



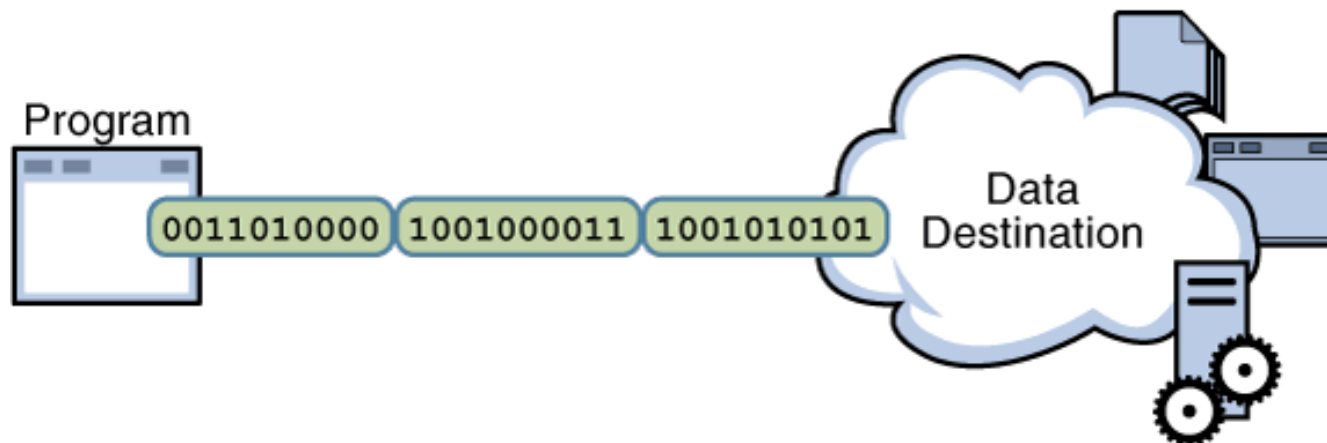
OMNIO: A Tool for I/O Recording, Analysis and Replay

Bryan Flynt
Cooperative Institute for Research in the Atmosphere
Colorado State University
Fort Collins, Colorado USA

Mark Govett
Advanced Technology and Outreach Branch
NOAA/ESRL/GSD
Boulder, Colorado USA

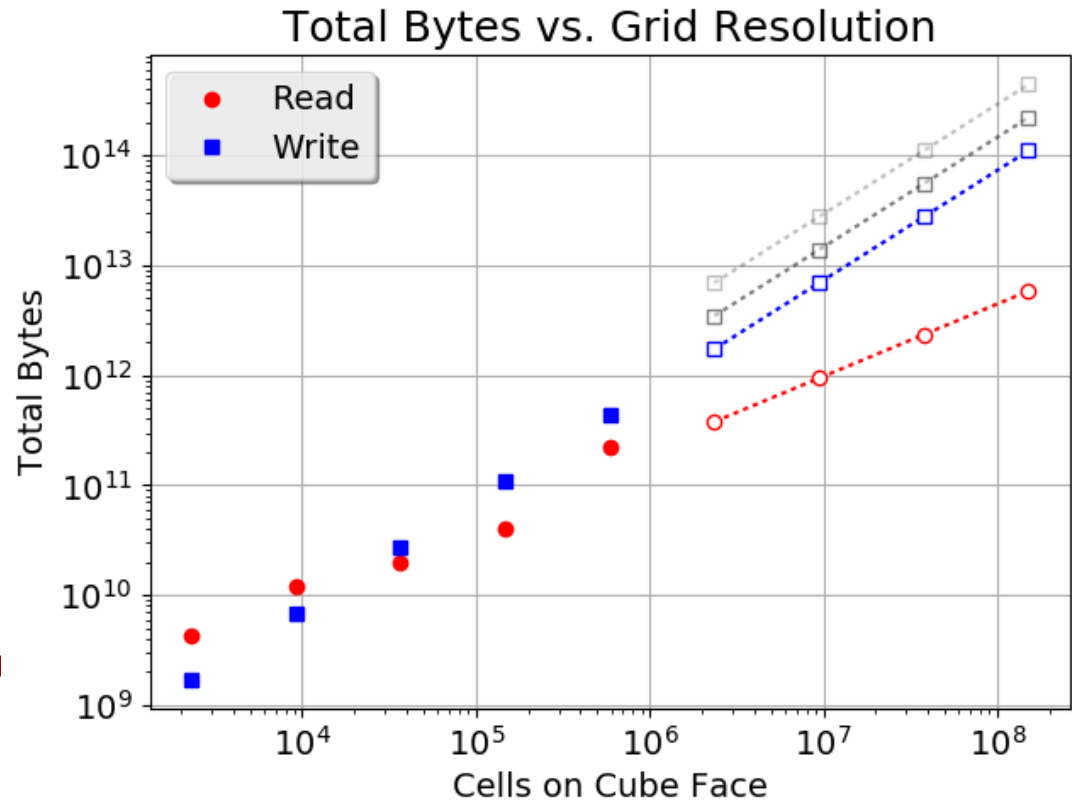
Introduction

- Motivation
- OMNIO Tool
 - Trace
 - Statistics
 - Preprocess
 - Replay
- Current Status
 - Architecture
 - Trace + Statistics
- Future
 - Preprocess + Replay
 - Development



Motivation

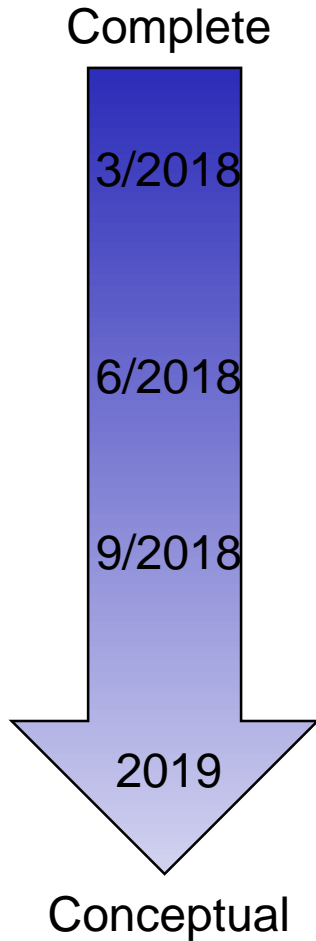
- Performance of storage system not keeping pace with compute
- File I/O can be a major portion of total run time
- As we move to higher resolution the problem only grows
- **Difficult to identify bottlenecks without knowing the application**
- **Difficult to benchmark I/O pattern without porting codes to other systems**



Purpose & Requirements

- Purpose
 - A self contained application to capture the I/O patterns of applications, examine them and replicate the pattern on a compute system of interest
- Requirements
 - Capture all POSIX level I/O calls and have capability to be extended in the future (MPI, NetCDF, etc.)
 - Record and replay calls within parallel applications using OpenMP and MPI
 - Present results in a human readable format to facilitate learning and modification of patterns for replay
- OMNIO Is Not
 - A library to facilitate I/O operations (i.e. ADIOS, PIO, etc.)

Current Overview (4 Tools)



- Trace
 - Trace an applications I/O pattern and record in log file
- Statistics
 - Create reports and charts from log files
 - Allow exploration using GUI
- Preprocessor
 - Process log files into instructions for replay and position data to read
- Replay
 - Replay a previously recorded I/O pattern

Trace Implementation

- Implemented in C++11
 - Leverage STL
 - chrono timers
 - map file descriptors
 - etc.
- Compile into four *.so files
 - Serial
 - OpenMP
 - MPI
 - MPI + OpenMP

Define a function pointer type and declare one variable

```
using open_ptr_type = int(*)(const char* path, int flags, ...);  
static open_ptr_type open; //< Inside struct Posix
```

Look up the “actual” function pointer from the symbol table at startup

```
open_ptr_type Posix::open = (open_ptr_type)dlsym(RTLD_NEXT, "open");
```

Code Example

```
ssize_t read(int fd, void *buf, size_t count){
    Logger& logger = Logger::instance();
    FileMap& fmap = FileMap::instance();

    std::string path = fmap.getFilepath(fd);

    ssize_t result;
    if( logger.not_active(path) ){
        result = Posix::read(fd,buf,count);
    }
    else {
        Event ev("read");
        ev.setStartTime();
        result = Posix::read(fd,buf,count);
        ev.setStopTime();
        ev.insert("path",path);
        ev.insert("_fd",fd);
        ev.insert("count",count);
        ev.insert("result",result);
        logger.log(ev);
    }
    return result;
}
```

Same interface as intercepted call

Maps file descriptor to file name

Shortcut to "actual" call if not logging file

Time call

Record all arguments

Buffered Output Log

Trace Implementation (cont.)

- Currently Records (24 calls)

<code>open</code>	<code>read</code>	<code>lseek</code>
<code>open64</code>	<code>pread</code>	<code>lseek64</code>
<code>creat</code>	<code>pread64</code>	<code>fsync</code>
<code>creat64</code>	<code>readv</code>	<code>fdatasync</code>
<code>mkstemp</code>	<code>write</code>	<code>close</code>
<code>mkostemp</code>	<code>pwrite</code>	<code>MPI_Init</code>
<code>mkstemp</code>	<code>pwrite64</code>	<code>MPI_Init_thread</code>
<code>mkostemp</code>	<code>writew</code>	<code>MPI_Finalize</code>

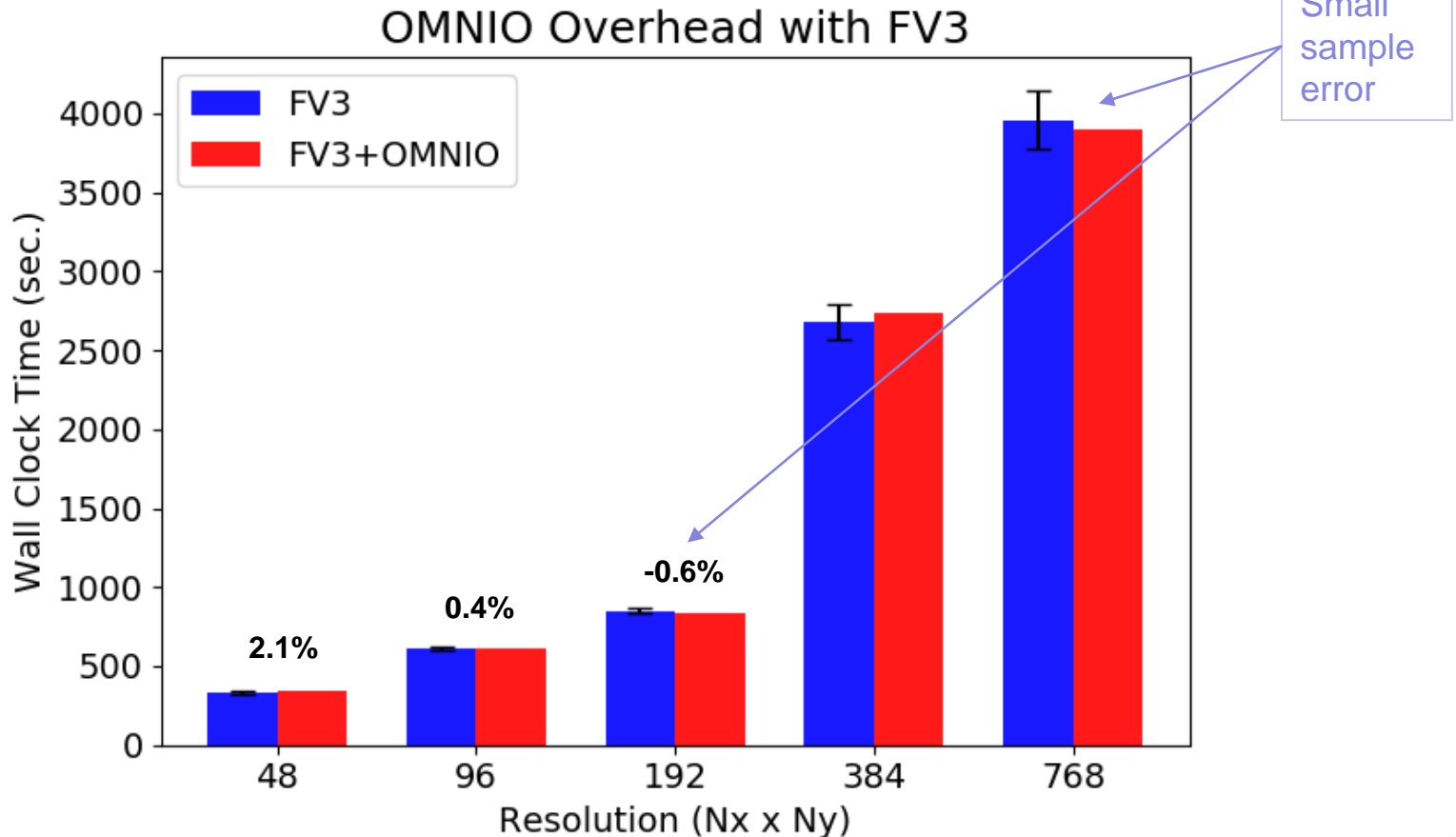
- Pre-Loaded* into Symbol Table

<code>trace::open(const char* path, int flags, ...)</code>
<code>trace::read(int fid, void* buf, size_t count)</code>
<code>trace::close(int fid)</code>
<code>posix::open(const char* path, int flags, ...)</code>
<code>posix::read(int fid, void* buf, size_t count)</code>
<code>posix::close(int fid)</code>

* LD_PRELOAD on most Linux systems
DYLD_INSERT_LIBRARIES on MacOS systems

Minimal Overhead

- Overhead less than run-to-run variability



Log Files

- Generates one log file per thread per rank
 - Minimal information to keep file size down
 - Exclude records using wildcard patterns (/path/to/not/record/*)

```
5423817573909 5423875777154 MPI_Init result=0
5423877029892 5423877095612 open fd=20,flags=16777729,mode=438,path=dummy.0.0.fmt
5423877156219 5423877168729 write count=4008,fd=20,path=dummy.0.0.fmt,result=4008
5423877217106 5423877276436 close fd=20,path=dummy.0.0.fmt,result=0
5423877976538 5423877981412 open fd=20,flags=16777216,path=dummy.0.0.fmt
5423878007071 5423878016098 read count=8192,fd=20,path=dummy.0.0.fmt,result=4008
5423878036707 5423878041087 close fd=20,path=dummy.0.0.fmt,result=0
5423878058396 5423882881788 MPI_Finalize result=0
```

Statistics Tool

- Converts log files into:
 - Reports
 - Python for portability
 - Text output
 - Charts
 - Matplotlib library
 - Visual represent report data
 - Graphical User Interface
 - Chrome web browser tracing

FileName: INPUT/gfs_data.tile6.nc

--- Total Operations ---

Open = 2
Close = 2
Read = 342
Write = 0
Seek = 336
Sync = 0

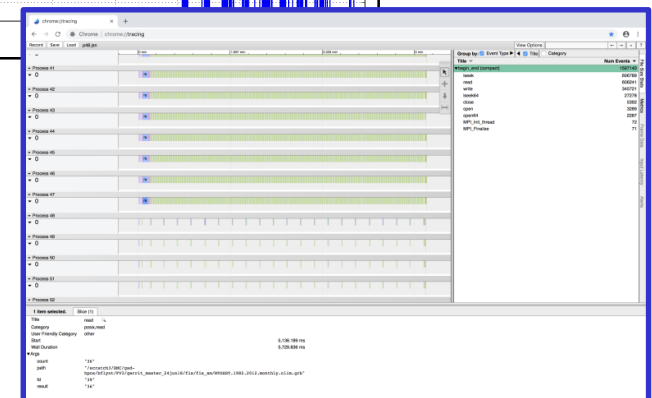
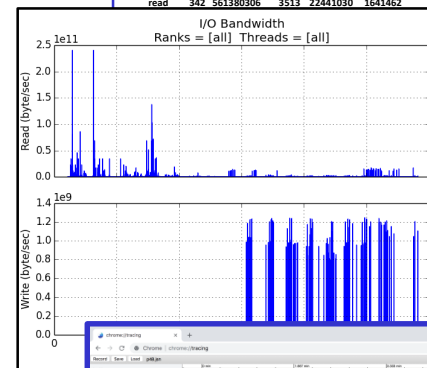
Total = 682

--- Size (Bytes) ---

Operation	#	Total	Minimum	Maximum	Average
read	342	18802438	8	65536	54977

--- Time (milliseconds) ---

Operation	#	Total	Minimum	Maximum	Average
open	2	24882	12228	12654	12441
close	2	35449	3848	31601	17724
read	342	561380306	3513	22441030	1641462



Statistics Tool (Reports)

- Statistics

FileName: INPUT/gfs_data.tile6.nc

--- Total Operations ---

Open = 2
Close = 2
Read = 342
Write = 0
Seek = 336
Sync = 0

Total = 682

--- Size (Bytes) ---

Operation # Total Minimum Maximum Average

read 342 18802438 8 65536 54977

--- Time (microseconds) ---

Operation # Total Minimum Maximum Average

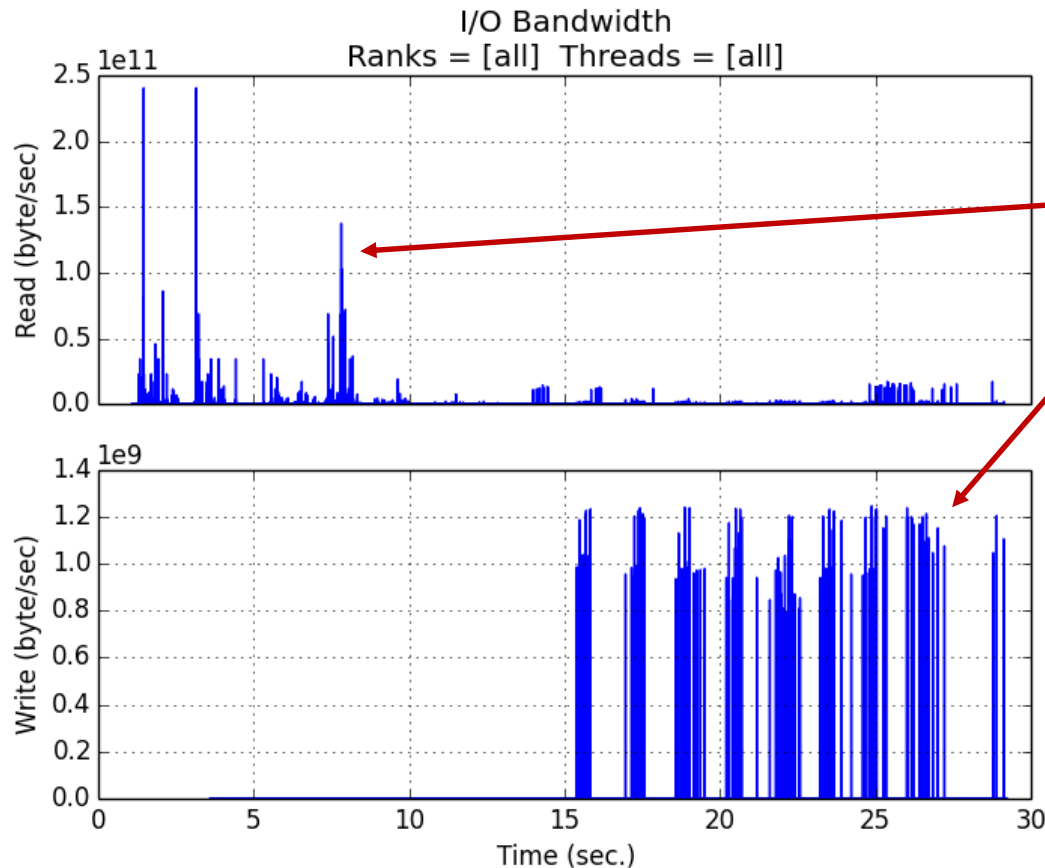
open 2 24882 12228 12654 12441
close 2 35449 3848 31601 17724
read 342 561380306 3513 22441030 1641462
seek 336 206476 373 3949 614

- Statistics Include

- Granularity
 - File
 - Rank
 - Thread
- Operations
 - Count
 - Size
 - Bandwidth
 - Time
- Statistics
 - Minimum
 - Maximum
 - Average

Statistics Tool (Charts)

- Visually represent operations

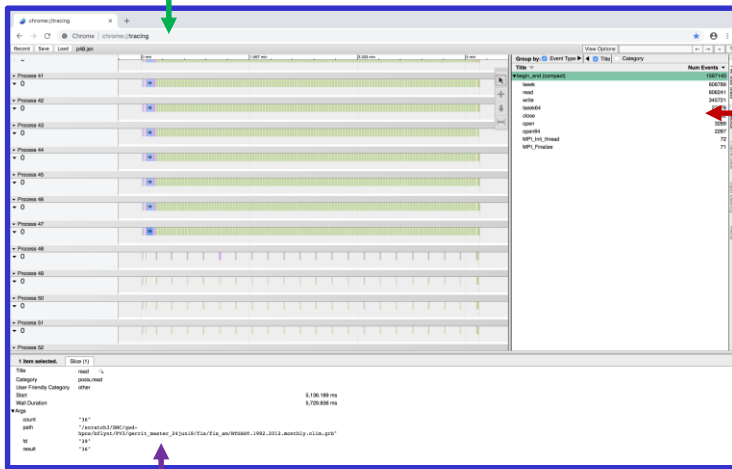
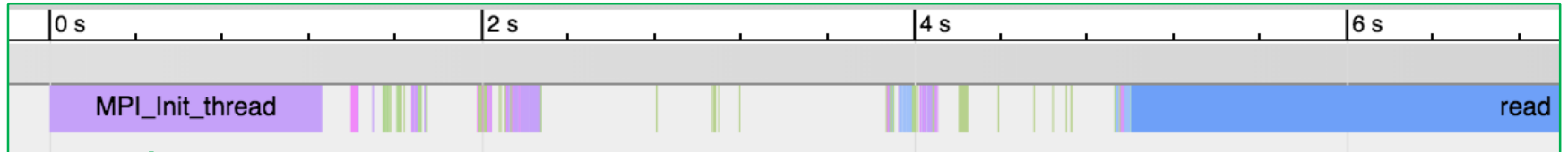


Instantaneous
bandwidth at
given time

- Example: FV3.v0 C96 Test Case
- 24 Nodes – 12 PPN
 - Aggregate Bandwidth
 - All Ranks + All Threads

Statistics Tool (GUI)

- Chrome Web Browser



View Options		←	→	»	?
Group by:	<input type="radio"/> Event Type	<input checked="" type="checkbox"/> Title	<input type="checkbox"/> Category		
Title	Num Events	File Size Stats			
lseek	606789				
read	606241				
write	345721				
lseek64	27278				
close	5392				
open	3289				
open64	2287				
MPI_Init_thread	72	Metrics			
MPI_Finalize	71				

Title	read
Category	posix,read
Start	5,000.718 ms
Wall Duration	3,648.749 ms
▶Args	
count	"42"
path	"/scratch3/BMC/gsd-hpcs/bflynt/FV3/gerrit_master_24jun18/fix/fix_am/RTGSST.1982.2012.monthly.clim.grb"
fd	"32"
result	"42"

Preprocessor

- Implementation
 - Python translator
 - Command line arguments

Converts trace
format to JSON file
of commands

```
5423817573909 5423875777154 MPI_Init result=0
5423877029892 5423877095612 open fd=20,flags=1677729,mode=438,path=dummy.0.0.fmt
5423877156219 5423877168729 write count=4008,fd=20,path=dummy.0.0.fmt,result=4008
5423877217106 5423877276436 close fd=20,path=dummy.0.0.fmt,result=0
5423877976538 5423877981412 open fd=20,flags=16777216,path=dummy.0.0.fmt
5423878007071 5423878016098 read count=8192,fd=20,path=dummy.0.0.fmt,result=4008
5423878036707 5423878041087 close fd=20,path=dummy.0.0.fmt,result=0
5423878058396 5423882881788 MPI_Finalize result=0
```

```
{
  "global": {
    "time_replay": "dt",
    "write_overwrite": "False"
  },
  "0": {
    "0": {
      "events": [
        {"start": 1023,
         "end": 2145,
         "command": "read",
         "count": 8192,
         "location": "/path/to/file.nc"},
        {"start": 2203,
         "end": 2318,
         "command": "read",
         "count": 8192,
         "location": "/path/to/file.nc"}]
    }
  }
}
```

Preprocessor (cont.)

- Considerations
 - Reading
 - Use existing file
 - Same location
 - Different location
 - Create dummy data file
 - Writing
 - Same location (overwrite?)
 - Different location
 - Coordinate locations
 - Write then read again
- Filtering by
 - File size
 - Locations
- Timing
 - ASAP
 - Delta time (Δt)
 - Same start time
- Scaling
 - Increase/decrease operation size for benchmarks

Replay

- Implementation
 - ~~Python implementation~~ (prototype)
 - Wrapped POSIX calls to insure the exact call is used
 - open vs. open64
 - lseek vs. lseek64
- Options
 - mpi4py - Requires installation beyond basic Python
 - C++ - More complex implementation code

Development

- GitHub Repository
 - Currently private
 - Collaborators welcome
- Managing through
 - Feature branches
 - Pull requests

The screenshot displays the GitHub interface for the repository `NOAA-GSD / Exascale-IO`. At the top, navigation links include `Pull requests`, `Issues`, `Marketplace`, and `Explore`. The repository is marked as `Private` and has `1` watch, `0` stars, and `0` forks. Below the repository name, there are tabs for `Code`, `Issues`, `Pull requests`, `Projects`, `Wiki`, `Insights`, and `Settings`. The repository name `Exascale IO` is followed by an `Edit` button. A progress bar shows `54` commits, `3` branches, `0` releases, and `2` contributors. A `Branch: develop` dropdown and a `New pull request` button are visible. Below this, a table lists recent commits:

Commit	Message	Time
<code>Bryan Flynt and Bryan Flynt</code>	Corrected bug in tools	Latest commit e614826 5 days ago
<code>doc</code>	Added folders and .gitignore files	7 months ago
<code>omnio</code>	Corrected bug in tools	5 days ago
<code>test</code>	Added test/replay and CMakeFile	5 months ago
<code>.gitignore</code>	Made some corrections for gnu compilers to cmake files	7 months ago
<code>CMakeLists.txt</code>	Debugged omnio2json.py and working	5 months ago
<code>LICENSE</code>	Create README and LICENSE	8 months ago
<code>README.md</code>	Transfer from BitBucket	8 months ago

Below the commit history, the `README.md` file is shown. The title is `OMNIO`. The text reads: `OMNIO is the all knowing file I/O reporting tool. It is a stand alone library which is loaded on top of the symbol tables before a program is executed. This way the first call to a read or write function from the executable will be intercepted by OMNIO`

Conclusions

- Motivation

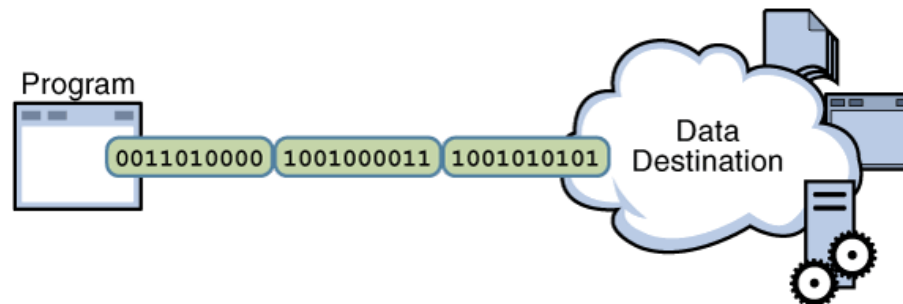
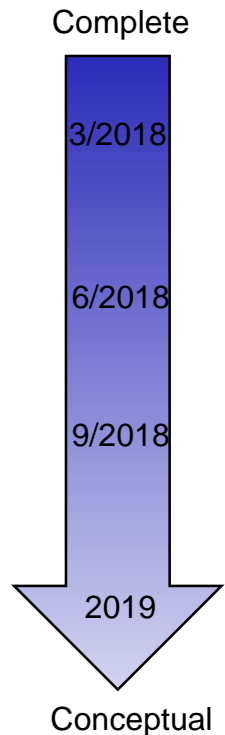
- Understand current applications
- Benchmark future system architectures

- OMNIO Tool

- Trace
- Statistics
- Preprocess
- Replay

- Current Status

- Complete
 - Trace + Statistics
- Prototyped
 - Preprocessor
- Future
 - Replay



Questions



bryan.flynt@noaa.gov

Back-Up

- FV3
 - nemsio output
 - 5 day forecast
 - 6 hour frequency

