Alpaca: Task-based intermittent computing

Shubham Bhargava (shubhamb), Devin Qu (devinq)

Project Webpage

https://github.com/deoxys1087/alpaca

Project Description

The emergence of energy-efficient processor architectures allows low-powered devices to operate entirely through energy harvested from their environment. The device will slowly gather energy into an energy buffer (e.g., a capacitor). Once it accumulates enough power, the device switches on and runs, quickly depleting the stored energy. Software must execute in the intermittent execution model: brief bursts of execution interspersed with charging periods. This model presents a unique challenge to programmers, who must ensure programs make progress while behaving as if running with continuous power.

Alpaca is a low-overhead programming model for intermittent computing. In Alpaca, the programmer structures the program as a set of tasks that each individually fit in the device's energy budget. The Alpaca compiler and runtime combine to ensure tasks execute atomically, maintain consistency between volatile and nonvolatile memory, and the program progresses at the task granularity. It accomplishes this by privatizing task-shared data and committing changes at the end of the task. We will evaluate Alpaca's performance in continuous and intermittent power scenarios, memory overhead, programmer effort, and if Alpaca is a viable alternative to traditional checkpoint-based approaches.

- **75% goal**: Basic task implementation and test environment. Framework identifies and privatizes scalars. Programs behave correctly and make forward progress.
- **100% goal:** Complete Alpaca implementation, including compile-time anti-dependency analysis and runtime version-backed bitmasks for privatizing arrays.
- **125% goal:** Implement pointer analysis and improved array analyses. Test on real hardware.

Logistics

Plan of Attack and Schedule

- **Week 1 (10/28):** Research and planning
  - Both - Review Alpaca paper, decide how to implement tasks, what LLVM passes to implement, and detail of our test environment.
- **Week 2 (11/4)**:
  - Shubham - Set up emulator for testing
  - Devin - Implement tasks frontend and start on analysis pass for task-shared scalars
- **Week 3 (11/11):**
  - Shubham - Implement two-phase commit
  - Devin - Complete LLVM analysis pass, implement privatization pass
  - Both - Add microbenchmarks for scalar privatization
- **Week 4 (11/18):**
  - Shubham - Implement runtime for array privatization
  - Devin - Implement LLVM pass for array privatization
  - Both - Evaluate scalar benchmarks and complete milestone report
- **Week 5 (11/25):**
  - Shubham - Implement runtime for improved array analysis
  - Devin - Implement pointer analysis in LLVM
  - Both - Add benchmarks for array privatization
- **Week 6 (12/2):**
  - Both - Finish implementation, evaluate benchmarks and complete final report

The critical path will be weeks 2 and 3, which is the core implementation of the framework, consisting of the privatization identification, runtime commit library, and rewriting memory accesses to the private copies of the data.

Milestone

For the milestone, we plan on completing our 75% goal. The framework should support creating tasks and should identify and privatize any task-shared scalars that are part of write-after-read dependencies.

Literature Search

We have read the original paper and found it interesting, relevant to the class, and likely achievable by two Master's students. Since our project is meant to replicate Alpaca, we have also looked at the implementation of Alpaca. This is a more recent task-based intermittent computing paper by many of the same authors which could also be used as a comparison.

The authors of Alpaca also designed EDB, a debugger for energy harvesting devices. They used it while designing Alpaca and will likely be useful if we get a chance to test our implementation on real hardware.

Resources Needed

We intend to use the Clang and the LLVM framework to compile our binaries, which are readily available for many backend targets. Clang (with the C preprocessor) will allow us to implement the task-based framework using C macros. To emulate a test environment, we will likely use the Unicorn Emulator. The Unicorn Emulator will be useful for emulating power failures and interrupted execution. Given its ease of use, it will be useful in incrementally testing the features of our implementation. If time permits, we may use a TI-MSP430 to compare our framework against the original paper. We don't currently have it and will need to acquire it soon.

Getting Started

So far, we have read the original paper on Alpaca thoroughly to make sure that we can achieve it. Since the target platform for intermittent computing is specialized embedded hardware, we have looked into using emulators such as QEMU or Unicorn Engine as an alternative. There are no immediate roadblocks although we should concurrently focus on acquiring the relevant hardware if we decide that we want to test our implementation beyond emulators.