OMPIO: a modular architecture for parallel I/O

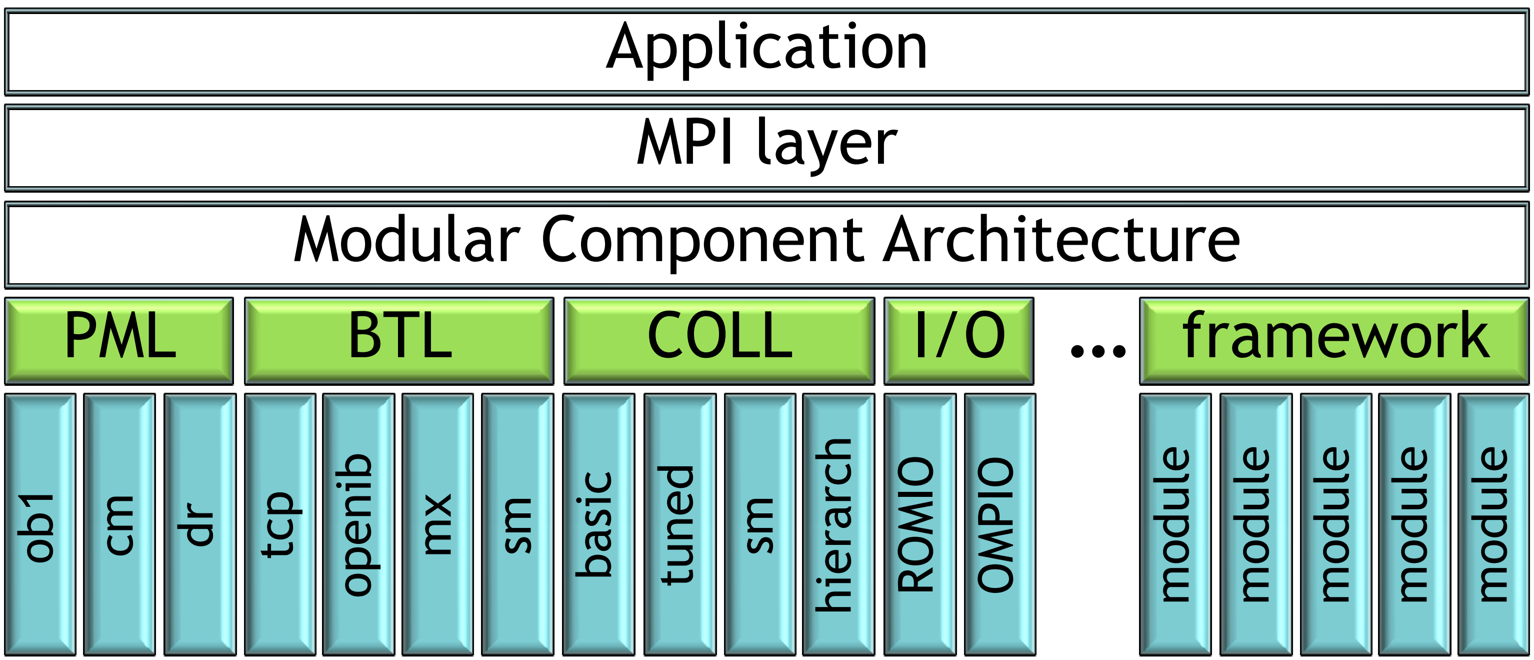
Edgar Gabriel

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

Contributors

- University of Houston:
  - Mohamad Chaarawi
  - Suneet Chandok, Ketan Kulkarni
- Oak Ridge National Laboratory:
  - Rainer Keller, Richard Graham
- University of Tennessee:
  - George Bosilca

Open MPI overview



OMPIO Design Goals (I)

- Highly modular architecture for parallel I/O
  - e.g. separate individual and collective I/O operations
    - some collective I/O algorithms only useful for certain hardware configurations
  - selection of alternatives not necessarily based on the file system utilized
    - shared file pointer operations
    - caching strategy

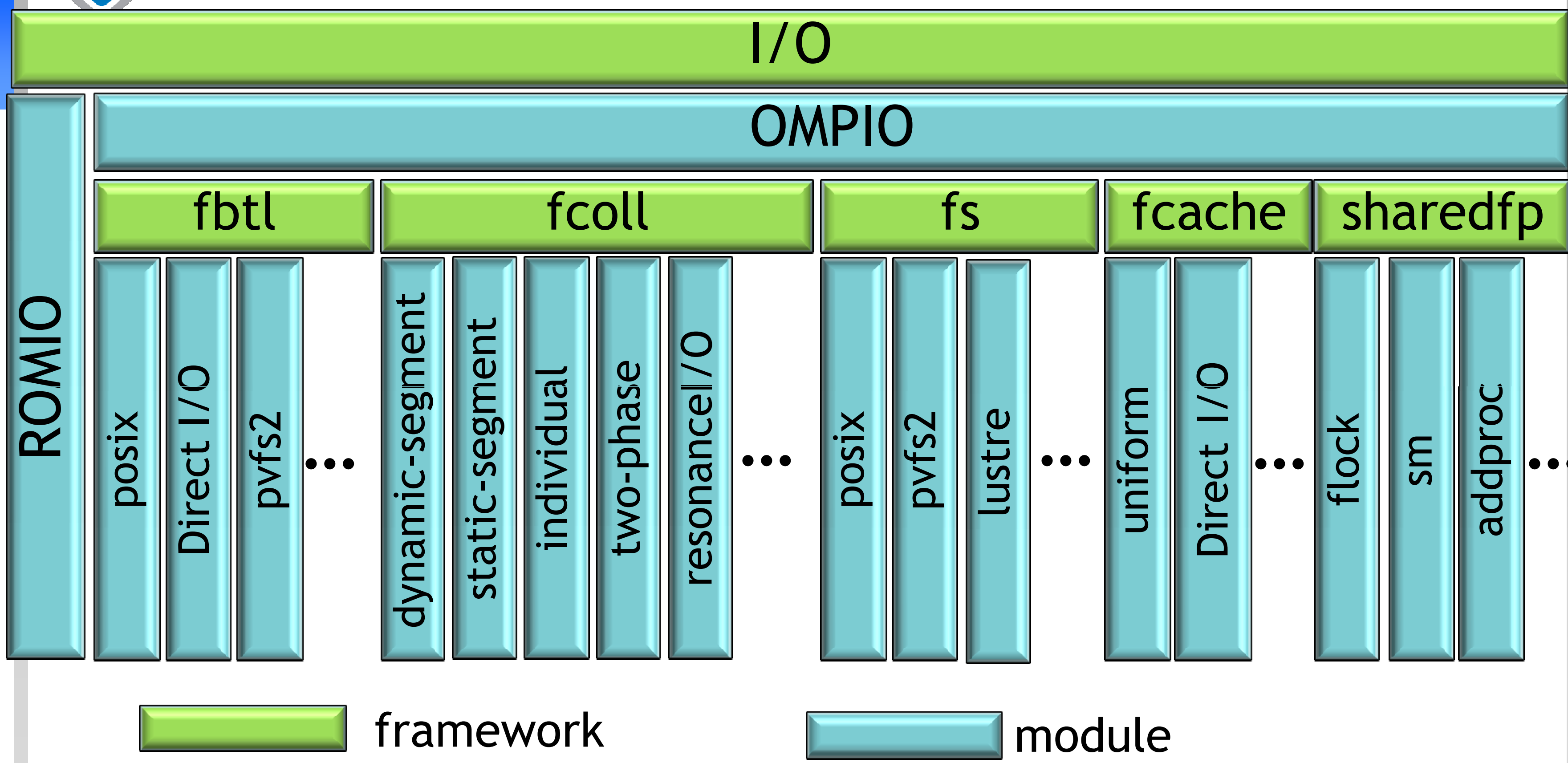
OMPIO Design Goals (II)

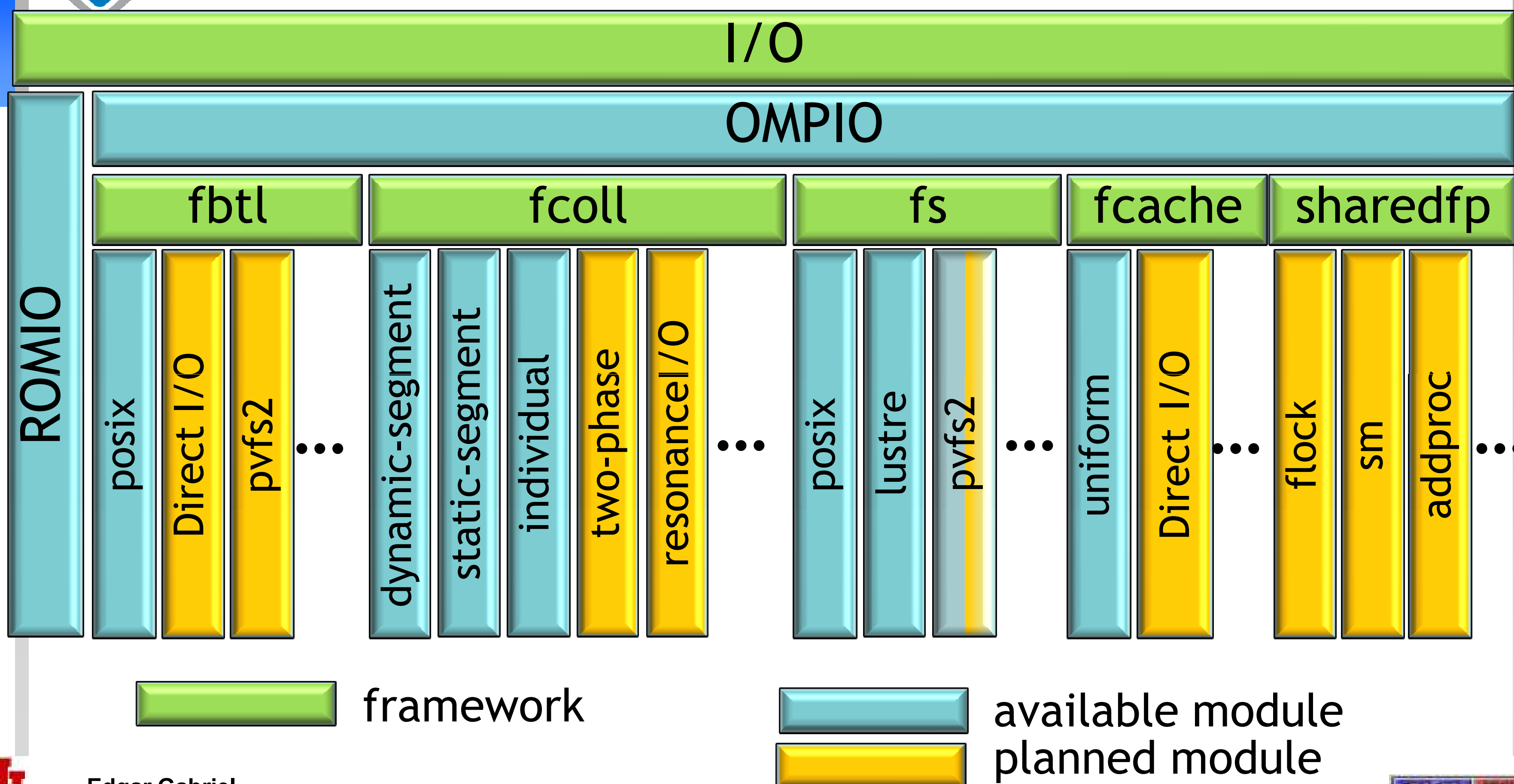
- Tighter Integration with Open MPI library
  - derived data type optimizations
  - data conversion functionality
  - progress engine for non-blocking I/O operations

  - ease the modification of parameters of a given module
  - ease the development and dropping of new modules

OMPIO Design Goals (III)

- Adaptability
  - enormous diversity of I/O hardware and software solutions
    - number of storage server, bandwidth of each storage server
    - network connectivity
      - in-between I/O nodes
      - between compute and I/O nodes
      - message passing network between compute nodes
  - ease the modification of parameters of a given module
  - ease the development and dropping of new modules


I/O
OMPIO
ROMIO
fbtl
fcoll
fs
fcache
sharedfp
posix
Direct I/O
pvfs2
dynamic-segment
static-segment
individual
two-phase
resonancel/O
posix
pvfs2
lustre
uniform
Direct I/O
flock
sm
addproc
framework
module

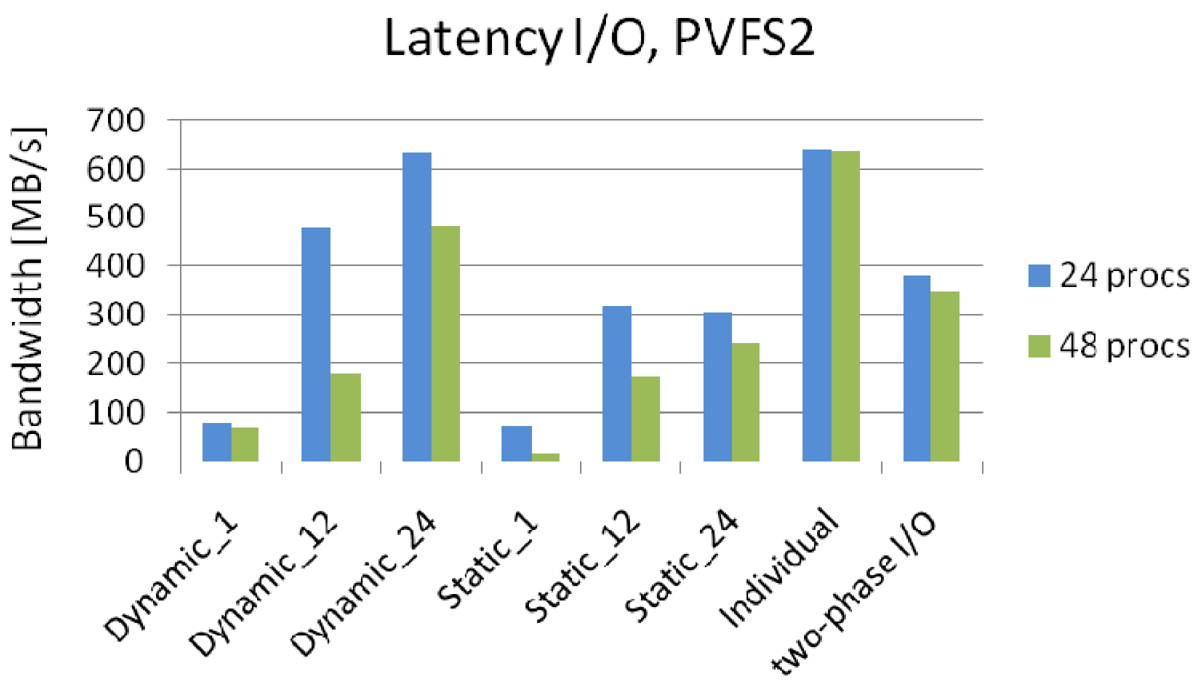


Edgar Gabriel

Case study: tuning collective write operations

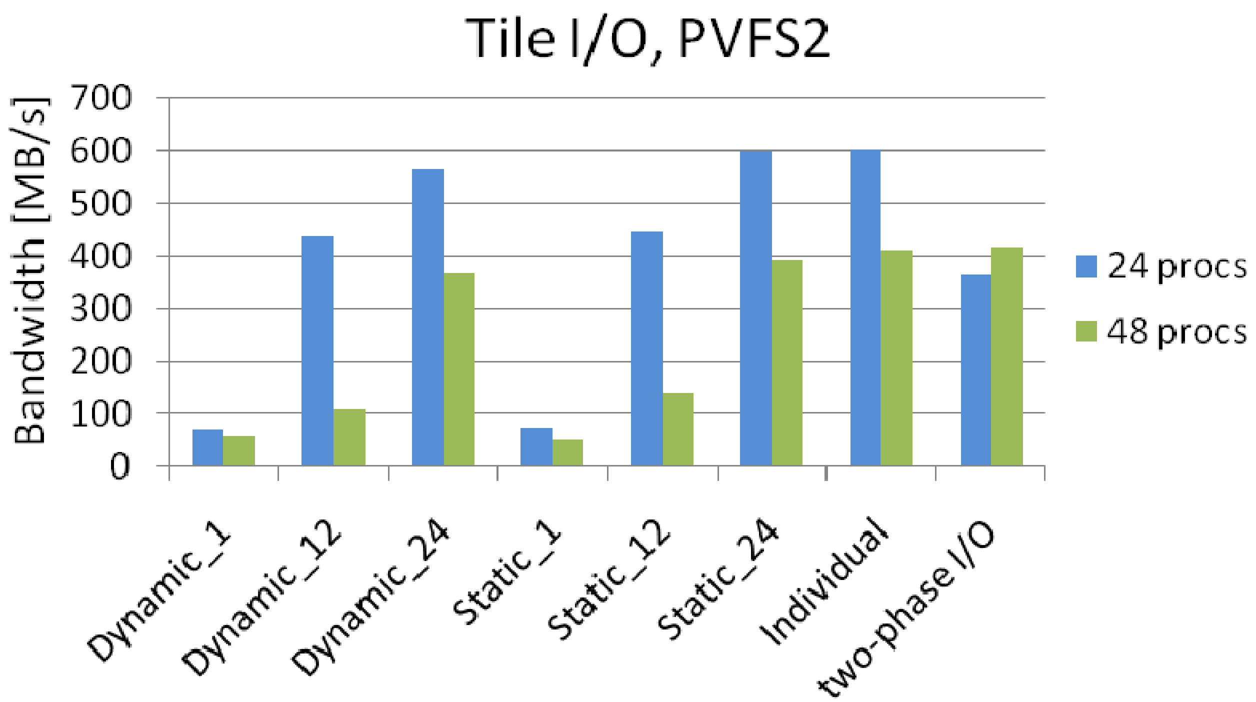
- Three modules currently available
  - **Dynamic segmentation**: re-arrange data of multiple processes optimizing disk access by creating process sub-groups
  - **Static segmentation**: re-arrange data of multiple processes optimizing the communication between the processes by creating sub-groups
  - **Individual**: each process handles its own data items, incorporating additional scheduling of the processes to prevent congestion on the I/O level.

Case study: tuning collective I/O



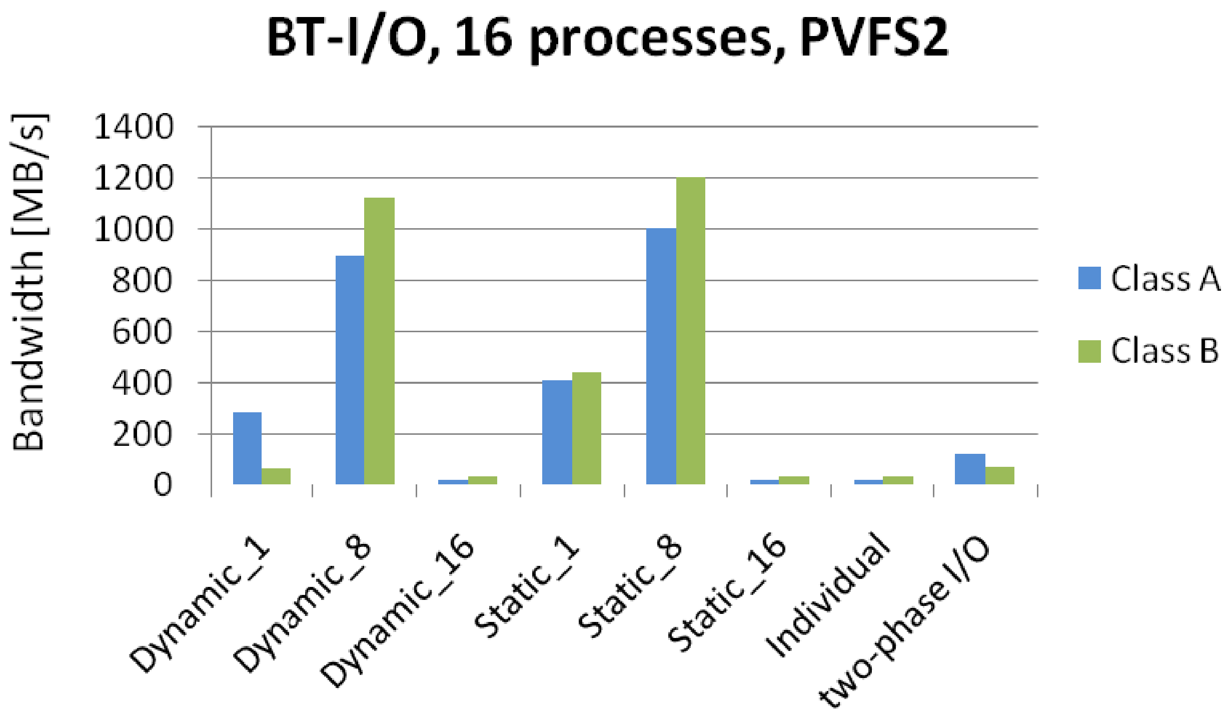
Parallel Software Technologies Laboratory PSTL

Case study: tuning collective I/O



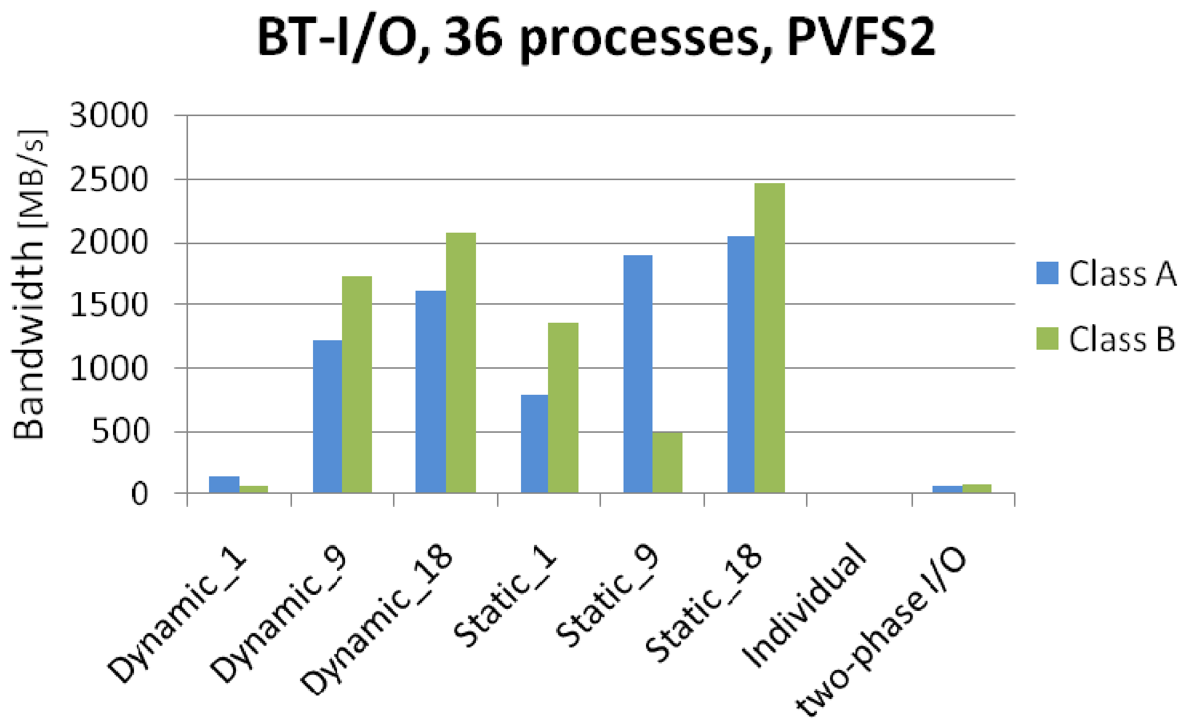
100x100 pixels, 32k blocks

Case study: tuning collective I/O



Parallel Software Technologies Laboratory PSTL

Case study: tuning collective I/O



CS@UH

Conclusion

- Overall infrastructure mostly implemented
  - non-blocking operations currently missing
- List of modules work in progress
  - community involvement envisioned and welcomed!

- Collective I/O algorithms currently being further extended
  - new grouping concepts for dynamic and static segmentation algorithms
  - new scheduling strategies for the individual algorithms