Pattern-driven Parallel I/O Tuning

Babak Behzad[1], Surendra Byna[2], Prabhat[2],
Marc Snir[1,3]
[1]University of Illinois at Urbana-Champaign, [2]Lawrence Berkeley National Laboratory, [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

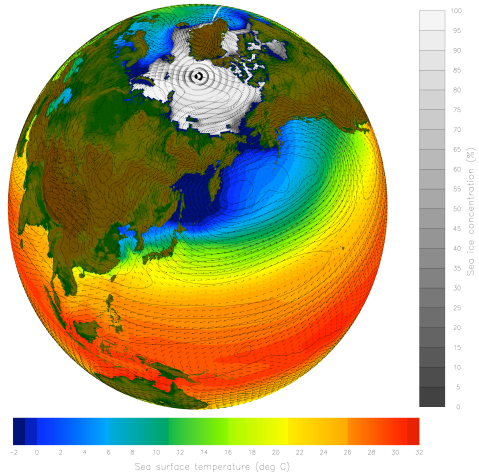


Figure: NCAR's CESM Visualization

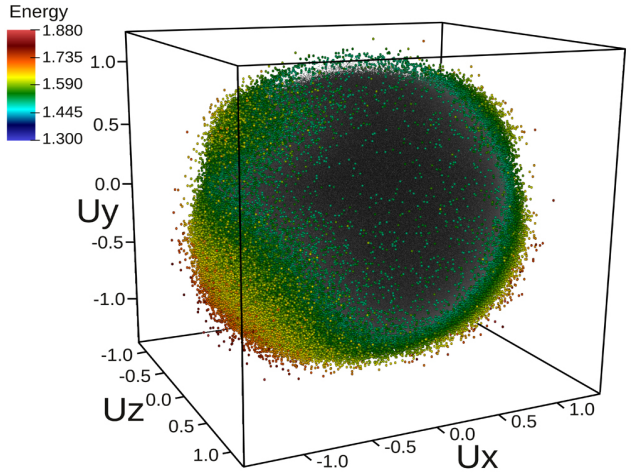
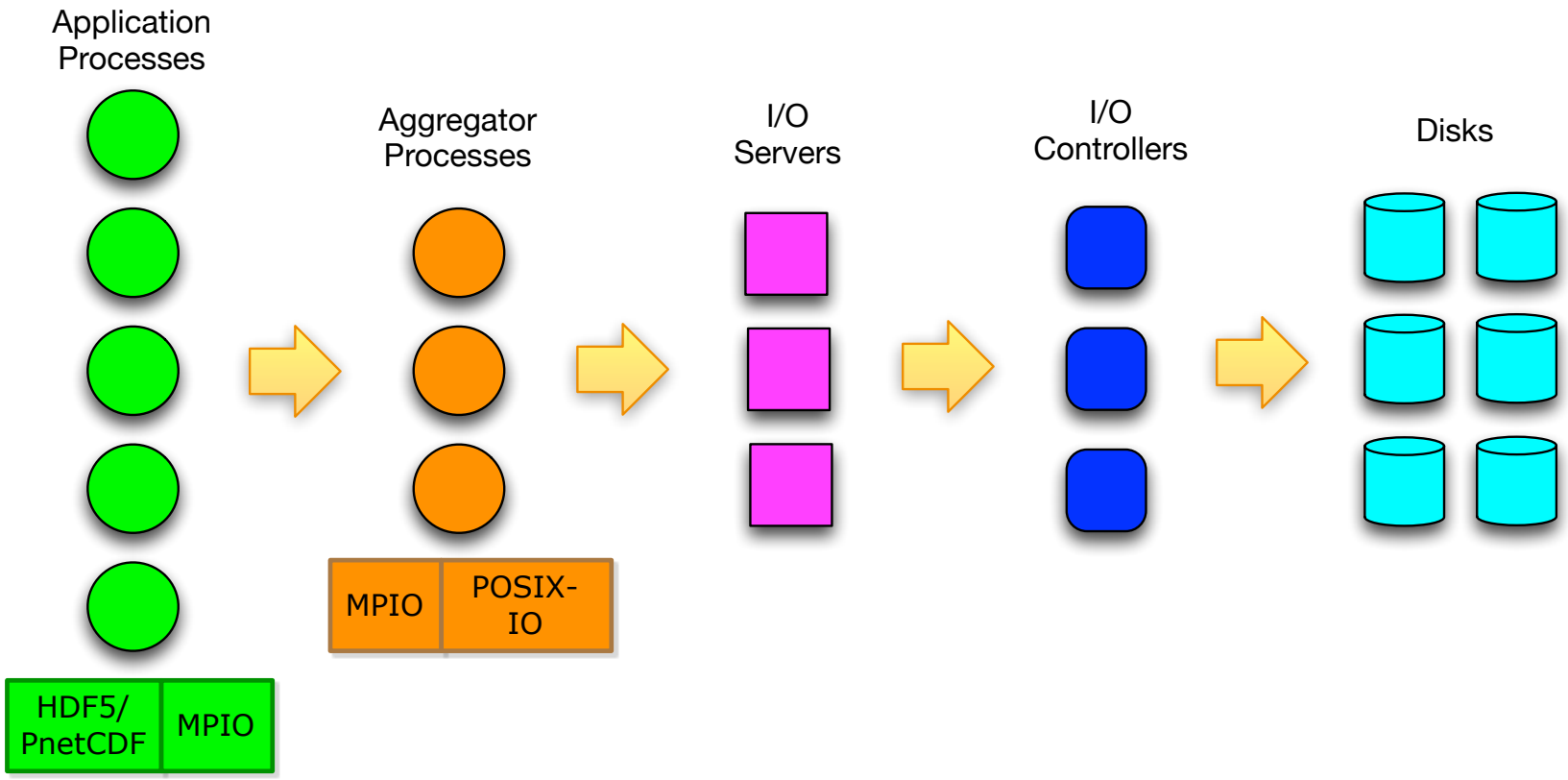


Figure: 1 trillion-electron VPIC dataset

Parallel I/O Subsystem

- I/O subsystem is complex
- There are a large number of knobs to set



Motivation by Related Work

Recent work at LANL on I/O Patterns by J. He et al. (HPDC'13)

"A typical I/O stack ignores I/O structures as data flows between layers... Eventually distributed data structures resolve into simple offset and length pairs in the storage system regardless of what initial information was available. In this study, we propose techniques to rediscover structures in unstructured I/O and represent them in a lossless and compact way."

Contributions

- We provide a new representation for I/O patterns based on the traces of high-level I/O libraries, such as HDF5.
  - This definition contains the global view of I/O accesses from all MPI processes in parallel applications.
- We develop a trace analysis tool for identifying I/O patterns of an application automatically.
- We show that using our runtime library, users can achieve significant portion of the peak I/O performance for arbitrary I/O patterns.

Addition to our Autotuning Framework

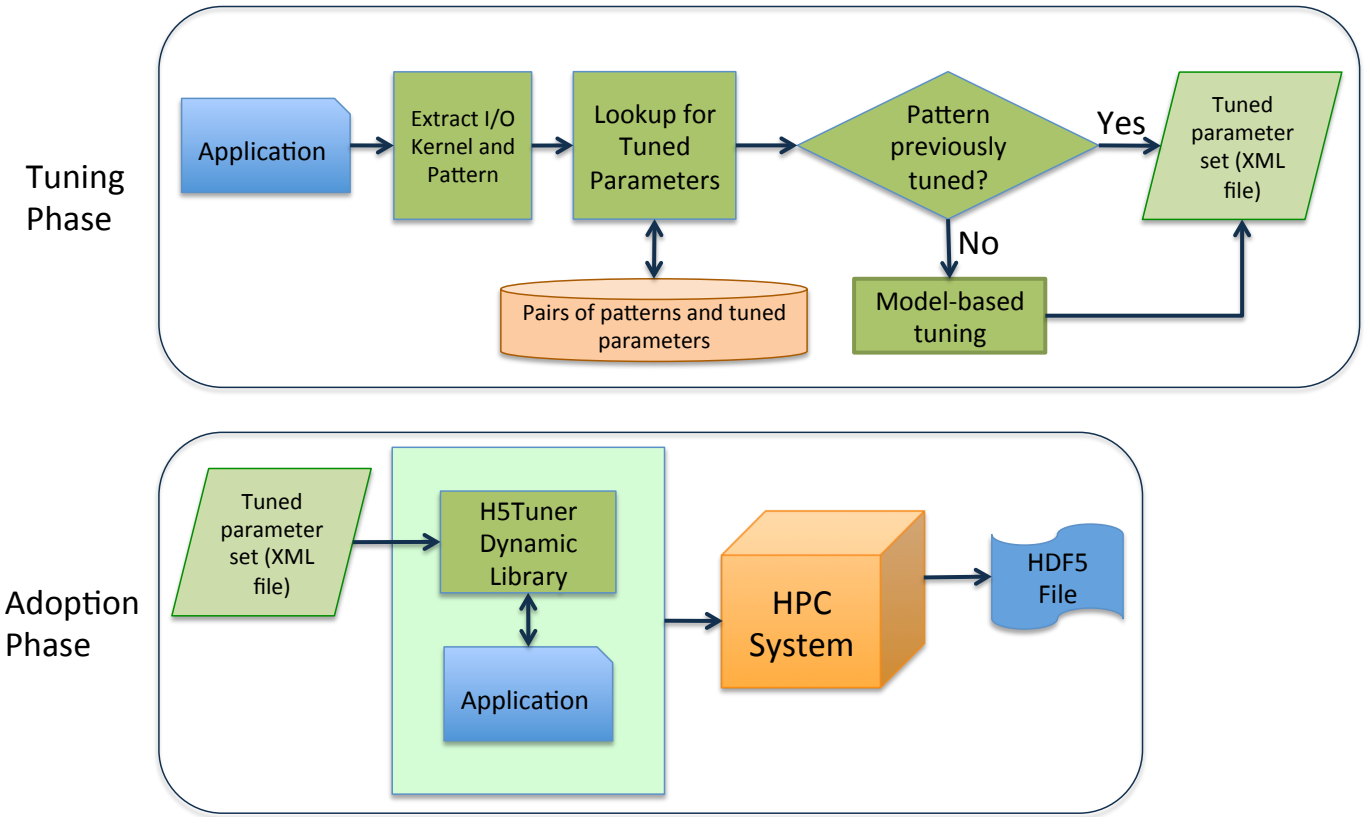
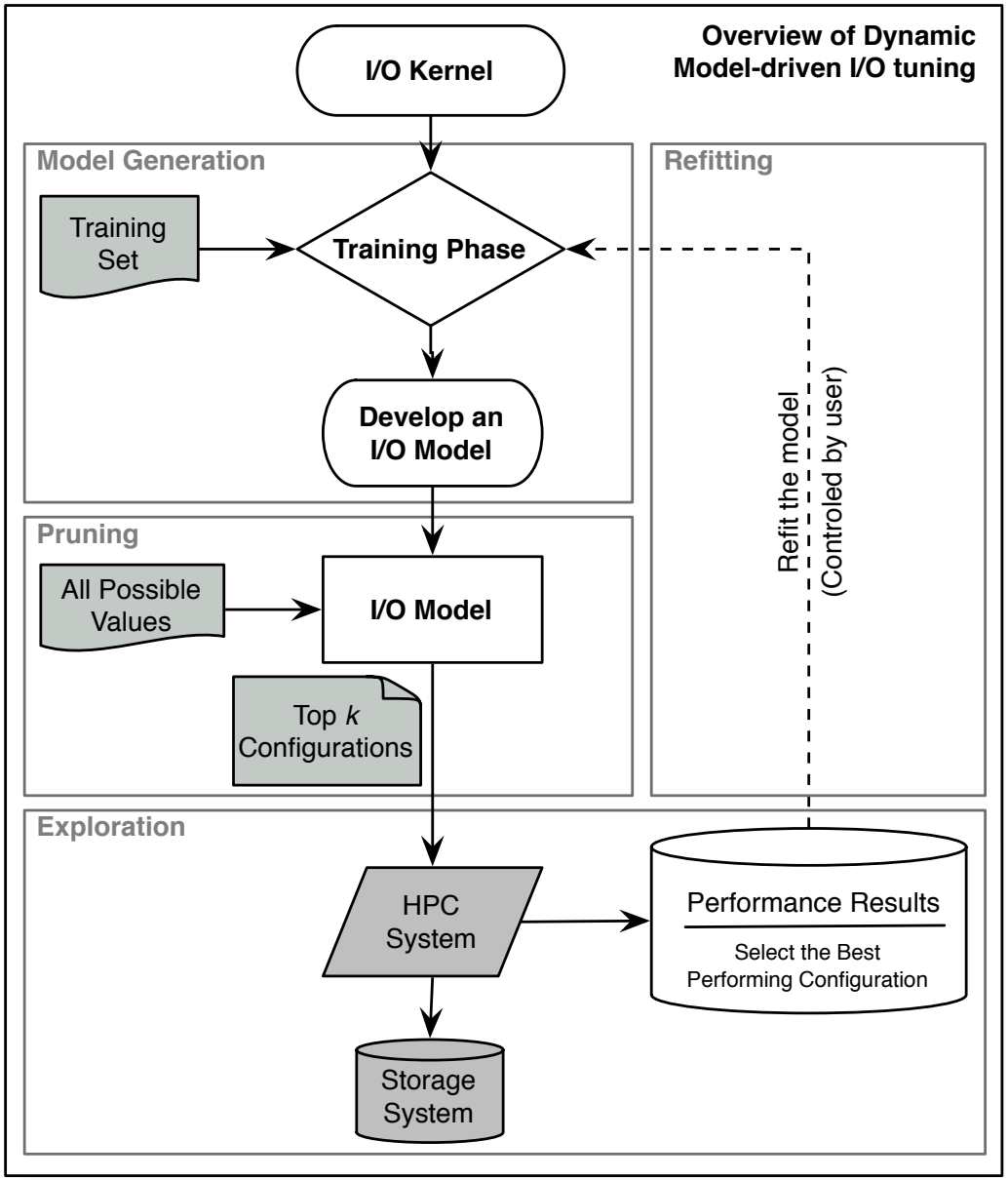


Figure: Architecture Design of our proposed runtime system for Tuning I/O

Autotuning Framework Review



I/O Pattern Definition

- Many ways of defining an I/O pattern of an application
- The key: Learn from the database community and separate the I/O pattern of an application into two categories:
  1. **Physical Pattern:** Related to the hardware configuration and is specific to file system, platform, etc. $\rightarrow$ These are all discussed in our previous work and statistical models have been proposed for it.
  2. **Logical Pattern:** Defined at the application level and the focus of this work. Takes the number of processors that run the application into account along with the distribution of the data between them, etc.

Background: I/O Traces

```
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
```

Figure: An I/O trace generated by the Recorder for a simple parallel application called `pH5Example`

I/O Pattern Definition: `H5S_select_hyperslab`

- Higher-level I/O libraries give us much more concepts in order to define and distinguish the the I/O operations.
- One of these concepts and probably the main one is the concept of *selection* in HDF5.
- Selection is an important feature of HDF5 library to select different parts of a file and memory.
- It also is the main point of difference for the processes to choose different parts of the file in a parallel I/O application.

$\longrightarrow$ We base our definition of I/O patterns on the concept of selection.

I/O Pattern Definition: H5S_select_hyperslab

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)
```

Rank 0:

```
H5Sselect_hyperslab (...,H5S_SELECT_SET,{0;0},{1;1},{6;24},NULL) 0
```

Rank 1:

```
H5Sselect_hyperslab (...,H5S_SELECT_SET,{6;0},{1;1},{6;24},NULL) 0
```

Rank 2:

```
H5Sselect_hyperslab (...,H5S_SELECT_SET,{12;0},{1;1},{6;24},NULL) 0
```

Rank 3:

```
H5Sselect_hyperslab (...,H5S_SELECT_SET,{18;0},{1;1},{6;24},NULL) 0
```

Figure: The four HDF5 hyperslab selection function calls across different ranks of a parallel four-process run of pH5Example

I/O Pattern Abstraction: HPF Terminology

- In order to abstract these patterns into one metric to be able to compare to, we make use of array distribution notation also used in High Performance Fortran.
- Below is a short description of each of these distributions:
  1. **Block Distribution:** Each process gets a single contiguous block of the array
  2. **Cyclic Distribution:** Array elements are distributed in a round-robin manner
  3. **Degenerate Distribution:** Represented by $*$, is basically no distribution or serial distribution. It means that all the elements of this dimension is assigned to one processor.

In Action: H5Analyze

- H5Analyze is a code we have developed based on pattern analysis provided by Zou et al. for analyzing HDF5 read and write traces.

$\longrightarrow$ <2D, (BLOCK, *), (6, 24)>

```
$ ./H5Analyze WRITE 1 testlog/pH5example_4 4
.
.
.
I/O Pattern with HPF Terminology:
Dataset name: output/ParaEg0.h5/Data1
  - Dimension: 2
  - Distribution: <BLOCK, DEGENERATE>
  - Size: <6, 24>
Dataset name: output/ParaEg0.h5/Data2
  - Dimension: 2
  - Distribution: <BLOCK, DEGENERATE>
  - Size: <6, 24>
```

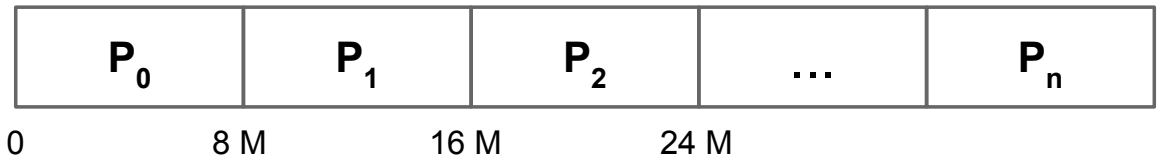
Figure: Output of H5Analyze for pH5example code

VPIC-IO accesses

- VPIC-IO (plasma physics): Vector Particle-In-Cell (VPIC) is a computer code simulating plasma behavior.

[start, stride, count, block]

P_0 = [{0}, {1}, {8 M}, {0}]
P_1 = [{8 M}, {1}, {8 M}, {0}]
P_2 = [{16 M}, {1}, {8 M}, {0}]
...



⟶ **VPIC-IO:** `<1D, BLOCK, 8388608>`

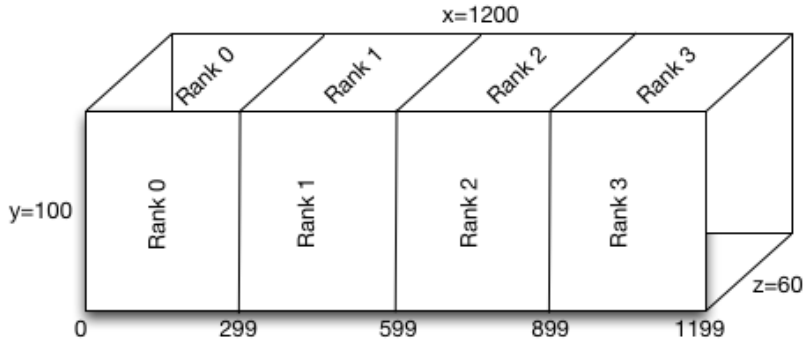
GCRM-IO accesses

- GCRM-IO (global atmospheric model): Global Cloud Circulation Model (GCRM), is an atmospheric model taking large convective clouds into global climate models.

[start, stride, count, block]

P_0 = [{0,0,0}, {1,1,1}, {1,26,327680}, {0,0,0}]
P_1 = [{0,0,327680}, {1,1,1}, {1,26,327680}, {0,0,0}]
P_2 = [{0,0,655360}, {1,1,1}, {1,26,327680}, {0,0,0}]

...



⟶ **GCRM-IO:** `<3D, (*, *, BLOCK), (1, 1, 327680)>`

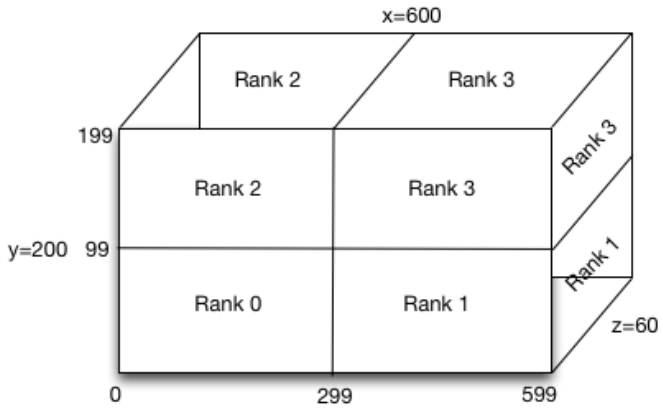
VORPAL-IO accesses

- VORPAL-IO (accelerator modeling): VORPAL is an acceleration modeling and computation plasma framework.

[start, stride, count, block]

P_0 = [{0,0,0}, {1,1,1}, {60,100,300}, {0,0,0}]
P_1 = [{0,0,300}, {1,1,1}, {60,100,300}, {0,0,0}]
P_2 = [{0,100,0}, {1,1,1}, {60,100,300}, {0,0,0}]

...



$\longrightarrow$ **VORPAL-IO:** `<3D, (BLOCK, BLOCK, BLOCK), (60, 100, 300)>`

Experimental Setup: Platforms

1. **NERSC/Hopper**
   - Cray XE6
   - Lustre Filesystem
   - Each file at max 156 OSTs
   - 26 OSSs
   - Peak I/O Performance (one file per process): 35 GB/s
2. **NERSC/Edison**
   - Cray XC30
   - Lustre Filesystem
   - Each file at max 96 OSTs
   - 24 OSSs
   - Peak I/O Performance (one file per process): 48 GB/s

Experimental Setup: Applications

1. **IOR-1D:** In order to have IOR issue write patterns similar to VPIC- IO, we configured it to use its HDF5 interface: `./ior -s 8 -w -b 32m -t 32m`.
2. **Resemble-VORPAL-IO-3D:** A synthetic benchmark with similar I/O pattern to VORPAL-IO benchmark but with different block sizes of $64 \times 128 \times 256$ instead of $60 \times 100 \times 300$ of VORPAL-IO.
3. **FLASH-IO:** Based on the output of `H5Analyze` tool, FLASH-IO has 34 datasets, out of which 24 of them have the same size as the largest size of the file. We choose those as the pattern of FLASH-IO. These datasets are 4D and their pattern of these dataset are also the same: $\longrightarrow$ `<4D, (BLOCK, *, *, *)>` $\rightarrow \approx$ GCRM-IO.

Results: IOR-1D – The same I/O Pattern as VPIC-IO

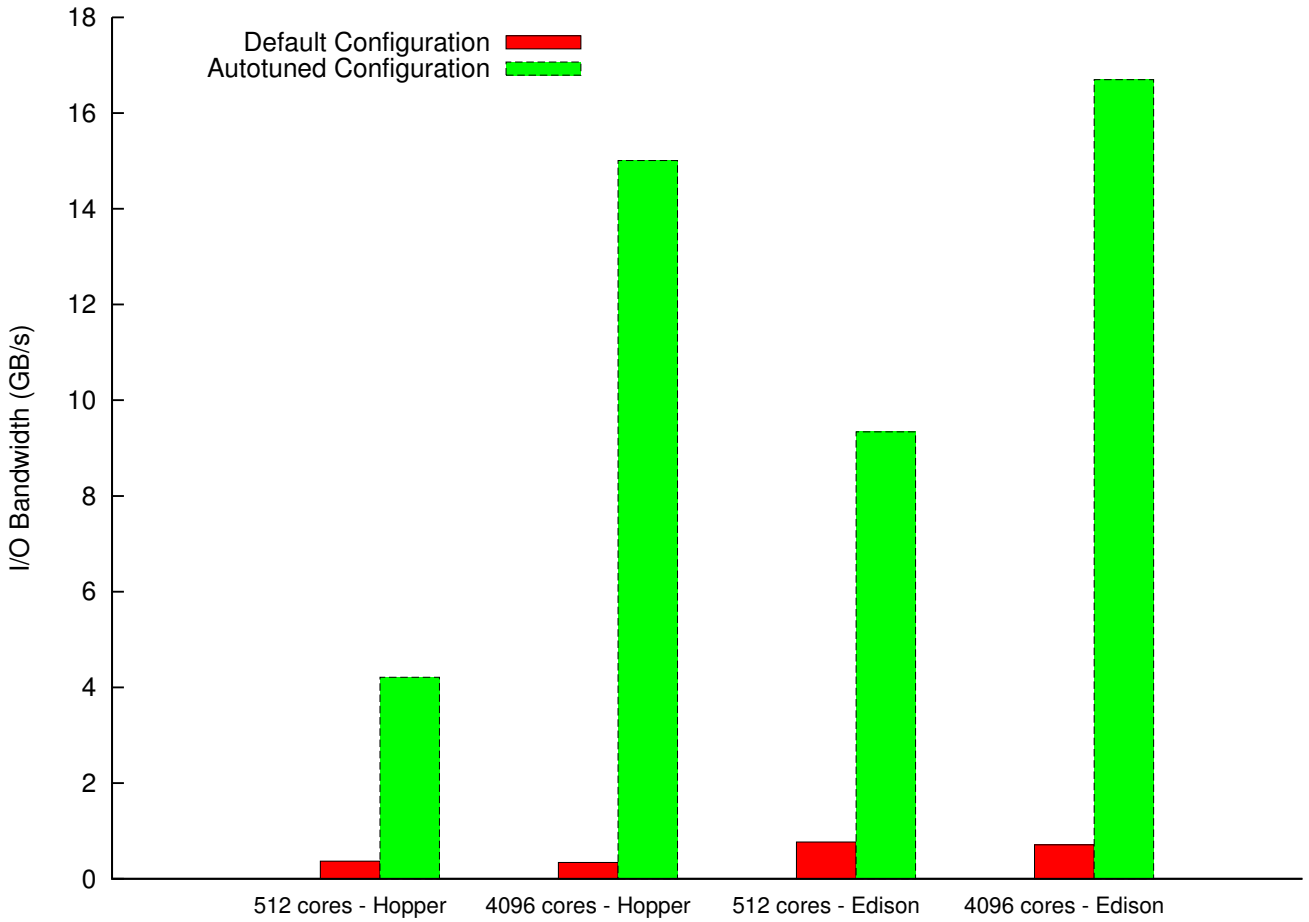


Figure: The I/O performance of the autotuned IOR on Hopper and Edison compared the default configuration.

Results: Resemble-VORPAL-IO-3D – Different I/O Pattern than VORPAL-IO

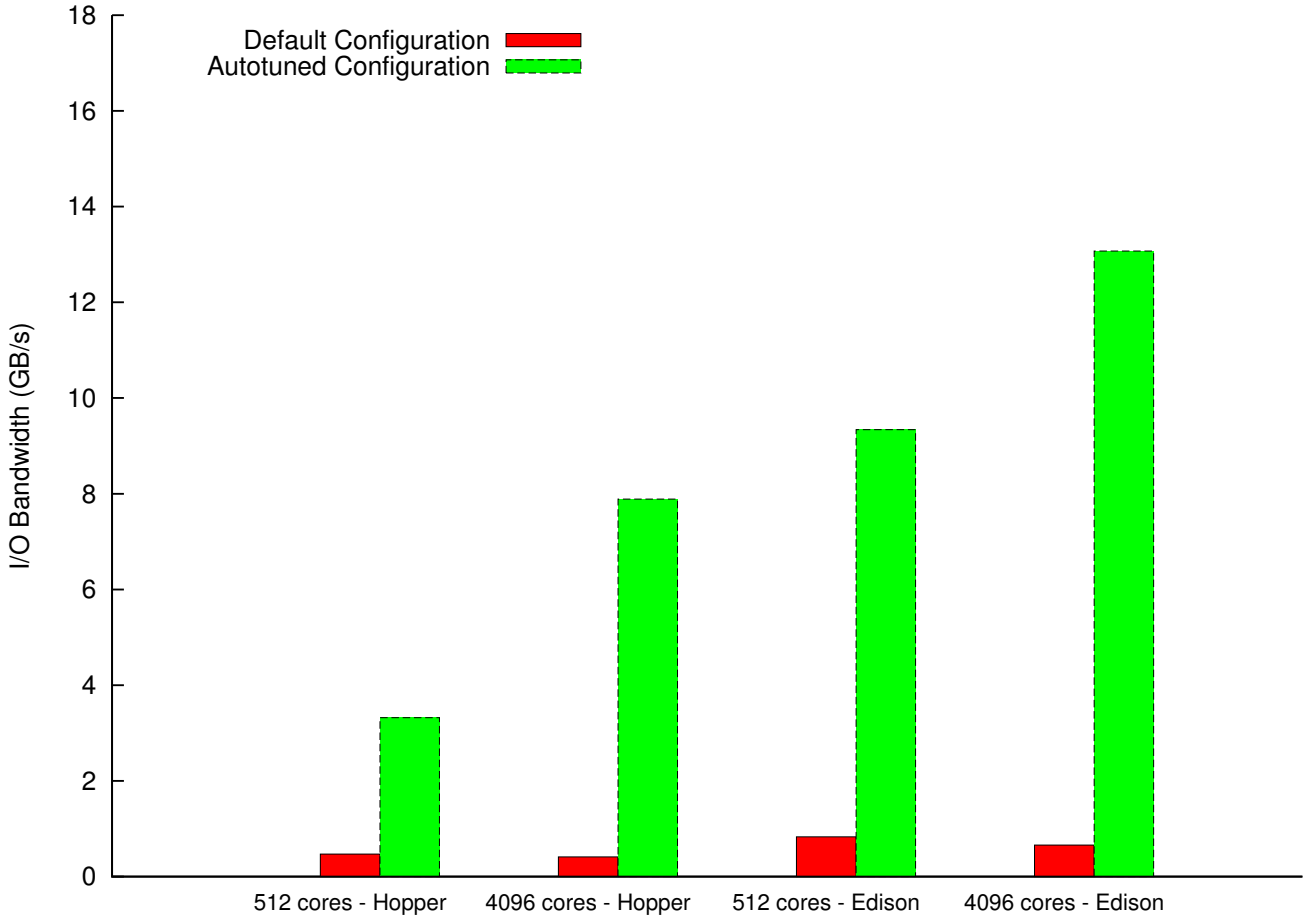


Figure: The I/O performance of the autotuned Resemble-VORPAL-IO-3D on Hopper and Edison compared the default configuration.

Results: FLASH-IO – A new application

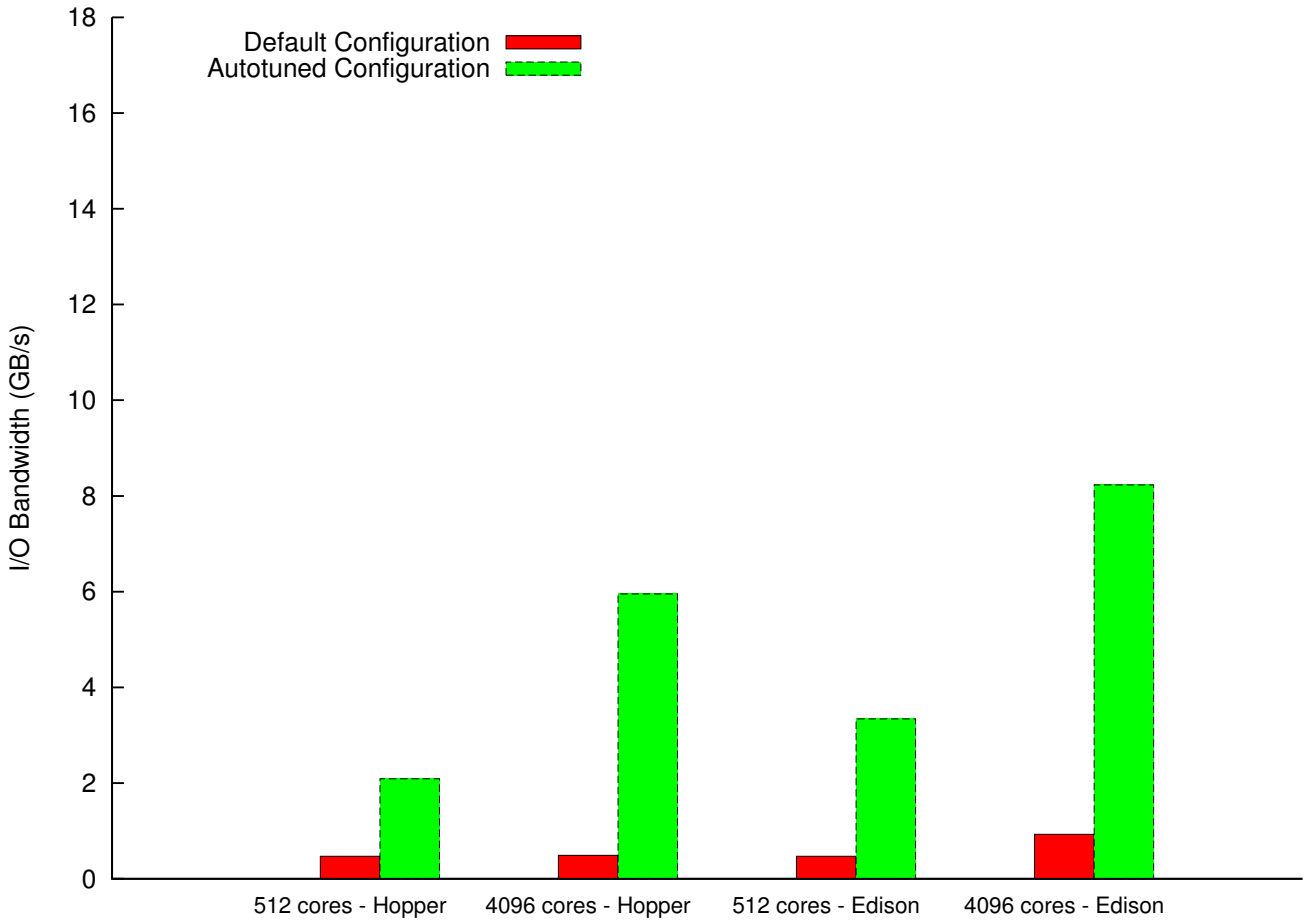


Figure: The I/O performance of the autotuned FLASH-IO application on Hopper and Edison compared the default configuration.

Conclusions and Future Work

- In this paper, we propose a pattern-driven autotuning framework to solve poor HPC I/O performance problem.
- We show that using high-level patterns, one can tune different sets of applications ranging from the ones which have tuned before the ones which are similar to the ones before, and totally new ones.
- The framework consists of components to extract I/O patterns, tune configuration for the detected patterns, *store them in a database of patterns associated with their I/O model, and finally map an arbitrary I/O pattern to a previously tuned model in order to improve its I/O performance*.

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 National Energy Research Scientific Computing Center.