electronics

MDPI

Article

EPSim-C: A Parallel Epoch-Based Cycle-Accurate Microarchitecture Simulator Using Cloud Computing

Minseong Kim [1], Seon Wook Kim [1] and Youngsun Han [2,*]

[1] Department of Electrical and Computer Engineering, Korea University, Seoul 02841, Korea; minseong@korea.ac.kr (M.K.); seon@korea.ac.kr (S.W.K.)
[2] Department of Electronic Engineering, Kyungil University, Gyeongsan 38428, Korea
* Correspondence: youngsun@kiu.ac.kr; Tel.: +82-53-600-5549

Received: 28 May 2019; Accepted: 19 June 2019; Published: 24 June 2019

Abstract: Recently, computing platforms have been being configured on a large scale to satisfy the diverse requirements of emerging applications like big data and graph processing, neural network, speech recognition and so on. In these computing platforms, each computing node consists of a multicore, an accelerator, and a complex memory hierarchy, which are connected to other nodes using a variety of high-performance networks. Up to now, researchers have been using cycle-accurate simulators to evaluate the performance of computer systems in detail. However, the execution of the simulators, which models modern computing architecture for multi-core, multi-node, datacenter, memory hierarchy, new memory, and new interconnection, is too slow and infeasible; since the architecture has become more complex today, the complexity of the simulator is rapidly increasing. Therefore, it is seriously challenging to employ them in the research and development of next-generation computer systems. To solve this problem, we previously presented EPSim (Epoch-based Simulator), which defines epochs that can be run independently by dividing the simulation run into several sections and executes them in parallel on a multicore platform, resulting in only the limited simulation speedup. In this paper, to overcome the computing resource limitations on multi-core platforms, we propose a novel EPSim-C (EPSim on Cloud) simulator that extends EPSim and achieves higher performance using a cloud computing platform. EPSim-C is designed to perform the epoch-based executions in a massively parallel fashion by using MapReduce on Hadoop-based systems. According to our experiments, we have achieved a maximum speed of 87.0× and an average speed of 46.1× using 256 cores. As far as we know, EPSim-C is the only existing way to accelerate the cycle-accurate simulator on cloud platforms; thus, our significant performance enhancement allows researchers to model and research current and future cutting-edge computing platforms using real workloads.

Keywords: microarchitecture simulation; epoch-based execution; parallel simulation; cloud computing

1. Introduction

Recently, as the demand for big data analysis and neural network processing has increased explosively, the use of data-intensive applications has also drastically increased [1,2]. Because data-intensive processing grows linearly in execution time with data size and has data-level parallelism, its performance can be effectively improved through parallelization [3,4]. Thus, there have been various approaches to accelerate the applications by exploiting data-level parallelism using immense amounts of hardware resources, such as in cloud and neural computing platforms [5–7] which are comprised of a variety of state-of-the-art multiple CPUs and GPUs, large-scale memory, high-speed network connections, etc. [8–11]. For example, we can use a cloud computing system consisting of many computing instances from the Amazon Elastic Compute Cloud (Amazon EC2) [8,12,13].

Each instance can be optionally configured using 1 to 128 cores, 1 to 3904 GB of memory, 100 M to 25 Gbps interconnects, and so on. We can now use high-performance nodes with more than tens of cores [14–16], GPUs with 5120 stream multiprocessors, and 16 GB of HBM2 memory [17]. Dedicated neural processing units and data analysis accelerators have been also widely adopted [18–20]. Therefore, these computing platforms are becoming increasingly diverse and complex to satisfy the different needs of the emerging processing technologies.

Many cycle-accurate full-system simulators have been used to model computing systems and evaluate their performance in detail [21–25]. However, since this approach takes too much simulation time, it is practically impossible to apply real workload or use it related to large-scale computing systems such as the cloud for their research and development.

There have been many studies on accelerating the simulators to resolve the problems above using a variety of hardware and software methods. FAST [26] is a hardware-assisted scheme that performs functional modeling through software and timing modeling via field programmable gate arrays (FPGAs). SimPoints [27] simulates critical code sections by investigating the most frequently executed sequences of basic blocks. Both Sniper [28] and IntervalSim [29] utilize abstraction models to estimate core performance without tracking all instructions, meaning they incur a trade-off between simulation speed and accuracy. P-DRAMSim2 [30], Transformer [31], and P-Mambo [32] employ multiple threads on multiple cores to accelerate simulation. Some threads perform functional modeling, and others play the cycle-accurate simulation. EPSim [33], which is our base model, was designed to perform epoch-based parallel microarchitecture simulation that produces deterministic results. Each epoch is identified based on the interactions between cycle-accurate timing simulations and functional emulations such that an epoch can be simulated independently of other epochs in parallel, meaning the epochs can exploit large-scale parallelism.

However, these previous solutions, except for EPSim, are constrained in maximizing performance via parallelization because of their inherently limited parallelism. For example, Transformer executes functional modeling and timing modeling in parallel. SlackSim [34] assigns one thread to each core of a target chip multiprocessor (CMP). In contrast, EPSim can exploit much higher parallelism. Figure 1 presents the number of epochs in EPSim for the SPEC CPU2006 benchmarks [35]. Each epoch can be executed in parallel, independently of the others. Because the average number of epochs in the SPEC CPU2006 benchmarks is approximately 164 million, it would be possible to improve the performance dramatically by employing significantly more computing resources for the simulation.

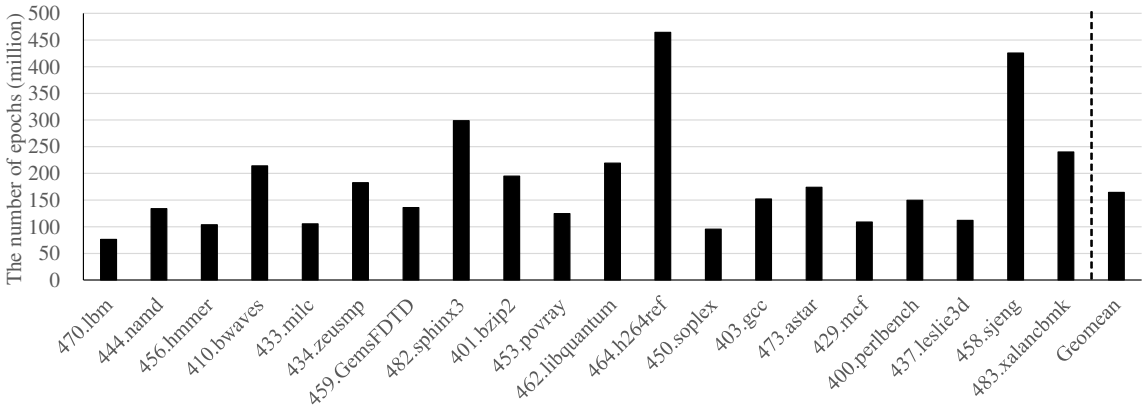


Figure 1. The number of epochs for the SPEC CPU2006 benchmarks [33].

To exploit massive parallelism for EPSim, we can utilize a cloud computing environment that supports large amounts of computing resources. Cloud computing platforms, such as Amazon Web Services (AWS) [13], Microsoft Azure [9], IBM Cloud [10], and Oracle Cloud [11], are widely used because they provide shared computing resources, such as servers, networks, and storage, through the Internet, regardless of their physical locations or system environments. The cloud computing

platforms utilize a cost-effective on-demand charging system that provides great convenience in terms of manageability and maintainability because users do not need to manage physical equipment directly. Using only a few clicks, we can scale up a computing system by using more powerful hardware devices and scale out by using a large number of devices, or vice versa.

In this paper, we propose an EPSim-C (EPSim on Cloud) simulator that extends EPSim [33], an epoch-based parallel microarchitecture simulator, to run in a cloud computing environment. Therefore, we can dramatically improve the performance of the simulation by exploiting the intrinsically massive parallelism of epochs defined in EPSim.

Because EPSim runs multiple epoch-based simulations at the same time, EPSim applied to the cloud environment can achieve a significant performance advantage by using its large amount of computing resources. EPSim also assigns one epoch to each core, runs the epoch-based simulation at the same time, and then merges the results of the echo-based simulation, which is the same process used in the MapReduce programming model [36]. Therefore, we designed EPSim-C based on MapReduce and found that, using Apache Hadoop-based cloud computing systems [37], we could achieve significant performance gains. For evaluation, we constructed a cloud computing system by adopting one NameNode and 17 DataNodes from Amazon EC2 and employing the Cloudera Hadoop platform, which supports YARN MR2, HDFS, HUE, Hive, etc. [38–40]. As a result, we have confirmed that EPSim-C achieves a maximum speed of 87.0× and an average speed of 32.6× using 256 cores on 16 DataNodes, providing performance scalability. In addition, in the Hadoop-based cloud computing system, we have developed an optimization method that reduces work-management overhead by grouping multiple epochs. The optimization made the average speed increased to 46.1× using 256 cores. As a result, these high-performance enhancements enable computing system researchers to work with real workloads.

In summary, we make the following contributions in this paper:

- We explore the parallelism of EPSim and accelerate its simulation significantly using cloud computing to exploit the massive parallelism.
- We present a Hadoop-based implementation of EPSim-C simulator to utilize the parallelism by applying the MapReduce programming model.
- We obtained on average 46.1× and up to 87.0× of the simulation speedup with 256 cores in a representative commercial cloud computing environment, i.e., AWS.

As far as we know, our EPSim-C is the first to accelerate microarchitecture simulations using cloud computing, providing performance scalability while modeling a cutting-edge computing platform for real workloads.

The remainder of this paper is organized as follows: Section 2 describes the background and the motivation of our study. In Section 3, we detail our parallel microarchitecture simulation method using cloud computing. We evaluate the performance of our method in terms of speedup and time distribution in Section 4. We provide related works and discussion in Section 5. Finally, our conclusions are made in Section 6.

2. Background and Motivation

In this section, we first describe cloud computing and a full-system microarchitecture simulator, i.e., MARSSx86. In addition, we present our previous multi-core based parallel EPSim simulator derived from MARSSx86. Finally, we identify the necessity for a new EPSim-C simulator using cloud computing on the basis of ideal speedups on many cores.

2.1. Cloud Computing

Figure 2 presents a conceptual diagram of a cloud platform. The cloud is an on-demand computing platform that provides shared computing resources, such as servers, storage, networks, applications, and services, via the Internet. The cloud computing empowers clients to utilize servers, storage,

and networks with just a few clicks while ignoring maintenance and management of the physical hardware components. The cloud clients can access cloud services by using a web browser, mobile applications, thin clients, terminal emulators, and so on over the Internet.

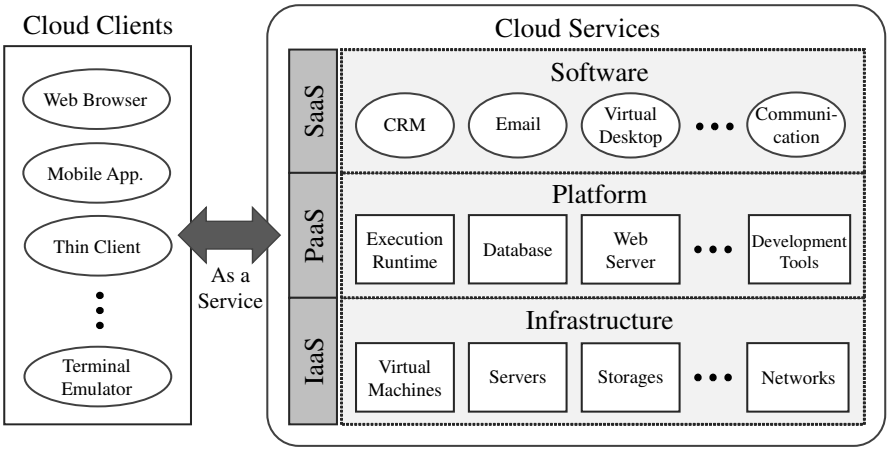


Figure 2. Conceptual diagram of a cloud platform.

The cloud services are classified into three categories: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). First, SaaS provides a variety of enterprise applications, such as CRM (Customer Relationship Management), ERP (Enterprise Resource Planning), email, and virtual desktops, which can be accessed using web browsers on a mobile phone, tablet, PC (Personal Computer), and so on. Software developers can significantly enhance the manageability and maintainability of applications by employing these services because application creators are not required to perform software upgrades, security patches, and so on. Second, to help establish web-based cloud applications, PaaS provides enterprise applications for developers as an integrated environment with development tools, execution runtime, databases, web servers, and so on. This service provides all of these resources with minimal cost and complexity for the maintenance and management of its hardware, software, provisioning, and hosting. Finally, IaaS provides all the computing resources necessary for application infrastructures, such as virtual machines, servers, storage, and networks, on a pay-on-use basis.

2.2. Microarchitecture Simulation: MARSSx86

A full-system microarchitecture simulator allows us to model and simulate systems ranging from applications to operating systems (OS) with a target physical machine. It provides powerful functionality to study and to test the runtime behaviors and interactions between computer architectures, operating systems, and applications for research and development [41]. In general, the simulator consists of a functional model that preserves functional correctness and a timing model that simulates detailed microarchitectural behaviors through interactions.

Figure 3 presents the architecture of the MARSSx86 system, which is a widely used full-system microarchitecture simulator. MARSSx86 runs as a virtual machine for evaluating and developing x86 ISA-based platforms. The simulator consists of a tightly coupled functional system and cycle-accurate timing simulator, called QEMU and PTLSim [23,42], respectively. QEMU functionally emulates a variety of guest applications on guest operating systems by adopting various types of emulation devices ranging from CPUs to network interface cards (NICs) [43]. QEMU translates guest instructions into host instructions using a code generator and executes the translated code on the host machine at near-native speed. PTLSim models x86-based computing platforms at a low level [24] and supports various hardware configurations, such as in-order/out-of-order, single/multicores, cache hierarchy, coherence protocols, deep memory hierarchy, hardware TLB, branch predictors, and peripherals. QEMU and PTLSim interact with each other to share architectural states for correct execution and

handle interrupts, exceptions, and complex instructions that cannot be handled by PTLSim alone. The overhead related to this interaction accounts for up to 35% of total simulation time [31,44].

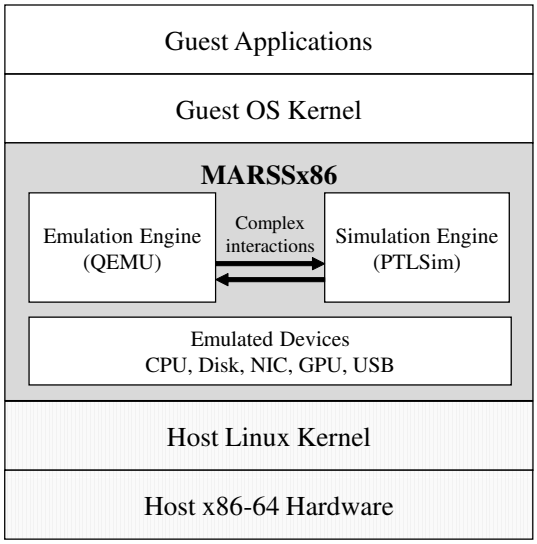


Figure 3. MARSSx86 simulator framework.

2.3. EPSim: Epoch-Based Parallel MARSSx86 Simulator

As mentioned in Section 2.2, the interaction related overhead takes up a considerable portion of the total simulation time. In order to accelerate the MARSSx86 simulator by eliminating the causal interactions, EPSim [33] was developed. EPSim is an epoch-based cycle-accurate parallel MARSSx86 simulator. When the simulation is executed, the interaction between QEMU and PTLSim is removed, and the simulation interval that PTLSim can independently perform is defined as an epoch. Many epochs defined in this way are applied to PTLSim, respectively, and executed in parallel to improve performance. Figure 4 presents the epochs and the simulation flow in the MARSSx86 simulator. MARSSx86 executes a sequence of instructions in translation blocks, which are another form of a basic block used in MARSSx86. Once PTLSim encounters interrupts, exceptions, or complex instructions that it cannot handle, an internal exception is triggered for switching from timing simulation to functional emulation, which is handled by QEMU. Once the issue is handled, the simulator switches back to timing simulation. Therefore, in order to remove the interaction between the two processes, an epoch is defined as a sequence of translation blocks that does not involve any interactions between QEMU and PTLSim. For example, *epoch 0* is defined to be the same as *TB 0* and *epoch 1* is comprised of multiple translation blocks, i.e., *TB 1* to *TB M-1*.

In order to simulate such epochs using PTLSim, essential live-in architectural states called epoch snapshots are required, such as CPU states, memory states, and memory data. QEMU in MARSSx86 only performs function simulation, detects epochs, and records epoch snapshots. PTLSim performs epoch simulation on the epoch snapshots generated by QEMU in a completely parallel manner using multicore resources. Finally, the results of each epoch simulation are combined at the end of the entire simulation. As a result, the epoch-based parallel simulator has large-scale parallelism because epochs can be simulated independently in a completely parallel manner. Further details can be found in Kim et al. [33].

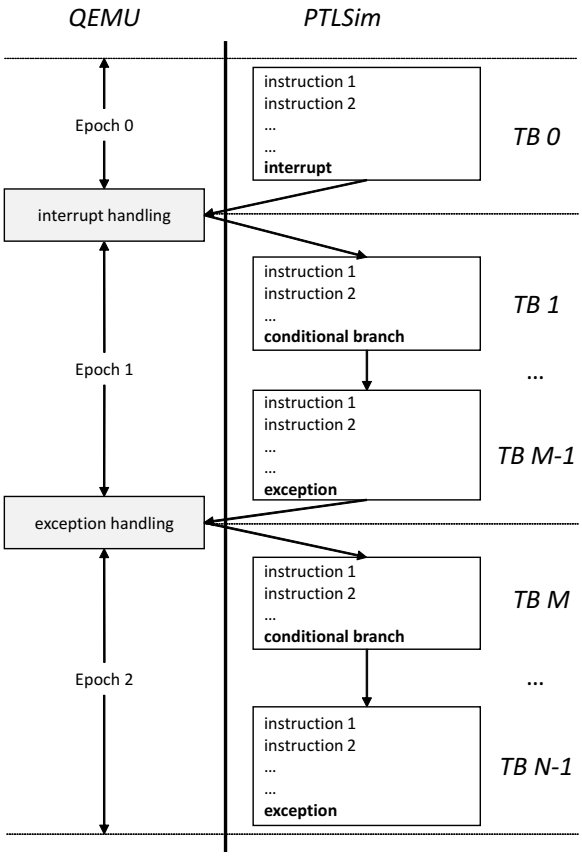


Figure 4. Epochs and simulation flow in the MARSSx86 simulator [33].

2.4. Motivation

Figure 5 presents the speedup achieved by our existing epoch-based parallel microarchitecture simulator, called EPSim, using a multicore platform on the SPEC CPU2006 benchmarks [33]. EPSim obtained an average speedup of 1.0×, 3.7×, 6.9×, and 13.0× using 1, 4, 8, and 16 cores, respectively, in a multicore machine. These results confirm that EPSim has obvious performance scalability depending on the number of available cores. Thus, we can estimate the ideal speedup by assigning one epoch to each core and ignoring parallelization overheads with 32, 64, 128, and 256 cores, as shown in Figure 6. We obtained an ideal speedup of 20.4× with 32 cores, 29.9× with 64 cores, 39.3× with 128 cores, and 46.8× with 256 cores.

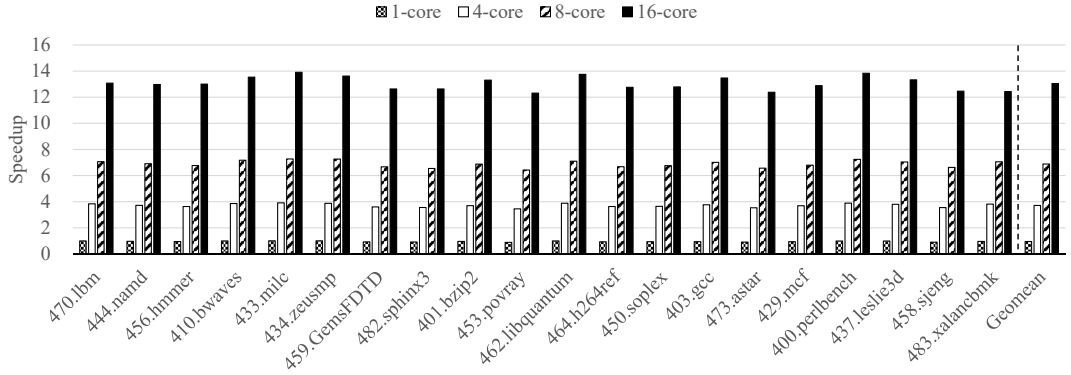


Figure 5. The speedup achieved by EPSim using a multicore platform on the SPEC CPU2006 benchmarks [33].

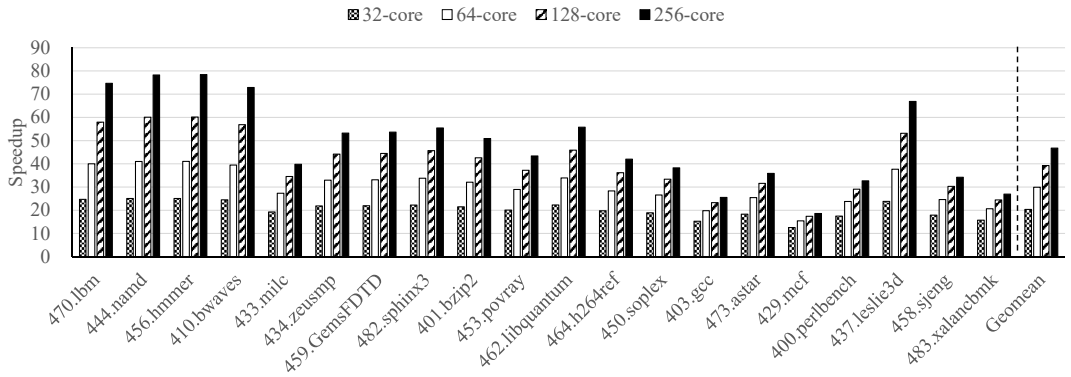


Figure 6. The ideal speedup of EPSim using many cores on the SPEC CPU2006 benchmarks.

However, because the EPSim simulator is designed to utilize only a single multicore machine, it cannot exploit extremely large-scale parallelism. In other words, because the average number of epochs is approximately 164 million as in Figure 1, we are certain that performance can be improved dramatically by employing many more cores for simulation. Therefore, to fully exploit the massive parallelism available in the epoch-based simulation, we propose epoch-based parallel microarchitecture simulation in a cloud computing platform that can employ a large number of high performance computing resources on demand.

3. Parallel Microarchitecture Simulation Using Cloud Computing

In this section, we explore the massive parallelism of our epoch-based parallel simulator using the MapReduce programming model. Additionally, we present a Hadoop-based implementation of epoch-based parallel MARSSx86 simulation.

3.1. Exploring Massive Parallelism Using MapReduce

The existing epoch-based parallel MARSSx86 simulator was designed to accelerate simulation on a single machine by executing multiple epochs simultaneously at the process-level. However, the performance of the simulator is not sufficient to manage real workloads. Cloud computing platforms, which have become a popular computing paradigm [45], are one potential solution to this problem because simulations consist of epochs that are highly parallel and independent of each other. The simulator effectively exploits massive parallelism through cloud computing by adopting the MapReduce programming model because its processing structure similar to that of EPSim.

Figure 7 depicts the logical data flow in a MapReduce program. The program is comprised of *Map* and *Reduce* functions. The *Map* function, i.e., mapper, is designed to return intermediate data values by filtering and sorting input data. The *Reduce* function, i.e., reducer, produces output data by combining the intermediate data from the mapper. A task scheduler feeds each input split into the mappers. The mappers then execute a predefined process and output intermediate files as partial results. After being shuffled and partitioned, the intermediate files are fed into the reducers, which merge the partial results into final output. In summary, the flow of MapReduce has a similar structure to EPSim [33]. An epoch (input split) is assigned to each core, simulated (mapped), and partial results (intermediate files) are produced. The final output is merged (reduced) from the partial results.

In order to adopt the MapReduce framework, we design functions for the mapper to take epochs as input splits. The mapper simulates the instructions of the input epochs sequentially by using PTLSim and records the simulation results in intermediate files. The reducers then return the final output data by combining all the partial results from the mappers. By employing the MapReduce framework on a cloud computing platform, total simulation time is notably diminished because we can use a very large number of mappers and reducers simultaneously.

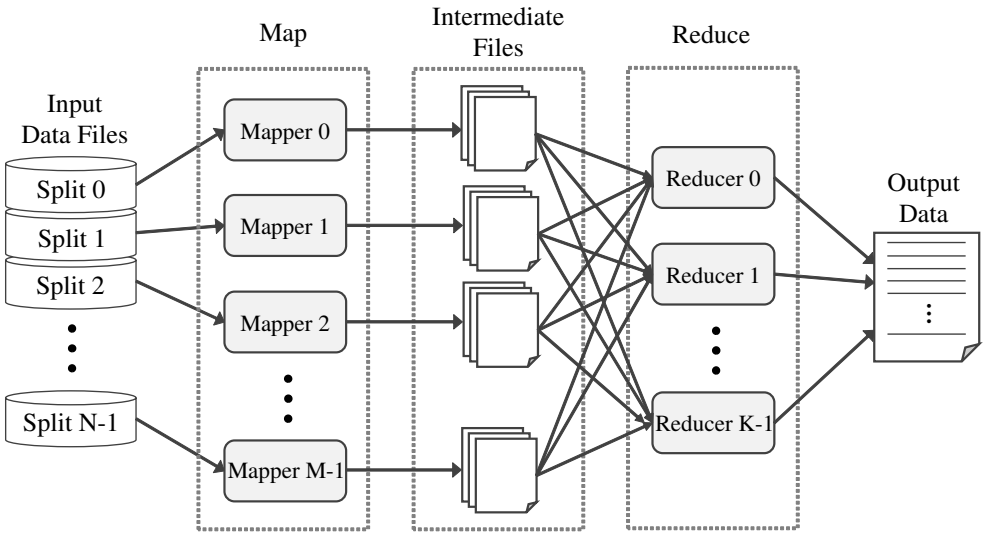


Figure 7. Logical data flow in a MapReduce program.

3.2. *Hadoop-Based Implementation of Epoch-Based Parallel MARSSx86 Simulation*

Figure 8 depicts the overall structure of our proposed microarchitecture simulator that performs epoch-based parallel simulation using the Hadoop infrastructure [37]. For employing Java-based Hadoop, the Java native interface (JNI) is used for the mapper to access the epoch-based MARSSx86 simulation that is written in C/C++. We assume that the input data, i.e., epoch snapshots (ES), were previously generated by QEMU and are located on the Hadoop Distributed File System (HDFS).

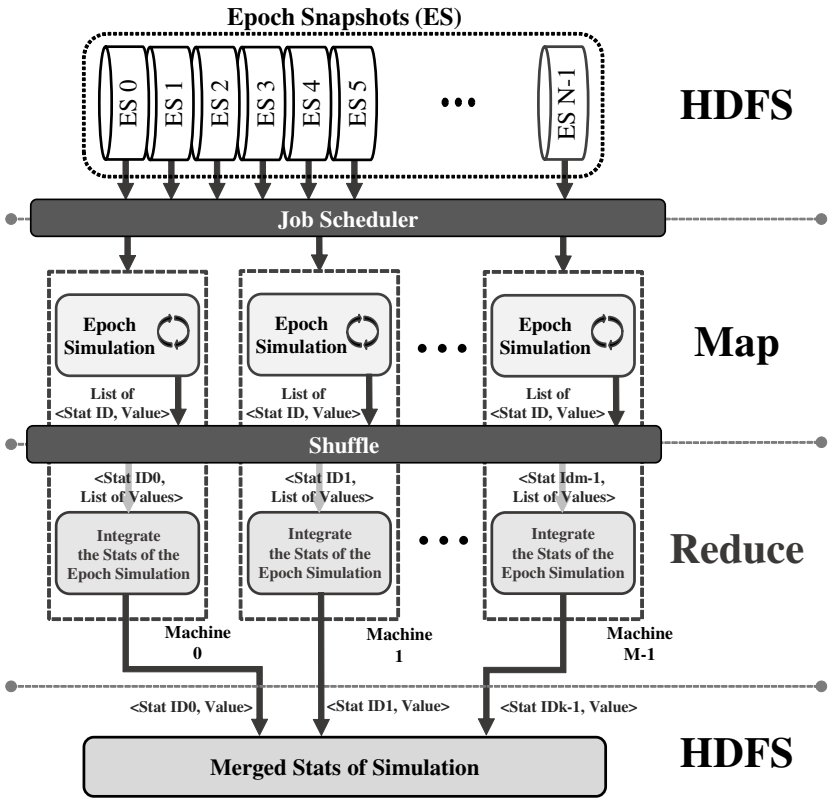


Figure 8. Architecture of our Hadoop-based parallel microarchitecture simulator. Stat represents the statistical results collected during a simulation period. ES indicates an epoch snapshot.

The simulator mainly consists of the following four components: input data reader, mapper, reducer, and output data writer. First, the input data reader reads all of the epoch snapshots from the HDFS and delivers them to the Hadoop job scheduler. Second, after the job scheduler assigns each epoch snapshot to mapper functions, the mapper performs epoch simulation independently in parallel. The mapper also returns a stats key and value pair, i.e., the ID of a simulation cycle and the total simulated cycles, for each epoch snapshot. We can extend the mapper to provide additional stats key and value pairs as needed. In our implementation, the routine for performing epoch-based MARSSx86 simulation is written in C/C++, meaning that it can be accessed by the mapper using the JNI. Third, a *reduce* method, i.e., the reducer, takes a stats key and list of associated values as inputs and accumulates the list of values to obtain a final result for the key. Finally, the results, i.e., a stats key and merged values, are written to an output file on the HDFS. We confirm that we are able to evaluate microarchitectures instantly using EPSim-C for epoch-based parallel simulation on a cloud computing platform.

4. Evaluation

In this section, we describe our experimental setup in detail and evaluate the performance of our proposed simulation architecture through a variety of experiments.

4.1. Experimental Setup

Table 1 lists the experimental parameters for the Hadoop-based cloud computing system that we employed to evaluate EPSim-C for the epoch-based parallel simulation. The Hadoop system was constructed by utilizing hardware computing resources from Amazon EC2 of AWS [8,13]. We also adopted Cloudera's open-source Apache Hadoop distribution, i.e., CDH, to easily manipulate the Apache Hadoop framework, including Hadoop YARN MR2, HDFS, etc. [37–39].

Table 1. Hadoop-based cloud computing system for our experiment.

Feature	Description
Cloud computing platform	Amazon EC2 of AWS
Hadoop platform	Cloudera Manager, CDH
Hadoop framework	Hadoop YARN MR2, HDFS
Number of NameNodes	1 NameNode with 4 cores
Number of DataNodes	16 DataNodes with 2, 4, 8, 16 cores per node
Network bandwidth	450 Mbps with 2 cores per node 750 Mbps with 4 cores per node 1000 Mbps with 8 cores per node 2000 Mbps with 16 cores per node

In detail, the Hadoop system consists of a single NameNode and 16 DataNodes. Briefly, the NameNode indicates a master node in the Apache Hadoop HDFS architecture that maintains and manipulates the data blocks being stored within the DataNodes, i.e., slave nodes in the HDFS [46]. The NameNode provides a unified HDFS namespace to clients for regulating their access to data files located on the distributed file system. The NameNode also manages file system operations, such as opening, closing, and renaming files and directories, and determines the mapping of split data blocks from data files onto DataNodes. Internally, the DataNodes store the data blocks in their attached storage redundantly and are responsible for serving read and write requests from clients. DataNodes also perform the *map* and *reduce* procedures for Hadoop applications using their data blocks. We designated 16 *reduce* tasks to be created and executed when the number of completed *map* tasks exceeded 80% of the total tasks. For performance evaluation, we configured each DataNode to utilize 2, 4, 8, and 16 cores of an Intel Xeon E5-2686 v4. Thus, the experiments were performed using

a large number of cores, i.e., 32, 64, 128, and 256 because we utilized 16 DataNodes. Additionally, because we derived *m*4 instances from AWS EC2, we were able to adopt different network bandwidths of 450, 750, 1000, and 2000 Mbps when using 32, 64, 128, and 256 cores, respectively.

We obtained performance results by performing microarchitecture simulation of 1 billion instructions for each of the SPEC CPU2006 benchmarks. The target machine for simulation was specified as a default out-of-order core of MARSSx86 with 4 GB of main memory. In order to compare the performance results of the cloud implementation to conventional MARSSx86, the performance of MARSSx86 was measured using the *m*4 instance from AWS EC2, which utilizes two cores of an Intel Xeon E5-2686 v4.

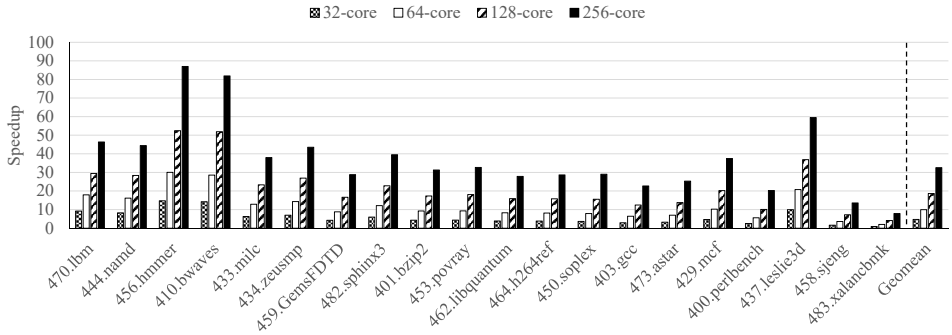
4.2. Performance Results

In this section, we evaluate the performance enhancements of our proposed EPSim-C simulator sequentially in terms of speedup, task execution time, task management overhead, network overhead, and load balance.

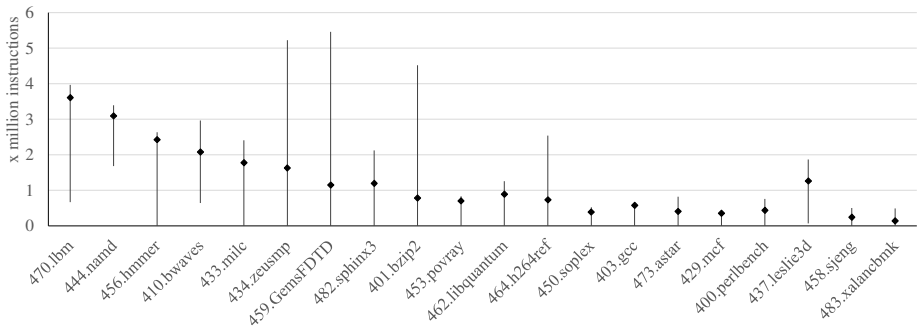
First, of all, Figure 9a presents the speedups obtained by EPSim-C for epoch-based parallel simulation on 32, 64, 128, and 256 cores compared to the conventional MARSSx86 simulator. We obtained an average speedup of 4.7× on 32 cores, 9.9× on 64 cores, 18.6× on 128 cores, and 32.6× on 256 cores. Thus, we confirmed that EPSim-C achieves outstanding performance and scalability by thoroughly utilizing its computing resources. When the number of cores is increased from 32 to 64, the speedup more than doubles because of the increased network bandwidth, i.e., 450 to 750 Mbps. Additionally, we achieved a maximum speedup of 87.0× for the 456.hmmer benchmark and a minimum speedup of 7.8× for the 483.xalancbmk benchmark when using 256 cores. To investigate the causes of these performance variations, we analyzed four different factors that contribute the overall performance of parallel and distributed computing: task execution time, task management overhead, network overhead, and load imbalance [47,48]. Task execution time represents the time spent to perform a given task. Task management overhead is an overhead for managing a large number of tasks, including assigning tasks to each computing resource, and checking completed tasks. Network overhead means the delay time due to the actual network, and load imbalance refers to the distribution of tasks across multiple nodes.

Second, in our experiments, task execution time was highly dependent on the average length of epochs, i.e., the number of instructions to be simulated. Figure 9b depicts the average, maximum, and minimum lengths of epochs in each benchmark. We obtained significantly higher speedups of 87.0× and 81.9× using 256 cores for the 456.hmmer and 410.bwaves benchmarks, respectively because, they have longer average lengths of epochs, i.e., long task execution times. In contrast, we achieved relatively low speedups of 13.6× and 7.8× using 256 cores for the 458.sjeng and 483.xalancbmk benchmarks, respectively because, their average epoch lengths were relatively short. Although task execution time was dominant in the performance improvements for most benchmarks, some benchmarks were influenced by the other three factors, which will be discussed further. For example, both the 470.lbm and 444.namd benchmarks have longer average epoch lengths than the 456.hmmer benchmark, but their speedups were smaller.

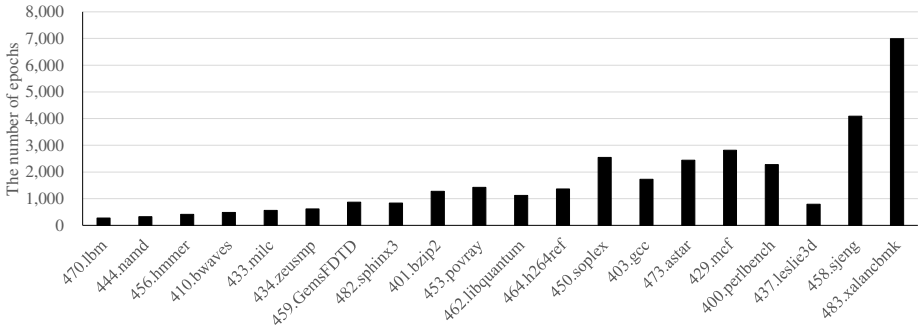
Third, Figure 9c presents the numbers of epochs that are related to task management overhead. In EPSim-C, the number of tasks is equivalent to the number of epochs for each benchmark. Because task management overhead increases proportionally to the number of tasks, we can enhance the performance to a greater degree as the number of epochs decreases. For the 458.sjeng and 483.xalancbmk benchmarks, we found that the performance of EPSim-C was significantly constrained by the large number of epochs, especially when the number of epochs was greater than 4000.



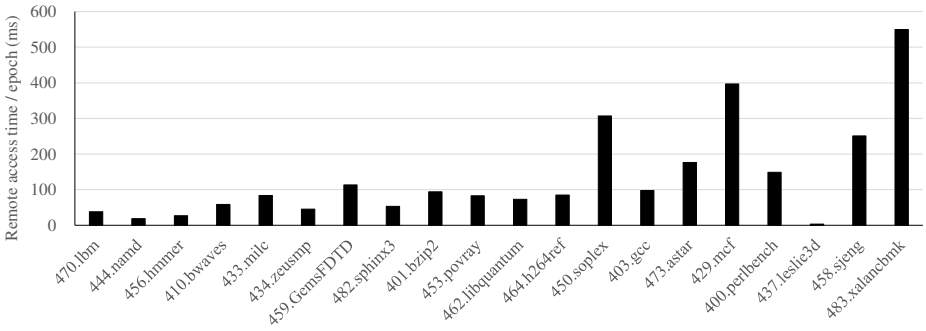
(**a**) Speedups with different numbers of cores.



(**b**) The average, maximum, and minimum lengths of epochs. The top and bottom of each vertical line represent the maximum and minimum lengths of the epochs (the number of instructions to be simulated per epoch) in a benchmark, respectively. The dot indicates the average length of the epochs in a benchmark.



(**c**) The number of epochs.



(**d**) The average time per epoch for remote access through the network.

Figure 9. Results of Hadoop-based microarchitecture simulation on the SPEC CPU2006 benchmarks.

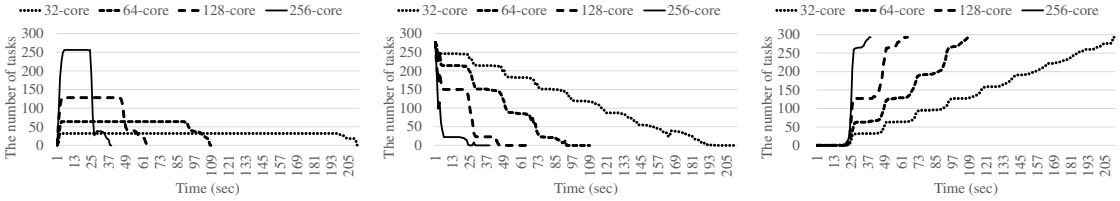
Fourth, the average remote access time per epoch, which was computed by dividing total networking time by the numbers of epochs in Figure 9c, is shown in Figure 9d. This value indicates the network overhead for each benchmark. For most benchmarks, network overhead was well handled by Hadoop framework, but it significantly limited the performance improvements for some benchmarks. For example, despite the 437.leslie3d benchmark having a similar length and number of epochs to the 434.zeusmp and 459.GemsFDTD benchmarks, we obtained a much higher speedup for the 437.leslie3d benchmark than for the others. This is because the remote access time per epoch for the 437.leslie3d benchmark was considerably smaller on average (3 ms). In contrast, the speedup obtained for the 483.xalancbmk benchmark was extremely limited because its remote access time per epoch was 550 ms and it had a large number of epochs.

Finally, load balancing is characterized by the numbers of allocated tasks, pending tasks, and completed tasks. For most of benchmarks, we confirmed that EPSim-C could maintain good load balancing when using different numbers of cores, such as 32, 64, 128, and 256, on the cloud computing platform. Therefore, in this paper, we only discuss the three benchmarks that had distinguishing characteristics in terms of load balancing. Figures 10–12 present the load balance over time for the 470.lbm, 410.bwaves, and 401.bzip2 benchmarks, respectively. The number of tasks in Figures 10–12 are the sum of the numbers of *map* tasks and *reduce* tasks. Because *reduce* tasks are created and allocated during active execution, the number of pending tasks at the start of the process is not equal to the number of completed of tasks at the end of the process.

Figure 10a shows that the number of allocated tasks for the 470.lbm benchmark was measured as the maximum number of available cores in the first round between 0 and 25 s. However, it only used 37 cores in the second round despite there being 256 cores available. As shown in Figure 10b, the number of pending tasks decreases from 277, i.e., the total number of *map* tasks for the 470.lbm benchmark, to 21 after 256 tasks are allocated and executed at the start of the first round using all 256 cores. Additionally, Figure 10c indicates that the number of completed tasks increases rapidly immediately following the first round, but then grows more gradually to 293, including 16 *reduce* tasks. Therefore, for the 470.lbm benchmark, we were able to confirm that most cores were not occupied after the first round, i.e., underutilization of computing resources. Because the 444.namd and 433.milc benchmarks have similar tendencies to the 470.lbm benchmark, their performance improvements were significantly limited compared to the other benchmarks because of load imbalance, despite having longer task execution time and lower task management overhead. In other words, since the lengths of each epoch are variously distributed, and in particular, the difference between the maximum length of epochs and the minimum length of epochs is large as shown in Figure 9, causing the load imbalance. The load imbalance is relative and becomes less if we simulate with the same number of epochs as other benchmarks to be mentioned.

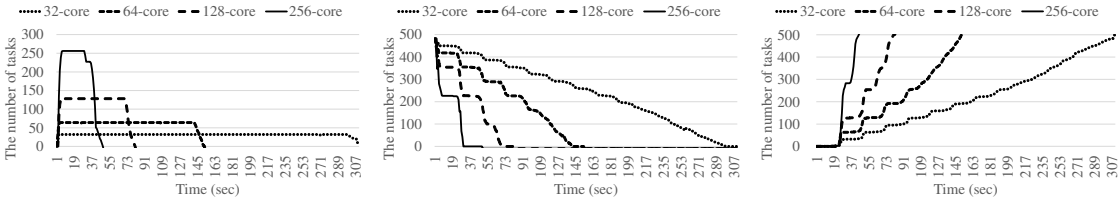
Compared to the 470.lbm benchmark, we obtained slightly better load balancing for the 410.bwaves benchmark, as shown in Figure 11. This is because the lengths of epochs are well distributed, the average is located in intermediate, and the difference between maximum and minimum is not large, as shown in Figure 9. This leads to a more significant speedup for the 410.bwaves benchmark than for the 470.lbm benchmark, although the average length of epochs in the former was shorter than that in the latter. Figure 11a reveals that the numbers of allocated tasks for the 410.bwaves benchmark were almost identical to the maximum numbers of available cores, i.e., 32, 64, 128, and 256, for the majority of execution time. As shown in Figure 11b, when using 256 cores, the number of pending tasks diminishes from 481, i.e., the total number of *map* tasks in the 410.bwaves benchmark, to 225 because 256 of the tasks begin execution at the start of the first round. In contrast to the 470.lbm benchmark, because the remaining tasks, including 225 *map* tasks and 16 *reduce* tasks, utilize most of the available cores, there is no notable underutilization of computing resources. From these experiments, we also found that the 456.hmmer benchmark is very similar to the 410.bwaves benchmark in terms of load balance.

In addition, Figure 12 presents the load balance in the 401.bzip2 benchmark, which represents most of the benchmarks not mentioned previously. In these cases, because the total number of epochs is large enough versus the number of available cores, the Hadoop framework performs load balancing, meaning that there is no significant performance degradation because of load imbalance.



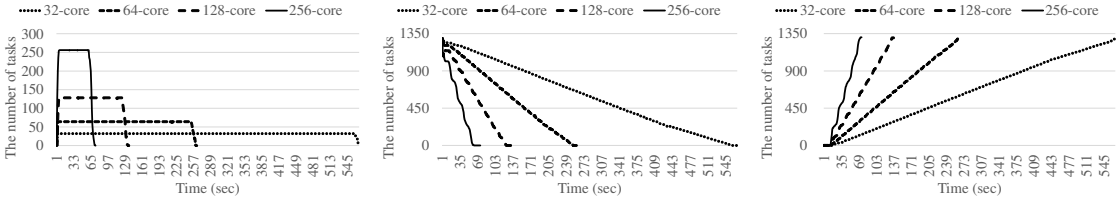
(**a**) Allocated. (**b**) Pending. (**c**) Completed.

Figure 10. The number of allocated, pending, and completed tasks over time for the 470.lbm benchmark.



(**a**) Allocated. (**b**) Pending. (**c**) Completed.

Figure 11. The number of allocated, pending, and completed tasks over time for the 410.bwaves benchmark.



(**a**) Allocated. (**b**) Pending. (**c**) Completed.

Figure 12. The number of allocated, pending, and completed tasks over time for the 401.bzip2 benchmark.

4.3. Task Management Optimization

In the previous section, we found that task management overhead significantly degrades the performance of specific benchmarks involving a large number of tasks. To resolve this problem, we adopted a simple optimization strategy to reduce the total number of tasks, i.e., an optimized task simulates several epochs sequentially, whereas un-optimized tasks perform the simulation of only one epoch. In other words, we reduced the number of input files because a single *map* task is required to manipulate each input file in the Hadoop-based cloud computing system. For our optimization strategy, we extended each input file to contain one or more epochs such that the total number of tasks is less than four times the number of available cores, i.e., not the number of available cores, by suitably splitting input data. Because tasks in the Hadoop framework are typically assigned based on the location of data used to execute the task, the number of tasks being the same as the number of available cores may lead to load imbalance and consume significant network resources [49].

Figure 13 presents the reduction of tasks achieved by the optimization strategy when using 256 cores. The leftmost eight benchmarks and the 437.leslie3d benchmark did not require optimization because the number of tasks was sufficiently small without optimization, i.e., the number of epochs

was less than 1024. For the other benchmarks, we decreased the number of tasks by 66.4% on average by applying our optimization strategy. The number of tasks for the 483.xalancbmk benchmark was significantly reduced from 6991 to 542 by using our optimization strategy, i.e., 92% of tasks were eliminated.

Therefore, we obtained considerable performance improvements by using our task management optimization strategy when utilizing 256 cores, as shown in Figure 14. [Reviewer1 - Point2] In particular, we achieved more significant performance improvements for the benchmarks with large numbers of epochs in Figure 9c. The speedup for the 483.xalancbmk benchmark increased from 7.8× to 33.6× and that for the 458.sjeng benchmark increased from 13.6× to 37.7×. By employing our optimization strategy for 256 cores, we improved the performance of proposed epoch-based simulation from 32.6× to 46.1× on average compared to the conventional MARSSx86 simulator.

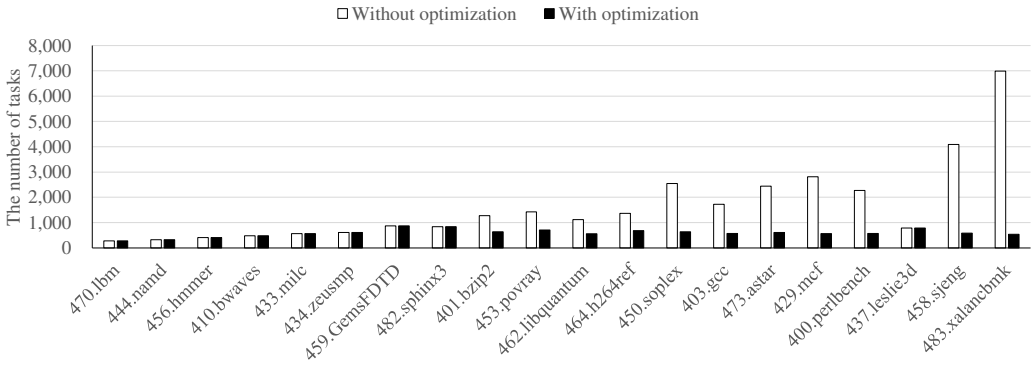


Figure 13. Reduction in the number of tasks using task management optimization strategy on 256 cores.

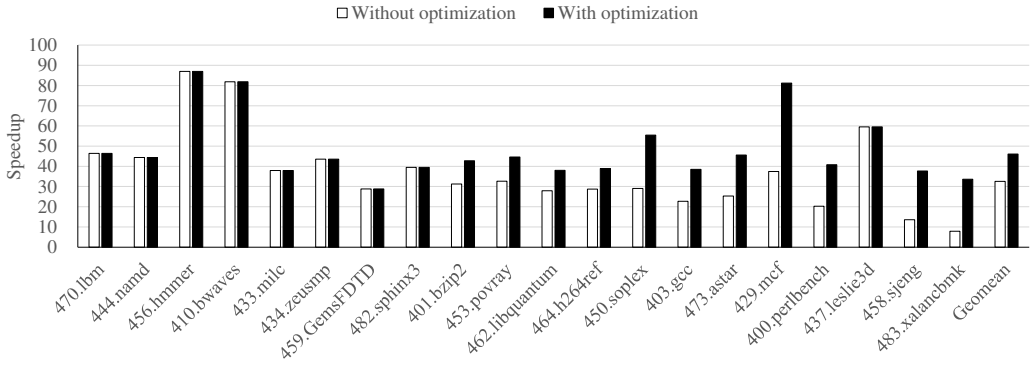


Figure 14. Performance improvements via task management optimization when using 256 cores.

5. Related Works and Discussion

In this section, we compare and discuss previous studies on reducing simulation time for microarchitecture simulations. Table 2 summarizes the characteristics of related schemes and our work in terms of parallelism, performance scalability, and usefulness of cloud computing.

Table 2. Comparison of the related schemes with our EPSim. FM and TM denote functional and timing models, respectively.

Category	Scheme	Parallelism	Scalability	Cloud Computing
High-abstraction-level modeling	Sniper [28] IntervalSim [29]	No	No	No
Sampling-based	SimPoints [27]	No	No	No
Hardware-assisted	FAST [26]	FM and TM	Bad	No
Parallelizing functional and timing simulations	Transformer [31,44]	FM and TM	Bad	No
Multicore and multithreading simulation	P-Mamabo [32] SlackSim [34]	Target core	Limited	No
Multicore and cloud computing simulation	EPSim [33]	Epoch (Massive)	Good	Yes

5.1. Multicore and Multithreading Simulation

Many previous studies accelerated cycle-accurate simulation by adopting multithreading on multicore platforms. P-Mambo [32] is a multithreaded implementation of IBM's full-system simulator, called Mambo [50], for modeling PowerPC systems. SlackSim [34] assigns each core of a target chip multiprocessor (CMP) to one thread and then spreads threads across several machines for parallelization. In order to reduce synchronization overhead, it parallelizes execution with the inclusion of simulation time slack. Graphite [51] is a distributed parallel simulator infrastructure that simulates many-core processors containing dozens to thousands of cores. Each core of the target architecture is assigned to one thread and the threads are spread across multiple machines for parallel execution. BigSim [52] serves as a performance prediction model for target machines and uses MPI to exploit large-scale parallelism. These simulators assign target cores to multicore platforms, sometimes across several machines. Therefore, synchronization overhead becomes significant when the number of cores used increases, meaning that they do not have sufficient parallelism to execute simulations in parallel using a cloud computing platform.

There have also been studies where a simulator itself is accelerated on a multicore platform through the analysis of internal simulator properties. MARSSx86 [23] utilizes a tightly coupled design between functional and timing models, as described in Section 2. However, the sequential execution of functional models, interactions, and timing models results in significant overhead for simulation. Transformer [31,44] separates sequential executions into functional models and timing models, then executes them in parallel while postponing interaction as long as possible to prevent interaction latency. Additionally, EPSim [33], which we used in our framework, eliminates interactions and separates the functional and timing models into epochs. Thus, high performance scalability can be achieved on multicore platforms. We adopt EPSim because of the massive parallelism available through epochs and exploit them on a cloud computing platform to accelerate microarchitecture simulation by executing epochs in parallel.

5.2. Sampling-Based Simulation

Instead of simulating an entire application, sampling-based simulation conducts cycle-accurate simulation of only a few simulation points within an application. SimPoints [27] selects simulation points that identify the various phases of a running application using basic block vectors (BBVs) that represent the frequencies of executed sequences of basic blocks. For similar BBVs, SimPoints assumes that their performance is similar and only simulates some representative BBVs. SMARTS [53] uses a systematic sampling technique to select simulation points periodically by tuning several

sampling variables, including size, mean, coefficient of variation, confidence level, confidence interval, and systematic-sampling interval. Although the speedups achieved by these simulators are significant, their results are less accurate because they only simulate a few simulation points. In addition, they do not exploit any parallelism in execution.

5.3. High-Abstraction-Level Modeling

High-abstraction-level modeling simulators coordinate the details of cycle-accurate simulators and find their midpoint, i.e., one-IPC simulators that only simulate core IPCs with a tradeoff between accuracy and speed. Sniper [28] and IntervalSim [29] both use interval simulation to balance this trade-off. In order to define the intervals, these simulators use miss events (branch misprediction or cache and TLB misses) and adopt an analytical model for events to predict core performance without any detailed simulation of the core's pipeline stages. EPSim, which is used in our framework, simulates each core's pipeline stages in detail via epoch-by-epoch simulation. Thus, we are able to obtain various accurate performance metrics.

5.4. Hardware Acceleration

There have been studies on adopting dedicated hardware resources for architectural simulators. ProtoFlex [54,55] is a hybrid functional simulator that simulates common operations, such as ALU instructions, on FPGAs, while complex behaviors, such as disk I/Os, are modeled in software. FAST [26] simulates functional models using software and timing models using special hardware in parallel. Although these studies have achieved remarkable performance, they require specific hardware, i.e., FPGAs. They also require significant effort to realize their hardware implementations. However, specialized hardware resources and complex implementations are not required for EPSim-C.

6. Conclusions

Although cloud computing has grown dramatically and is useful for a wide variety of data-intensive applications, studies on accelerating microarchitecture simulation using cloud computing have not been performed. In this study, we adapted EPSim to a cloud computing platform to exploit massive parallelism. We concentrated on utilizing the massive parallelism of the existing epoch-based parallel simulator, i.e., EPSim, and found that the MapReduce process closely resembles the process used by our existing simulator. Therefore, we mapped epoch-based execution to MapReduce on a Hadoop-based system.

We verified EPSim-C using a Hadoop-based cloud computing system with one NameNode and 16 DataNodes on Amazon EC2 and achieved an average speedup of 4.7× on 32 cores, 9.9× on 64 cores, 18.6× on 128 cores, and 32.6× on 256 cores, while enhancing scalability, which allows us to model state-of-the-art computing platforms for real workloads. In order to minimize the performance degradation caused by task management overhead, we applied a simple optimization technique to reduce the number of tasks by merging multiple epochs into one task. With this optimization, we obtained an average speedup of 46.1x on 256 cores. In particular, the speedup for the 483.xalancbmk benchmark, which has a large number of epochs, was improved by 4.3× through optimization, i.e., from 7.8× to 33.6×, using 256 cores. To the best of our knowledge, the proposed framework is the first approach that accelerates microarchitecture simulation using cloud computing, which allows us to model the state-of-the-art computing platforms for real workloads while providing performance scalability.

In future work, we are going to analyze the performance when the working data set to be simulated increases significantly because task management overhead, network overhead, and the characteristics of load imbalance are expected to change. We also need to study a variety of performance and security trends in detail when using more computing resources in a cloud computing environment [56,57].

Author Contributions: All authors contributed to this work. Investigation, M.K.; Methodology, Y.H.; Supervision, Y.H.; Writing—original draft, M.K.; Writing—review & editing, S.W.K. and Y.H.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1C1C1008789).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, X.; Qiu, J. *Cloud Computing for Data-Intensive Applications*; Springer Publishing Company Incorporated: New York, NY, USA, 2014.
2. Yang, X.; Wallom, D.; Waddington, S.; Wang, J.; Shaon, A.; Matthews, B.; Wilson, M.; Guo, Y.; Guo, L.; Blower, J.D.; et al. Cloud Computing in e-Science: Research Challenges and Opportunities. *J. Supercomput.* **2014**, *70*, 408–464. [CrossRef]
3. Casanova, H.; Giersch, A.; Legrand, A.; Quinson, M.; Suter, F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel Distrib. Comput.* **2014**, *74*, 2899–2917. [CrossRef]
4. Bian, Z.; Wang, K.; Wang, Z.; Munce, G.; Cremer, I.; Zhou, W.; Chen, Q.; Xu, G. Simulating Big Data Clusters for System Planning, Evaluation, and Optimization. In Proceedings of the 2014 43rd International Conference on Parallel Processing, Minneapolis, MN, USA, 9–12 September 2014; pp. 391–400. [CrossRef]
5. Zhao, B.; Zhong, J.; He, B.; Luo, Q.; Fang, W.; Govindaraju, N.K. GPU-Accelerated Cloud Computing for Data-Intensive Applications. In *Cloud Computing for Data-Intensive Applications*; Li, X., Qiu, J., Eds.; Springer: New York, NY, USA, 2014; pp. 105–129. [CrossRef]
6. Li, B.; Mazur, E.; Diao, Y.; McGregor, A.; Shenoy, P. A Platform for Scalable One-pass Analytics Using MapReduce. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, Athens, Greece, 12–16 June 2011; pp. 985–996. [CrossRef]
7. Yoon, D.H.; Kang, S.K.; Kim, M.; Han, Y. Exploiting Coarse-Grained Parallelism Using Cloud Computing in Massive Power Flow Computation. *Energies* **2018**, *11*, 2268. [CrossRef]
8. Amazon. Amazon Elastic Compute Cloud—Cloud Server & Hosting. 2017. Available online: https://aws.amazon.com/ec2 (accessed on 30 November 2017).
9. Microsoft. Microsoft Azure Cloud Computing Platform & Services. 2017. Available online: https://azure.microsoft.com (accessed on 30 November 2017).
10. IBM. IBM Cloud. 2017. Available online: https://www.ibm.com/cloud (accessed on 30 November 2017).
11. Oracle. Oracle Cloud. 2017. Available online: https://www.oracle.com/cloud (accessed on 30 November 2017).
12. Hazelhurst, S. Scientific Computing Using Virtual High-performance Computing: A Case Study Using the Amazon Elastic Computing Cloud. In Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology, Wilderness, South Africa, 6–8 October 2008; pp. 94–103. [CrossRef]
13. Amazon. Amazon Web Services—Cloud Computing Services. 2017. Available online: https://aws.amazon.com (accessed on 30 November 2017).
14. Sodani, A.; Gramunt, R.; Corbal, J.; Kim, H.S.; Vinod, K.; Chinthamani, S.; Hutsell, S.; Agarwal, R.; Liu, Y.C. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro* **2016**, *36*, 34–46. [CrossRef]
15. Jeffers, J.; Reinders, J. *Intel Xeon Phi Coprocessor High Performance Programming*, 1st ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2013.
16. Rahman, R. *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*, 1st ed.; Apress: Berkely, CA, USA, 2013.
17. Nvidia. NVIDIA Tesla V100. 2017. Available online: https://www.nvidia.com/en-us/data-center/tesla-v100/ (accessed on 26 July 2017).
18. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv* **2016**, arXiv:1603.04467.
19. Subramaniyan, A.; Das, R. Parallel Automata Processor. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 600–612. [CrossRef]

20. Micron. Micron Automata Processing. 2017. Available online: http://www.micronautomata.com (accessed on 20 July 2017).
21. Wenisch, T.F.; Wunderlich, R.E.; Ferdman, M.; Ailamaki, A.; Falsafi, B.; Hoe, J.C. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro* **2006**, *26*, 18–31. [CrossRef]
22. Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S.K.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D.R.; Krishna, T.; Sardashti, S.; et al. The Gem5 Simulator. *SIGARCH Comput. Archit. News* **2011**, *39*, 1–7. [CrossRef]
23. Patel, A.; Afram, F.; Chen, S.; Ghose, K. MARSS: A full system simulator for multicore x86 CPUs. In Proceedings of the 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), San Diego, CA, USA, 5–9 June 2011; pp. 1050–1055.
24. Yourst, M.T. PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. In Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems Software, San Jose, CA, USA, 25–27 April 2007; pp. 23–34. [CrossRef]
25. Lee, K.; Cho, S. Accurately Modeling Superscalar Processor Performance with Reduced Trace. *J. Parallel Distrib. Comput.* **2013**, *73*, 509–521. [CrossRef]
26. Chiou, D.; Sunwoo, D.; Kim, J.; Patil, N.A.; Reinhart, W.; Johnson, D.E.; Keefe, J.; Angepat, H. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), Chicago, IL, USA, 1–5 December 2007; pp. 249–261. [CrossRef]
27. Sherwood, T.; Perelman, E.; Hamerly, G.; Calder, B. Automatically Characterizing Large Scale Program Behavior. In Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, 5–9 October 2002; pp. 45–57. [CrossRef]
28. Carlson, T.E.; Heirmant, W.; Eeckhout, L. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Seattle, WA, USA, 12–18 November 2011; pp. 1–12. [CrossRef]
29. Genbrugge, D.; Eyerman, S.; Eeckhout, L. Interval simulation: Raising the level of abstraction in architectural simulation. In Proceedings of the HPCA-16—2010 The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore, India, 9–14 January 2010; pp. 1–12. [CrossRef]
30. Han, M.; Kim, S.W.; Kim, M.; Han, Y. P-DRAMSim2: Exploiting Thread-level Parallelism in DRAMSim2. *IEICE Electron. Express* **2017**. [CrossRef]
31. Fang, Z.; Min, Q.; Zhou, K.; Lu, Y.; Hu, Y.; Zhang, W.; Chen, H.; Li, J.; Zang, B. Transformer: A functional-driven cycle-accurate multicore simulator. In Proceedings of the 2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 3–7 June 2012; pp. 106–114.
32. Wang, K.; Zhang, Y.; Wang, H.; Shen, X. Parallelization of IBM Mambo System Simulator in Functional Modes. *SIGOPS Oper. Syst. Rev.* **2008**, *42*, 71–76. [CrossRef]
33. Kim, M.; Park, C.; Han, M.; Han, Y.; Kim, S.W. Epsim: A Scalable and Parallel MARSSx86 Simulator With Exploiting Epoch-Based Execution. *IEEE Access* **2019**, *7*, 4782–4794. [CrossRef]
34. Chen, J.; Annavaram, M.; Dubois, M. SlackSim: A Platform for Parallel Simulations of CMPs on CMPs. *SIGARCH Comput. Archit. News* **2009**, *37*, 20–29. [CrossRef]
35. Kainaga, M.; Yamada, K.; Inayoshi, H. Analysis of SPEC benchmark programs. In Proceedings of the Eighth TRON Project Symposium, Tokyo, Japan, 21–22 November 1991; pp. 208–215. [CrossRef]
36. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
37. The Apache Software Foundation. Welcome to Apache Hadoop. 2017. Available online: http://hadoop.apache.org (accessed on 3 August 2017).
38. Cloudera. Machine Learning, Analytics, Cloud—Cloudera. 2017. Available online: https://www.cloudera.com (accessed on 3 August 2017).
39. Vavilapalli, V.K.; Murthy, A.C.; Douglas, C.; Agarwal, S.; Konar, M.; Evans, R.; Graves, T.; Lowe, J.; Shah, H.; Seth, S.; et al. Apache Hadoop YARN: Yet Another Resource Negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, CA, USA, 1–3 October 2013; p. 5. [CrossRef]
40. Thusoo, A.; Sarma, J.S.; Jain, N.; Shao, Z.; Chakka, P.; Anthony, S.; Liu, H.; Wyckoff, P.; Murthy, R. Hive: A Warehousing Solution over a Map-reduce Framework. *Proc. VLDB Endow.* **2009**, *2*, 1626–1629. [CrossRef]

41. Rosenblum, M.; Bugnion, E.; Devine, S.; Herrod, S.A. Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM Trans. Model. Comput. Simul. TOMACS* **1997**, *7*, 78–103. [CrossRef]
42. haeng Kang, S.; Yoo, D.; Ha, S. TQSIM: A fast cycle-approximate processor simulator based on QEMU. *J. Syst. Archit.* **2016**, *66*, 33–47. [CrossRef]
43. Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of the Annual Conference on USENIX Annual Technical Conference, Anaheim, CA, USA, 10–15 April 2005; p. 41.
44. Zhang, W.; Wang, H.; Lu, Y.; Chen, H.; Zhao, W. A Loosely-Coupled Full-System Multicore Simulation Framework. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 1566–1578. [CrossRef]
45. Iosup, A.; Ostermann, S.; Yigitbasi, M.N.; Prodan, R.; Fahringer, T.; Epema, D. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 931–945. [CrossRef]
46. Le, H.H.; Hikida, S.; Yokota, H. NameNode and DataNode Coupling for a Power-Proportional Hadoop Distributed File System. In *Database Systems for Advanced Applications*; Meng, W., Feng, L., Bressan, S., Winiwarter, W., Song, W., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 99–107.
47. Yin, J.; Wang, J.; Zhou, J.; Lukasiewicz, T.; Huang, D.; Zhang, J. Opass: Analysis and Optimization of Parallel Data Access on Distributed File Systems. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium, Hyderabad, India, 25–29 May 2015; pp. 623–632. [CrossRef]
48. Jackson, K.R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H.J.; Wright, N.J. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, USA, 30 November–3 December 2010; pp. 159–168. [CrossRef]
49. Lin, C.; Lin, Y. A Load-Balancing Algorithm for Hadoop Distributed File System. In Proceedings of the 2015 18th International Conference on Network-Based Information Systems, Taipei, Taiwan, 2–4 September 2015; pp. 173–179. [CrossRef]
50. Bohrer, P.; Peterson, J.; Elnozahy, M.; Rajamony, R.; Gheith, A.; Rockhold, R.; Lefurgy, C.; Shafi, H.; Nakra, T.; Simpson, R.; et al. Mambo: A Full System Simulator for the PowerPC Architecture. *ACM SIGMETRICS Perform. Eval. Rev.* **2004**, *31*, 8–12. [CrossRef]
51. Miller, J.E.; Kasture, H.; Kurian, G.; Gruenwald, C.; Beckmann, N.; Celio, C.; Eastep, J.; Agarwal, A. Graphite: A distributed parallel simulator for multicores. In Proceedings of the HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore, India, 9–14 January 2010; pp. 1–12. [CrossRef]
52. Zheng, G.; Kakulapati, G.; Kale, L.V. BigSim: A parallel simulator for performance prediction of extremely large parallel machines. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, NM, USA, 26–30 April 2004; p. 78. [CrossRef]
53. Wunderlich, R.E.; Wenisch, T.F.; Falsafi, B.; Hoe, J.C. SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. In Proceedings of the 30th Annual International Symposium on Computer Architecture, San Diego, CA, USA, 9–11 June 2003; pp. 84–97. [CrossRef]
54. Chung, E.S.; Nurvitadhi, E.; Hoe, J.C.; Falsafi, B.; Mai, K. PROToFLEX: FPGA-accelerated Hybrid Functional Simulator. In Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA, 26–30 March 2007; pp. 1–6. [CrossRef]
55. Chung, E.S.; Papamichael, M.K.; Nurvitadhi, E.; Hoe, J.C.; Mai, K.; Falsafi, B. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. Reconfig. Technol. Syst.* **2009**, *2*, 15. [CrossRef]
56. Shang, Y. Resilient Multiscale Coordination Control against Adversarial Nodes. *Energies* **2018**, *11*, 1844. [CrossRef]
57. Takabi, H.; Joshi, J.B.D.; Ahn, G. Security and Privacy Challenges in Cloud Computing Environments. *IEEE Secur. Priv.* **2010**, *8*, 24–31. [CrossRef]