ScalaIOTrace: Scalable I/O Tracing and Analysis

Karthik Vijayakumar[1], Frank Mueller[1], Xiaosong Ma[1,2], Philip C. Roth[2]

[1] Department of Computer Science, NCSU

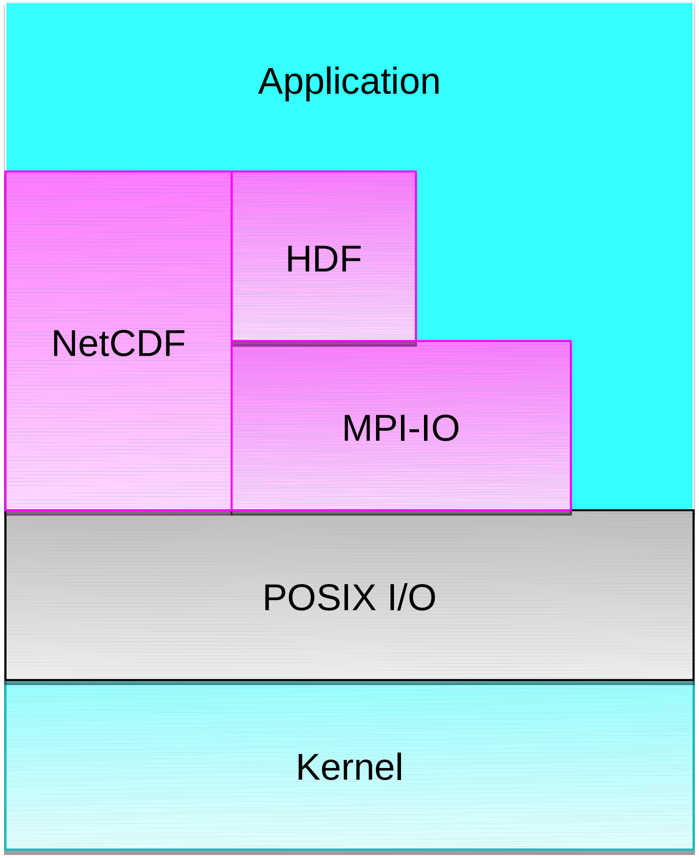
[2] Computer Science and Mathematics Division, ORNL

NC STATE UNIVERSITY
Department of Computer Science

OAK RIDGE NATIONAL LABORATORY
Managed by UT-Battelle for the Department of Energy

Problem

- Analyzing I/O behavior of parallel applications is difficult
  - Due to multi-level layering in parallel I/O
  - Abstractions hide complex implementation details
  - Difficult to analyze interactions between multiple layers
  - May hide bottleneck due to poor lower layer implementation



I/O Behavior Analysis

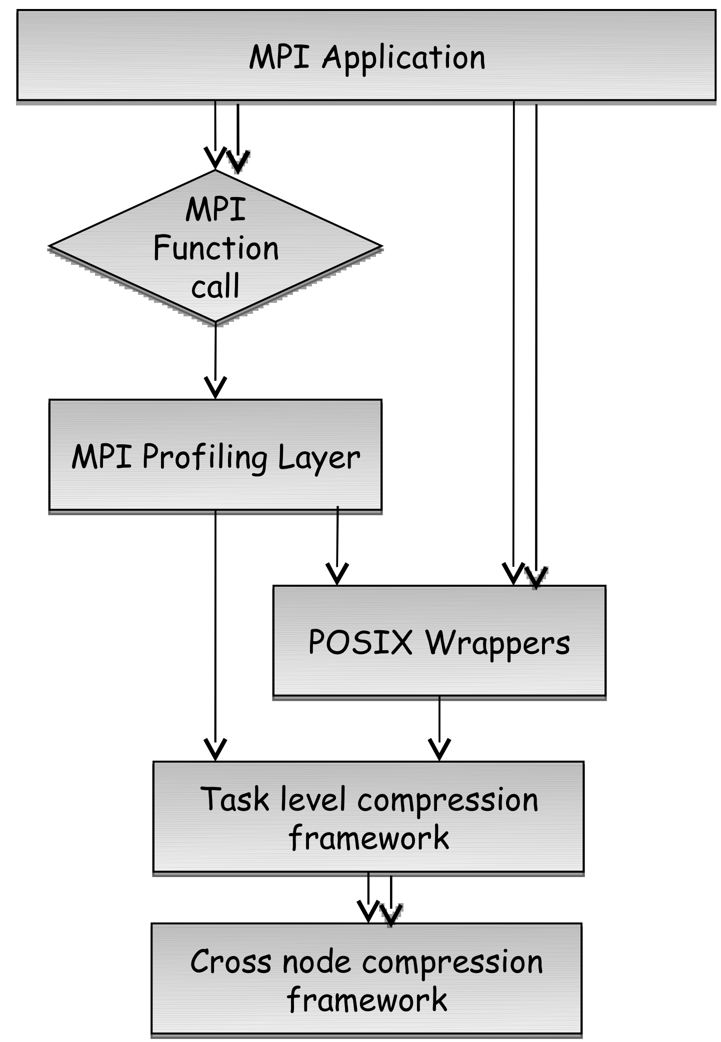
- Existing approaches and short comings
  - Application I/O kernels (Flash I/O)
    - \+ Exact application I/O behavior retained
    - \+ Simplify/Eliminate communication & computation
    - \- Analysis of scientific application is difficult
    - \- Requires substantial man-power to maintain
  - I/O event tracing
    - \+ Easy to generate traces by code instrumentation
    - \- Scalability issues due to large traces
    - \- Disturbance in application run due to trace accesses

Our Approach

- Trace-driven approach to analyze I/O behavior
- Goals
  - Extract traces at multiple layers of abstraction
  - Maintain structure of events
  - Automate trace analysis
  - Full replay (independent of original application)
  - Scalable
  - Lossless compression
  - Retain causal order
  - Preserve time
  - Facilitate customization/integration with other tools

ScalaIOTrace Design Overview

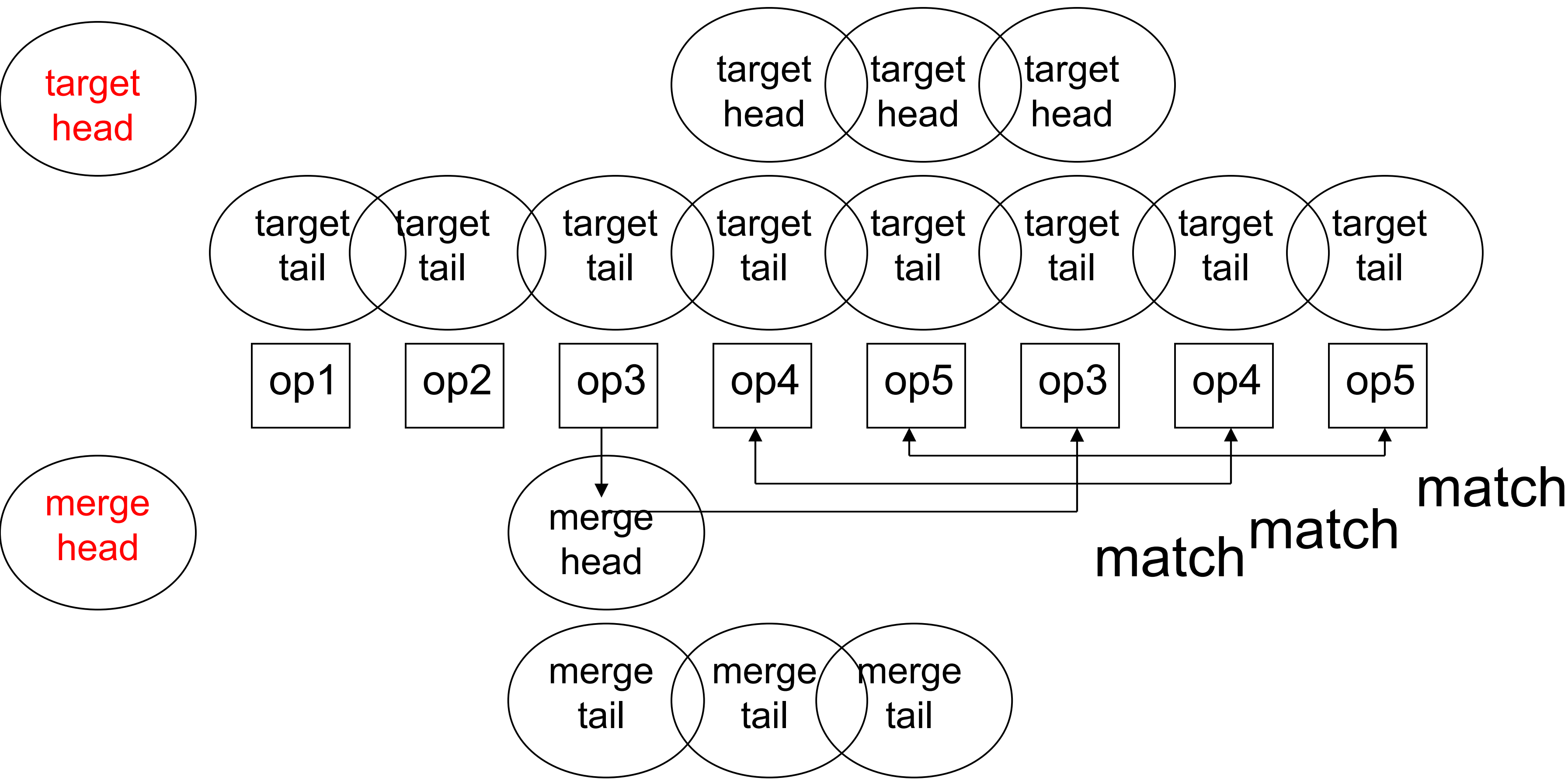
- Built on ScalaTrace [IPDPS '07] to collect MPI-IO & POSIX I/O traces
  - Use MPI profiling layer for MPI-IO and communication calls
  - Use GNU linker's link time function interposition facility for POSIX I/O calls
  - Compress at the task-level (**Intra-node**)
  - Compress across all nodes (**Inter-node**)



Intra-Node Compression Example

- Consider the MPI operation stream:

  op1, op2, op3, op4, op5, op3, op4, op5



Intra-Node Compression Example

- Consider the MPI operation stream:

  op1, op2, op3, op4, op5, op3, op4, op5

Regular Section Descriptor (RSD)

op1 op2 ((op3 op4 op5), iters = 2)

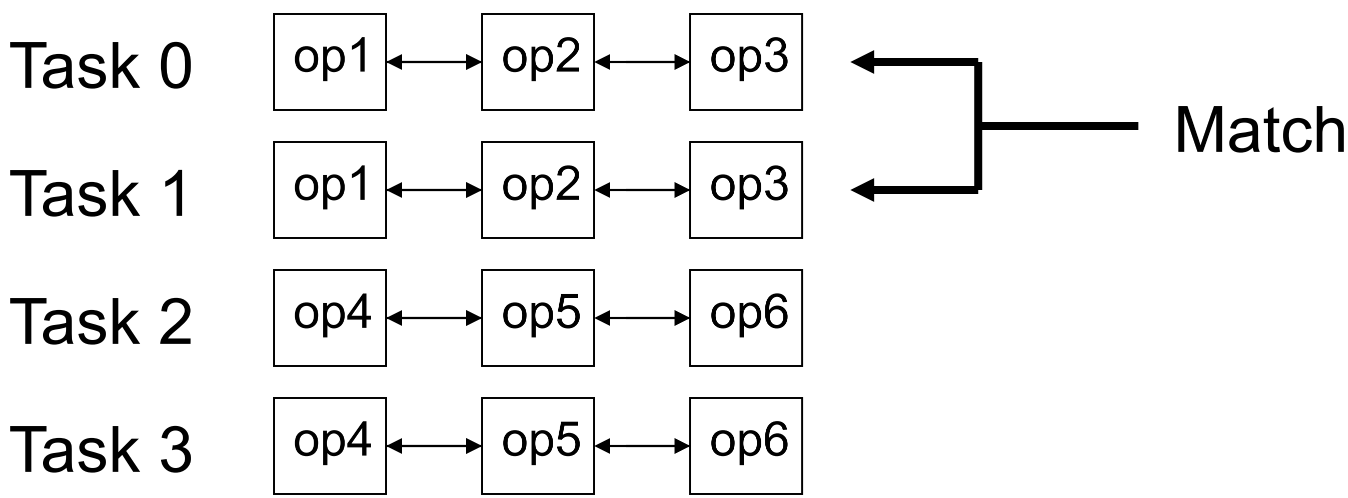
Power Regular Section Descriptor (PRSD)

Example:

op1 (op2 (op3 op4 op5), iters = 2), iters = 10)

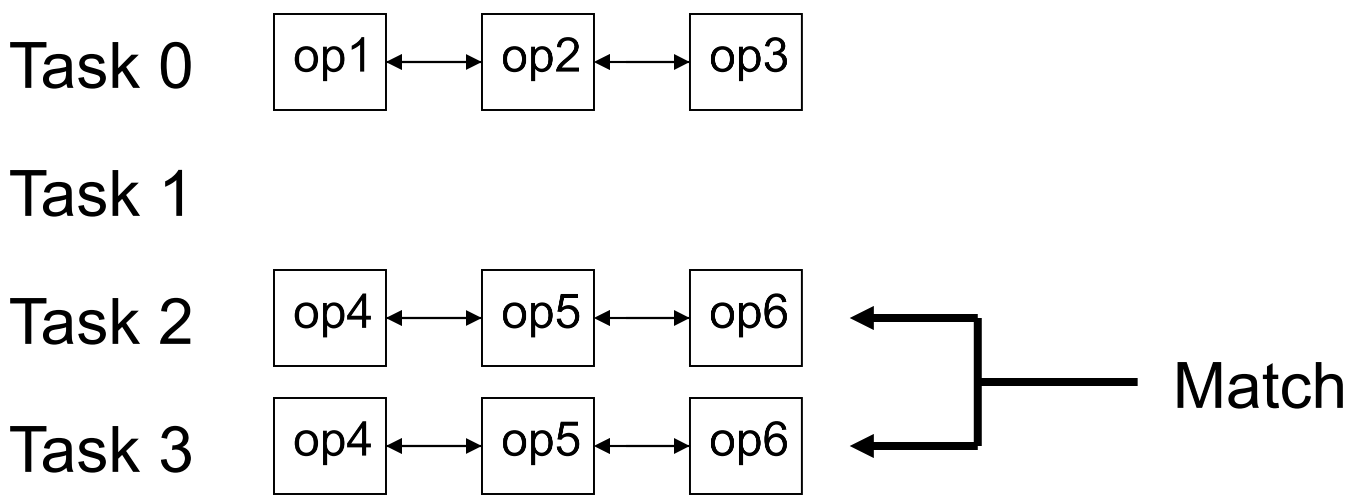
Inter-Node Compression Framework

- Invoked after all computations done (in MPI_Finalize wrapper)
- Merges operation queues produced by task-level framework
- Binary radix tree reduction process



Inter-Node Compression Framework

- Invoked after all computations done (in MPI_Finalize wrapper)
- Merges operation queues produced by task-level framework
- Binary radix tree reduction process



Time Preservation

- Recording absolute timestamps not scalable
- Record delta time
  - Computation delta
  - MPI delta
- Minor variations prevent exact match
- Aggregate stats (min/max/mean) per op
  - Traces lossless for communication parameters
  - But lossy for delta-time recording
- Histograms (dynamically balanced)

I/O Trace Collection

- Collecting application access behavior alone is not enough
- Interaction between different layers is important
  - Isolate application's behavior at a certain level
  - Correlate activities at multiple levels
- Idea: reuse infrastructure for lossless I/O tracing
- Collect MPI-IO(higher level) and POSIX I/O (lower level) traces
  - Expose multiple layers
  - Enable analysis of multi-level traces in a scalable way
  - Can be extended to any levels

MPI-IO Trace Generation

- Umpire [SC'00] → wrapper generator for MPI profiling layer
  - Initialization wrapper
  - Tracing wrapper
  - Termination wrapper
- File name compression
  - Checkpoint files are written periodically
  - File names typically have static and dynamic component
  - Merge if static components match
- I/O calls used repetitively to access shared files using **file offsets**
  - Encode access pattern into three fields <start, stride, total number of elements>

MPI-IO Trace Generation (cont.)

- Representation of file handles
  - Opaque pointers, no repetitive patterns
  - Store handle in a buffer, added to buffer on file open
  - Encoded with buffer offset during file access

- Support for custom data types (MPI_Type_create_darray)
  - Encoding similar to file handles

POSIX I/O Trace Generation

- POSIX I/O belong to lower level in I/O stack
  - Provide details on actual requests made to parallel file system
  - Also some application do not use higher level I/O libraries
- Enhanced Umpire tool to generate POSIX I/O wrappers
- Code Instrumentation using GNU link time interposition facility
  - "--wrap" option used to collect traces for open, write, etc.
  - Control redirected to interposition function "__wrap_open"
  - Separate library provided for POSIX wrappers.

Trace Replay

- ScalaIOTrace supports scalable replay engine
  - Reissues MPI-IO and communication calls
  - No trace decompression
  - Issues calls with original parameters with dummy payloads

- Time preserving replay
  - Simulate computation by adding delay
  - Pick delta value randomly from histogram +extremes (min/max)

Post-mortem Trace Analysis

- Problems in conventional trace analysis
  - Requires separate application runs
- Replay tool facilitates post-mortem analysis
- Generic event handlers provided for all recorded functions
  - User specific code can be added to collect information
- Facilitates anomaly detection by iterative refinements

Experimental Results

- Environment
  - Jaguar Cray XT4 at NCCS(ORNL)
  - AMD 2.1 GHz quadcore
  - 8GB of RAM/node
- Varied:
  - Number of nodes
- Examined metrics:
  - Trace file size
  - Aggregation results from trace analysis
- Results for
  - Flash I/O Benchmark
  - Parallel Ocean Program (Developed at LANL)

Flash I/O Benchmark

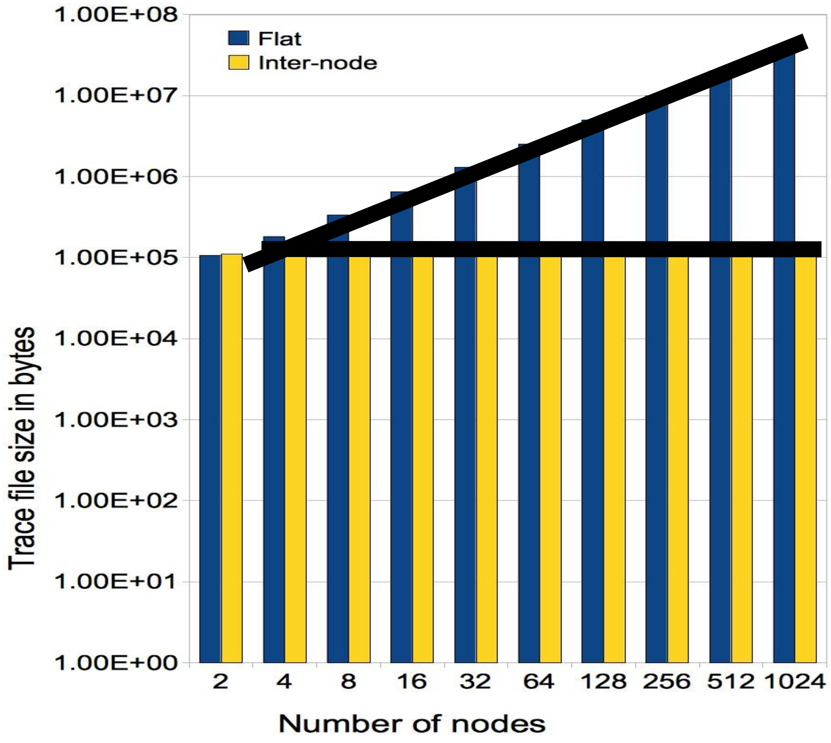
Simulates I/O behavior of FLASH (adaptive mesh hydrodynamics) application

Uses parallel HDF5 I/O library

Log scale: file size [Bytes], 2-1024 nodes

Two categories: No compression, Inter-node compression

- Uncompressed traces
  - Linear growth
- Inter-node compressed traces
  - Almost constant
  - Due to SPMD prog. Style



# nodes	MPI-IO at 0	POSIX I/O at 0	Comm. at 0	MPI-IO Other	POSIX I/O Others	Comm. Other
2-1024	194	171	299	85	56	299

Parallel Ocean Program (POP)

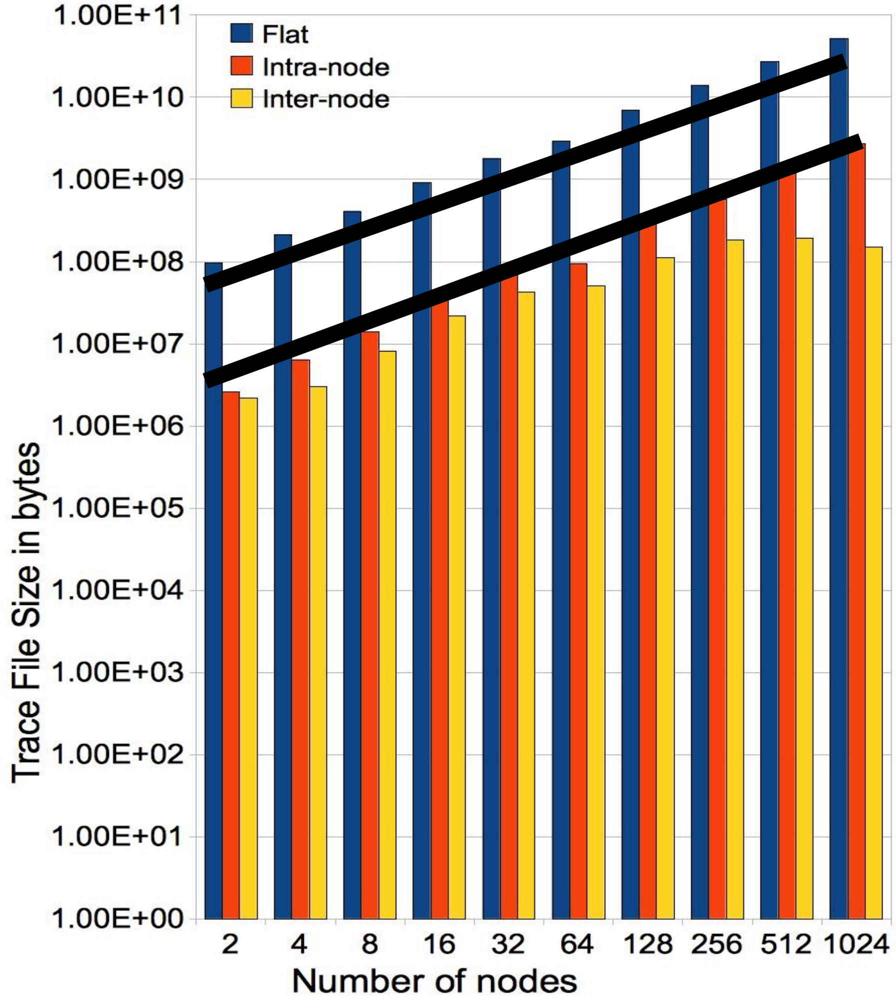
- Ocean Circulation Model
  - Developed at Los Alamos National Laboratory
- No parallel I/O
  - Uses NetCDF (in turn uses POSIX I/O)
  - Node 0 performs I/O, collects/distributes data from/to other nodes
- Problem size
  - 1 degree grid resolution
  - Problem size 320x384 grids
  - Vary max. # of blocks assigned to each node
- Goal: Analyze compression effectiveness of real scientific applications

POP – Trace File Size

Three categories: Flat/Intra-node/Inter-node

Log scale: file size [Bytes], 2-1024 nodes

- Vary # of blocks assigned to nodes with increasing nodes
- Intra-node compression
  - Linear growth
  - Size: order of magnitude less than flat traces
- Similar behavior for inter-node compressed traces
- Imperfect timestep loop compression
  - ε-convergence problem



POP – Post-mortem Analysis

Aggregation results:

- Results for I/O and comm. Ops (collective/blocking/non-block)
- Avg. # of operations for non-zero nodes.
- Blocking calls are I/O induced
  - Parallel I/O would have reduced comm. overhead

# nodes	I/O at 0	Coll. at 0	Block. at 0	NB at 0	Coll. Other	Block. Other	NB Other
2	1589	21247	129034	231714	21247	0	385350
4	1573	21257	179284	308952	21257	0	388838
8	1573	21277	210140	308952	21277	0	393393
16	1573	21317	1444912	386190	21317	0	447680
32	1573	21397	858648	386190	21397	0	451373
64	1573	12225	858648	386190	8575	0	382512
128	1573	21877	463372	386190	21877	0	441344
256	1573	22517	470288	386190	22517	0	426550
512	1573	23797	239932	386190	23797	0	424329
1024	1573	26357	240198	386190	26357	0	412485

- Comm. overhead increases due to strong scaling
- Comm. performed in sub-groups
  - Avg. non-blocking call for all others > than that of node 0

Future Work

- Introduce user-tunable imprecision
  - Exact iteration counts not useful in cases of convergence problems
  - Irregular trace events due to data-dependent conditionals
  - Trade off: Trace file size vs. lossless traces

- Enhance trace analysis framework
  - Provide configurable options to collect statistical information without user understanding trace file

Conclusion

- Aggressive trace compression
  - Near constant size trace file for Flash I/O
  - Two orders of magnitude smaller trace file for POP
- Capability to record traces at several layers
- Framework for post-mortem trace analysis
- Download URL:
  - http://moss.csc.ncsu.edu/~mueller/ScalaTrace/

NC STATE UNIVERSITY
Department of Computer Science

OAK RIDGE NATIONAL LABORATORY
Managed by UT-Battelle for the Department of Energy

Acknowledgements

- Funded in part by NSF grants 0237570 (Career), 0621470, 0937908, 0429653 and joint faculty appointment between Oak Ridge National Laboratory and NCSU
- Part of this work was performed under auspices of Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725
- Used resources of National Centre for Computational Sciences at ORNL, supported by Office of Science of the Department of Energy under contract DE-AC05-00OR22725

NC STATE UNIVERSITY
Department of Computer Science

OAK RIDGE NATIONAL LABORATORY
Managed by UT-Battelle for the Department of Energy