns cyclically among CP memories
  - programmer (or compiler) writes loop for each CP:
    - seek to next element of my column
    - read one element
- The problem:
  - file system sees: many many tiny requests!
    - overhead
    - cache thrashing
    - failed prefetching
    - disk-head seeks

What's Wrong?

- the interface is limited
  - no way to express non-contiguous file access
  - no way to express a collective I/O activity
- semantic information is lost
  - lost opportunities for optimization

Outline

- (Introduction)
- Disk-directed I/O
- Experiments
- Results
- Conclusions
- Future Work

Disk-directed I/O

- Key observation:
  - disks are a slow, block device
  - disks have a preferred access order
  - memories are a byte device
  - memories are random-access
  - *Let disks determine order and pace*
- Collective, high-level request *to IOPs*
  - IOPs now have the semantic information they need
- IOPS in control
  - arrange for all I/O
  - read and write CP memory

Experiments

- we implemented both
  - traditional caching
  - disk-directed I/O
- simulated parallel architecture:

MIMD, distributed-memory	32 processors
Compute processors (CPs)	16
I/O processors (IOPs)	16
Disks	16
Disk peak transfer rate	2.34 MB/s
File-system block size	8 KB
I/O buses (one per IOP)	16
Interconnect topology	6 x 6 torus
Interconnect bandwidth	200 x 10^6 Bps, bidirectional

Traditional Caching

- CP, for each contiguous request:
  - break up big requests into single-block requests
  - requests sent concurrently to IOPs
    - at most one outstanding per disk
  - DMA between user buffer and network
- IOP, for each request:
  - check cache
    - 2 buffers per CP per disk
    - LRU, write-behind, one-block prefetch
  - send reply to CP with requested data

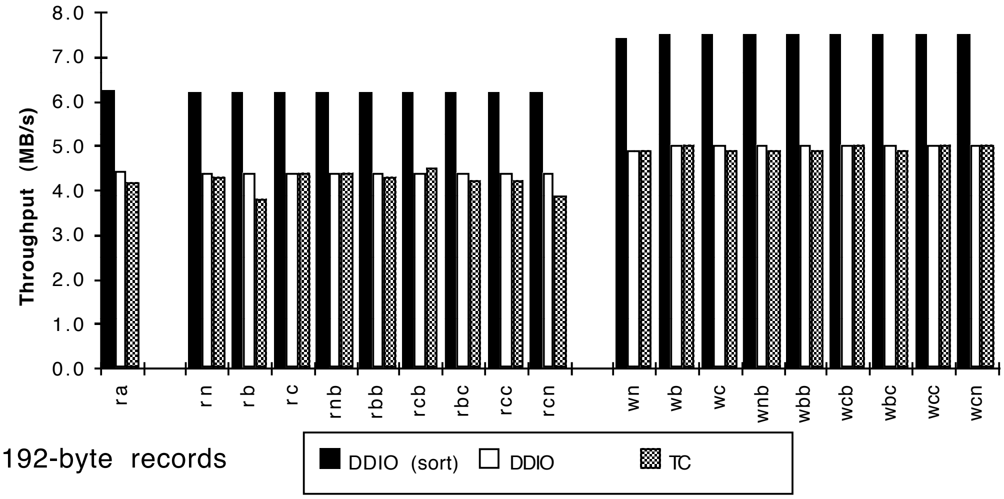
Disk-directed I/O

- CPs
  1. barrier
  2. one CP does:
     a. send request to all IOPs,
     b. wait for all IOPs to reply.
  3. barrier
- IOPs
  1. make list of blocks to move
     - it can sort list of blocks by location
  2. start two new threads:
     - allocate one-block buffer
     - repeat until done:
     - choose block from list
     - fill buffer with that block's data
     - empty buffer
  3. reply “done” to originating CP
- Special messages
  - <u>Memput</u> deposits data into user buffer
  - <u>Memget</u> replies with data from user buffer

Access patterns

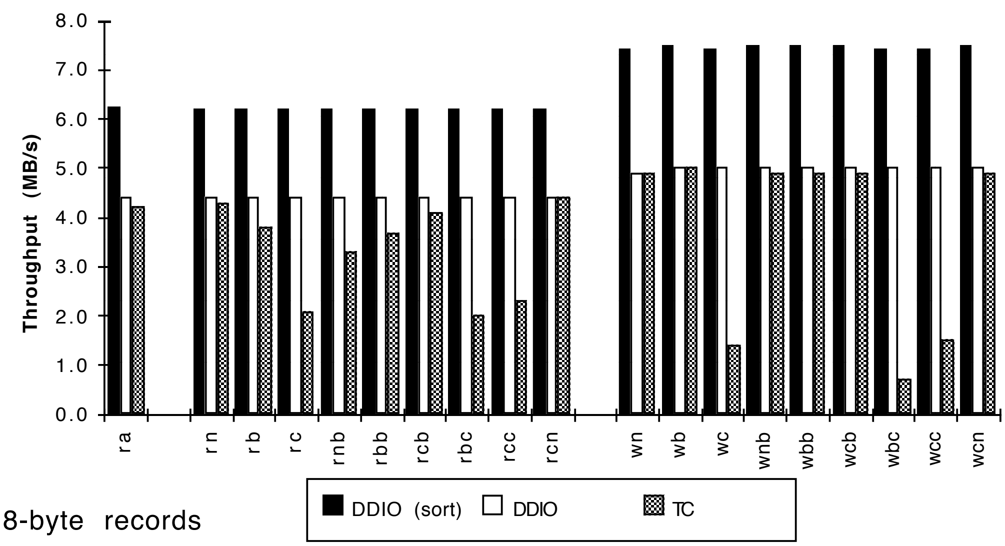
- Read and write matrices:
  - one- or two-dimensional
  - stored row-major order in file
  - distributed among CP memories in HPF patterns
  - element size 8 bytes or 8 Kbytes
- Files:
  - all 10 MB
  - striped across all 16 disks, by 8KB block
  - within each disk,
    - Random blocks
    - Contiguous
    - (real systems in between)

Results: random-blocks



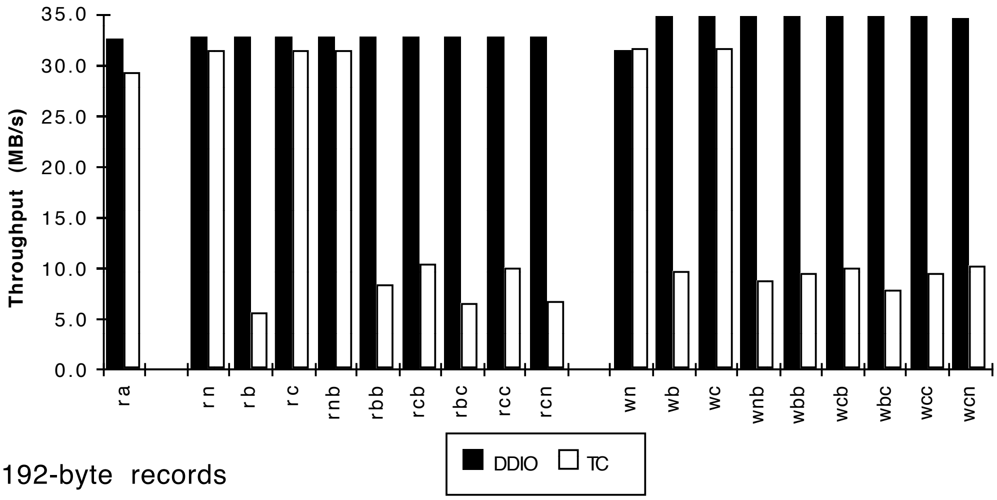
8192-byte records

Results: random-blocks



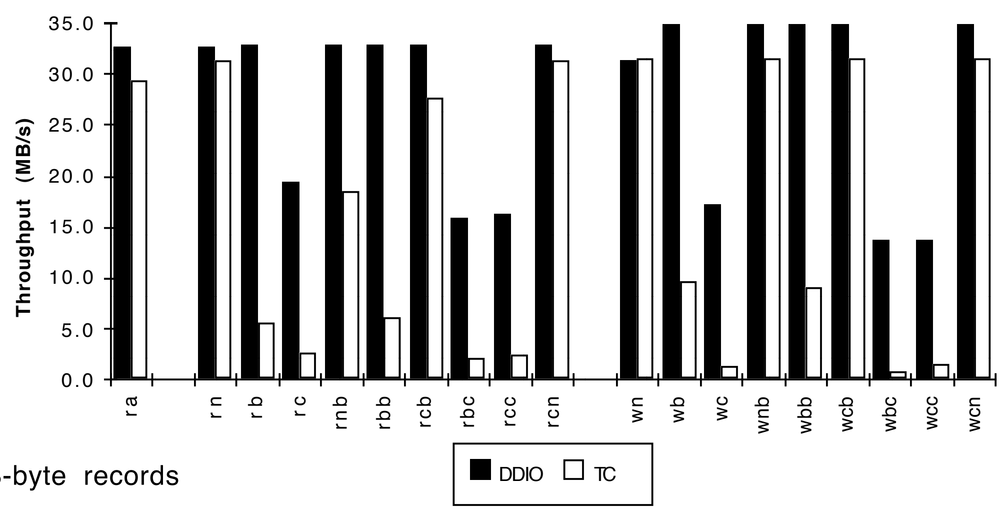
8-byte records

Results: contiguous



8192-byte records

Results: contiguous



8-byte records

Sensitivity

- Disk-directed I/O performance
  - unaffected by the number of CPs
  - scaled as it should
    - limited only by disk or bus bandwidth
- Traditional caching:
  - When fewer CPs than disks
    - some cyclic patterns could not keep all disks busy
  - Overhead a problem with more CPs
- Other record sizes: no surprises
- Bigger file sizes: no surprises

Conclusions

- Disk-directed I/O works:
  - consistent performance, independent of distribution.
  - near hardware limits, 93% of peak.
  - in one case, 18 times faster than traditional caching.
- How?
  - by reducing overhead
  - by sorting disk requests
  - by managing contiguous layouts

Conclusions

- Valuable for large, collective data transfers.
- but the *concept* is extensible:
  - irregular patterns
  - non-collective I/O
  - out-of-core algorithms
  - asynchronous I/O
  - filtering
  - uniprocessors
  - shared-memory architectures

Future Work

- “Real” application
- Gather/scatter messages
- Strided requests
- Collective-I/O interface
- Concurrent disk-directed activities

Parallel I/O on the WWW

http://www.cs.dartmouth.edu/pario.html

dfk@cs.dartmouth.edu