# IMPROVING I/O PERFORMANCE IN NEKTAR++

Rupert Nash (rupert.nash@ed.ac.uk)
EPCC
The University of Edinburgh

Nektar++ Workshop, Imperial College, 7/7/15

epcc

THE UNIVERSITY OF EDINBURGH

# ARCHER eCSE programme

- Funded by EPSRC
- Run by EPCC as part of our national HPC service
- You can get 12 PM of effort from a research software engineer (not necessarily at EPCC, could be you/your PDRA)
- Enable you to do more science on ARCHER
- Lightweight proposal (~10 pages)
- ~60% success rate
- Call 6 closes on Tuesday 15th September, 2015
- Call 7 closes on Tuesday 19th January, 2016

epcc | THE UNIVERSITY OF EDINBURGH

# This talk

- Going to talk about:
- Current situation of I/O in master
- Two areas where I'm improving Nektar's I/O:
  - Field I/O
  - Mesh I/O

epcc

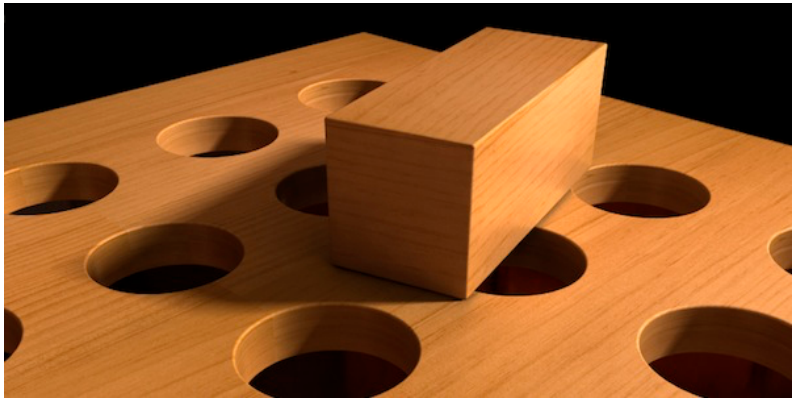
THE UNIVERSITY OF EDINBURGH

# Current state of I/O

- Entirely XML based
- XML is great!
  - ☺ self-describing
  - ☺ (quasi) human readable
  - ☺ widely used
- XML is horrible!
  - ☹ verbose
  - ☹ no random access
  - ☹ TinyXML* requires parsing whole document
  - ☹ only directly supports string data types
  - ☹ field data stored in a compressed, base64-encoded string

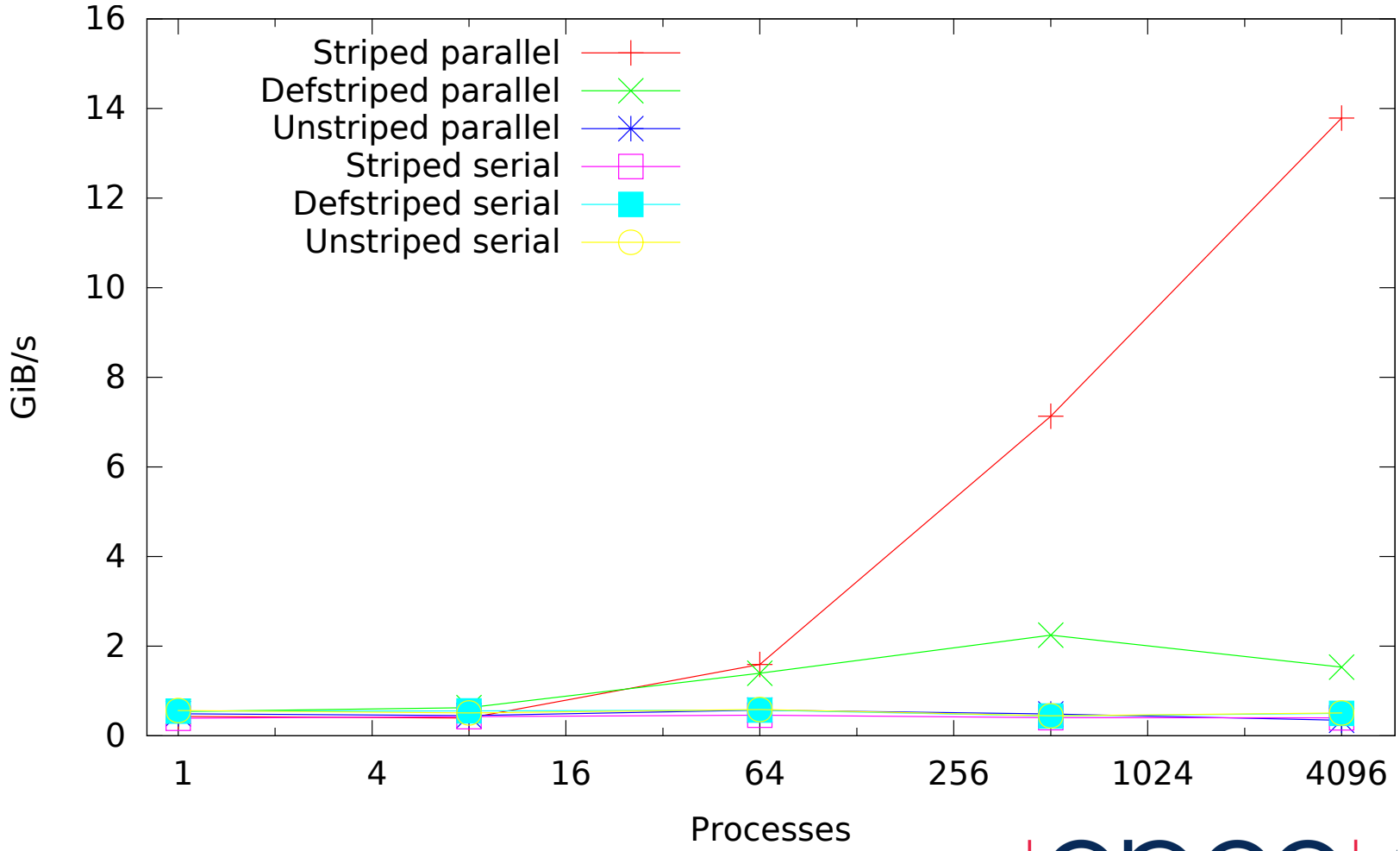
# Current state of I/O

- Mesh:
  - Read mesh on rank 0
  - Decompose
  - Write each partition to a new file
  - Each rank reads its partition
- Field output
  - Communicate IDs to rank 0
  - Rank 0 writes metadata
  - All ranks write data to own file



- This is a well-known ***anti-pattern*** for parallel I/O

# Archer write performance – Henty *et al.*

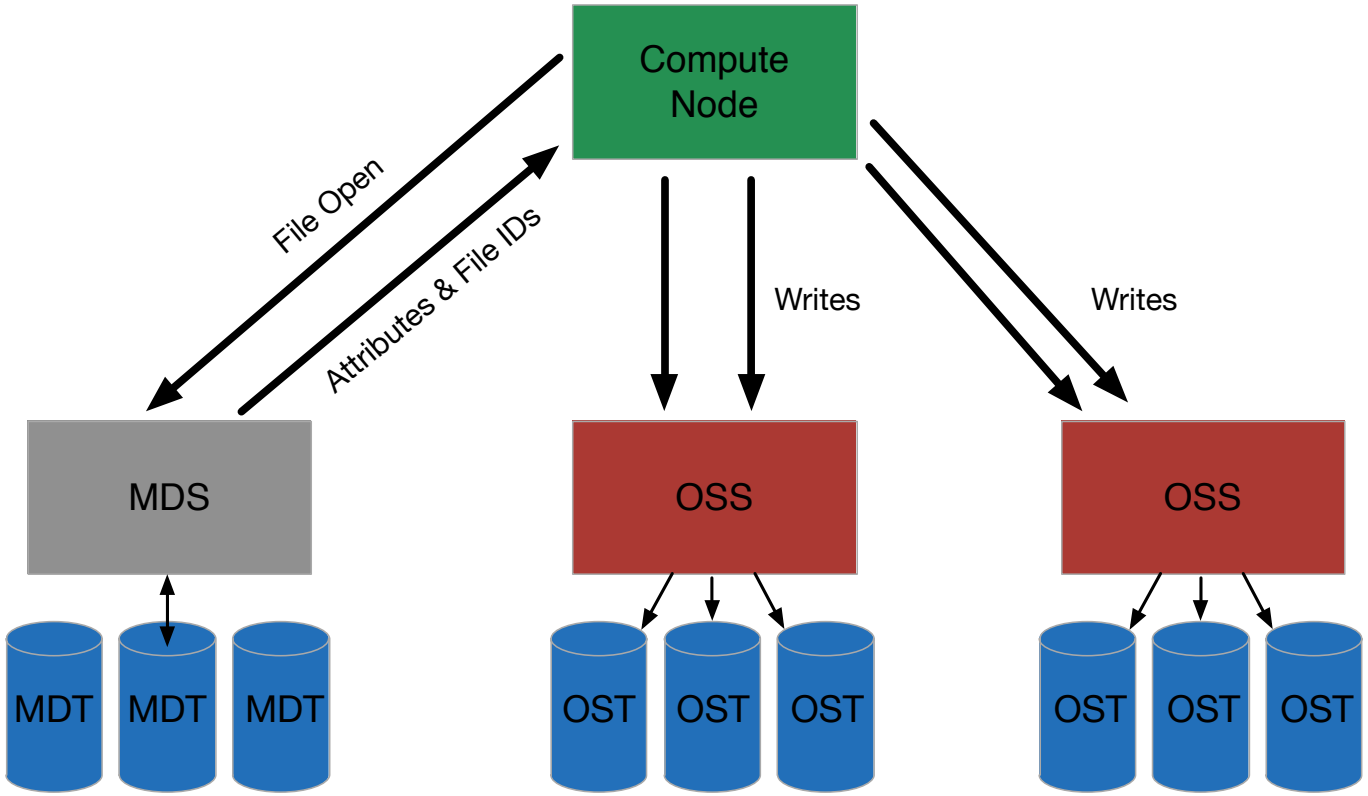



http://www.archer.ac.uk/documentation/white-papers/

epcc

THE UNIVERSITY OF EDINBURGH

# Why is collective IO faster?



# FieldIO

```
aorta> ls -lh aorta.fld/
total 143M
-rw------- 1 rnashnek ecse0213 186K Jun 17 11:24 Info.xml
-rw------- 1 rnashnek ecse0213 6.0M Jun 17 11:24 P0000000.fld
-rw------- 1 rnashnek ecse0213 5.5M Jun 17 11:24 P0000001.fld
-rw------- 1 rnashnek ecse0213 5.8M Jun 17 11:24 P0000002.fld
-rw------- 1 rnashnek ecse0213 5.6M Jun 17 11:24 P0000003.fld
-rw------- 1 rnashnek ecse0213 5.7M Jun 17 11:24 P0000004.fld
-rw------- 1 rnashnek ecse0213 5.6M Jun 17 11:24 P0000005.fld
<SNIP />
-rw------- 1 rnashnek ecse0213 6.1M Jun 17 11:24 P0000023.fld
```

# FieldIO – Info.xml

```
aorta> cat -lh aorta.fld/Info.xml
<?xml version="1.0" encoding="utf-8" ?>
<NEKTAR>
    <Metadata>
        <Provenance> <SNIP /> </Provenance>
        <Kinvis>0.0033333333333335</Kinvis>
        <Time>0</Time>
        <TimeStep>0.00050000000000001</TimeStep>
    </Metadata>
    <Partition FileName="P0000000.fld”>
        Long list of element IDs in the file
    </Partition>
    <ETC />
</NEKTAR>
```

epcc

THE UNIVERSITY OF EDINBURGH

# FieldIO – per-rank file

```
aorta> cat -lh aorta.fld/P0000000.fld
<?xml version="1.0" encoding="utf-8" ?>
<NEKTAR>
    <Metadata></Metadata>
    <ELEMENTS FIELDS="u,v,w,p" SHAPE="Tetrahedron"
              BASIS="Modified_A,Modified_B,Modified_C"
              NUMMODESPERDIR="UNIORDER:5,5,5"
              ID="LONG LIST OF IDS">
        base64EncodedDoubles9eJwsm3cg1e8Xx+0VsgnZe...
    </ELEMENTS>
    <ETC />
</NEKTAR>
```

epcc

THE UNIVERSITY OF EDINBURGH

# Alternative to XML

- HDF5 - Hierarchical Data Format www.hdfgroup.org

| | XML | HDF |
|---|---|---|
| Self describing | Yes | Yes |
| Human-readable | Yes… | No |
| Widely used | Yes | In HPC… |
| Random access | No | Yes |
| Binary data | No | Yes |
| Parallel IO | No | Yes |

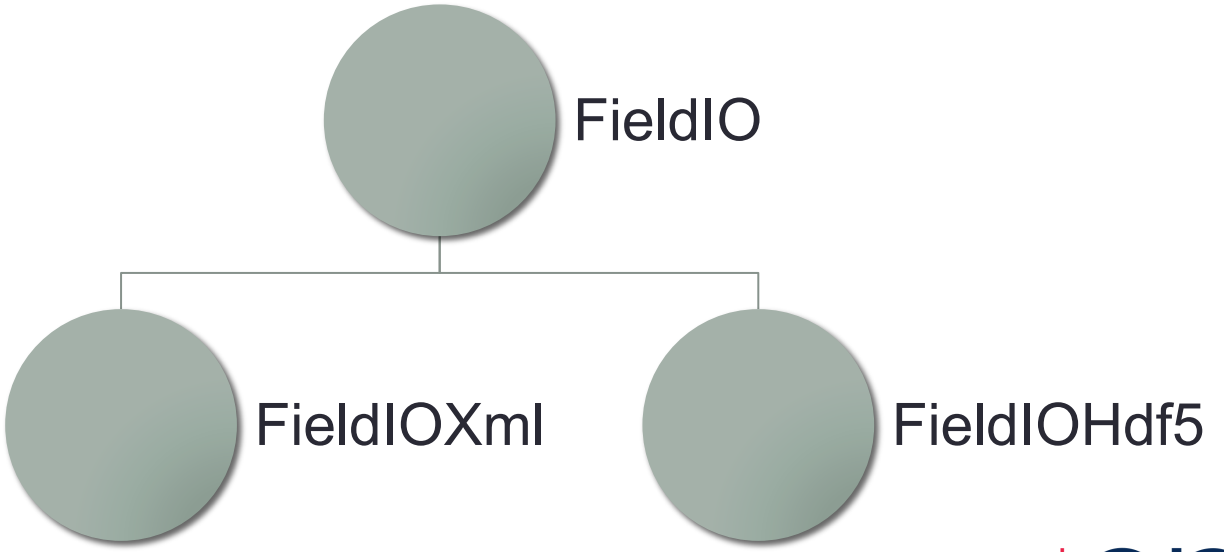
# New format – step 1

- Exactly the same structure (for now):

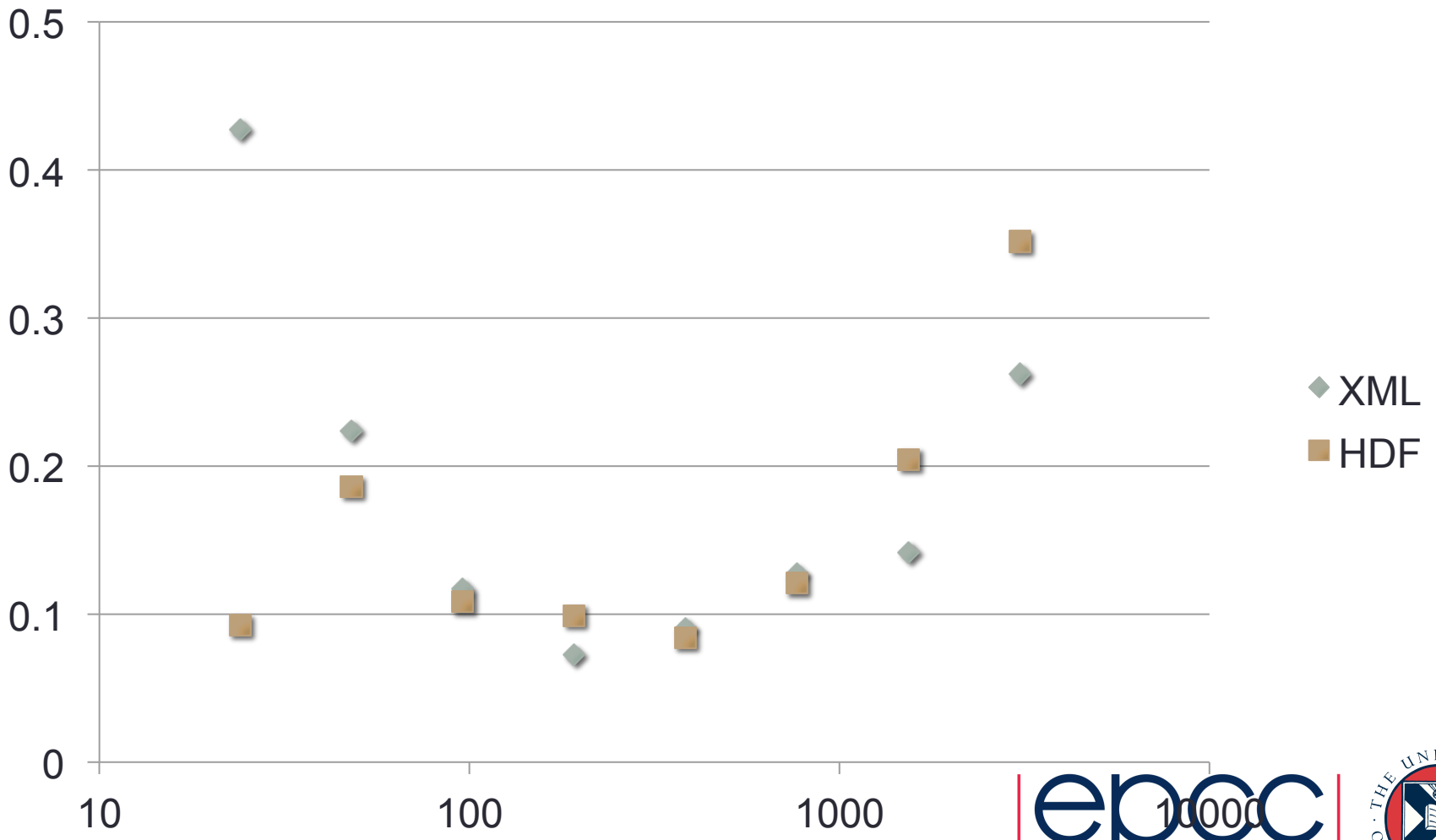

But format of per-process file is HDF

# Implementation

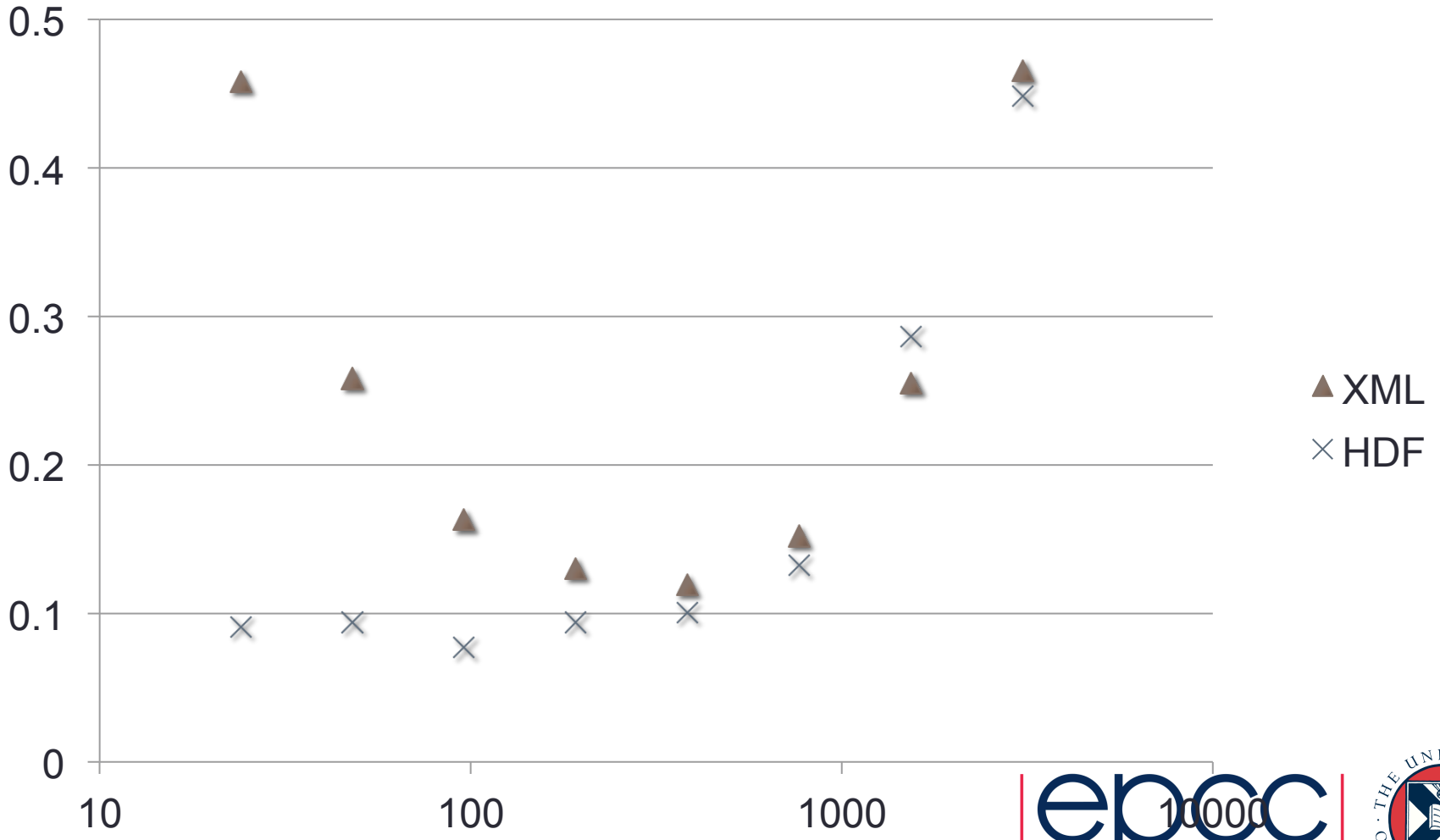
- Standard Factory pattern
- Creates the necessary subclass of FieldIO
- Tried to share as much code as possible



# Write performance



# Read performance



# How to try it out

- `> git checkout feature/hdf5`
- Configure with NEKTAR_USE_HDF5=ON
- `> make -j $NCORES install`
- Make tea…

# How to try it out – default solvers

- To write HDF, in your `conditions.xml` add:

```
<NEKTAR>
    <CONDITIONS>
        <SOLVERINFO>
            <I PROPERTY="FieldIO_Format" VALUE="Hdf5" />
        </SOLVERINFO>
    </CONDTIONS>
</NEKTAR>
```

- No changes needed to read HDF

# How to try it out – custom solvers

- Construct FieldIO objects using one of two factory methods

- Output:
  - `LibUtilities::MakeDefaultFieldIO(session)`
  - (Uses the property from previous slide)
- Input:
  - `LibUtilities::MakeFieldIOForFile(session, filename);`
  - (It will figure out what file type you've given it)

# Plan for FieldIO

- Add collective IO (i.e. all ranks write to the same file)
- Aim to get this done by mid Aug (I've done the groundwork)
- Improve FieldConvert performance by extracting elements of interest only
- Write some regression tests

# Mesh IO

```
<GEOMETRY DIM="3" SPACE="3">
  <VERTEX>
    <V ID="0">1.16423749e+01 3.93585456e+00 8.39724408e+00</V>
  </VERTEX>
  <EDGE>
    <E ID="0">    0  1   </E>
  </EDGE>
  <FACE>
    <Q ID="0">         0         1         2         3</Q>
    <T ID="1">         0         5         4</T>
  </FACE>
  <ELEMENT>
    <R ID="0">    0     1     2     3     4 </R>
  </ELEMENT>
  <CURVED>
    <E ID="0" EDGEID="15274" NUMPOINTS="7" TYPE="GaussLobattoLegendre"> 1.15975286e+01  ... </E>
  </CURVED>
  <COMPOSITE>
    <C ID="0"> R[0-21563] </C>
  </COMPOSITE>
  <DOMAIN> C[0,1] </DOMAIN>
</GEOMETRY>
```

epcc

THE UNIVERSITY OF EDINBURGH

# File sizes

| | Count | Min size | XML size |
|---|---|---|---|
| Vertices | 24,095 | 500 kB | 1.6 MB |
| Edges | 108,684 | 900 kB | 4.7MB |
| Faces | 147,032 | 1.9 MB | 8.9 MB |
| Elements | 62,441 | 1 MB | 3.5 MB |

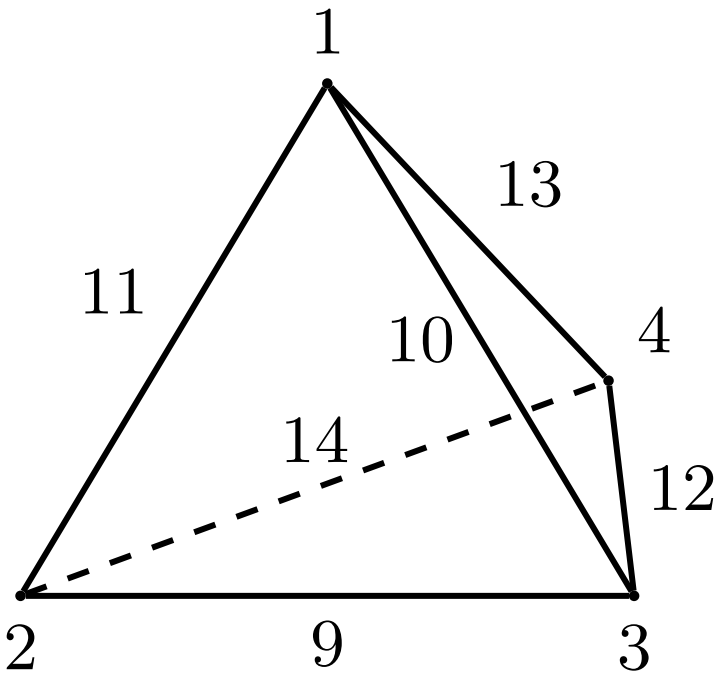
XML is flexible but verbose

# Loading time

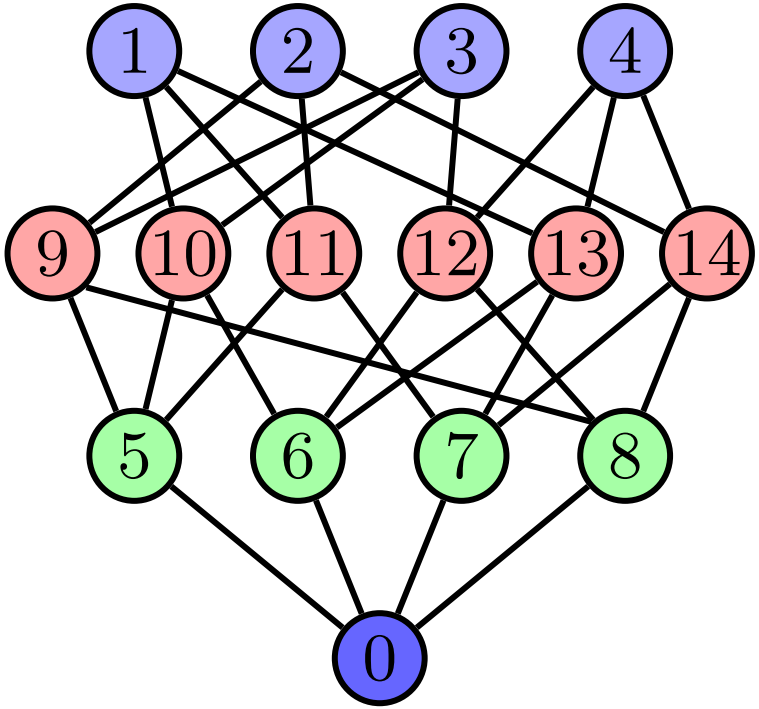
| Cores | Time / s | Cost |
|---|---|---|
| 24 | 22 | < 1p |
| 192 | Still queuing! | |
| 1536 | 249 | 106 core-hours / 90 p |

Aorta dataset, 100 k elements, 4 fields (u, v, w, p)

# Some maths – a Hasse diagramme




Vertex and edge numbering




Topological connectivity

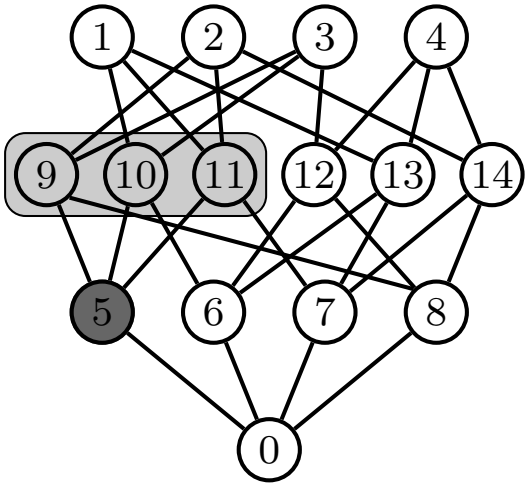
Credit: Michael Lange *et al*, arXiv:1505.04633v1

epcc

THE UNIVERSITY OF EDINBURGH

# Some maths – a Hasse diagramme

- Doesn't care about the dimension
- Can represent any mesh




(c) $cone(5) = \{9, 10, 11\}$

Cone (X) = those objects that directly make up X

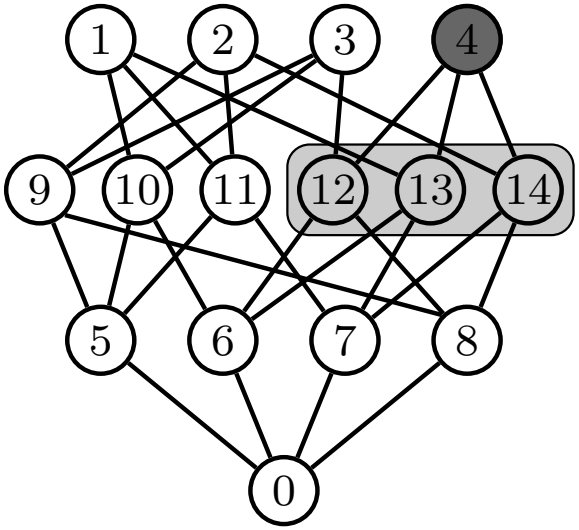
Credit: Michael Lange *et al*, arXiv:1505.04633v1

epcc

THE UNIVERSITY OF EDINBURGH

# Some maths – a Hasse diagramme

- Doesn't care about the dimension
- Can represent any mesh




(e) $support(4) = \{12, 13, 14\}$

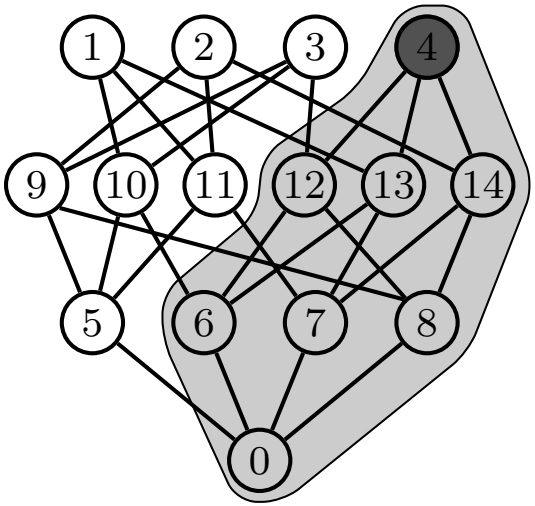
Support (X) = those objects that directly use me X

Credit: Michael Lange *et al*, arXiv:1505.04633v1

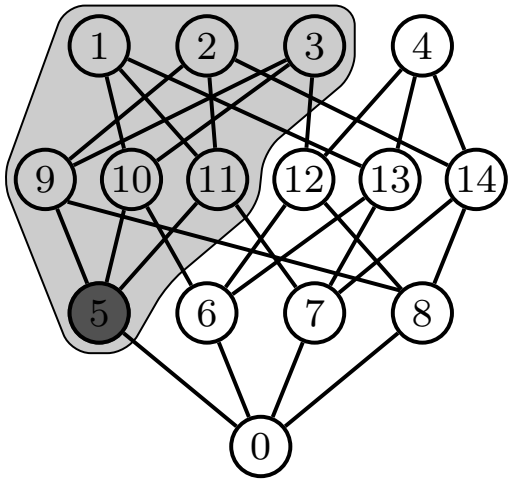
epcc

THE UNIVERSITY OF EDINBURGH

# Some maths – a Hasse diagramme




(f) $star(4) = \{0, 6, 7, 8, 12, 13, 14\}$




(d) $closure(5) = \{1, 2, 3, 9, 10, 11\}$

Credit: Michael Lange *et al*, arXiv:1505.04633v1

epcc

THE UNIVERSITY OF EDINBURGH

# Why do we care?

- Potentially useful for geometry-based preconditioners
  - (Get me all the cells that share a face with cell X. Or share an edge, etc.)
- Meshing maybe?
- **Can push the burden of maintenance onto a library**

# Library for this - PETSc

- PETSc has a sub-library for dealing with these objects, DMPlex.
- PETSc is very widely used
- Slightly impenetrable terminology and code-as-documentation, but that improving.
- Under active development (M Lange @ Imperial, M Knepley @ U Chicago)
- Can attach arbitrary data to any subset of entities, e.g.
  - Coordinates to vertices
  - Curvature data to edges

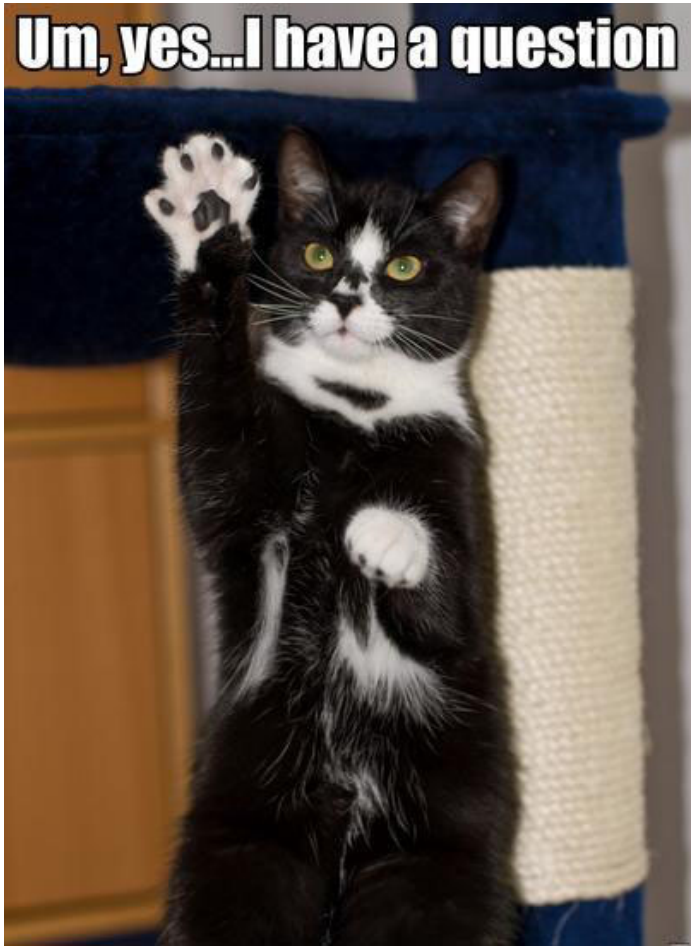
# State in Nektar++

- Added an output module to MeshConvert
- Unfortunately DMPlex's serialisation methods do not support hybrid meshes, due to over-conservative error-checking – the developers have mostly fixed this.
- A petsc-dev branch now supports output of these to HDF5 (thanks to Michael Lange)

# Future of DMPlex+Nektar++

- Retry MeshConvert with updated PETSc
- Add new mesh reading class that uses current approach but DMPlex/HDF format
- Use PETSc partitioning routines?

# Thank you




epcc

THE UNIVERSITY OF EDINBURGH