Shared Library Performance on Hopper

Zhengji Zhao[1)], Mike Davis[2)], Katie Antypas[1)], Yushu Yao, Rei Lee[1)] and Tina Butler[1)]

1) National Energy Scientific Computing center

2) Cray, Inc.

CUG 2012, 5/3/2012, Stuttgart, German

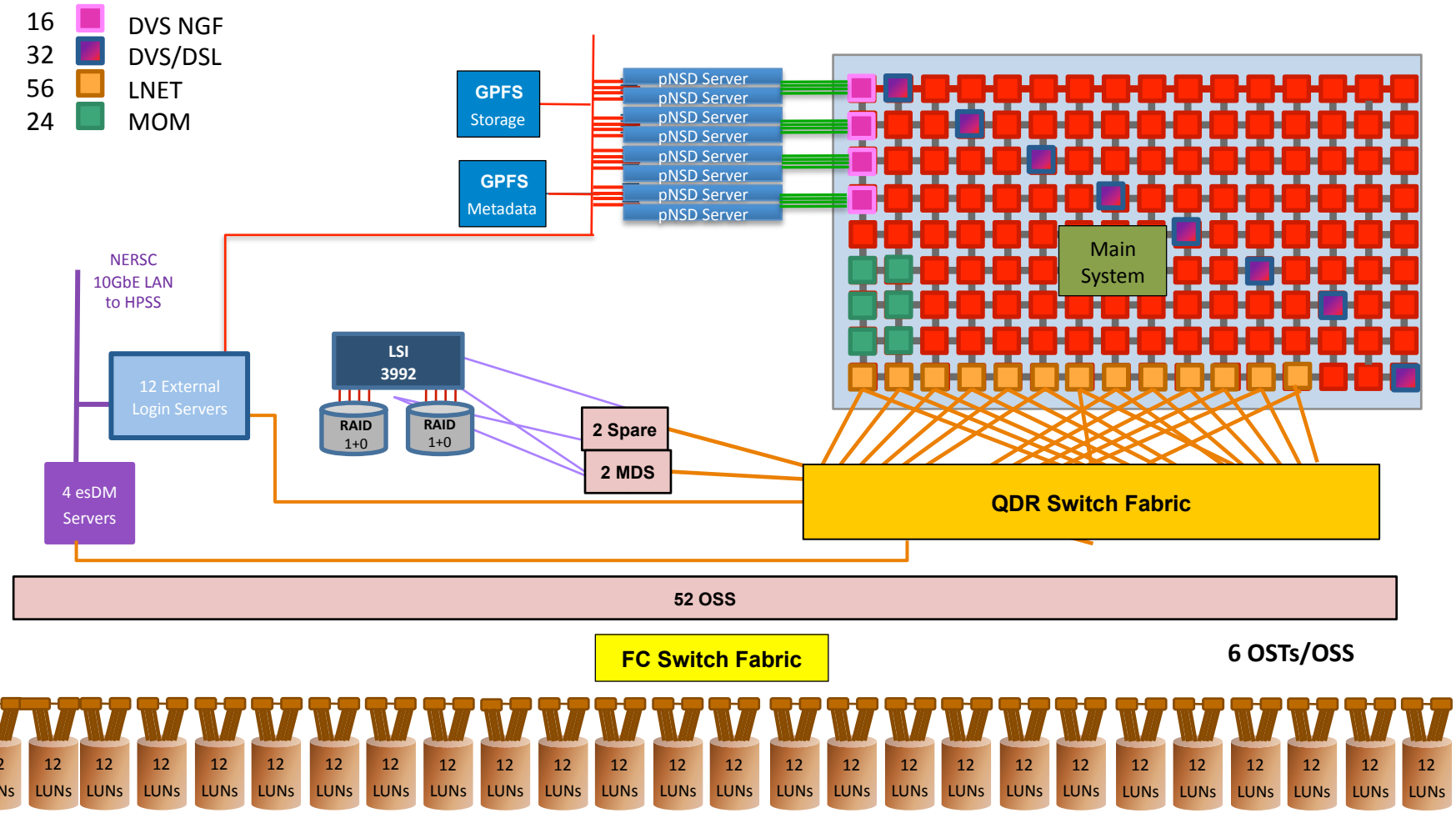
U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC National Energy Research Scientific Computing Center

BERKELEY LAB Lawrence Berkeley National Laboratory

Shared libraries are supported on Hopper compute nodes through DVS

Cray XE6, 6,384 nodes, 153,216 cores, 1.28 Petaflop/s peak



This has enabled shared library applications on Hopper which is one of the important workloads at NERSC.

Applications using shared libraries on Hopper have huge startup time

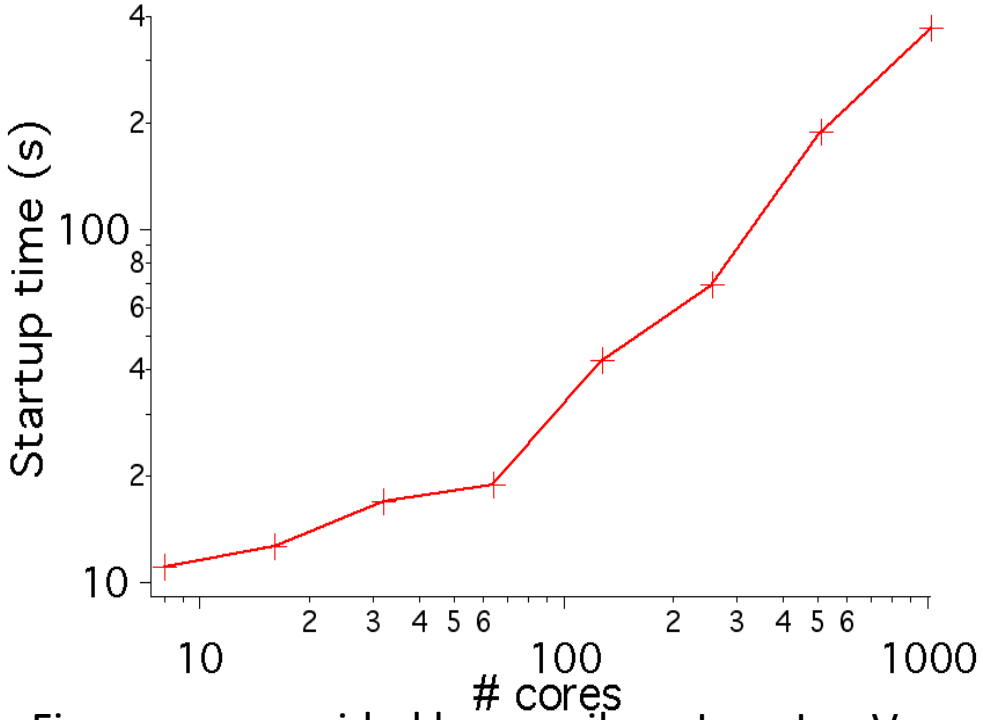


Figure was provided by user jlvay, Jean-Luc Vay

*) Starup time shown here is the module import Time only

*) User stored the shared libraries and python modules on /home file systems.

User Jean-Luc Vay:

"For 64 cores and up, the startup time scales linearly with the number of cores. It is close to 400 seconds for 1,024 cores, which means that a run with 8,000 cores will approach 1 hour of startup time and a run with 64,000 cores might take as much as 8 hours to start."

User David Grote:

"We would like to be able to run with up to 40,000 cores. "

Outline

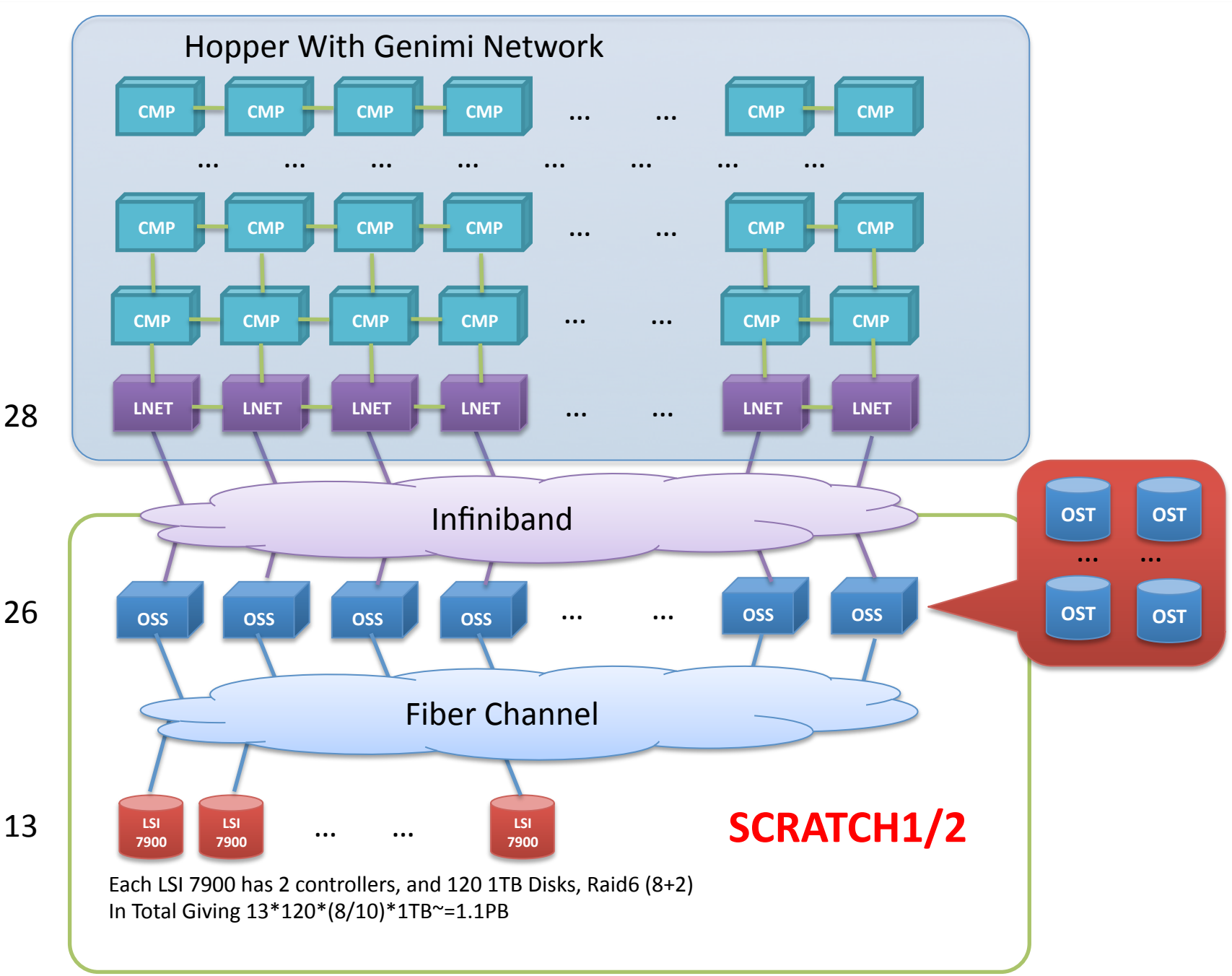
- Motivation
  - Analyze the shared library performance on Hopper and find a workaround to the huge library startup time, so users can run python applications at scale on Hopper.
- Benchmark codes: Pynamic and Warp
- Options and methods explored
  - Test on 5 different file systems with/without the using shared root file system
  - Compare the RPATH and /opt/cray/lib64
  - Shorter LD_LIBRARY_PATH and fewer user shared lib files
  - Custom python import function
  - DLcache and FMcache
- NERSC recommendation to users
- Conclusion

Benchmark code: Pynamic

- Pynamic is a python benchmark
  - Designed to test a system's ability to handle the Dynamic Linking and Loading requirements of Python-based scientific applications.
  - https://computation.llnl.gov/casc/Pynamic/pynamic.htm
- Pynamic configured with
  - 495 total shared objects
  - 280 python import statement
  - 215 utility libraries
- PyMPI integrates MPI into python interpreter
  - pynamic-pyMPI: links all shard objects to pyMPI binary at link time
  - pyMPI: vanila pyMPI, imports modules dynamically
- Total startup time for Pynamic:
  - Launch time (startup time) + module import time + module visit time

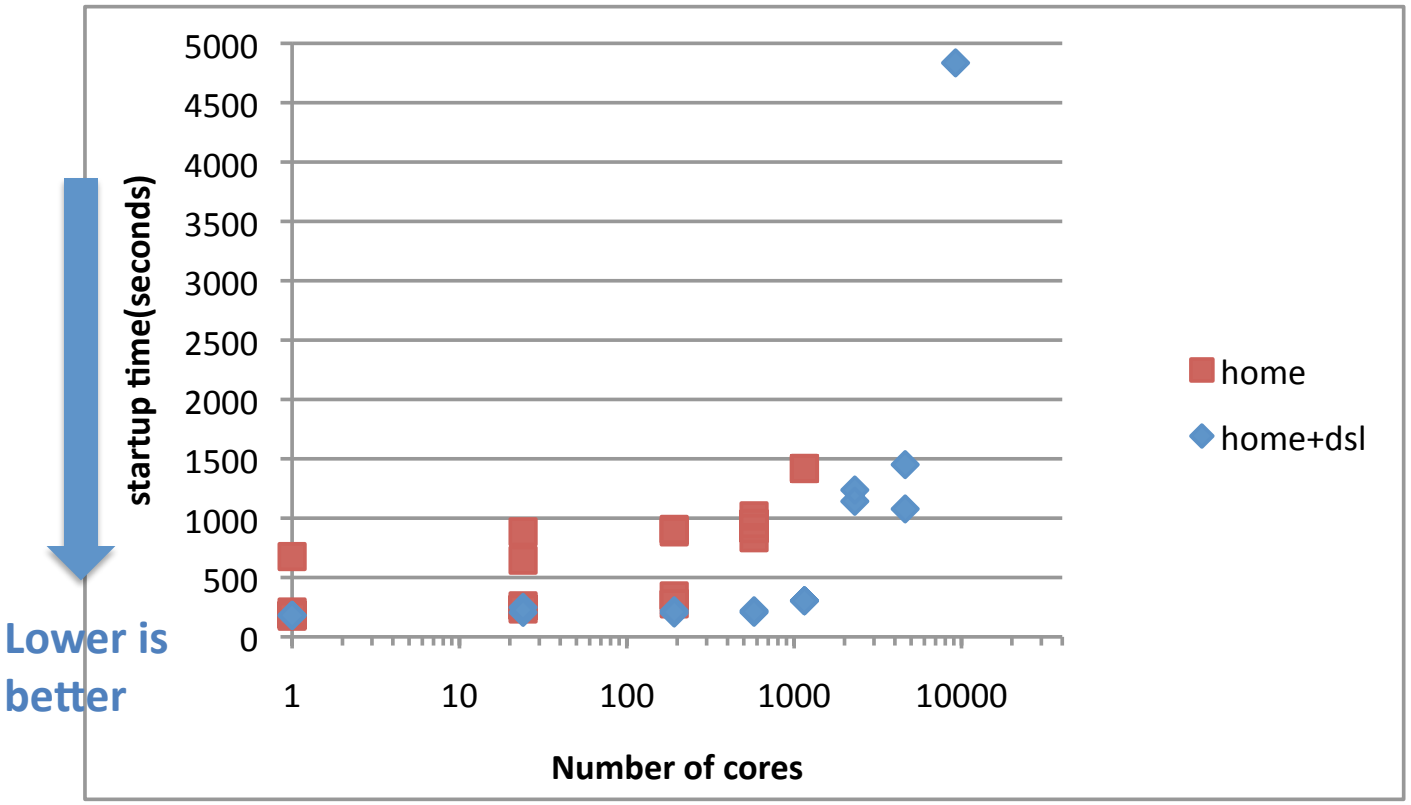
Five file systems on Hopper

- /scratch1/2- for production runs, especially data intensive runs
  - Lustre, 1.1PB, 35GB/s each. Local to Hopper
  - Hopper—28 LNETs—IB—52 OSS's—FC—13 LSI 7900
- /project - users store source code, binaries, data files to be shared between groups of users
  - GPFS, 1.4PB, 15Gb/s. Shared by all NERSC systems
  - Hopper—14 DVS–IB—8 pNSDs—FC—5 DDN 9900
- /global/scratch – for production runs
  - GPFS, 1.1PB, 15Gb/s. Shared by most NERSC systems
  - Hopper—14 DVS—IB—8 pNSDs—FC—5 DDN 9900
- /home - users store source code, binaries, data files …
  - GPFS, 40TB, 1.5GB/s. Shared by most NERSC systems
  - Hopper—DVS—2 NET servers—10GE—4 NSDs—FC—2 HDS AMS 2300
- /usr/common – store NERSC provided software
  - GPFS, 2.3TB. Shared by most NERSC systems
  - Hopper—DVS—2 NET servers—10GE—4 NSDs—FC—2 HDS AMS 2300



Note: There are two sets of identical configuration for SCRATCH1 and SCRATCH2

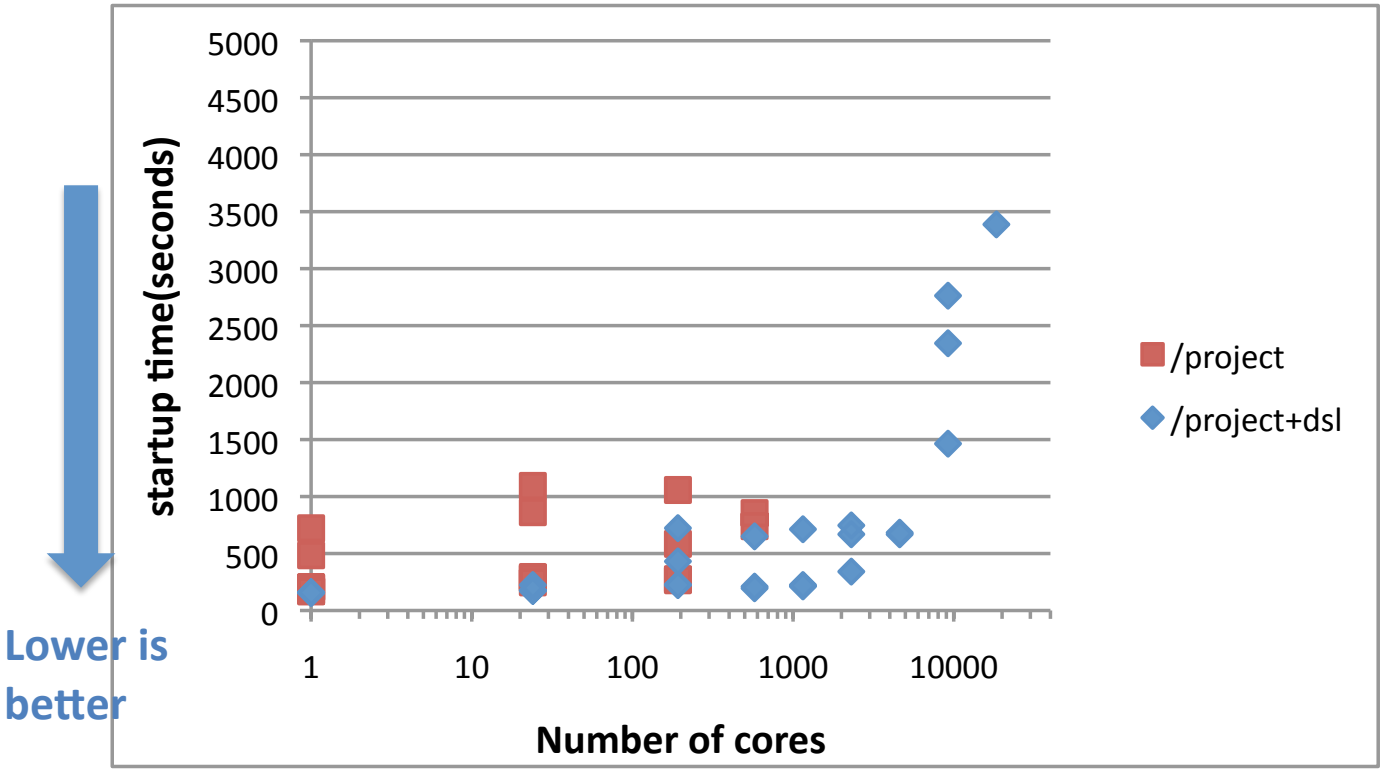
Comparison of Pynamic performance with and without using shared root file system in /home



/home: all shared libs reside in the global /home file system
/home+dsl: user shared libs reside in /home, system provided shared libs in shared root file system (CRAY_ROOTFS=DSL)

- The shared root file system reduces Pynamic startup time
- Without shared root, Pynamic doesn't run above 1000 cores

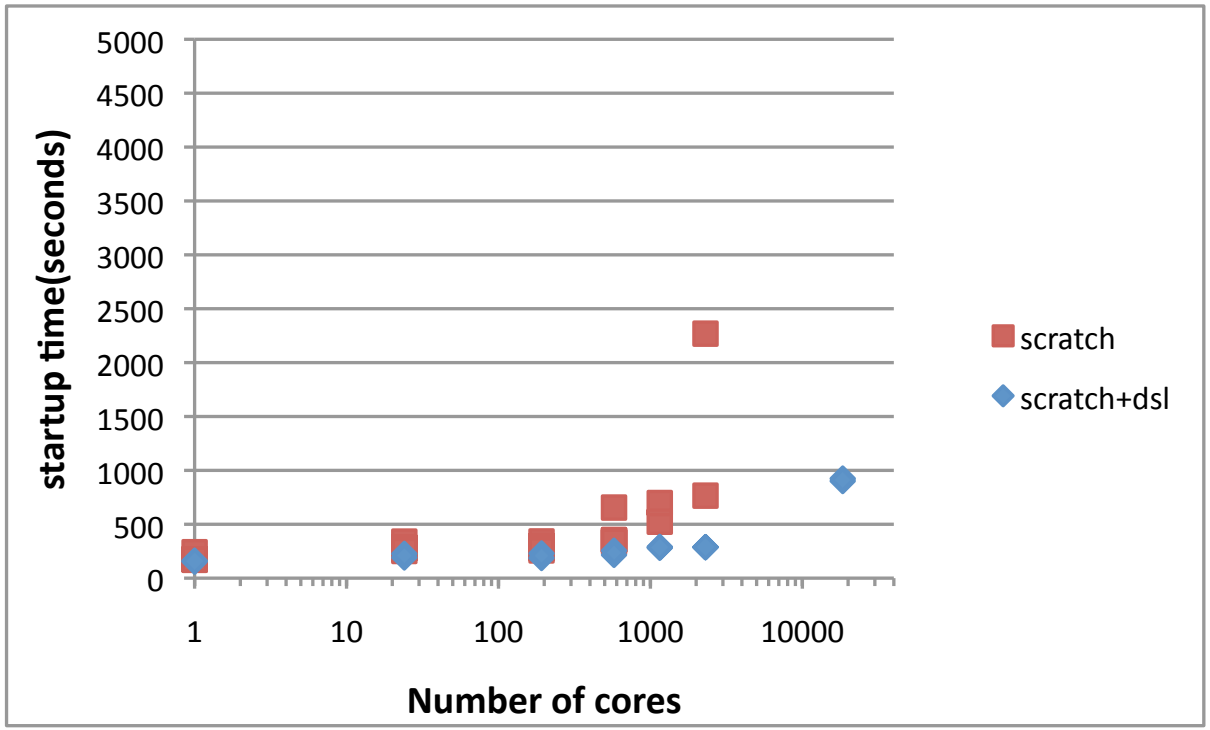
Comparison of Pynamic performance with and without using shared root in /project



/project: all shared libs reside in the /project file system
/project+dsl: user shared libs reside in /project, system provided shared libs in shared root file system (CRAY_ROOTFS=DSL)

- Shared root file system reduces the Pynamic startup time
- Without shared root file system, Pynamic can not run above 4000 cores

Comparison of Pynamic performance with and without using shared root in Lustre /scratch

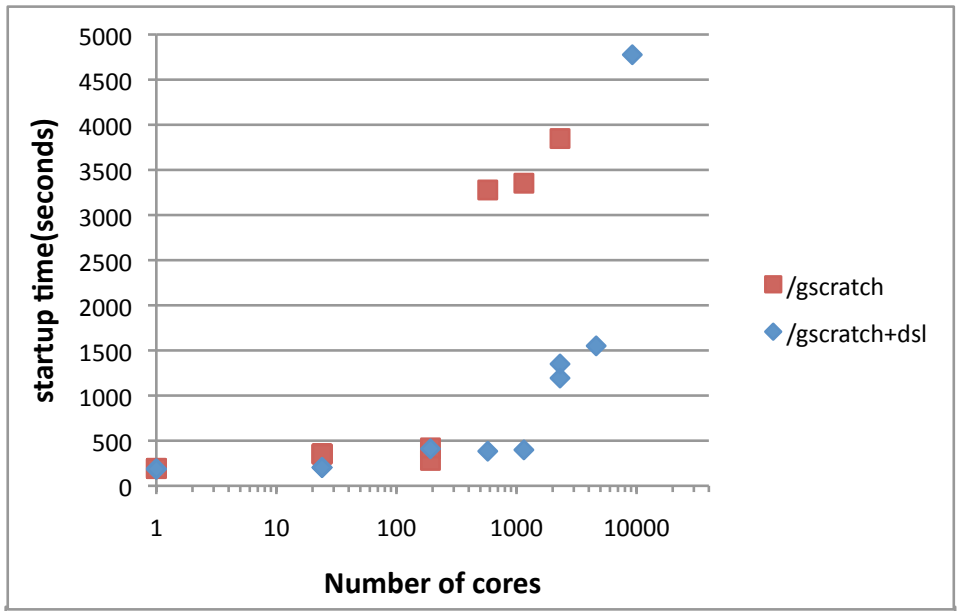


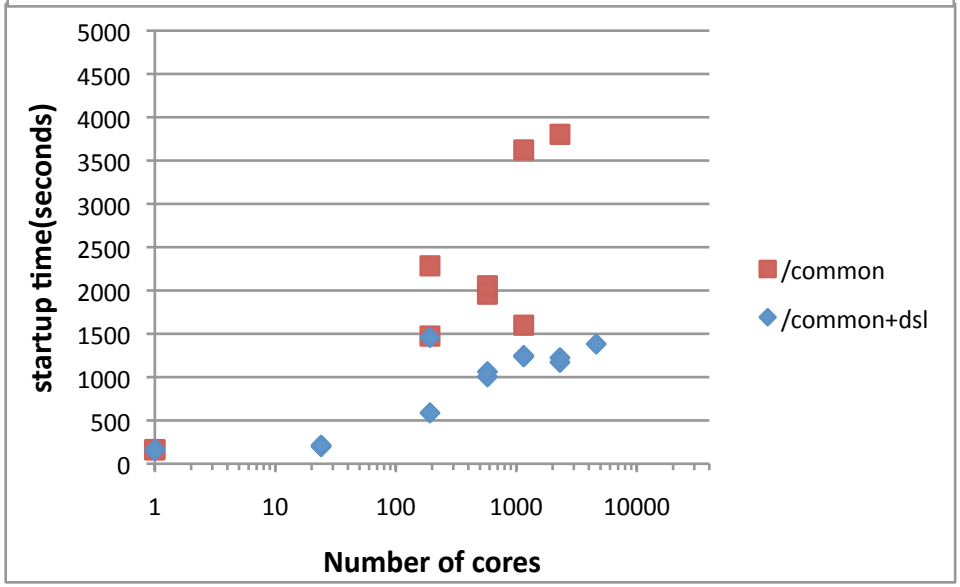
Lower is better

/scratch: all shared libs reside in the /scratch file system
/scratch+dsl: user shared libs reside in /scratch, system provided shared libs in shared root file system (CRAY_ROOTFS=DSL)

- Shared root file system reduces the Pynamic startup time.
- /scratch is the fastest by far

Pynamic performance when shared libraries reside in /global/scratch and /usr/common

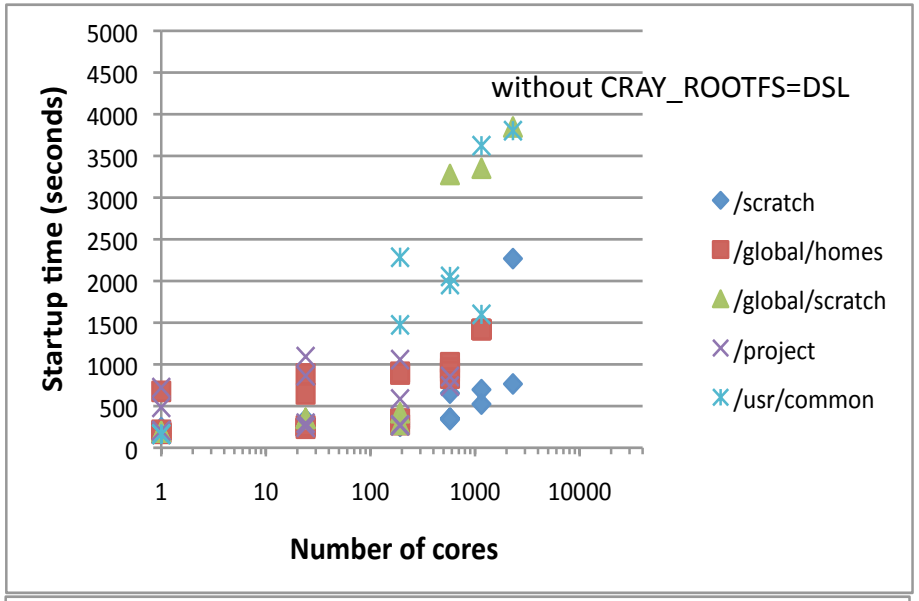


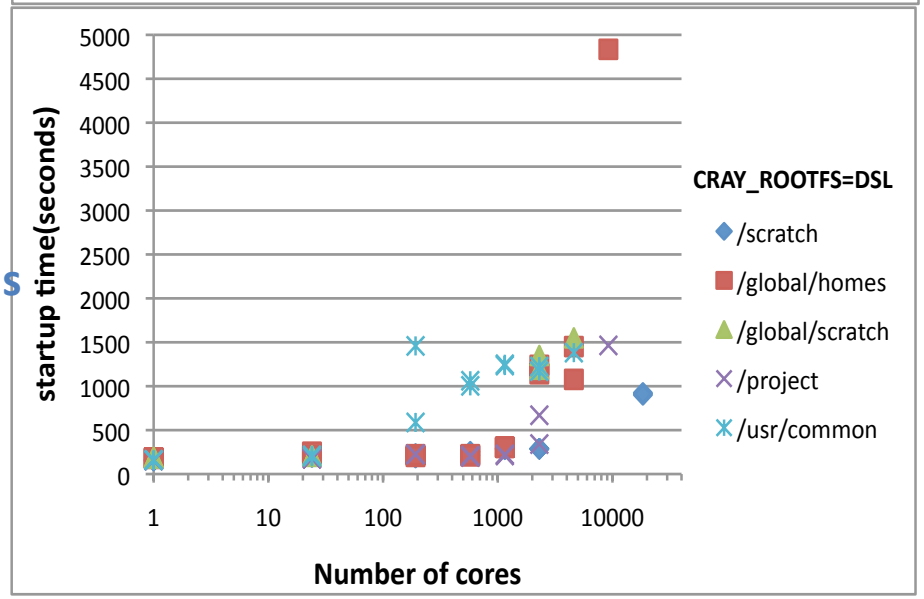


Lower is better

- Shared root file system reduces the Pynamic startup time when the shared libraries reside in /global/scratch and /usr/common.
- Without shared root file system, Pynamic doesn't run above 2000 cores

Across all five file systems, Lustre / scratch performs the best





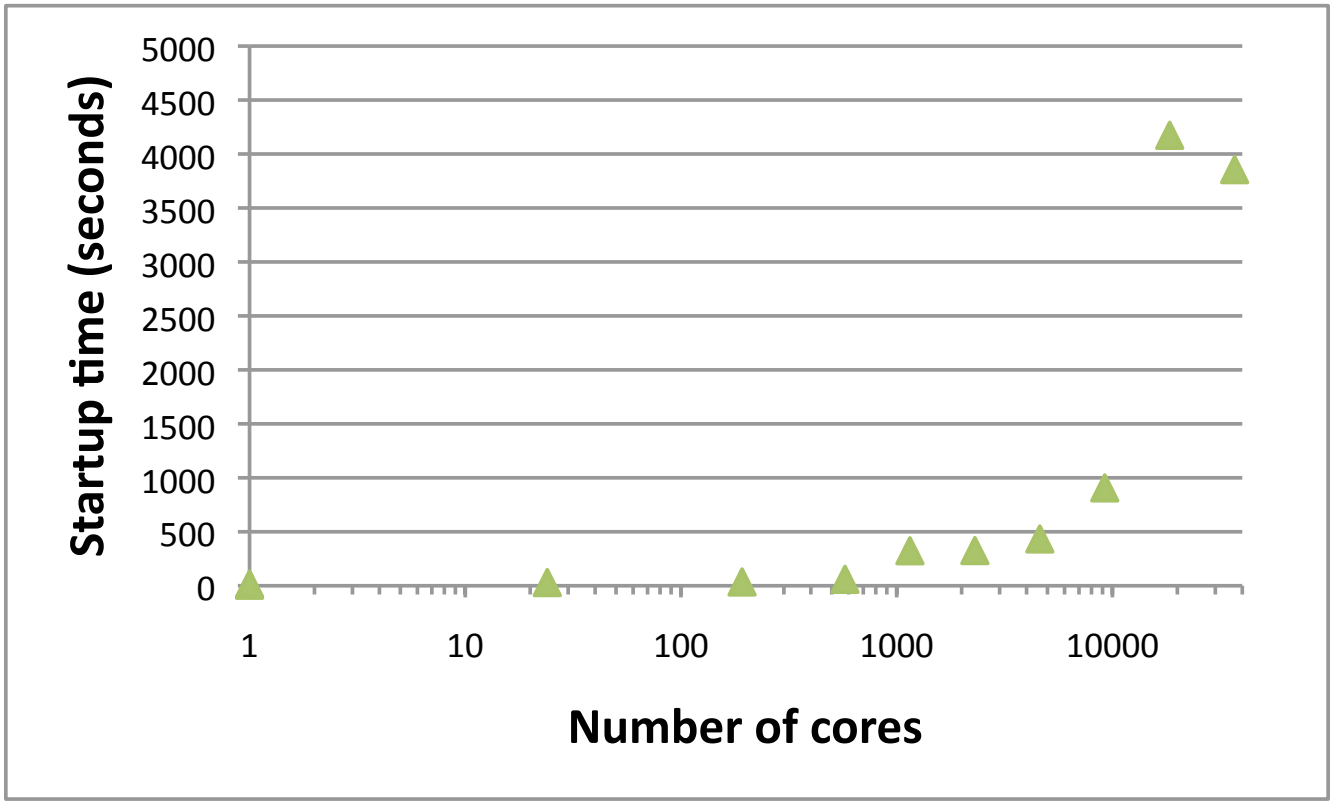
Lower is better

- The shared root file system helps the shared library performance on all 5 file systems.
- /scratch has the shortest total startup time for Pynamic. Thus /scratch is a candidate to store user shared libraries.
- All our following tests were done on /scratch file system with the shared root file system enabled.

User application benchmark: Warp

- Warp: modeling high intensity ion beams,
  - heavy ion fusion accelerator physics studies
  - Discrete particle model= particle-in-cell (PIC) + “lattice” of accelerator elements
  - Written primarily in Fortran90 + MPI, python user interface
  - http://hif.lbl.gov/theory/WARP_summary.html
- Has 48 shared libs, 260 imports
- Warp modules and shared libs reside in /scratch
- CRAY_ROOTFS=DSL
- Total startup time for Warp
  - launch time + module import time

Warp total startup time on /scratch with shared root enabled

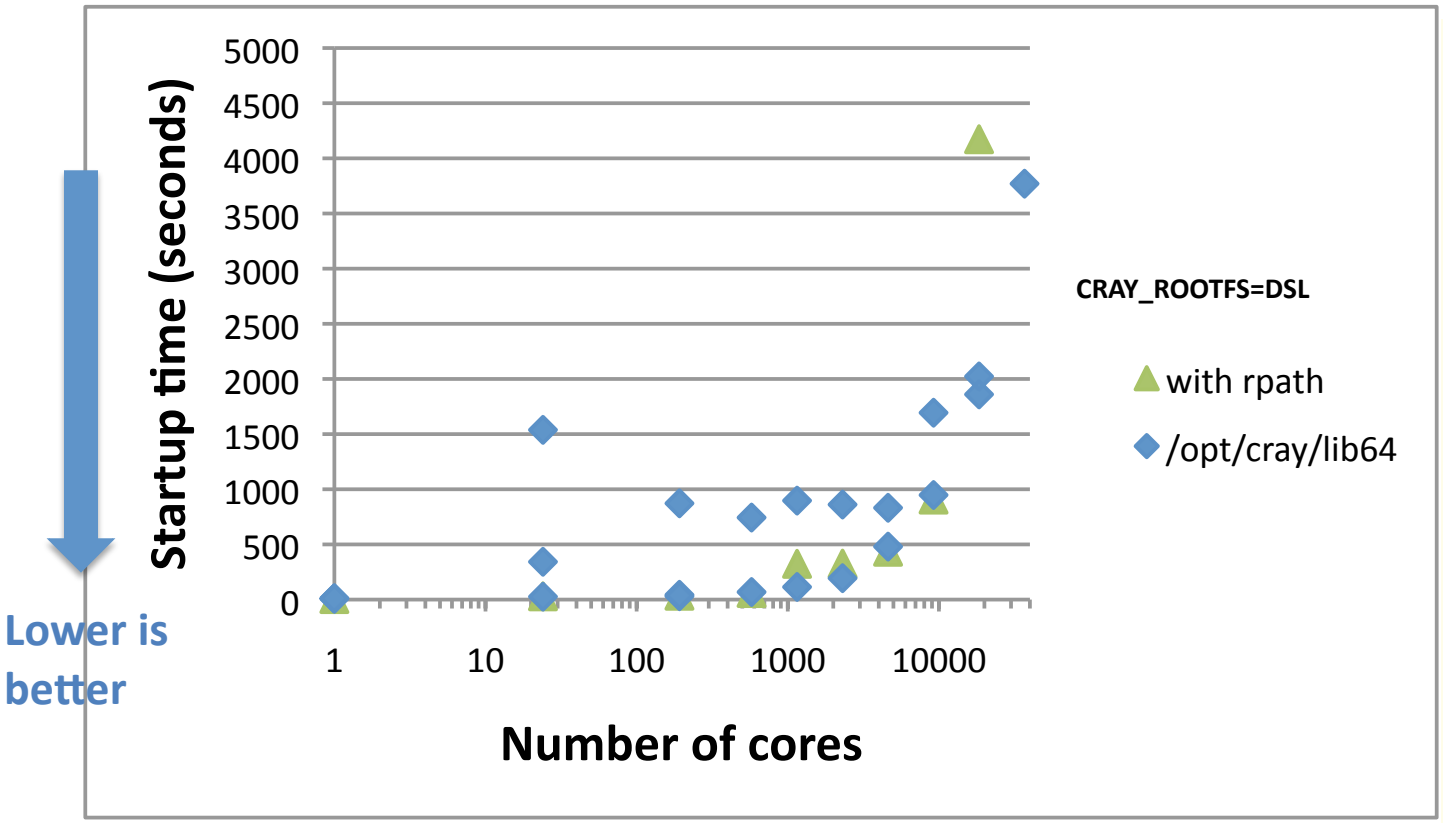


By adopting the "best" file system /scratch to store user shared libraries and python modules, Warp could startup 4 times faster than the original user prediction of 4.5 hours at 36K (user stored their files on /home).

Issues with /scratch:

- heavy contention for IO resources -large fluctuation in runtime
- the file system is subject to purge – not ideal for saving permanent files

Comparison of Warp startup time when shared library search default changes

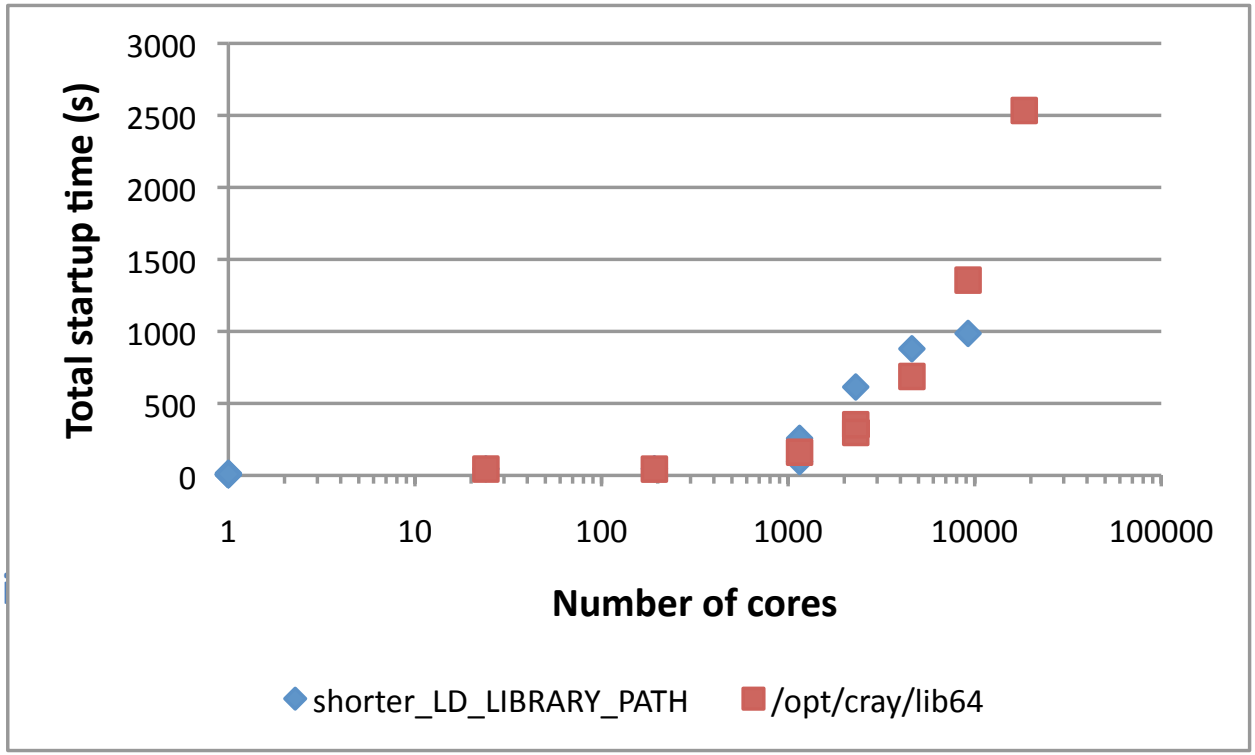


Cray approach:
In the past, RPATH was included in the dynamic executable with a path to the libraries to use at execution time.
Now, at execution time, dynamic shared objects for these libraries will be found in /opt/cray/lib64 using ld.so.cache.

*) Mpich libraries were found from /opt/cray/lib64 in the new approach in Warp code

- Changing execution time dynamic shared library default to /opt/cray/lib64 does not seem to help this specific user application.

Warp startup time with a shorter LD_LIBRARY_PATH

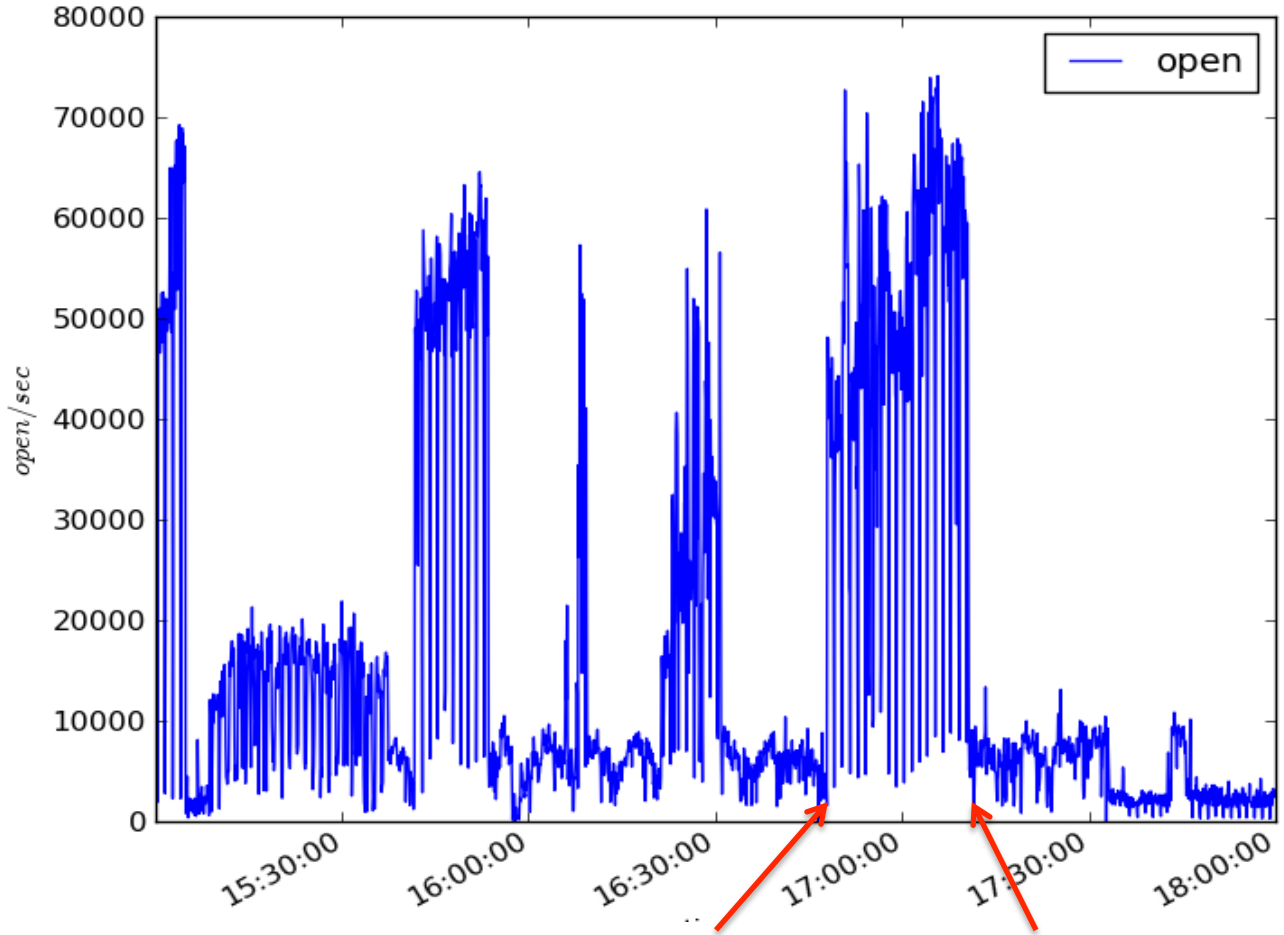


Lower is better

Shorter LD_LIBRARY_PATH: Put all the user shared libraries in one directory, so to reduce the number of search directories for shared libraries. The paths to the system provided libraries stayed the same.

Shorter LD_LIBRARY_PATH does not seem to improve the startup time for large scale runs significantly for Warp.

Lustre metadata server activities are high during large concurrency Python job run

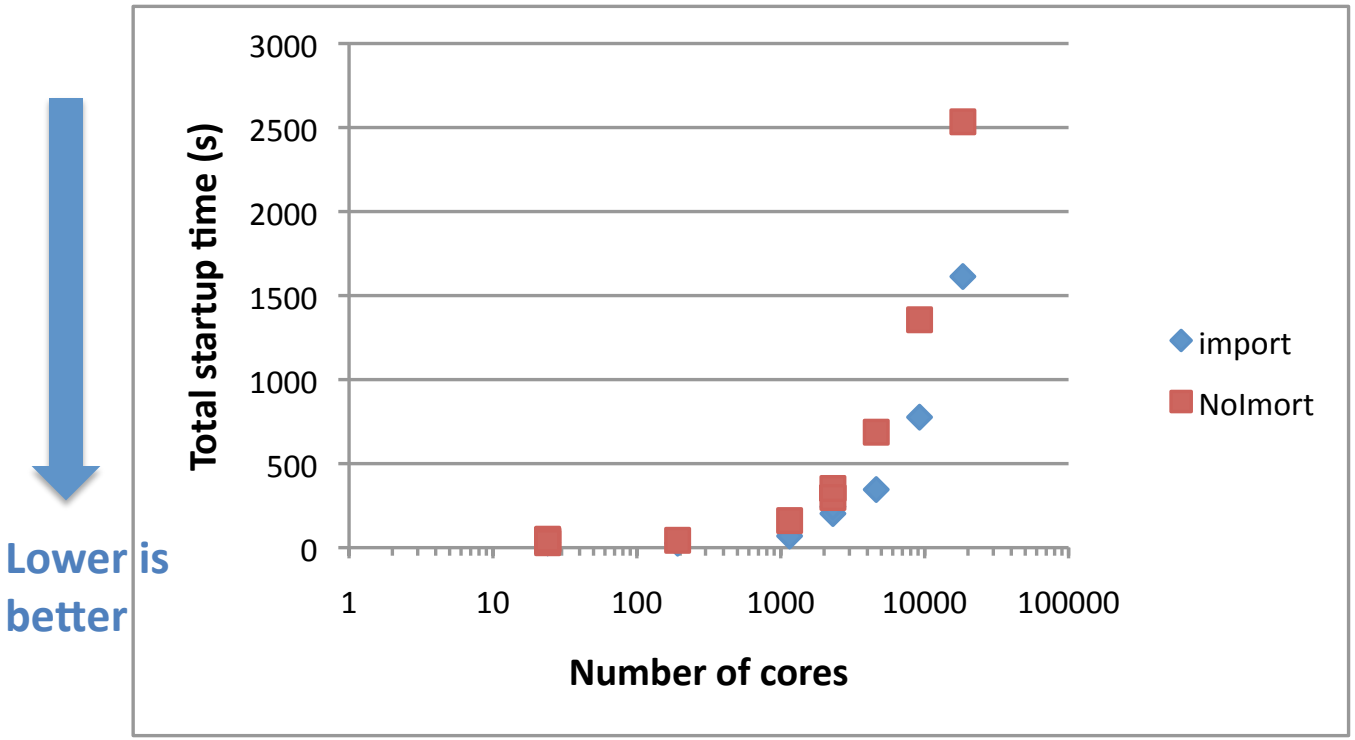


Host	JobID	User	Nodes	Cores	Start	Complete	Wall hrs	MPP hrs
Hopper	906357	zz217	768	18,432	10/17/11 16:47	10/17/11 17:10	0.382	5,276.16

When a Warp job at 18,342 cores was running, a huge IO metadata server activity was observed.

*) Figure is generated by Andrew Uselton's LMT tools

Custom import function significantly reduces total startup time of Warp



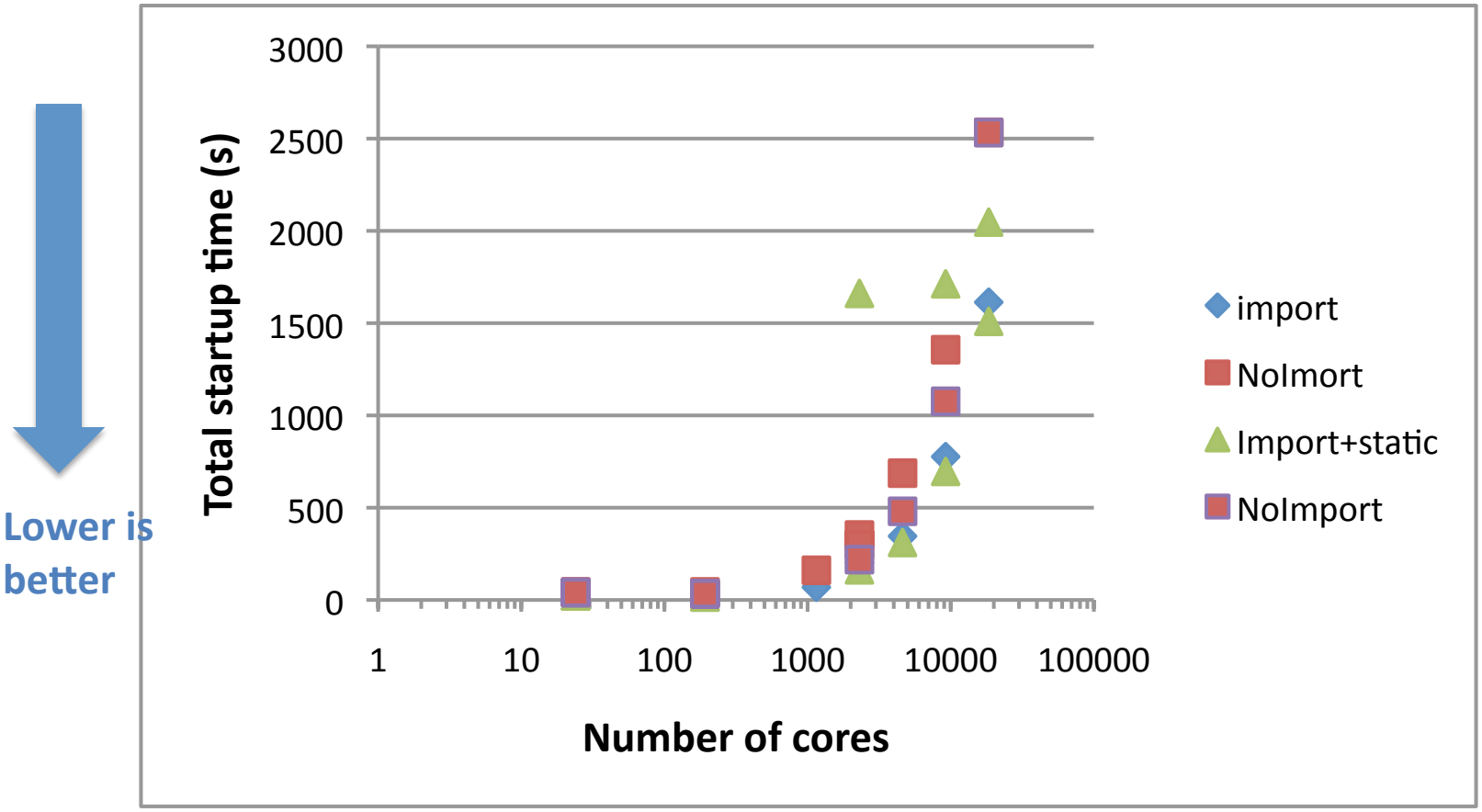
Modified Python/import.c and Python/marshal.c :Only rank 0 calls open(), stat() and read(), and all other ranks get the results of the calls without actually making the calls (ranks 1-N-1).

IO operations each core does during pyMPI startup (.so loading + module import time)

No. of opens	No. of stats	No. of reads
3388/495	1632/848	500

Looks promising …

Warp startup time with custom import + statically linked in MPI libraries



Statically linking MPI libraries to the pyMPI binary seems to help the total startup time of Warp.

DLcache and FMcache*

- DLcache
  - Optimizes the importing of shared-object libraries (.so files)
  - Similar in effect to static linking, but more flexible
- FMcache
  - Optimizes the importing of python modules (.py, .pyc files)
  - Same thing as “custom import” above (just repackaged)
- How to use them
  - Run app once in “trial mode” on small PE-count to write cache files
    - dlcache.dat, fmcache.dat
  - Run app again “for real” on large PE-count, reading cache files
- Operational Assumptions
  - Small-PE and large-PE runs read same .so, .py, .pyc files in same order
  - Each PE reads same .so, .py, .pyc files in same order

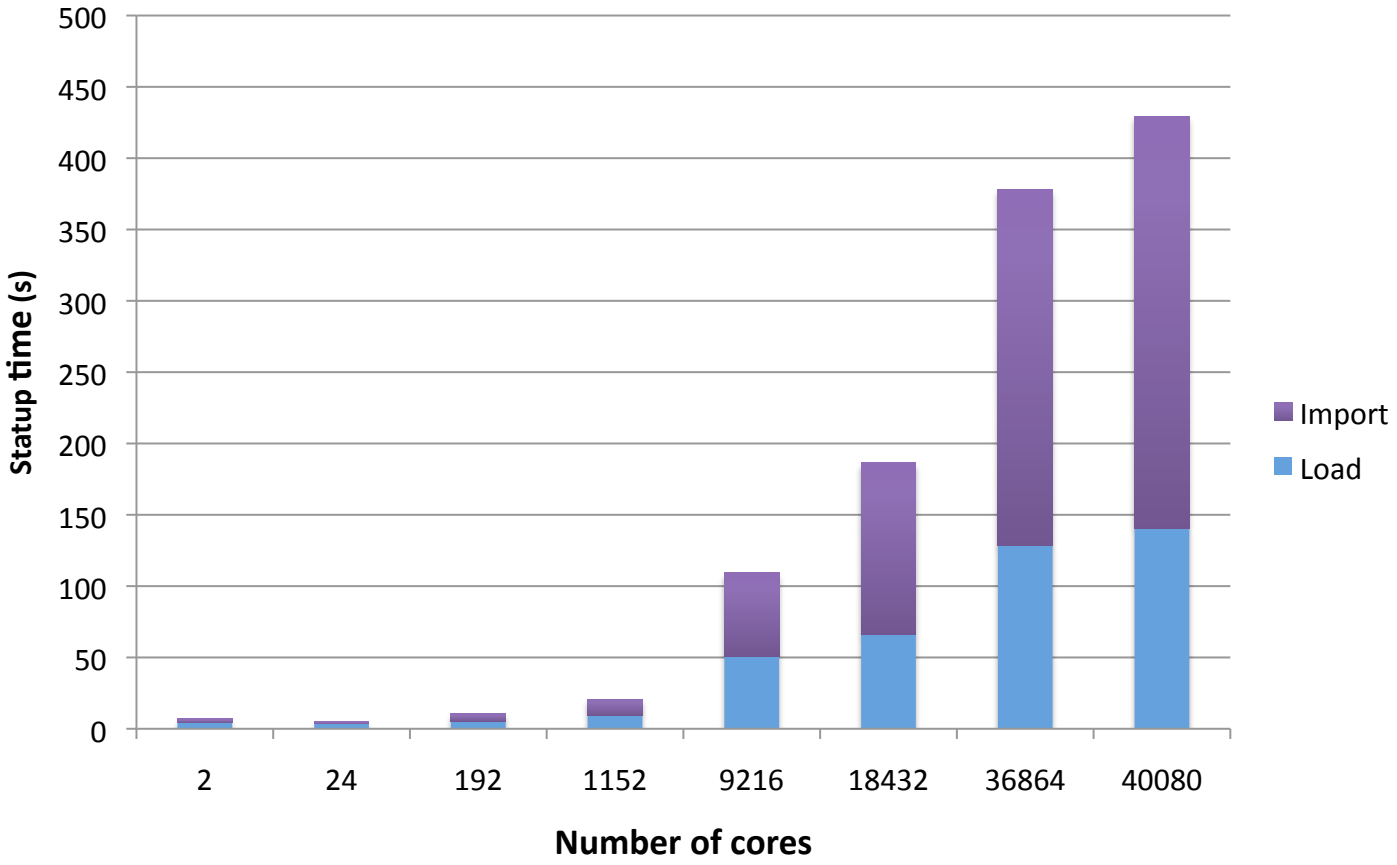
---

* DLcache and FMcache packages are developed by Mike Davis at Cray, Inc

DLcache and FMcache

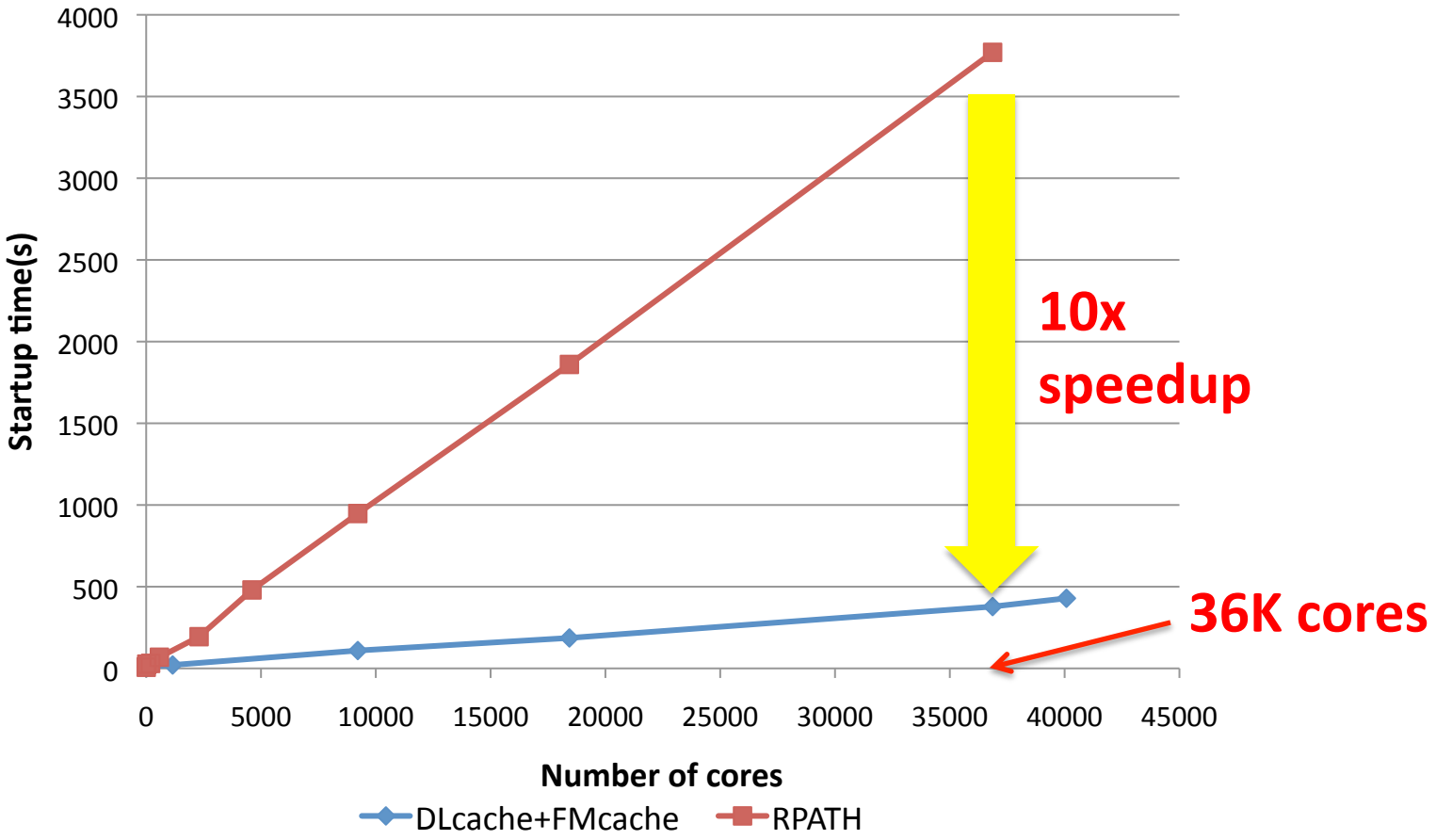
- How do they optimize?
  - Reduce the number of metadata operations (stat, open, close)
    - (N_{PE} * N_{OBJ} * N_{DIR}) becomes (N_{PE}) for DLcache
    - Becomes (1) for FMcache
  - Reduce the number of PEs doing I/O operations (read)
    - (N_{PE}) becomes (1) for FMcache
  - Reduce the cost/contention of metadata, I/O operations
    - dlcache.dat accessed from compute-node /tmp
    - fmcache.dat accessed from rank-0 memory (via MPI_Bcast)
  - Why two caching methods?
    - DLcache executes before MPI is initialized, so cannot use MPI

DLcache and FMcache allow Warp to startup in 7 minutes at 40K cores!



Warp users can run at their desired concurrency 40K within less than 7 minutes of startup time !

DLcache and FMcache reduce Warp startup time 10 times at 36k cores!



Warp startup time comparison to where we started with /scratch file system

- Stored user shared libraries and python modules in /scratch (Lustre) and used shared root.

If compared to the original user prediction, the speedup would be 42 times!

- Stored user shared libraries and python modules in /home (GPFS), and used shared root

NERSC Recommendations to shared library application users

- Shared root file system helps the share lib performance.
- /scratch + shared root file system is a candidate to store user shared libraries, but it is subject to unpredictable run time fluctuation due to heavy IO from other user jobs.
- DLcache and FMcache are most promising methods; custom import has a significant reduction on the startup time; using shorter LD_LIBRARAY_PATH and using /opt/cray/lib64 instead of RPATH seem not have an observable improvement under a production environment.
- Recommendations to shared lib users:
  - Use of DLcache and FMcache
  - Use /scratch system to store user shared libs

Acknowledgement

- Sue Kelly, Sandia National Laboratories
- Sean Zhu, NERSC summer student 2011
- David Grote, Jean-Luc Vay, WARP developers and NERSC users.
- Andrew Uselton, Nick Wright, Thomas Davis, David Skinner as well as NERSC staff in User Services Group