World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:12, No:3, 2018

Performance Evaluation of Parallel Surface Modeling and Generation on Actual and Virtual Multicore Systems

Nyeng P. Gyang

***Abstract*—**Even though past, current and future trends suggest that multicore and cloud computing systems are increasingly prevalent/ubiquitous, this class of parallel systems is nonetheless underutilized, in general, and barely used for research on employing parallel Delaunay triangulation for parallel surface modeling and generation, in particular. The performances, of actual/physical and virtual/cloud multicore systems/machines, at executing various algorithms, which implement various parallelization strategies of the incremental insertion technique of the Delaunay triangulation algorithm, were evaluated. *T*-tests were run on the data collected, in order to determine whether various performance metrics differences (including execution time, speedup and efficiency) were statistically significant. Results show that the actual machine is approximately twice faster than the virtual machine at executing the same programs for the various parallelization strategies. Results, which furnish the scalability behaviors of the various parallelization strategies, also show that some of the differences between the performances of these systems, during different runs of the algorithms on the systems, were statistically significant. A few pseudo superlinear speedup results, which were computed from the raw data collected, are not true superlinear speedup values. These pseudo superlinear speedup values, which arise as a result of one way of computing speedups, disappear and give way to asymmetric speedups, which are the accurate kind of speedups that occur in the experiments performed.

***Keywords*—**Cloud computing systems, multicore systems, parallel delaunay triangulation, parallel surface modeling and generation.

I. Introduction

THE emergence as well as the adoption and use of multicore processors (and, therefore, the systems these processors power) is a trend that has been ongoing for over a decade or so. There has also been the trend of the adoption and use of cloud computing systems. Processor designers have discovered that, by using the technique of clock speed increases, they have reached the limits of processor speed improvements. In order to increase processor speeds today, designers are resorting to placing multiple cores on the same processor chip. This shift towards producing multicore processors is responsible for parallel computing becoming increasingly ubiquitous, including both the access to and the applications for parallel computing becoming increasingly more widespread. In particular, this paradigm shift in processor design and manufacture is very largely and primarily responsible for the trend of the adoption/use of multicore systems, including multicore systems provisioned either physically or virtually (via cloud computing technology). These trends suggest and promise the future dominance of these kinds of system. In fact, [10] suggests that manycore processors will quickly extend and replace multicore ones, with the former processor type consisting of tens to hundreds of cores per processor chip.

The quest for alternative means of accomplishing processor speed increase and the derivable benefits of adopting and using cloud computing systems furnish the motivation for the trend towards the adoption and use of, respectively, multicore and cloud computing systems. One envisages that the emergence and future dominance of multicore and manycore processors will usher the birth of the era of the vast prevalence, ubiquity or preponderance of parallel computing (including parallel algorithms development, parallel programming, parallel processing systems development, etc.) and the era of "desktop supercomputing." Indeed, desktop supercomputing may be seen in the horizon already. It is easy to both see and underscore the aforementioned forecast because consumer computing devices, ranging from personal computers to smart phones, already come with and are powered by multicore processors today. Furthermore, cloud computing technology will facilitate the adoption and use of multicore systems, by enabling the provisioning of these systems more cheaply, more conveniently and virtually.

Cloud computing systems are also demonstrating that users who adopt and use these systems derive benefits, ranging from reduced Total Cost of Ownership (TCO) to more cheaply and conveniently orchestrated scalability. Just like with systems with the multicore architecture, systems with the Services Oriented Architecture (SOA) are also increasingly being adopted and used for a wide range of computing applications. The SOA is the basic architecture of cloud computing systems. The SOA comes in various "flavors" or service models, including Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Desktop as a Service (DaaS), and Disaster Recovery as a Service (DRaaS), etc. Various kinds of parallel computing systems (including multicore systems) are provisioned virtually, using cloud computing technology. Some may opine that the publicity and promotion accorded cloud computing, in recent times, is merely a hype and forecast that the following trend or development with cloud computing will decline, rather than increase, in the future: The adoption and use of what these

N. P. Gyang is with the Mathematical and Computer Sciences Department of the Metropolitan State University of Denver, USA (phone: 7192329020; e-mail: yeipeng@gmail.com).

forecasters construe as an overestimated, overhyped computing model, which is merely associated with a lot of hysteria and publicity. Despite this opinion and forecast that some may hold and make, cloud computing nevertheless furnishes a computing paradigm and architecture, which also promises to dominate future computing systems, architectures and platforms.

The goal of employing multicore systems, especially those provisioned virtually using cloud computing technology, for parallel surface modeling and generation is an auspicious and worthy one. This is especially true considering the findings, while conducting this study, that there is a dearth of research on the implementation and performance evaluation of parallel Delaunay triangulation for parallel surface modeling and generation, using multicore or cloud computing systems. Consequently, this research evaluates the performances, of actual/physical and virtual/cloud multicore systems/machines, at executing various algorithms that implement various parallelization strategies of the incremental insertion technique of the Delaunay triangulation algorithm.

Delaunay triangulation is itself a domain discretization or meshing algorithm with a variety of applications; application areas include (1) **Computer graphics:** Surface modeling, volume rendering (2) **Scientific visualization and interpolation in mathematical and natural sciences:** Mesh generation for Partial Differential Equation (PDE) solution techniques, such as the Finite Element Method (FEM) analysis, which is employed in applications such as Computational Fluid Mechanics (CFD), Computational Solid Mechanics (CSM), Computational Electromagnetics (CE), etc. (3) **Robotics:** Computer vision and image synthesis, pattern recognition, etc. and (4) **Structural networking for arbitrary point sets**.

This article is organized as follows: Section II furnishes a literature review on parallel Delaunay triangulation research; Section III presents a randomized incremental insertion algorithm (which is a technique of the Delaunay triangulation algorithm) and its evaluation methodology; Section IV furnishes a discussion on experimental results; Section V is a conclusion and Section VI furnishes recommendations for future work.

II. Literature Review for Research on Systems for Parallel Delaunay Triangulation

In the literature for research on the implementation and performance evaluation of parallel Delaunay triangulation for parallel surface modeling and generation, using multicore or cloud computing systems, very little work has been conducted to investigate the utilization of these systems for this purpose. These kinds of systems are actually very notably barely utilized for research on parallel Delaunay triangulation for parallel surface modeling and generation. In the literature, other kinds of parallel systems are typically employed for research on parallel Delaunay triangulation, including Connection Machines [17], [19], cluster (distributed) computing systems [16], [6], [4], [5], multiprocessor systems [11]-[13] and multicore systems [14], [15].

In the discussion in the literature, five different categories of the Delaunay triangulation algorithm can be identified. These categories of the Delaunay triangulation algorithm include the following: Divide-and-conquer, sweepline, incremental (including incremental construction and incremental insertion), gift-wrapping and convex-hull-based. These five methods are those for the direct Delaunay triangulation construction; there is also an indirect Delaunay triangulation construction technique, which starts with constructing the Voronoi diagram dual of any particular Delaunay triangulation and afterwards the Delaunay triangulation is constructed from the Voronoi diagram.

Puppo et al. [17], which is a seminal article on parallel Delaunay triangulation, present a range of topics on the subject of parallel terrain modeling. Research efforts on parallel Delaunay triangulation are on the design, implementation and/or evaluation of the Delaunay triangulation [1], [19], [9], on the investigation of properties, features and/or application areas of the Delaunay triangulation in general [14]-[17], [7], [2]-[5], [6] and on a widely known as well as widely and practically used Delaunay triangulation implementation work [18]. Furthermore, [8], which is a research effort that this article transmits, discusses and analyzes surface modeling and generation as well as parallel Delaunay triangulation.

In the literature, schemes and algorithms for parallel Delaunay triangulation are primarily and heavily de-pendent on a couple of things, namely: (1) The category or type of Delaunay triangulation algorithm employed and (2) The target parallel system type on which an algorithm was either implemented or intended to be implemented. The various types or categories of the parallel Delaunay triangulation algorithm that are typically discussed, in the literature, are the aforementioned five types while the various types of system targeted include shared-memory systems (including multiprocessor systems) and distributed computing, message-passing-based systems.

III. A Randomized Incremental Insertion Algorithm and Its Evaluation Methodology

Kolingerová and Kohout [13], [11], [12] discuss research on various parallelization strategies of, or approaches to, a basic parallel type of Delaunay triangulation algorithm; these three articles provide information on the design and implementation of this algorithm. Each of these different parallelization strategies or approaches – the various strategies include the batch, burglary, optimistic, optimistic (circle), optimistic (prev) and pessimistic strategies – represents a different approach/strategy to accomplishing synchronization for this basic parallel algorithm. The basic algorithm, which is referred to as a randomized incremental insertion algorithm, is an incremental insertion type of the Delaunay triangulation algorithm. This article presents research, which studies and extends these works.

The authors of the randomized incremental insertion algorithm used the various implemented parallelization strategies of, or approaches to, this algorithm to process both

artificially generated and real-life data sets. The artificially generated data sets are of various distributions, including the following: (A) Grid data set, (B) Uniform data set, (C) Gauss data set, (D) Cluster data set, (E) Arc data set, and (F) Sphere data set. The real-life data sets were obtained from the following: (A) A crater lake, which contains 100,001 points and (B) A whale, which contains 52,635 points. In the various experiments conducted in this research, in order to evaluate the various aforementioned parallelization strategies, algorithms for these strategies processed only the 100,001 points of the crater lake (in other words, the algorithms evaluated in this research did not process any of the aforementioned artificially generated data sets or the real-life data set obtained from the whale).

A. Operationalization of Variables and Research Design

The table of Fig. 1 furnishes information on the top-level research design for the study presented in this article. The various variables, on which data will be collected (as shown in Fig. 1), are determined from the following:

1.) Sequential algorithm execution times, obtained according to the following:
a) Whether the system/machine is virtual or actual.
b) Period of day – i.e., morning, afternoon and evening.
2.) Parallel algorithm execution times, obtained according to the following:
a) Whether the machine/system is virtual or actual.
b) Number of threads/cores used – i.e., 1, 2, 4, 6 and 8 threads/cores.
c) Parallelization strategy used – i.e., batch, pessimistic, optimistic, optimistic (circle), optimistic (prev) and burglary.
d) Period of day – i.e., morning, afternoon and evening.

After collecting data on the variables described above, other variables are subsequently computed; these other variables, which are parallel processing speedups and efficiencies, are computed from the values obtained/measured for (both sequential and parallel) algorithm execution times and number of processors used to achieve respective/corresponding speedups. The variables described above, plus the setup described in Fig. 1, together furnish the experimental conditions for performing various experiments with the algorithms.

Parallelization Strategies for the Randomized Incremental Insertion Algorithm	Cloud (virtual) Multicore System	Physical (actual) Multicore System
Batch		
Burglary	Data Collection on Various Variables	
Optimistic		
Optimistic (circle)		
Optimistic (prev)		
Pessimistic		

Fig. 1 Top-level Research Design

As a result of the fact that a repeated-measures experiment design is employed to address various hypotheses for this research, the dependent-means *t*-test is employed in the data analysis phase of the research. Fig. 2 is a generic table of variables, which was created for the *t*-test performed for each of *N* threads/cores (where *N* = 2, 4, 6 and 8); this means that, for each of the actual and virtual machines, four of such tables were created (each table was stored in a separate data analysis software file). Fig. 2 expatiates on Fig. 1 (note, however, that the table of Fig. 1 combines variables for both actual and virtual machines, while each of the tables illustrated in Fig. 2 is either for an actual or for a virtual machine – i.e., there are four Fig. 2 tables for the actual machine and four Fig. 2 tables for the virtual machine).

Figs. 2 and 3 show the variables used in the *t*-tests performed in this study. While the table of Fig. 2 shows six categorical variables and nine continuous variables, for each of *N* threads/cores, the table of Fig. 3 shows the same six categorical variables and six continuous variables, for each of *N* threads/cores. All the *t*-tests performed in this study were performed using these six categorical variables and 9 + 6 = 15 continuous variables. (The six categorical variables are the six algorithm types. The first set of nine continuous variables consists of the three performance metrics, with a value being expected or obtainable for each of these three performance metrics during three different times of day – i.e., morning, afternoon and evening. The other set of six continuous variables consists of the three performance metrics, with a value being expected or obtainable for each of these three performance metrics for both the actual and virtual machines.)

There are four other categorical variables in this study, namely: four different *N* threads/cores (where *N* = 2, 4, 6 and 8). For the *t*-tests performed in this study, the categorical variables are the independent variables, while the continuous variables are the dependent variables.

Parallelization Strategies for the Randomized Incremental Insertion Algorithm	V1	V2	V3	V4	V5	V6	V7	V8	V9
Batch									
Burglary									
Optimistic									
Optimistic (circle)									
Optimistic (prev)									
Pessimistic									

Fig. 2 Generic Table of Variables for the *t*-tests Performed for each *N* Threads/Cores (nine continuous variables)

Legend for Fig. 2:
V1 ≡ Variable 1 ≡ Morning_execution_time
V2 ≡ Variable 2 ≡ Afternoon_execution_time
V3 ≡ Variable 3 ≡ Evening_execution_time
V4 ≡ Variable 4 ≡ Morning_speedup
V5 ≡ Variable 5 ≡ Afternoon_speedup
V6 ≡ Variable 6 ≡ Evening_speedup
V7 ≡ Variable 7 ≡ Morning_efficiency
V8 ≡ Variable 8 ≡ Afternoon_efficiency
V9 ≡ Variable 9 ≡ Evening_efficiency

Legend for Fig. 3:
V1 ≡ Variable 1 ≡ Actual_execution_time
V2 ≡ Variable 2 ≡ Virtual_execution_time
V3 ≡ Variable 3 ≡ Actual_speedup
V4 ≡ Variable 4 ≡ Virtual_speedup

V5 ≡ Variable 5 ≡ Actual_efficiency
V6 ≡ Variable 6 ≡ Virtual_efficiency

Parallelization Strategies for the Randomized Incremental Insertion Algorithm	V1	V2	V3	V4	V5	V6
Batch						
Burglary						
Optimistic						
Optimistic (circle)						
Optimistic (prev)						
Pessimistic						

Fig. 3 Generic Table of Variables for the *t*-tests Performed for each *N* Threads/Cores (six continuous variables)

The repeated-measures experiment design, which has been adopted for this research, is presented as follows:

1) All the six parallelization strategies were run on the following two kinds of system, which have comparable platforms, including comparable system and processor specifications: A physical multicore system and a cloud multicore system. (The system and processor specifications of both kinds of system are furnished in Appendix I.)
2) All the six parallelization strategies were run, on an actual multicore system as well as a virtual multicore system, during three different times of the day, namely: Morning, afternoon and evening.

In both repeated-measures experimental designs/setups above, the same participating entities – i.e., the various parallelization strategies – were subjected or exposed to different experimental conditions/situations.

In the foregoing experiment design, the following control measure was employed to guide the process of data collection: As a result of the fact that this research entails an evaluation and a comparison of the processing performances of systems that process algorithms for the Delaunay triangulation, there is the question of whether or not the following machines or systems with the very same hardware and platform specifications, as well as processing the same benchmark, demonstrate the same parallel processing performance:

1) Physical multicore system.
2) Virtual multicore system, which is provided via the cloud.

To ensure that "apples are compared with apples," the following control measure was adopted: Sequential vs. parallel processing performances were compared only when both of these sequential and parallel processing performances were obtained from the same kind of system – i.e. performances on a physical, actual sequential system were compared only with performances on a physical, actual parallel system and performances on a virtual sequential system were compared only with performances on a virtual parallel system (with the virtual system being provided via the cloud).

The research design for this study also includes establishing the metrics for evaluating and comparing the performances of various algorithms and systems as well as the guidelines for planning/arranging how to go about – i.e. planning/arranging the procedure for – conducting experiments as well as performing data collection and data analysis.

B. Establishing the Metrics for Comparison and Evaluation of Algorithms

The basic performance metric is the running/execution time of various algorithms (including both serial/sequential and parallel). This basic performance metric plus other performance metrics (which are computed from execution times) were used to evaluate and compare the performance of the various algorithms, on both kinds of multicore machines, at processing a benchmark. The other performance metrics (in addition to execution time) include speedups and efficiencies of the algorithms. These three performance metrics comprise a framework for the evaluation and comparison of algorithms.

C. Data Collection

Data were collected on the performance of the algorithms and systems. The following couple of points furnish information on the methodology for conducting data collection:

1) Data were collected/obtained, for the algorithms on actual and virtual machines, as follows:
a) Data on the running times of the sequential and parallel algorithms were obtained.
b) The speedups of parallel algorithms over sequential algorithms were determined.
c) The efficiencies of parallel algorithms were determined.
2) Data were collected/obtained from both repeated-measures experiment designs/setups (which were mentioned in Section III.A).

D. Data Analysis

The various performance metrics, which were either obtained or determined/computed as mentioned in Section III.C, were used in conducting the data analysis for this study. This analysis entailed evaluating and comparing the performances or behaviors of algorithms representing the various parallelization strategies. The information, which is furnished by this analysis, includes the performances or behaviors of these algorithms, in terms of execution times, speedups and efficiencies.

The dependent-means *t*-test is employed to test for the statistical significance of the following differences (more details on these differences are furnished in [8]):

1) The differences between performance data obtained on (A) A physical multicore system and (B) A cloud multicore system.
2) The differences between performance data obtained, on an actual multicore system as well as a virtual multicore system, during (A) The morning run and the afternoon run (B) The morning run and the evening run and (C) The afternoon run and the evening run.

Before applying the dependent-means parametric *t*-test to the research data, the normality of the data was tested using the ***Kolmogorov-Smirnov (K-S) test***. An approach adopted in conducting the *t*-test analysis for this research is to (A) Use the (parametric) *t*-test when the dataset does not violate the assumption of normality and (B) Use a non-parametric version or equivalent of the *t*-test when the dataset violates the

assumption of normality (the tests for normality, which were performed on the research data, showed that some datasets are not normal). The non-parametric version or equivalent of the *t*-test that was used (when a dataset violates the assumption of normality) is the ***Wilcoxon signed-rank test***. All *t*-tests and tests for normality were performed using ***IBM® SPSS® Statistics, Version 22 (Release 22.0.0.0, 64-bit edition)***.

IV. Discussion on Experimental Results

The data obtained from this study enables a dual and simultaneous couple of evaluations, as follows: (1) The data may be used to evaluate the performance of the cloud and physical parallel systems used in the study and (2) The data may be used to evaluate the performance of the various algorithms studied. While this dual analysis is conducted in this article, [8] furnishes a broader set of results as well as analysis and discussion, of the study and results furnished in this article, than the article itself does. The discussion on experimental results is categorized into the following major aspects: (1) Speedups and efficiencies characteristics, (2) Summary of *t*-tests performed, (3) Performance of actual machine vs. virtual machine in terms of execution time, (4) Performance of actual machine vs. virtual machine in terms of speedup and efficiency, (5) Performances of various algorithms on both machines during different runs, and (6) Asymmetric speedup performance for the batch strategy parallel program.

A. Speedups and Efficiencies Characteristics

The experimental results, which are given in Figs. 3-7, are not unlike those typically obtained from running parallel programs; i.e., (A) Speedups generally increase as the number of threads/cores increases, (B) Speedups generally reach a peak value, as the number of threads/cores approach, reach and exceed some value, and (C) Efficiencies generally decrease as the number of threads/cores increases. These kinds of result in the foregoing points numbers B and C are demonstrated to a greater extent for a parallel processing system that is characterized by poorer scalability. In other words, for non-superlinear scalability, the better the scalability of a parallel processing system, the more linear the graph of the system's speedup performance will be as well as the more constant the graph of the system's efficiency performance will be (non-superlinear scalability is meant here to include both linear scalability and sublinear scalability). Experimental results also show that speedups are generally greater than half of the number of threads/cores employed and parallel processing efficiency values are generally greater than 50%.

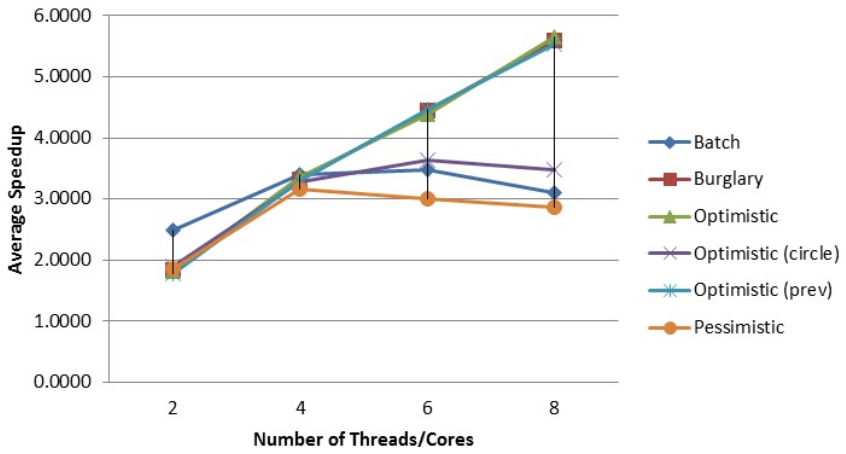

Fig. 4 Average Speedups for Actual Machine

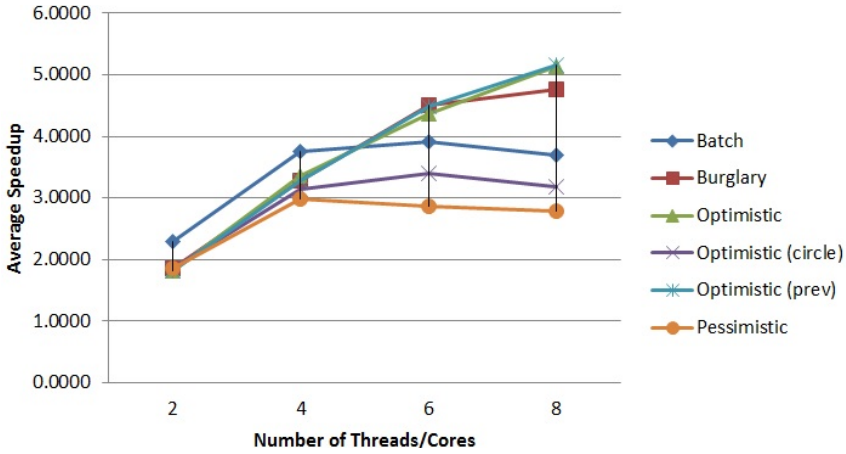

Fig. 5 Average Speedups for Virtual Machine

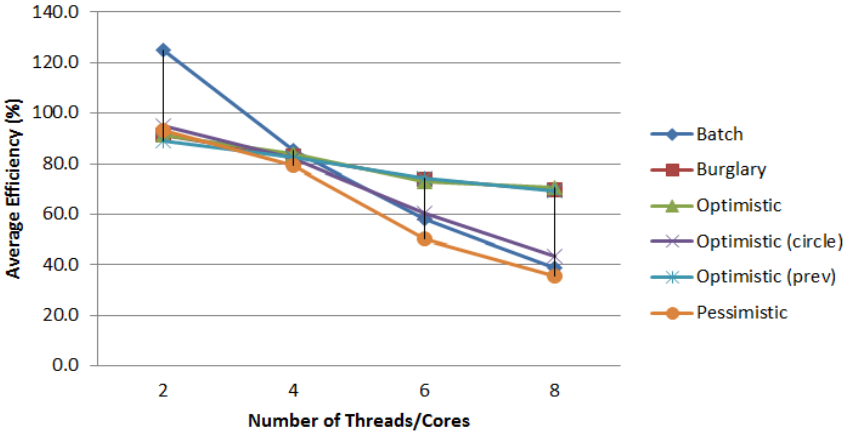

Fig. 6 Average Efficiencies for Actual Machine

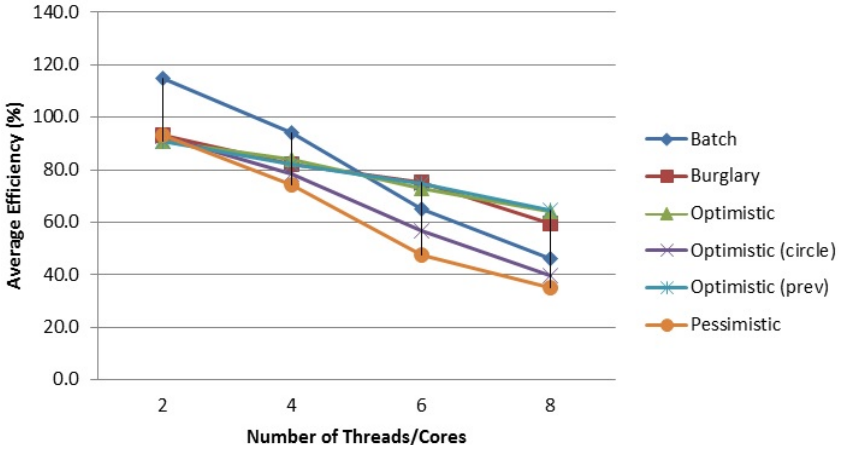

Fig. 7 Average Efficiencies for Virtual Machine

B. Summary of t-Tests Performed

Gyang [8] presents detailed results of the *t*-test and Wilcoxon signed-rank test analyses performed; these results show the following, for the *t*-tests that were performed in the study: (A) For the six categorical variables and nine continuous variables, differences were taken between 36 pairs of variables, for each of the actual and virtual systems and (B) For the six categorical variables and six continuous variables, differences were taken between 12 pairs of variables, for both of the actual and virtual systems.

The aforementioned 36 pairs of variables (per machine type) are computed as follows: Differences are taken between nine pairs of the nine continuous variables for each of the 4 different numbers of threads/cores – i.e., each of 2, 4, 6 and 8 threads/cores has three performance metrics and, for each of these performance metrics, the metrics in three different metric pairs are compared (each pair is obtained from each of the morning-afternoon, morning-evening and afternoon-evening runs); this yields 9 × 4 = 36 variable pairs. The aforementioned 12 pairs of variables (for both machine types) are computed as follows: Differences are taken between three pairs of the six continuous variables for each of the four different numbers of threads/cores – i.e., each of 2, 4, 6 and 8 threads/cores have three performance metrics and, for each of these performance metrics, the metrics in one metric pair are compared (this pair is obtained from the two different types of machine, i.e., actual and virtual); this yields 3 × 4 = 12 variable pairs.

Table I furnishes the results obtained, from the *t*-test and Wilcoxon signed-rank test analyses performed, on the differences between (A) The 36 pairs of variables per machine type and (B) The 12 pairs of variables for both machine types. The following are a summary of the results (which are presented in Table I) for the 36 and the 12 variables:

- The following results were obtained, out of the 36 possible variable pairs for the actual machine:
  - Two variable pair differences are statistically significant with respect to the execution time metric.
  - Three variable pair differences are statistically significant with respect to the speedup metric.
  - Four variable pair differences are statistically significant with respect to the efficiency metric.
  - The remaining 27 variable pairs are not statistically significant with respect to any of the metrics.
- The following results were obtained, out of the 36 possible variable pairs for the virtual machine:
  - One variable pair difference is statistically significant with respect to the execution time metric.

- Five variable pair differences are statistically significant with respect to the speedup metric.
- Five variable pair differences are statistically significant with respect to the efficiency metric.
- The remaining 25 variable pairs are not statistically significant with respect to any of the metrics.

- The following results were obtained, out of the 12 possible variable pairs for both machines:
  - The four variable pair differences with respect to the execution time metric are statistically significant.
  - The remaining eight variable pair differences (which are with respect to the speedup and efficiency metrics) are not statistically significant.

Furthermore, in general, no variable pairs, with respect to any particular performance metric, demonstrated considerably more statistically significant differences than others.

TABLE I
INFORMATION ON DIFFERENCES THAT ARE STATISTICALLY SIGNIFICANT

Machine Experimented on	N-Thread/Core Performance Metrics	Statistically Significant Differences
Actual machine	2-Thread/Core performance metrics	There are no statistically significant differences between any of the variable pairs
	4-Thread/Core performance metrics	There are no statistically significant differences between any of the variable pairs
	6-Thread/Core performance metrics	The following differences are statistically significant: ▪ The speedup difference between afternoon and evening runs ▪ The efficiency difference between afternoon and evening runs ▪ The execution time difference between morning and evening runs ▪ The speedup difference between morning and afternoon runs ▪ The speedup difference between morning and evening runs ▪ The efficiency difference between morning and afternoon runs ▪ The efficiency difference between morning and evening runs
	8-Thread/Core performance metrics	The following differences are statistically significant: ▪ The execution time difference between morning and afternoon runs ▪ The efficiency difference between morning and afternoon runs
Virtual machine	2-Thread/Core performance metrics	The following difference is statistically significant: The execution time difference between morning and afternoon runs
	4-Thread/Core performance metrics	The following differences are statistically significant: ▪ The speedup difference between morning and afternoon runs ▪ The efficiency difference between morning and afternoon runs
	6-Thread/Core performance metrics	The following differences are statistically significant: ▪ The speedup difference between morning and afternoon runs ▪ The speedup difference between morning and evening runs ▪ The speedup difference between afternoon and evening runs ▪ The efficiency difference between morning and afternoon runs ▪ The efficiency difference between morning and evening runs ▪ The efficiency difference between afternoon and evening runs
	8-Thread/Core performance metrics	The following differences are statistically significant: ▪ The speedup difference between morning and evening runs ▪ The efficiency difference between morning and evening runs
Actual machine vs. virtual machine	2-Thread/Core performance metrics 4-Thread/Core performance metrics 6-Thread/Core performance metrics 8-Thread/Core performance metrics	The following differences are statistically significant for all the Threads/Cores (i.e., 2, 4, 6 and 8 Thread/Cores): Execution time differences between runs on actual and virtual machines

TABLE II
AVERAGE EXECUTION TIMES FOR ACTUAL MACHINE

Parallelization Strategy	1 Thread/Core	2 Threads/Cores	4 Threads/Cores	6 Threads/Cores	8 Threads/Cores
Batch	1291.67	680.36	500.31	488.07	567.43
Burglary	1818.33	928.75	511.61	382.82	304.50
Optimistic	1823.00	933.09	505.33	387.78	301.12
Optimistic (circle)	1685.67	895.76	518.56	468.02	489.18
Optimistic (prev)	1860.00	956.49	514.33	381.31	307.83
Pessimistic	1776.33	914.28	536.87	563.92	594.04

TABLE III
AVERAGE EXECUTION TIMES FOR VIRTUAL MACHINE

Parallelization Strategy	1 Thread/Core	2 Threads/Cores	4 Threads/Cores	6 Threads/Cores	8 Threads/Cores
Batch	2827.00	1515.00	924.54	887.71	945.06
Burglary	3674.33	1861.33	1057.67	770.28	738.03
Optimistic	3664.33	1908.00	1036.28	792.83	675.95
Optimistic (circle)	3501.00	1869.00	1107.33	1023.00	1091.33
Optimistic (prev)	3642.33	1913.33	1063.67	777.20	673.25
Pessimistic	3536.00	1862.00	1166.00	1214.00	1242.33

TABLE IV
RATIOS OF AVERAGE EXECUTION TIMES

Ratios of Average Execution Times for Actual Machine to Average Execution Times for Virtual Machine					
Parallelization Strategy	**1 Thread/Core**	**2 Threads/Cores**	**4 Threads/Cores**	**6 Threads/Cores**	**8 Threads/Cores**
Batch	2.19	2.23	1.85	1.82	1.67
Burglary	2.02	2.00	2.07	2.01	2.42
Optimistic	2.01	2.04	2.05	2.04	2.24
Optimistic (circle)	2.08	2.09	2.14	2.19	2.23
Optimistic (prev)	1.96	2.00	2.07	2.04	2.19
Pessimistic	1.99	2.04	2.17	2.15	2.09

C. Performance of Actual Machine vs. Virtual Machine in Terms of Execution Time

Tables II and III furnish averages of execution times, for each parallelization strategy, on both the actual and virtual machines; these averages are taken over morning, afternoon and evening runs.

D. Performance of Actual Machine vs. Virtual Machine in Terms of Speedup and Efficiency

An analysis of Figs. 3-7 shows that these graphs are not very different from each other – i.e., data, on both speedups and efficiencies, shows that both the actual and virtual machines demonstrated very similar performances (in terms of these metrics) at executing the programs for the various parallelization strategies (however, this information should be considered in light of the fact that the actual machine is approximately twice faster than the virtual machine at executing the programs for the various parallelization strategies). This result is consistent with the result of the *t*-test performed on the data, which shows that the difference between the performances of both machines, in terms of the performance metrics of speedup and efficiency, is not statistically significant. This difference, which is not statistically significant, between the performances of both machines (in terms of the speedup and efficiency metrics) may be explained by the fact that these metrics were computed independently for both machines. Consequently, even though machine A executes a particular benchmark significantly faster than another machine B executes the same benchmark and both machines have about the same specifications, both machines can actually have comparable or similar speedups and efficiencies at processing the same benchmark.

E. Performances of Various Algorithms on both Machines during Different Runs

1) For each machine, the vast majority of statistically significant differences, between different runs, were with respect to the speedup and efficiency metrics (these differences were nearly exclusively with respect to the speedup and efficiency metrics and were with respect to the execution time metric on only a very few occasions). This result is somewhat the converse, a flip over, of the results on statistically significant differences between the performances of the machines themselves. In particular, note the following result summaries:

a) Out of nine statistically significant differences between different runs for the actual machine, only two were with respect to the execution time metric (the other seven were with respect to the speedup and efficiency metrics).

b) Out of 11 statistically significant differences between different runs for the virtual machine, only one was with respect to the execution time metric (the other 10 were with respect to the speedup and efficiency metrics).

2) For both machines, only about a quarter of all the possible variable pairs had statistically significant differences. In particular, consider the following result summaries:

a) Out of the 36 possible variable pairs for the actual machine, 27 variable pairs are not statistically significant with respect to any of the metrics.

b) Out of the 36 possible variable pairs for the virtual machine, 25 variable pairs are not statistically significant with respect to any of the metrics.

3) For the various algorithms, there were either no or only a few statistically significant differences (with respect to any performance metrics) between different runs when these algorithms are used with 2, 4 or 8 threads/cores. The preponderance of these statistically significant differences was observed when 6-threads/cores are used. In particular, consider the following result summaries:

a) Out of nine statistically significant differences between different runs for the actual machine, seven were observed when 6-threads/cores are used (the remaining two were observed when 8-threads/cores are used).

b) Out of 11 statistically significant differences between different runs for the virtual machine, six were observed when 6-threads/cores are used (the remaining five were observed when 2, 4 or 8 threads/cores are used).

An explanation for the aforementioned finding is that these results are either an anomaly or are caused by some phenomenon (i.e., since the vast majority of these statistically significantly different metrics are observed when 6-threads/cores were used with all parallelization strategies, this suggests that all these parallelization strategies are having relatively highly variable performances between various runs when 6-threads/cores were used). Regarding this observation and possible anomaly, information presented in the graphs in Figs. 3-7 may be able to shed some more light. An analysis or appraisal of these graphs shows that, in general, as the number of threads/cores increases from two through eight, there is a change in the general behavior of speedups and efficiencies at the point when the number of threads/cores gets to be equal to six.

4) In general, no particular pair of runs (i.e., either the morning and afternoon runs, the morning and evening

runs or the afternoon and evening runs) demonstrated considerably more statistically significant differences (with respect to various performance metrics) than other pairs of runs.

An explanation for the finding in point number one above is that these results are an anomaly. Execution time is a raw performance metric that is derived ***directly*** from the machines; consequently, this metric reveals an intrinsic characteristic or property of the machines. Unlike this metric, the speedup and efficiency performance metrics were derived or computed from other quantities (by using the usual formulae for computing speedup and efficiency); these metrics reveal combined characteristics or properties of both machine and algorithm taken or considered together.

In terms of differences in performance during different runs on the same machine, it is not unreasonable to expect that the statistical significance of a metric that reveals intrinsic behaviors of a machine will be less anomalous than the statistical significance of a metric that reveals combined behaviors of both machine and algorithm taken or considered together. The result that the metrics, which had statistically significant differences, do not represent intrinsic characteristics or properties of the machines appears to be a reasonable basis for the suggestion that the finding in point number one above may simply be an anomaly.

In this study, statistically significant differences, between the performances of the same benchmark algorithms, on the same machine, during different runs of these algorithms, were observed. This is not an observation that one would expect to encounter and the finding in point number two above (which shows that this observation is not very prevalent) maybe provide ***some*** support for not expecting this result. However, it is only for a physical machine that it is reasonable to not expect this observation; it is reasonable to expect this observation for a cloud machine. This is because there should be more variability, in performance between different runs, for the virtual machine than for the actual machine (if at all any significant variability for the actual machine).

The reason for expecting the being of the aforementioned greater variability in performance for the virtual machine (than for the actual machine) is because the physical machine does not share its hardware with other machines at all, while the cloud machine typically shares some common underlying hardware with other virtual machines. One would expect that any particular virtual/cloud machine would demonstrate a significantly wider range of different, varying performances or capabilities during any long enough period (say, for example, a 24-hour period; this is a period for which the usage characteristics of a system may repeat in a pattern that occurs during 24-hour-long cycles). A greater variable performance for the cloud machine should be expected because, for different times during a long enough period, the following would happen:

1) . Unlike for a physical system, for a cloud system, the probability that multiple (virtual) machines would be competing for some single underlying hardware's resources and capabilities is nonzero.
2) During those times, when multiple virtual machines actually compete for some single underlying hardware's resources and capabilities, the performance of each individual virtual machine changes significantly or considerably, to one extent or another, depending on the number of virtual machines that get to run as well as the loads that these virtual machines get to process

F. Asymmetric Speedup Performance for the Batch Strategy Parallel Program

The implementation of the batch parallelization strategy results in speedup that is asymmetric. This speedup is characterized as symmetric using the following criteria (which are considered together, in combination) when two threads are used: (1) These two threads are assigned different kinds of tasks to perform and (2) The amount of work, which is performed by the one thread that is specified to be spawned, is significantly greater than the amount of work performed by the specialized thread; this observation is supported by the fact that, in Tables I and II, the reciprocals of speedup values, for the batch strategy parallel program, are significantly greater than 0.5 (a user of the batch strategy parallel program specifies that the program should spawn one of these threads, while the other thread is a specialized thread that also gets to be spawned). The variables in Tables V and VI are defined as follows:

1) The execution time t_S: This is the time it takes the serial version of the program to compute the Delaunay triangulation.
2) The execution time t_P: This is the time it takes two threads to compute the Delaunay triangulation. In the case of the batch strategy, these two threads are 1-thread that is specified to be spawned plus the specialized thread. In the case of the other strategies, these two threads are 2-threads that are specified to be spawned.

TABLE V
RECIPROCALS OF SPEEDUP VALUES WHEN 2-THREADS ARE USED (ACTUAL MACHINE)

Parallelization Strategy	Morning Run (t_S = 1746 ms)		Afternoon Run (t_S = 1681 ms)		Evening Run (t_S = 1671 ms)	
	t_P (ms)	Reciprocal of speedup (t_P/t_S)	t_P (ms)	Reciprocal of speedup (t_P/t_S)	t_P (ms)	Reciprocal of speedup (t_P/t_S)
Batch	1256.00	0.72	1361.00	0.81	1258.00	0.75
Burglary	918.39	0.53	933.78	0.56	934.07	0.56
Optimistic	909.74	0.52	934.27	0.56	955.25	0.57
Optimistic (circle)	892.50	0.51	911.26	0.54	883.51	0.53
Optimistic (prev)	1025.00	0.59	940.05	0.56	904.41	0.54
Pessimistic	888.00	0.51	923.04	0.55	931.81	0.56

TABLE VI
RECIPROCALS OF SPEEDUP VALUES WHEN 2-THREADS ARE USED (VIRTUAL MACHINE)

Parallelization Strategy	Morning Run (t_S = 3348 ms)		Afternoon Run (t_S = 3608 ms)		Evening Run (t_S = 3453 ms)	
	t_P (ms)	Reciprocal of speedup (t_P/t_S)	t_P (ms)	Reciprocal of speedup (t_P/t_S)	t_P (ms)	Reciprocal of speedup (t_P/t_S)
Batch	2798.00	0.84	2905.00	0.81	2778.00	0.80
Burglary	1803.00	0.54	1887.00	0.52	1894.00	0.55
Optimistic	1846.00	0.55	1922.00	0.53	1956.00	0.57
Optimistic (circle)	1842.00	0.55	1910.00	0.53	1855.00	0.54
Optimistic (prev)	1880.00	0.56	1962.00	0.54	1898.00	0.55
Pessimistic	1833.00	0.55	1851.00	0.51	1902.00	0.55

Going by the two criteria employed to characterize or define asymmetric speedup, we may generalize the observation of asymmetric speedup beyond only the case when two threads are used. Even though the data presented in Tables V and VI is only for the case when two threads are used, there is no reason to doubt that similar results will be seen for the case when $x > 2$ threads used, where x is defined, for the batch strategy as $x = \boldsymbol{N}$ threads specified to be spawned + the one specialized thread and, for the other strategies, as $x =$ only the $\boldsymbol{N}$ threads specified to be spawned. Furthermore, the $\boldsymbol{N}$ threads, which a user of the batch strategy parallel program specifies that the program should spawn, are assigned the same task to perform and this task is different than the task that is assigned to the specialized thread to perform.

The graphs in Figs. 3-7 show what looks like superlinear speedup performance for the batch strategy parallel program, when a user of the program specifies that two threads should be spawned. However, this performance is not truly superlinear speedup performance. Rather, this batch strategy parallel program performance is as a result of the fact that the speedup and efficiency values data, which is plotted in the graphs of Figs. 3-7, was computed (for the parallel programs of all the parallelization strategies) based on using $\boldsymbol{N_{specified}}$ (i.e., number of threads specified to be spawned), not based on using $\boldsymbol{N_{spawned}}$ (i.e., number of threads that were actually spawned). For the batch strategy parallel program, $\boldsymbol{N_{spawned}} = \boldsymbol{N_{specified}} + 1$ while, for the parallel programs of all the other strategies, $\boldsymbol{N_{spawned}} = \boldsymbol{N_{specified}}$.

It is worth noting that the implementation of the batch strategy parallel program results in either asymmetric or pseudo superlinear speedup performance for this parallel program, depending on which of two ways is employed to compute speedup and efficiency values for the program. One of these ways (by which to compute speedup and efficiency values) is to use $\boldsymbol{N_{specified}}$ (as has been done and the resulting data plotted in Figs. 3-7). When speedup and efficiency values are computed this way, pseudo superlinear performance is observed. The other way (by which to compute speedup and efficiency values) is to use $\boldsymbol{N_{spawned}}$. When speedup and efficiency values are computed this way, we have speedup that is asymmetric (recall the two criteria employed for the aforementioned definition of asymmetric speedup).

For the batch strategy parallel program, $\boldsymbol{N_{spawned}} = \boldsymbol{N_{specified}} + \boldsymbol{1}$ because the implementation of this parallel program is such that, whenever a user of the program specifies that only $\boldsymbol{N}$ threads should be spawned, the program actually spawns $\boldsymbol{N}$ + 1 threads [11]-[13]. This explanation for the observed pseudo superlinear scalability performance is also supported by data collected during the experiments in this study (in particular, the data presented in Tables VII and VIII and the charts in Figs. 8-13). Regarding this data collected during the experiments in this study, consider the following variables and the ensuing discussion:

1) The execution time, t_S, when the serial version of the program is run. This serial version of the program is independent on any of the various parallelization strategies employed for the various parallel versions of the algorithm (i.e., it is a sequential version of the algorithm, which has not been parallelized in any way, let alone been parallelized using any of the various parallelization strategies employed for the various parallel versions of the algorithm).
2) The execution time, t_T, when any of the various parallel versions of the program is run and it is specified that this parallel version of the program should be executed with only one thread used/spawned (each of the various parallel versions of the algorithm has been parallelized using one or the other of the various parallelization strategies).

TABLE VII
RELATIONSHIP BETWEEN t_S - t_T AND 2-THREAD/CORE EFFICIENCY (ACTUAL MACHINE)

Parallelization strategy	Morning Run		Afternoon Run		Evening Run	
	t_S - t_T (ms)	2-Thread/Core Efficiency (%)	t_S - t_T (ms)	2-Thread/Core Efficiency (%)	t_S - t_T (ms)	2-Thread/Core Efficiency (%)
Batch	490	132.0	320	120.8	413	122.1
Burglary	-19	95.1	-194	90.0	-144	89.4
Optimistic	-53	96.0	-157	90.0	-161	87.5
Optimistic (circle)	30	97.8	-40	92.2	51	94.6
Optimistic (prev)	-180	85.2	-90	89.4	-212	92.4
Pessimistic	33	98.3	-119	91.1	-145	89.7

TABLE VIII
RELATIONSHIP BETWEEN t_S - t_T AND 2-THREAD/CORE EFFICIENCY (VIRTUAL MACHINE)

Parallelization strategy	Morning Run		Afternoon Run		Evening Run	
	t_S - t_T (ms)	2-Thread/Core Efficiency (%)	t_S - t_T (ms)	2-Thread/Core Efficiency (%)	t_S - t_T (ms)	2-Thread/Core Efficiency (%)
Batch	550	113.4	703	111.3	675	119.2
Burglary	-200	92.8	-229	95.6	-185	91.2
Optimistic	-457	90.7	-47	93.9	-80	88.3
Optimistic (circle)	-198	90.9	76	94.5	28	93.1
Optimistic (prev)	-493	89.0	215	91.9	-240	91.0
Pessimistic	-282	91.3	125	97.5	-42	90.8

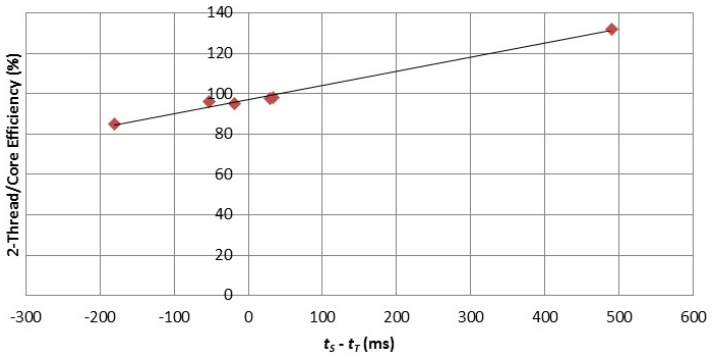

Fig. 8 2-Thread/Core Efficiency against t_S - t_T (Morning Run on Actual Machine)

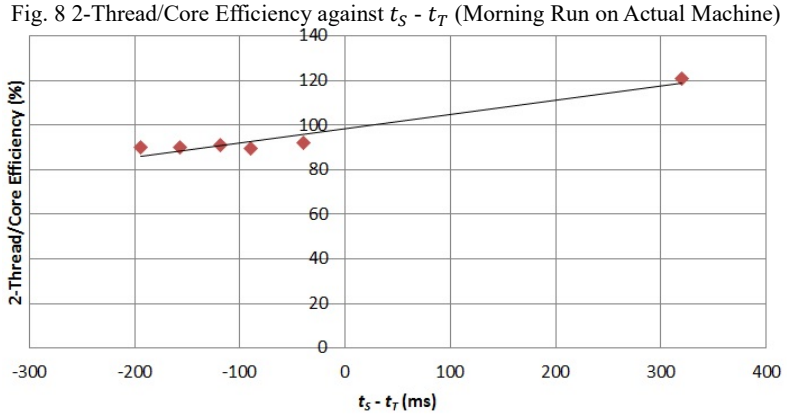

Fig. 9 2-Thread/Core Efficiency against t_S - t_T (Afternoon Run on Actual Machine)

Tables VII and VIII show the relationship, between the difference t_S - t_T and the 2-Thread/Core efficiency values, for both the virtual and actual machines as well as for all parallelization strategies. Figs. 8-13 furnish graphs/charts of the data in Tables VII and VIII. The data in Tables VII and VIII show that, in general, the higher the difference t_S - t_T, the higher the 2-Thread/Core efficiency value; this relationship is also shown by the linear trend lines in Figs. 8-13.

The data in Tables VII and VIII show that the difference t_S - t_T ranges from +320ms to +703ms, during all runs on both actual and virtual machines, for the batch strategy parallel program, which is the program that demonstrates pseudo superlinear speedup. The following points, which may be derived from Tables VII and VIII and Figs. 8-13, suggest that the serial/sequential program is possibly performing some extra processing, which the 1-thread-spawned instance of the batch strategy parallel program is not performing:

1) $t_S > t_T$ for all cases when the batch strategy parallel program runs, on either the actual or virtual machine, during any run. This fact is true for only the batch strategy parallel program alone (and untrue for the other versions of the parallel program, which are implemented using the other parallelization strategies).
2) The differences t_S - t_T are greater, by far, for the batch strategy than these differences for the various other strategies (on either the actual or virtual machine, during any run).

The foregoing data/facts, about the batch parallelization strategy, support and demonstrate the insight that, for this strategy, the executing serial version of the randomized incremental insertion algorithm does more work than the executing parallel version of the algorithm using the batch parallelization, with 1 thread spawned.

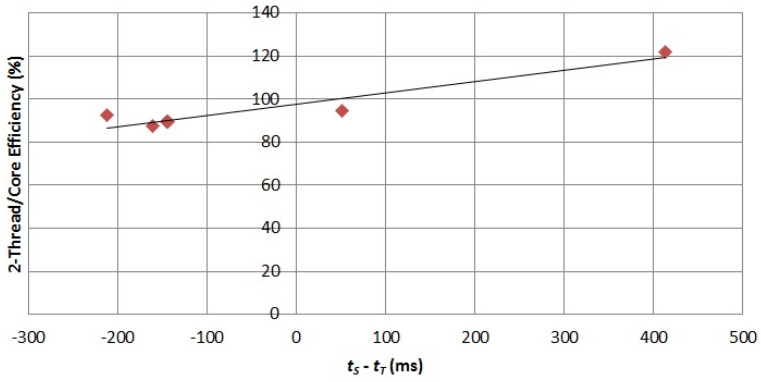

Fig. 10 2-Thread/Core Efficiency against t_S - t_T (Evening Run on Actual Machine)

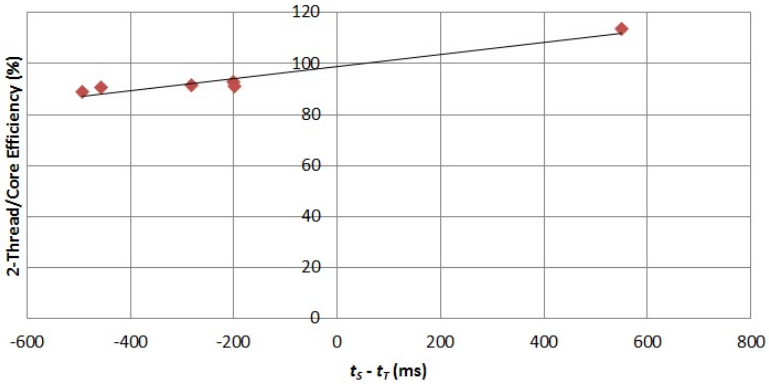

Fig. 11 2-Thread/Core Efficiency against t_S - t_T (Morning Run on Virtual Machine)

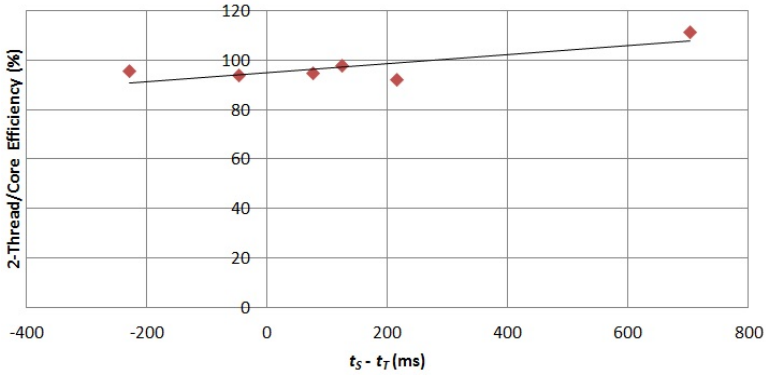

Fig. 12 2-Thread/Core Efficiency against t_S - t_T (Afternoon Run on Virtual Machine)

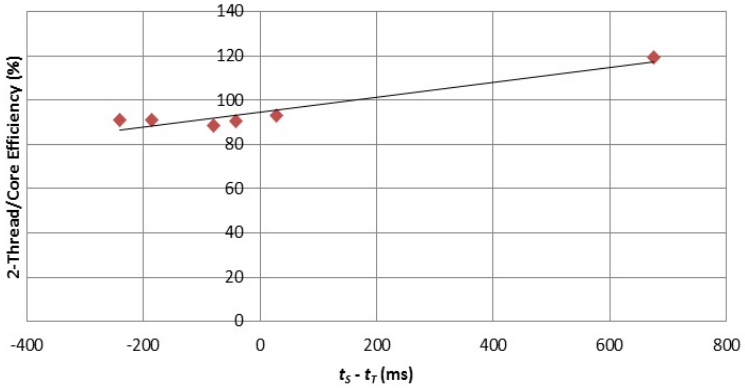

Fig. 13 2-Thread/Core Efficiency against t_S - t_T (Evening Run on Virtual Machine)

V. Conclusion

This study evaluated the performances of two types of the multicore system at executing various parallelization strategies of the incremental insertion technique of the Delaunay triangulation algorithm. The multicore systems used in the study are actual or physical and virtual or cloud multicore machines. The background to and methodology employed for this study is presented. The study produced results that are analyzed as well as herein presented and discussed/interpreted in fairly great detail. Results show that the differences between the performances of both machines were statistically significant with respect to the execution time metric, but not with respect to the speedup and efficiency metrics (the actual machine is approximately twice faster than the virtual machine at executing the programs for the various parallelization strategies). The results also show that some of the differences between the performances of various algorithms on both machines, during different runs of the algorithms on the systems, were statistically significant with respect to various performance metrics employed in this study. A perusal of results shows the scalability behaviors of the various parallelization strategies. An interesting result observed is that, depending on which of two ways is employed to compute speedup and efficiency values, the batch strategy parallel program results in either asymmetric or pseudo superlinear speedup performance for this program (these pseudo superlinear speedup values were observed with only the batch strategy parallel program, when the number threads specified to be spawned is equal to 2).

VI. Future Work

This study investigated the performances of only two types of one class of parallel systems, namely: (1) Cloud multicore system and (2) Physical multicore system. Future studies, which extend this research, may investigate questions and hypotheses on the performance, characteristics and behavior of parallel systems, which have not been considered in this study, for parallel surface modelling and generation using parallel Delaunay triangulation. The recommended areas for future research, which are herein furnished, are given in the backdrop or context of the understanding that systems on which parallel processing may be performed are getting increasingly ubiquitous – multicore and similar parallel systems promise as well as are envisaged to be increasingly ubiquitous. Consequently, any identified significance or benefits of using these systems promises to be a benefit for not only the present, but also for the foreseeable future.

In the not-too-distant history of computing, parallel computing systems have been only high-end systems that get to be exclusively reserved for only very few users, especially in the academia and government; however, a trend that has been around in the past decade or so, as well as shows promise of being around for some time into the future, is a great increase in the prevalence of parallel computing systems. The prevalence of these systems is such that they are quite ubiquitous indeed (a very apt example of this current and future trend is the abundance of multicore systems, which have enabled parallel computing systems – ranging from personal computers to smart phones – to be made available in the hands of "ordinary" people, for performing their everyday computing tasks).

Furthermore, it is easy to foresee a future in which research in and development of parallel algorithms, for the following applications, will continue to be undertaken: Computational geometry applications, in general, and Delaunay triangulation applications, in particular. A major motivation, for these forecast research and development efforts, will be the goal of exploiting the many, varied and increasingly ubiquitous computing systems on which parallel processing can be accomplished or performed. Consequently, it is also easy to see how these forecast research and development efforts would be apt recommended areas for future research – i.e., research in and development of parallel algorithms in an area/field, which entails the application of parallel computing concepts, principles and techniques for constructing the Delaunay triangulation.

Another recommendation for future research is that a study be conducted and a control measure, which was not incorporated into this study, would be adopted in the recommended study. Since the platforms of the actual and virtual machines used in this study are not ***exactly*** the same, but are only ***comparably*** the same, it appears like it is useful to conduct research that employs system platform specifications that are exactly the same (rather than merely comparably the same). This recommendation is given despite an assumption made in [8] that, in general, the slight differences that exist between the platform specifications of the physical and cloud machines used in this research should not be significant for this particular study. This recommended research will eliminate the following possibility, which this assumption merely advises is ***unlikely***: The observed statistically significant discrepancy, between the execution time performances of the physical and cloud machines, is due to the differences between the platform specifications of both machines.

Some related areas, which are also recommended for further research, for either various individual Delaunay triangulation algorithms or various categories of this algorithm, include evaluating and investigating – including comparing and contrasting – the following: (A) Code complexities of the program codes that implement the algorithms and (B) Computational complexities of the algorithms, in terms of the following and particularly for multicore systems: (i) Execution/running time (ii) Storage space required and (iii) Processor-storage accesses required.

References

[1] Blelloch, G. E., Miller, G. L., Hardwick, J. C. and Talmor, D. (1999). Design and Implementation of a Practical Parallel Delaunay Algorithm. *Algorithmica, 24*(3-4), 243-269. doi: 10.1007/pl00008262

[2] Blelloch, G. E., Miller, G. L., and Talmor, D. (1996). *Developing a practical projection-based parallel Delaunay algorithm.* Paper presented at the Proceedings of the twelfth annual symposium on Computational geometry, Philadelphia, Pennsylvania, USA.

[3] Chen, M.-B., Chuang, T.-R., and Wu, J.-J. (2001). *Efficient Parallel Implementations of 2D Delaunay Triangulation with High Performance Fortran.* Paper presented at the PPSC.

[4] Chen, M.-B., Chuang, T.-R., and Wu, J.-J. (2002). *A parallel divide-and-conquer scheme for Delaunay triangulation.* Paper presented at the Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on.

[5] Chen, M.-B., Chuang, T.-R., and Wu, J.-J. (2006). Parallel divide-and-conquer scheme for 2D Delaunay triangulation: Research Articles. *Concurr. Comput. : Pract. Exper., 18*(12), 1595-1612. doi: 10.1002/cpe.v18:12

[6] Cignoni, P., Laforenza, D., Perego, R., Scopigno, R., and Montani, C. (1995). Evaluation of parallelization strategies for an incremental Delaunay triangulator in E^3. *Concurrency: Practice and Experience, 7*(1), 61-80. doi: 10.1002/cpe.4330070106

[7] Cignoni, P., Montani, C., Perego, R. and Scopigno, R. (1993). Parallel 3D Delaunay Triangulation. *Computer Graphics Forum, 12*(3), 129-142. doi: 10.1111/1467-8659.1230129

[8] Gyang, N. P. (2014). *Performance evaluation of parallel surface generation from LiDAR point clouds on actual and virtual multicore systems* (Doctoral dissertation, Colorado Technical University).

[9] Hardwick, J. C. (1997). *Implementation and evaluation of an efficient parallel Delaunay triangulation algorithm*. Paper presented at the Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures, Newport, Rhode Island, USA.

[10] Held, J., Bautista, J. and Koehl, S. (2006). From a few cores to many: A tera-scale computing research overview. *white paper, Intel*.

[11] Kohout, J. and Kolingerová, I. (2003). Parallel Delaunay triangulation in E3: make it simple. *The Visual Computer, 19*(7-8), 532-548. doi: 10.1007/s00371-003-0219-x

[12] Kohout, J., Kolingerová, I. and Ára, J. (2005). Parallel Delaunay triangulation in E^2 and E^3 for computers with shared memory. Parallel Comput., 31(5), 491-522. doi: citeulike-article id:3386087, doi: 10.1016/j.parco.2005.02.010

[13] Kolingerová, I. and Kohout, J. (2002). Optimistic parallel Delaunay triangulation. *The Visual Computer, 18*(8), 511-529. doi: 10.1007/s00371-002-0173-z

[14] Lo, S. H. (2012a). Parallel Delaunay triangulation – Application to two dimensions. *Finite Elements in Analysis and Design, 62*(0), 37-48. doi: http://dx.doi.org/10.1016/j.finel.2012.07.003

[15] Lo, S. H. (2012b). Parallel Delaunay triangulation in three dimensions. *Computer Methods in Applied Mechanics and Engineering, 237–240*(0), 88-106. doi: http://dx.doi.org/10.1016/j.cma.2012.05.009

[16] Park, C.-M., Lee, S., and Park, C.-I. (2001). An improved parallel algorithm for delaunay triangulation on distributed memory parallel computers. *Parallel Processing Letters, 11*(02n03), 341-352. doi: doi:10.1142/S0129626401000634

[17] Puppo, E., Davis, L., De Menthon, D. and Teng, Y. A. (1994). Parallel terrain triangulation. *International Journal of Geographical Information Systems, 8*(2), 105-128. doi: citeulike-article-id:10466135 doi: 10.1080/02693799408901989

[18] Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. Paper presented at the Selected papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering.

[19] Teng, Y. A., Sullivan, F., Beichl, I. and Puppo, E. (1993). *A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation*. Paper presented at the Proceedings of the 1993 ACM/IEEE conference on Supercomputing, Portland, Oregon, United States.