Scalable Tracing of MPI Events and Performance Metrics

Tao Yan, Qingguo Xu, Jiyu Luo, Jingwei Sun*, Guangzhong Sun

University of Science and Technology of China

Background

Tracing is a basic approach to analyzing the performance of MPI programs.

Besides MPI event trace, some performance analysis tasks also require detailed runtime metrics from specific performance counters during different execution periods of HPC applications.

With the growth of parallel scales, storing the trace and performance metrics of MPI programs becomes increasingly challenging.

Related Work

Existing mature MPI trace analysis tools such as tau, Vampir, mpiP, etc. only provide general-purpose domain-independent compression (for example, using the zlib library) or aggregate representation (summarizing the trace as a series of statistical values such as mean and variance).

The methods in existing research papers, such as Scalatrace, DwarfCode, LCR, etc., either use design pattern strings for pattern matching to find the loop structure in Trace and the repeated parts between processes, or realize compression through time series modeling.

Related Work

The problems of the prior art are as follows:

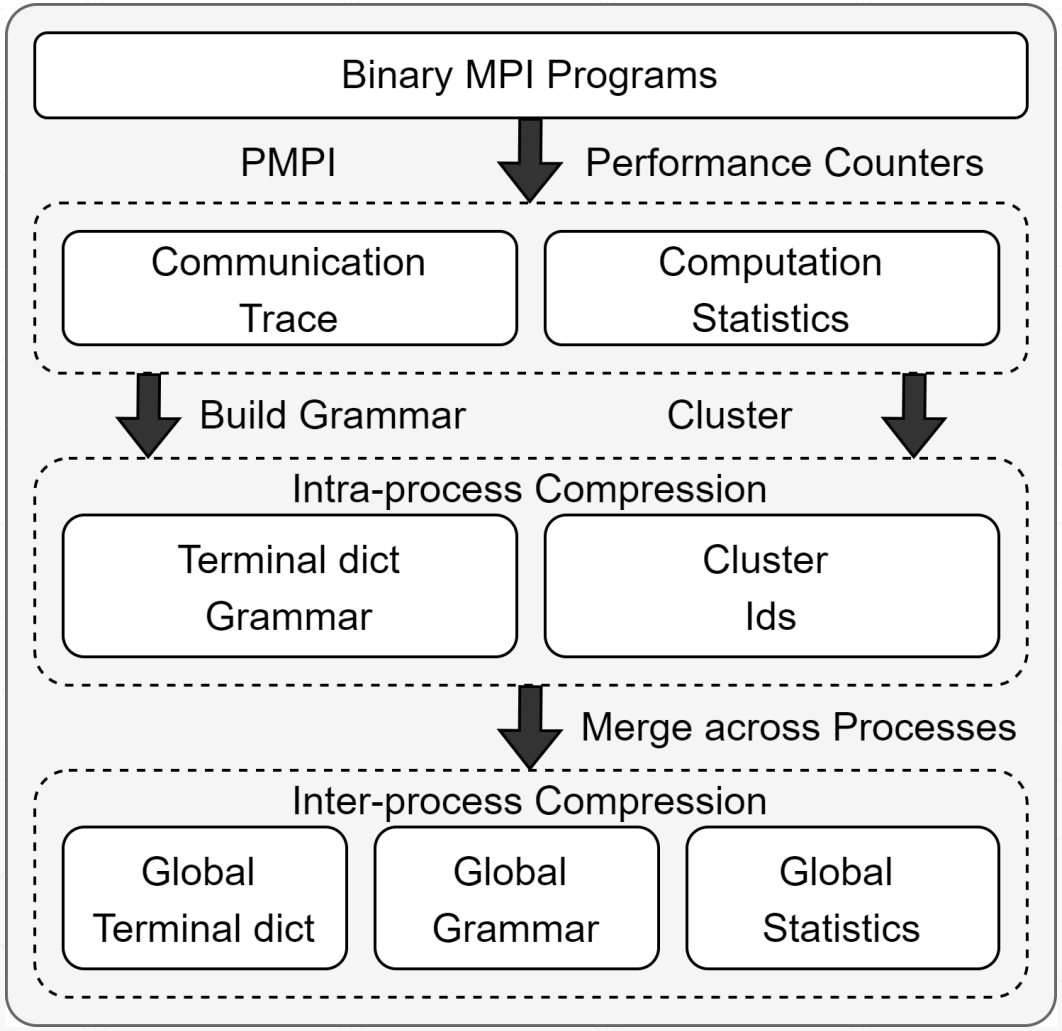
- Common domain-independent compression methods do not have high compression ratios.
- The information loss of the aggregate representation method is too high to restore the MPI program behavior.
- Methods for generating scalable traces only focus on MPI communication traces, ignoring computing performance data

Our contributions

- We implement a method that can losslessly compress communication traces of MPI programs. By leveraging the SPMD feature across processes, a novel method of merging traces across processes is proposed to further reduce the size of the compressed traces.
- We propose a clustering-based method for compressing performance counter data that outperforms existing lossy compression methods for floating-point data.
- We conduct a series of experiments to compare our tool and existing works, including Scalatrace, DwarfCode, and LCR.The results show that our tool can achieve generally higher compression ratio and less time cost.

Overview of Scalable Tracing Framework

- Tracing with PMPI and PAPI
- Intra-process compression
- Inter-process compression

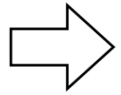


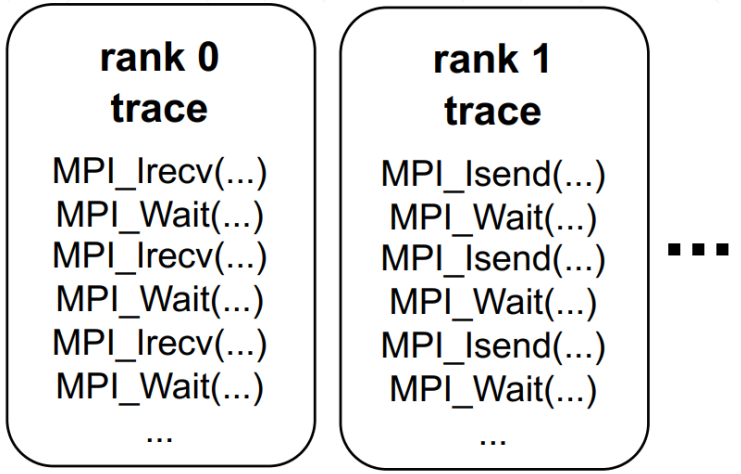
Overview of tracing and compressing method

Tracing

- Comm MPI communication traces
- Our tracing tool is implemented based on mpiP and uses PMPI to trace MPI events

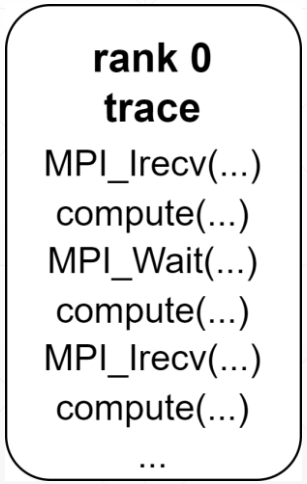
```
for(i = 0; i < 100; i++) {
    if(rank % 2) {
        MPI_Isend(...);
        MPI_Wait(...);
    } else {
        MPI_Irecv(...);
        MPI_Wait(...);
    }
}
```





communication traces

- Record performance metrics information between two MPI functions
- We currently record information from hardware performance counters due to its generality in applications



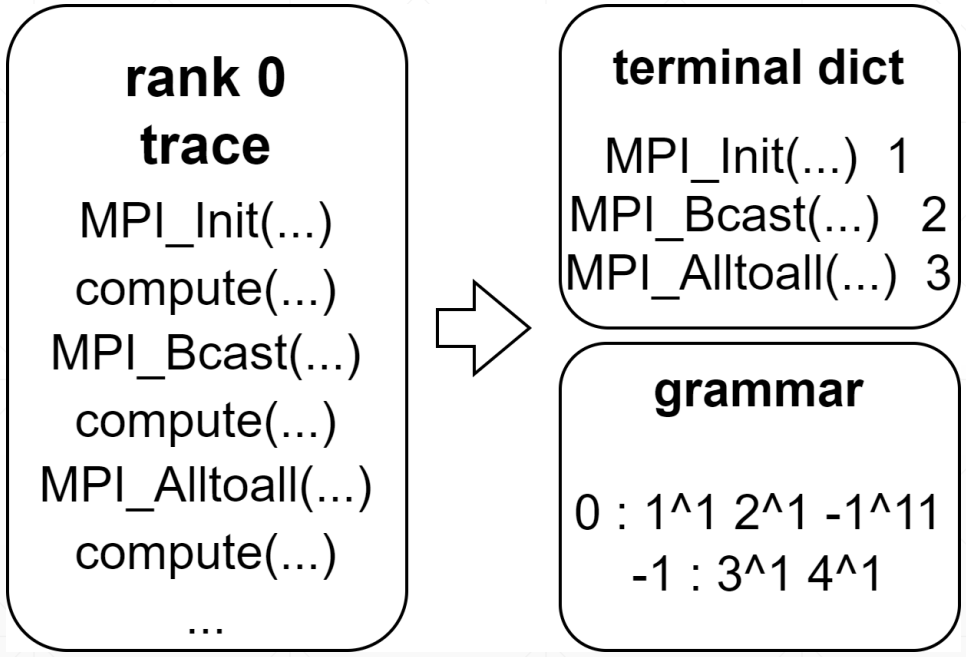
communication traces and performance metrics

Optimization

- To better leverage the similarity between events
- Maintain a pool of free ids for many MPI objects
  - Record only what MPI objects represent instead of their specific values by using a pool of free ids.
  - Example: MPI_Request and MPI_Comm.
  - Benefits: Reduces the amount of data that needs to be recorded, making it easier to analyze communication traces and speeding up compression procedures.
- Use relative ranks to reduce communication trace redundancy
  - Record the value of some variables as offsets relative to the caller's rank to reduce communication trace redundancy.
  - Example: Communication patterns in HPC applications that use a Cartesian grid.
  - Benefits: Reduces the amount of data that needs to be recorded, making it easier to analyze communication traces and speeding up compression procedures.

Intra-process compression

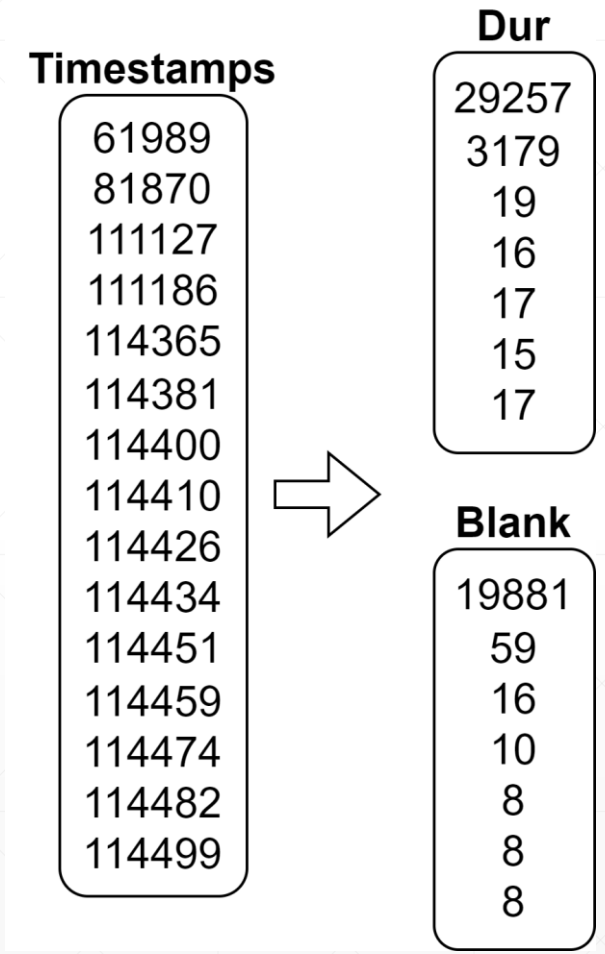
- Context-free grammar
  - It can find repetitive structures in symbol sequences and extract them
  - It is defined as $G = \{V,\ \Sigma,\ S,\ R\}$ which represent non-terminals, terminals, start symbol, and generating rules respectively
- Compress MPI trace to a context-free grammar and symbol dictionary
- Use space-optimized sequitur algorithm to compress



An example of compress a trace to a grammar

Intra-process compression

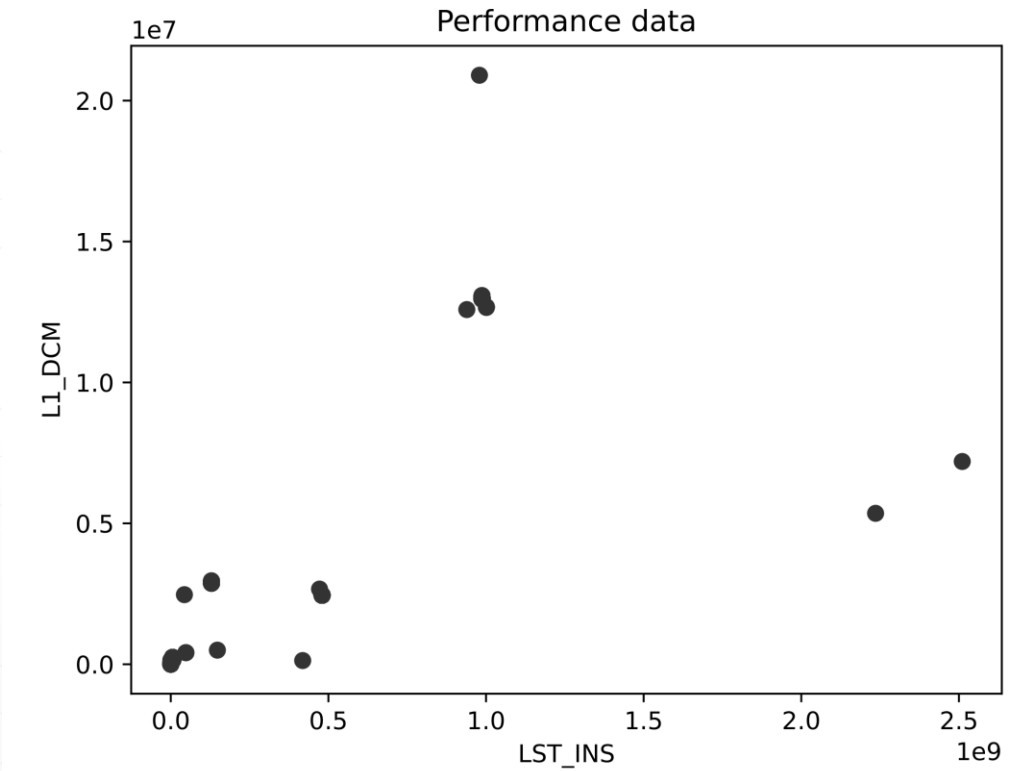
- Time information is processed as *dur* for communication function durations and *blank* for computing intervals
- Two lossless compression methods, zlib and lzma
- Two lossy compression methods, a statistical method that records only the mean and standard deviation and SZ3 lossy compression algorithm



Time information processing

Intra-process compression

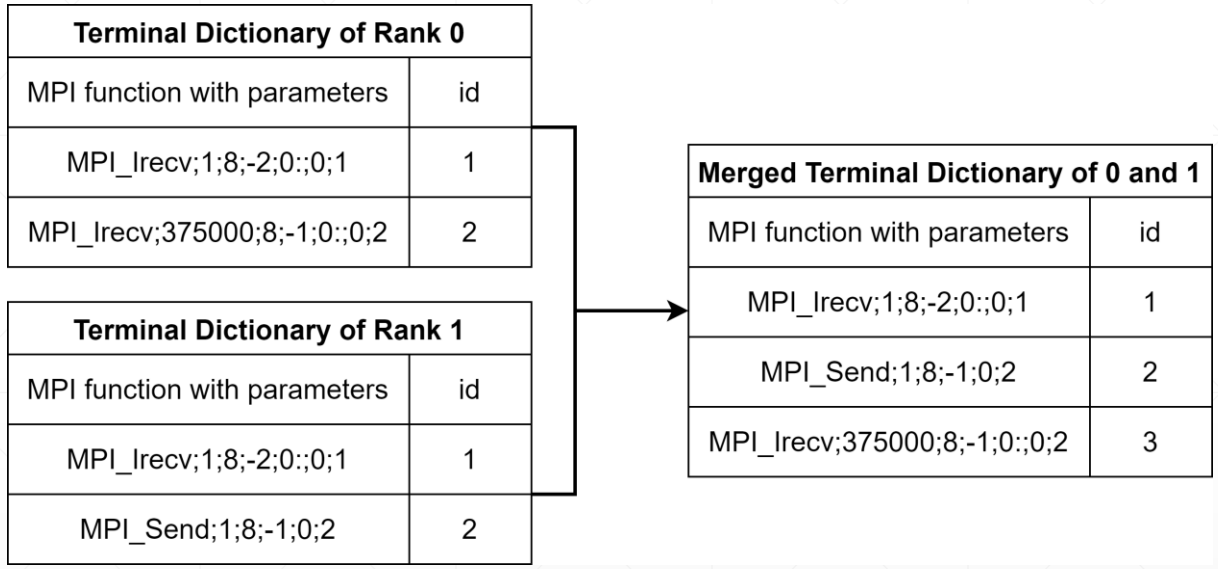
- The performance counter data itself comes from the execution of code blocks, and the number of code blocks is limited. Therefore, there are only a limited number of combinations for performance counter data. Due to performance variance, the data may be scattered within a certain range, making it possible to achieve compression through clustering methods.
- With the ability to represent similar data by the value of the cluster center, our method offers advantages compared to other compression methods, enabling higher compression ratios.



Scatter plot of two types of performance data in NPB LU

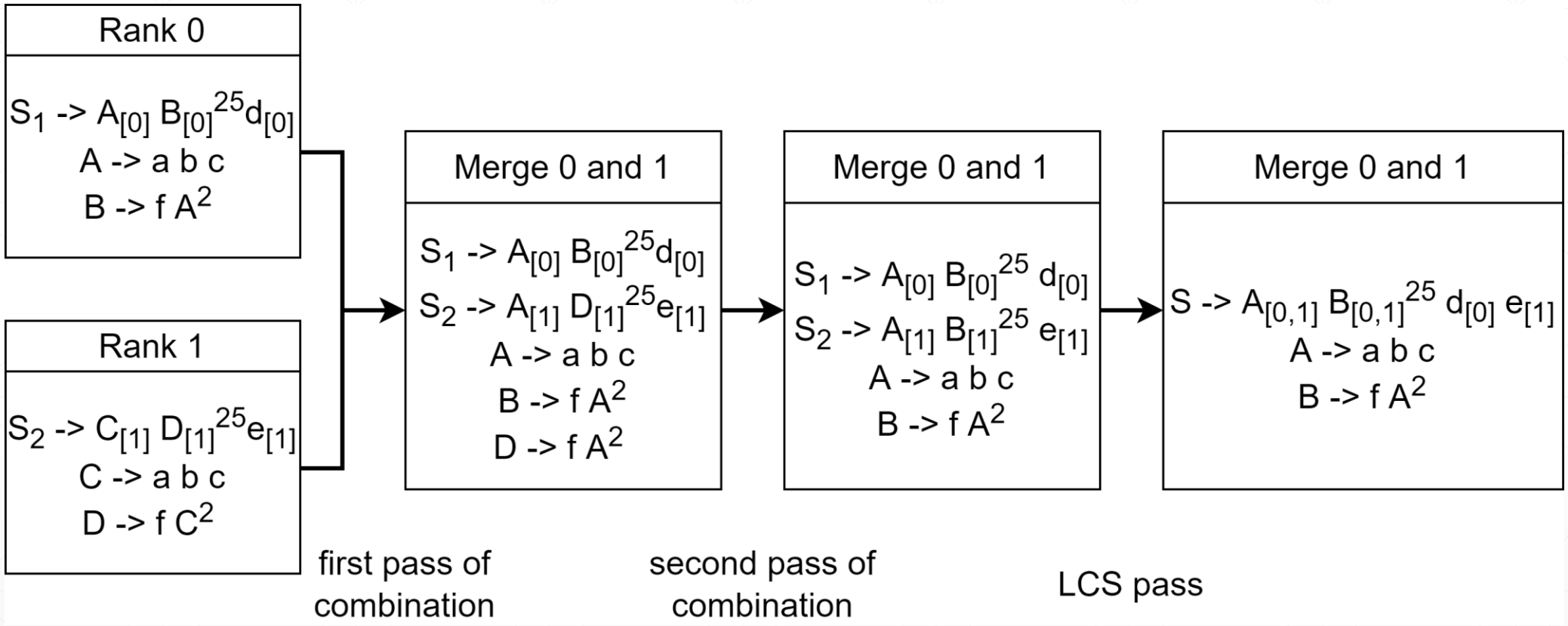
Inter-process compression

- Merge the terminal dictionaries of all processes and assigning each terminal a global id
- Utilize the SPDM features of the program
- Merge grammars with different methods according to the number of unique grammars



Example of merging parts of two terminal dictionaries

Inter-process compression



Example of merging two grammars

Evaluation

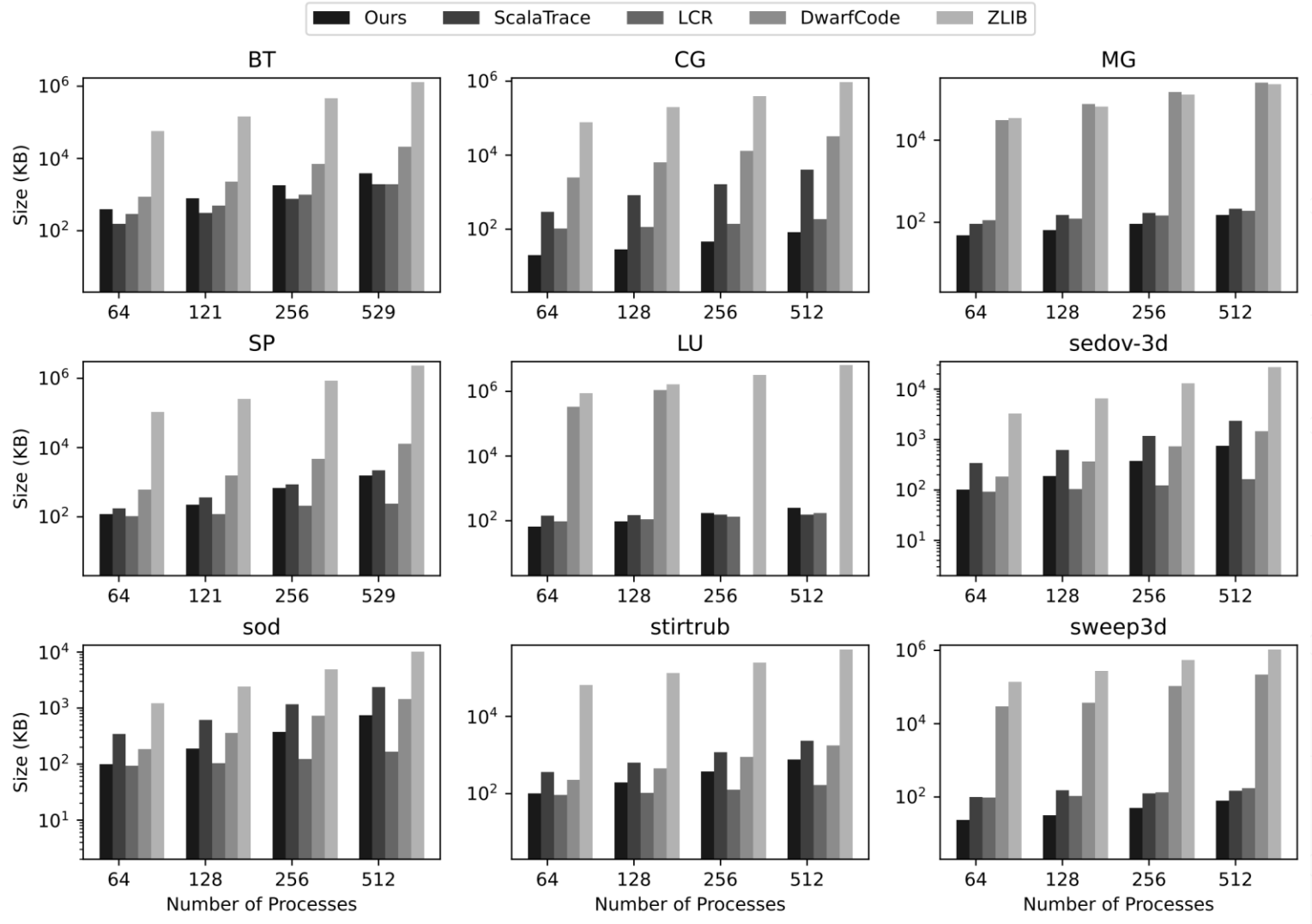
- Selected Programs:
  - CG, MG, BT, SP and LU from NAS Parallel Benchmark (NPB).
  - SWEEP3D (solver for neutron transport).
  - FLASH (production software package for handling general compressible flow problems).
- Problem Sizes:
  - NPB3.3.1: D
  - FLASH: 64x64x64
  - SWEEP3D: 1000x1000x1000
- Experimental Setup:
  - Processor: Intel® Xeon Scale 6248 CPU (20 cores2.5 GHz).
  - Memory: 192 GB DDR4 2933 MHz RAM.
  - Network: Mellanox HDR100.
  - Compiler: GCC 4.8.5
  - MPI: OpenMPI 3.1.0.

Evaluation

- Baseline Method
  - zlib
  - Dwarfcode
  - Scalatrace
  - LCR

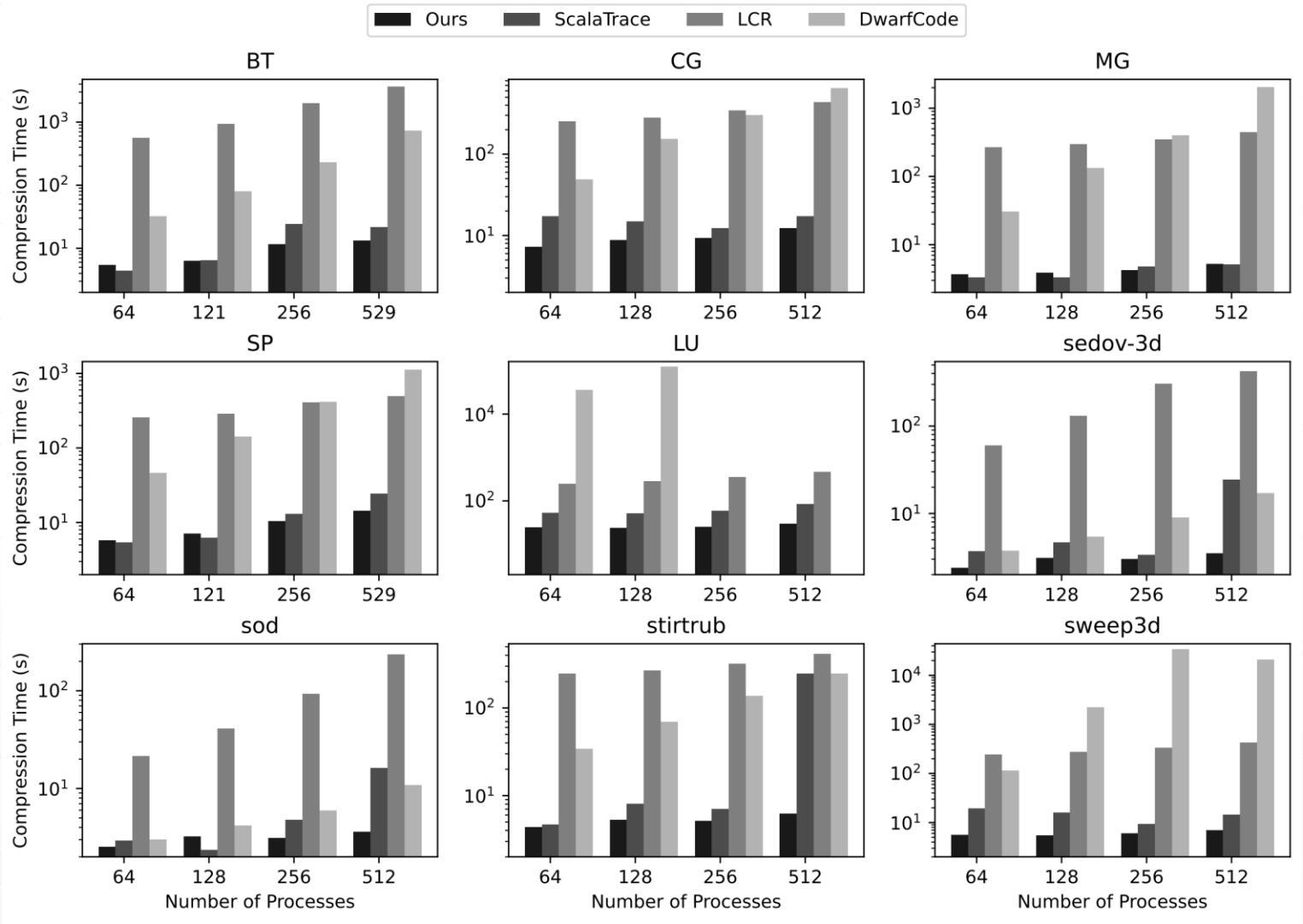
Evaluation

- Compression ratio



Evaluation

- Compression time



Evaluation

- Clustering-based method compare with SZ3

Program	Clustering-based Method			SZ3 Method		
	CR	PSNR	NRMSE	CR	PSNR	NRMSE
BT	4785	47.20	0.004	338.09	29.45	0.033
CG	6442	99.81	1E-5	779.69	50.11	0.003
LU	4345	90.84	2E-5	5825	53.67	0.002
MG	2167	54.06	0.001	197.29	42.77	0.007
SP	6806	56.27	0.001	798.75	39.54	0.010
Sedov	989.85	86.18	5E-5	25.13	46.73	0.004
StirTurb	5062	76.34	1E-4	490.57	52.78	0.002
Sod	357.18	61.97	7E-4	9.22	42.86	0.007
sweep3d	2525	55.34	0.001	366.03	29.76	0.032

Compress result of one counter record

Program	Clustering-based Method			SZ3 Method		
	CR	PSNR	NRMSE	CR	PSNR	NRMSE
BT	5027	56.77	0.001	129.01	31.66	0.026
CG	20412	93.44	2E-5	2379	53.34	0.002
LU	10945	89.92	3E-5	10500	53.37	0.002
MG	2809	98.09	1E-5	495.91	45.60	0.005
SP	17371	60.48	9E-4	625.84	41.06	0.008
Sedov	1820	89.38	3E-5	71.45	58.68	0.001
StirTurb	13278	104.51	6E-6	1147	50.67	0.002
Sod	901.80	96.01	1E-5	25.42	47.87	0.004
sweep3d	1208	59.21	0.001	160.22	29.00	0.035

Compress result of three counter record

Conclusion and future work

- Conclusion
  - propose a scalable tracing tool for recording and compressing MPI events and performance metrics.
  - Compared with existing trace compression works, the proposed tool can achieve generally higher compression ratio and less time cost, so it is more scalable to larger scale executions of MPI programs.
- Future work
  - Enhance its ability to deal with workloads with more different features and scales.
  - Investigate efficient downstream performance analysis techniques based on the tool.