Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 327 (2016) 93–107

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Model-based Dynamic Control of Speculative Forays in Parallel Computation [4]

Kalyan S. Perumalla[1] Mohammed M. Olama[2]
Srikanth B. Yoginath[3]

Computational Sciences and Engineering
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Abstract

In simulations running in parallel, the processors would have to synchronize with other processors to maintain correct global order of computations. This can be done either by blocking computation until correct order is guaranteed, or by *speculatively* proceeding with the best guess (based on local information) and later correcting errors if/as necessary. Since the gainful lengths of speculative forays depend on the dynamics of the application software and hardware at runtime, an online control system is necessary to dynamically choose and/or switch between the blocking and speculative strategies. In this paper, we formulate the reversible speculative computing in large-scale parallel computing as a dynamic linear feedback control (optimization) system model and evaluate its performance in terms of time and cost savings as compared to the traditional (forward) computing. We illustrate with an exact analogy in the form of vehicular travel under dynamic, delayed route information. The objective is to assist in making the optimal decision on what computational approach is to be chosen, by predicting the amount of time and cost savings (or losing) under different environments represented by different parameters and probability distribution functions. We consider the cases of Gaussian, exponential and log-normal distribution functions. The control system is intended for incorporating into speculative parallel applications such as optimistic parallel discrete event simulations to decide at runtime when and to what extent speculative execution can be performed gainfully.

Keywords: Reversible execution, Parallel Computing, Speculative Execution, Model-based Execution

[1] Email: `perumallaks@ornl.gov`

[2] Email: `olamahussemm@ornl.gov`

[3] Email: `yoginathsb@ornl.gov`

[4] This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Dept. of Energy. Accordingly, the U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

http://dx.doi.org/10.1016/j.entcs.2016.09.025
1571-0661/© 2016 The Author(s). Published by Elsevier B.V.

1 Introduction

1.1 Speculative Reversible Parallel Computation

Reversible computing is a relaxation of conventional forward-only computing [1]. In reversible computing, execution is designed to make it possible to go forward as well as backward: the application running on any processor is designed to change the direction of execution on demand. Such a reversible execution framework is useful in many contexts, such as database transactions and parallel discrete event simulations. In these applications, inter-processor synchronization forms a major cost which arises from the need of each processor to get information often from other processors on the next local trajectory to follow. One way to avoid blocking and wasting time waiting for information from other processors is to proceed with a best estimate based on local information, and then rely on reversible computation to retrace any incorrect portions of the trajectory. These portions of local execution without waiting for perfect global knowledge are called speculative forays. The higher the fraction of correctness of the speculative foray, the greater the gain; the lower the fraction, the lower the gain (or, worse, the more negative the gain). It is very hard to ascertain the best strategy ahead of time; in fact, it is preferable to have a runtime controller that can dynamically make decisions on when and to what extent the speculative foray should be allowed. In other words, an online control system is needed so that the speculative forays are dynamically controlled.

Here, we investigate an approach based on a model-based control system design that is agnostic to the specific application. We explore different stochastic distributions for the main runtime dynamics, and analyze their efficacies in synthetic experiments. The overall problem is defined in terms of two competing objectives: time to completion and total cost for completion. We explore the space with representative (normalized) parameter values in order to gain an initial understanding of the efficacy of our approach. Our work presented here differs from previous analyses in the literature on rollback-based parallel computation. Previous works focused on analyzing the applications and application classes as a whole in determining apriori the average, best and worst cases of blocking and speculative strategies [5,6,7,8,9]. In contrast, we focus on the design problem of online control to dynamically choose (and potentially switch) between blocked and speculative execution, and also attempt to maintain the lengths of speculative forays as an option of runtime control.

1.2 Analogy

Consider a setting in which a driver is driving a vehicle to some final destination **F** to which the path is incrementally obtained. Suppose the driver is currently at some milestone point **A**. Further suppose that, after reaching **A** some processing time $R(t)$ is needed to determine the route to the next milestone point **B**. So, the vehicle is stationary while the driver waits for the next milestone **B** to be determined. To save time, the driver may guess the next route and start driving. By the time the next milestone **B** is determined, the vehicle may have already gone ahead from **A** to another milestone **C**. It has to now drive back from **C** to the point **D** at which the route deviated from the intended route to **B**. Thus, from **C**, it drives back to **D** and then proceeds to **B**. This process repeats at **B** on the way to the final destination **F** as illustrated in Figure 1.

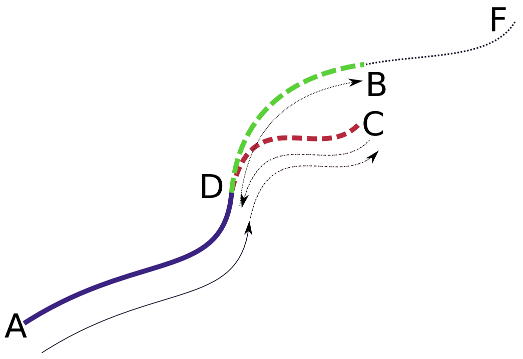


Fig. 1. Speculatively traversed path from **A** to **B** achieved via an aggresive traversal that proceeds to **C** which is later discovered to be correct only until **D** and so retraced from **C** to **D** on the way to **B**, and eventually to **F**

Let $\overset{\leadsto}{ab}$ denote the distance traveled from point A to point B using speculative travel. Let $\overrightarrow{ab}$ be the shortest (non-speculative) distance for direct travel between points A and B.

By driving ahead, some time is potentially saved. The actual savings depends on how much of the path is common between the routes **A-C** and **A-B**, that is, how long **A-D** is relative to **A-B**. Let the driving time for **A-D** be $t_{\overrightarrow{ad}}$, and for **C-D** be $t_{\overrightarrow{cd}}$. Then, the net time gained is $t_{\overrightarrow{ad}} - t_{\overrightarrow{cd}}$, illustrated in Figure 2.

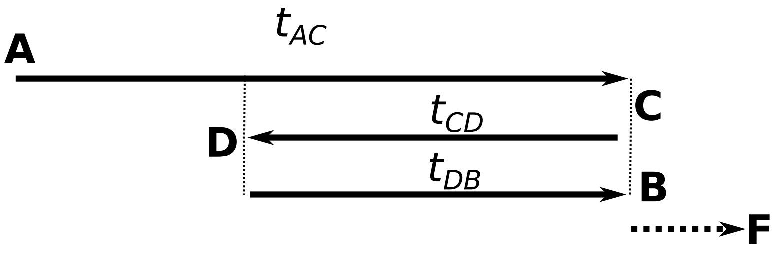


Fig. 2. Net cost of speculative foray equals total time for forward plus reverse paths

Note that the gain is *not* as large as $t_{\overset{\leadsto}{ac}} - t_{\overrightarrow{cd}}$. This is because the reversal **C-D** is undertaken *after* the correct path is known. So, if we had not driven ahead, we would have reached **D** in time $t_{\overrightarrow{ad}}$, whereas, due to driving ahead, we have to drive **C-D** which takes a time penalty of $t_{\overrightarrow{cd}}$. Thus, the net gain in time is only $t_{\overrightarrow{ad}} - t_{\overrightarrow{cd}}$. This also means that, depending on the actual amount of deviation from the intended path, the time savings in fact could be positive, zero, or even negative. The savings are positive when **D** is much closer to **C** than to **A**. Similarly, the savings are negative when **D** is farther from **C** than from **A**.

While there is the potential to gain (positive) savings in time, there is always an additional non-negative fuel cost incurred by driving ahead. Let $L_{\overrightarrow{pq}}$ be the fuel cost of driving the distance $\overrightarrow{pq}$ directly between points P and Q. Then, the total fuel cost $L_{\underset{ab}{\leadsto}}$ is given by

$$L_{\underset{ab}{\leadsto}} = L_{\vec{ad}} + L_{\vec{dc}} + L_{\vec{cd}} + L_{\vec{db}} = L_{\vec{ab}} + 2L_{\vec{cd}}. \tag{1}$$

Assuming the fuel cost to be proportional to the distance travelled, Equation 1 becomes

$$L_{\underset{ab}{\leadsto}} = W(\overrightarrow{ab} + 2\overrightarrow{cd}) \tag{2}$$

where W is constant. In this article, we consider $W = 1$, which represents the case where the fuel cost is normalized to be equal to distance. This cost function will be compared with the other scenario, where the driver waits to the next mile stone **B** to be determined. In this case, we consider the situation where the driver turns off the vehicle while waiting to save fuel. And once the next mile stone is determined, the driver turns on the vehicle and drives to the next mile stone **B**. In this situation, we assume there is an additional fuel cost for staring up the vehicle, denoted by S. So, the total fuel cost $L_{\underset{ab}{\leadsto}}$ for the waiting scenario is described by

$$L_{\underset{ab}{\leadsto}} = W\overrightarrow{ab} + S \tag{3}$$

The analogy of the preceding model of reversible speculative driving is exact with that of reversible computing [1] in large-scale parallel computation. The notion of awaiting the determination of route **A-B** corresponds to the true computational path that a processor needs to take in a correct computation. The speculative foray **A-C** corresponds to the path the processor may take when it avoids completely blocking its computation while awaiting information from other processors (i.e., for inter-processor synchronization and communication to complete). The common path **A-D** corresponds to the gain in computational time obtained by the parallel program as a result of its speculative foray. The fuel cost corresponds to the aggregate electrical energy consumed by the computation, both in forward and in reverse directions.

In practice, the waiting time to determine the route to the next milestone, $R(t)$, the speculative distance (or time) to drive ahead without waiting for the exact route information, $d_{\vec{ac}}$, and the distance (or time) to drive back to the intended route after the next milestone is determined, $d_{\vec{cd}}$, are all random, and therefore, can be represented as random variables with certain distributions. The objective is to determine whether to wait for instructions about the exact route information before driving ahead (*Strategy 1*) or to drive ahead by guessing the route without waiting to receive the instructions about the exact route (*Strategy 2*).

2 Problem Formulation and Assumptions

In this paper, we design a dynamic feedback controller (optimizer) to investigate the performance of the proposed reversible speculative computing and assist in making the right decision on what strategy to select (*Strategy 1* or *2*). The optimal decision is based on the amount of time and cost savings (or losing) under different environments represented by different parameters and probability distribution functions. In the next section, we represent the reversible speculative forays problem by a linear state-space model that is optimized using a dynamic feedback control formulation.

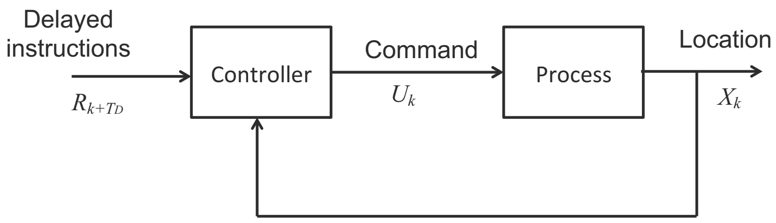


Fig. 3. The reversible speculative execution problem represented by a dynamic feedback control system

The reversible speculative computing problem can be represented by the dynamic feedback control system shown in Figure 3. The process model represents the node processing model, which can be represented in state-space formulation as

$$X_k = AX_{k-1} + BU_{k-1} \tag{4}$$

where X_k is the state at time k, U_k is the execution time duration at k, A and B are constant model parameters. The value of B represents the processor speed, and the parameters may be normalized such that $A = 1$. The value of B is positive when the computation is in forward direction (vehicle moves forward) and negative when the computational direction is reversed (vehicle moves backward).

The input to the controller is the series of randomly time-delayed reference instruction, R_{k+T_D}, at time k that provides information on the correct computational route, and T_D is the random time-delay. These instructions should arrive at the controller at time k, but it is randomly delayed due to the inter-processor synchronization or communication time.

At each time instant k, the controller (optimizer) provides the time duration for computing ahead (or backward) based on the current state, X_k (provided by the feedback link), computational speed, B, and the intended destination, R_k, which is provided to the controller as a randomly delayed instruction, R_{k+T_D}. Two strategies are considered in our analysis:

- *Strategy 1* (baseline): For each time instant k, the controller waits for the delayed instruction about the exact information on the computational trajectory before the beginning of the next phase of execution (the computation remains paused until the controller receives instruction about the correct path forward). Once the instruction is received, the controller computes the required time for the vehicle to reach the destination, U_k, and the execution starts at constant speed till it reaches the intended destination in U_k units of time.
- *Strategy 2* (proposed): For each time instant k, the controller does not wait for the delayed instruction about the exact trajectory information, but guesses the route and continues executing for some (random) amount of time, $T_F \leq T_D$. After that, the computation is paused for the remaining time till the instruction

is received (if $T_F < T_D$). Once the instruction is received, the controller finds out how much the execution deviated from the intended state trajectory and computes the required time for the computation to go back to the intended trajectory, $T_R \leq T_F$, and then the execution starts moving at constant speed till it reaches the closest point to the intended (correct) trajectory. After the computation eventually meets the intended trajectory, the controller computes the remaining time to destination and the computation proceeds till it reaches the intended destination.

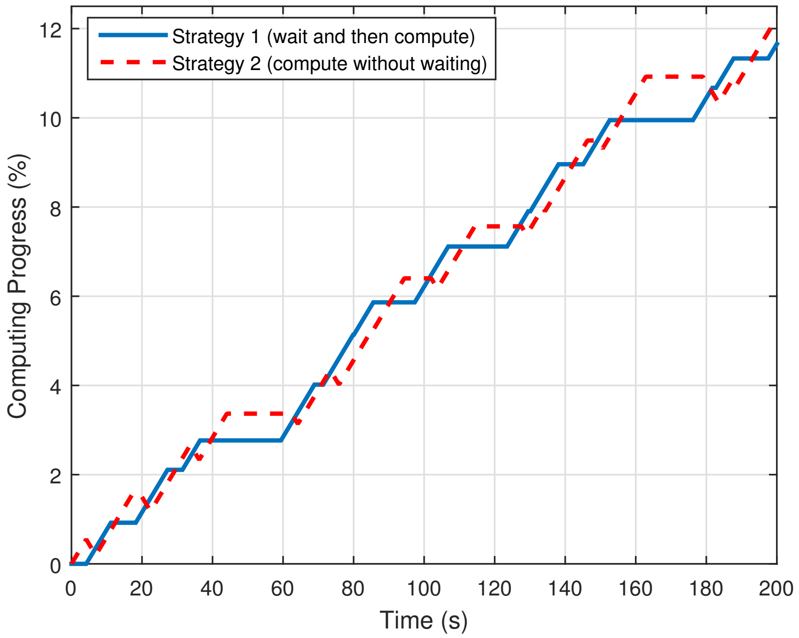


Fig. 4. Illustration of the controller's operation showing the dynamic tradeoff between the two strategies

3 Computation and Communication Delay Distributions

We consider three cases in our analysis: the reverse time is (1) Gaussian distributed, (2) exponentially distributed, and (3) lognormally distributed. The rationale for the different distributions is based on the expected variations in the parallel computational environment and also in the type of application. The simplest one is the Gaussian model for reverse time, which essentially models an incorrect phase that is largely independent of forward execution, making all speculative foray lengths equally probable. The lognormal distribution is suited for the environments in which the probability of being incorrect in the speculative foray exponentially increases with the length of the foray in general. This models highly increasing uncertainty if processors are de-coupled. Referring to Figure 1, the probability of reversal increases exponentially as the distance d of **A-C** is increased ($d_{\vec{cd}} \propto e^{d_{\vec{ac}}}$). With these distributions, the following operation is assumed.

(1) Inter-arrival times for instructions, T_D, is exponentially distributed with mean μ.

(2) Forward computation time, T_F, is Gaussian distributed with mean $F = \mu$ and standard deviation $SD = \mu/3$.

(3) Reverse computation time (deviation from actual route), T_R, is random with means $R = d$, $d/2$, $d/3$, $d/4$, etc., where d is the forward computation distance. We focus on three common cases for the probability distributions of the reverse computating time: Gaussian, exponential, or lognormal.

(4) Speculatively executed time cannot exceed the waiting time for instruction, $T_F \leq T_D$.

(5) Reverse computation time cannot exceed the forward computation time (distance), $T_R \leq T_F$.

The dynamic tradeoff possible between the two strategies (wait and compute versus compute without waiting) can be seen in Figure 4 that shows a zoomed view of the trajectories taken by the two strategies.

4 Numerical Results and Analysis

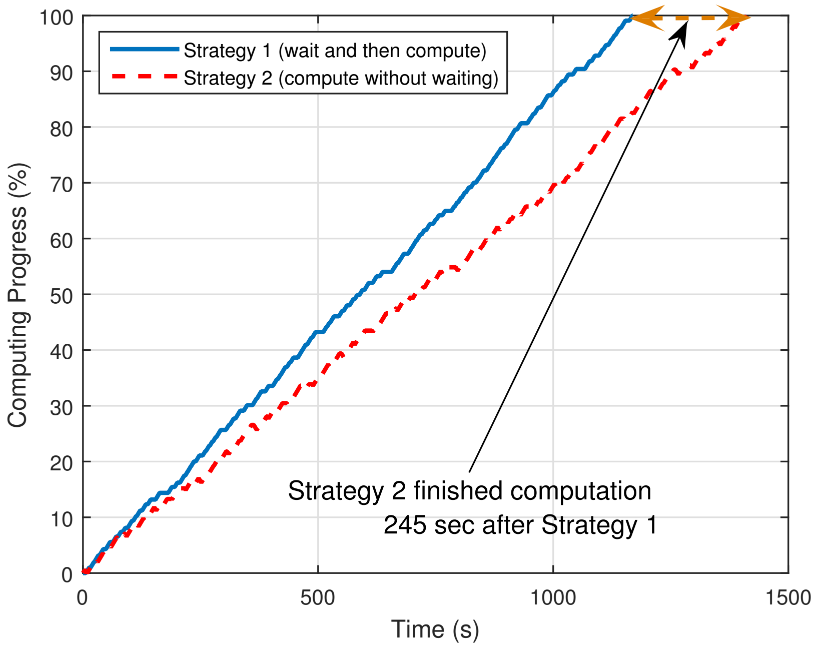
In this section, we present numerical results for the time savings and energy costs for the various scenarios.

The following parameter values are used in the simulations: $X_0 = 0$; $A = 1$; $B = 2$; the inter-arrival times, T_D, of instructions is exponentially distributed with mean $\mu = 5$; the compute ahead time, T_F, is Gaussian distributed with mean $F = \mu$ and $SD = \mu/3$; the destination point for each time step is uniformly distributed in $[10, 20]$; total number of time steps to the final destination, K, is 100; the additional energy cost for staring up the computation at each time step, S, is 6 units; and the total number of Monte Carlo simulation runs is 1000.

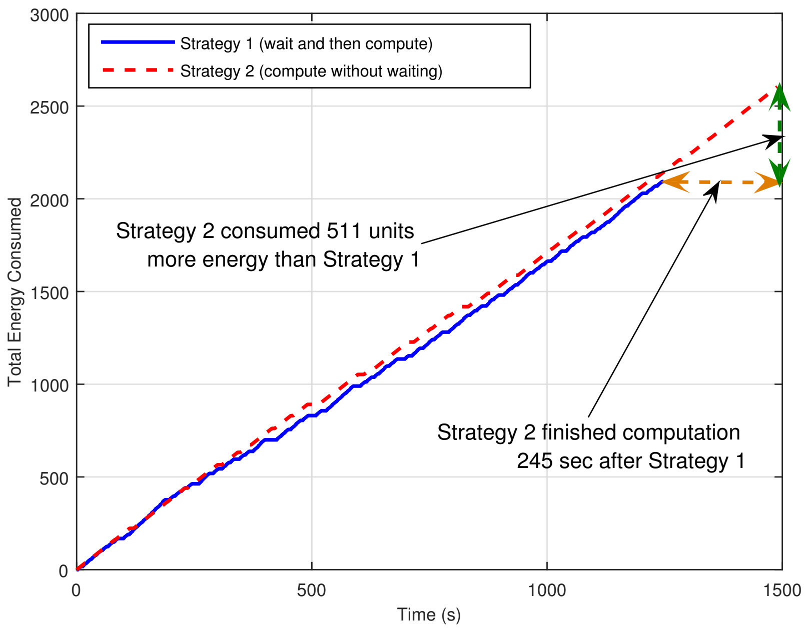
Due to considerations of space, we present the detailed performance charts only for the Guassian cases (Figure 5, Figure 6, Figure 7), but provide summaries of results for others in tabular form in Table 1.

4.1 Case 1: The Reverse Travelled Time is Gaussian Distributed

Figure 5(a) demonstrates the computation as a function of time when the reverse computation length is Gaussian distributed with mean $R = d$, where d is the forward computational length. We observe in this scenario that *Strategy 2* computes to the final state about 245 seconds after *Strategy 1*. This is expected since the reverse trajectory is long (has the same mean as the forward path). Figure 5(b) demonstrates the total energy cost as a function of time for the same scenario. We also observe that *Strategy 2* consumed about 511 units more energy than *Strategy 1*. So, for the parameter values used in this scenario, *Strategy 1* would be selected.

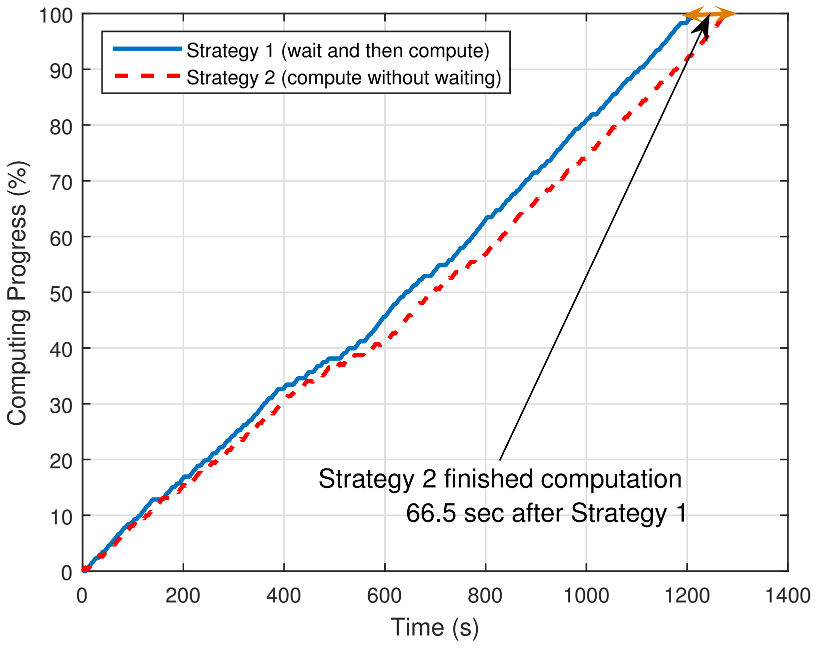


(a) Travel route shows that the time saving is -245 sec

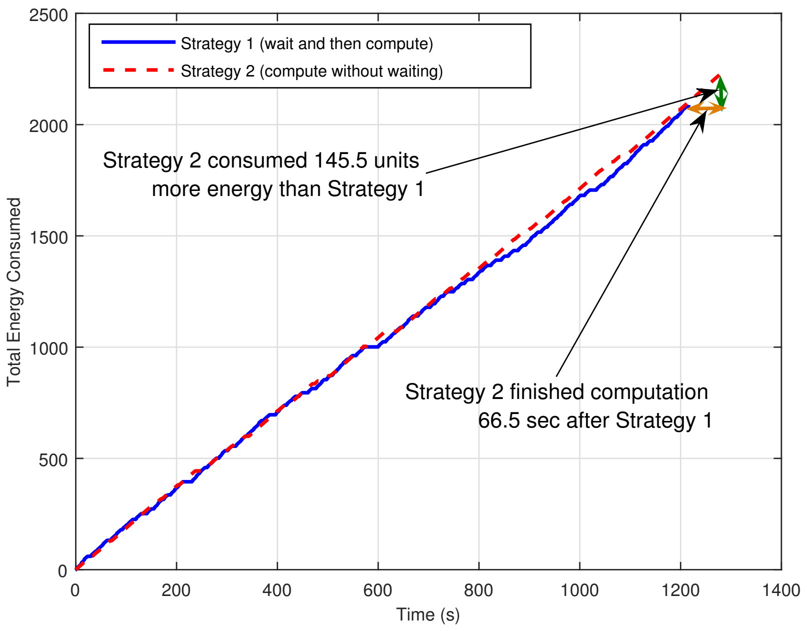


(b) Fuel cost shows that the cost saving is -511 units

Fig. 5. Numerical results for the case where the reverse travelled distance is Gaussian distributed with mean $R = d$ and $SD = d/3$

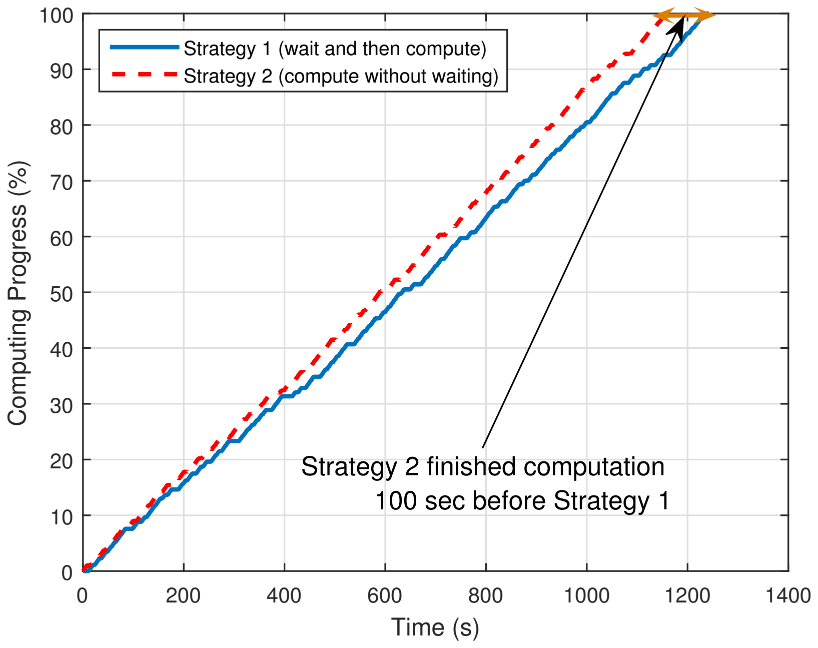


(a) Travel route shows that the time saving is -66.5 sec

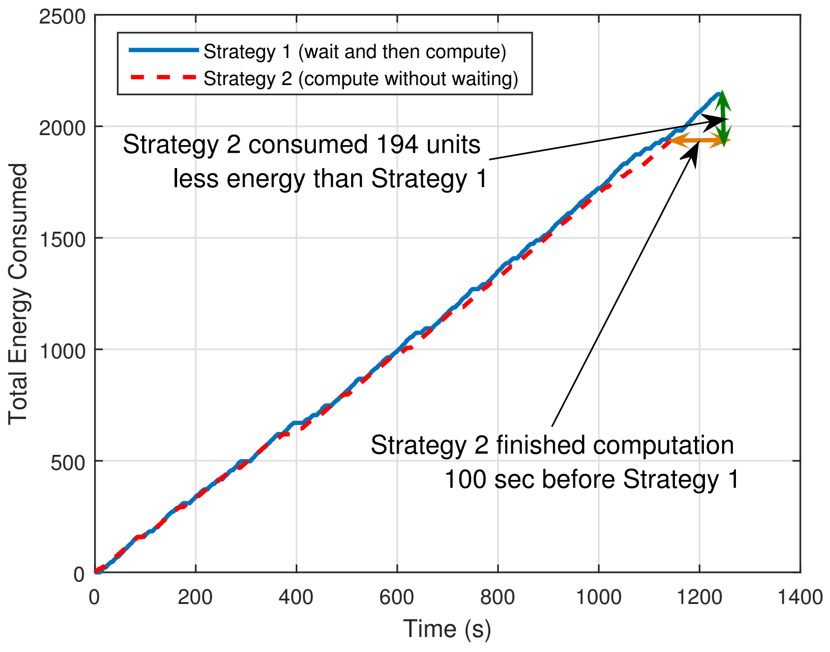


(b) Fuel cost shows that the cost saving is -145.5 units

Fig. 6. Numerical results for the case where the reverse travelled distance is Gaussian distributed with mean $R = d/2$ and $SD = d/6$



(a) Travel route shows that the time saving is 100 sec



(b) Fuel cost shows that the cost saving is 194 units

Fig. 7. Numerical results for the case where the reverse travelled distance is Gaussian distributed with mean $R = d/4$ and $SD = d/12$

Now, we consider the case where the reverse length is Gaussian distributed with mean $R = d/2$, while all other parameters are kept the same. Figure 6 demonstrates the numerical results for this scenario. We observe in this scenario that *Strategy 2* arrived to the final destination about 66.5 seconds after *Strategy 1*, and *Strategy 2* consumed about 145.5 units more energy than *Strategy 1*. We were expecting to see that both strategies arrive at destination at about the same time, but they do not coincide. This is because the go ahead and reverse execution times are not pure Gaussian distributed due to assumptions 4 and 5 presented in the previous section. So, for the parameter values used in this scenario, *Strategy 1* would be selected as optimal.

Now, we consider the case where the reverse length is Gaussian distributed with mean $R = d/4$, while all other parameters are kept the same. Figure 7 demonstrates the numerical results for this scenario. We observe in this scenario that *Strategy 2* arrived at the final state about 100 seconds before *Strategy 1*, and *Strategy 2* consumed about 194 units less energy than *Strategy 1*. So, for the parameter values used in this scenario, *Strategy 2* would be selected.

4.2 Case 2: The Reverse Travelled Time is Exponentially Distributed

Table 1 shows the execution time when the reverse execution length is exponentially distributed with mean $R = d$, where d is the forward execution length. We observe in this scenario that *Strategy 2* arrived at the final state about 114 seconds after *Strategy 1*. This is expected since the reverse length is large (has the same mean as the forward execution). We also observe that *Strategy 2* consumed about 259 units more energy than *Strategy 1*. So, for the parameter values used in this scenario, *Strategy 1* would be selected. Comparing this scenario with the Gaussian distributed scenario that has the same mean (described in Figure 5), we observe that the exponentially distributed reverse time performs better.

Now, we consider the case where the reverse execution length is exponentially distributed with mean $R = d/2$, while all other parameters are kept the same. We observe in this scenario that *Strategy 2* has similar performance as *Strategy 1*. It is expected to see that both scenarios arrive at the final state at about the same time since the mean of the reverse travel route is half of the mean of the forward execution length. Also, it is a coincidence that both scenarios consume the same energy cost. For example, if the additional energy fuel cost for staring up the computation at each time step in *Strategy 1*, S, is less (or more) than 6 units, *Strategy 1* will consume less (or more) fuel cost than *Strategy 2*, while maintaining the same completion times for both scenarios.

Now, we consider the case where the reverse execution length is exponentially distributed with mean $R = d/4$, while all other parameters are kept the same. We observe in this scenario that *Strategy 2* arrived at the final state about 108 seconds before *Strategy 1*, and *Strategy 2* consumed about 240 units less energy than *Strategy 1*. So, for the parameter values used in this scenario, *Strategy 2* would be selected.

4.3 Case 3: The Reverse Travelled Time is Log-Normally Distributed

Table 1 show the times when the reverse execution length is log-normally distributed with mean $R = d$ and $SD = d/3$, where d is the forward execution length. Note that the mean R here represents the mean of the corresponding Gaussian random variable, Y, that generated the lognormal random variable using the relation e^Y. We observe in this scenario that *Strategy 2* arrived at the final state about 289 seconds after *Strategy 1*. This is expected since the reverse execution length increases exponentially with the forward execution length d. We also observe that *Strategy 2* consumed about 585 units more energy than Strategy 1. So, for the parameter values used in this scenario, *Strategy 1* would be selected as optimal. Comparing this scenario with the Gaussian and exponentially distributed scenarios that have the same mean, we observe that the log-normally distributed reverse time performs the worst.

Now, we consider the case where the reverse execution length is log-normally distributed with mean $R = d/6$, while all other parameters are kept the same. We observe in this scenario that *Strategy 2* arrived at the final destination about 54 seconds after *Strategy 1*, and *Strategy 2* consumed about 143 units more energy than *Strategy 1*.

Now, we consider the case where the reverse execution length is log-normally distributed with mean $R = d/12$, while all other parameters are kept the same. We observe in this scenario that *Strategy 2* arrived at the final state about 45 seconds before *Strategy 1*, and *Strategy 2* consumed about 53 units less energy than *Strategy 1*. So, for the parameter values used in this scenario, *Strategy 2* would be selected.

Now, we consider the case where the reverse execution length is log-normally distributed with mean $R = d/8$, while all other parameters are kept the same. We observe in this scenario that *Strategy 2* has similar time savings as *Strategy 1*. Both scenarios arrive at the final state at about the same time. However, *Strategy 2* consumed about 38.37 units more energy than *Strategy 1*. Analytical derivations for the time savings where both scenarios arrive at the same time are described next.

Let Y be a Gaussian random variable with mean R and variance σ^2, i.e. $Y \sim (R, \sigma^2)$, then $G \sim e^Y$ is a log-normal random variable with mean $e^{R+\frac{\sigma^2}{2}}$ [2]. In general, for *Strategy 2* to gain time savings over *Strategy 1*, the expected value of the reverse execution length should be less than half of the expected value of the forward execution length as described earlier in Section 1. This can be formulated as

$$E(G) < \frac{F}{2} \tag{5}$$

where $E(\cdot)$ is the expectation operator and F is the expected value of the forward execution length. Since the expected value of G is $e^{R+\frac{\sigma^2}{2}}$, Equation 5 becomes

$$e^{R+\frac{\sigma^2}{2}} < \frac{F}{2} \tag{6}$$

Let $R = \frac{F}{M}$, our objective is to compute the value of M such that *Strategy 2* gains time savings over *Strategy 1* (arrive at destination earlier). In addition, in our simulations we used $\sigma = \frac{F}{3M}$. Substituting the mean and variance into Equation 6, we get

$$\frac{F}{M} + \frac{1}{2}\left(\frac{F}{3M}\right)^2 < \ln\frac{F}{2} \tag{7}$$

which gives

$$\frac{(18M + F)}{M^2} < \frac{18}{F}\ln\frac{F}{2} \tag{8}$$

In our simulations, we used $F = 5$. So, by solving numerically for M in Equation 8, we get M should be greater than 6.727 for time savings. However, the results show that M should be close to 8. This mismatch between simulation and analytical results is due to assumptions 4 and 5 presented in the previous section that cause violations to the exact Gaussian and lognormal distributions used in the analytical derivations.

Mean (R)	**Time Saving (sec)**	**Energy Saving**
Case 1: The reverse travelled time is Gaussian Distributed		
$R = d$	-245.0	-511.0
$R = d/2$	-66.5	-145.2
$R = d/4$	100.0	194.0
Case 2: The reverse travelled time is Exponentially Distributed		
$R = d$	-114.0	-259.0
$R = d/2$	-8.0	1.0
$R = d/4$	108.0	240.0
Case 3: The reverse travelled time is Log-Normally Distributed		
$R = d$	-289.0	-585.0
$R = d/6$	-54.0	-143.0
$R = d/8$	-7.2	-38.4
$R = d/12$	45.0	53.0

Table 1
Numerical results showing the time and energy savings resulting from the reversible speculative forays as compared to the traditional forward computing. Note that the mean of the reverse computation time (R) is represented in terms of the forward computation time (d)

4.4 Practical Considerations

The developed technique can be employed in any parallel processing platform to speed up the computational process per unit time. It can be used as a prediction mechanism that predicts the performance of the presented strategies (*Strategies 1* and *2*) in terms of time savings and processing cost savings. The predictor inputs are based on the application environment. As discussed earlier, the predictor inputs are: the processing speed, B, the probability distribution of inter-arrival times of the sync instructions from other processors, T_D, the probability distributions of the guessed operations (drive ahead) time, T_F, and reverse operations (reverse travelled) time, T_R, the extra (fuel) cost for staring up the processors (vehicle) from an idle state, S, and the total number of time steps to the final destination, K. These input parameters should be known or estimated from the application environment. The prediction accuracy depends on how close the estimated environment parameters are to the actual ones. The outputs of the predictor are the predicted time and processing cost savings (positive or negative) by using the proposed strategy (*Strategy 2*) over the baseline strategy (*Strategy 1*). Based on the predicted performance savings, the proper strategy will be selected.

The proposed predictor is currently implemented in Matlab [3], but it can be converted to a standalone executable using Matlab Coder [4]. Depending on the application, it can be executed either in each processor or in one (or few) processor(s) and generalizing the selected decision to all processors.

5 Summary and Conclusions

In this paper, we proposed the reversible speculative computing that has an exact analogy with reversible speculative driving. The reversible speculative forays problem is characterized by a linear state-space model and optimized using a dynamic feedback control formulation. We investigated the performance of the developed technique under various environments represented by various probability distribution functions and model parameters. Under the same model parameter values, numerical results showed that exponentially distributed reverse time outperforms the other two investigated distributions in this study, and the Gaussian distributed reverse time outperforms the log-normally distributed one. In addition, reverse speculative forays outperform the traditional one when the reverse time is less than half of the forward time, i.e., $t_{\overrightarrow{cd}} < \frac{t_{\overrightarrow{ac}}}{2}$. We have chosen Monte-Carlo simulations method for investigating the performance of the developed technique because of the difficulty of conducting analytical derivations due to assumptions 4 and 5 that place upper bounds on the distribution functions. The developed technique can be employed in parallel processing platforms to speed up the speculative computational forays per unit time. It can be used as a prediction mechanism that predicts the performance of the two strategies in terms of time savings and net cost savings. It can be implemented in each processor or in one (or few) processor(s) and for generalizing the selected decision to all processors.

References

[1] Kalyan S. Perumalla, "Introduction to Reversible Computing," ISBN 978-1439873403, CRC Press, 2013.

[2] A. M. Mood, F. A. Graybill, and D. C. Boes, "Introduction to the Theory of Statistics," 3rd Ed., New York: McGraw-Hill, 1974. pp. 540-541.

[3] http://www.mathworks.com

[4] http://www.mathworks.com/products/matlab-coder

[5] A. Gupta, I. Akyildiz, and R. Fujimoto, "Performance Analysis of Time Warp With Multiple Homogeneous Processors," IEEE Transactions on Software Engineering, Volume 17, Issue 10, pages 1013–1027, 1991.

[6] D. Jefferson, and A. Witkowski , "An approach to performance analysis of timestamp-driven synchronization mechanism," Proceedings of the 3rd ACM Annual Symposium on Principles of Distributed Computing, 1984.

[7] R. J. Lipton and D. W. Mizell, "Time warp vs. Chandy-Misra: a worst-case comparison," Proceedings of the SCS Multiconference on Distributed Simulation, Volume 22, Number 1, pages 137–143, 1990.

[8] O. R. Gonzalez, A. Tejada and W. S. Gray, "Analysis of design trade-offs in the rollback recovery method for fault tolerant digital control systems," Proceedings of the American Control Conference, Volume 6, pages 4801-4806, 2002.

[9] C. Hou, D. Quinlan, D. Jefferson, R. Fujimoto and R. Vuduc, "Loop synthesis for program inversion," Proceedings of the 4th Workshop on Reversible Computation, Copenhagen, Denmark, pages 72–84, 2012.