1 Objective

I am trying to obtain the experimental setup of the "ATLAS: Aging-Aware Task Replication for Multicore Safety-Critical Systems" paper. The experimental setup of this paper can be divided into three main parts: System and Task Model, Aging Model, and Reliability Model.

1.1 Experimental Setup Overview

1.1.1 System and Task Model

- The experimental setup involves a multicore system with m cores. These cores are responsible for executing n periodic hard real-time tasks.
- Each task in the system is characterized by its worst-case execution time (WCET) and its period. The deadline for each task is equal to its period.

1.1.2 Aging Model

- The aging model in the experiment focuses on the effects of Negative Bias Temperature Instability (NBTI), which impacts PMOS transistors.
- The model calculates how aging affects the core frequency, simulating how performance degrades as the system ages.

1.1.3 Reliability Model

- In the experimental setup, tasks' reliability is assessed by considering transient faults, which are modeled using a Poisson process.
- The reliability model ensures that the system is robust by calculating the probability that at least one replica of each task will execute correctly, even in the presence of faults. This redundancy is vital for maintaining the overall reliability of the system.

1.2 Toolchain used

- gem5: Simulates system behavior.
- McPAT: Estimates power, area, and timing.
- HotSpot: Analyzes thermal profiles.
- QUILT: Generates floorplans and thermal models.
- TSP: Ensures thermal safety by setting power limits.

1.2.1 gem5

First, the gem5 simulator is used. Gem5 takes as input the system architecture and task configurations. It simulates the detailed behavior of the system, including how tasks are executed on the multicore platform. The output from gem5 includes detailed data on execution times and power consumption for each task on each core.

1.2.2 McPAT

Next, the simulation results from gem5 are fed into McPAT. McPAT uses these results to provide estimates on power, area, and timing for the simulated system. Essentially, it translates the execution characteristics from gem5 into detailed power and thermal models. The output from McPAT includes the power consumption profiles for the different cores, which is essential for subsequent thermal analysis.

1.2.3 HotSpot

These power profiles are then used as input for HotSpot, a tool designed for thermal modeling. HotSpot requires the floorplan of the chip, which is generated by QUILT, and the power traces from McPAT. HotSpot simulates the thermal behavior of the chip, producing detailed thermal profiles. These profiles show how heat is distributed across the chip during operation, which is crucial for understanding the thermal impacts of different task scheduling strategies.

1.2.4 QUILT

QUILT is used to generate the floorplan of the platform based on the architectural configurations and results from McPAT. It also helps in extracting the RC thermal model of the platform, which HotSpot uses for accurate thermal simulation.

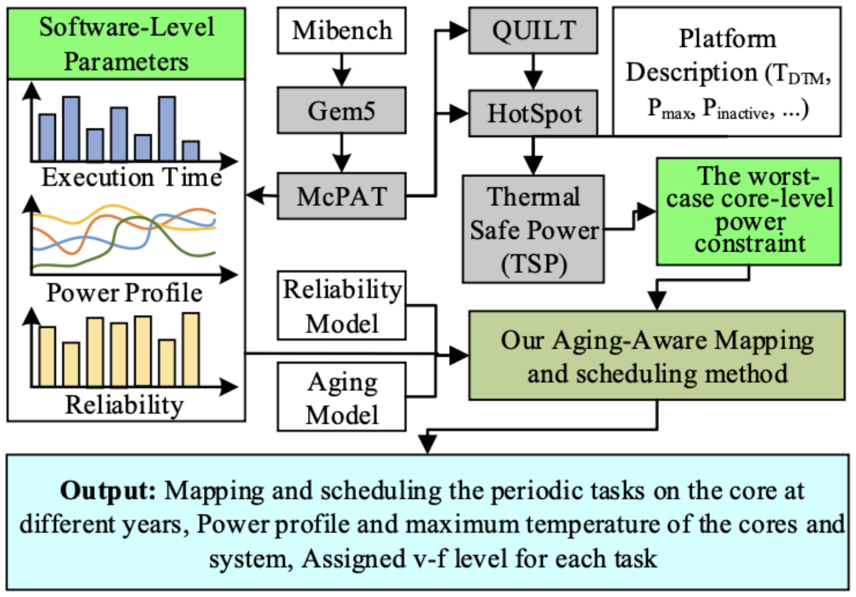
1.2.5 TSP (Thermal Safe Power)

Finally, TSP (Thermal Safe Power) utilizes the power and thermal data provided by McPAT and HotSpot to compute core-level power constraints. These constraints ensure that the power consumption of each core remains within safe thermal limits to prevent overheating. The output from TSP includes these thermal safe power limits, which are used to guide the task scheduling and replication strategies to ensure that the system operates reliably within its thermal constraints.

2 Installing and Using the gem5 simulator

2.1 gem5

gem5 is a powerful simulator used for computer system architecture research. It supports a wide range of architectures and configurations. For more details, please take a look at this link.


Software-Level Parameters
Execution Time
Power Profile
Reliability
Mibench
Gem5
McPAT
QUILT
HotSpot
Platform Description (T_{DTM}, P_{max}, $P_{inactive}$, ...)
Thermal Safe Power (TSP)
The worst-case core-level power constraint
Reliability Model
Aging Model
Our Aging-Aware Mapping and scheduling method

Output: Mapping and scheduling the periodic tasks on the core at different years, Power profile and maximum temperature of the cores and system, Assigned v-f level for each task

Fig. 6: The tool flow of our experiments to analyze feasibility, schedulability, reliability, and temperature.

Figure 1: Experimental setup in the ATLAS paper

2.2 Prerequisites

- **git**: gem5 uses git for version control.
- **gcc**: gcc is used to compile gem5. Version 8 or greater must be used. We support up to gcc Version 12. *Note: gcc Version 9 may be used but is not officially supported due to it increasing gem5 Object File sizes.*
- **Clang**: Clang can also be used. At present, we support Clang 7 to Clang 16 (inclusive).
- **SCons**: gem5 uses SCons as its build environment. SCons 3.0 or greater must be used.
- **Python 3.6+**: gem5 relies on Python development libraries. gem5 can be compiled and run in environments using Python 3.6 or later.
- **protobuf 2.1+ (Optional)**: The protobuf library is used for trace generation and playback.
- **Boost (Optional)**: The Boost library is a set of general-purpose C++ libraries. It is a necessary dependency if you wish to use the SystemC implementation.

For more details, please take a look at this link.

2.3 Installation

The installation process involves cloning the gem5 repository, installing dependencies, and building the simulator.

2.4 Cloning the gem5 Repository

```
git clone https://github.com/gem5/gem5
cd gem5
```

2.5 Installing Dependencies

For Ubuntu:

```
sudo apt-get update
sudo apt-get install -y build-essential python3 python3-dev scons
   swig zlib1g-dev m4 libprotobuf-dev protobuf-compiler libgoogle-
   perftools-dev libboost-all-dev pkg-config
```

2.6 Building gem5 for ARM Architecture

```
scons build/ARM/gem5.opt -j {cpus}
```

2.7 Syscall Mode

In syscall mode, gem5 simulates only the CPU and memory of a system. It does not run a complete operating system but instead intercepts and emulates system calls made by the application. This mode is faster and simpler, making it suitable for simulating single applications and obtaining quick results. However, it is less accurate for simulating the interactions between hardware and a full operating system.

2.8 Full System Mode

In full system mode, gem5 simulates a complete computer system, including the CPU, memory, and peripherals. It can boot and run a full operating system, such as Linux. This mode provides a more accurate and detailed simulation of system behavior, including interactions between the hardware and operating system. It is useful for evaluating system-level performance, debugging, and conducting detailed architectural studies. However, it requires more setup and is slower compared to syscall mode.

2.9 Requirements for Running ARM Full System Mode

To run ARM full system mode in gem5, you need the following components:

- **ARM Kernel Image**: A kernel image compiled for the ARM architecture.
- **Disk Image**: A disk image containing the root filesystem and necessary binaries.
- **Bootloader (Optional)**: Depending on your setup, you might need a bootloader like U-Boot.
- **Device Tree Blob (DTB) (Optional)**: A DTB file that describes the hardware configuration.
- **Gem5 Built for ARM**: Ensure gem5 is compiled for the ARM architecture.

For more details, please take a look at this link.

2.10 Running gem5 for ARM big.LITTLE Architecture

The demo `fs_bigLITTLE.py` script is used to simulate ARM big.LITTLE architecture. It requires several command line arguments to configure the simulation.

- `--cpu-type`: Specifies the type of CPU
- `--num-cpus`: Specifies the number of CPUs.
- `--mem-size`: Specifies the size of the memory.
- `--disk-image`: Path to the disk image file.
- `--kernel`: Path to the kernel image file.

Run the `fs_bigLITTLE.py` **script with appropriate arguments**:

```
build/ARM/gem5.opt configs/example/fs_bigLITTLE.py --cpu-type=
   AtomicSimpleCPU --num-cpus=4 --mem-size=2GB --disk-image=/path/to
   /your/disk-image.img --kernel=/path/to/your/kernel-image
```

Currently, I am trying to generate a disk image with benchmarks installed on it using Packer tools.