the ECIOPROF interposer

a proof-of-concept study for I/O-profiling on AIX

Oliver Treiber
HPC group, ECMWF

goals and approach

- **profile I/O on a per-process/task level on AIX OS**
  - by generating trace events with low performance impact
  - post-processing trace data conveniently with perl/python/...
- **should be generically applicable and easy-to-use**
  - no relinking, just set $LIBPATH/$LDR_PRELOAD envvars in tiny wrapper script
  - meaningful tracefile naming, e.g., include MPI rank and command profiled
- **(standard and generic) approach: wrapping some functions in libc.a**
  - general: also applicable to other UNICES (one would think...)
  - BUT: intercept not only calls into libc.a, but also internal calls internal within libc.a
    - example: fprintf() will eventually call write() to flush the buffer

      ➔ would like to intercept both the fprintf() and the resulting write()
    - cf., IBM’s proprietary libtkio/libmio, or GNU linker’s “–wrap” option used in NERSC’s IPM
  - many other possible applications for such intercepting
    - e.g., mirroring I/O for selected paths to “shadow filesystems”
    - though, this better be done at application-level if have source code access

and then: some sugar on top ...

- **low-overhead high-resolution timestamps**
  - simply use mftb instruction on POWER cpus via inline asm
- **walk the stack's saved link registers for instruction-stacktrace to annotate trace events**
  - can help easily identify from what function a particular event stems from
- **asynchronous I/O request completion timing**
- **some control through environment variables**
  - e.g., profile child processes, too? any, or only selected binaries?
- **... and whatever else comes to mind**
  - it is “our source”/”open source”? – so can do whatever we like
    - as opposed to vendor’s toolkit shipped as binary modules, etc...
    - e.g., maybe chase some AF_UNIX sockets, too?

teaser: let's do some AIO…

```
void aio_from_subfunc( struct aiocb64* cb) {
  aio_write64( cb ); }

int main( int argc, char** argv )
{
  […]
  aio_fd1 = open( "aio.1.out" , O_RDWR|O_CREAT, S_IWUSR|S_IRUSR );

  cb_a.aio_sigevent.sigev_notify = SIGEV_NONE; […]

  cb_a.aio_fildes = aio_fd1; cb_a.aio_buf = obuf;
  cb_a.aio_offset = 0;
  cb_a.aio_nbytes = sizeof( obuf )/3;

  cb_b.aio_fildes = aio_fd1; cb_b.aio_buf = obuf;
  cb_b.aio_offset = sizeof(obuf)/3;
  cb_b.aio_nbytes = sizeof( obuf )/4;

  cb_c.aio_fildes = aio_fd1; cb_c.aio_buf = obuf;
  cb_c.aio_offset = sizeof(obuf);
  cb_c.aio_nbytes = sizeof( obuf )/5;

  aio_from_subfunc( &cb_a );
  aio_write64( &cb_b );

  for( int i=0; i<100000000; i++ ) { /* waste time */ }

  aio_from_subfunc( &cb_c );
  […]
```

and, for a fistful of keystrokes: get a post-processed I/O profile

```
INFO: read 18 symbols from hello_aio
INFO: cooked trace data from 'ECIOPROF.hello_aio.913758.1288734339'

'/s1t_gpfs/s1t_home/filesets/s1t_home_systems/syg/work/iowrap/aio.1.out': 3285537 bytes
  'MaxAIOWait_ms' => 56,
  'OpenDuration_us' => '224',
  'AIOWriteBytes' => 3285537,
  'MinimumAIOWait_ms' => 20,
  'AIOOpsChronological' => [
    '2ms:aiow:0x100000474=.aio_from_subfunc()+0x14:1398101 B at offset 0, request 110400af0',
    '2ms:aiow:0x1000006cc=.main()+0x18c:1048576 B at offset 1398101, request 110400b70',
    '38ms:aio_done:aiowreq 110400af0, 36ms after issue',
    '58ms:aio_done:aiowreq 110400b70, 56ms after issue',
    '729ms:aiow:0x100000474=.aio_from_subfunc()+0x14:838860 B at offset 4194304, req 110400bf0',
    '749ms:aio_done:aiowreq 110400bf0, 20ms after issue' ]
  'AIOWriteRequests' => 3,
  'AIOWriteMainFunction' => {
    '0x100000474 = .aio_from_subfunc()+0x14' => 2,
    '0x1000006cc = .main()+0x18c' => 1 },
  'CloseDuration_us' => '20148',
  'OpenMainFunction' => { '0x10000062c = .main()+0xec' => 1},
  'MaxAIOQueueDepth' => 2,
  'OpenTimeInProcess_ms' => 0,
  'OpenTime' => '0.000:1288734339.119:Tue Nov  2 21:45:39 2010',
  'OpenFor_ms' => 729,
  'CloseTime' => '0.729:1288734339.848:Tue Nov  2 21:45:39 2010',
  'BytesWritten' => 3285537,
  'MaxAIOWriteSize' => 1398101,
  'MinAIOWriteSize' => 838860
```

side remarks: truss as poor man’s I/O profile

- **on AIX, intercept syscalls with truss (="strace")**
  - use something like
    truss –o <tracefile> –f –t open,close,kwrite,kread,... <binary>“
    in wrapper script
    - if using shell wrappers with poe, make sure to use ksh93!
      (look for ksh93 in PE manuals... encounter puzzling problems otherwise)
  - gets you started quick and easy, but **noticeable performance impact**
  - not so great for selecting which files/paths to profile
  - not so great for profiling libc buffered streams, e.g., fprintf(), ...
  - no real support for aio
- **but: gets you started within less than a minute ;-)**

ECIOPROF: status and DISCLAIMER

- **currently, this is merely proof-of-concept exploration**
  - so far: simple, “hobby” interest/private background noise activity
  - so far: only drafty implementation, code not nicely refactored/documented
  - very few lines of code so far with some inessential limitations
    - one wants to be aware of these before using
    - but no principal limitations – could easily be fixed by more robust implementation (e.g., currently fixed array used to track process’ file descriptors)
  - e.g., more compact trace format easy to implement (but tedious)
- **but, it appears to work quite nicely...**
  - have already uncovered “suboptimalities” in ECMWF production codes like operational model or 4d-VAR
    - unnecessary file I/O, setvbuf omission or bugs in tuning streams to GPFS blocksizes
    - how long does the operational model's asynchronous field database I/O take?
  - or: e.g., profile frequently called perl scripts using many imports
    - how much time sourcing modules until we actually start "real work" in the process?

ECIOPROF implementation: a few technical teasers

- **build an alternative profiling libc.a instrumented with wrappers**
    - wrappers write events into buffered stream using fprintf()
  - stitch into this new libc.a wrappers for descriptors and definitions for *read*(), *fread*() etc after renaming originals using **–brename AIX linker** gymnastics
    - this is key to break up and wrap libc-internal calls to write() “from” streams, e.g.!
  - for calling into original “pass-through” symbols like open(), close(), … exported from kernel, in their wrappers find references through **dlsym**() on a handle obtained from **dlopen**(“/usr/lib/libc.a”)
    - system’s libc.a is thus mapped as well
  - walk the stack quickly with **inline assembler** for cheap stacktraces
    - cf. POWER ABI subroutine linkage conventions
  - can internally make use of such stacktrace info for some “**hacking**”
    - e.g., for tracing “nasty varg” fprintf(), profile the “backend” fixed signature _doprnt() service instead– but do not profile _doprnt when it has been called from sprintf()
  - add **hidden SIGEV_SIGNAL** to AIO control blocks and register bespoke signal handler to capture timing info for "aioserver kproc done"
    - in using this, need be aware re interruptible system calls

a slightly more comprehensive “hello_world” example

- “simple” source with
  - aio_write64()
  - “Posix” I/O: open(), write(), ...
  - libc buffered stream I/O: fopen(), fread(), fprintf(), fgets(), ...
  - also: fork to a Fortran binary
    - to demo it works with Fortran runtime
    - to demo it follows kids

source of “hello_c” demo, part 1

```
[include some standard C header files...]
char obuf[4096*1024], ibuf[4096], stream_buffer[4096*256];
int aio_fd, posix_io_fd; FILE* buffered_stream;

void write_posix_io( int fd ) {
  write( fd, obuf, sizeof( obuf ) ); }

void my_fwrite( FILE* stream ) {
  fwrite( obuf, sizeof( obuf ), 1, stream ); }

int main() {
  /* initialise output buffer */
  memset( obuf, 'x', sizeof(obuf)); obuf[sizeof(obuf)-1]= 0;

  /* do some async i/o */
  struct aiocb64 cb; const struct aiocb64 *aio_req_list[1] = { &cb };
  memset( &cb, 0, sizeof(cb) );
  aio_fd = open( "aio.out" , O_RDWR|O_CREAT, S_IWUSR|S_IRUSR );
  cb.aio_fildes = aio_fd;
  cb.aio_buf = obuf;
  cb.aio_nbytes = sizeof( obuf );
  cb.aio_sigevent.sigev_notify = SIGEV_NONE;
  aio_write64( &cb );

  /* do some posix io */
  posix_io_fd = open( "posix_io.out" , O_RDWR|O_CREAT, S_IWUSR|S_IRUSR );
  write( posix_io_fd, obuf, sizeof(obuf) );
  write_posix_io( posix_io_fd );
  close( posix_io_fd );
 [...]
```

source of “hello_c” demo, part 2

```
[...]
   /* do some buffered io */
   buffered_stream = fopen( "buffered_stream.out", "r+" );
   setvbuf( buffered_stream, stream_buffer, _IOFBF, sizeof(stream_buffer) );
   fprintf( buffered_stream, "%s", "ciao, mundo!\n" );
   fputs( "hello fputs...!\n", buffered_stream );
   my_fwrite( buffered_stream );
   fseek( buffered_stream, 0, 0 );
   fgets( ibuf, 1024, buffered_stream );
   fprintf( stderr, "read string: %s\n", ibuf );
   fread( ibuf, 32, 1, buffered_stream );
   ibuf[32] = 0;
   fprintf( stderr, "read from stream: %s\n", ibuf );
   fclose( buffered_stream );

   /* more posix io, second round on same path */
   posix_io_fd = open( "posix_io.out" , O_RDWR|O_CREAT, S_IWUSR|S_IRUSR );
   write( posix_io_fd, obuf, sizeof(obuf) );
   write_posix_io( posix_io_fd );
   close( posix_io_fd );

   /* fork a fortran "hello world" */
   if ( ! fork() ) { execl( "hello_fortran", "hello_fortran", 0 ); }
   else { wait( 0 ); }

  /* wait for aio to finish */
  aio_suspend64( aio_req_list, 1, 0);
  close( aio_fd );

  exit( 0 ); }
```

1MB buffer

fwrite() of 4MB buffer

source of “hello_fortran” demo

```fortran
program hello_fortran

call ciaomundo()

open(unit=10,file='fortran.output' )
write (10,*) "fortranout"
close(10)
end

subroutine ciaomundo()
  print *, "ciao, mundo"
  open(unit=11,file='fortran.output.ciao_mundo' )
  write (11,*) "howdy"
  close(11)
  return
end

```

ease-of-use: perform the actual profiling of “hello_c” (and its child “hello_fortran”)…

```
# invoke binary with ECIOPROF wrapper, asking also that lowlevel stacktraces
be produced for paths matching “buffered” or “fortran”

#> export ECIOPROF_LLTRACE_PATHS="buffered|fortran"
#> ECIOPROF.64 hello_c
-> /home/systems/syg/bin/ECIOPROF.64 traces in /home/systems/syg/tests/iowrap
[... output ...]

# check real output files have actually been generated ;-)
#> ls *out*
aio.out                     fortran.output              posix_io.out
buffered_stream.out         fortran.output.ciao_mundo

# list tracefile generated
#> ls -ltr ECIOPROF.hello*
-rw-r--r--    1 syg      systems         523 Oct 27 13:43
        ECIOPROF.hello_fortran.540710.1288186990
-rw-r--r--    1 syg      systems        1329 Oct 27 13:43
        ECIOPROF.hello_c.472036.1288186990
```

contents of “raw” tracefile for the “hello_fortran” process

```
#> cat ECIOPROF.hello_fortran.217532.1288187550

#ECIOPROF!compiled: Jul 22 2010 17:17:08
#timebase!1288187550.503187!Wed Oct 27 13:52:30 2010
#cwd!/s1b_gpfs/s1b_home_systems/syg/tests/iowrap
#ancestry!217532-hello_fortran:176586-hello_c:
36028797018962!62!13!w!1!13!100000660
2!22!6!o!/s1b_gpfs/s1b_home_systems/syg/tests/iowrap/fortran.output.ciao_mundo!67108866!438!1000006d0
#trcbk_open!/s1b_gpfs/s1b_home_systems/syg/tests/iowrap/fortran.output.ciao_mundo!:
  0x900000000a38f40:0x900000000d40ef4:0x900000000d4055c:0x900000000d44e24:0x900000000d7bd60:
    0x1000006d0:0x10000049c:0x100000320
2!21799!7!w!6!7!100000740
24!169!0!c!6!100000740
24!20!6!o!/s1b_gpfs/s1b_home_systems/syg/tests/iowrap/fortran.output!67108866!438!100000508
#trcbk_open!/s1b_gpfs/s1b_home_systems/syg/tests/iowrap/fortran.output!:
  0x900000000a38f40:0x900000000d40ef4:0x900000000d4055c:0x900000000d44e24:0x900000000d7bd60:
    0x100000508:0x100000320
24!17116!12!w!6!12!100000578
41!110!0!c!6!100000578
```

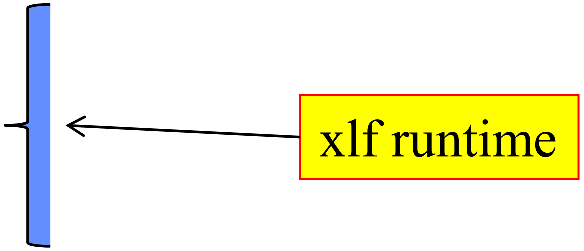
“environment”prolog

full link-register low-level traceback through libc.a, Fortran runtime and executable

writing 12 bytes to fd 6 from `0x100000578`
in executable, 24ms into execution, call took 17116us

just FYI: the stacktrace from previous slide in dbx...

```
#> dbx -E  LIBPATH=/home/systems/syg/tests/iowrap.64
     -E LDR_PRELOAD="/home/systems/syg/tests/iowrap.64/libc.a(shr_64.o):\
         /home/systems/syg/tests/iowrap.64/libc.a(posix_aio_64.o)" hello_fortran
#> stopi in open
[...]
(dbx) t
iowrappers.open(path = "fortran.output.ciao_mundo", flags = 67108866, mode = 438),
   line 419 in "iowrappers.c"
open64.open64(0xffffffffffffafa0, 0x200000002, 0x1b6, 0xffffffffffffaea0, 0x2, 0x1, 0x0, 0x1100278f0)
   at 0x900000000a38f40
TryOpen() at 0x900000000d40ef4
DoOpen() at 0x900000000d4055c
OpenCmd() at 0x900000000d44e24
_xlfIOCmd() at 0x900000000d7bd60
ciaomundo(), line 12 in "hello.f"
hello_fortran(), line 3 in "hello.f"
```



xlf runtime

cooked: buffered stream events, part1: some summary info (excerpt)

```
'/s1t_gpfs/[...]/buffered_stream.out': 5242906 bytes (5.000MB) total
  'OpenDuration_us' => '36',
  'ReadDuration_us' => 2596,
  'WriteDuration_us' => 4717,
  'FwriteDuration_us' => 5380,
  'FreadDuration_us' => 3751,
  'CloseDuration_us' => '12',

  'MaximumFwriteSize' => 4194304,
  'MaximumReadSize' => 1048576,
  'MaximumWriteSize' => 1048576,
  'MaximumFreadSize' => 32,

  'MinimumWriteSize' => 26,
  'MinimumFreadSize' => 13,
  'MinimumFwriteSize' => 13,
  'MinimumReadSize' => 1048576,
```

```
'NumberWrites' => 5,
'NumberSeeks' => 1,
'NumberFSeeks' => 1,
'NumberFputs' => 1,
'NumberReads' => 1,
'NumberFgets' => 1,
'NumberFprintf' => 1,
'NumberFwrites' => 3,
'NumberFreads' => 2

'BytesRead' => 1048576,
'BytesFread' => 45,
'BytesFwritten' => 4194330,
'BytesWritten' => 4194330,
```

cooked: buffered stream events, part1: some detail info (excerpt)

```
'/s1t_gpfs/[...]/buffered_stream.out': 5242906 bytes (5.000MB) total

 'BufferedOpsChronological' => [
    '16ms:fprintf:13 B written',
    '16ms:fputs:16 B written',
    '16ms:fwrite:4194304 B written',
    '21ms:fseek:fseek(.,0,0)',
    '23ms:fgets:13 B read',
    '27ms:fread:32 B read' ],

 'PosixOpsChronological' => [
    '16ms:w:0x100000500 = .my_fwrite()+0x20:1048576 B written',
    '18ms:w:0x100000500 = .my_fwrite()+0x20:1048576 B written',
    '19ms:w:0x100000500 = .my_fwrite()+0x20:1048576 B written',
    '20ms:w:0x100000500 = .my_fwrite()+0x20:1048576 B written',
    '21ms:w:0x100000710 = .main()+0x170:26 B written',
    '23ms:s:0x100000710 = .main()+0x170:seek(.,0,0)',
    '24ms:r:0x100000728 = .main()+0x188:1048576 B read' ],
```

“this” buffered I/O causes
“that” Posix I/O
[recall: setvbuf(1MB)]

questions...

?