Improving Nektar++ IO Performance for Cray XC Architecture

Michael Bareford, ARCHER CSE Team
michael.bareford@epcc.ed.ac.uk

With thanks to Nick Johnson and Michèle Weiland
EPCC, University of Edinburgh

Contents

- Introduction
  - Nektar++
  - IO Methods (XML, HDF5, SIONlib)
  - ARCHER, Cray XC30

- Test Cases
  - Small, Aortic Arch, ~ 2.5 MB
  - Large, Road Racing Car, ~ 5.5 GB

- Nektar++ IO Classes
  - Checkpoint file formats

- Results and Conclusions

Nektar++

Nektar++ v4.4.0 (MPI)
http://www.nektar.info

Imperial College
London

An open-source spectral element code that combines the accuracy of spectral methods with the geometric flexibility of finite elements, specifically, *hp*-version FEM.

Supports several scalable solvers for many sets of partial differential equations, from (in)compressible Navier-Stokes to the bidomain model of cardiac electrophysiology.

Nektar++ IO Methods

1. **XML**, one checkpoint file per process

   Performance expected to degrade significantly above 10^4 cores due to meta IO. And HPC resources can be restricted by file count (inode limit).

2. **HDF5** (v1.8.14), single-shared checkpoint file

   Necessary to record explicitly which parts of a HDF5 dataset belong to which MPI process.

   https://www.hdfgroup.org

3. **SIONlib** (v1.6.2), single-shared checkpoint file

   Meta-data concerning decomposition is recorded automatically.

   https://apps.fz-juelich.de/jsc/sionlib/docu/index.html

Introducing ARCHER

Advanced **R**esearch **C**omputing **H**igh **E**nd **R**esource

www.archer.ac.uk

Introducing ARCHER

Cray XC30 MPP, 4920 Compute Nodes

- Dual Intel Xeon processors (Ivy Bridge), 24 cores, 64 GB

Lustre File System

- 48 Object Storage Targets (OSTs)
- Default striping is 1 MiB stripe stored on 1 OST
- Theoretical peak performance of 30 GB/s

Strong scaling tests run over a range of core (node) counts.

- 768 (32), 1536 (64), 3072 (128) and 6144 (256)

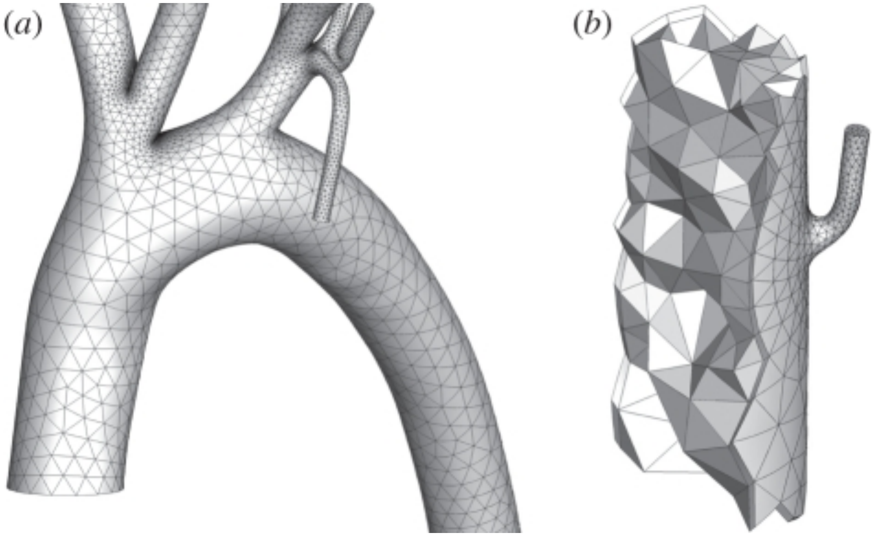
Small Test Case: Aortic Arch

Mesh contains ~150k elements (tetrahedra)

Simulates blood flow through a (rabbit's) aortic arch .

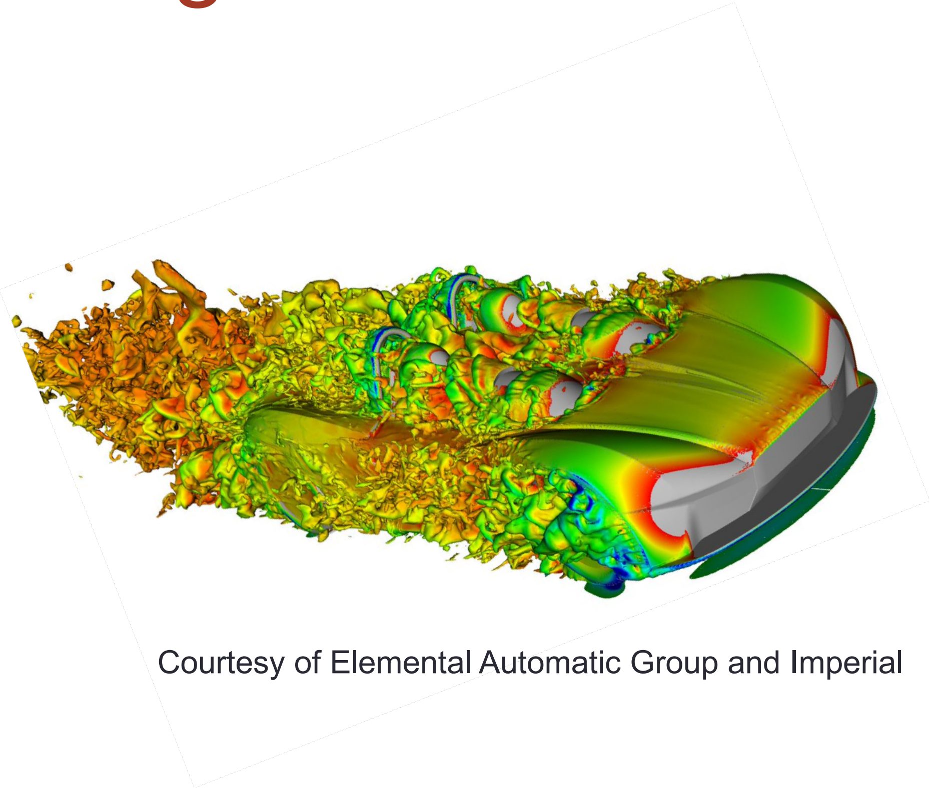
Advection-diffusion-reaction solver (mass transport).

Total checkpoint data approx. 2.5 MB



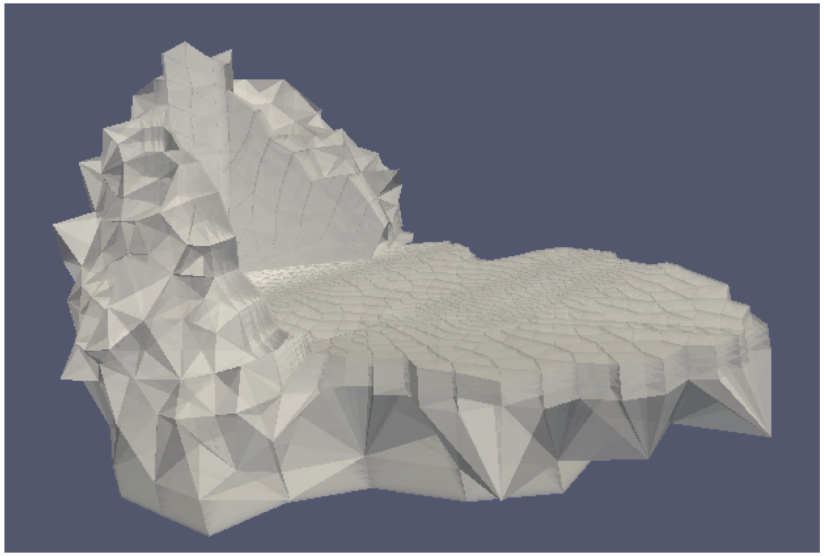
Vincent, Plata, Hunt et al., J R Soc Interface. 2011

Large Test Case: Racing Car



Courtesy of Elemental Automatic Group and Imperial

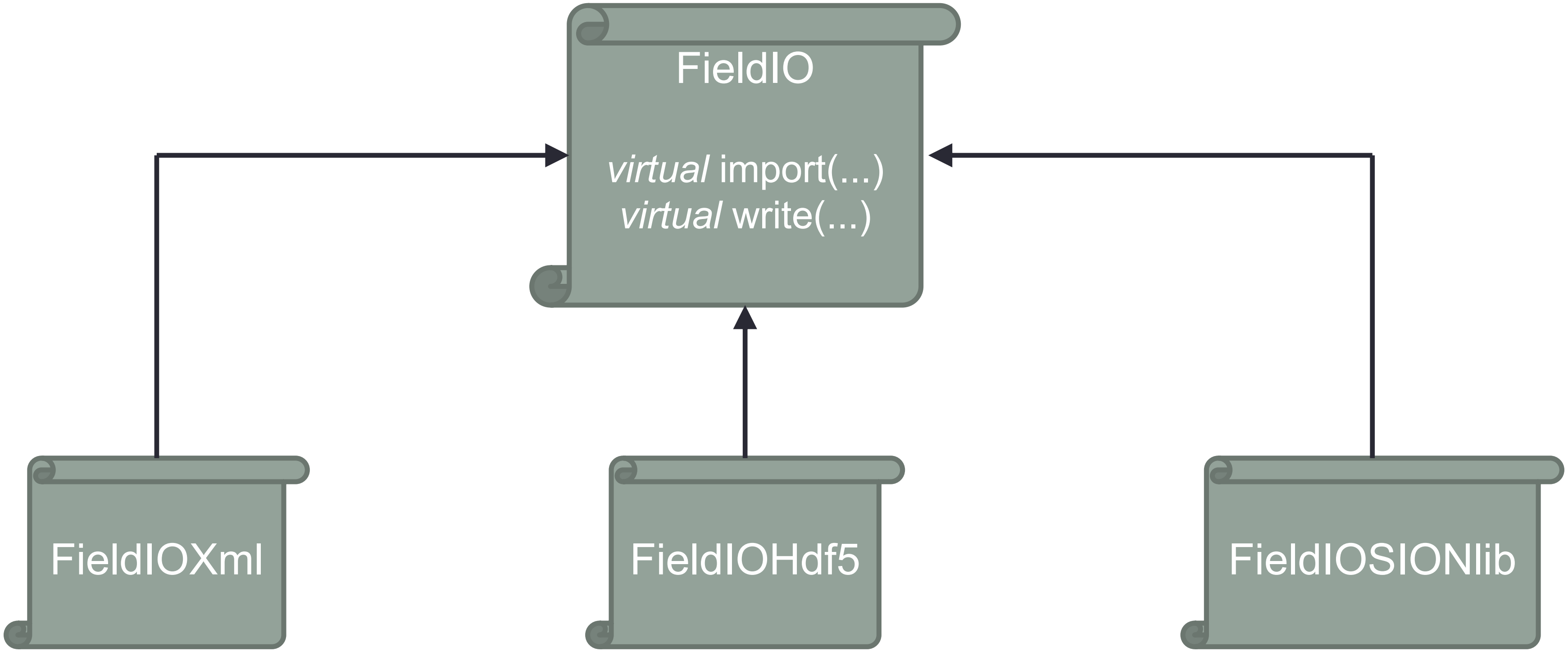
Mesh contains ~3.5 million elements
(tetrahedra and prisms)



Total checkpoint data approx. 5.5 GB

Simulates air flow around a road racing car (www.elementalcars.co.uk) using Incompressible Navier Stokes solver.

Nektar++ FieldIO Hierarchy



FieldIO

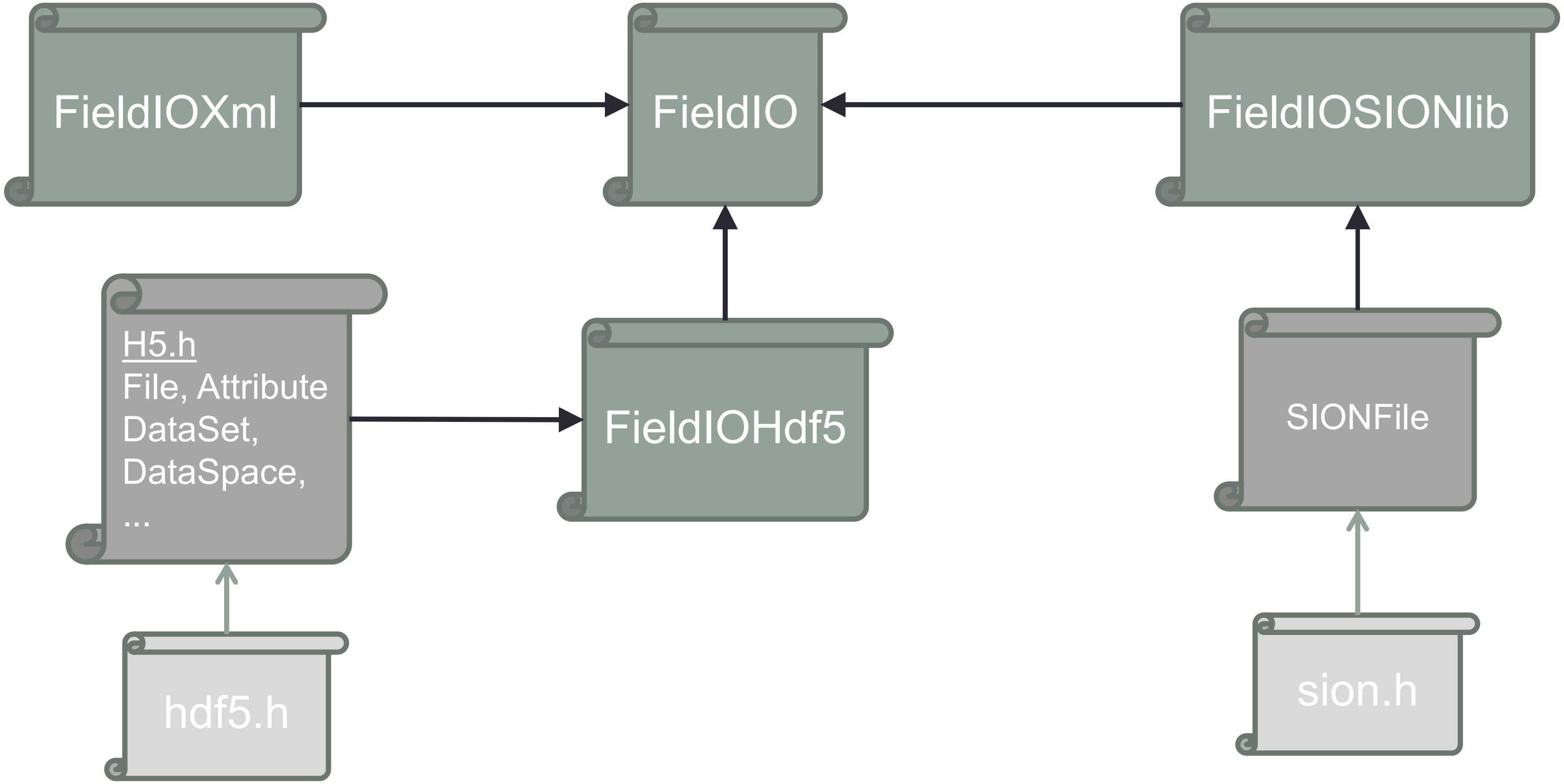
virtual import(...)
virtual write(...)

FieldIOXml

FieldIOHdf5

FieldIOSIONlib

Nektar++ FieldIO Hierarchy



FieldIOXml

FieldIO

FieldIOSIONlib

H5.h
File, Attribute
DataSet,
DataSpace,
...

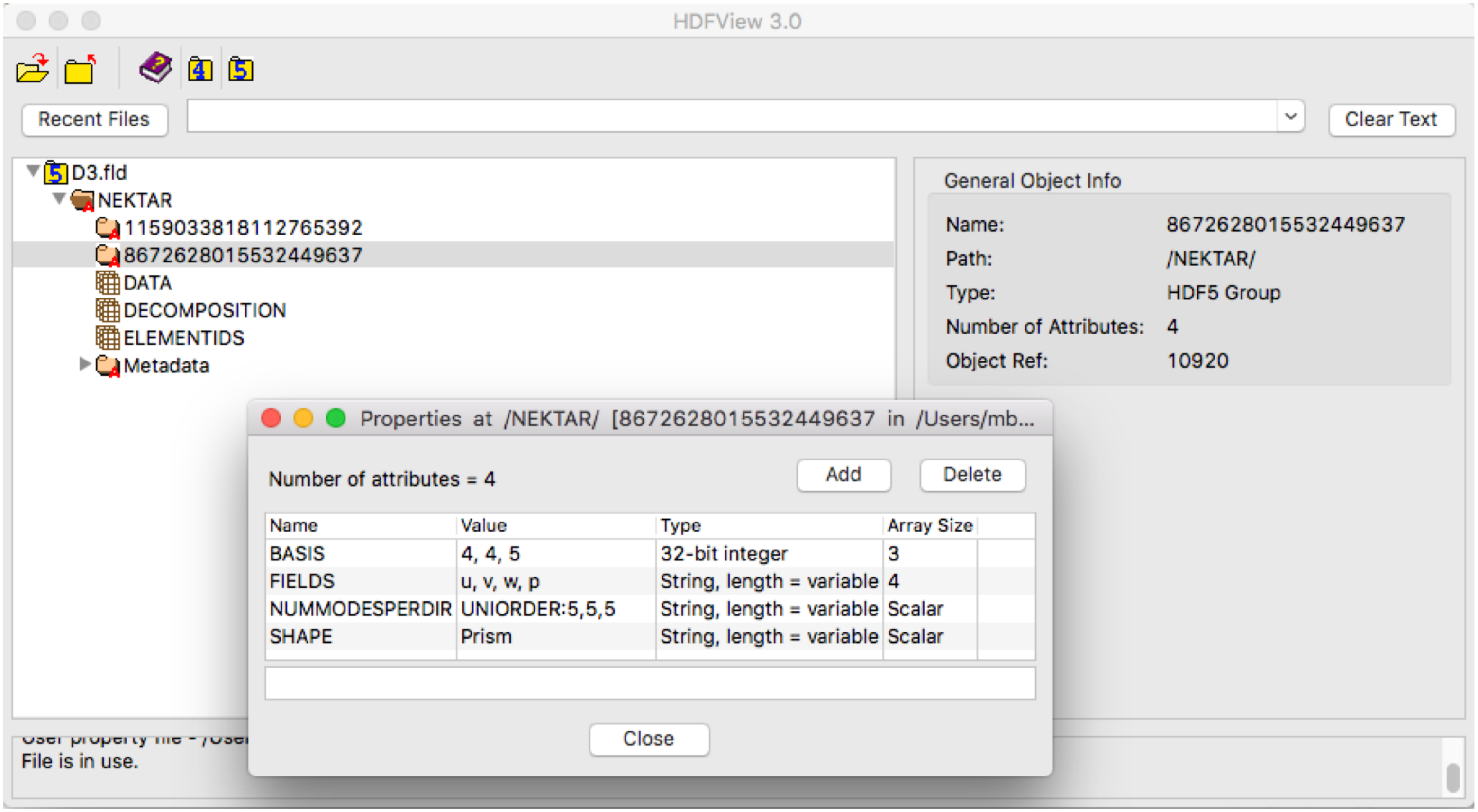
FieldIOHdf5

SIONFile

hdf5.h

sion.h

HDF5 Checkpoint File Format



- D3.fld
  - NEKTAR
    - 1159033818112765392
    - 8672628015532449637
    - DATA
    - DECOMPOSITION
    - ELEMENTIDS
    - Metadata

General Object Info

Name:	8672628015532449637
Path:	/NEKTAR/
Type:	HDF5 Group
Number of Attributes:	4
Object Ref:	10920

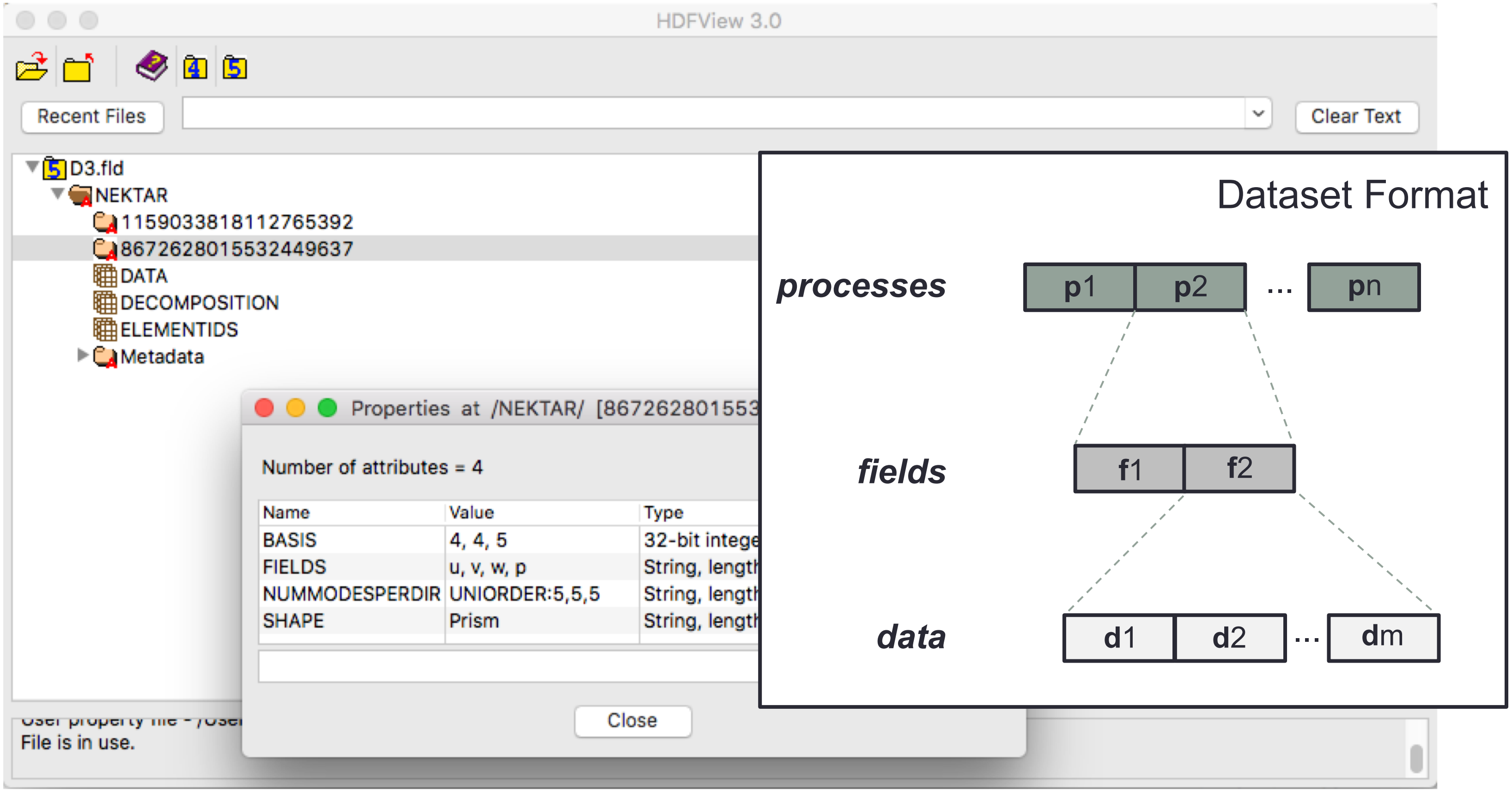
Properties at /NEKTAR/ [8672628015532449637 in /Users/mb...

Number of attributes = 4

Name	Value	Type	Array Size
BASIS	4, 4, 5	32-bit integer	3
FIELDS	u, v, w, p	String, length = variable	4
NUMMODESPERDIR	UNIORDER:5,5,5	String, length = variable	Scalar
SHAPE	Prism	String, length = variable	Scalar

User property file - /User
File is in use.

HDF5 Checkpoint File Format



D3.fld
NEKTAR
1159033818112765392
8672628015532449637
DATA
DECOMPOSITION
ELEMENTIDS
Metadata

Properties at /NEKTAR/ [867262801553

Number of attributes = 4

Name	Value	Type
BASIS	4, 4, 5	32-bit intege
FIELDS	u, v, w, p	String, length
NUMMODESPERDIR	UNIORDER:5,5,5	String, length
SHAPE	Prism	String, length

User property file - /User
File is in use.

Nektar++ Class: FieldIOHdf5

The job of creating the structure of the HDF5 file is delegated to a single root process.

Root process writes the decomposition data having collected all the necessary metadata from the other processes. Then scatters dataset indexes to all other processes.

All MPI processes *write collectively* to the appropriate locations within the element datasets.

Possible for each process to perform one write no matter how many fields it is handling. This is done through the use `H5S_SELECT_OR` to access a dataset at multiple locations using just one write operation.

Nektar++ Class: FieldIOHdf5

The job of creating the structure of the HDF5 file us delegated to a single root process.

Root also writes the decomposition data having collected all the necessary metadata from the other processes.

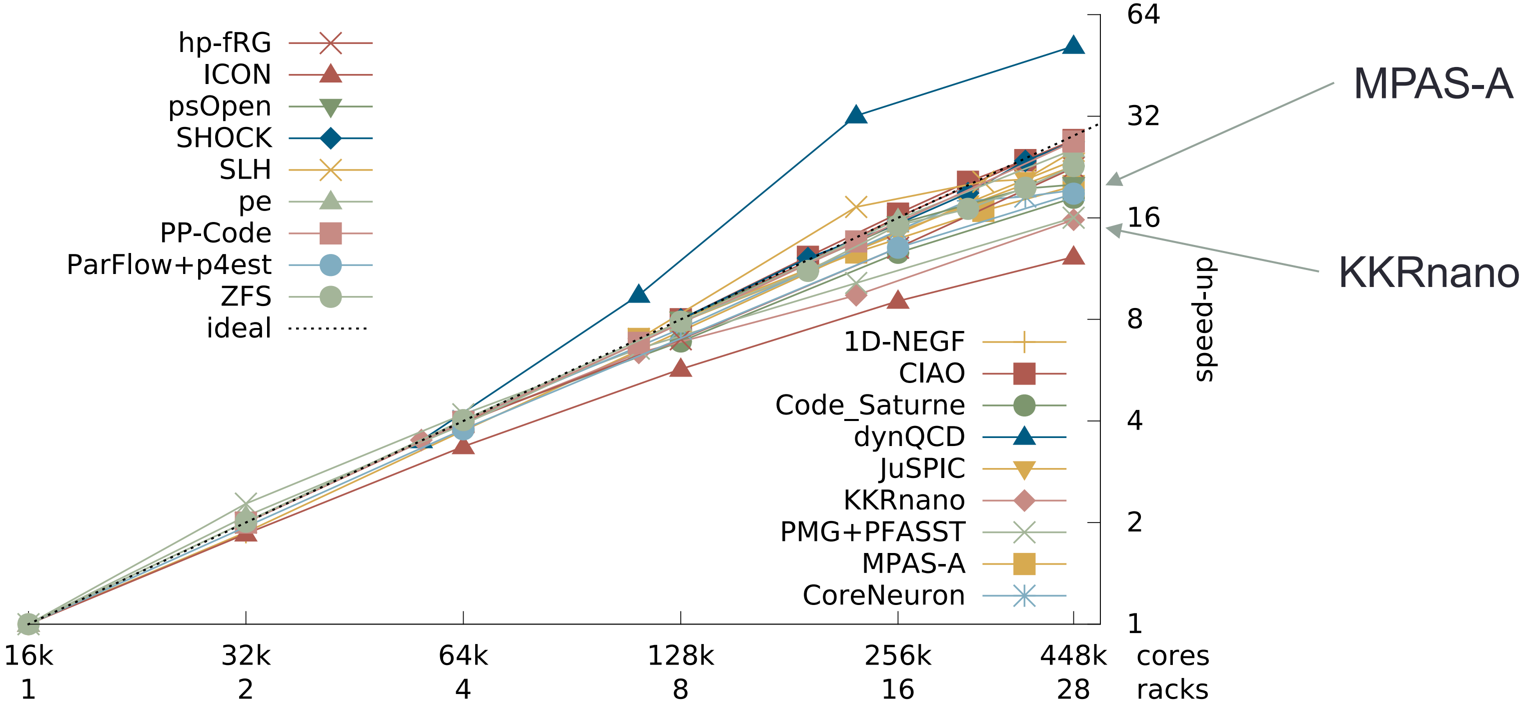
All MPI processes write collectively to the appropriate locations within the element datasets. Possible for each process to perform one write no matter how many fields it is handling. This is done through the use H5S_SELECT_OR to write data for more than one field at the same time.

The *import* routine offers the capability for the caller to redefine the mapping between processes and elements. This flexibility incurs a cost in the form of extra reads: specifically, the entire decomposition dataset has to be communicated to all processes.

Although HDF5 will buffer decomposition data.

Why SIONlib?

Strong Scaling of High-Q Club member application codes on JUQUEEN (Blue Gene/Q)



MPAS-A

KKRnano

The High-Q Club: Experience with Extreme-scaling Application Codes
Brommel, et al. 2018

Nektar++ Class: FieldIOSIONlib

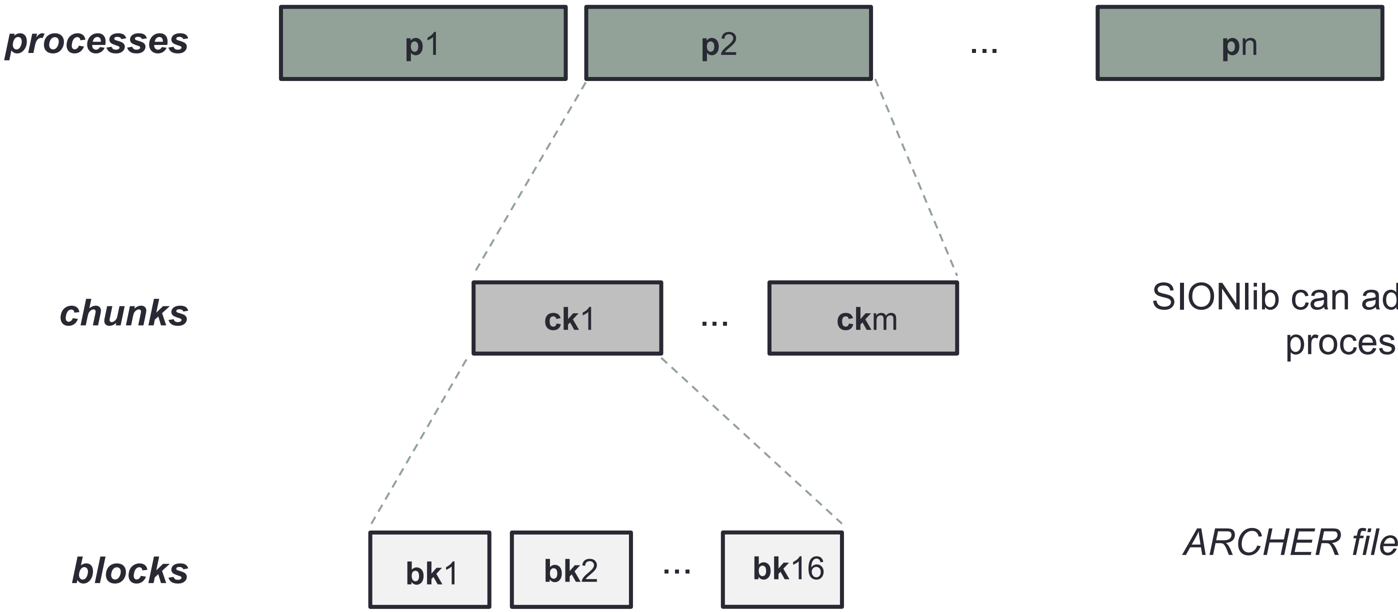
No need to record decompositional data explicitly: the SIONlib library does that for us.

Each process simply loops over the fields it handles, writing out or reading in the field definition, element IDs and element data in turn.

Collective writes need extra care if processes are handling different number of fields. For the racing car test case, most processes will be handling two field types (prisms and tetrahedrons), but some may be handling prisms only.

A quick solution is for those prism-only processes to output a second field with dummy data (a single byte). Of course, only the number of `sion_coll_fwrite` calls need to match between processes not the amount of data written.

SIONlib Checkpoint File Format



SIONlib can add extra chunks to a process area if necessary.

ARCHER file system block size 65,536 bytes

The section of the checkpoint file assigned to any MPI process will not cross a file system boundary.

FieldIOBenchmarker

First, we ran the aorta and racing car simulations for the core counts mentioned (768 to 6144) in order to generate the various checkpoint files.

The IO benchmarker was used to perform read and write operations 10 times for each core count, allowing an average execution time to be calculated.

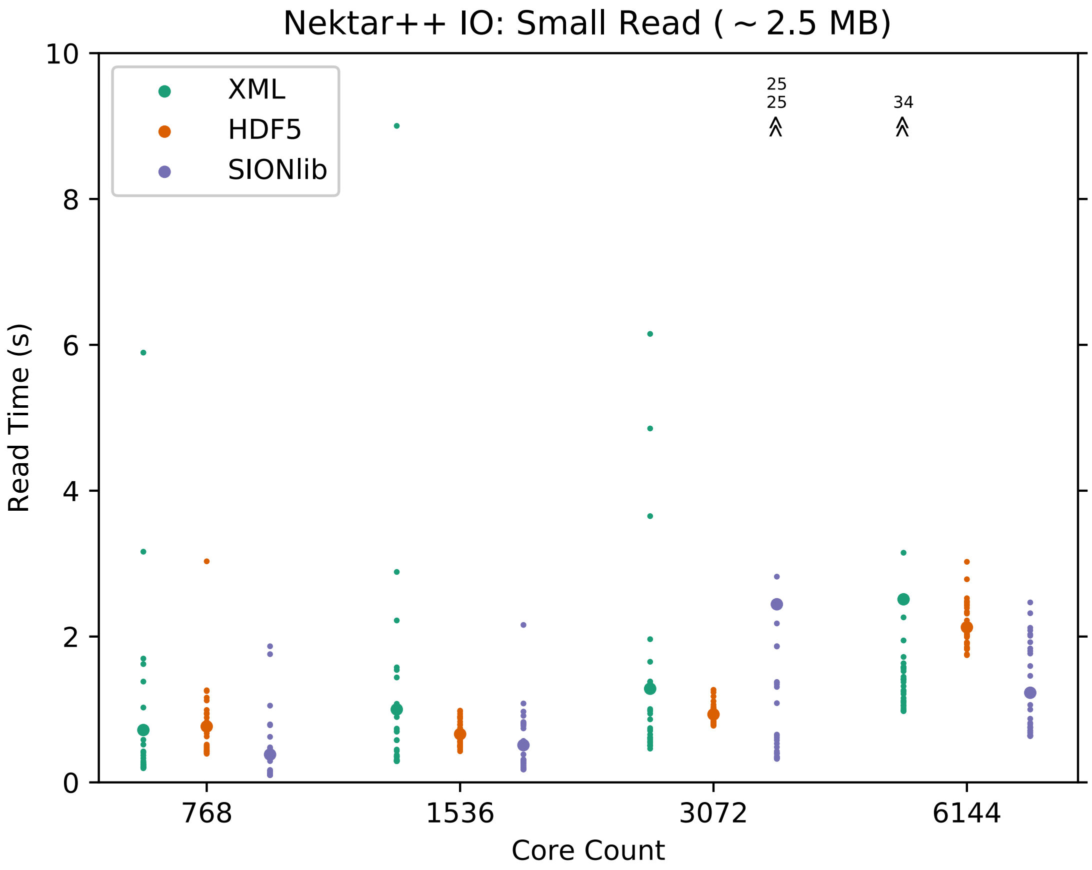
```
iomethod = [xml, hdf5, sionlib]

for i in 1..10
  for j in 1..len(iomethod)
    aprun –n ncores benchmarker(FieldIO::import, iomethod[j])

for i in 1..10
  for j in 1..len(iomethod)
    aprun –n ncores benchmarker(FieldIO::write, iomethod[j])
```

60 separate read/write operations per core count

Aortic Arch: Checkpoint Read



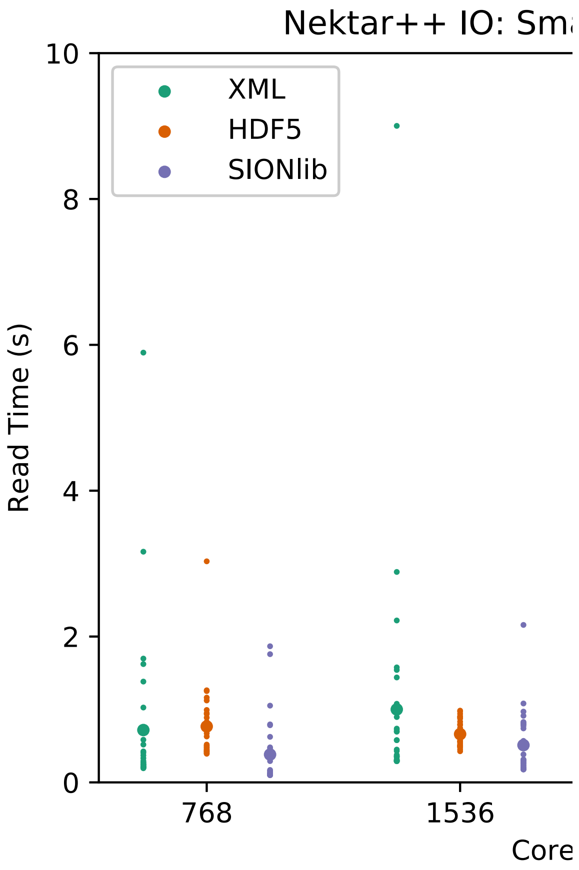
Each vertical column of dots covers several datasets (30 data points) taken on different days.

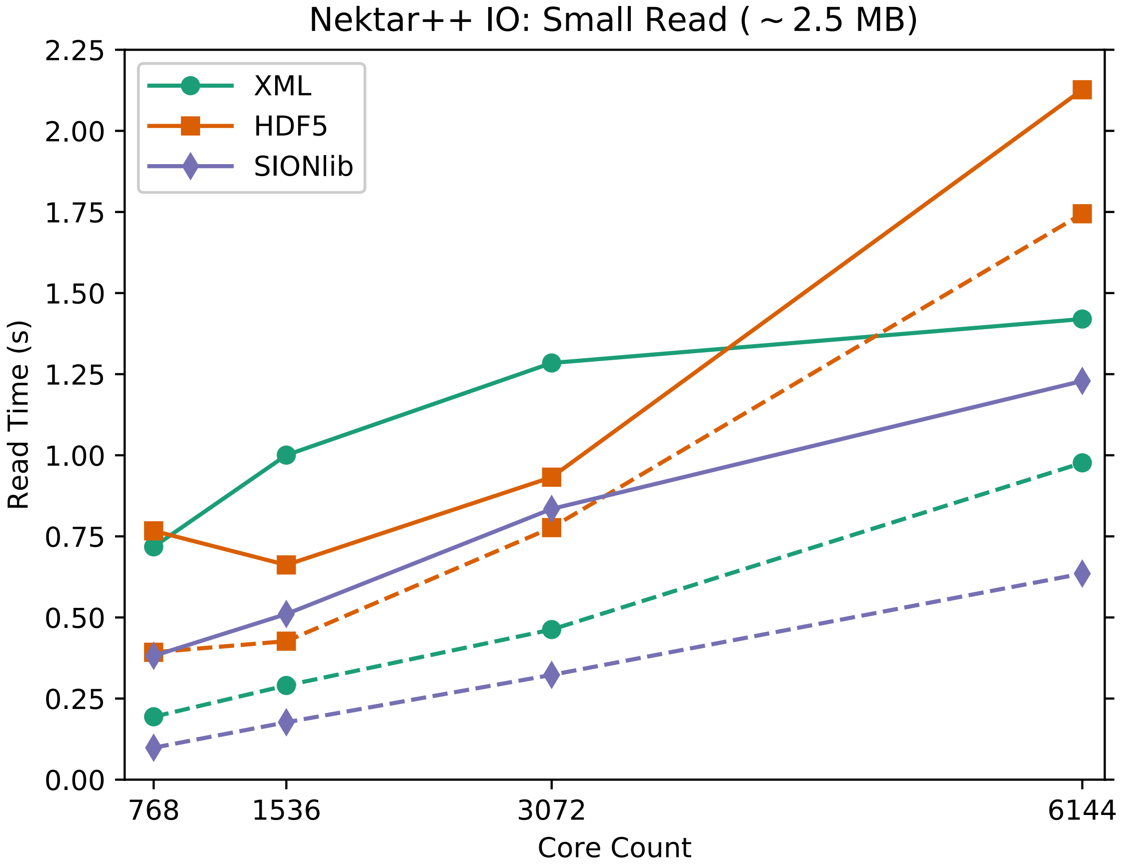
Enlarged data points represent the average time.

Extreme Outliers > 20 IQR (above top of 3rd quartile)

SIONlib achieved the fastest read times

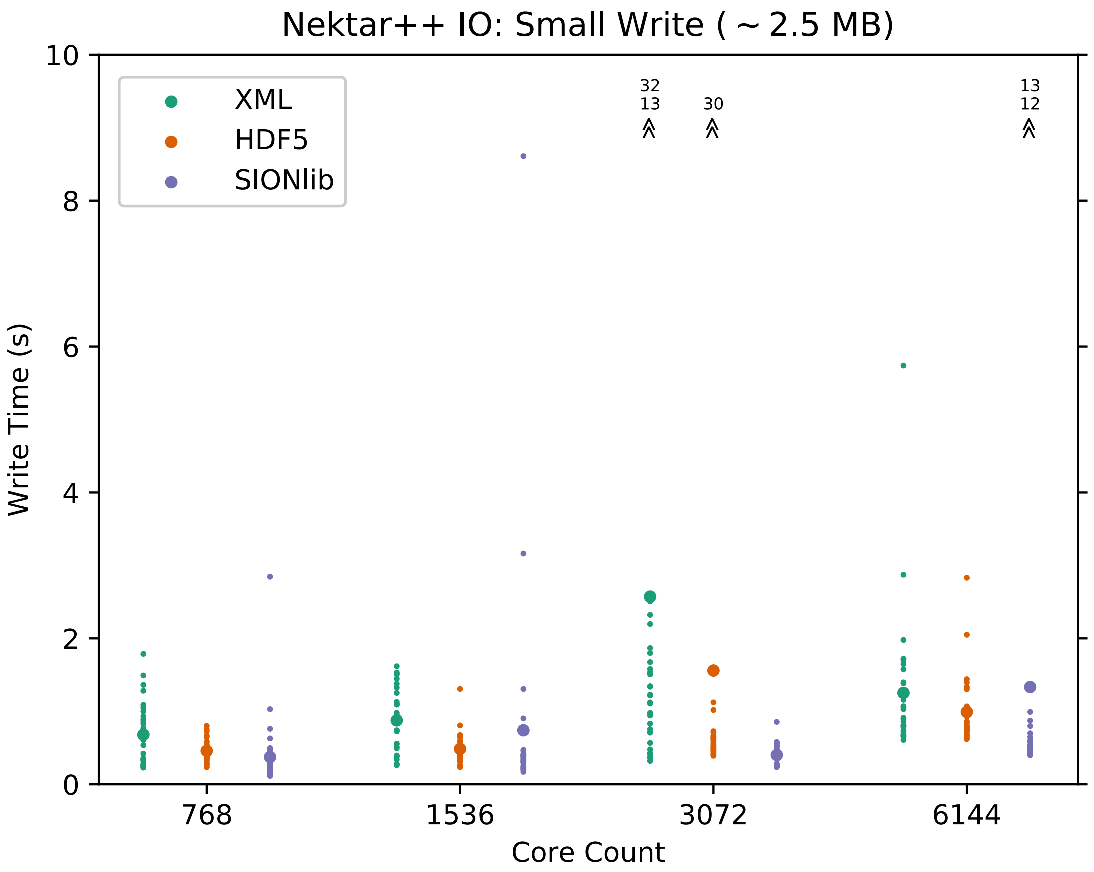
Aortic Arch: Checkpoint Read





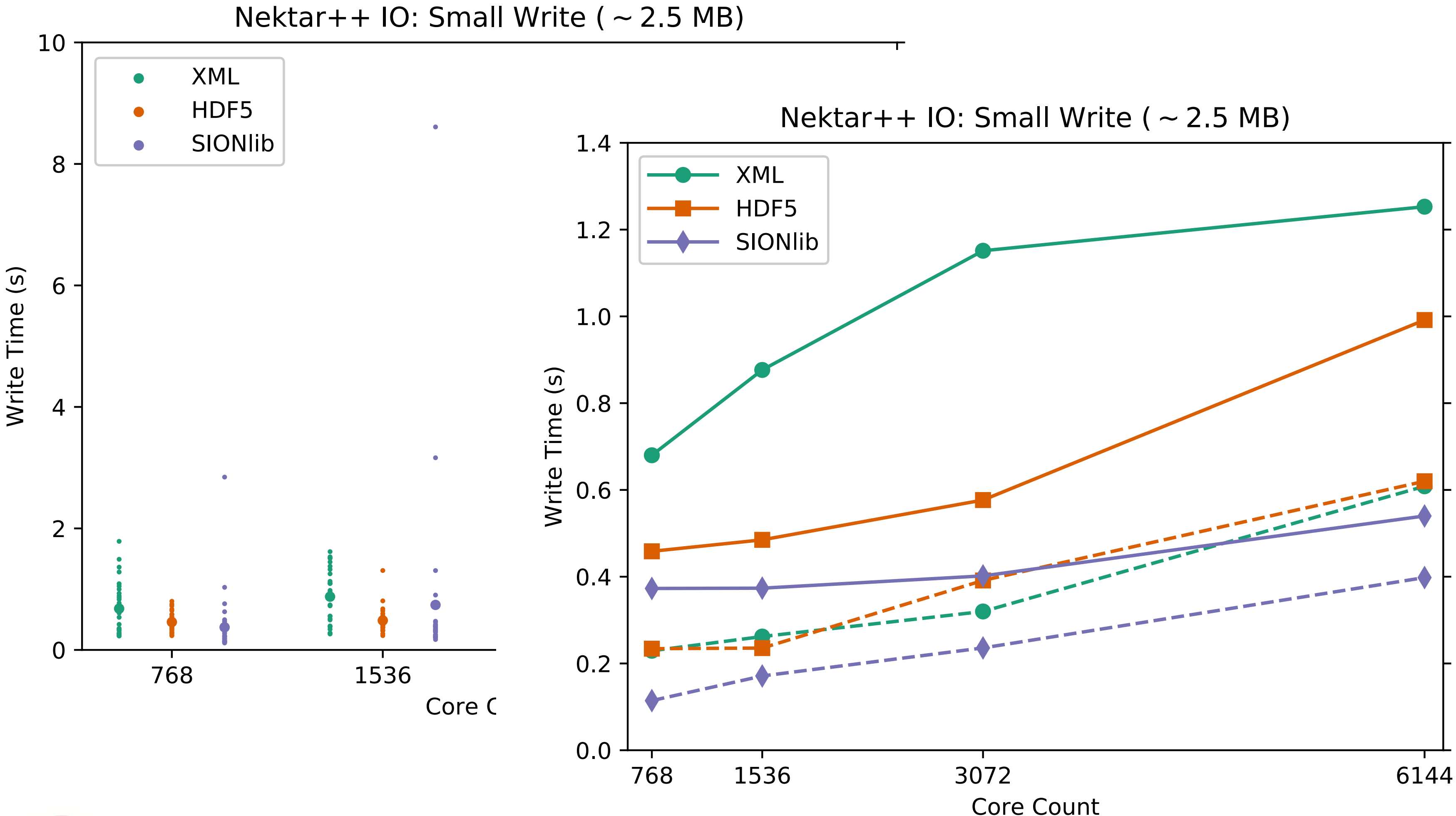
SIONlib achieved the fastest read times

Aortic Arch: Checkpoint Write



Extreme Outliers > 12 IQR appear for all IO methods.

Aortic Arch: Checkpoint Write



Writing to Single Shared File

HDF5 nominates a subset of MPI ranks to act as data *aggregators* or writers.
#aggregators = 48

SIONlib also funnels data to designated writers or *collectors*.
#collectors = 32

In SIONlib *collective* mode, a collector receives data from process *i* and writes that data to the area of the checkpoint file reserved for process *i*.

However, it's more performant for each collector to instead write all the data into their own area. This is know as *collectivemerge* mode.

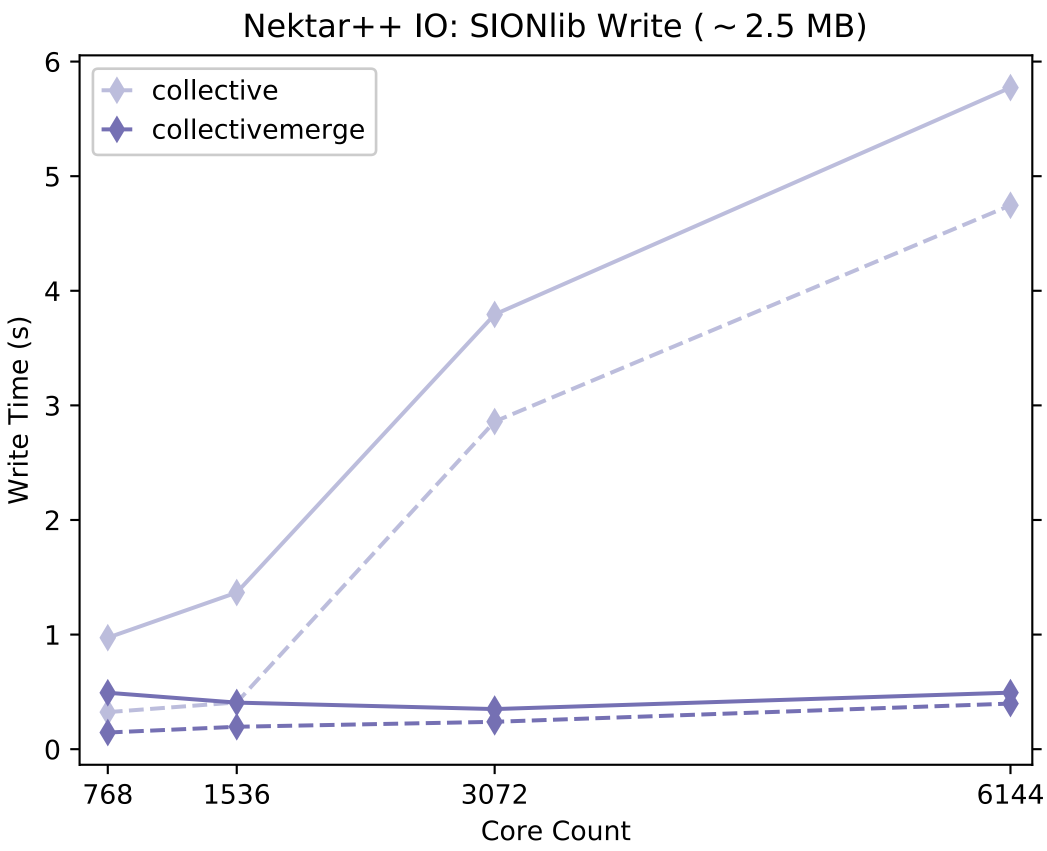
Writing to Single Shared File

HDF5 nominates a sub
#aggregators = 48

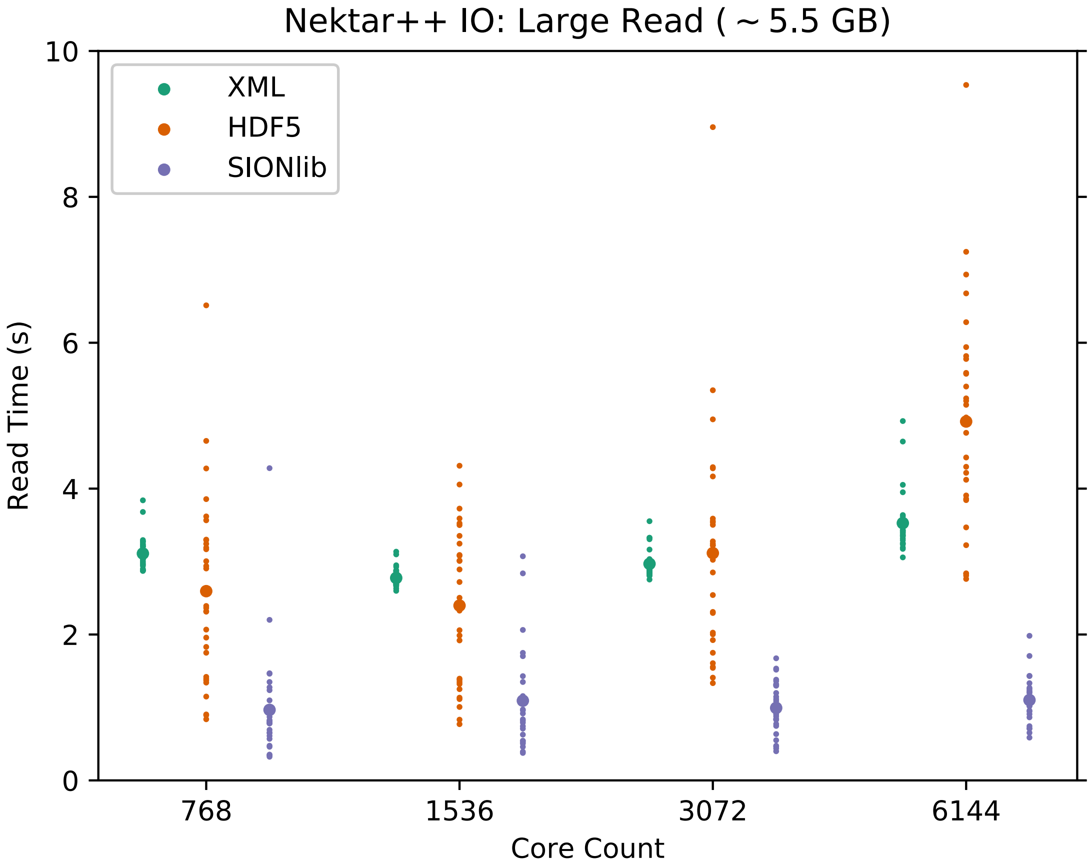
SIONlib also funnels da
#collectors = 32 regard

In SIONlib *collective* m
process *i* and writes tha
reserved for process *i*.

However, it's more perf
all the data into their ov
mode.



Racing Car: Checkpoint Read



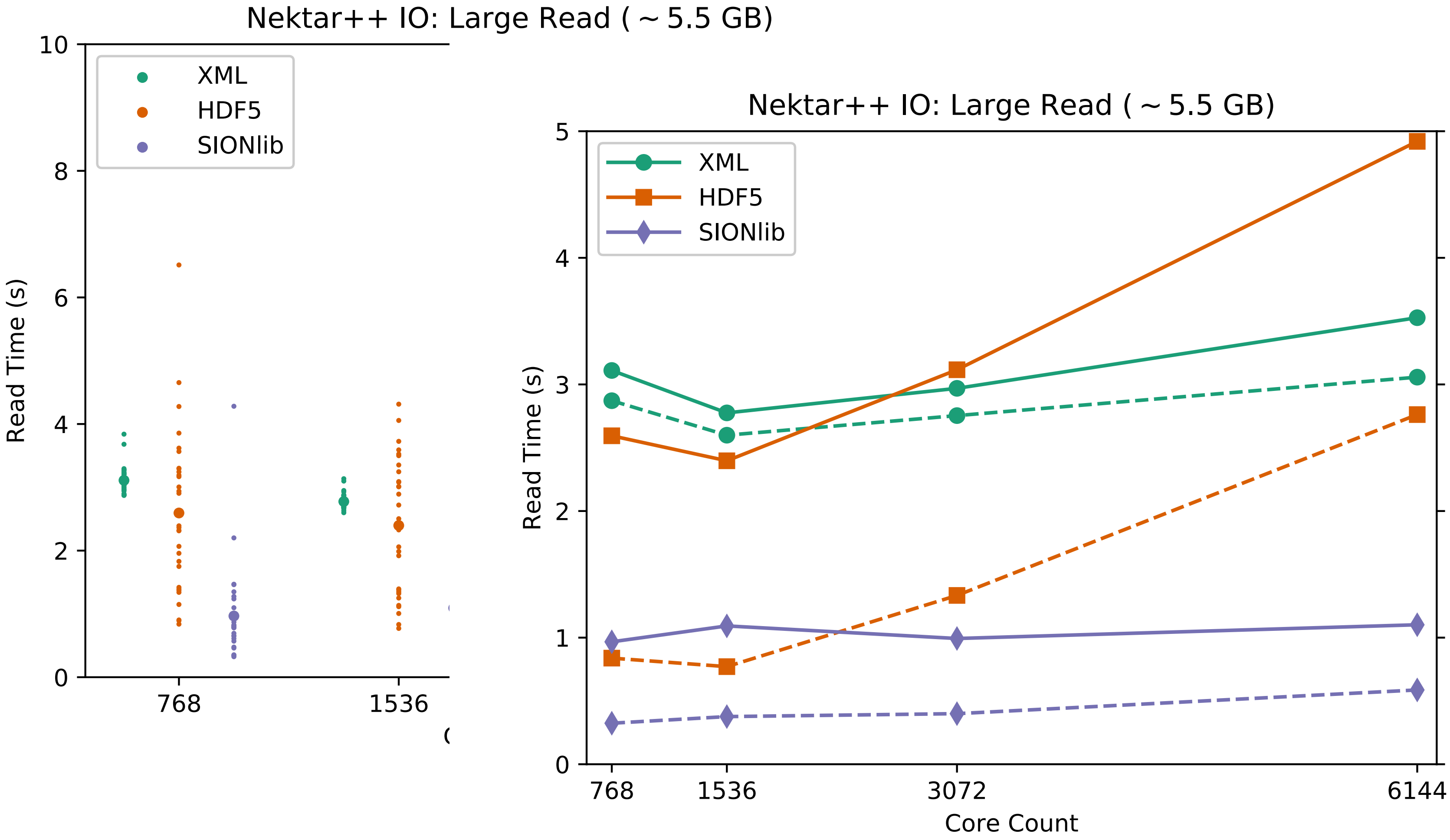
Default Lustre file settings (1 MiB, 1 OST) used for aorta arch checkpoint files.

Racing car test case is 2000 times larger.

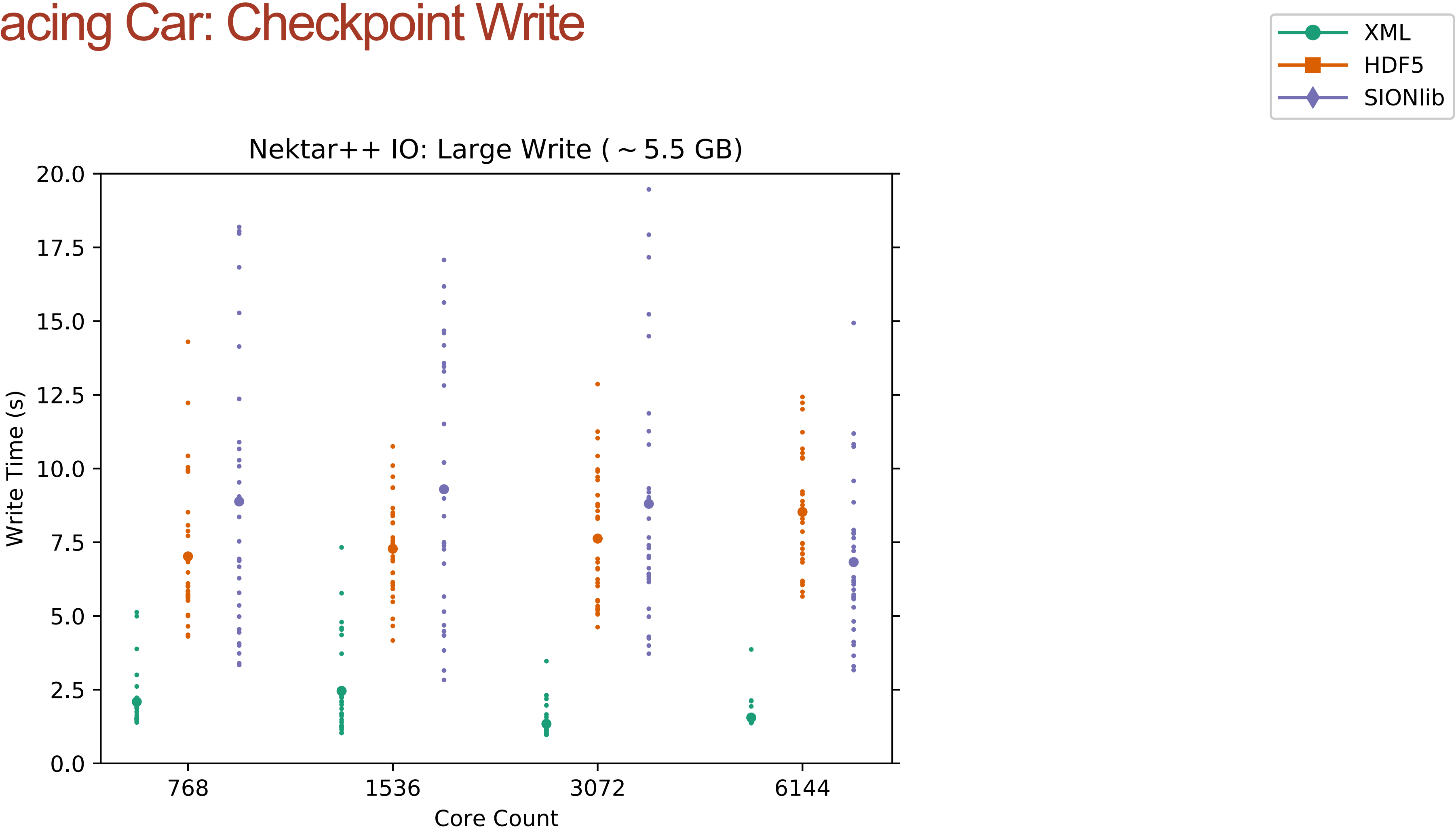
XML: 1 MiB, 1 OST

Stripe count set to -1 (use all available OSTs) for HDF5 and SIONlib.

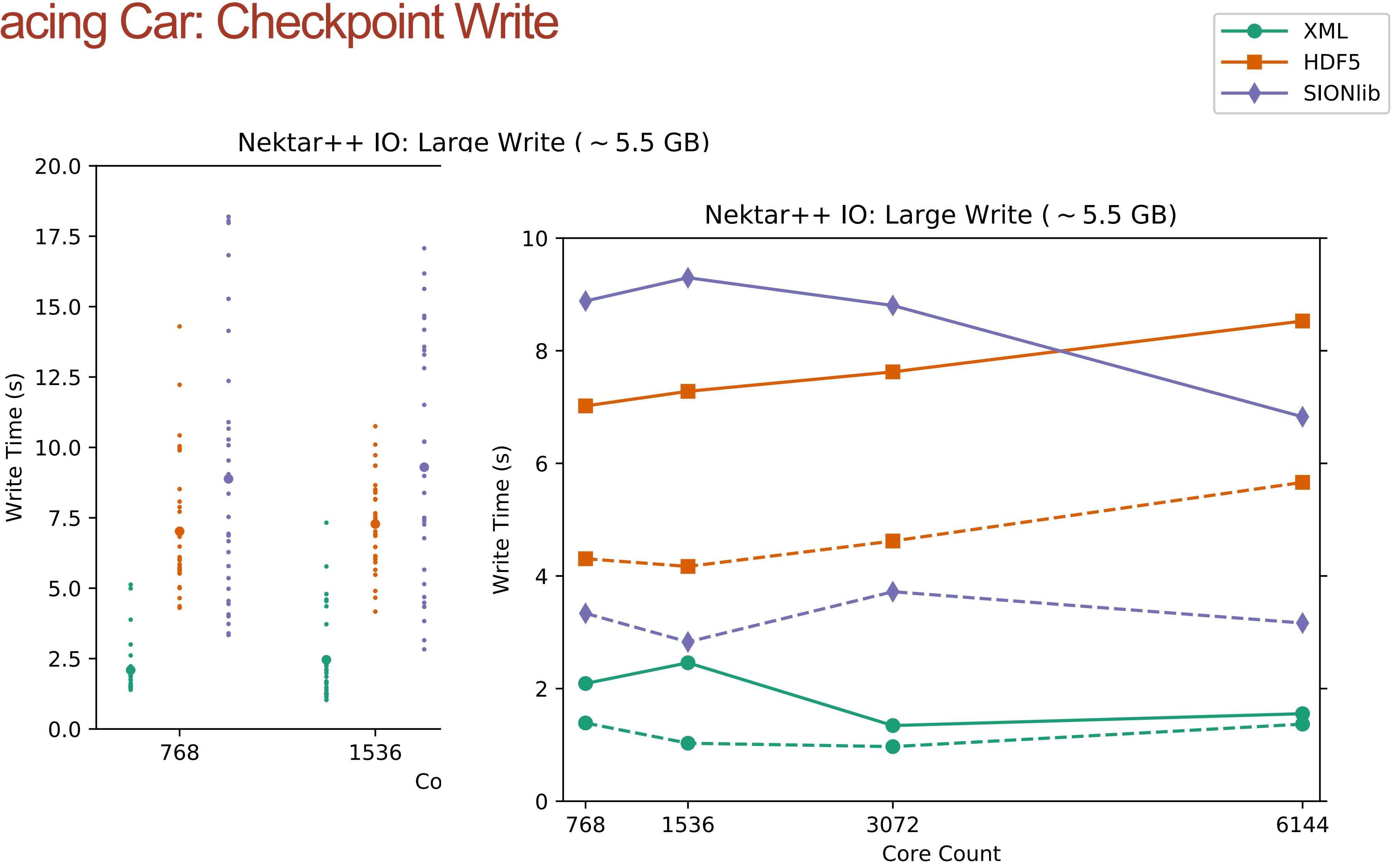
Racing Car: Checkpoint Read



Racing Car: Checkpoint Write



Racing Car: Checkpoint Write



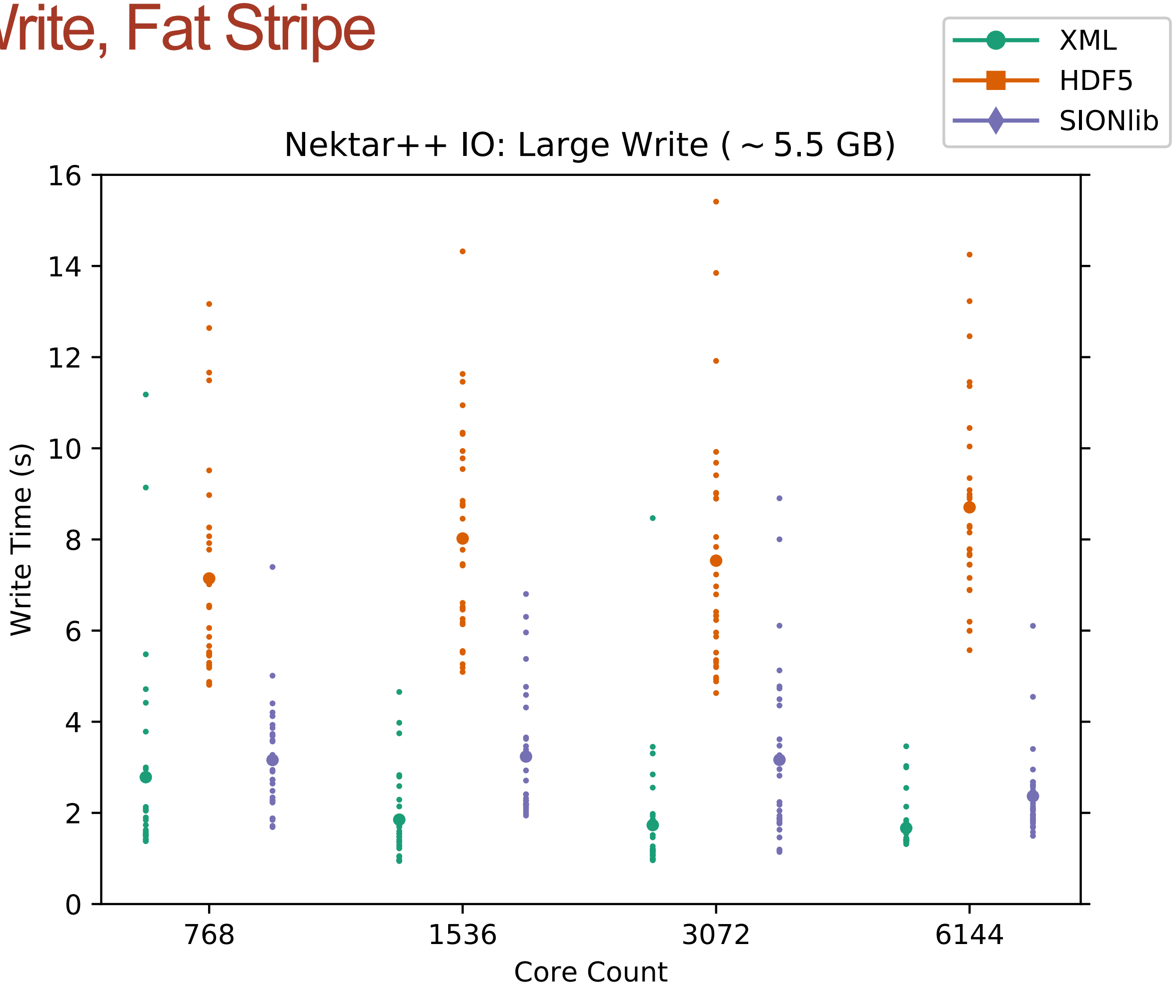
Racing Car: Checkpoint Write, Fat Stripe

Increase Lustre stripe size to a value equivalent to the amount of data handled by each HDF5 aggregator / SIONlib collector.

HDF5: 128 MiB

SIONlib: 256 MiB

All writers can write data within one stripe, i.e., each writer has a dedicated OST (in theory).



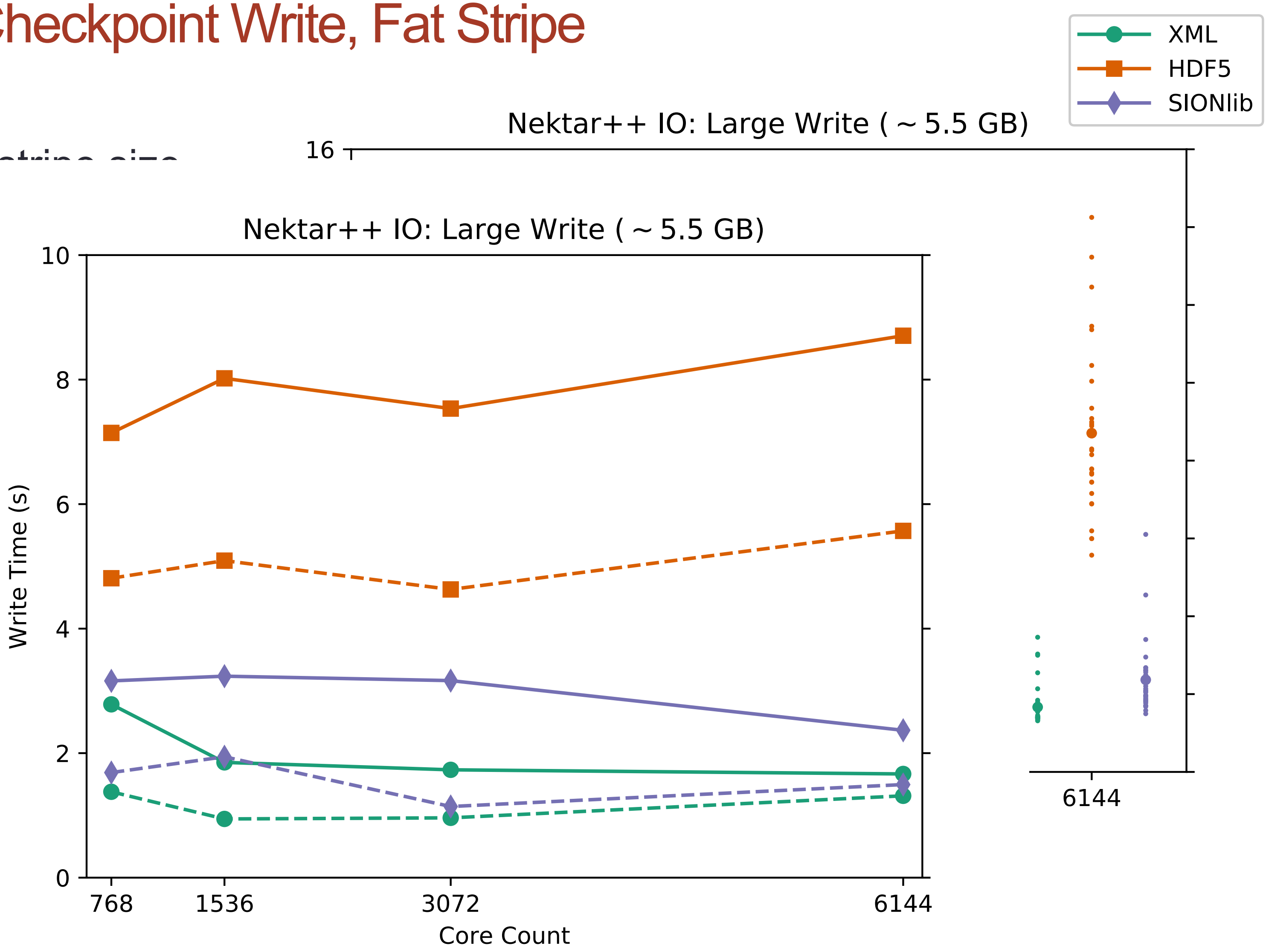
Racing Car: Checkpoint Write, Fat Stripe

Increase Lustr
to a value equ
amount of dat
each HDF5 ag
SIONlib collec

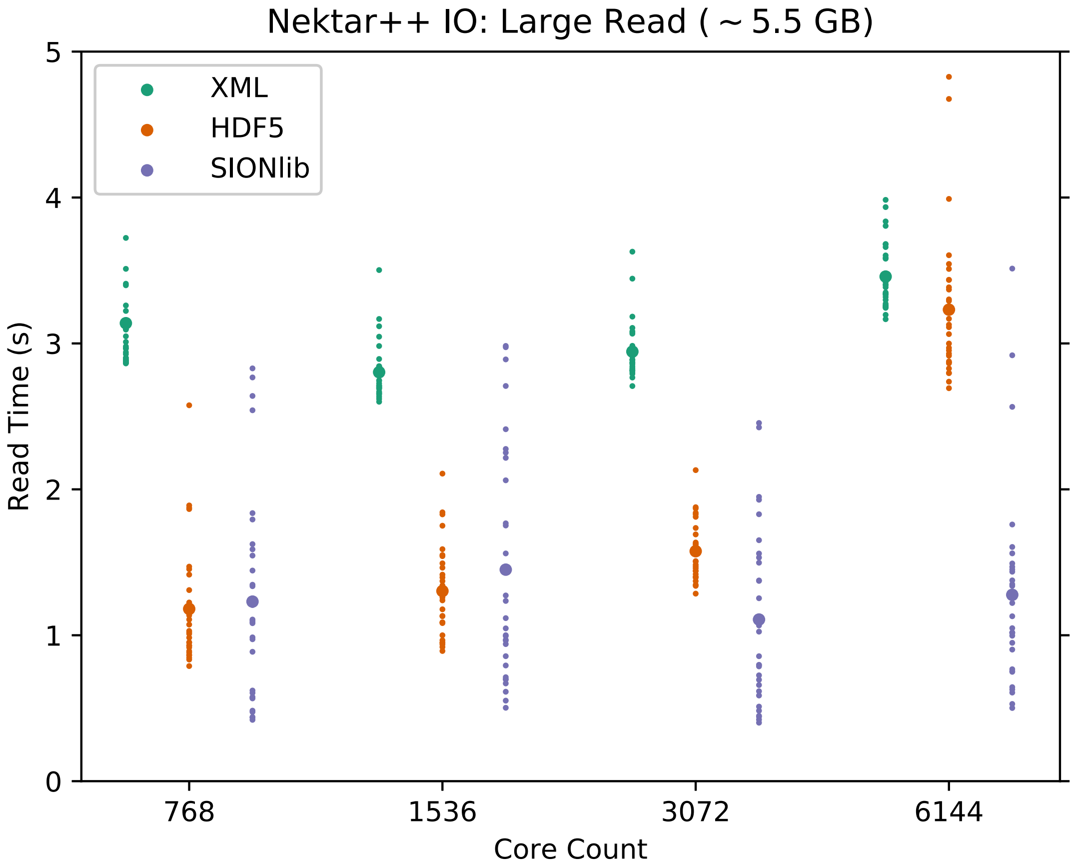
HD

SION

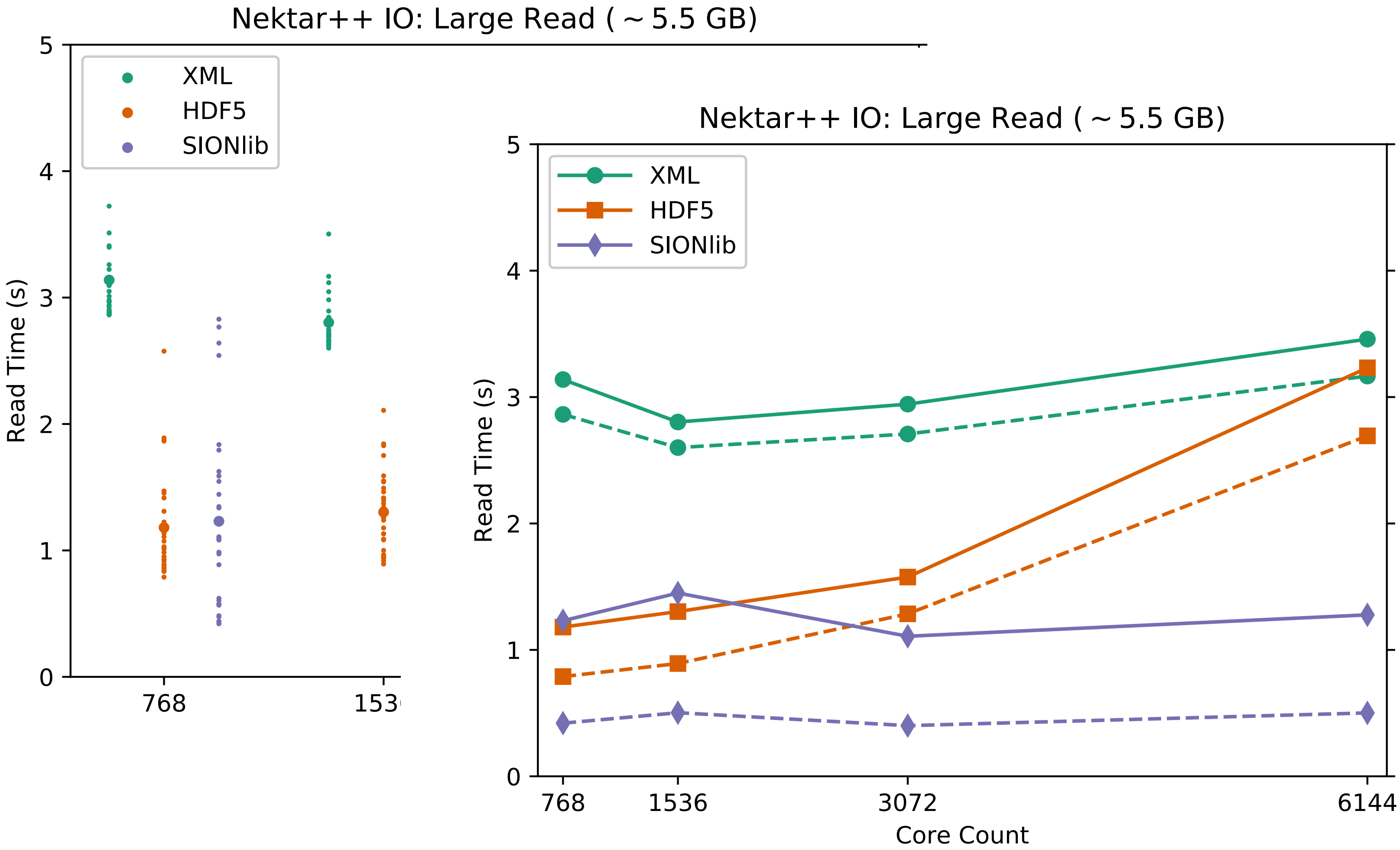
All writers car
within one stri
writer has a d



Racing Car: Checkpoint Read, Fat Stripe



Racing Car: Checkpoint Read, Fat Stripe



Results Summary (NB: all I/O times in context of Nektar++)

Small Aortic Arch Checkpoint File (2.5 MB)

Result sets contained significant scatter, necessary to detect and remove outliers (8 out of 720).

`FieldIOSIONlib` fastest at reading and writing.
`FieldIOXml` slowest at writing: 2.3 times SIONlib result.
`FieldIOHdf5` slowest at reading: 1.7 times SIONlib.

Using a stripe count of -1 worsened IO speeds for HDF5 and SIONlib.

Results Summary (NB: all I/O times in context of Nektar++)

Small Aortic Arch Checkpoint File (2.5 MB)

Result sets contained significant scatter, necessary to detect and remove outliers (8 out of 720).

`FieldIOSIONlib` fastest at reading and writing.
`FieldIOXml` slowest at writing: 2.3 times SIONlib result.
`FieldIOHdf5` slowest at reading: 1.7 times SIONlib.

Using a stripe count of -1 worsened IO speeds for HDF5 and SIONlib.

Large Racing Car Checkpoint File (5.5 GB)

No outliers, stripe size set to 128 MiB (HDF5) and 256 MiB (SIONlib). SIONlib write mode set to *collectivemerge*.

`FieldIOSIONlib` fastest at reading, but not writing.
Mean write times at 6144 cores, 1.6 s (XML), 2.4 s (SIONlib), 6.8 s (HDF5).
Minimum times, 1.3 s (XML) and 1.5 s (SIONlib).

Conclusions

SIONlib is the preferred choice for single-shared file as a result of two advantages, lower decompositional overhead and a greater responsiveness to Lustre file settings.

Expectation is that higher core counts (> 10^4) will slow XML compared to SIONlib.

Conclusions

SIONlib is the preferred choice for single-shared file as a result of two advantages, lower decompositional overhead and a greater responsiveness to Lustre file settings.

Expectation is that higher core counts (> 10^4) will slow XML compared to SIONlib.

Use of SIONlib *collectivemerge* mode means extra work required before simulation can be restarted from checkpoint file.

The original checkpoint writers must self-identify and then distribute the data stored in their area according to the original sender (MPI rank). Fortunately, this would also be a one-off cost.

Conclusions

SIONlib is the preferred choice for single-shared file as a result of two advantages, lower decompositional overhead and a greater responsiveness to Lustre file settings.

Expectation is that higher core counts (> 10^4) will slow XML compared to SIONlib.

Use of SIONlib *collectivemerge* mode means extra work required before simulation can be restarted from checkpoint file.

The original checkpoint writers must self-identify and then distribute the data stored in their area according to the original sender (MPI rank). Fortunately, this would also be a one-off cost.

However should be possible to improve `FieldIOHdf5` if one assumes static element partition (i.e., no load balancing), since the decomposition dataset would need to gathered just once.