Dear reviewers,

Thanks for providing valuable feedbacks. While we accept most of the suggestions, questions and ambiguity will be addressed as follows.

**Novelty of this work:**

Our idea is based on state machine replication, and it is similar to redundant multi-threading in that one leading thread (process) is running ahead of trailing threads (processes). However, our approach is novel in that it tunes the execution rates of the main and shadow processes in a finer grain, in order to achieve a "parameterized" trade-off between completion time and energy consumption. Further, we take advantage of the recovery time after each failure and "leap" the shadows to achieve forward progress, largely improving performance in terms of both completion time and energy consumption. This differs from RMT, of which the "leaping" of the trailing thread results in extra overhead. We will add this analysis to the related work.

**Types of failures tolerated:**

The proposed approach aims to tolerate both permanent failures (e.g., CPU and socket) and transient failures (e.g., OS, runtime), where robust coding techniques alone are not sufficient. Further, we assume the fail-stop model, which is the same assumption in checkpointing and replication. We will state our failure model clearly at the beginning of Section 4.

**Process replication for systems with GPUs:**

GPU is only one potential direction to improve parallelism. Even for future systems where most of the FLOPS come from GPUs, it may be necessary to have a hybrid fault tolerance framework, where our approach is still effective.

**Additional RAM cost in energy consumption:**

Both replication and in-memory checkpointing double the amount of memory. Checkpointing in storage is slow and new fast checkpointing schemes buffer the data in memory. Here we need to point out a misunderstanding from reviewer 2: "This work assumes improved energy efficiency over modular redundancy due to using less CPU resources..." In fact, our assumption is that there is a fixed number of CPUs, but how the application is split into tasks differs among the fault tolerance approaches considered in this paper. This fact can be clearly seen in our models (Eq. 7 and 8).

**Comparison to redundancy:**

The reason why lazy shadowing outperforms process replication in completion time in some cases is that, given a fixed number of CPUs, collocation allows shadow processes to use fewer than half of the CPUs, allowing more CPUs to be allocated to main processes, effectively reducing the workload of each process. This allows shorter completion time for lazy shadowing when there is no failure. However, failures would incur additional delays for lazy shadowing. That's why the winner of the two approaches depends on the scenario. Another question is why process replication has constant 2x time. We agree that process replication can achieve "about 1x baseline runtime" when you double the computing nodes. However, we assume a fixed number of CPUs and process replication only uses half of them for normal computation (the other half for redundancy), resulting in the constant 2x completion time.

**Performance hiccups:**

This also occurs with uncoordinated checkpointing. Reviewer 2 is correct that lazy shadowing is better suited for loosely coupled workloads. However, the goal of this paper is to show that lazy shadowing also applies for tightly coupled workloads. The hiccups are considered in our models (Fig. 5 and Eq. 3). Also, motivated by this, we propose a technique termed "leaping shadows" that overlaps the recovery time after each failure with the time needed to roll forward the shadows, to reduce the impact of this effect.

**Lack of implementation:**

We have an MPI based implementation, referred to as SrMPI, which can execute HPC workloads with lazy replicas. The implementation issues are discussed in Section 4.3. The preliminary results show that lazy shadowing effectively reduces overall energy, and the amount of savings is application dependent. Due to limitation of space, we are preparing another paper to discuss the implementation and results in more details.

**Failure of a single core affects multiple processes:**

We have discussed this in Section 4.2 (shadowed sets), and quantified this effect with "application failure probability" in Section 5.1. The overhead of the resulted re-execution is included in the calculation of time and energy.

**The assumption that the computation phases between two sync calls is short may not always be reasonable:**

Correct. The longer the interval between sync calls is, the looser the coupling is among parallel tasks. Lazy shadowing is better suited for loosely coupled workloads.

**How are execution rates determined:**

It is easy to formulate an optimization problem using the analytical models in Section 5, with the objective to minimize energy. The output would be the optimal execution rates.

**Results indicate at N=1000000 Lazy shadowing leads to longer completion time than process replication however the text is contradictory:**

Our conclusion is "Figure 6(c) reveals that the most time efficient choice depends on MTBF. More specifically, process replication is more suited when MTBF is low while otherwise Lazy Shadowing is better." This is consistent with Fig. 6(c), Fig. 7(a) and Fig. 8(a). We will modify the text to avoid confusion.

**If the main process completes an interval without fault, does the shadow process need to “catch-up”:**

No. The "catch-up" only happens when a main process fails to take advantage of the idle time during recovery.

**The paper assumes that failures do no occur during recovery:**

We have this assumption since during recovery, only one shadow process (whose associated main process has just failed) is doing normal execution while other cores are idle, thus the failure probability during recovery is negligible.

**According to Figure 6(a), there seems to be a large drop in reliability compared with the simple process replication (10X):**

We believe what people care most about are time and energy. The reliability for checkpointing would be much worse as it does not use replication at all. However, it is preferred over process replication for current supercomputers simply because it is more time and energy efficient.

Other suggestions or corrections are all accepted and will be reflected in the revised manuscript.