Parallel Software Technologies Laboratory PSTL

Parallel I/O for OpenSHMEM

Edgar Gabriel

Parallel Software Technologies Laboratory,
Department of Computer Science
University of Houston
gabriel@cs.uh.edu

UNIVERSITY of HOUSTON

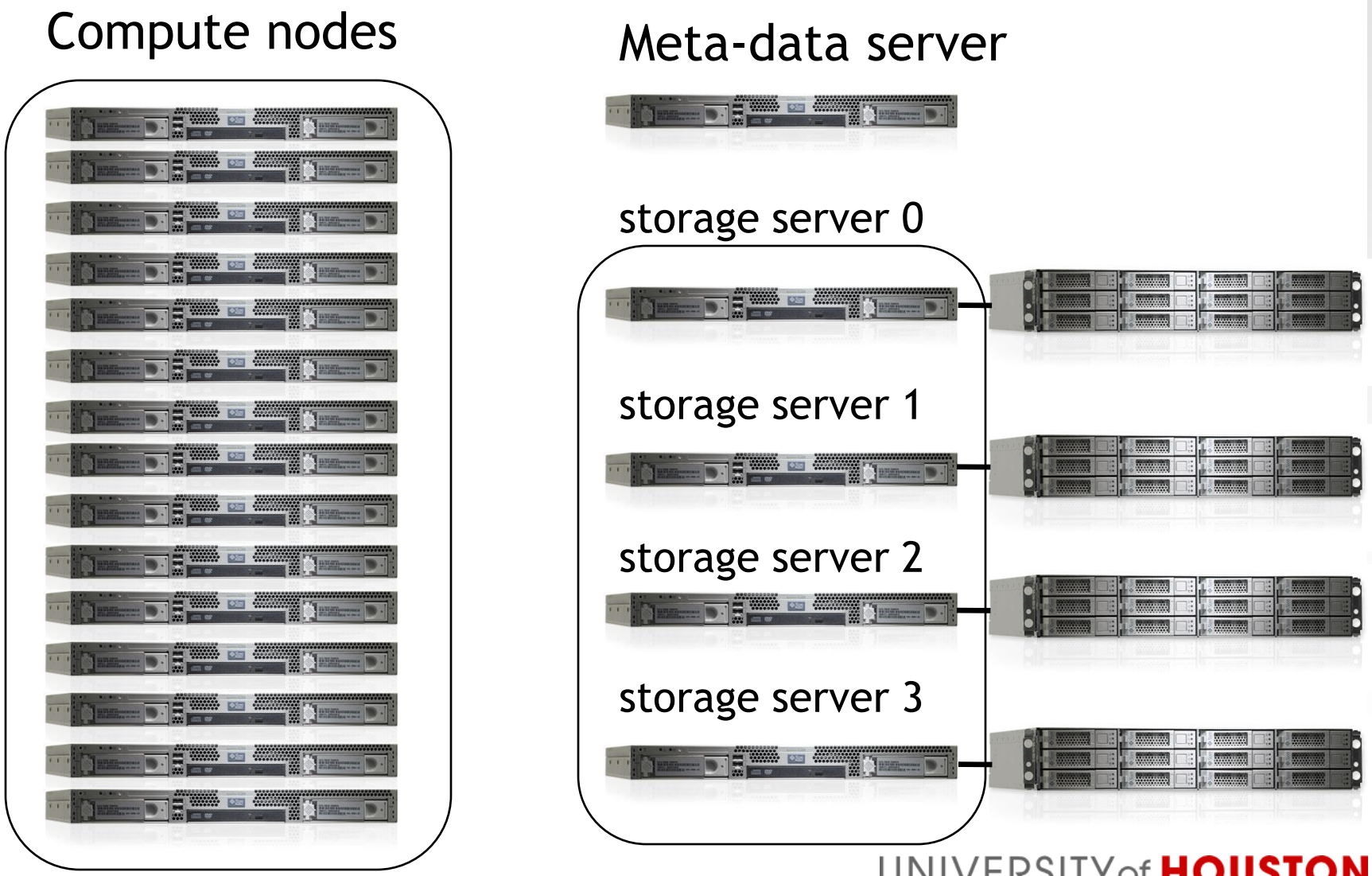
Outline

- Motivation
- Parallel I/O in MPI
- Parallel I/O for OpenMP
- Parallel I/O API for OpenSHMEM
- Conclusions

Motivation

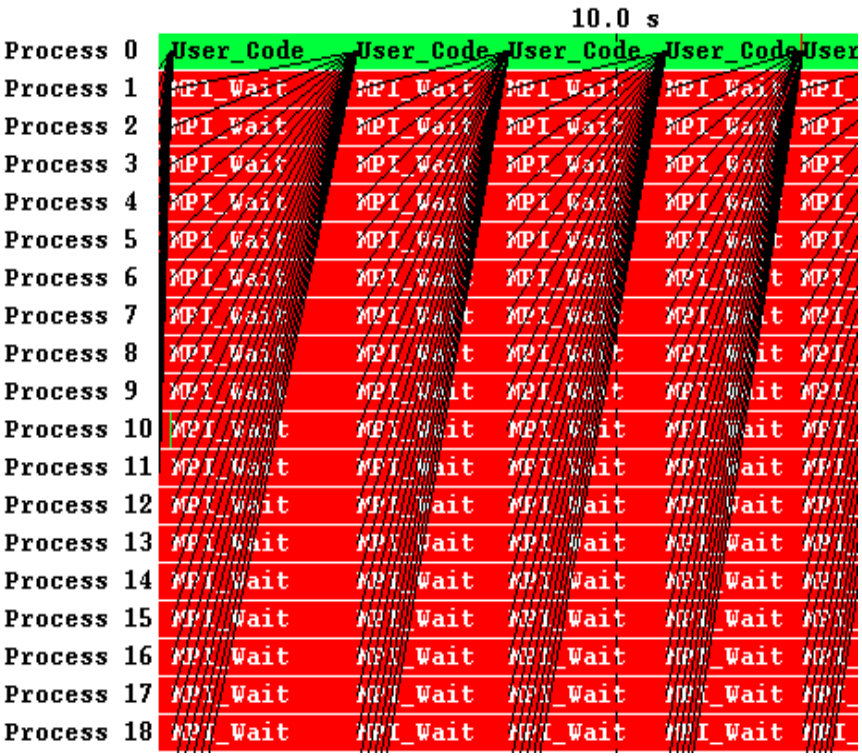
- Study by LLNL (2005):
  - 1 GB/s I/O bandwidth required per Teraflop compute capability
  - On average, 5 times more write operations than read operations
- Current High-End Systems:
  - K Computer: ~11 PFLOPS, ~96 GB/s I/O bandwidth using 864 OSTs
  - Jaguar (2010): ~1 PFLOPS, ~90 GB/s I/O bandwidth using 672 OSTs

Parallel File Systems: Conceptual overview



Application Perspective

- Sequential I/O
  - A single process executes file operations
  - Leads to load imbalance
- Individual I/O
  - Each process operates on separate files
  - Pre/Post-processing required



- Parallel I/O
  - Processes access (different parts of) the same file
  - Consistency and efficiency issues

I/O and data decomposition

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

Process 0:

```
read(…, offset=0,  length=2)
read(…, offset=8,  length=2)
read(…, offset=16, length=2)
read(…, offset=24, length=2)
```

- Individual Read/Write operations lead to many, small I/O requests from each process
- Arbitrary order of I/O requests from the file system perspective

  ⇨ suboptimal performance

Collective I/O in parallel applications

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

Process 0:

```
read(…, offset=0,  length=4)
Send (…,length=2,dest=3,…)
read(…, offset=8, length=4)
Send (…,length=2,dest=3,…)
read(…, offset=16, length=4)
Send (…,length=2,dest=3,…)
read(…, offset=24, length=4)
Send (…,length=2,dest=3,…)
```

- Collective I/O:
  - Fewer and larger I/O requests posted
  - Coordinates I/O access across processes

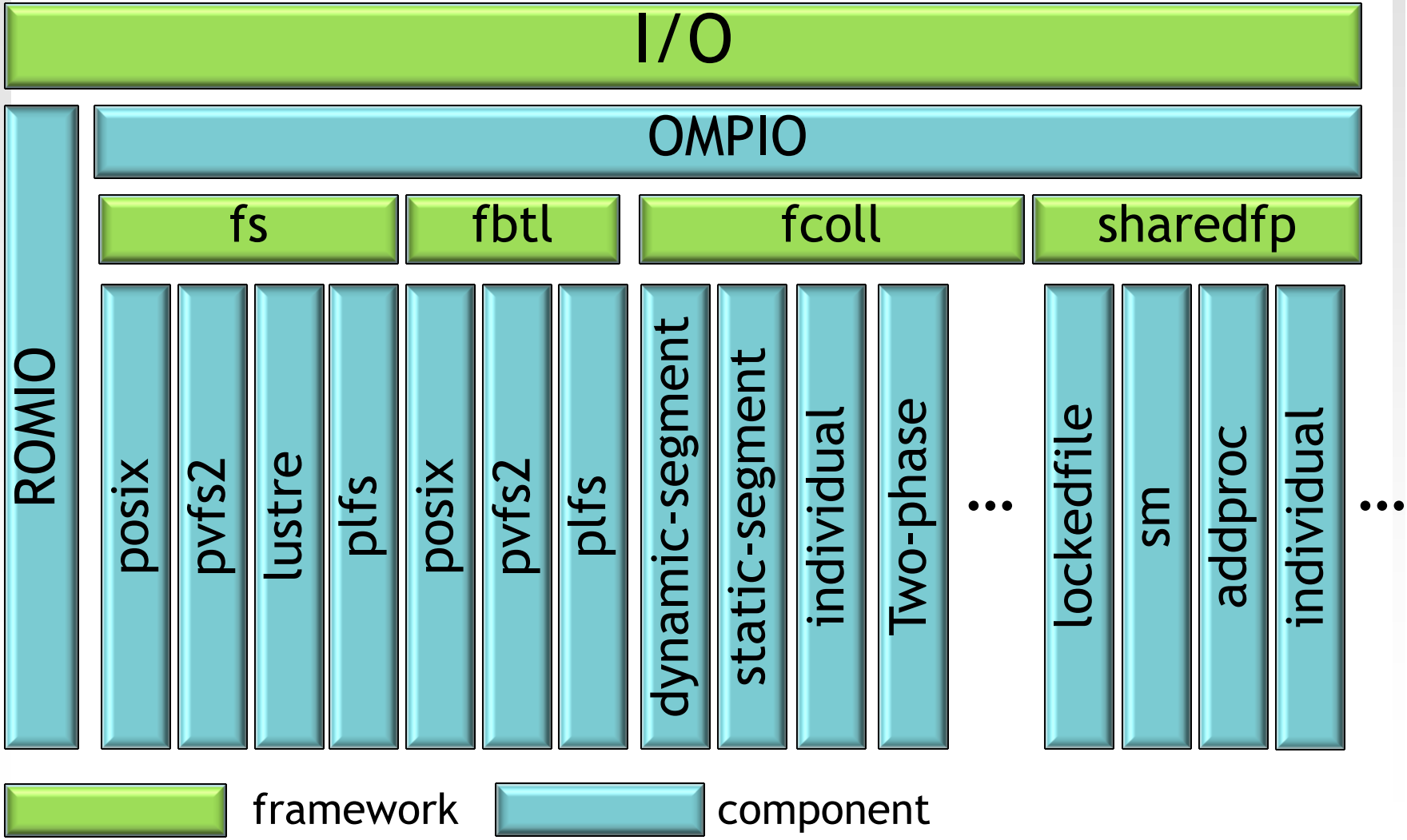
MPI I/O

- MPI (Message Passing Interface) version 2 introduced the notion of parallel I/O
  - **Relaxed consistency semantics**: updates to a file might initially only be visible to the process performing the action
  - **Non-blocking I/O:** asynchronous I/O operations
  - **Collective I/O**: group I/O operations
  - **File view**: registering an access plan to the file in advance

MPI I/O Performance Benefits

Image size / no. of spectral channels	No. of processes	POSIX I/O	MPI Individual I/O	MPI Collective I/O
8192 / 21	48	94.76	65.77	48.56
8192 / 21	64	91.22	71.41	46.4
12281 / 21	72	182.32	116.05	92.06
12281 / 21	80	170.23	115.2	89.99
12281 / 21	88	177.01	117.75	92.06

OMPIO frameworks overview



Collective I/O algorithms

- Combination of I/O and communication used to read/write data from/to file

Data contiguity ← Two-phase I/O | Dynamic segmentation | Static segmentation | Individual → Communication cost

- Collective I/O selection logic: [EuroMPI 2011]
- Automatic selection of number of aggregators: [IASDS 2011]
- I/O access based process placement: [EuroMPI 2013]

No. of procs.	Tile Size	fcoll module	OMPIO bandwidth	ROMIO bandwidth
256	64 Bytes	Two-phase I/O	2167 MB/s	411 MB/s
256	1 MB	Dynamic Segmentation	2491 MB/s	517 MB/s

Parallel I/O for OpenMP

- OpenMP: shared memory parallel programming model
  - Based (mostly) on compiler directives
  - No support for parallel I/O in OpenMP
- Need for threads to be able to read/write to same file
  - Without locking file handle
  - Applicable for all languages supported by OpenMP
- API specification:
  - All routines are library functions (not directives)
  - Routines implemented as collective functions
  - Shared file pointer between threads
- Integration with the OpenUH compiler

OpenMP Parallel I/O Specs

Overview of Interfaces (*read*)

File Manipulation		`omp_file_open_all()`
		`omp_file_close_all()`
Different Arguments	Regular I/O	`omp_file_read_all()`
		`omp_file_read_at_all()`
	List I/O	`omp_file_read_list_all()`
		`omp_file_read_list_at_all()`
Common arguments	Regular I/O	`omp_file_read_com_all()`
		`omp_file_read_com_at_all()`
	List I/O	`omp_file_read_com_list_all()`
		`omp_file_read_com_list_at_all()`

Example Interfaces

int err = ***omp_file_read_all*** ([private'] void* buffer, long length, [shared] int fd, [shared'] int hint)

- Collective file read function, no explicit file offset. Data read from file in order threads' openmp assigned IDs.

int err = ***omp_file_read_list_at_all*** ([private'] void** buffer, long* length, [private'] off_t* offsets, int size, [shared] int fd, [shared'] int hint)

- Collective file read function with list input, threads specify file offset explicitly

- 'hint' tells the library if buffers are contiguous
  - OMP_FILE_CONTIG, OMP_FILE_NONCONTIG, NULL

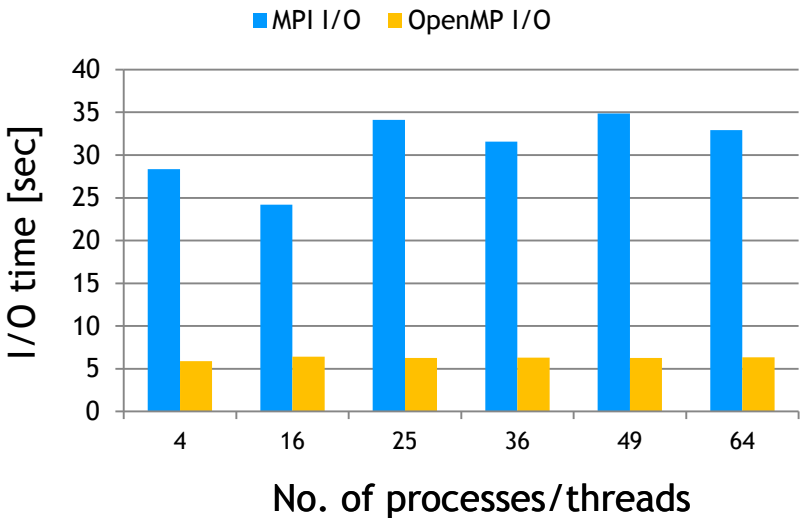
Performance Results

- OpenMP version of the NAS BT Benchmark
- Extended to include I/O operations
- 48-core AMD Magny-Cours node on two PVFS2 files systems

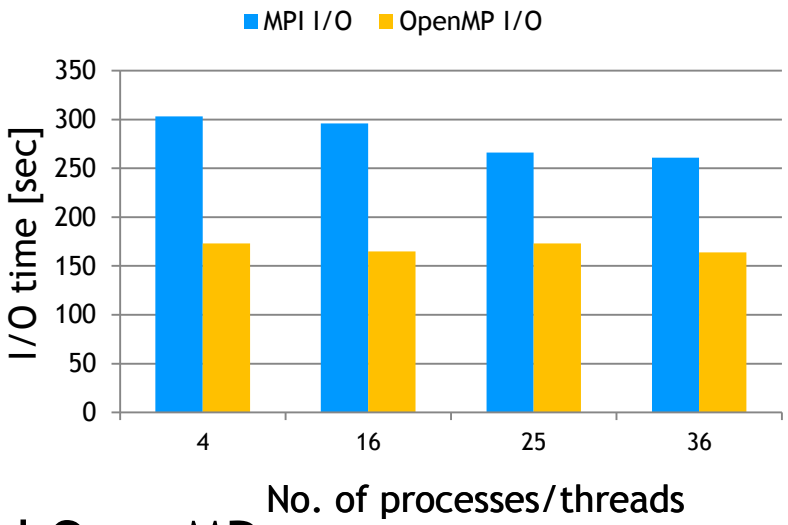
No. of Threads	I/O time on PVFS2 [sec]	I/O time on PVFS2-SSD [sec]
1	410	691
2	305	580
4	168	386
8	164	368
16	176	368
32	172	368
48	168	367

Comparison MPI I/O to OpenMP I/O

Lustre@TU Dresden



PVFS2@UH



- NAS BT Benchmark for MPI and OpenMP
- Single node performance
- OpenMP I/O outperforms MPI I/O
  - Ghostcells in MPI ⇨ discontiguous access
  - OpenMP I/O uses internally PLFS on Lustre ⇨ reduces locking contention

OpenSHMEM Individual I/O Interfaces

Offset into file

```
void shmem_double_write ( off_t t_start, const double
*source, size_t nelemens, int *file);
```

File identifier

```
void shmem_double_iwrite ( off_t t_start, const double
*source, ptrdiff_t tst, ptrdiff_t sst, size_t nelems, int
*file);
```

Stride in file

Stride at source PE

- Would love to have another count argument to specify the number of elements per block

OpenSHMEM Collective I/O Interfaces

```
void shmem_double_write_all ( off_t t_start, const
double *source, size_t nelems, int PE_start, int
logPE_stride, int PE_size, int* file, long *pSync);
```

```
void shmem_double_iwrite_all (off_t t_start, const
double *source, ptrdiff_t tst, ptrdiff_t sst,
size_t nelems, int PE_start, int logPE_stride, int
PE_size, int *file, long *pSync);
```

Comparison of Approaches

Stand Alone Library

- No need for separate interfaces for every programming model
- Easier transition from one programing model to another

Language Specification

- Allows to exploit specific properties of a specification
  - MPI: Derived datatypes, communicators
  - OpenMP: private variables
  - OpenSHMEM: active set of PEs, symmetric heap?

Summary

- I/O of paramount importance for large scale applications and architecture
- Parallel I/O interfaces useful for
  - Performance
  - Exploiting features of the specification
- Parallel I/O API for OpenSHMEM doable
- Prototyping within Open MPI next