Automatic Generation of I/O Kernels for HPC Applications

Babak Behzad[1], Hoang-Vu Dang[1], Farah Hariri[1],
Weizhe Zhang[2], Marc Snir[1,3]

[1]University of Illinois at Urbana-Champaign,
[2]Harbin Institue of Technology,
[3]Argonne National Laboratory

Data-driven Science

- Modern scientific discoveries driven by massive data
- Stored as files on disks managed by parallel file systems
- Parallel I/O: Determining performance factor of modern HPC
  - ◇ HPC applications working with very large datasets
  - ◇ Both for checkpointing and input and output

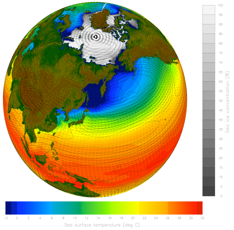


Figure 1: NCAR's CESM Visualization

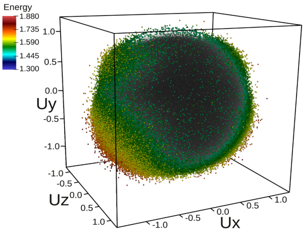


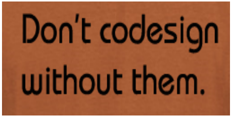
Figure 2: 1 trillion-electron VPIC dataset

Motivation: I/O Kernels

- An I/O kernel is a miniature application generating the same I/O calls as a full HPC application
- I/O kernels have been used for in the I/O community for a long time. But they are:
  - ◇ hard to create
  - ◇ outdated soon
  - ◇ not enough
- Why do we use I/O Kernels?
  - ◇ Better I/O performance analysis and optimization
  - ◇ I/O autotuning
  - ◇ Storage system evaluation
  - ◇ Ease of collaboration

Generating I/O kernels automatically

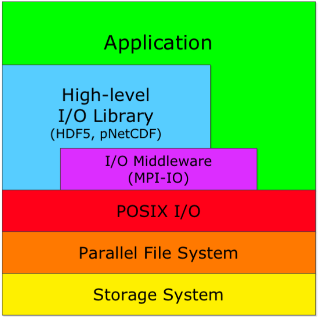
- Derive I/O kernels of HPC applications automatically without accessing the source code
  - If possible, will always have latest version of I/O kernels
  - I/O complement to the HPC applications co-design effort i.e. miniapps such as Mantevo project



- Challenges in generating I/O kernels of HPC applications automatically
  - Large I/O trace files
  - How to merge traces in large-scale?
  - How to generate correct code out of the I/O traces?

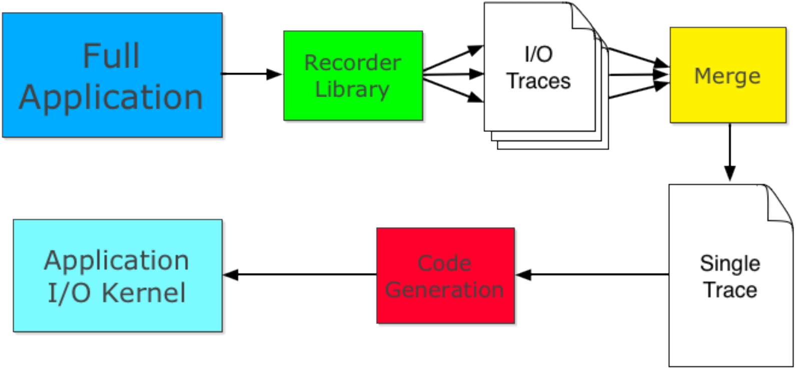
I/O Stack

- High-level I/O Library: Match storage abstraction to domain
- I/O Middleware: Match the programming model (MPI), a more generic interface
- POSIX I/O: Match the storage hardware, presents a single view

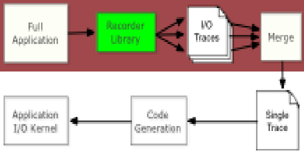


Our Approach

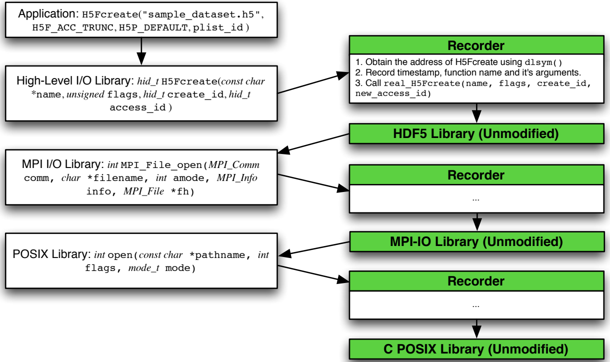
- Trace the I/O operations at different levels using Recorder
  - Gather p I/O trace files generated by p processes running the application
- Merge these p trace files into a single I/O trace file
- Generate parallel I/O code for this merged I/O trace



Recorder



- A multi-level tracing library developed to understand the I/O behavior of applications
- Does not need to change anything in the source code, just link



- It captures traces in multiple libraries
  1. HDF5 → We envision the actual trace and replay

pH5Example traced by the Recorder

- Figure below shows an example of a trace file generated using our Recorder only at HDF5 level.
- From a parallel HDF5 example application called `pH5Example`, distributed with the HDF5 source code.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

pH5Example traced by the Recorder

1. This application creates a file using `H5Fcreate()` function;
2. A dataspace of size 24×24 is built.
3. Two datasets are created based on this dataspace.
4. Each MPI rank selects a hyperslab of these datasets by giving the start, stride, and count array.
5. Data are being written to these two datasets.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

pH5Example traced by the Recorder

1. This application creates a file using `H5Fcreate()` function;
2. A dataspace of size 24×24 is built.
3. Two datasets are created based on this dataspace.
4. Each MPI rank selects a hyperslab of these datasets by giving the start, stride, and count array.
5. Data are being written to these two datasets.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

pH5Example traced by the Recorder

1. This application creates a file using `H5Fcreate()` function;
2. A dataspace of size 24×24 is built.
3. Two datasets are created based on this dataspace.
4. Each MPI rank selects a hyperslab of these datasets by giving the start, stride, and count array.
5. Data are being written to these two datasets.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

pH5Example traced by the Recorder

1. This application creates a file using `H5Fcreate()` function;
2. A dataspace of size 24×24 is built.
3. Two datasets are created based on this dataspace.
4. Each MPI rank selects a hyperslab of these datasets by giving the start, stride, and count array.
5. Data are being written to these two datasets.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

pH5Example traced by the Recorder

1. This application creates a file using `H5Fcreate()` function;
2. A dataspace of size 24×24 is built.
3. Two datasets are created based on this dataspace.
4. Each MPI rank selects a hyperslab of these datasets by giving the start, stride, and count array.
5. Data are being written to these two datasets.

```
1396296304.23583 H5Pcreate (H5P_FILE_ACCESS) 167772177 0.00003
1396296304.23587 H5Pset_fapl_mpio (167772177,MPI_COMM_WORLD,469762048) 0 0.00025
1396296304.23613 H5Fcreate (output/ParaEg0.h5,2,0,167772177) 16777216 0.00069
1396296304.23683 H5Pclose (167772177) 0 0.00002
1396296304.23685 H5Screate_simple (2,{24;24},NULL) 67108866 0.00002
1396296304.23688 H5Dcreate2 (16777216,Data1,H5T_STD_I32LE,67108866,0,0,0) 83886080 0.00012
1396296304.23702 H5Dcreate2 (16777216,Data2,H5T_STD_I32LE,67108866,0,0,0) 83886081 0.00003
1396296304.23707 H5Dget_space (83886080) 67108867 0.00001
1396296304.23708 H5Sselect_hyperslab (67108867,0,{0;0},{1;1},{6;24},NULL) 0 0.00002
1396296304.23710 H5Screate_simple (2,{6;24},NULL) 67108868 0.00001
1396296304.23710 H5Dwrite (83886080,50331660,67108868,67108867,0) 0 0.00009
1396296304.23721 H5Dwrite (83886081,50331660,67108868,67108867,0) 0 0.00002
1396296304.23724 H5Sclose (67108867) 0 0.00000
1396296304.23724 H5Dclose (83886080) 0 0.00001
1396296304.23726 H5Dclose (83886081) 0 0.00001
1396296304.23727 H5Sclose (67108866) 0 0.00000
1396296304.23728 H5Fclose (16777216) 0 0.00043
```

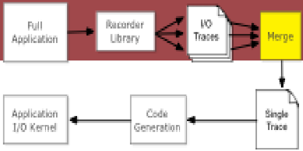
Trace files differences across ranks

- Trace lines are mostly the same on different MPI ranks except for the following difference:

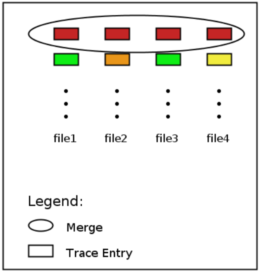
Rank 0:
```
1396296304.23708 H5Sselect_hyperslab (67108867,H5S_SELECT_SET,{0;0},{1;1},{6;24},NULL) 0 0.00002
```
Rank 1:
```
1396296304.23716 H5Sselect_hyperslab (67108867,H5S_SELECT_SET,{6;0},{1;1},{6;24},NULL) 0 0.00001
```
Rank 2:
```
1396296304.23714 H5Sselect_hyperslab (67108867,H5S_SELECT_SET,{12;0},{1;1},{6;24},NULL) 0 0.00002
```
Rank 3:
```
1396296304.23708 H5Sselect_hyperslab (67108867,H5S_SELECT_SET,{18;0},{1;1},{6;24},NULL) 0 0.00002
```

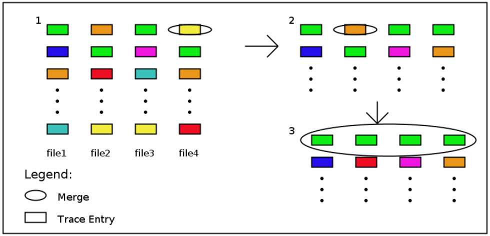
- Function signature:
  herr_t `H5Sselect_hyperslab(`*hid_t* `space_id,` *H5S_seloper_t* `op,` *const hsize_t* `*start,` *const hsize_t* `*stride,` *const hsize_t* `*count,` *const hsize_t* `*block)`

Merging



- This tool merges these traces into 1 merged trace
- Works at HDF5 level
- There may be different scenarios happening at the merging process:

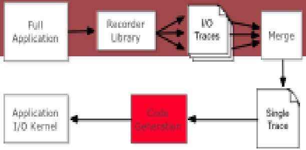




Merging pH5Example traces generated by the Recorder

```
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Pcreate, argc=1, args=[H5P_FILE_ACCESS], R=[ 167772177 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Pset_fapl_mpio, argc=3, args=[167772177,MPI_COMM_WORLD,469762048], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Fcreate, argc=4, args=[output/ParaEg0.h5,2,0,167772177], R=[ 16777216 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Pclose, argc=1, args=[167772177], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Screate_simple, argc=3, args=[2,{24;24},NULL], R=[ 67108866 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dcreate2, argc=7, args=[16777216,Data1,H5T_STD_I32LE,67108866,0,0,0], R=[ 83886080 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dcreate2, argc=7, args=[16777216,Data2,H5T_STD_I32LE,67108866,0,0,0], R=[ 83886081 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dget_space, argc=1, args=[83886080], R=[ 67108867 ], }
{ same=4:  diffarg=1 diffret=0 }
{ file=0, func=H5Sselect_hyperslab, argc=6, args=[67108867,0,{0;0},{1;1},{6;24},NULL], R=[ 0 ], }
{ file=1, func=H5Sselect_hyperslab, argc=6, args=[67108867,0,{6;0},{1;1},{6;24},NULL], R=[ 0 ], }
{ file=2, func=H5Sselect_hyperslab, argc=6, args=[67108867,0,{12;0},{1;1},{6;24},NULL], R=[ 0 ], }
{ file=3, func=H5Sselect_hyperslab, argc=6, args=[67108867,0,{18;0},{1;1},{6;24},NULL], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Screate_simple, argc=3, args=[2,{6;24},NULL], R=[ 67108868 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dwrite, argc=5, args=[83886080,50331660,67108868,67108867,0], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dwrite, argc=5, args=[83886081,50331660,67108868,67108867,0], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Sclose, argc=1, args=[67108867], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dclose, argc=1, args=[83886080], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Dclose, argc=1, args=[83886081], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Sclose, argc=1, args=[67108866], R=[ 0 ], }
{ same=4:  diffarg=0 diffret=0 }
{ file=0, func=H5Fclose, argc=1, args=[16777216], R=[ 0 ], }
```

Code Generation



- Once one merged trace is generated, we can generate an SPMD MPI code for it
- Buffers are allocated with the same size read from the trace file. Their data is randomly generated.
- HDF5 is easier to generate code for, because every object has an integer identifier
  - ◇ Therefore a map is used to map HDF5 ids in the merged trace to generated variable names

pH5Example Code Generation

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include "hdf5.h"

int main(int argc, char* argv[])
{
   int mpi_rank, mpi_size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);

   hid_t hid_0 = H5Pcreate(H5P_FILE_ACCESS);
   MPI_Comm comm_1 = MPI_COMM_WORLD;
   MPI_Info info_1;
   MPI_Info_create(&info_1);
   H5Pset_fapl_mpio(hid_0, comm_1, info_1);
   hid_t hid_2 = H5Fcreate("output/ParaEg0.h5", H5F_ACC_TRUNC, H5P_DEFAULT, hid_0);
   H5Pclose(hid_0);
   hsize_t cur_dims_0[] = {24,24};
   hid_t hid_3 = H5Screate_simple(2, cur_dims_0, NULL);
   hid_t hid_4 = H5Dcreate2(hid_2, "Data1", H5T_STD_I32LE, hid_3, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
   hid_t hid_5 = H5Dcreate2(hid_2, "Data2", H5T_STD_I32LE, hid_3, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
   hid_t hid_6 = H5Dget_space(hid_4);
   hsize_t start_1[2];
   start_1[0] = 6 * mpi_rank + 0;
   start_1[1] = 0 * mpi_rank + 0;
   hsize_t stride_1[] =  {1,1};
   hsize_t count_1[] =  {6,24};
   H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
   hsize_t cur_dims_2[] = {6,24};
   hid_t hid_7 = H5Screate_simple(2, cur_dims_2, NULL);
   hssize_t npoints_3 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_4 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_0 = npoints_3 * size_dtype_4;
   void *dummy_data_1 = (void *) malloc(total_size_0);
   H5Dwrite(hid_4, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_1);
   hssize_t npoints_5 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_6 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_2 = npoints_5 * size_dtype_6;
   void *dummy_data_3 = (void *) malloc(total_size_2);
   H5Dwrite(hid_5, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_3);
   H5Sclose(hid_6);
   H5Dclose(hid_4);
   H5Dclose(hid_5);
```

pH5Example Code Generation

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include "hdf5.h"

int main(int argc, char* argv[])
{
   int mpi_rank, mpi_size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
```

```
    hid_t hid_0 = H5Pcreate(H5P_FILE_ACCESS);
    MPI_Comm comm_1 = MPI_COMM_WORLD;
    MPI_Info info_1;
    MPI_Info_create(&info_1);
    H5Pset_fapl_mpio(hid_0, comm_1, info_1);
    hid_t hid_2 = H5Fcreate("output/ParaEg0.h5", H5F_ACC_TRUNC,
H5P_DEFAULT, hid_0);
    H5Pclose(hid_0);
```

```
H5P_DEFAULT, H5P_DEFAULT);
   hid_t hid_6 = H5Dget_space(hid_4);
   hsize_t start_1[2];
   start_1[0] = 6 * mpi_rank + 0;
   start_1[1] = 0 * mpi_rank + 0;
   hsize_t stride_1[] =  {1,1};
   hsize_t count_1[] =  {6,24};
   H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
   hsize_t cur_dims_2[] = {6,24};
   hid_t hid_7 = H5Screate_simple(2, cur_dims_2, NULL);
   hssize_t npoints_3 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_4 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_0 = npoints_3 * size_dtype_4;
   void *dummy_data_1 = (void *) malloc(total_size_0);
   H5Dwrite(hid_4, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_1);
   hssize_t npoints_5 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_6 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_2 = npoints_5 * size_dtype_6;
   void *dummy_data_3 = (void *) malloc(total_size_2);
   H5Dwrite(hid_5, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_3);
   H5Sclose(hid_6);
   H5Dclose(hid_4);
   H5Dclose(hid_5);
```

pH5Example Code Generation

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include "hdf5.h"

int main(int argc, char* argv[])
{
   int mpi_rank, mpi_size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
```

```
   hid_t hid_0 = H5Pcreate(H5P_FILE_ACCESS);
   MPI_Comm comm_1 = MPI_COMM_WORLD;
H5P_
```

```
    hsize_t cur_dims_0[] = {24,24};
    hid_t hid_3 = H5Screate_simple(2, cur_dims_0, NULL);
    hid_t hid_4 = H5Dcreate2(hid_2, "Data1", H5T_STD_I32LE, hid_3,
H5P_DEFAULT,                               H5P_DEFAULT, H5P_DEFAULT);
    hid_t hid_5 = H5Dcreate2(hid_2, "Data2", H5T_STD_I32LE, hid_3,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
    hid_t hid_6 = H5Dget_space(hid_4);
```

```
   start_1[0] = 6 * mpi_rank + 0;
   start_1[1] = 0 * mpi_rank + 0;
   hsize_t stride_1[] =  {1,1};
   hsize_t count_1[] =  {6,24};
   H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
   hsize_t cur_dims_2[] = {6,24};
   hid_t hid_7 = H5Screate_simple(2, cur_dims_2, NULL);
   hssize_t npoints_3 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_4 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_0 = npoints_3 * size_dtype_4;
   void *dummy_data_1 = (void *) malloc(total_size_0);
   H5Dwrite(hid_4, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_1);
   hssize_t npoints_5 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_6 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_2 = npoints_5 * size_dtype_6;
   void *dummy_data_3 = (void *) malloc(total_size_2);
   H5Dwrite(hid_5, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_3);
   H5Sclose(hid_6);
   H5Dclose(hid_4);
   H5Dclose(hid_5);
```

pH5Example Code Generation

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include "hdf5.h"

int main(int argc, char* argv[])
{
   int mpi_rank, mpi_size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
```

```c
   hid_t hid_0 = H5Pcreate(H5P_FILE_ACCESS);
   MPI_Comm comm_1 = MPI_COMM_WORLD;
```

```c
   hsize_t cur_dims_0[] = {24,24};
```

```c
   hid_t hid_7 = H5Screate_simple(2, cur_dims_2, NULL);
   hssize_t npoints_3 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_4 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_0 = npoints_3 * size_dtype_4;
   void *dummy_data_1 = (void *) malloc(total_size_0);
   H5Dwrite(hid_4, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT,
dummy_data_1);
```

```c
   hsize_t count_1[] =  {6,24};
   H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
   hsize_t cur_dims_2[] = {6,24};
   hid_t hid_7 = H5Screate_simple(2, cur_dims_2, NULL);
   hssize_t npoints_3 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_4 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_0 = npoints_3 * size_dtype_4;
   void *dummy_data_1 = (void *) malloc(total_size_0);
   H5Dwrite(hid_4, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_1);
   hssize_t npoints_5 = H5Sget_select_npoints(hid_7);
   size_t size_dtype_6 = H5Tget_size(H5T_STD_I32LE);
   long long total_size_2 = npoints_5 * size_dtype_6;
   void *dummy_data_3 = (void *) malloc(total_size_2);
   H5Dwrite(hid_5, H5T_STD_I32LE, hid_7, hid_6, H5P_DEFAULT, dummy_data_3);
   H5Sclose(hid_6);
   H5Dclose(hid_4);
   H5Dclose(hid_5);
```

Code Compression - How to differentiate between processors

- **Using conditions:** The most straightforward solution to this problem is to use an if - else statement and put each of the ranks operations in their corresponding if clause.

```
if(mpi_rank == 0) {
    hsize_t stride_1[] =  {1,1};
    hsize_t count_1[] =  {6,24};
    hsize_t start_1[] = {0, 0};
    H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
}
else if(mpi_rank == 1) {
    hsize_t stride_1[] =  {1,1};
    hsize_t count_1[] =  {6,24};
    hsize_t start_1[] = {6, 0};
    H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
}
else if(mpi_rank == 2) {
    hsize_t stride_1[] =  {1,1};
    hsize_t count_1[] =  {6,24};
    hsize_t start_1[] = {12, 0};
    H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
}
else if(mpi_rank == 3) {
    hsize_t stride_1[] =  {1,1};
    hsize_t count_1[] =  {6,24};
    hsize_t start_1[] = {18, 0};
```

Code Compression - How to differentiate between processors

- **Using memory:** The second solution is to trade constant memory for code size. The way this works is that for every number or array which is different for different ranks, a new dimension is added corresponding to the rank of the MPI processes.

```
hsize_t start_1[4][2] = {
    {0,0},
    {6,0},
    {12,0},
    {18,0}
};
hsize_t stride_1[] =  {1,1};
hsize_t count_1[] =  {6,24};
H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1[mpi_rank], stride_1, count_1, NULL);
```

Code Compression - How to differentiate between processors

- **Identifying the relationship with MPI ranks:** In most of the cases, there is a simple relationship between the offsets of the file a process is accessing and rank of that process.

```
hsize_t start_1[2];
start_1[0] = 6 * mpi_rank + 0;
start_1[1] = 0 * mpi_rank + 0;
hsize_t stride_1[] =  {1,1};
hsize_t count_1[] =  {6,24};
H5Sselect_hyperslab(hid_6, H5S_SELECT_SET, start_1, stride_1, count_1, NULL);
```

- In addition to the benefits of memory and code size, this option makes it possible to scale the code to arbitrary number of processors.

Code Compression - Find out relation with loop index

- In addition to the previous problem, we need identification and compression of loop constructs too

- In most I/O applications, no HDF5 calls in a loop leading to small I/O traces

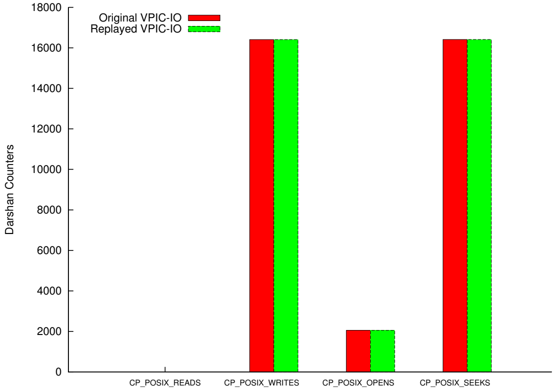
- We have developed a linear suffix tree based pattern matching tool to be used with the merger
  - This tool tells the code generation if and how many an expression is being repeated
  - The code generator will generate a loop for it
  - Again the problem of identifying the relationship of the numbers with the loop index!

Experimental Setup

- All the traces are gathered on the Stampede Dell cluster at Texas Advanced Computing Center (TACC)
  - A 10 PFLOPS supercomputer of more than 6400 nodes, each with 2 Intel Xeon E5 processors, with 16 cores per node
- Checked for three I/O kernels on 2048 cores generating about 500 GB of data:
  - VPIC-IO
  - VORPAL-IO
  - GCRM-IO
- Two factors to evaluate:
  - Correctness of the framework
  - Quality of the generated code

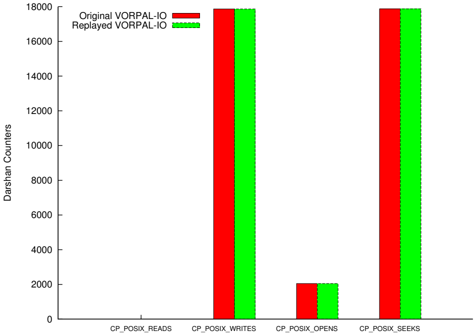
Correctness of the framework - VPIC-IO

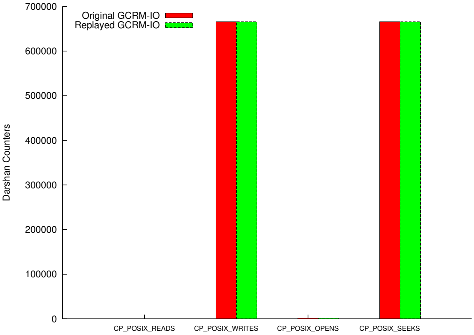
- The exact same value for all the 4 Darshan counters: `CP_POSIX_READ`, `CP_POSIX_WRITES`, `CP_POSIX_OPENS`, `CP_POSIX_SEEKS`
- The output files generated is exactly correct, both file size and output of using `h5dump` utility → Metadata is correct too



Correctness of the framework - VORPAL-IO and GCRM-IO

- Same as VPIC-IO, same value for all the 4 Darshan counters
- Same as VPIC-IO, correct output files generated and metadata





Quality of the generated code - Represented by its size

- VPIC-IO and GCRM-IO have generated code of size proportional to the original code
- VORPAL-IO however has much larger generated source code.
  - Complex relationship between the starting addresses of the 3D blocks assigned to the processes and their MPI ranks.
  - Fall back to using memory (solution #2)
  - 2048 cores were used for these experiments
  - Easy for the program developer to put this relationship in the generated code and reduce the size though

I/O Benchmark	Original Code	Generated Code	with user's help
VPIC-IO	8 KB	8 KB	8 KB
VORPAL-IO	12 KB	616 KB	36 KB
GCRM-IO	36 KB	12 KB	12 KB

Table 1: Comparison of the source code size of Original and Generated I/O Benchmarks

Related Work

- File System-level
  - Tracefs: Stony Brook University (Erez Zadok et al.)
- POSIX-level
  - //Trace: Carnegie Mellon University (Greg Ganger, et al.)
- MPI-IO-level
  - Scala-H-Trace: North Carolina State University and ORNL (Frank Mueller , Xiaosong Ma, et al.)
  - RIOT-IO: University of Warwick (Stephen Jarvis, et al.)
- Application-level
  - This work: University of Illinois and ANL → HDF5
  - Skel-IO: ORNL → ADIOS (Jeremy Logan, Scott Klasky, et al.)

Conclusion and Future Work

- It is easier to trace and generate I/O kernels at higher-level I/O libraries such as HDF5.
- Our framework consists of a:
  - ◇ A recorder library to trace the higher-level I/O operations
  - ◇ A merger tool which merges traces recorded on each process
  - ◇ A code generator generating the I/O kernel out of the merged I/O trace
- We have shown the applicability of this framework for three I/O kernels with very different I/O patterns.
- As the main future work, we are working on ways of automatically identifying the relationship between numbers and their ranks. We also are thinking about support pNetCDF library as well.

This code is available and free

- Both the recorder and replayer are available at: `https://github.com/babakbehzad`
- Please take a look at it and let us know how we can make it better.

Thank you for your attention

- Any Questions?

- Babak Behzad
- bbehza2@illinois.edu
- www.engr.illinois.edu/~bbehza2
- `https://github.com/babakbehzad`

Acknowledgements

- This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
- This research used resources of the Texas Advanced Computing Center and the Argonne Leadership Computing Facility.