# Taming Parallel I/O Complexity with Auto-Tuning

Babak Behzad[1], Huong Vu Thanh Luu[1], Joseph Huchette[2], Surendra Byna[3], Prabhat[3], Ruth Aydt[4], Quincey Koziol[4], Marc Snir[1,5]

[1]University of Illinois at Urbana-Champaign, [2]Rice University, [3]Lawrence Berkeley National Laboratory, [4]The HDF Group, [5]Argonne National Laboratory




# Parallel I/O is Essential

- Parallel I/O: An essential component of modern HPC
  - Scientific simulations periodically write their state as checkpoint datasets
  - Input and output datasets have become larger and larger
- Optimal I/O performance: Critical for utilizing the power of HPC machines
  - Slow I/O reduces machine utilization and wastes energy consumption

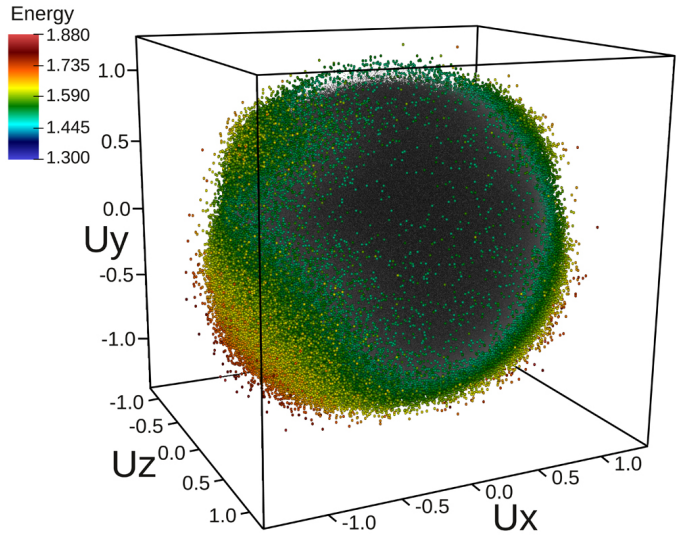
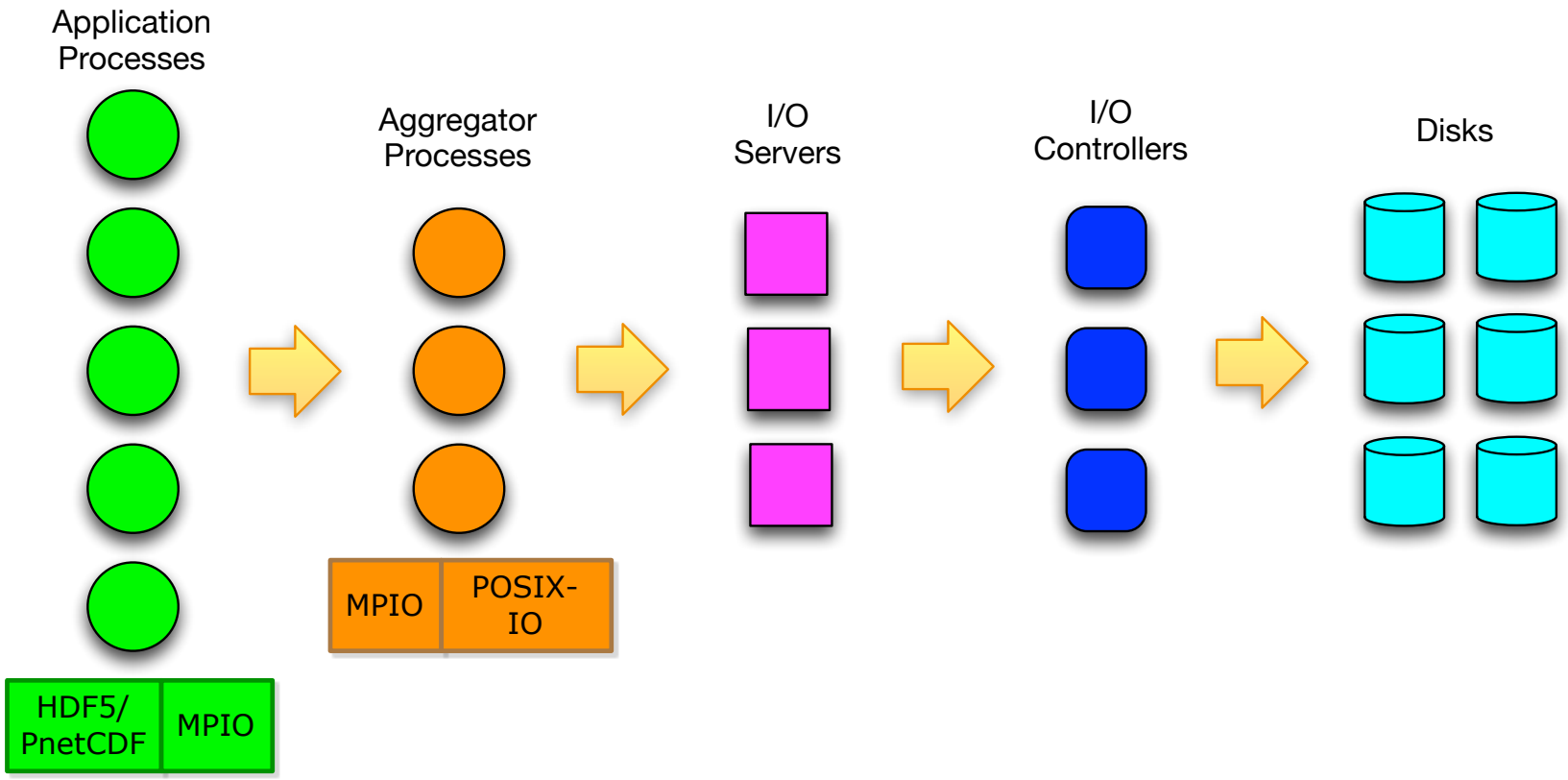



Figure : 1 trillion-electron dataset of VPIC generating 30 TB of data. Image by Oliver Rubel

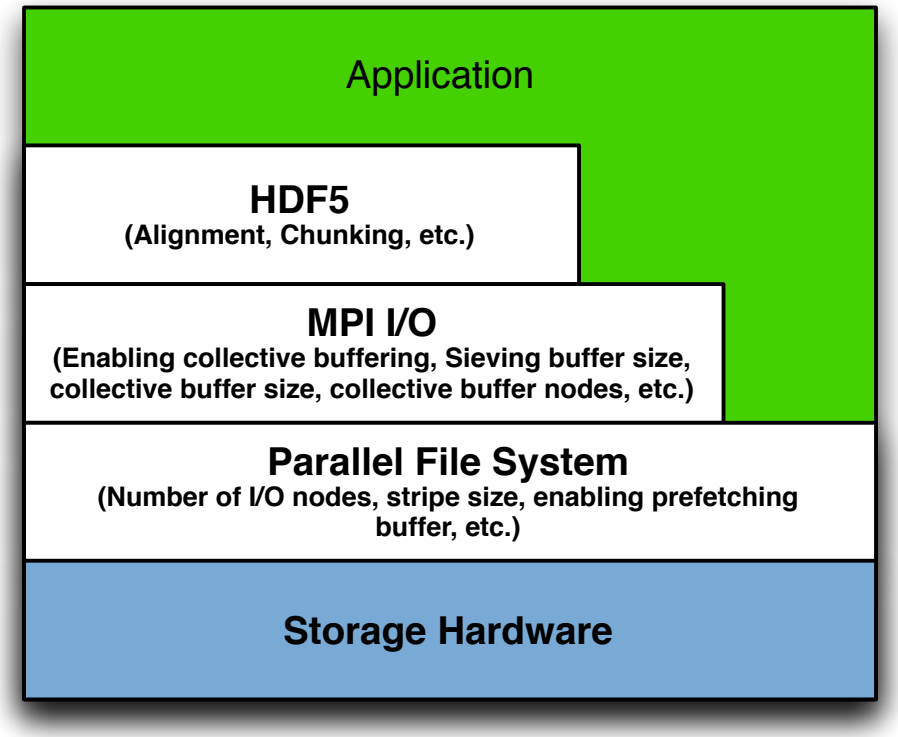
# Parallel I/O

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



# Complexities of Parallel I/O

- Different optimization parameters at each layer of the I/O software stack
- Complex inter-dependencies between these layers
- Highly dependent on the application, HPC platform, and problem size/concurrency
- Application developers need good I/O performance without becoming experts on the I/O



# Taming Parallel I/O Complexity: Auto-Tuning

- An I/O auto-tuning framework to find optimization parameters of the I/O stack
  - Targets the entire stack
  - Discovers I/O tuning parameters at each layer that result in good I/O rates

- The main challenges in this I/O auto-tuning system are
  1. Selecting an effective set of tunable parameters at all layers of the stack and their values → **H5Evolve**
  2. Applying the parameters to applications or I/O benchmarks without modifying the source code → **H5Tuner**

# Challenge 1: Selecting an Effective Set of Tunable Parameters

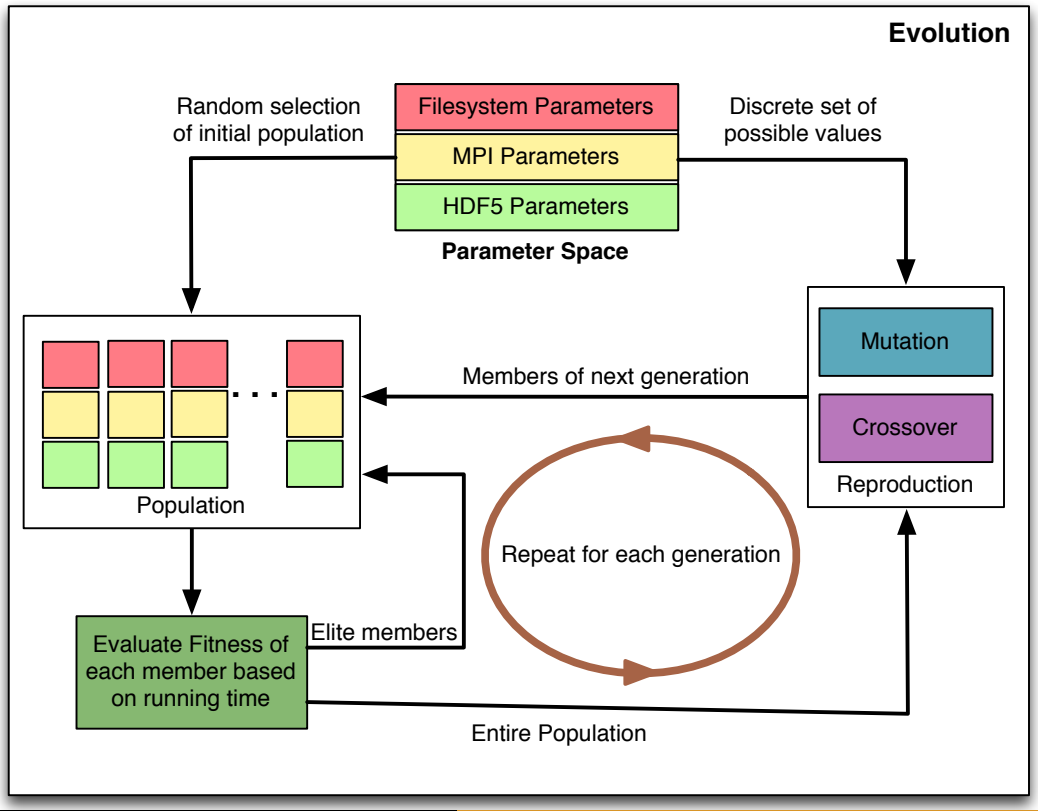
- Based on previous experience, a few parameters most affecting the I/O performance were selected for tuning
- A subset of different values for these parameters were explored

| Parameter | Min | Max | # Values |
|---|---|---|---|
| Lustre Stripe Count | 4 | 156/160 | 10 |
| Lustre Stripe Size / CB Size | 1 MB | 128 MB | 8 |
| MPI-IO Collective Buffering Nodes | 1 | 256 | 12 |
| HDF5 Alignment | (1,1) | (16KB, 32MB) | 14 |
| GPFS `bglockless` | True | False | 2 |
| GPFS `IBM_largeblock_io` | True | False | 2 |
| HDF5 Chunks Size | 10 MB | 2 GB | 25 |

- 13,440 possible configurations on Lustre without chunking!
- 336,000 possible configurations on Lustre with chunking!

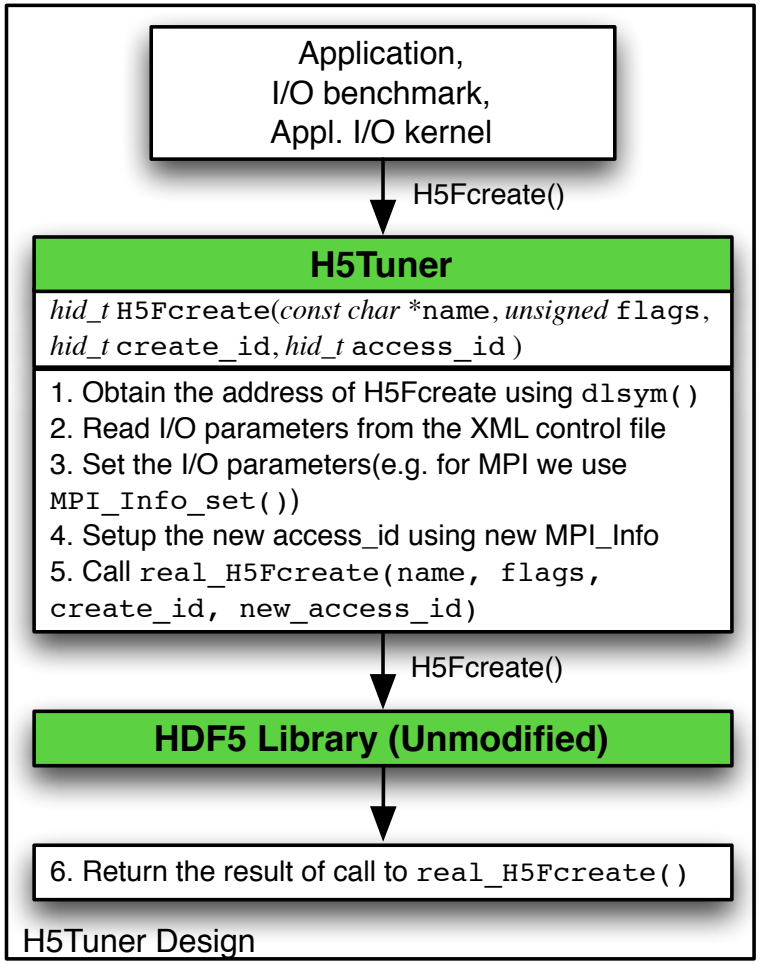
# H5Evolve: Sampling the Search Space to Find Good Configurations

- Very large parameter space
- Possibly long execution time of a trial run
- Use Genetic Algorithm (GA) to converge to a good configuration with a number of trials



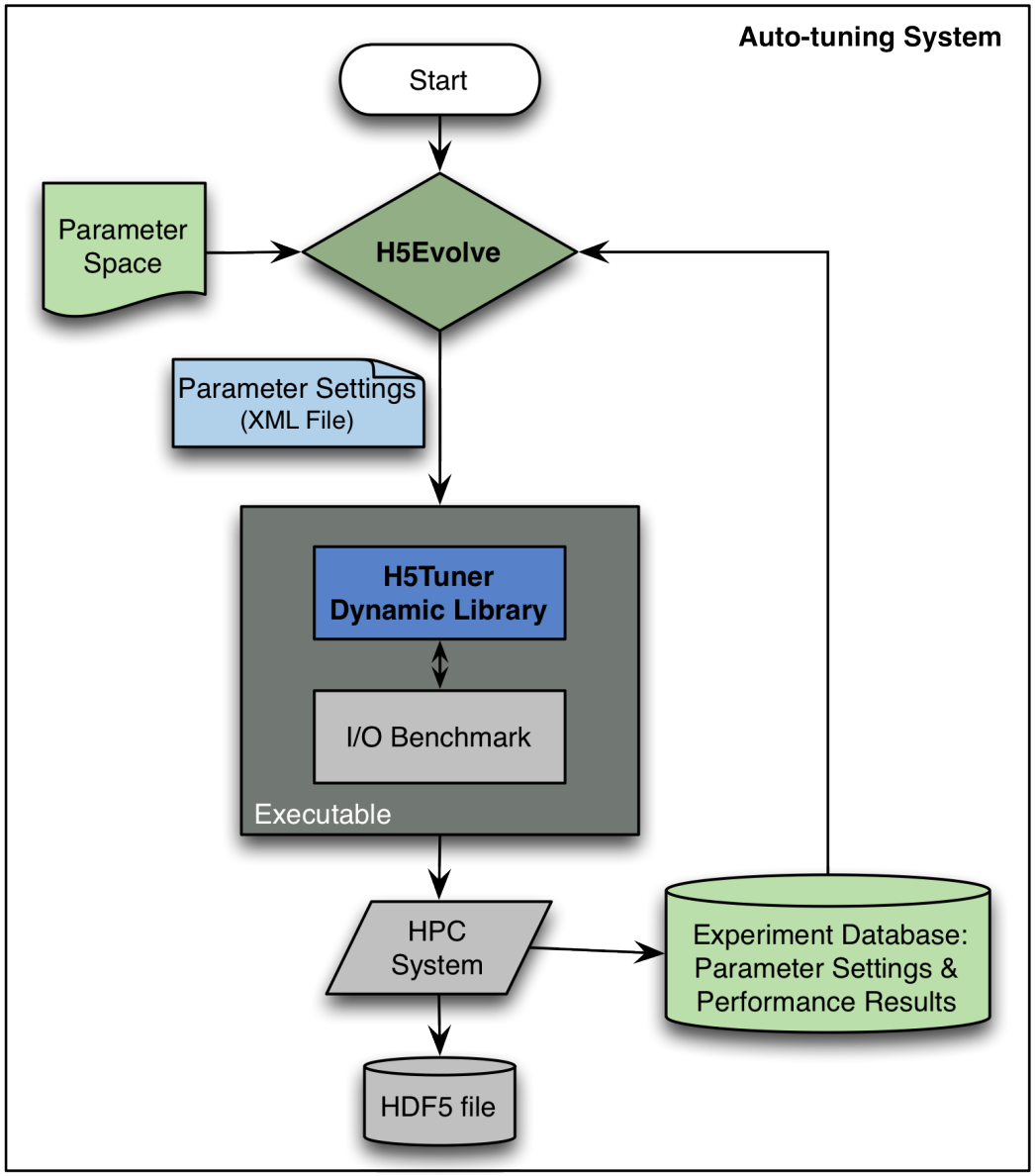
# Challenge 2: Setting I/O Parameters at Runtime (H5Tuner)

- The H5Tuner dynamic library
- Set the parameters of different levels of the I/O stack at runtime
- No source code modification or recompilation is needed



H5Tuner Design

# Overall Architecture of the Auto-Tuning Framework



# 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. **ALCF/Intrepid**
   - IBM BG/P
   - GPFS Filesystem
   - 640 IO Nodes
   - 128 file servers
   - Peak I/O Performance (one file per process): 47 GB/s (write)
3. **TACC/Stampede**
   - Dell PowerEdge C8220
   - Lustre Filesystem
   - Each file at max 160 OSTs
   - 58 OSSs
   - Peak I/O Performance (one file per process): 159 GB/s

# Experimental Setup: Application I/O Kernels

1. **VPIC-IO**
   - IO-Kernel manually derived from VPIC plasma physics application
   - Writes 1D array
2. **VORPAL-IO**
   - IO-Kernel manually derived from VORPAL accelerator modeling
   - Writes 3D block-structured grid
3. **GCRM-IO**
   - IO-Kernel manually derived from GCRM atmospheric model
   - Writes semi-structured mesh

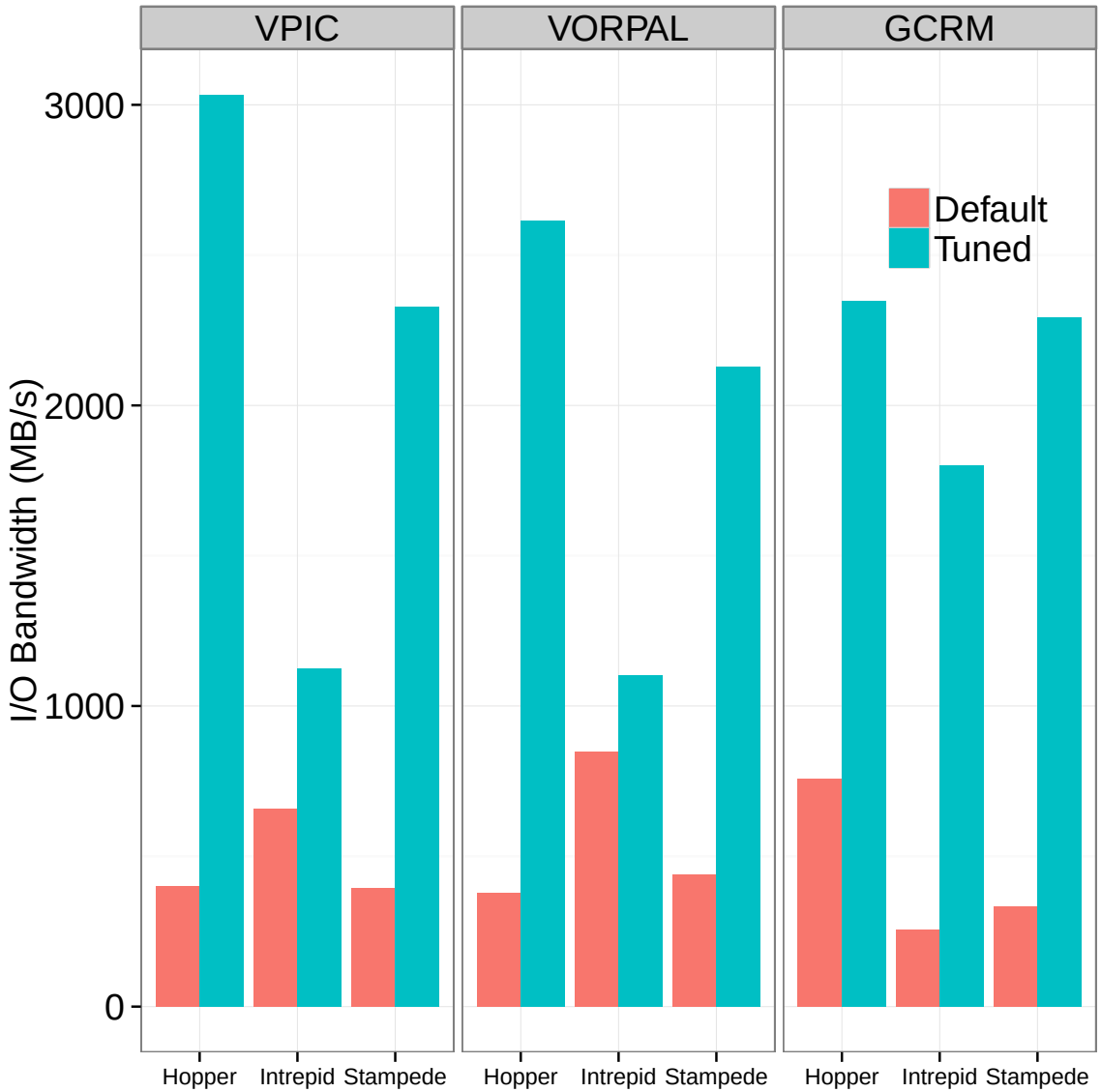
# Experimental Setup: Concurrency and Dataset Sizes

- Weak-scaling configuration to test the auto-tuning framework
- One output file shared between all the processors

| I/O Benchmark | 128 Cores | 2048 Cores | 4096 Cores |
|---|---|---|---|
| VPIC-IO | 32 GB | 512 GB | 1.1 TB |
| VORPAL-IO | 34 GB | 549 GB | 1.1 TB |
| GCRM-IO | 40 GB | 650 GB | 1.3 TB |

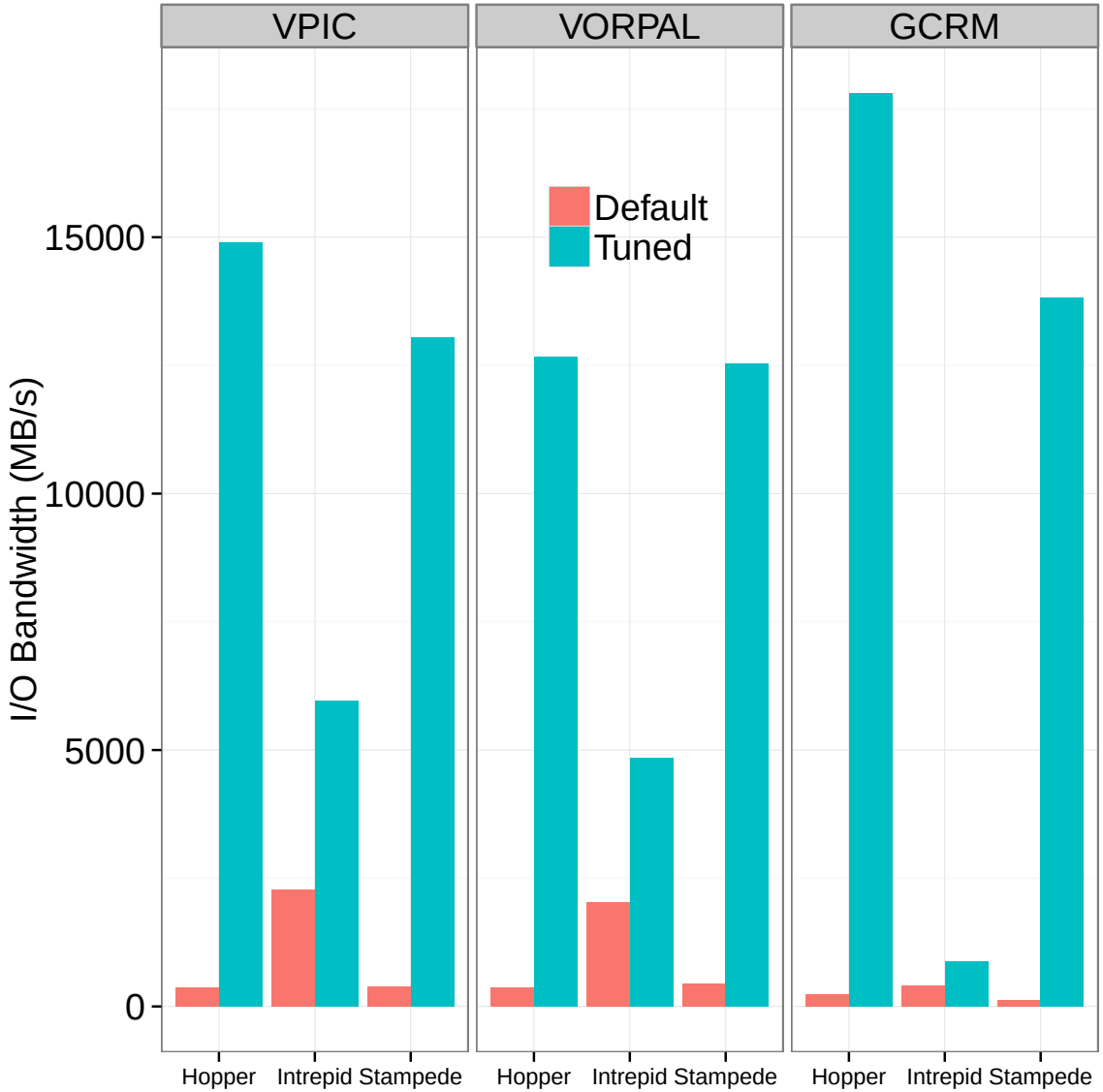
# Performance Improvement Results - 128 cores

- Speedup Range: 1.3x - 7.5x
- With default settings, Intrepid is doing better than Hopper and Stampede
- Lustre has more room for improvement



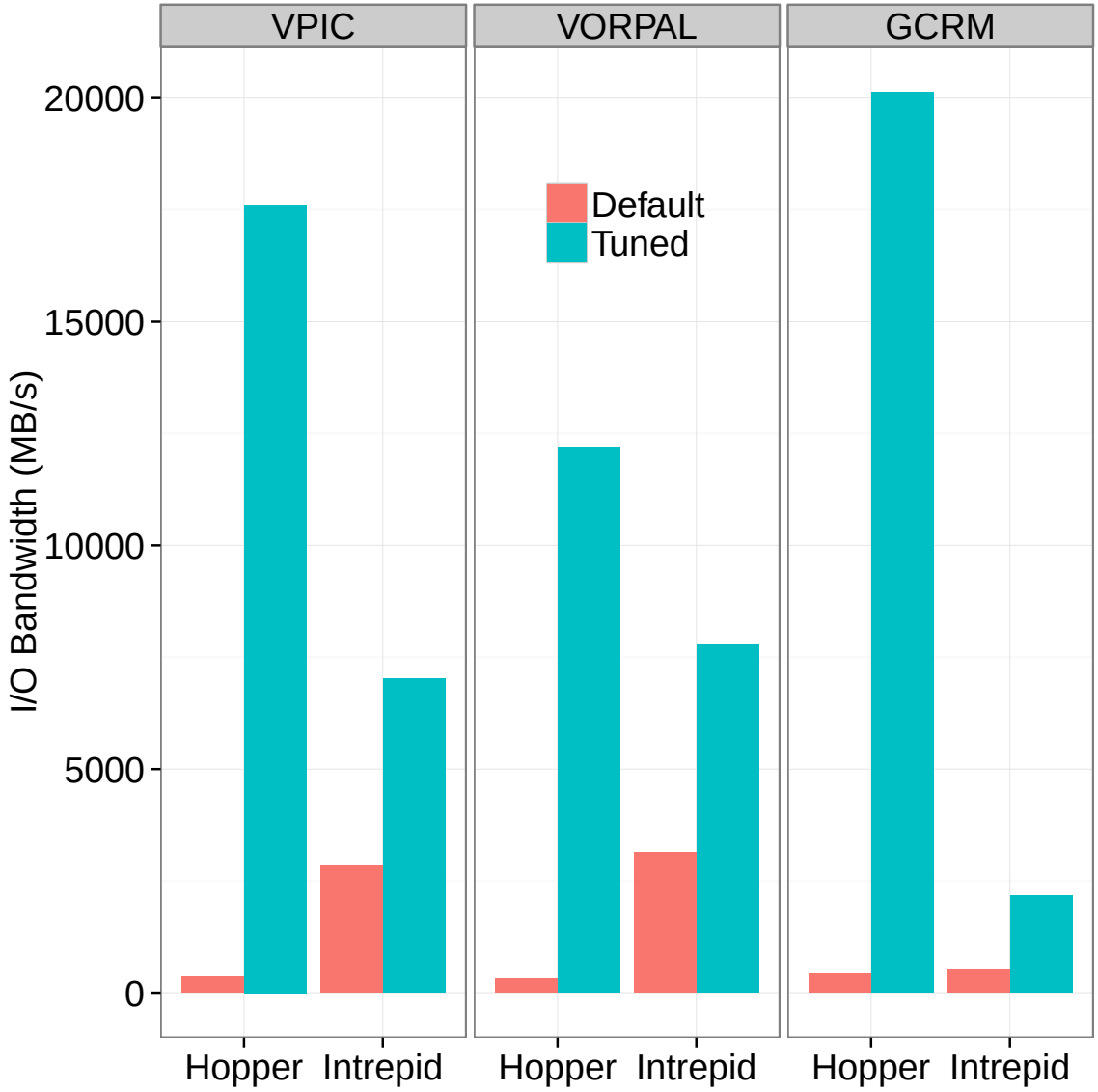
# Performance Improvement Results - 2048 cores

- Speedup Range: 2.1x - 107.7x
- Up to 17 GB/s of bandwidth
- At higher scale, higher speedups are observed



# Performance Improvement Results - 4096 cores

- Speedup Range: 2.4x - 50.6x
- Up to 20 GB/s of bandwidth
- Due to lack of allocations, Stampede results are missing



# Tuned Parameters: VPIC-IO 2048 cores

- Same application, same scale, different platforms: Different parameters
- One default value for the parameters: Not good

| I/O Kernel | System | Tuned Parameters |
|---|---|---|
| VPIC-IO | Hopper | `strp_fac=156, strp_unt=32MB, cb_nds=512, cb_buf_size=32MB, align=(1K,64K)` |
| VPIC-IO | Intrepid | `bgl_nodes_pset=512, cb_buf_size=128MB, bglockless=true, largeblock_io=false, align=(8K, 1MB)` |
| VPIC-IO | Stampede | `strp_fac=128, strp_unt=8MB, cb_nds=512, cb_buf_size=8MB, align=(8K, 2MB)` |

# Tuned Parameters: VORPAL-IO 2048 cores

- Different applications, different tuned configuration

| I/O Kernel | System | Tuned Parameters |
|---|---|---|
| VORPAL-IO | Hopper | `strp_fac=156, strp_unt=32MB, cb_nds=128, cb_buf_size=32MB, align=(4K,256K)` |
| VORPAL-IO | Intrepid | `bgl_nodes_pset=128, cb_buf_size=128MB, bglockless=true, largeblock_io=true, align=(8K, 8MB)` |
| VORPAL-IO | Stampede | `strp_fac=160, strp_unt=2MB, cb_nds=512, cb_buf_size=2MB, align=(8K, 8MB)` |

# Tuned Parameters: GCRM-IO 2048 cores

| I/O Kernel | System | Tuned Parameters |
|---|---|---|
| GCRM-IO | Hopper | `strp_fac=156,`<br>`strp_unt=32MB,`<br>`chunk_size=(1,26,327680)=32MB,`<br>`align=(2K, 64KB)` |
| GCRM-IO | Intrepid | `chunk_size=(1,26,1048760)=1GB,`<br>`align=(1MB, 4MB)` |
| GCRM-IO | Stampede | `strp_fac=160,`<br>`strp_unt=32MB,`<br>`chunk_size=(1,26,1048760)=1GB,`<br>`align=(1MB, 4MB)` |

# Future Work

- Better optimization techniques for faster convergence
- Automatic generation of I/O kernels from full applications
- Characterizing the noise on the storage system
- I/O Motifs