Evaluating Similarity-based Trace Reduction Techniques for Scalable Performance Analysis

Kathryn Mohror and Karen L. Karavanic
Portland State University
{kathryn,karavan}@cs.pdx.edu

ABSTRACT

Event traces are required to correctly diagnose a number of performance problems that arise on today's highly parallel systems. Unfortunately, the collection of event traces can produce a large volume of data that is difficult, or even impossible, to store and analyze. One approach for compressing a trace is to identify repeating trace patterns and retain only one representative of each pattern. However, determining the similarity of sections of traces, i.e., identifying patterns, is not straightforward. In this paper, we investigate pattern-based methods for reducing traces that will be used for performance analysis. We evaluate the different methods against several criteria, including size reduction, introduced error, and retention of performance trends, using both benchmarks with carefully chosen performance behaviors, and a real application.

1. INTRODUCTION

Today's high-end architectures contain tens to hundreds of thousands of processors, pushing application scalability challenges to new heights. Performance analysis is a necessary step to adapt codes to utilize a target high end machine. Correct diagnosis of certain complex performance problems that arise on high end systems requires detailed event traces. An "event" is a runtime occurrence of a program activity, such as a machine instruction or basic block execution, memory reference, function call, or a message send or receive. Generating event traces involves writing a time stamped record for each event, into a buffer or file for later analysis. Unfortunately, the collection of event traces presents scalability challenges: the act of measurement perturbs the target application; and the large volume of collected data increases the perturbation, and results in data files that are difficult, or even impossible, to store and analyze [24]. Several documented cases describe performance problems that appear only when the application is run at a large scale [18, 27], driving the need to be able to collect event traces for large runs. We have a conundrum: we need traces to correctly diagnose important performance problems, but the sheer volume of data collected makes collecting full traces at the very least prohibitive, and in the worst case impossible. For this reason, solving the scaling challenges of event tracing is an important problem for high end computing.

Given the challenges of tracing at the high end, one might be tempted to avoid it entirely. Profiling, for example, provides summary information and therefore exhibits better scaling behavior. However, the types of information provided by profiling are, in many cases, too limited for correct diagnosis of certain performance problems [7, 36]. An example of such a performance problem is "Late Sender" in a message-passing program. This is the situation where the receiving process waits at a blocking receive call waiting because the sending process hasn't yet reached the matching send call. While a profile could indeed show that excessive time was being spent in receive operations, the data is not sufficient to distinguish between a late sender or some other root cause, such as network contention that caused the message to be received late. In contrast, an event trace captures the relative timing of events, and would show that the send operations started late and caused the receive operations to block. Tracing is also useful for showing the causality of events [31, 12]; the interactions between program elements, that can be difficult or impossible to understand from static analysis [22, 20]; and event patterns that reveal properties of programs, such as performance problems and locations of possible optimization [21].

One promising approach to highly scalable tracing is to filter or reduce the trace in some manner, either during or after the collection of trace records. Users who need to collect trace data currently resort to ad-hoc measures to reduce the amount of data collected; for example, tracing a reduced number of iterations of a loop. These measures have the potential to miss the performance problem altogether, e.g. if the problem doesn't occur during the measured iterations. One method for reducing the size of traces is to identify similar sections of a trace and retain only one representative of each pattern. However, determining the similarity between traces or sections of traces is not straightforward. The probability that any two trace sections will have exactly the same measurements is very small, so any similarity method will allow some amount of differences between similar traces. Despite this, it is critical that any differences allowed do not mask information needed for correct performance diagnosis.

Requirements for the accuracy and types of information in a trace vary based on the intended use: correctness testing and debugging, simulation, or performance analysis. *Correctness testing and debugging* generally only require that the trace retain the relative ordering of events that have the potential to affect each other: events within a single process or thread and synchronization events across processes or threads. For example, inspecting a trace of a parallel program could indicate the reason for a deadlock situation by showing the ordering of synchronization operations; a parallel program might hang because a process is waiting for a message that was never sent. *Simulation* requires traces that retain the order of events and possibly some timing information. Traces for simulation can be used to predict application performance on new or theoretical hardware. The events in the trace can be replayed using either averaged or predicted timing information for the new hardware. Generally, a single time value is used for all event occurrences instead of individual timing measurements for each event occurrence. For example, the average time to execute a send operation could be used as the time for all send operations in the trace. This tradeoff allows acceptable accuracy with faster time to simulated results and smaller trace files. *Performance* *analysis* requires not only the relative ordering of events, but the timing information for individual events. Performance problems do not necessarily occur with a high degree of regularity, e.g. in every iteration of a loop, so individual event timings are needed to show the root causes of problems. For example, trace data can show a time-varying load imbalance in a parallel job, which causes some ranks to be late to a synchronization operation at varying times during the program execution. The individual event timings can show what events are taking more time in the slower ranks and in what iterations the slowness occurs.

In this work, our goal is to determine a similarity metric that yields adequate trace reduction and also retains the information needed for correct performance analysis. Achieving our goal required that we answer several key questions:

- *What metrics can we use to evaluate and compare trace difference methods?* In addition to file size reduction, we developed and used metrics for error, greatest possible file size reduction (i.e. potential for repeated patterns), and consistency of performance diagnosis.
- *How much error should be allowed?* Values that will likely never be exactly equal need to be compared. We had to decide how much each measurement can vary, and weigh the consequences of the amount of error. If we are matching traces for the purpose of trace compression, then a larger allowed error between traces would mean larger number of matches, and thus a smaller trace file. However, the larger error might prevent the correct performance diagnosis from being made.
- *How can we measure the "goodness" of each approach?* Most trace compression studies report the reduction of file size achieved; but no matter how much compression is achieved, if the reduced trace no longer contains the data needed for accurate performance diagnosis, the method is not useful for our purpose. We evaluate each approach not just on amount of compression, but also on amount of error and consistency of diagnosis, and discuss the tradeoffs in weighting the different metrics.

In this study, we perform a comparative evaluation of similarity metrics in current or proposed use for trace reduction. To evaluate the effectiveness of the similarity metrics, we apply the same trace reduction technique to full execution traces, varying the similarity method used to determine repeating patterns within the trace. Then we compare the results using three metrics: file size reduction, trace error, and retention of performance trends.

2. RELATED WORK

Previously proposed methods for reducing the sizes of traces for the purpose of performance analysis include deletion of similar trace sections; trace sampling; statistical clustering; and signal processing.

Knüpfer and Spooner define two sections of traces as similar if the call graph context and measurements of the events are equal. Knüpfer defines equality using both relative and absolute differences [19]; Spooner et al. use the relative difference in instruction counts [30]. Another approach defines similarity by event names. Chung et al. use a filter that detects repeated communication patterns [6]; they keep performance data for only one instance of each pattern. Freitag et al. use a periodicity detector to notice repeating sequences of events and keep a reduced number of iterations of each sequence [8]. Similarly, Yan and Schmidt detect repeating sequences of events and store the average measurements of those events [36]. Noeth and Mueller also detect repeated sequences of message-passing events and store one copy of each sequence; they optionally store summary information about the events, such as average measurements [26]. In later work, they include the ability to store more detailed timing information: statistical "delta" times, histograms, or histograms by call sequence [28].

Other efforts use trace sampling to reduce trace size. Carrington et al. use trace sampling to reduce the amount of time it takes to gather memory reference traces for the purpose of performance modeling [3]. They collect data for a reduced number of executions of the basic blocks in a program. Vetter presents a method for statistically sampling MPI events [32]. Each time an MPI event is encountered, it is either sampled or not. For each sampled event, the tool can record statistics, log the event to a trace file, or ignore the data. Gamblin et al. use statistical sampling with a user-specified confidence interval and metric. [10].

Aguilera et al. [2], Nickolayev et al.[25], and Lee et al. [23] apply statistical clustering to traces and select a representative trace for each cluster of processes. Nickolayev and Lee use the Euclidean distance for clustering, while Aguilera uses a metric based on the amount of communication between two processes.

Several groups apply methods from signal processing to traces. Casas et al. and Huffmire et al. use the Haar wavelet transform to automatically determine the phases of a program [4, 16]. Gamblin et al. use the CDF 9/7 wavelet transform to compress traces collected for the purposes of detecting load imbalance [9]. Hauswirth et al. use dynamic time warping to decide when two traces are similar for aligning multiple traces [14].

Researchers have evaluated several methods for deciding the goodness of a particular trace similarity metric. To our knowledge, ours is the only comparative study of the methods to see what is most appropriate for the purposes of performance analysis. Ratn et al. use aggregate statistical measures, such as total time spent in a function, to evaluate their method [28]. Gamblin et al. compute a trace confidence measure to evaluate their trace sampling results, which is tells the percentage of time the mean trace of sampled processes is within an specified error bound of the mean trace of the full trace [10]. In their wavelet transform method, Gamblin et al. use a root mean square measure to estimate the error in reduced traces [9]. They also present qualitative results, showing a visualization based on a reduced trace compared with one from a complete trace. Yan et al. compare the measurements in their reduced trace against the real trace time stamp by time stamp and produce both a relative and absolute measure of the overall differences [35]. In addition, they also present whole program statistical measurements and visualizations for qualitative comparison.

3. TRACE REDUCTION

In this section we describe our approach for trace reduction. Section 3.1 details our trace segmentation technique, and Section 3.2 describes the different similarity metrics we use to compare segments. This paper focuses exclusively on intra-process reduction, that is, reducing the size of each individual per-task trace. In practice these individual traces are first collected separately, then merged into a single trace file representing the entire application run. Therefore, reducing each

```
int main(){
        start_segment("init");
        MPI_Init();
        end_segment("init");
        for(i=0; i < 100; ++i){
            start_segment("main.1");
            do_work();
            MPI_Allgather();
            end_segment("main.1");
        }
        for (j=0; j < 10; ++j){
             start_segment("main.2");
             do_other_work();
             end_segment("main.2");
             while(k < otherRanks){
                start_segment("main.2.1");
                MPI_Sendrecv();
                end_segment("main.2.1");
             }
        }
        start_segment("final");
        MPI_Finalize();
        end_segment("final");
}
```

Figure 1: Segment Context Marking. We show a single function, main() with the instructions added to mark the segment contexts. We mark initialization, finalization, and all loops. The segment context names are hierarchical: the second loop is marked "main.2" and its subloop is marked "main.2.1". Segment marking is automated using a dynamic instrumentation library.

per-task trace prior to merging will reduce the application trace accordingly.

3.1 Trace Collection and Segments

We collected full traces of time stamped function entries and exits for the benchmarks and application as follows. First we insert segment markers into the source code that are repeated in the trace during execution. We define *segments* as follows: the initial segment starts at entry to `main`; for each program loop containing at least one measured event, we stop the current segment before the loop starts, start a new segment at the top of each loop iteration, stop the segment at the bottom of the loop iteration, and start a new segment after the last iteration of the loop completes; and end the final segment at program termination. The *segment context* is the section of code, for example, the main.1 loop in Figure 1. We used the dynamic instrumentation library Dyninst [15] to instrument the full application for both function entry and exit tracing as well as inserting segment begin and end markers. The simple benchmarks were marked manually.

We compare the segments for each context pair wise to determine if they are similar. If they are, we say that the segments *match* and retain a single representative segment. Each segment s_i contains an ordered list of events $E_i = \{e_0, e_1, \ldots, e_m\}$. We maintain a list *storedSegments*, which contains the segments that represent the performance behaviors in the execution, and a list *segmentExecs* that holds the starting times and identifier of each representative segment so that we can later recreate a full trace. Given an equivalence operator ≈ for some similarity metric, and a segment s_{new} that has events E_{new} the algorithm comparing segments is as follows:

For i = 0 to len(E_{new}):
    $E_{new}[i].start = E_{new}[i].start - s_{new}.start$
    $E_{new}[i].end = E_{new}[i].end - s_{new}.start$
$s_{new}.end = s_{new}.end - s_{new}.start$
match = False
For i = 0 to len(*storedSegments*):
    s_{stored} = *storedSegments*[i]
    match = *compareSegments*(s_{new}, s_{stored})
    If *match* = True:
        segmentExecs = *segmentExecs* ∪ ($s_{stored}.id, s_{new}.start$)
        break
If not *match*:
    $s_{new}.id$ = *getNewId*()
    segmentExecs = *segmentExecs* ∪ ($s_{new}.id, s_{new}.start$)
    $s_{new}.start$ = 0
    storedSegments = *storedSegments* ∪ s_{new}.

Boolean compareSegments(s_{new}, s_{stored}):
    If $s_{new}.context \neq s_{stored}.context$: return False
    If len(E_{new}) ≠ len(E_{stored}): return False
    For i = 0 to len(E_{new}):
        If $E_{new}[i].id \neq E_{stored}[i].id$: return False
    If $s_{new} \approx s_{stored}$: return True
    Else: return False

Note that a segments match requires that segments have the same context and the same number of events occurring in the same order. We give examples of segment matching in Figure 2.

3.2 Similarity Metrics

We used several methods to decide the similarity of segments. Each of these is described below. Our choices were inspired by methods used by other researchers to reduce traces (See Section 2.). They fell into two categories: distance methods and iteration-based methods.

3.2.1 Distance Methods

The distance methods produce a difference measure, which is then compared against a user-supplied threshold to determine the presence or absence of a match. Several of the difference methods are standard methods for computing distances between values and sets of values. We use the relative difference (*relDiff*), absolute difference (*absDiff*), and three variations on the Minkowski distance (*Manhattan*, *Euclidean*, *Chebyshev*), and wavelet transforms (*avgWave*, *haarWave*).

***relDiff*.** We compare the relative differences between each event measurement against a user-defined threshold; if greater, the events are not equal:

$$relDiff(x_1, x_2) = \frac{|x_1 - x_2|}{\max(x_1, x_2)}.$$

To see how *relDiff* matches segments, we consider our example in Figure 2. We compute the relative differences between each of the paired measurements in the segments. If any are above our chosen threshold, say 0.5, then the match fails. Comparing s2 with s1, we first compare the start times of the `do_work` event: x_1=1 and x_2=1, with relative difference 0. Since the relative difference is less than 0.5, we continue on computing relative differences. Next we check the end times for the `do_work` event. Here we compute a relative difference: x_1=17 and x_2=40, giving a relative difference of 0.58. This is above our threshold, so the segments do not match. When we compare s2

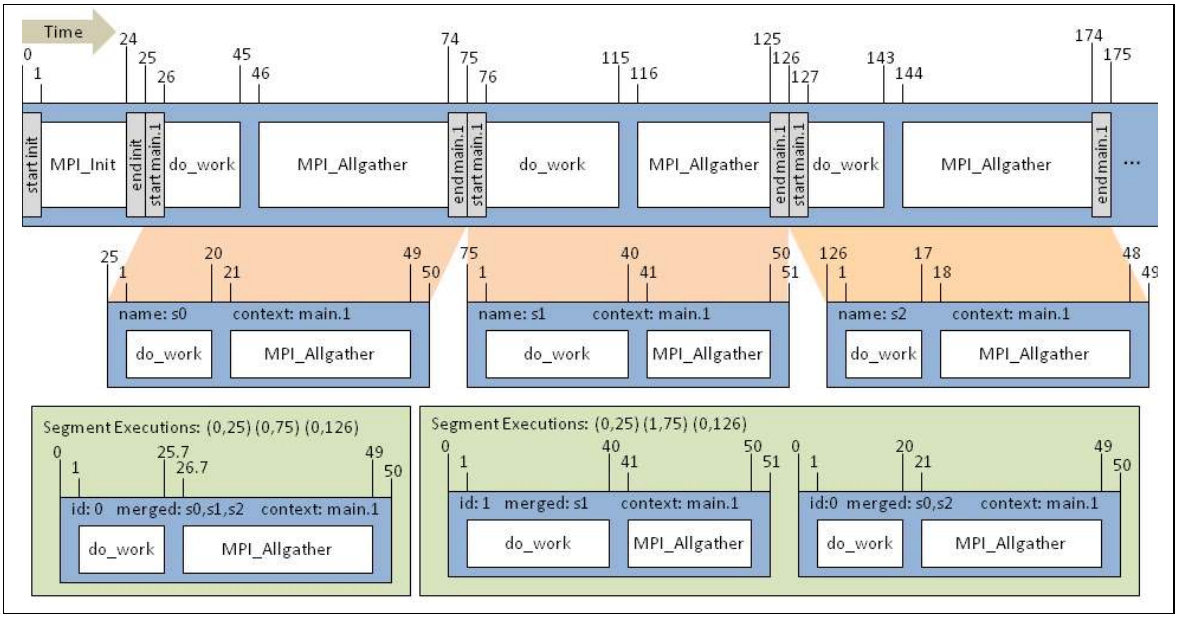

Figure 2: Trace and Segments Example. Here we show a portion of an example trace and three segments to illustrate segment matching. The top bar represents a portion of a trace for the program in Figure 1. Time increases from left to right, and time values are indicated above the bar. Segments markers are shown as light gray rectangles with vertical text that indicates the context of the segment. Events are shown in white boxes. Below the trace, we show the result of segmentation. In each of the three segments, the time stamps for the events and ending time of segments are adjusted relative to the start time of the segment. We name the segments s0, s1, and s2. In the bottom row, we show two examples of segment matching (See Section 3.2.).

with s0, we find that no differences are greater than 0.15 (x_1=17, x_2=20), so the segments match. The new segment is discarded since its behavior is reflected in the measurements in s0.

The relative difference function compares each measurement with its paired counterpart in isolation. The computed difference is proportional to the magnitude of the paired measurements, meaning that larger differences between larger measurements don't overshadow differences in smaller measurements. Because the difference between each measurement pair will be judged in isolation, the relative difference should be one of the strictest difference criteria in our set. The choice of threshold used will have a large bearing on the degree of matching, and hence on the reduction in file size.

One problem with *relDiff* appears when comparing time stamps in a series. For example, assume the threshold for comparing time stamps is 0.25. When we compare events that start at times 1 and 2, the relative difference is $\frac{2-1}{2} = 0.5$. This would result in a failure to match the events even though there is a difference of only one time unit between the events. In contrast, if we compare events that start at 100 and 125, the relative difference is 0.2, which is a match even though there is a difference of 25 time units. We expect *relDiff* to produce reduced traces with a low amount of error, but with less file size reduction.

absDiff. As with the *relDiff*, each measurement is compared with its counterpart. A fixed size difference, determined by a threshold, is allowed for each measurement pair. Using our example segments in Figure 2, and a threshold of 20, we see that s2 will not match s1, because the end times of `do_work` are 23 time units apart. However, there are no differences larger than 3 between s2 and s0, so those two segments match. The threshold choice has an impact on file size and accuracy. We expect this method to produce fairly accurate results, especially with respect to the timing of events across processes, because unlike *relDiff* it will not have an unfair bias towards events that occur later in the trace.

Manhattan, *Euclidean*, and *Chebyshev***.** We compute the Minkowski distance between segments using the formula in Eq. 1. If the distance is greater than a user-specified threshold multiplied by the maximum value in the event measurements, then the events are not equal. The Manhattan, Euclidean, and Chebyshev distances are special cases of the Minkowski distance, with *m* equal to 1, 2, and $\lim_{m\to\infty}$ respectively [13]. The Chebyshev distance is defined to be the largest difference between two measurements.

Eq. 1

$$L_m = \left\{\sum_{i=1}^{n} |x_i - y_i|^m\right\}^{1/m}$$

Using our example in Figure 2, to compare s2 and s1, we create a vector of the measurements for s2, (49, 1, 17, 18, 48), and one for s1, (51, 1, 40, 41, 50). The Manhattan, Euclidean, and Chebyshev distances between these vectors are 50, 32.6, and 23, respectively. The largest measurement in the pair of vectors is 51. If we choose a threshold of 0.2, then the highest the computed distance can be for a match is 10.2, so s2 and s1 will not match using any of the Minkowski distances. When we compare s0, (50, 1, 20, 21, 49), with s2, we get distances of 8,

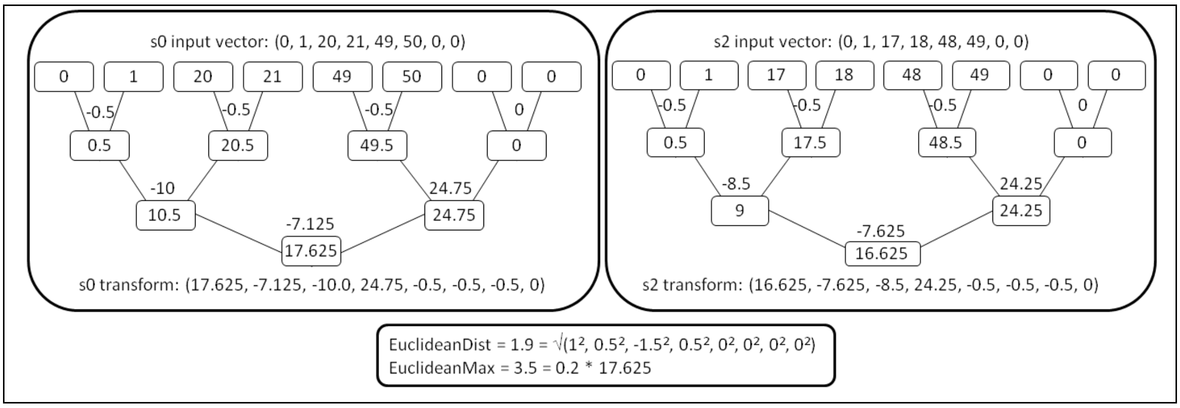

Figure 3: Wavelet Transform Example. Here we show two example average wavelet transforms. We iteratively compute averages (shown in boxes) and differences (shown between edges) for pairs of numbers, starting with the original vector. To compare the two transforms of s0 and s2, we compute the Euclidean distance between them and compare it against a threshold (0.2) multiplied by the largest element in the vectors (17.625).

4.5, and 3. The maximum value in the two vectors is 50, so the highest the distances can be for a match is 10. This means that s2 would match s0 for each of these distance metrics.

There are several issues to consider for the Minkowski distances:

- As *m* increases in the Minkowski distance (See Eq. 1.), the influence of the larger differences increases, and the influence of the smaller differences decreases. In the extreme case of the Chebyshev distance, only the maximum difference has any bearing on the distance value. As the number of measurements being compared increases, the values of the Manhattan and Euclidean distances increase. Given vectors of constant differences greater than 1, the Manhattan distance increases quite rapidly linearly, and the Euclidean distance increases in the manner of $\sqrt{x}$. If the differences are all between 0 and 1, the computed distances increase more slowly.
- When time stamp values are being compared, e.g. start time and end time for events, the values are always increasing within a segment. This means that longer segments are judged less critically than shorter segments, because the maximum values that are compared with the distance measurement are larger.

Based on these trends, we expect that the Manhattan distance would give the most accurate results, because it gives larger weight to the smaller differences. The Euclidean distance would give slightly less accurate results, given the bias towards larger differences. The Chebyshev distance would be least accurate, because it only accounts for the largest difference measure.

Wavelet transform. The discrete wavelet transform iteratively decomposes a signal of size L into two subsignals of size L/2. The first L/2 values give the trends in the original signal, and the second L/2 values give the fluctuations. Intuitively, it computes the averages and differences between pairs of numbers [17]. We give examples of transformations in Figure 3.

We use two wavelet transforms in our experiments: the average transform described in Figure 3 (*avgWave*), and the Haar transform (*haarWave*). The Haar transform is very similar to the average transform, with the only difference being that the averages and differences are multiplied by $\sqrt{2}$ [33]. For example, the trends computed in step 3 in Figure 3 would be ($9\sqrt{2}$, $24.25\sqrt{2}$). For our implementation, we construct a vector for each of the segments to be compared. The first element of each vector is the relative start time of the segment, which is 0 in all cases. This is followed by the event entry and exit time stamps for all events in the segment. The last element is the exit time of the segment. Both transforms require an input vector with a length that is a power of two. We allocate space for the vector so that its length is the next power of two after the number of time stamps in the vector. We zero-pad the vector after the last time stamp element to the end. To compare transformed vectors, we compute the Euclidean distance between them [5] and compare it against a threshold multiplied by the largest value in the pair of transformed vectors. In Figure 3, we show an example comparison of the segments s0 and s2 from Figure 2. Because the computed Euclidean distance, 1.9, is less than the maximum allowed, 3.5, s0 and s2 match.

For both transforms, the values in the transformed vectors will be smaller than the values in the original vectors. The Haar transform has several properties that the average transform does not, including preservation of the Euclidean distance [5]. However, its values will be larger than those of the average transform since all values are multiplied by $\sqrt{2}$. For the Haar transform, we expect more accurate results than from the Euclidean distance because the maximum value in the transformed vector will be smaller than the maximum value in the original vector, so the threshold test will be stricter. The values in the vector from the average transform will be smaller still; however, the Euclidean distance is not preserved, so the potential exists for a less strict test than the Euclidean distance.

3.2.2 Iteration-based Methods

We chose two iteration-based methods: *iter_k* and *iter_avg*.

***iter_k*.** Only keep a fixed number of each traced segment of code. We expect this method to produce small data files. For our example in Figure 2, if we chose k=3, we would keep all three copies of the main.1 segment in the list of stored segments. However, if k=2, then we would keep s0 and s1 and discard s2.

***iter_avg*.** Keep the average measurements for each traced section of code. We expect this method to produce the smallest data sizes, since segments with the same context and same events will always match. To illustrate this method, we use the segments in Figure 2 and the stored segments scenario on the left. For this method, we never have more than one copy of the main.1 segment, and end up with a single copy of the main.1 segment that contains averages of the values of s0, s1, and s2.

We expect that these methods will produce fairly accurate data for applications that have little behavior variability, but poorly for applications that do have performance variabilities.

4. EVALUATION METHODOLOGY

In this section we detail our framework for the evaluation of similarity metrics. We investigate traces collected for a set of benchmarks with known behaviors, and for a full application, running on a Linux cluster. Our evaluation focuses on three metrics: file size reduction, amount of error in the trace, and retention of performance trends. For file size reduction we simply compare the sizes of the reduced traces to the full-sized traces from which they were derived. We calculate the trace error by recreating an approximated full-sized trace from the reduced version, then comparing it to the actual full trace. We evaluate retention of performance trends by feeding the actual and approximated full traces into a performance analysis tool and examining any differences in the results.

4.1 Benchmarks

We crafted our benchmarks to represent classes of performance behaviors that occur in parallel programs on high end systems. These performance behaviors can appear with a high degree of regularity, sporadically, or progressively change over the iterations in the execution. To reflect this, we created a set of regularly behaving benchmarks, a set of irregularly behaving benchmarks, and a benchmark that simulates dynamic load balancing. Because we know the behavior patterns in each benchmark, we can evaluate how well each of the methods retains the performance behaviors.

We used the APART Test Suite (ATS) to create our benchmarks. The ATS a collection of utilities designed to create programs with known behavior for testing parallel performance tools [11]. We chose behavior patterns from the ATS that represent performance problems that require trace data for correct diagnosis. For parallel programs, these performance behaviors fall into four categories based on the communication pattern being used. We describe these communication patterns here using MPI functions as examples.

- N →1. *N processes send data to 1 process.* If any of the sending processes are late, then the receiving process blocks, waiting for them to execute the send operation. Example MPI functions for this pattern are `MPI_Reduce` and `MPI_Gather`, with corresponding performance behavior problems *early_reduce* and *early_gather*.
- 1 →N. *1 process sends data to N processes*. If the sending process is late, then all N receiving processes will block until the send is executed. Example functions are `MPI_Bcast` and `MPI_Scatter`. The corresponding performance problems are *late_broadcast* and *late_scatter*.
- 1 →1. *1 process sends to 1 process.* There are two cases. In the case of a non-blocking send and a blocking receive, if the sending process is late, the receiving process will block. In the case of a synchronous send, the sending process will block if the receiving process is late. Example communication routines are `MPI_Ssend` and `MPI_Recv`, with corresponding performance problems *late_receiver* and *late_sender*.
- N →N. *N processes send to N processes*. Here, all N processes depend on all other processes involved in the communication to proceed. If any of the N are late, then the rest of the processes block until all have reached the communication routine. An example is `MPI_Barrier` with corresponding performance problem *imbalance_at_barrier*.

Benchmarks with Regular Behavior. We chose five example benchmarks provided with ATS with regular behavior: early_gather, imbalance_at_mpi_barrier, late_receiver, late_sender, and late_broadcast. Each of the benchmarks simulates a program with the given behavior problem with the same severity in each iteration. In other words, all iterations of each program will exhibit the performance problem and all iterations should be very similar. All runs had 8 processes.

We expect the similarity methods to do relatively well on this set of benchmarks since the iterations have regular behavior. They should be able to find a large number of segments matches and still retain the correct performance behaviors.

Benchmarks with Irregular Behavior. For this category, we used ATS to create new benchmarks with irregular behavior. The benchmarks simulate the system interference identified by Petrini et al. when they ran an application on ASCI Q [27]. The system interference prevented the application from scaling as predicted. The benchmarks contain iterations with work periods that last approximately 1 *ms* followed by a communication step, using the communication patterns described previously. The load for each process is constant in each iteration and across processes: the only performance problem comes from the interference. We simulated the system noise using timers to interrupt the processes as described by Petrini et al. We used two simulation scenarios. The first was a 32-process run, with each of the 32 processes simulating the interrupts specific to the 32 nodes in an ASCI Q cluster. The second was also a 32-process run, but with the simulated amount of system interruptions that would occur if there were 1024 processes in the run. When we refer to the benchmarks in the first category, we use the communication pattern and either a *_32* or a *_1024*, to indicate whether 32 or 1024 processes were simulated, respectively.

For these benchmarks, we expect the methods to find a high number of matches, since most iterations are very similar. However, it will be important that they don't falsely match undisturbed and disturbed iterations, as this has the potential to mask or amplify the periodic behavior changes due to the simulated interruptions.

Dynamic Load Balancing. Here, we used ATS to create a program that simulates an application that does dynamic load balancing. For this benchmark, the performance of the iterations starts at about 1 *ms* and gets progressively worse, with one-half of the processes doing more work each iteration and the other half doing less work in each iteration, until the "load balancer" is triggered. The "load balancer" readjusts the amount of work on each processor to be equal. The performance problem exhibited by this program is *imbalance at mpi all to all*, which falls in the N-to-N communication category. This benchmark is referred to as *dyn_load_balance* and was run with 8 processes.

For this benchmark, we expect less overall matching since behavior changes with each iteration and very close performance behaviors reoccur only after each simulated load balance. Here it will be important that the similarity methods do not match segments with larger differences because the load imbalance may no longer be apparent in the reduced trace.

4.2 Application

We chose Sweep3D 2.2b, a structured mesh application that computes a 1-group time-independent discrete ordinates three-dimensional Cartesian geometry neutron transport problem [1]. Structured mesh applications have a regular partitioning of the data, where all interior data blocks have equal numbers of neighbors. It is likely that the performance will be very regular over the course of the program, which means that the reduction methods should be able to find a large number of segment matches without introducing a large amount of error. We collected traces for two runs of this application: an 8-process run with input file input.50, *sweep3d_8p*; and a 32-process run with input input.150, *sweep3d_32p*.

4.3 Evaluation Criteria

We chose four criteria to evaluate the metrics: percentage of full trace file size, degree of matching, approximation distance, and retention of correct performance trends.

4.3.1 Percentage of Full Trace File Size

We present the savings in file size as a percentage of the full, non-reduced trace file, as a relative measure of size reduction.

4.3.2 Degree of Matching

The degree of matching metric is a measure of how many segment matches occurred. We define it to be the ratio of the number of matches to the number of possible matches. The number of possible matches is limited by the structure of the program. For example, some portions of the code may only execute one time, e.g. an initialization step, and will not match any other event sequence in the trace. A possible match between segments exists if: the segments represent the same code location; they contain the same events in the same order; and all message passing calls and parameters are the same.

4.3.3 Approximation Distance

We estimate the error in the trace by recreating a full trace from the reduced trace and comparing each time stamp with its counterpart in the original full trace. The approximation distance metric tells what absolute difference 90% of time stamps had compared to the originals.[1]

4.3.4 Retains Correct Performance Trends

Arguably, the most important criterion for evaluating a trace matching metric for the purposes of performance analysis is deciding whether or not the reduced trace still indicates the same performance problems as the full trace. For example, if an analyst inspecting a full trace detects a late sender performance problem, the same problem should be detected in the reduced trace with approximately the same severity. The KOJAK tool set was developed to aid parallel performance analysts in the challenging task of performance diagnosis [34]. KOJAK's EXPERT tool reads in a trace file and produces a data file containing performance diagnoses. Each diagnosis consists of a metric, a code location, and a severity for each thread in the run [29]. KOJAK's CUBE tool reads in the analysis data and presents a visualization to the user, indicating the most important performance trends in the trace in a hierarchical manner.

We use the CUBE visualization tool to compare the performance diagnoses for the recreated traces against the diagnoses for the full trace (See Figure 4.). We determine whether a performance analyst would come to the same conclusions about the reduced trace as the full trace. If not, then the reduced trace is not adequate for performance analysis. We admit that this is a subjective test; however, we followed a set of guidelines when deciding if the diagnoses were sufficiently similar, so all the methods were subjected to the same criteria.

5. EVALUATION STUDIES

In this section, we present the results of two studies evaluating the similarity methods using the criteria and programs described in Section 4. We first present a threshold study for the similarity methods from the distance metric category. From this study, we choose a threshold for each of these methods that represents the best tradeoff in terms of file size reduction, measurement error, and retention of performance trends. In the second study, we present the results of a comparative study of the similarity methods, using the thresholds found to be best for each method in the threshold study.

5.1 Threshold Study

We investigated the behavior of the methods in reducing the traces of the benchmarks while varying the thresholds that determine whether two given segments should match or not match. The thresholds for *relDiff*, Minkowski distances, and the wavelet transforms were 0.1, 0.2, 0.4, 0.6, 0.8, and 1.0. The thresholds for *iter_k* were 1, 10, 50, 100, 500, and 1000, and for *absDiff* were powers of 10 from 10^1 to 10^6. Since no thresholds are used with the *iter_avg* method, it was not included in this study. The criteria we used to evaluate the methods were file size, approximation distance, and retention of performance trends (For file size reduction and approximation distance, see Figures 10-16 in the Appendix for the benchmarks and Figures 17-19 for sweep3d. For retention of performance trends, see Tables 1-18 in the Appendix.). For each method, we chose a representative threshold to be used when comparing the methods against each other.

relDiff. The file size for each benchmark and the sweep3d runs decreased relatively steadily with increasing threshold. The approximation distance remained small until the 0.8 threshold, after which there was a large jump for many of the benchmarks and sweep3d_32p. Performance trends were correctly retained for most programs up to a threshold of 0.8. Based on the jump in approximation distance and loss of performance trends after threshold 0.8, we chose 0.8 as the best threshold for relDiff.

absDiff. Here the file sizes for the benchmarks and sweep3d dropped off fairly quickly at a threshold of 100 and continued to decrease slightly with increasing threshold. The approximation distance stayed relatively low up to a threshold of 10^4, after which there was a sharp increase for several of the benchmarks and sweep3d_32p. Performance trends were retained for most

[1] When recreating full traces for the iter_k method, we used the last segment that executed of each pattern to fill in the segment executions that were not collected. Alternatives include using the average measurements from the k collected segments, or using the centroid of those k segments as determined by a clustering algorithm.

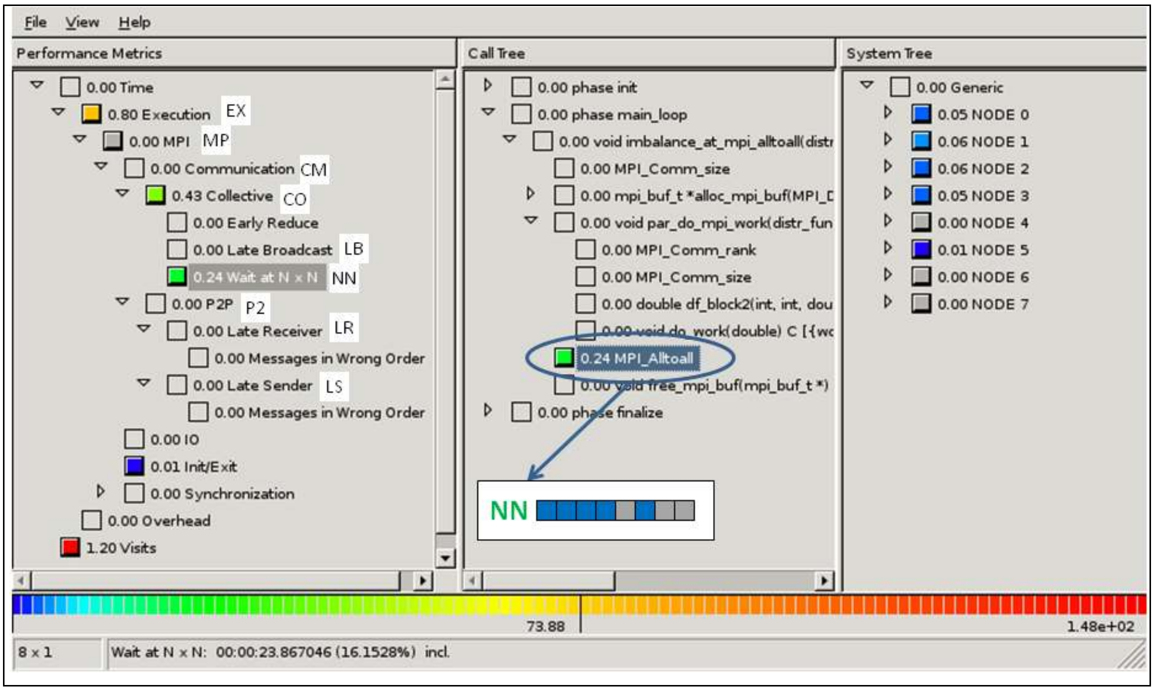

Figure 4: KOJAK Performance Analysis and Derivation of Our Performance Diagnosis Representation. Here we show a screenshot of KOJAK's EXPERT tool displaying the performance diagnosis for dyn_load_balance. The color bar on the bottom shows the severity levels, with blue being low and red high, and gray indicating 0 or close to 0. The left panel shows the performance metrics; the middle panel shows the code locations; and the right panel shows the processes. The color blocks next to each metric, code location, and process show the severity for the selected combination. Above, we have selected the function MPI_Alltoall and the "Wait at NxN" metric. This combination has green or "medium-low" severity and the severity is close to 0 for ranks 4, 6, and 7 and fairly low for ranks 0-3 and 5. We represent this diagnosis by abbreviating the metric name, e.g. NN for "Wait at N x N," coloring the metric abbreviation according to the severity indicated in the code location pane, and coloring squares for each process according to their severity levels. White squares indicate negative severities. We show the abbreviations we use for selected KOJAK metrics in white rectangles next to the metric names.

programs at a threshold of less than 10^3. Because the file sizes were relatively low and performance trends were retained at 10^3, we chose 10^3 as the representative threshold for *absDiff*.

Manhattan, Euclidean, and Chebyshev. When observing file sizes changes, the Manhattan and Euclidean methods behaved quite similarly; the Chebyshev method showed some differences. For the Manhattan and Euclidean methods with the regular benchmarks, the 1-to-1 irregular benchmarks, and sweep3d, file sizes decreased relatively steadily with increasing threshold; with the other irregular benchmarks, the file size decreased only slightly with increasing threshold, because a matching that was close to optimal was reached early, at a threshold of 0.1. For Chebyshev with the 1-to-1 irregular benchmarks and sweep3d, file size decreased with increasing threshold; with the regular benchmarks and remaining irregular benchmarks, file size was relatively constant with increasing threshold. For all three methods, we observed the following behavior in approximation distance: with the regular benchmarks, approximation distance was relatively constant with increasing threshold; with the 1-to-1 irregular benchmarks, approximation distance increased with increasing threshold; with the remaining benchmarks, the approximation distance remained low until after the threshold of 0.8, after which there was a large jump. For sweep3d and Manhattan and Euclidean, approximation distance increased with increasing threshold; for Chebyshev, the approximation distance was small and relatively constant until after the 0.8 threshold. For retention of performance trends, the Manhattan distance did well up to a threshold of 0.4, and the Euclidean and Chebyshev distances did well up to 0.2. We based our selection of best thresholds for these methods on the retention of performance trends metric, because we consider this metric to be the most important. We chose 0.4 as the best threshold for the Manhattan distance and 0.2 for the Euclidean and Chebyshev distances.

Wavelet Transforms. For all evaluation criteria, *avgWave* and *haarWave* performed similarly. For all programs, file sizes decreased with increasing threshold, up to the point of perfect matching, after which no further decrease in size is possible. The best threshold in this category appears to be 0.4 for both methods, because file size decrease levels off after this threshold. The approximation distance for both methods remained steady with increasing threshold for the regular benchmarks and the irregular N-to1, N-to-N, and 1-to-N benchmarks. The approximation distance increased with increasing thresholds for the irregular 1-to-1 benchmarks and sweep3d. The threshold 0.2 is best for approximation distance, because of the relatively higher values for the dyn_load_balance benchmark and sweep3d after this threshold. For the majority of programs, performance trends were retained for both methods at thresholds below 0.2. For these reasons, we chose 0.2 as the best threshold for the wavelet transform methods.

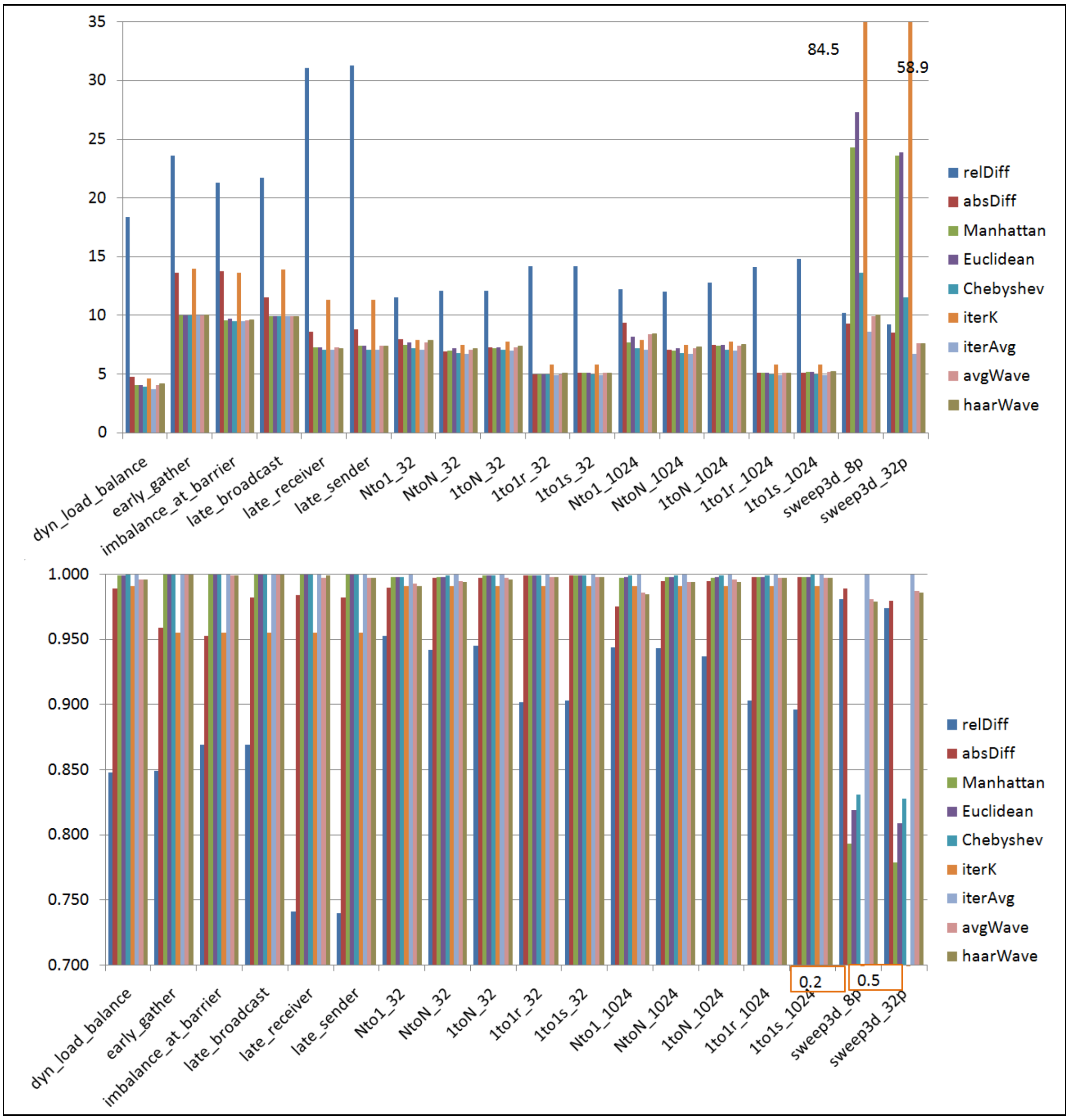

Figure 5: Percentage File Sizes and Degree of Matching.

iter_k. Generally speaking, there was an increase in file size and decrease in approximation distance with increasing k. Performance trends were retained for must programs up to threshold 10. The choice for the best k wasn't clear, but we chose k=10 as the best because the performance trends were retained for most programs at this threshold.

5.2 COMPARATIVE STUDY

In this section, we present comparative results for the different methods using size and degree of matching; approximation distance; and retention of performance trends as the evaluation criteria. Based on the results of the threshold study in Section 5.1, we present results for the best performing threshold for each method: 0.8 for *relDiff*, 1000 for *absDiff*, 0.4 for *Manhattan*, 0.2 for *Euclidean* and *Chebyshev*, 10 iterations for *iter_k*, and 0.2 for *avgWave* and *haarWave*.

5.2.1 Size and Degree of Matching

We present the data for reduction of traces for each method in Figure 5. The *iter_avg* method gives the best case values for this category, since exactly one segment is retained per loop with this method.

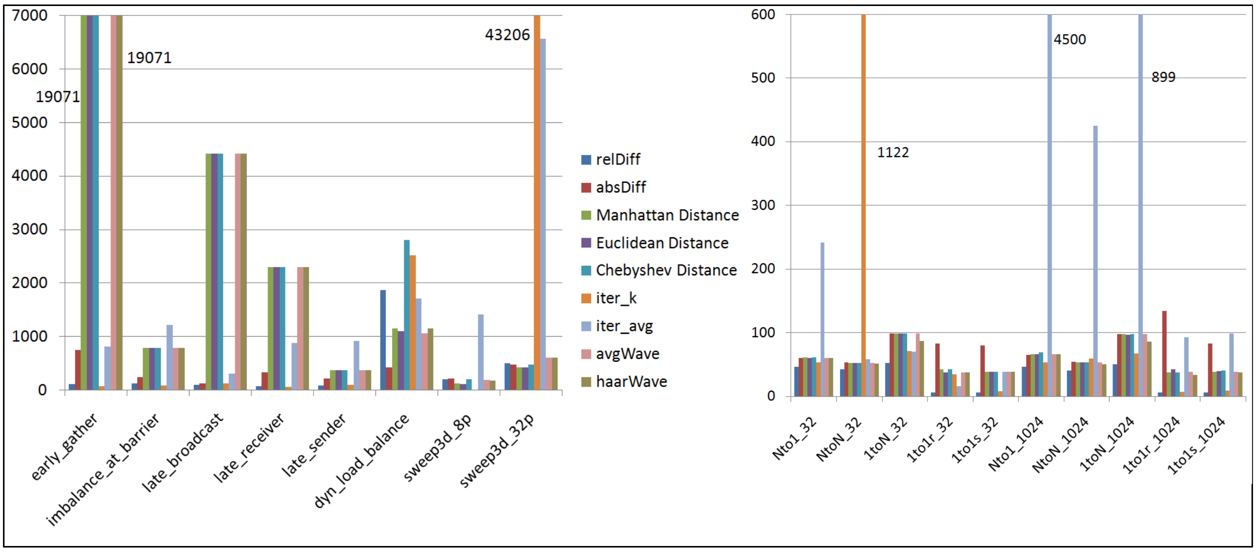

Figure 6: Approximation Distance Results for All Methods at Default Thresholds.

The benchmark data shows that for the most part, the degree of matching for each of the methods is greater than 0.9, meaning that greater than 90% of the segments were matched. Exceptions occur with *relDiff*, which had degree of matching scores as low as 0.74. *RelDiff* had the highest file sizes and lowest degree of matching scores. The next largest file sizes are generated with the *iter_k* method; however, they are not much higher than those for the other methods. The Minkowski distances, *avgWave*, and *haarWave* all have nearly identical results, with *Chebyshev* having a very slight advantage over the others. *AbsDiff* had only slightly larger file sizes than the Minkowski distances.

For sweep3d, the results are somewhat different. Because this application has very regular behavior, we expected the results to be similar to those of the benchmarks. However, because of the program structure, there are more segments, as well as differences within the segments, e.g. message passing parameters, that cause segments not to match. We see that *iter_k* performed the worst, with the highest file sizes and lowest degree of matching scores. This is because *iter_k* needed to keep 10 copies of each individual segment, regardless of how similar in performance they actually were, whereas the high degree of matching often results in fewer than 10 copies. The next worst performing were the Minkowski distances, again with *Chebyshev* having the smallest file sizes. The wavelet methods performed best, followed by *absDiff* and *relDiff*, each with very close to perfect matching and lowest possible file sizes.

The obvious best method in this category is *iter_avg*, since all segments match by definition. A comparison of the average file sizes for each of the other methods yields the following ranking: *avgWave*, *haarWave*, *Chebyshev*, *absDiff*, *Manhattan*, *Euclidean*, *iter_k*, *relDiff*.

5.2.2 Approximation Distance

Figure 6 shows the approximation distance results for each of the methods. High values for iter_k and iter_avg mean that there is irregularity in the execution that is not being captured in the iterations that are retained. High values for absDiff give a rough indication of the absolute difference of time stamps from the true values in the full trace. High values for the Minkowski and wavelet methods mean that there are high maximum values in the set of values being compared, relative to the distance between those values.

The methods show similar trends across the benchmarks with regular behavior. The *relDiff*, *absDiff*, *iter_k*, and *iter_avg* methods have consistently low values. The Minkowski distances, *avgWave*, and *haarWave* transform behave similarly, and have the highest values overall. The results for the dyn_load_balance benchmark show a different set of behavior, with *absDiff* having the lowest value, followed by *avgWave*, *Euclidean*, *Manhattan*, and *haarWave*. The interference benchmarks had lower overall approximation distance values than the other benchmarks, with similar results across the benchmarks. The worst performing methods in this case were *iter_avg* and *iter_k*. However, the approximation distance values are low in comparison to those for the other set of benchmarks.

The results for sweep3d show *iter_avg* performing the worst for the 8-process run, and *iter_k* and *iter_avg* the worst for the 32-process run, indicating that there are performance behaviors not being captured by those two methods.

The methods that performed the best in this category are *relDiff*, followed by *absDiff*, and then *iter_avg*. The rest of the methods allowed significant error into at least one of the reduced traces.

5.2.3 Retention of Performance Trends

We present summaries of the performance diagnoses given by KOJAK for selected benchmarks in Figures 7 and 8. We show how we derive the performance diagnoses charts and abbreviations for metric names in Figure 4. For the benchmarks with regular behavior, nearly all the methods performed quite

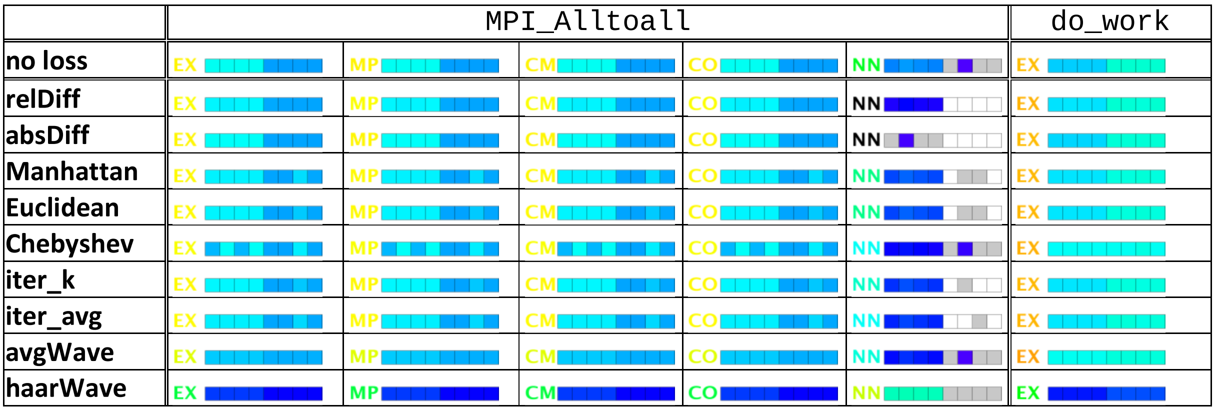

Figure 7: KOJAK Performance Trends for dyn_load_balance For Each Method at Default Thresholds. Here we show the results for each reduction method in the MPI_Alltoall and do_work functions. The first row shows the diagnoses for the full trace. Each box in a row shows a performance diagnosis for a single combination of metric and code location.

well. For late_receiver, all methods except *iter_avg* performed equally well, with all performance trends retained. The results for *iter_avg* with late_receiver showed differences significant enough that they may lead to an inaccurate performance assessment. For early_gather, all but the Minkowski distances, *avgWave*, and *haarWave* retained the correct performance trends. The results for imbalance_at_barrier showed that the Minkowski distances, *absDiff*, *iter_avg*, *avgWave*, and *haarWave* retained the performance trends, while *relDiff* and *iter_k* both showed a negative value for the major performance diagnosis. The amount of error introduced into the reduced traces caused time stamps to be skewed enough that the performance diagnoses resulted in negative values.

We show the major performance trends for dyn_load_balance in `MPI_Alltoall` and `do_work` as reported by the KOJAK tools for the full trace and all methods in Figure 7. The results for the no loss trace clearly indicate that the lower ranks are spending more time in `MPI_Alltoall`, because the upper ranks are spending more time in `do_work`. None of the methods gave perfect results for the dyn_load_balance benchmark; however, *absDiff*, *Manhattan*, *Euclidean*, *avgWave*, and *haarWave* gave the closest performance diagnoses because for the most part they maintained the performance differences due to load imbalance between the upper and lower ranks. Although *Manhattan*, *Euclidean*, *avgWave*, and *haarWave* lost the disparity in `do_work`, the diagnosis "Wait at NxN" is non-negative and maintains the disparity in behavior. *AbsDiff* maintained the disparity in performance in `do_work`, but reported that "Wait at NxN" was negative. All other methods lose the expected disparity in `do_work`.

For the interference benchmarks, all methods did pretty well on the N-to-1 and 1-to-N benchmarks, with the exception of *iter_avg*, which failed on three benchmarks, and *Chebyshev*, which failed on Nto1_1024. *AbsDiff* did less well on the 1-to-1 and N-to-N benchmarks. We show the data for 1to1r_1024 in Figure 8. *AbsDiff* picked up on the variations in the iterations due interference, which caused some performance diagnoses to be skewed in a positive or negative direction. The best performers for these benchmarks were *Manhattan*, *Euclidean*, and *avgWave*, followed by *relDiff*, and *haarWave*. *AbsDiff* and *iter_avg* both only showed correct diagnoses for one benchmark, 1to1r_32 and 1to1s_32, respectively.

For sweep3d_8p and sweep3d_32p, all methods but *iter_avg* and *iter_k* produced correct data. *Iter_k* showed a non-existent disparity in rank performance in `pmpi_recv` in sweep3d_8p and a greatly inflated severity in `pmpi_recv` in sweep3d_32p. *Iter_avg* showed a much lower severity in `sweep_` than did the no-loss trace for both sweep3d_8p and sweep3d_32p.

The best methods in this category were *Manhattan*, *Euclidean*, and *avgWave* which correctly diagnosed 17 out of the 18 execution traces. HarrWave did second best, correctly diagnosing 16. The rest of the methods in order were: *relDiff* (14); *absDiff* and Chebyshev (13); *iter_k* (12); and *iter_avg* (6). The relatively poor performance of *iter_k* in this category could be due to our choices in implementing this method[1]. It is possible that the first iterations are more subject to variabilities in execution, before the processes synchronize into their regular behavior patterns, and that the last segment is not the best choice as a fill in for missing segments. *AbsDiff* seemed to amplify differences in the traces with interference, while *iter_avg* seemed to smooth out behavior patterns.

5.2.4 Discussion

For *relDiff*, we expected low error and relatively large files, which is exactly what we found to be true. For *absDiff*, we expected low error. We did find that *absDiff* had lower error when compared to most methods. We expected the Minkowski distances would favor long segments and error would be lowest for *Manhattan*, followed by *Euclidean*, and highest for *Chebyshev*. While we did definitely see more error in the traces produced by the *Chebyshev* method, the differences in the results for the *Manhattan* and *Euclidean* methods were largely undistinguishable. We expected *iter_k* and *iter_avg* to produce low error traces for programs with regular behavior and for *iter_avg* to have the lowest overall file sizes. We indeed found that *iter_k* did well for regularly behaving programs and less well for programs with varying behavior patterns. *Iter_avg* produced better results for the regular benchmarks than the irregular ones; the averaging of measurements tended to cause loss of information needed for diagnosis. For *avgWave* and *haarWave*, we expected stricter comparisons than *Euclidean*.

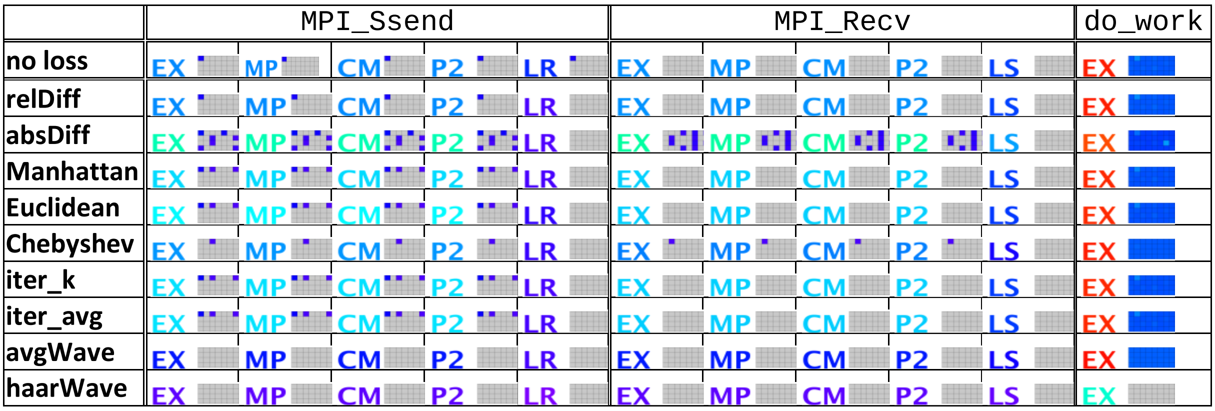

Figure 8: KOJAK Performance Trends for 1to1r_1024 for Each Method at Default Thresholds.

Indeed, the wavelet transforms produced slightly larger files for the benchmark traces; however, the reduced traces of sweep3d were smaller than those produced by *Euclidean*.

To determine best method for comparing traces, we take the highest ranking methods from each category and weigh the importance of each of the categories. The best methods from the size category were *iter_avg*, followed by *avgWave*, *haarWave*, and Chebyshev. Those from the approximation distance category were *relDiff* and *absDiff*, followed by *iter_avg*. Finally, the methods that best retained performance trends were *avgWave*, *Manhattan*, *Euclidean*, and *haarWave*. One could argue that the absolute most important criteria for judging these methods is whether or not they retain the correct performance trends, because that is the point of collecting the traces in the first place. However, almost equally important is the ability to collect, store, and analyze the trace data *at all*. Given that *avgWave* performed well in both the size and retention of performance trends categories, we choose *avgWave* as the best method of the ones studied for comparing traces.

6. CONCLUSIONS

We have developed a new methodology for evaluating definitions for similarity between event traces for the purpose of performance analysis. We identified criteria for comparing the similarity methods: file size reduction, degree of matching, approximation distance, and retention of correct performance trends. We applied these criteria, using benchmarks with known performance behaviors, as well as with the application sweep3d. Overall, the *avgWave* method had the best retention of performance behaviors and good trace file size reduction. The greatest trace file reductions were achieved with the *iter_avg* method; however, the error in those traces led to loss of important performance trends in the data. Because of this we found that using the *avgWave* method was the best trade-off in terms of error in the reduced trace and file size reduction.

Future directions for this work include investigating additional difference methods, such as trace sampling; and evaluating the methods against a richer set of full application traces.

7. ACKNOWLEDGMENTS

We thank our reviewers for their valuable suggestions; and Bernd Mohr and Felix Wolf for feedback on an earlier draft. This work was supported in part by the U.S. Department of Energy's Office of Science through the SciDAC2 award entitled Performance Engineering Research Institute, and by a Portland State University Faculty Enhancement Grant. We thank Lawrence Livermore National Laboratory for machine cycles and storage resources.

REFERENCES

[1] The ASCI sweep3D readme file. http://www.c3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html, January 2009.

[2] M. G. Aguilera, P. J. Teller, M. Taufer, and F. Wolf. A systematic multi-step methodology for performance analysis of communication traces of distributed applications based on hierarchical clustering. In *IPDPS*, 2006.

[3] L. Carrington, A. Snavely, X. Gao, and N. Wolter. A performance prediction framework for scientific applications. In *Workshop on Performance Modeling - ICCS*, 2003.

[4] M. Casas, R. M. Badia, and J. Labarta. Automatic phase detection of MPI applications. In C. H. Bischof, H. M. Bücker, P. Gibbon, G. R. Joubert, T. Lippert, B. Mohr, and F. J. Peters, editors, *PARCO*, volume 15 of *Advances in Parallel Computing*, pages 129–136. IOS Press, 2007.

[5] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133, Mar 1999.

[6] I.-H. Chung, R. E. Walkup, H.-F. Wen, and H. Yu. MPI performance analysis tools on Blue Gene/L. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC'06)*, page 123, New York, NY, USA, 2006. ACM.

[7] T. Fahringer, M. Gerndt, B. Mohr, F. Wolf, G. Riley, and J. Traff. Knowledge specification for automatic performance analysis. Technical Report Revised Edition, ESPRIT IV Working Group on Automatic Performance Analysis: Resources and Tools (APART), http://www.fz-juelich.de/apart-1/reports/wp2-asl.ps.gz, January 2001.

[8] F. Freitag, J. Corbalan, and J. Labarta. A dynamic periodicity detector: Application to speedup computation. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01), San Francisco, CA, USA*, April 23-17 2001.

[9] T. Gamblin, B. de Supinski, M. Schulz, R. Fowler, and D. Reed. Scalable load-balance measurement for SPMD codes. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[10] T. Gamblin, R. Fowler, and D. A. Reed. Scalable methods for monitoring and detecting behavioral equivalence classes in scientific codes. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'08), Miami, FL*, April 14-28 2008.

[11] M. Gerndt, B. Mohr, and J. L. Träff. A test suite for parallel performance analysis tools. *Concurrency and Computation: Practice and Experience*, 19(11):1465–1480, August 2007.

[12] W. Gu, G. Eisenhauer, K. Schwan, and J. Vetter. Falcon: on-line monitoring and steering of large-scale parallel programs. *Concurrency: Practice and Experience*, 10(9):699–736, Dec 1998.

[13] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2005.

[14] M. Hauswirth, A. Diwan, P. F. Sweeny, and M. C. Mozer. Automating vertical profiling. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pages 281 – 296, October 16-20 2005.

[15] J. Hollingsworth, B. Miller, and J. Cargille. Dynamic program instrumentation for scalable performance tools. In *Proceedings of Scalable High Performance Computing Conference, Knoxville, TN, USA*, pages 841–850, May 23-25 1994.

[16] T. Huffmire and T. Sherwood. Wavelet-based phase classification. In *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 95–104, New York, NY, USA, 2006. ACM.

[17] A. Jensen and A. la Cour-Harbo. *Ripples in Mathematics: The Discrete Wavelet Transform*. Springer-Verlag, 2001.

[18] L. V. Kalé, S. Kumar, G. Zheng, and C. W. Lee. Scaling molecular dynamics to 3000 processors with projections: A performance analysis case study. In *International Conference on Computational Science (ICCS 2003), Melbourne, Australia and St. Petersburg, Russia*, pages 23–32, June 2-4 2003.

[19] A. Knüpfer. A new data compression technique for event based program traces. In *International Conference on Computational Science*, pages 956–965, 2003.

[20] D. Kranzlmüller, S. Grabner, and J. Volkert. Event graph visualization for debugging large applications. In *SPDT '96: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools, Philadelphia, Pennsylvania, USA*, pages 108–117, 1996.

[21] D. Kranzlmüller, A. Knüpfer, and W. E. Nagel. Pattern matching of collective MPI operations. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04), Las Vegas, Nevada, USA*, June 21-24 2004.

[22] D. B. Lange and Y. Nakamura. Object-oriented program tracing and visualization. *IEEE Computer*, 30(5):63–70, May 1997.

[23] C. W. Lee, C. Mendes, and L. V. Kalé. Towards scalable performance analysis and visualization through data reduction. In *13th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2008) held in conjunction with IPDPS 2008*, 2008.

[24] K. Mohror and K. L. Karavanic. Towards scalable event tracing for high-end systems. In *High Performance Computing and Communications, Third International Conference (HPCC 2007), Houston, Texas, USA*, pages 695–706, September 26-28 2007.

[25] O. Nickolayev, P. Roth, and D. Reed. Real-time statistical clustering for event trace reduction. *International Journal of High Performance Computing Applications*, 11(2):69–80, 1997.

[26] M. Noeth, F. Mueller, M. Schulz, and B. R. de Supinski. Scalable compression and replay of communication traces in massively parallel environments. In *21th International Parallel and Distributed Processing Symposium (IPDPS'07)*, March 2007.

[27] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC'03), Phoenix, Arizona, USA*, page 55, November 15-21 2003.

[28] P. Ratn, F. Mueller, B. R. de Supinski, and M. Schulz. Preserving time in large-scale communication traces. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 46–55, New York, NY, USA, 2008. ACM.

[29] F. Song, F. Wolf, N. Bhatia, J. Dongarra, and S. Moore. An algebra for cross-experiment performance analysis. In *Proc. of the International Conference on Parallel Processing (ICPP)*, pages 63–72, Montreal, Canada, August 2004. IEEE Society.

[30] D. P. Spooner and D. J. Kerbyson. Performance feature identification by comparative trace analysis. *Future Generation Comp. Syst.*, 22(3):369–380, 2006.

[31] E. Thereska, B. Salmon, J. D. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger. Stardust: tracking activity in a distributed storage system. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2006, Saint Malo, France*, pages 3–14, June 26-30 2006.

[32] J. Vetter. Dynamic statistical profiling of communication activity in distributed applications. In *Proceedings of ACM SIGMETRICS 2002 International Conference on Measurement and Modeling of Computer Systems, Marina Del Rey, CA, USA*, pages 240–250, June 15-19 2002.

[33] J. S. Walker. *A Primer on Wavelets and Their Scientific Applications*. Chapman & Hall/CRC, 2008.

[34] F. Wolf, B. Mohr, J. Dongarra, and S. Moore. Automatic analysis of inefficiency patterns in parallel applications. *Concurrency and Computation: Practice and Experience*, 19:1481–1496, 2007.

[35] J. C. Yan, H. H. Jin, and M. A. Schmidt. Performance data gathering and representation from fixed-size satistical data. Technical Report NAS-98-003, NASA Ames Research Center, 1998.

[36] J. C. Yan and M. Schmidt. Constructing space-time views from fixed size trace files – getting the best of both worlds. In *Parallel Computing: Fundamentals, Applications and New Directions, Proceedings of the Conference (ParCo'97), Bonn, Germany*, pages 633–640, September 19-22 1997.

APPENDIX

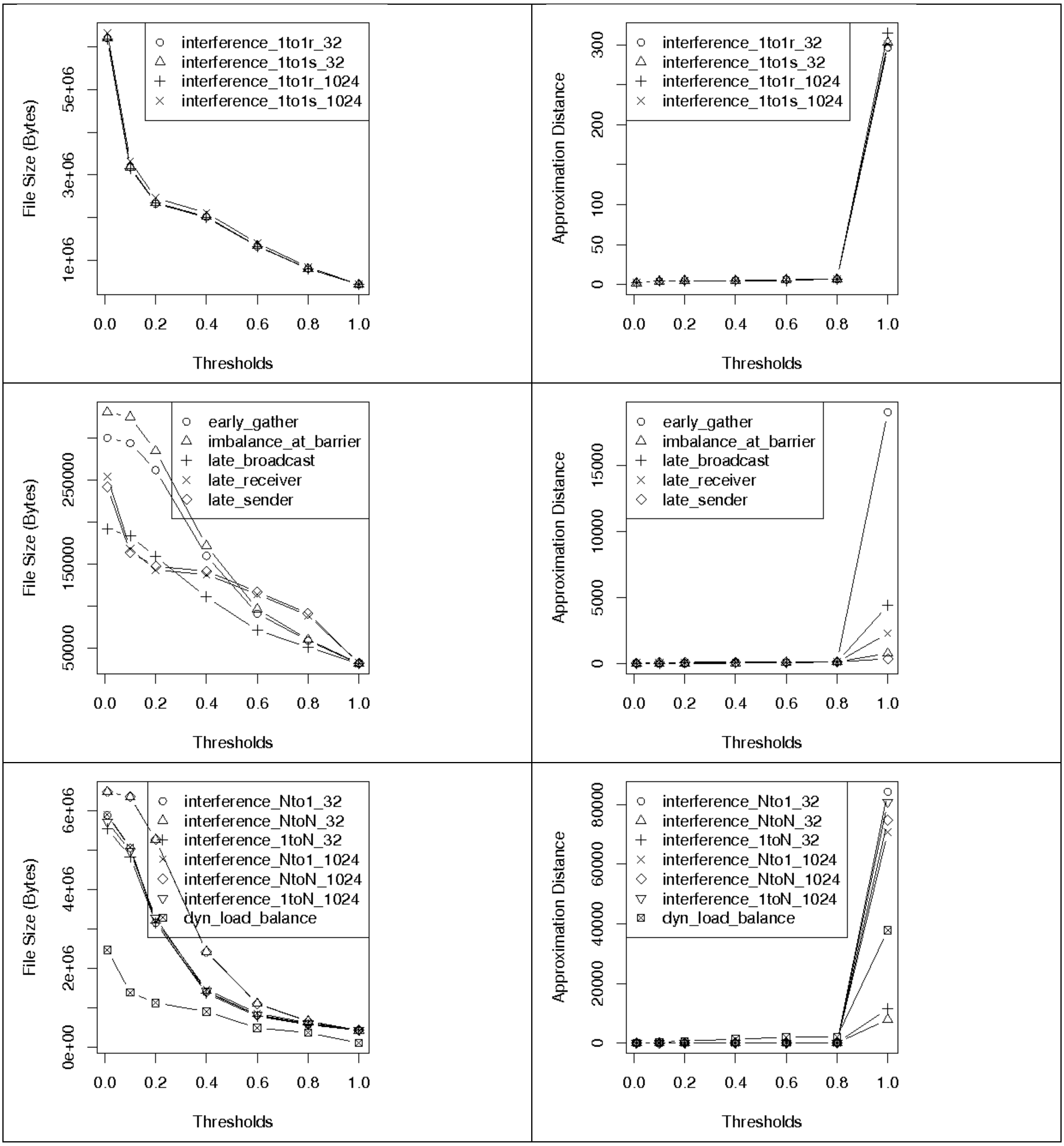

Figure 9: File Size and Approximation Distance for Varying Duration Thresholds and Relative Distance

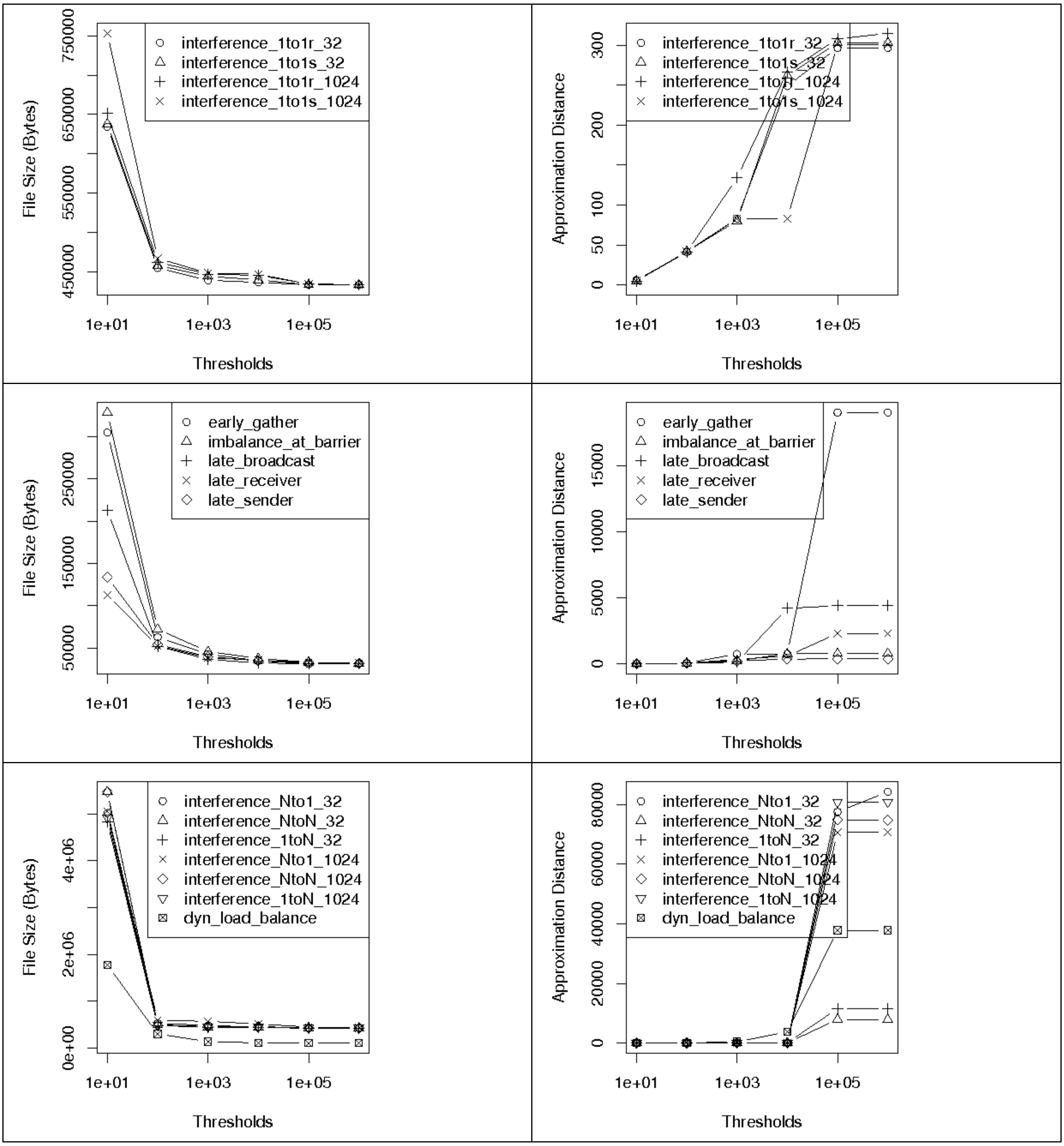

Figure 10: File Size and Approximation Distance for Varying Threshold and Absolute Distance

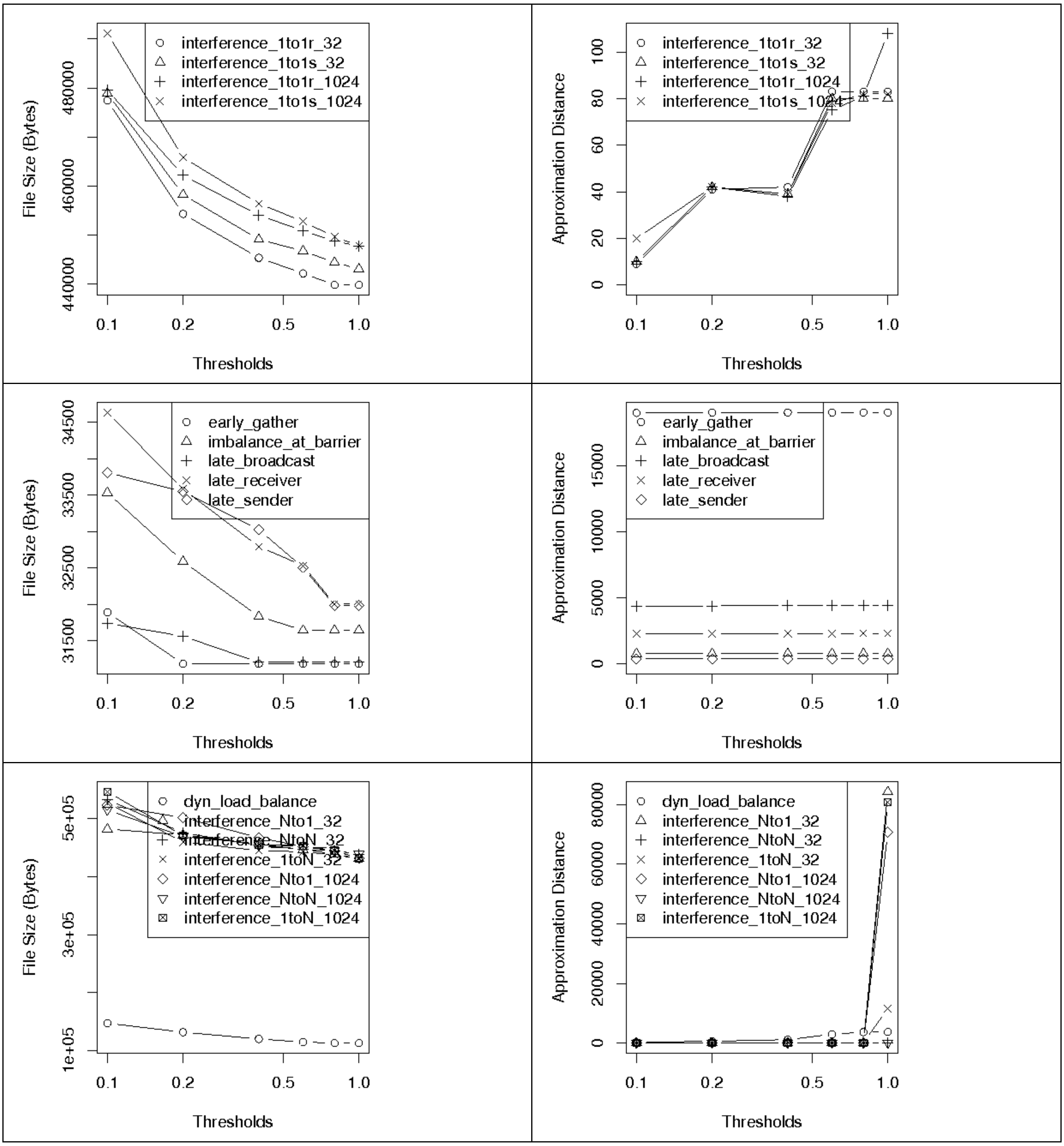

Figure 11: File Size and Approximation Distance for Varying Threshold and Manhattan Distance

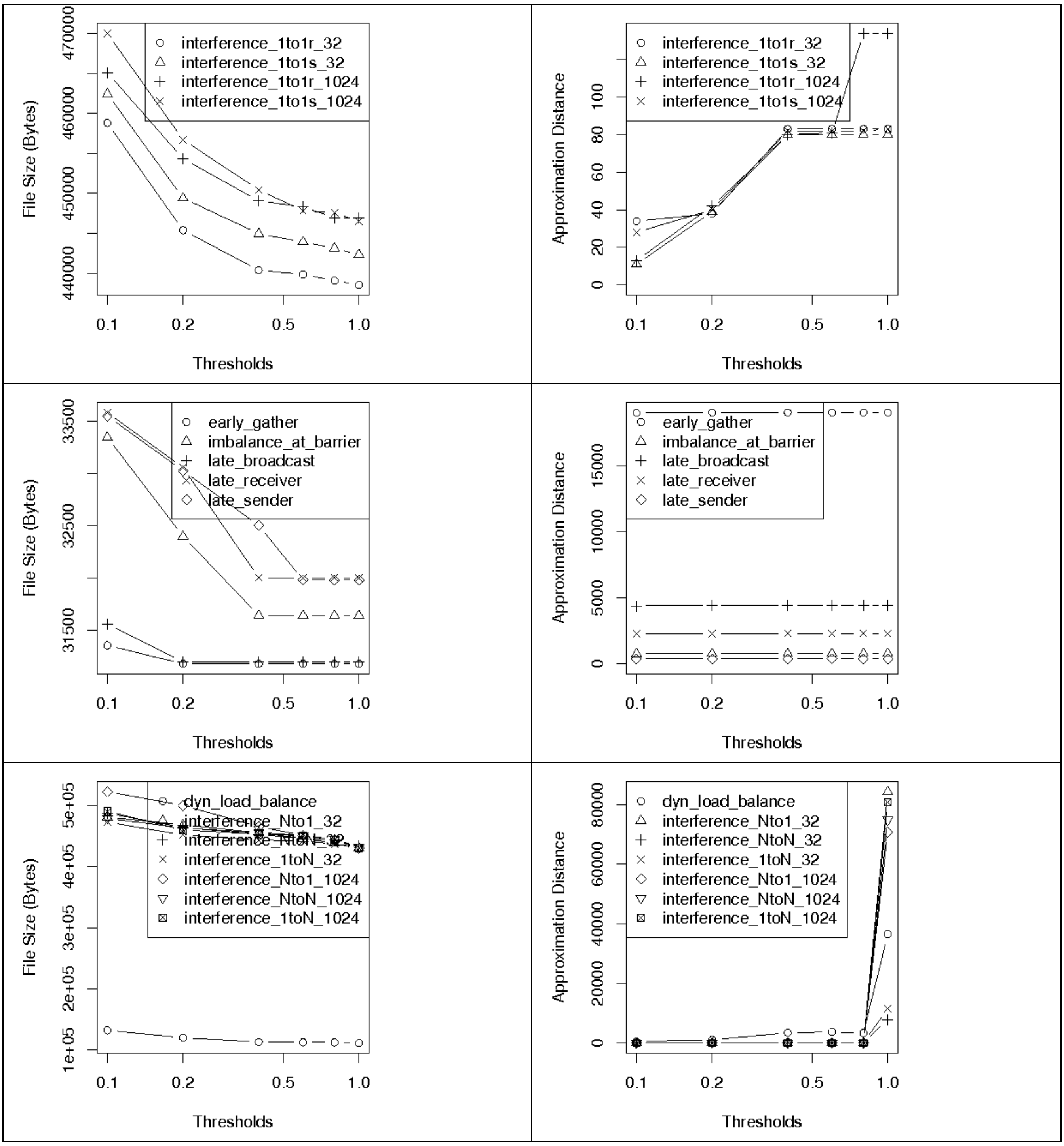

Figure 12: File Size and Approximation Distance for Varying Threshold and Euclidean Distance

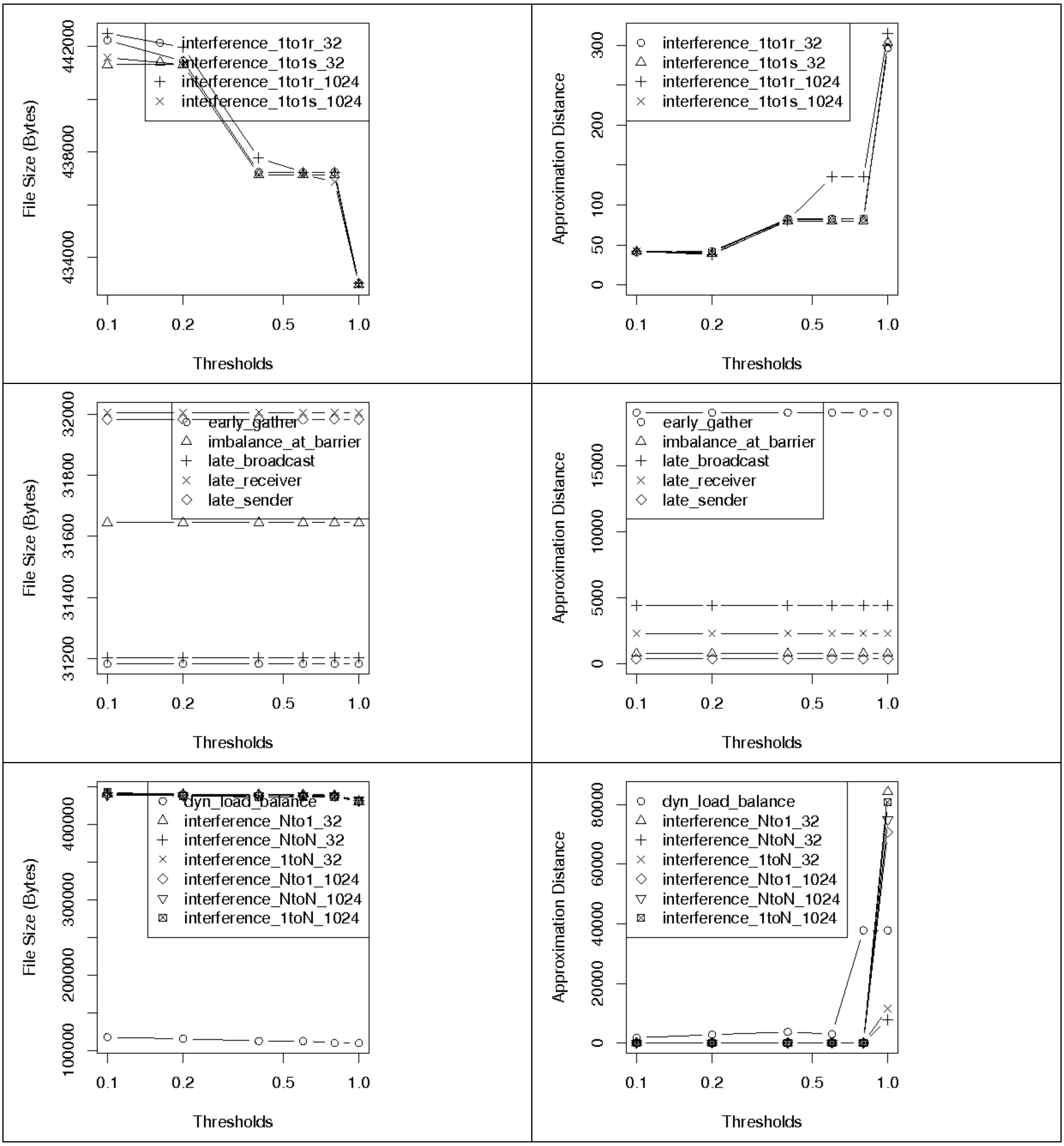

Figure 13: File Size and Approximation Distance for Varying Threshold and Chebyshev Distance

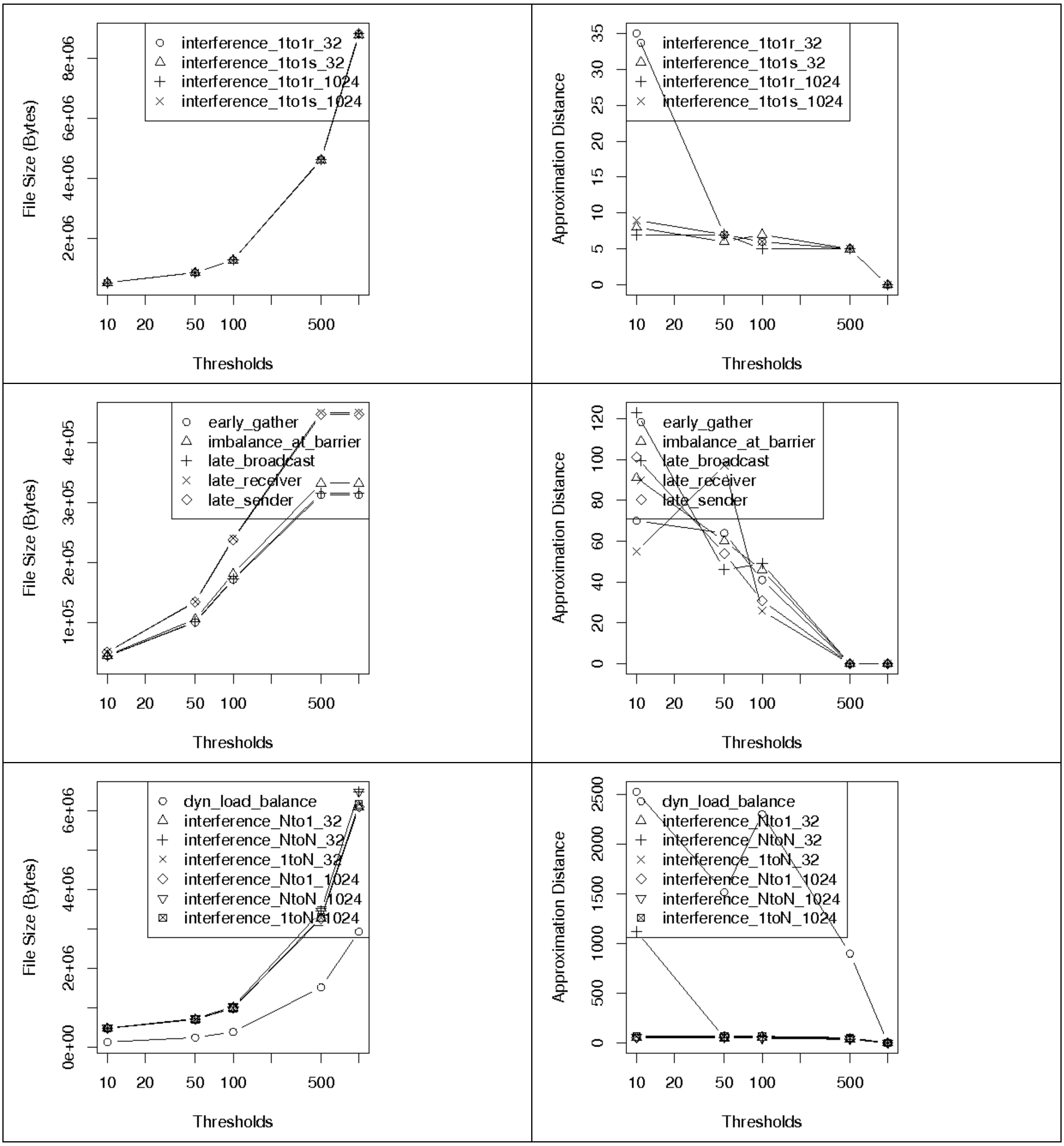

Figure 14: File Size and Approximation Distance for Varying Threshold and Keep k Iterations

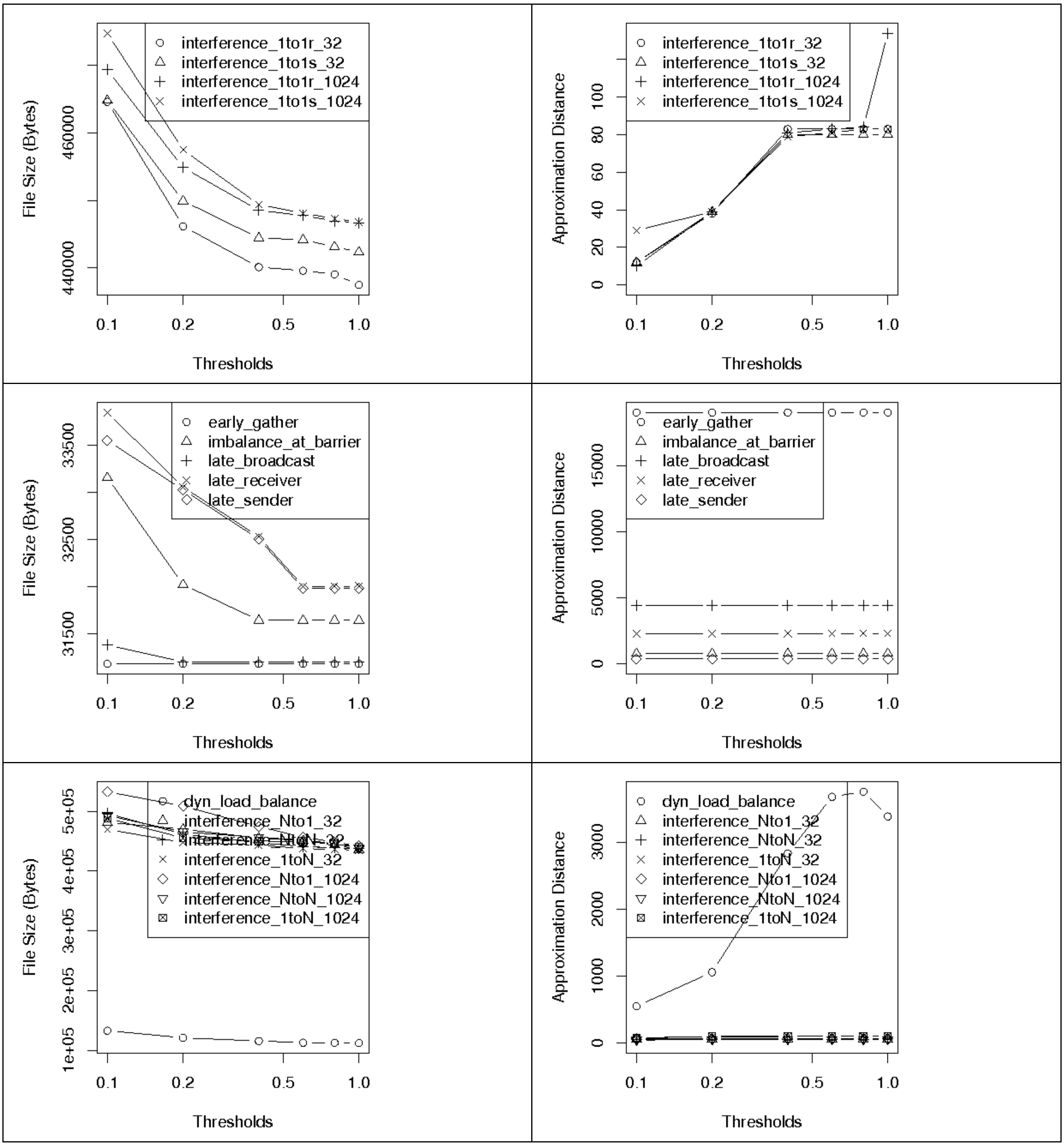

Figure 15: File Size and Approximation Distance for Varying Threshold and Average Wavelet Transform

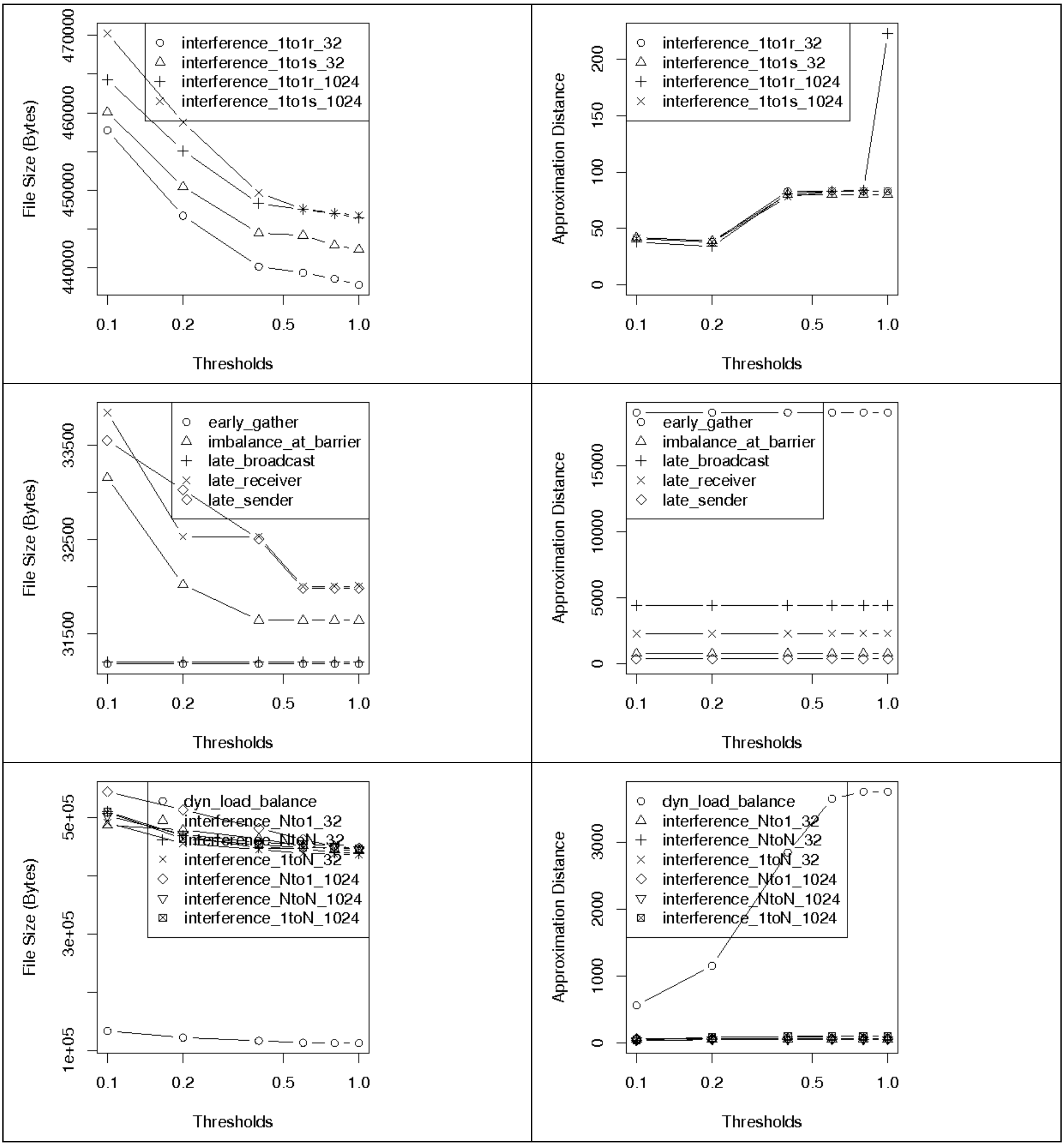

Figure 16: File Size and Approximation Distance for Varying Threshold and Haar Wavelet Transform

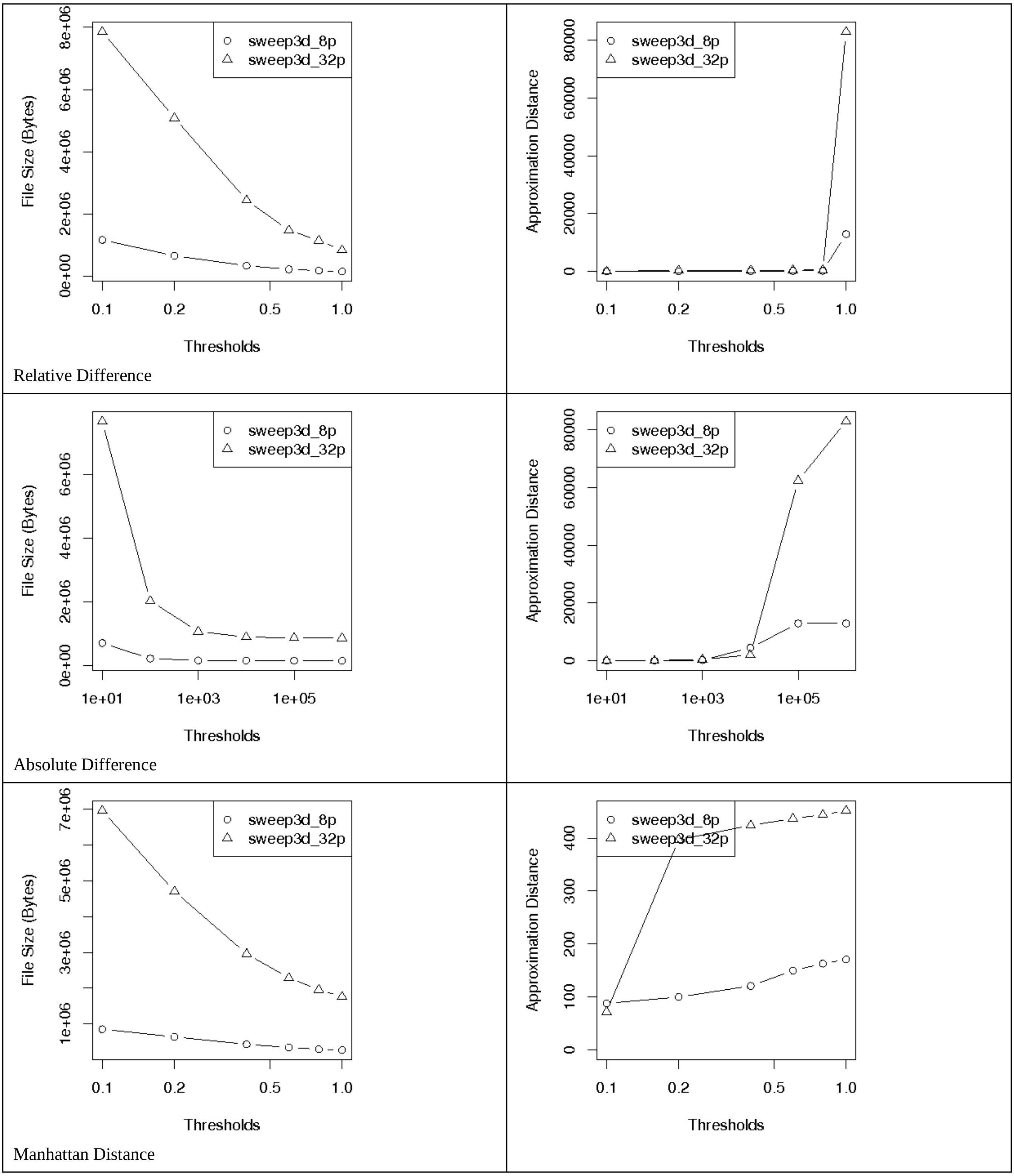

Figure 17: File Size and Approximation Distance for Varying Thresholds for Sweep3d and relDiff, absDiff, Manhattan

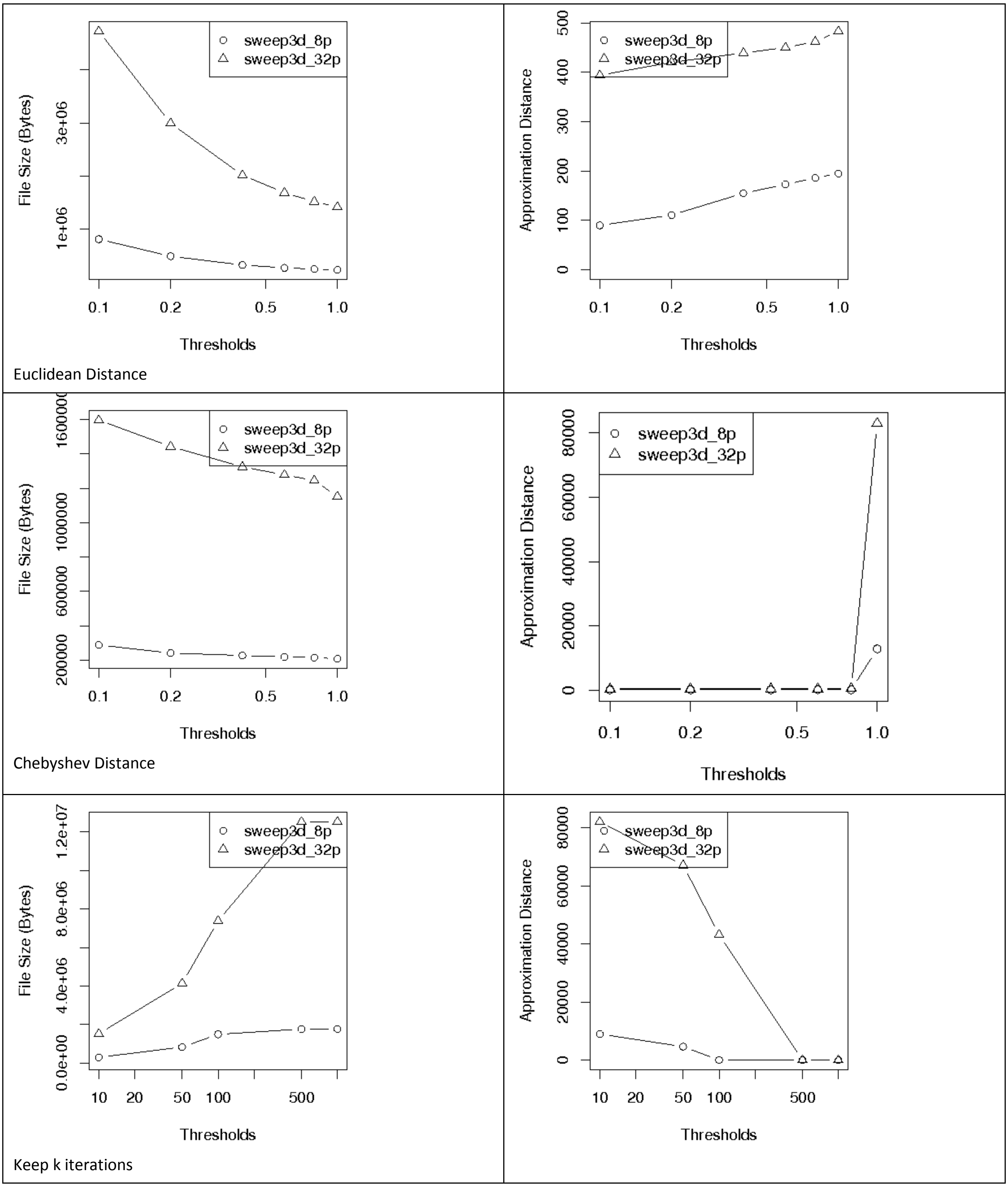

Figure 18: File Size and Approximation Distance for Varying Thresholds for Sweep3d and Euclidean, Chebyshev, iter_k

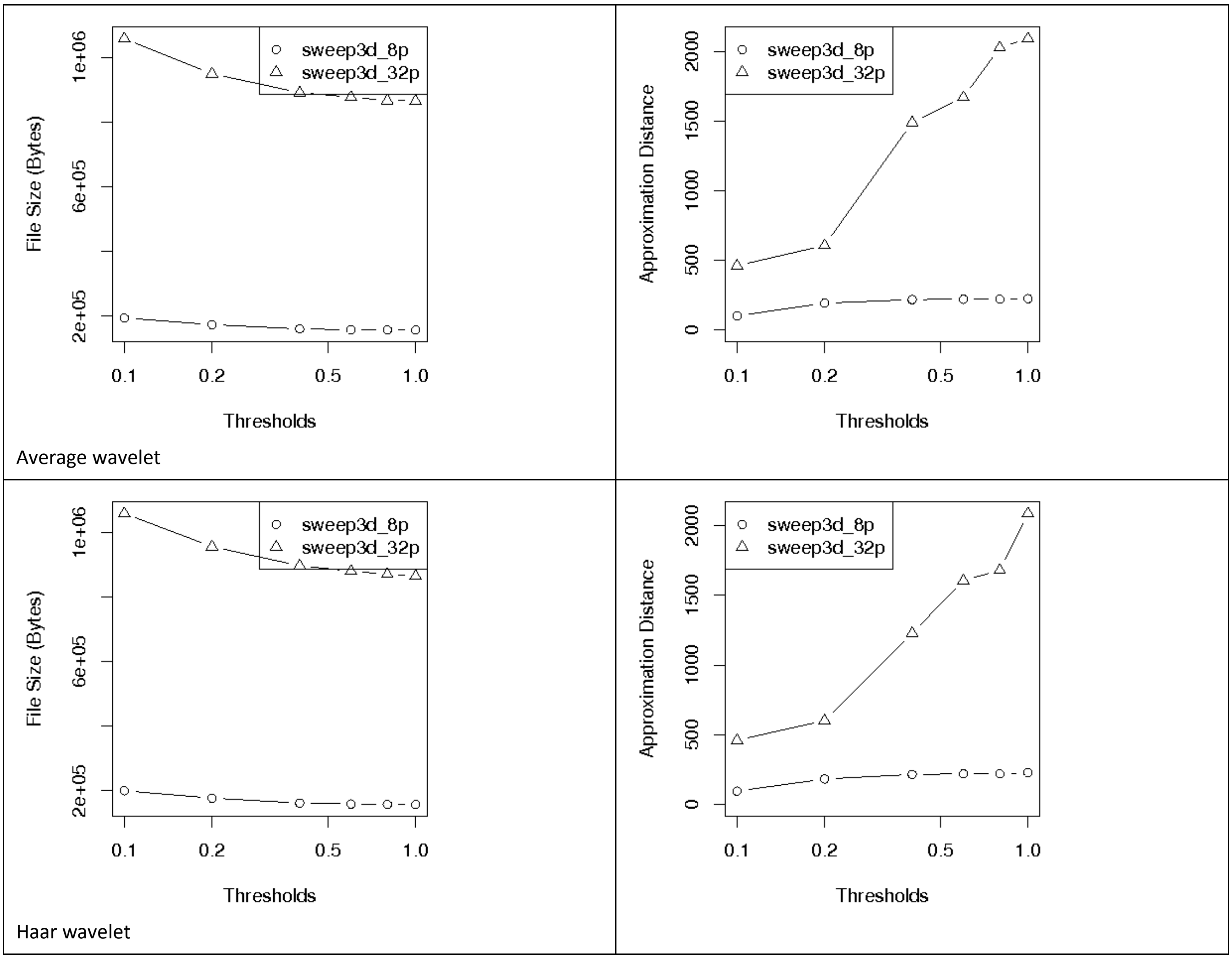

Figure 19: File Size and Approximation Distance for Varying Thresholds for Sweep3d and Wavelet Transforms

Table 1: Retention of Performance Trends with Varying Thresholds for dyn_load_balance

		MPI_Alltoall					do_work
	no loss	EX	MP	CM	CO	NN	EX
relative difference	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
absolute difference	10	EX	MP	CM	CO	NN	EX
	100	EX	MP	CM	CO	NN	EX
	1000	EX	MP	CM	CO	NN	EX
	10000	EX	MP	CM	CO	NN	EX
	100000	EX	MP	CM	CO	NN	EX
	1000000	EX	MP	CM	CO	NN	EX
Manhattan distance	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
Euclidean distance	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
Chebyshev distance	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
Average Wavelet	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
Haar Wavelet	0.1	EX	MP	CM	CO	NN	EX
	0.2	EX	MP	CM	CO	NN	EX
	0.4	EX	MP	CM	CO	NN	EX
	0.6	EX	MP	CM	CO	NN	EX
	0.8	EX	MP	CM	CO	NN	EX
	1.0	EX	MP	CM	CO	NN	EX
Keep k iterations	500	EX	MP	CM	CO	NN	EX
	100	EX	MP	CM	CO	NN	EX
	50	EX	MP	CM	CO	NN	EX
	10	EX	MP	CM	CO	NN	EX
	1	EX	MP	CM	CO	NN	EX
	average	EX	MP	CM	CO	NN	EX

Table 2: Retention of Performance Trends with Varying Thresholds for early_gather

		MPI_Gather					do_work
	no loss	EX	MP	CM	CO	ER	EX
relative difference	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
absolute difference	10	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	1000	EX	MP	CM	CO	ER	EX
	10000	EX	MP	CM	CO	ER	EX
	100000	EX	MP	CM	CO	ER	EX
	1000000	EX	MP	CM	CO	ER	EX
Manhattan distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Euclidean distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Chebyshev distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Average Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Haar Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Keep k iterations	500	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	50	EX	MP	CM	CO	ER	EX
	10	EX	MP	CM	CO	ER	EX
	1	EX	MP	CM	CO	ER	EX
	average	EX	MP	CM	CO	ER	EX

Table 3: Retention of Performance Trends with Varying Threshold for imbalance_at_mpi_barrier

		MPI_Barrier					do_work
	no loss	EX	MP	SN	BA	WB	EX
relative difference	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
absolute difference	10	EX	MP	SN	BA	WB	EX
	100	EX	MP	SN	BA	WB	EX
	1000	EX	MP	SN	BA	WB	EX
	10000	EX	MP	SN	BA	WB	EX
	100000	EX	MP	SN	BA	WB	EX
	1000000	EX	MP	SN	BA	WB	EX
Manhattan distance	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
Euclidean distance	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
Chebyshev distance	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
Average Wavelet	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
Haar Wavelet	0.1	EX	MP	SN	BA	WB	EX
	0.2	EX	MP	SN	BA	WB	EX
	0.4	EX	MP	SN	BA	WB	EX
	0.6	EX	MP	SN	BA	WB	EX
	0.8	EX	MP	SN	BA	WB	EX
	1.0	EX	MP	SN	BA	WB	EX
Keep k iterations	500	EX	MP	SN	BA	WB	EX
	100	EX	MP	SN	BA	WB	EX
	50	EX	MP	SN	BA	WB	EX
	10	EX	MP	SN	BA	WB	EX
	1	EX	MP	SN	BA	WB	EX
	average	EX	MP	SN	BA	WB	EX

Table 4: Retention of Performance Trends with Varying Threshold for late_broadcast

		MPI_Bcast					do_work
	no loss	EX	MP	CM	CO	LB	EX
relative difference	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
absolute difference	10	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	1000	EX	MP	CM	CO	LB	EX
	10000	EX	MP	CM	CO	LB	EX
	100000	EX	MP	CM	CO	LB	EX
	1000000	EX	MP	CM	CO	LB	EX
Manhattan distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Euclidean distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Chebyshev distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Average Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Haar Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Keep k iterations	500	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	50	EX	MP	CM	CO	LB	EX
	10	EX	MP	CM	CO	LB	EX
	1	EX	MP	CM	CO	LB	EX
	average	EX	MP	CM	CO	LB	EX

Table 5: Retention and Performance Trends with Varying Thresholds for late_receiver

		MPI_Ssend					do_work
	no loss	EX	MP	CM	P2	LR	EX
relative difference	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
absolute difference	10	EX	MP	CM	P2	LR	EX
	100	EX	MP	CM	P2	LR	EX
	1000	EX	MP	CM	P2	LR	EX
	10000	EX	MP	CM	P2	LR	EX
	100000	EX	MP	CM	P2	LR	EX
	1000000	EX	MP	CM	P2	LR	EX
Manhattan distance	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
Euclidean distance	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
Average Wavelet	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LR	EX
	0.2	EX	MP	CM	P2	LR	EX
	0.4	EX	MP	CM	P2	LR	EX
	0.6	EX	MP	CM	P2	LR	EX
	0.8	EX	MP	CM	P2	LR	EX
	1.0	EX	MP	CM	P2	LR	EX
Keep k iterations	500	EX	MP	CM	P2	LR	EX
	100	EX	MP	CM	P2	LR	EX
	50	EX	MP	CM	P2	LR	EX
	10	EX	MP	CM	P2	LR	EX
	1	EX	MP	CM	P2	LR	EX
	average	EX	MP	CM	P2	LR	EX

Table 6: Retention of Performance Trends with Varying Thresholds for late_sender

		MPI_Recv					do_work
	no loss	EX	MP	CM	P2	LS	EX
relative difference	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
absolute difference	10	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LS	EX
	1000	EX	MP	CM	P2	LS	EX
	10000	EX	MP	CM	P2	LS	EX
	100000	EX	MP	CM	P2	LS	EX
	1000000	EX	MP	CM	P2	LS	EX
Manhattan distance	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
Euclidean distance	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
Average Wavelet	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LS	EX
Keep k iterations	500	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LS	EX
	50	EX	MP	CM	P2	LS	EX
	10	EX	MP	CM	P2	LS	EX
	1	EX	MP	CM	P2	LS	EX
	average	EX	MP	CM	P2	LS	EX

Table 7: Retention of Performance Trends with Varying Thresholds for Nto1_32

		MPI_Gather					do_work
	no loss	EX	MP	CM	CO	ER	EX
relative difference	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
absolute difference	10	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	1000	EX	MP	CM	CO	ER	EX
	10000	EX	MP	CM	CO	ER	EX
	100000	EX	MP	CM	CO	ER	EX
	1000000	EX	MP	CM	CO	ER	EX
Manhattan distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Euclidean distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Chebyshev distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Average Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Haar Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Keep k iterations	500	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	50	EX	MP	CM	CO	ER	EX
	10	EX	MP	CM	CO	ER	EX
	1	EX	MP	CM	CO	ER	EX
	average	EX	MP	CM	CO	ER	EX

Table 8: Retention of Performance Trends with Varying Thresholds for NtoN_32

		MPI_Barrier						do_work
	no loss	EX	MP	SN	BA	WB	BC	EX
relative difference	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
absolute difference	10	EX	MP	SN	BA	WB	BC	EX
	100	EX	MP	SN	BA	WB	BC	EX
	1000	EX	MP	SN	BA	WB	BC	EX
	10000	EX	MP	SN	BA	WB	BC	EX
	100000	EX	MP	SN	BA	WB	BC	EX
	1000000	EX	MP	SN	BA	WB	BC	EX
Manhattan distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Euclidean distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Chebyshev distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Average Wavelet	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Haar Wavelet	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Keep k iterations	500	EX	MP	SN	BA	WB	BC	EX
	100	EX	MP	SN	BA	WB	BC	EX
	50	EX	MP	SN	BA	WB	BC	EX
	10	EX	MP	SN	BA	WB	BC	EX
	1	EX	MP	SN	BA	WB	BC	EX
	average	EX	MP	SN	BA	WB	BC	EX

Table 9: Retention of Performance Trends with Varying Thresholds for 1toN_32

		MPI_Bcast					do_work
	no loss	EX	MP	CM	CO	LB	EX
relative difference	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
absolute difference	10	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	1000	EX	MP	CM	CO	LB	EX
	10000	EX	MP	CM	CO	LB	EX
	100000	EX	MP	CM	CO	LB	EX
	1000000	EX	MP	CM	CO	LB	EX
Manhattan distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Euclidean distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Chebyshev distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Average Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Haar Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Keep k iterations	500	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	50	EX	MP	CM	CO	LB	EX
	10	EX	MP	CM	CO	LB	EX
	1	EX	MP	CM	CO	LB	EX
	average	EX	MP	CM	CO	LB	EX

Table 10: Retention of Performance Trends with Varying Thresholds for 1to1r_32

		MPI_Ssend					MPI_Recv					do_work
	no loss	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Relative Difference	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Absolute Difference	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Manhattan distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Euclidean distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Average Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Keep k iterations	500	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	50	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	average	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX

Table 11: Retention of Performance Trends with Varying Thresholds for 1to1s_32

		MPI_Ssend					MPI_Recv					do_work
	no loss	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Relative Difference	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Absolute Difference	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Manhattan distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Euclidean distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Average Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Keep k iterations	500	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	50	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	average	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX

Table 12: Retention of Performance Trends with Varying Thresholds for Nto1_1024

		MPI_Gather					do_work
	no loss	EX	MP	CM	CO	ER	EX
relative difference	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
absolute difference	10	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	1000	EX	MP	CM	CO	ER	EX
	10000	EX	MP	CM	CO	ER	EX
	100000	EX	MP	CM	CO	ER	EX
	1000000	EX	MP	CM	CO	ER	EX
Manhattan distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Euclidean distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Chebyshev distance	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Average Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Haar Wavelet	0.1	EX	MP	CM	CO	ER	EX
	0.2	EX	MP	CM	CO	ER	EX
	0.4	EX	MP	CM	CO	ER	EX
	0.6	EX	MP	CM	CO	ER	EX
	0.8	EX	MP	CM	CO	ER	EX
	1.0	EX	MP	CM	CO	ER	EX
Keep k iterations	500	EX	MP	CM	CO	ER	EX
	100	EX	MP	CM	CO	ER	EX
	50	EX	MP	CM	CO	ER	EX
	10	EX	MP	CM	CO	ER	EX
	1	EX	MP	CM	CO	ER	EX
	average	EX	MP	CM	CO	ER	EX

Table 13: Retention of Performance Trends with Varying Thresholds for NtoN_1024

		MPI_Barrier						do_work
	no loss	EX	MP	SN	BA	WB	BC	EX
relative difference	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
absolute difference	10	EX	MP	SN	BA	WB	BC	EX
	100	EX	MP	SN	BA	WB	BC	EX
	1000	EX	MP	SN	BA	WB	BC	EX
	10000	EX	MP	SN	BA	WB	BC	EX
	100000	EX	MP	SN	BA	WB	BC	EX
	1000000	EX	MP	SN	BA	WB	BC	EX
Manhattan distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Euclidean distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Chebyshev distance	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Average Wavelet	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Haar Wavelet	0.1	EX	MP	SN	BA	WB	BC	EX
	0.2	EX	MP	SN	BA	WB	BC	EX
	0.4	EX	MP	SN	BA	WB	BC	EX
	0.6	EX	MP	SN	BA	WB	BC	EX
	0.8	EX	MP	SN	BA	WB	BC	EX
	1.0	EX	MP	SN	BA	WB	BC	EX
Keep k iterations	500	EX	MP	SN	BA	WB	BC	EX
	100	EX	MP	SN	BA	WB	BC	EX
	50	EX	MP	SN	BA	WB	BC	EX
	10	EX	MP	SN	BA	WB	BC	EX
	1	EX	MP	SN	BA	WB	BC	EX
	average	EX	MP	SN	BA	WB	BC	EX

Table 14: Retention of Performance Trends with Varying Thresholds for 1toN_1024

		MPI_Bcast					do_work
	no loss	EX	MP	CM	CO	LB	EX
relative difference	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
absolute difference	10	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	1000	EX	MP	CM	CO	LB	EX
	10000	EX	MP	CM	CO	LB	EX
	100000	EX	MP	CM	CO	LB	EX
	1000000	EX	MP	CM	CO	LB	EX
Manhattan distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Euclidean distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Chebyshev distance	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Average Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Haar Wavelet	0.1	EX	MP	CM	CO	LB	EX
	0.2	EX	MP	CM	CO	LB	EX
	0.4	EX	MP	CM	CO	LB	EX
	0.6	EX	MP	CM	CO	LB	EX
	0.8	EX	MP	CM	CO	LB	EX
	1.0	EX	MP	CM	CO	LB	EX
Keep k iterations	500	EX	MP	CM	CO	LB	EX
	100	EX	MP	CM	CO	LB	EX
	50	EX	MP	CM	CO	LB	EX
	10	EX	MP	CM	CO	LB	EX
	1	EX	MP	CM	CO	LB	EX
	average	EX	MP	CM	CO	LB	EX

Table 15: Retention of Performance Trends with Varying Thresholds for 1to1r_1024

		MPI_Ssend					MPI_Recv					do_work
	no loss	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Relative Difference	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Absolute Difference	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Manhattan distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Euclidean distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Average Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Keep k iterations	500	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	50	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	average	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX

Table 16: Retention of Performance Trends with Varying Thresholds for 1to1s_1024

		MPI_Ssend					MPI_Recv					do_work
	no loss	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Relative Difference	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Absolute Difference	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1000000	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Manhattan distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Euclidean distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Average Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.2	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.4	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.6	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	0.8	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1.0	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
Keep k iterations	500	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	100	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	50	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	10	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	1	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX
	average	EX	MP	CM	P2	LR	EX	MP	CM	P2	LS	EX

Table 17: Retention of Performance Trends with Varying Thresholds for sweep3d_8p

		pmpi_recv_						sweep_
	no loss	EX	MP	CM	P2	LS	MO	EX
relative difference	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
absolute difference	10	EX	MP	CM	P2	LS	MO	EX
	100	EX	MP	CM	P2	LS	MO	EX
	1000	EX	MP	CM	P2	LS	MO	EX
	10000	EX	MP	CM	P2	LS	MO	EX
	100000	EX	MP	CM	P2	LS	MO	EX
	1000000	EX	MP	CM	P2	LS	MO	EX
Manhattan distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Euclidean distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Average Wavelet	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Keep k iterations	500	EX	MP	CM	P2	LS	MO	EX
	100	EX	MP	CM	P2	LS	MO	EX
	50	EX	MP	CM	P2	LS	MO	EX
	10	EX	MP	CM	P2	LS	MO	EX
	1	EX	MP	CM	P2	LS	MO	EX
	average	EX	MP	CM	P2	LS	MO	EX

Table 18: Retention of Performance Trends with Varying Thresholds for sweep3d_32p

		pmpi_recv_						sweep_
	no loss	EX	MP	CM	P2	LS	MO	EX
relative difference	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
absolute difference	10	EX	MP	CM	P2	LS	MO	EX
	100	EX	MP	CM	P2	LS	MO	EX
	1000	EX	MP	CM	P2	LS	MO	EX
	10000	EX	MP	CM	P2	LS	MO	EX
	100000	EX	MP	CM	P2	LS	MO	EX
	1000000	EX	MP	CM	P2	LS	MO	EX
Manhattan distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Euclidean distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Chebyshev distance	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Average Wavelet	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Haar Wavelet	0.1	EX	MP	CM	P2	LS	MO	EX
	0.2	EX	MP	CM	P2	LS	MO	EX
	0.4	EX	MP	CM	P2	LS	MO	EX
	0.6	EX	MP	CM	P2	LS	MO	EX
	0.8	EX	MP	CM	P2	LS	MO	EX
	1.0	EX	MP	CM	P2	LS	MO	EX
Keep k iterations	500	EX	MP	CM	P2	LS	MO	EX
	100	EX	MP	CM	P2	LS	MO	EX
	50	EX	MP	CM	P2	LS	MO	EX
	10	EX	MP	CM	P2	LS	MO	EX
	1	EX	MP	CM	P2	LS	MO	EX
	average	EX	MP	CM	P2	LS	MO	EX