EDONIO: Enhanced Distributed Object Network I/O Library

E.F. D'Azevedo

C.H. Romine

http://www.epm.ornl.gov/~romine/

EDONIO

- C and Fortran callable replacements for NX I/O calls on Intel iPSC and Paragon.
- Provides disk caching for enhanced performance.
- EDONIO translates I/O calls into messages that update disk cache.
- EDONIO uses extended int’s to access files up to 16Terabytes.

Premise of EDONIO

- Total network bandwidth greatly exceeds disk I/O bandwidth.
- Aggregate memory of iPSC/Paragon used as large disk cache.
- Non-sequential concurrent access to shared files is desirable.
- Optimal PFS performance is achieved for concurrent large contiguous blocks.

EDONIO

- Unix/C binary file, NOT Fortran unformatted file
- All processors (not subgroup) participate in I/O to shared file
- Uses M_ASYNC mode for high performance

Cache

- Disk cache holds disk blocks of 64Kbytes
- Data cache holds *read-only remote* data of 8Kbytes pages
- Disk blocks *statically* assigned in block wrapped mapped fashion
- Simple Least Recently Used strategy

EDONIO Operations

- Standard I/O calls replaced as follows:
  - » open becomes do_open or DOOPEN
  - » read becomes do_read or DOREAD
  - » write becomes do_write or DOWRITE
  - » lseek becomes do_lseek or DOLSEEK
  - » flush becomes do_flush or DOFLUSH
  - » close becomes do_close or DOCLOSE
- Most I/O during preload, close and flush

do_nio

- Synchronous call to initialize package
- do_nio(int myid, int nproc)
- Also initializes IPX message subsystem
- Required before any EDONIO calls

do_open

- Returns a ``file descriptor'' for later file access (do_read, do_write, do_close)
- All processors must open file as synchronous operation.
- Optional call to synchronous “do_lsize”.
- Require read/write permission on writeonly files.

do_read

- Assuming requested data not in local data-cache (read-only files):
  - » determines location of data in network
  - » IPX sends appropriate messages to obtain data
  - » may require I/O to reload disk block
  - » received data are used to satisfy read request
  - » data pages are cached, if file is read-only

do_read

- Error to call “do_read” with write-only files
- Error to read past end-of-file
- Seek pointer updated to next byte in file
- Times may vary substantially, depending on access pattern and cache

do_preload

- Synchronous operation to read ahead and preload disk cache
- Start reading from min file pointer (beginning of file after do_open)
- Will not displace data already in disk cache. Use “do_csize” to make room or displace existing data.

do_write

- Location of requested data determined
- Messages sent to owner processors requesting updates
- Simultaneous updates of overlapping data undefined
- May require reloading of disk block before update

do_csize

- Synchronous call to expand or contract cache used
- Default 512K for data cache, 4096K for disk cache
- Tip:avoid paging with large cache size, use “vm_stat” to monitor free pages
- Hold entire file in core for best performance

do_gsync/do_check

- Polling version of IPX
- “do_check” to perform polling
- “do_gsync” before gdhigh to purge IPX messages
- Avoid message tags 0-10 and over 8Mil

do_eseek / do_lseek

- Simply resets local file pointer to indicated value
- do_eseek returns extended integer for files over 2Gigabytes
- Independent file pointers, need care on SEEK_END operations

do_flush

- Forces disk I/O, writing current image of file in memory to disk
- Writes only “dirty” disk blocks, cache is intact.
- Supplied to avoid catastrophic loss due to crash/power failure and loss of cache
- Synchronous call requiring all processors to participate

do_close

- do_flush followed by destruction of file cache
- disk I/O done on large contiguous blocks to optimize performance
- All processors must participate
- Tip: file not automatically closed on exit, need explicit call to “do_close”

Differences

- Require read/write permisson even on write-only files
- Care on SEEK_END operations with do_lseek
- Care with blocking primitives (such as gdhigh, gsync, crecv) with polling version of EDONIO
- Exact file size even with do_lsize

EDONIO Results

- Synthetic benchmark to generate element to vertex list
- 200x200x200 grid (256MBytes), 300x300x300 grid (864MBytes)
- Default 512K and 4096K for data and disk cache
- Run on xps35, affected by other disk activities

EDONIO Results

- Total Cache size increase with more processors
- Physical I/O with “wclose” and “preload”
- Preload and Close times increase with more processors
- Same volume of message spread across more processors

EDONIO vs NX (256MB)

proc	wopen	write	wclose	ropen	preload	read	rclose
16	3.1(1.3)	31.3(141.2)	1.9(0.2)	1.4(0.8)	2.2	84.5(89.5)	0.3(0.2)
32	3.1(2.1)	15.5(122.4)	3.2(0.4)	1.5(1.3)	3.6	30.1(49.3)	0.4(0.4)
64	3.0(3.5)	5.3(118.6)	7.6(0.7)	2.5(2.1)	7.9	7.2(48.0)	0.8(0.7)
128	4.7(4.7)	3.0(89.2)	11(1.5)	4.3(3.7)	7.7	4.0(47.5)	1.6(1.4)

EDONIO vs NX (864MB)

proc	wopen	write	wclose	ropen	preload	read	rclose
32	2.1(1.5)	45.9(262.0)	5.2(0.4)	2.6(2.3)	3.4	120(111)	0.4(0.3)
64	2.9(2.8)	24.1(218.1)	7.3(0.7)	2.8(2.2)	6.5	56.7(109)	0.8(0.7)
128	4.9(4.5)	14.1(360.3)	23 (1.5)	4.6(4.8)	15.8	21.4(105)	1.5(1.5)

xps5 100x100x100 (32MBytes)

proc	wopen	write	wclose	ropen	preload	read	rclose
16	0.8(0.6)	2.3(142.9)	5.3(0.1)	0.9(0.2)	6.7	2.4(77.6)	0.1(0.1)
32	0.9(0.7)	1.3(146.8)	5.5(0.2)	1.1(0.3)	6.9	1.3(76.1)	0.2(0.2)
8	0.6(0.4)	4.5(153.3)	5.3(0.1)	0.9(0.2)	25.3	12.8(87.7)	4.4(0.1)
4	0.4(1.0)	25.8(186.5)	22 (0.1)	0.5(0.2)	9.0	26.8(95.3)	0.1(0.1)
4	0.5(0.5)	16.6(164.5)	11 (3.7)	0.4(0.2)	7.7	26(100.5)	0.1(0.1)

Example

- xps5:/home/xps5/u0/efdazedo/TEST
  - » ex3.F, ex3.sh (EDONIO), ex3nx.sh (NX)
- precompiled library
  - » nipxnode.o libdo.newio.a
  - » link in SAME order
  - » should work on other Paragon systems
- cpp macro expansion with “fwrap” awk postprocessing