I/O workload characterization in MPI applications

Darshan Introduction

Yushu Yao
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

Phil Carns
Mathematics and Computer Science Division
Argonne National Laboratory

Motivation

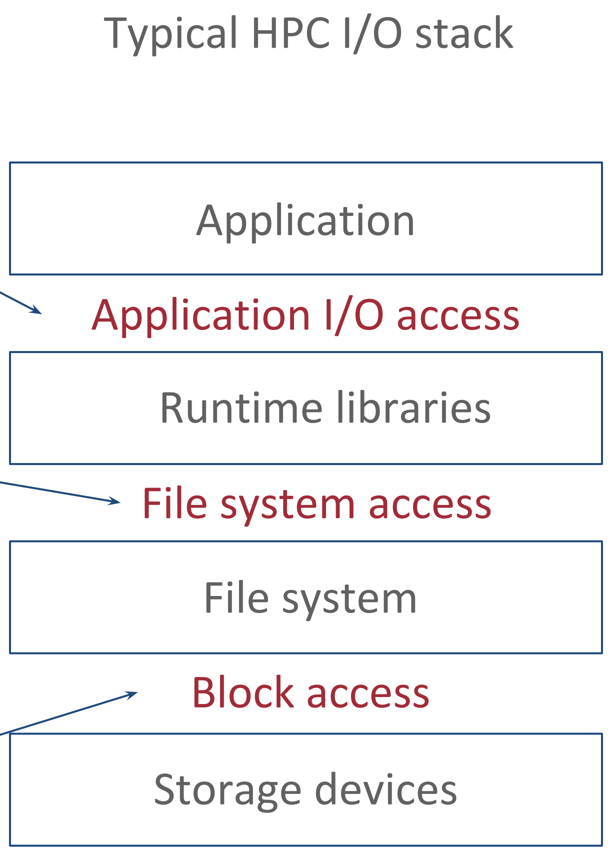
I/O behavior plays a key role in application performance and scientific productivity.

- Challenges to understanding I/O behavior:
  - *Applications are complex*. There may be many components accessing different files at different times in different ways.
  - I/O performance is difficult to isolate from computation and network performance
  - I/O performance is sensitive to changes in access methods, libraries, file systems, and hardware
  - I/O performance may be perturbed by the very tools used to instrument it

In this tutorial we will present an introduction to I/O challenges and tools that can be used to diagnose them.

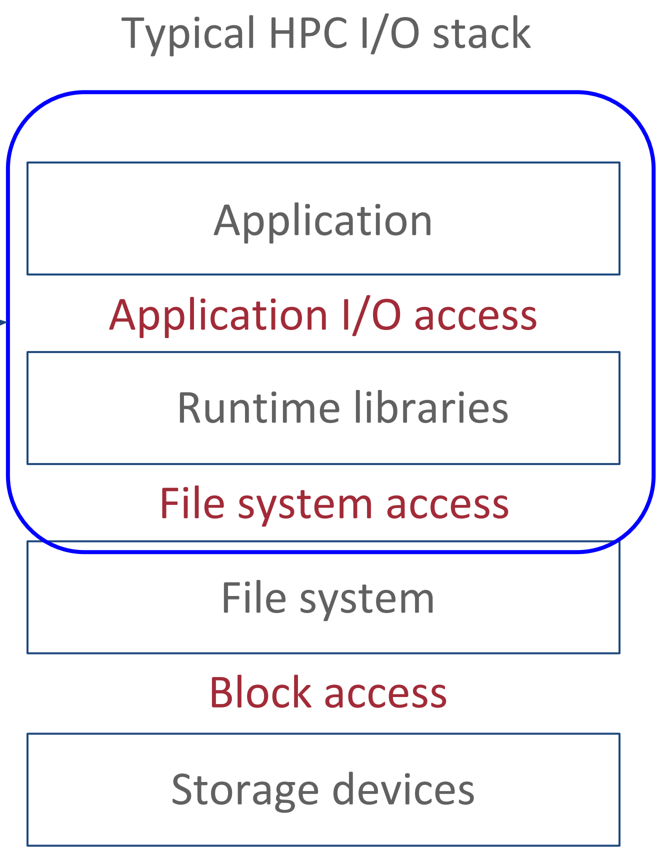
I/O instrumentation methods

- Application profiling or tracing tools
  - I/O access expressed by application
  - **Examples**: Darshan, IPM, Scalatrace, Tau, etc.
- File system or POSIX instrumentation
  - I/O access patterns after they have been translated by I/O libraries such as MPI-IO or HDF5
  - **Examples**: Lustre and GPFS diagnostics, as well as most profiling and tracing tools
- Block device instrumentation
  - Raw access to disks
  - **Examples**: vendor-specific diagnostics, Linux block tracing, etc.



I/O instrumentation methods

- When tuning an application, the application I/O access and file system access levels are the most important to understand
- Block access is useful from a system tuning and utilization point of view, but is difficult to map to application performance
- In this tutorial we will focus on a specific software tool, Darshan, that instruments application level behavior at the application process level.



Why Darshan?

Key properties:

- Portability:
  - Works on IBM Blue Gene, Cray, and Linux environments
  - Compatible with all popular compilers
  - Compatible with all popular MPI implementations
- Minimal perturbation of application performance
  - Will not change behavior in any measureable way
  - You can leave it "on" at all times
- Low barrier to entry
  - Usually you can enable darshan instrumentation by just compiling your application with the right compiler script

Darshan is a lightweight, scalable I/O characterization tool that transparently captures I/O access pattern information from production applications.

Darshan overview

- Open source runtime library
  - Instrumentation is inserted at build time (for static executables) or at run time (for dynamic executables)
  - Captures POSIX I/O, MPI-IO, and limited HDF5 and PNetCDF functions
- Minimal application impact
  - Low memory consumption
  - Reduces, compresses, and aggregates data at MPI_Finalize() time
  - Instrumentation enabled via software modules, environment variables, or compiler scripts
  - No source code or makefile changes
  - No file system dependencies

How to use Darshan

- Compile a C, C++, or FORTRAN program that uses MPI
- Run the application
- Look for the ***Darshan log file***
  - This will be in a particular directory (depending on your system's configuration)
  - <dir>/<year>/<month>/<day>/<username>_<appname>*.darshan.gz
- Use Darshan command line tools to analyze the log file
- **Darshan does not capture a trace of all I/O operations**: instead, it reports key statistics, counters, and timing information for each file accessed by the application.

- *Application must run to completion and call MPI_Finalize() to generate a log file*

Darshan analysis tool example

- ❑ Each job instrumented with Darshan produces a single characterization log file
- ❑ Darshan command line utilities are used to analyze these log files
- ❑ Example: Darshan-job-summary.pl produces a 3-page PDF file summarizing various aspects of I/O performance

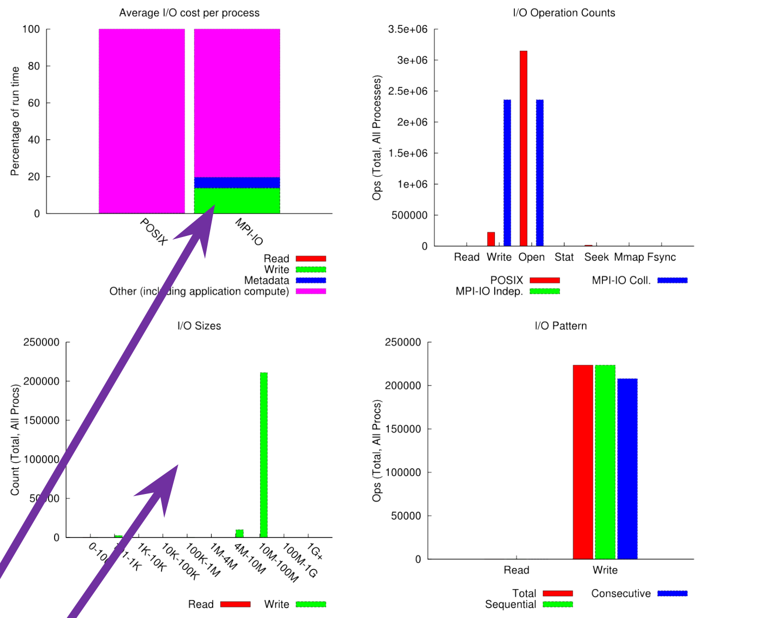
- ❑ This figure shows the I/O behavior of a 786,432 process turbulence simulation (production run) on the Mira system at ANL
- ❑ Application is write intensive and benefits greatly from collective buffering

Example measurements: % of runtime in I/O

access size histogram

(9/25/2013) 1 of 3

jobid: 149563	uid: 6729	nprocs: 786432	runtime: 2751 seconds



Most Common Access Sizes

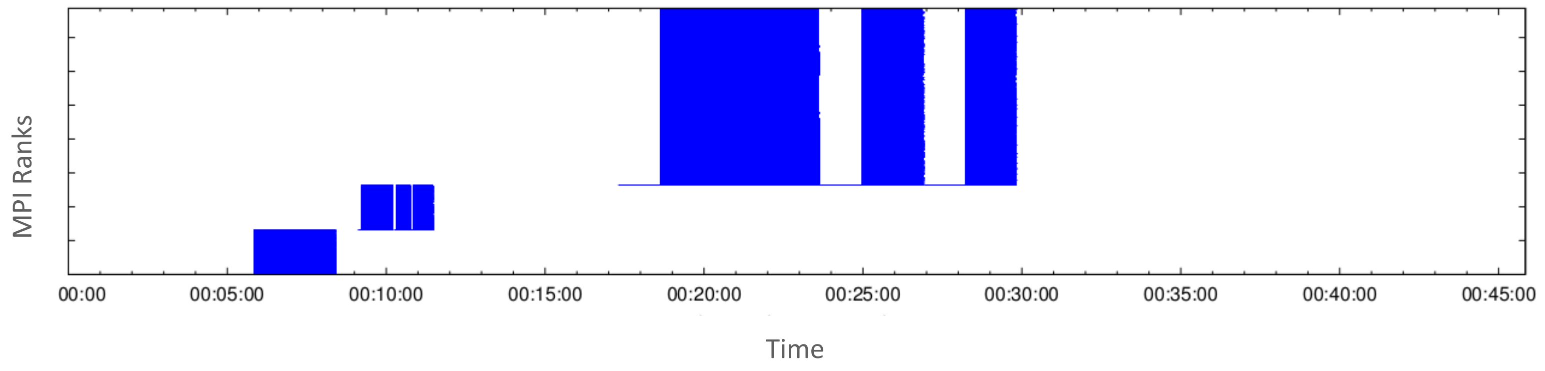
access size	count
16777216	210977
8388608	9866
256	2598
68	9

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	17	199G	1.6T
read-only files	1	2.0K	2.0K
write-only files	13	260G	1.6T
read/write files	0	0	0
created files	13	260G	1.6T

/gpfs/mira-fs0/projects/WallModJet/cwhamman/wavelength/./rbturb.x 11 specs.in

Darshan analysis tool example



This graph (and others like it) are on the second page of the darshan-job-summary.pl output. This example shows intervals of I/O activity from each MPI process.

Using Darshan analysis tools

- See online documentation: http://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html
- Key tools:
  - **darshan-job-summary.pl**: creates pdf file with graphs useful for initial analysis
  - **darshan-summary-per-file.sh**: similar to above, but creates a separate pdf file for each file opened by the application
  - **darshan-parser**: dumps all information into ascii (text) format

Darshan-parser example (see all counters related to write operations):

```
"darshan-parser user_app_numbers.darshan.gz |grep WRITE"
```

See documentation above for definition of output fields

Darshan installation

- The system that we are using (Edison, a Cray XC30 system operated by NERSC) already has Darshan installed and automatically enabled for all users
- What if you want to install Darshan on your own system?

Notes on Darshan installation

- Darshan can be installed:
  - system-wide (available to all users)
  - in a user's home directory (no root access required)
    - There is no difference in functionality

- Two components:
  - darshan-runtime: installed on an HPC system to instrument MPI applications
  - darshan-util: installed on a workstation to analyze Darshan log files (log files themselves are portable)

- darshan-runtime installation steps vary depending on the platform
  - Cray:
    - preferred method uses Cray modules
  - MPICH-based systems with static linking (e.g. Blue Gene):
    - preferred method uses utilities to generate augmented compiler scripts
  - systems with dynamic linking:
    - LD_PRELOAD to add instrumentation at run time

- darshan-util installation is generic for almost any unix-like platform

This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract Nos. DE-AC02-06CH11357 and DE-AC02-05CH11231 including through the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization.