SIOX: A Flexible Approach

Julian Kunkel Jakob Lüttgau Daniel Schmidtke[1]

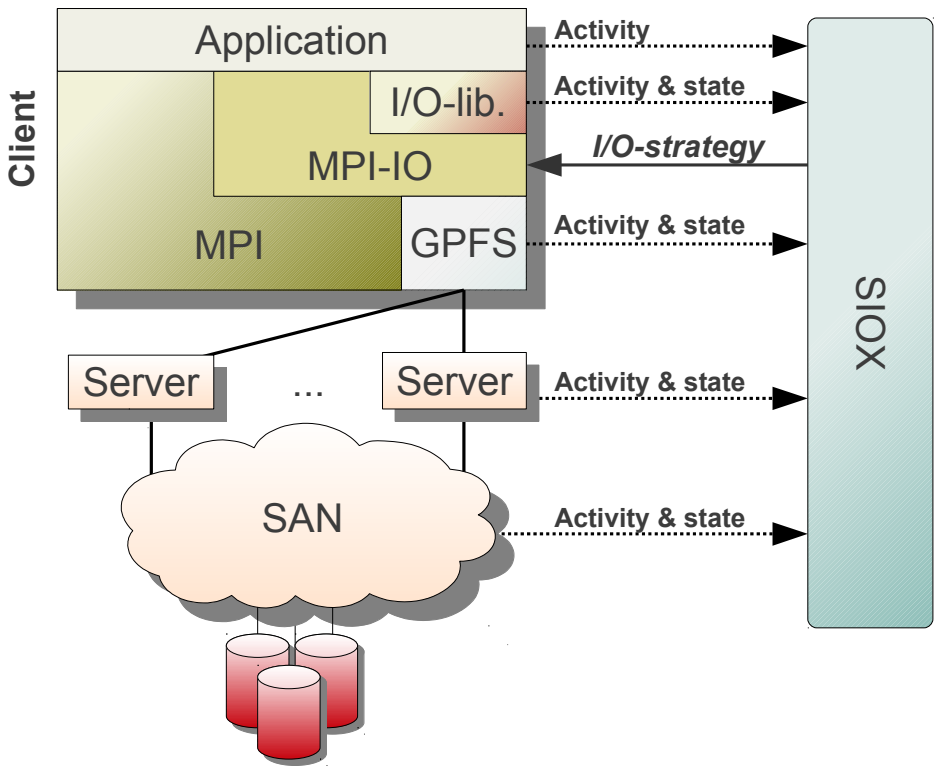
German Climate Computing Center

Analyzing Parallel I/O BoF
SC 2016

[1]University of Hamburg

Outline

Project Goals



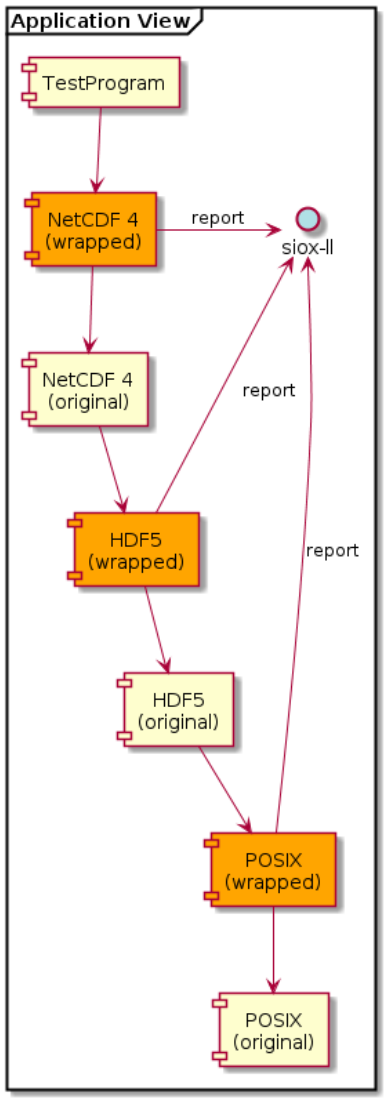
SIOX is a flexible prototype for

- collecting and analyzing
  - activity patterns and
  - performance metrics

in order to

- assess system performance
- locate and diagnose problem
- learn & apply optimizations
- intelligently steer monitoring

Low-Level API – Overview and Instrumentation



- C-Interface for monitoring / analysis
  - Monitor activities and system statistics
  - Query suitable optimization
- Relies on modules to
  - store activities
  - store and query (ontology and system) information
- Instrumentation uses low-level-API
  - A tool and workflow is provided; already instrumented:
    - POSIX (stdio and low-level)
    - MPIIO
    - NetCDF (initial)
    - HDF5 (initial)

Extensibility for Alternate APIs

Workflow

1. Annotate a header file
2. Tool `siox-wrapper-generator` creates intercepting libraries
   - Run-time instrumentation with LD_PRELOAD
   - Compile-time instrumentation using `ld -wrap`
3. `siox-inst` tool simplifies instrumentation

Header annotations for `MPI_File_write_at()`

```
//@activity
//@activity_link_size fh
//@activity_attribute filePosition offset
//@splice_before ''int intSize; MPI_Type_size(datatype, &intSize);
                  uint64_t size=(uint64_t)intSize*(uint64_t)count;''
//@activity_attribute bytesToWrite size
//@error ''ret!=MPI_SUCCESS'' ret
int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void * buf, int count,
                      MPI_Datatype datatype, MPI_Status * status);
```

Modularity of SIOX

- The SIOX architecture is flexible and developed in C++ components
- License: LGPL, vendor friendly
- Upon start-up of (instrumented) applications, modules are loaded
- Configuration file defines modules and options
  - Choose advantageous plug-ins
  - Regulate overhead
- For debugging, **reports** are output at application termination
  - SIOX may gather statistics of (application) behavior / activity
  - Provide (internal) module statistics

Features of the Working Prototype

- Activity plug-ins
  - *GenericHistory* plug-in tracks performance, proposes MPI hints
  - Fadvise (ReadAhead) injector
  - *FileSurveyor* prototype – Darshan-like
- Monitoring
  - Application (activity) behavior
  - Ontology and system information
  - Data can be stored to various backends (Files, Postgres, [Plugin])
  - Trace Reader (POSIX)
  - Trace Replay (POSIX)
  - Transparent Runtime Optimization

Reporting: FileSurveyor

- Easy to collect and track relevant application statistics
- FileSurveyor prototype collects POSIX/MPI access statistics
- Only 1000 LoC
- *... Yes we'll pretty print things at some point ...*

```
[...] "(Aggregated over all files)"/Accesses = (40964,40964,40964)
...
[...] "/mnt/lustre/file.dat"/Accesses = (40964,40964,40964)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Random, long seek = (20481.8,20480,20482)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Random, short seek = (0,0,0)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Sequential = (0.2,0,2)
[...] "/mnt/lustre/file.dat"/Bytes = (8.38861e+09,8.38861e+09,8.38861e+09)
[...] "/mnt/lustre/file.dat"/Bytes/Read per access = (204780,204780,204780)
[...] "/mnt/lustre/file.dat"/Bytes/Total read = (4.1943e+09,4.1943e+09,4.1943e+09)
[...] "/mnt/lustre/file.dat"/Seek Distance/Average writing = (1.0238e+06,1.0238e+06,1.02382e+06)
[...] "/mnt/lustre/file.dat"/Time/Total for opening = (3.9504e+08,3.66264e+08,4.38975e+08)
[...] "/mnt/lustre/file.dat"/Time/Total for reading = (1.47169e+11,1.0968e+11,1.76617e+11)
[...] "/mnt/lustre/file.dat"/Time/Total for writing = (1.08783e+12,1.03317e+12,1.16192e+12)
[...] "/mnt/lustre/file.dat"/Time/Total for closing = (1.0856e+11,6.11782e+10,1.46834e+11)
[...] "/mnt/lustre/file.dat"/Time/Total surveyed = (1.34568e+12,1.34568e+12,1.3457e+12)
```

Figure: Example report created by FileSurveyor and aggregated by MPIReporter (shortened excerpt). The number format is (average, minimum, maximum).

Trace Reader

Concepts

- Supports different file and database back-ends
- Plug-in based
  - Text output
  - Time-offset plots for files

Example text output created by the trace-reader

```
0.0006299 ID1 POSIX open(POSIX/descriptor/filename="testfile",
  POSIX/descriptor/filehandle=4) = 0
0.0036336 ID2 POSIX write(POSIX/quantity/BytesToWrite=10240,
  POSIX/quantity/BytesWritten=10240, POSIX/descriptor/filehandle=4,
  POSIX/file/position=10229760) = 0 ID1
0.0283800 ID3 POSIX close(POSIX/descriptor/filehandle=4) = 0 ID1
```

Changing I/O Behavior on the Fly

Motivation

- What is the benefit of implementing an I/O optimization in the code?
  - Traditional methodology: (estimate), implement, evaluate

  ⇒ Time consuming!

Alternative strategies

- Trace and record application I/O, then alter and replay I/O
- Intercept I/O and manipulate directly

Pro/Cons

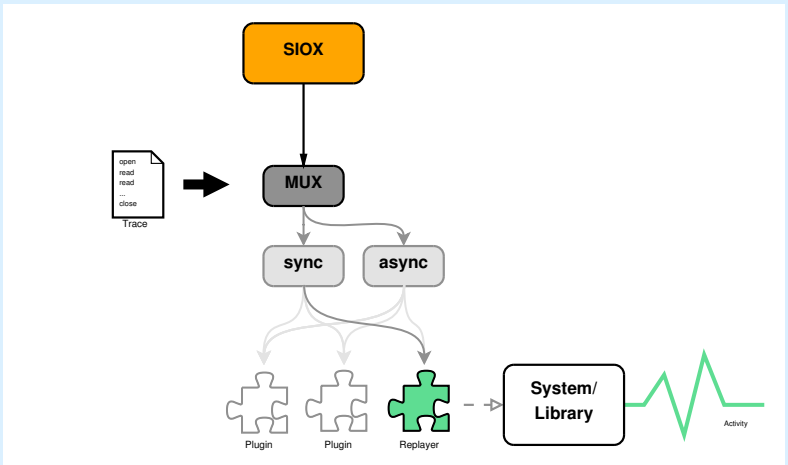
\+ Implement an optimization once, test/run with many applications

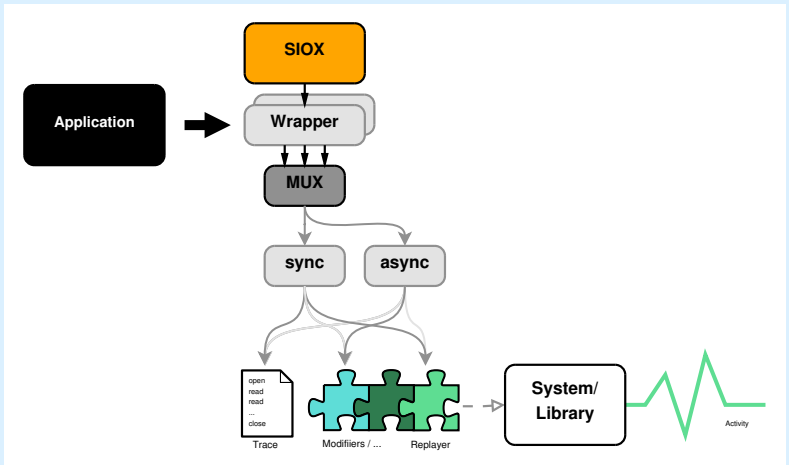
\- Looses some performance due to interception

Modifications to SIOX

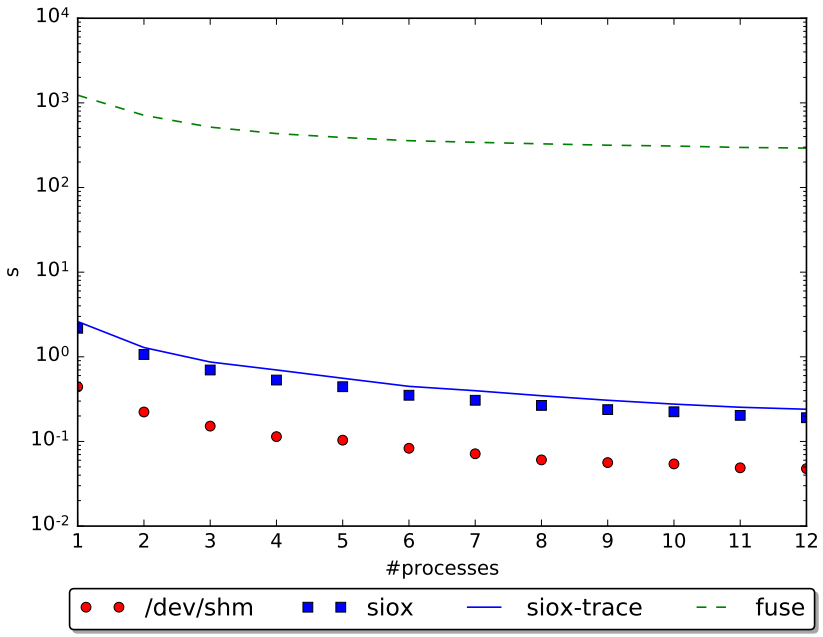
- The proposed strategies have been implemented (for a subset of POSIX)
- We extracted the execution of calls from the monitoring path
  - Now, a playback plugin executes calls
  - The same plugins can be used in trace/replay scenarios
  - \+ This also reduced the complexity of the interception layer

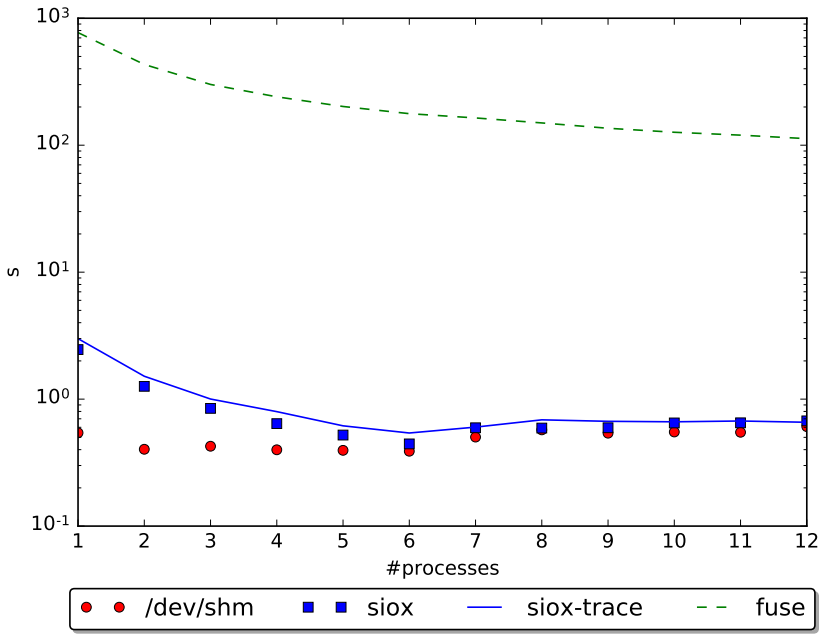
Modification during trace-replay and online-playback



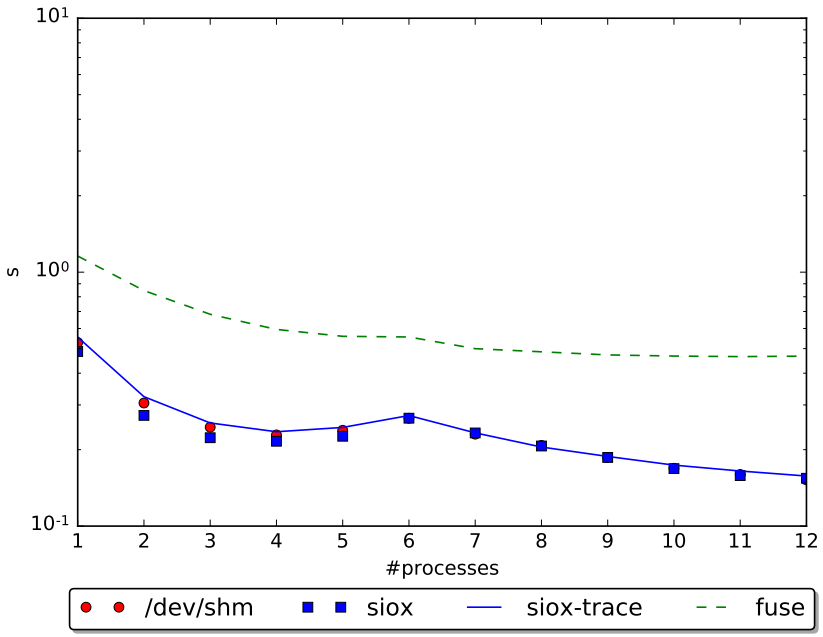


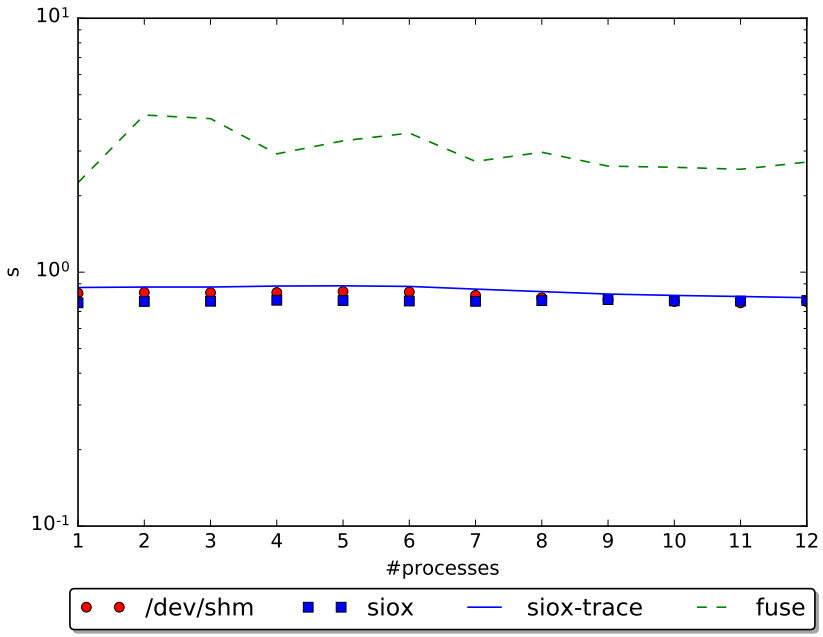
SHMEM vs. SIOX vs. FUSE for Small Requests (16k)





SHMEM vs. SIOX vs. FUSE for Large Requests (1GiB)





Summary

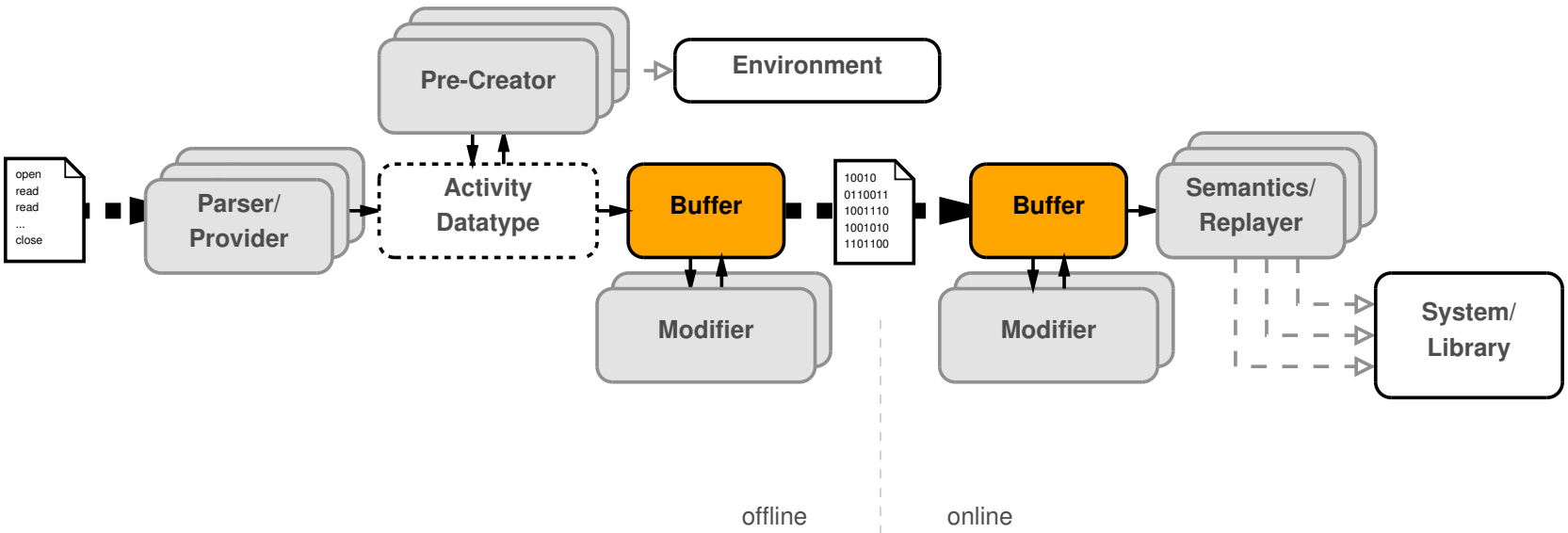
- SIOX aims to capture and optimize I/O
- We can change behavior without modifying code!
  - Design the optimization once, apply on many applications
  - Useful to evaluate strategies without implementing them (again)
- The goal of SIOX is a modular and open system

Outlook

- Apply SIOX to more applications on DKRZ's Mistral supercomputer
  - 3.6 PetaFLOPS
  - 54 PB (Lustre)
- Improve intelligence
  - Information about predicted storage class (e.g. cached, uncached)
  - Performance predictors for anomaly detection
  - Machine learning plug-ins
  - Online optimization
  - System-wide reasoning logic
- Stretch monitoring annotations to also create replay plugins
- Act as source for DKRZ system-wide monitoring system
  - Will integrate statistics e.g. knowledge/assessments for jobs
  - Optional to run applications with SIOX

Thank you.

A complete environment for testing alternative optimizations



System Configuration

Test system

- 10 compute nodes
- 10 I/O nodes with Lustre

Compute Nodes

- Dual-socket Intel Xeon X5650@2.67 GHz
- Ubuntu 12.04
- Applications are compiled with: GCC 4.7.2, OpenMPI 1.6.5

I/O Nodes

- Intel Xeon E3-1275@3.4 GHz, 16 GByte RAM
- Seagate Barracuda 7200.12 (ca. 100 MiB/s)
- CentOS 6.5, Lustre 2.5

Overhead

- Due to asynchronous handling applications are never stalled
- A call to SIOX in the order of several μs
  - We see room for improvement, and have some solutions in mind!
- Initialization of SIOX with fixed costs
- SIOX IPC handles 90,000 (1 KiB) msgs per second
- PostgreSQL only 3,000 activities (we'll need to invest more time)

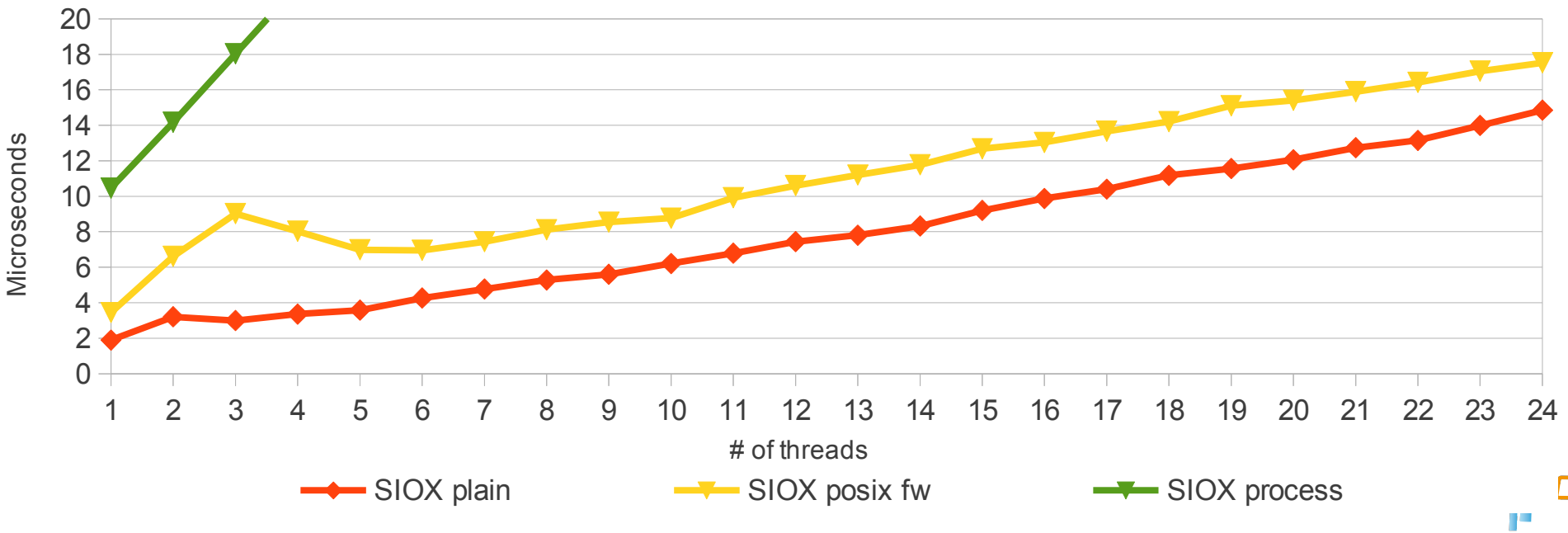


Figure: Overhead per thread due to critical regions in the modules.

Observable Performance – Discussion

Bad news

- For fast I/O operations several μs is expensive
- Additionally, locks protect several modules

$\Rightarrow$ I/O calls are synchronized (max. 100K Ops/s)

Good news

- We are already monitoring overhead

$\Rightarrow$ We will integrate methods to control the overhead

- Flexible and easy configuration can strip costly calls

Application runs?

For the ICON climate model, only initialization overhead is measurable

- A DB cache module reducing overhead