Project Report

Intelligent Task Scheduler for Heterogeneous Multi-Core Processors

Author: Krish Sudeep Shah

Course: M.E. Computer Science

Institute: BITS Pilani, Hyderabad Campus

1. Abstract

The paradigm of processor design has shifted from homogeneous multi-core systems to heterogeneous architectures, most notably ARM's big.LITTLE, which utilizes both high-performance and power-efficient cores. This shift presents a significant challenge for traditional operating system schedulers. This project proposes the design and simulation of an intelligent, multi-tiered process scheduler in C++ specifically for such heterogeneous System-on-Chip (SoC) architectures. The proposed solution evolves from a baseline heterogeneity-aware policy to an adaptive AI-driven scheduler that dynamically classifies processes based on their runtime behavior. Furthermore, the simulation incorporates a thermal management model to account for real-world hardware constraints like thermal throttling. The objective is to demonstrate a scheduling strategy that optimizes for three critical metrics simultaneously: system performance, power consumption, and thermal efficiency.

2. Introduction

2.1. Background

Modern mobile and embedded systems, powered by SoCs from industry leaders, prioritize both computational power and energy efficiency. To achieve this balance, they employ heterogeneous computing cores:

- **Performance-Cores (P-cores):** Fast, powerful cores designed for computationally intensive tasks (e.g., gaming, UI rendering) at the cost of higher power consumption and heat generation.
- **Efficiency-Cores (E-cores):** Slower, low-power cores designed for background or less-demanding tasks (e.g., email sync, notifications), significantly extending battery life.

2.2. Problem Statement

A naive OS scheduler that treats all cores as a single, uniform pool is fundamentally

inefficient in a heterogeneous environment. Assigning a background task to a P-core wastes power, while assigning a demanding task to an E-core results in poor user experience. An effective scheduler must therefore be intelligent enough to understand both the nature of the tasks and the specific capabilities of the cores. The core problem is to design a scheduling algorithm that dynamically assigns processes to the most appropriate core type to optimize for performance, power, and thermal output.

3. System Architecture and Proposed Algorithms

This project will simulate a multi-core processor environment and implement a series of increasingly sophisticated scheduling algorithms.

3.1. System Model

The simulation will be built around the following core components:

- **Cores:** A configurable pool of P-cores and E-cores.
- **Processes:** A stream of incoming processes, each with a state (Ready, Running, Waiting). Processes are characterized by their behavior—either **CPU-Bound** (long computation bursts) or **I/O-Bound** (short bursts followed by long waits).
- **Ready Queue:** A queue holding processes that are ready to be scheduled.

3.2. Algorithm 1: Heterogeneity-Aware Scheduler

This baseline algorithm is an improvement over a naive approach. It relies on static, predefined tags to identify process types.

- **Policy:**
    1. Processes tagged as CPU-BOUND are given high priority and are scheduled on available P-cores.
    2. Processes tagged as I/O-BOUND are given lower priority and are scheduled on available E-cores.

3.3. Algorithm 2: Adaptive AI Scheduler

This is the primary innovation, removing the need for static tags by learning a process's type at runtime.

- **Policy:**
    1. **Initial Placement:** A new, unknown process is safely placed on an E-core.
    2. **Behavioral Observation:** The scheduler tracks the cpu_time and wait_time for each process.
    3. **Dynamic Classification:** It calculates a **CPU Intensity Ratio** (cpu_time / (cpu_time + wait_time)). If the ratio exceeds a certain threshold, the process is dynamically classified as CPU-BOUND and becomes a candidate for

migration to a P-core. Otherwise, it remains on E-cores.

3.4. Enhancement: Thermal Management Model

To make the simulation more realistic, a thermal model is included.

- **Logic:**
  1. Each core has a temperature attribute. Running CPU-bound tasks on P-cores increases their temperature.
  2. A "thermal ceiling" is defined. If a P-core's temperature exceeds this ceiling, it enters a **throttled state**, operating at reduced performance.
  3. The scheduler must now make a more complex decision: is it better to run a high-priority task on a hot, throttled P-core or a cool, but slower, E-core?

4. Implementation and Evaluation

4.1. Implementation Details

The simulation will be implemented in C++ using standard libraries. The core design will revolve around three main classes:

- Process: Contains PID, state, priority, and behavioral counters (cpu_time, wait_time).
- Core: Contains Core ID, type (P-core/E-core), state (idle/busy), and temperature.
- Scheduler: Manages the ready queue, the list of cores, and implements the scheduling policies.

4.2. Evaluation Metrics

To objectively compare the performance of the different schedulers, the following Key Performance Indicators (KPIs) will be tracked:

1. **Average Turnaround Time:** The average time from a process's creation to its completion.
2. **CPU Utilization:** The percentage of time P-cores and E-cores are actively running tasks.
3. **Simulated Power Consumption:** A weighted model: Power = (P-core_active_time * 3) + (E-core_active_time * 1).

6. Conclusion

This project aims to tackle a critical and relevant challenge in modern operating systems. By designing and simulating a scheduler that is aware of heterogeneous cores, is adaptive to process behavior, and is conscious of thermal constraints, this work will demonstrate a comprehensive understanding of core OS principles. The expected results will validate that an intelligent scheduling approach can yield

substantial gains in both system performance and power efficiency, aligning directly with the design goals of leading SoC manufacturers.