Yue Zequn (A0129884M)

Luo Zijian (A0224725H)

A Survey on Energy Efficient Multicore and Multiprocessor Systems in IOT: Architecture, Thread Scheduling and Communication

EE5902 CA Report

***Abstract –* In this project, we explored six papers which aim to achieve energy efficiency in multicore and multiprocessor systems in the field of IoT** applications. In “A Hierarchical Reconfigurable Micro-coded Multi-core Processor for IoT Applications[1]**”, the author proposed a simplified logic and shallow pipelined reconfigurable multi-core architecture which utilize long microinstructions for better energy efficiency. In “**ECAP: Energy Efficient CAching for Prefetch Blocks in Tiled Chip MultiProcessors[2]**”, the author explore the technique of using nearby chip free cache set as virtual cache and Confidence-Aware Replacement policy (CARP) to avoid extra energy consumption for unnecessary memory fetching. In “**Energy-Efficient Hardware-Accelerated Synchronization for Shared-L1-Memory Multiprocessor Clusters[3]**”, the author introduced a light-weight hardware-supported synchronization solution to reduce the synchronization overhead in terms of cycles and energy.** **In “****A Two-Tiered Heterogeneous and Reconfigurable Application Processor for Future Internet of Things*[4]*”, the author propose a two-tiered heterogeneous processor architecture for IoT that renders energy efficiency. In “Efficient Thread Mapping for Heterogeneous Multicore IoT Systems*[5]*”, the author offers a thread mapping method combined with their CPU utilization and core capacity on heterogeneous configurations. In “A task-efficient sink node based on embedded multi-core soC for Internet of Things*[6]*”, the author designs the Weighted-Least Connection(WLC) task schedule technique to improve the efficiency of a multi-core Task-Efficient Sink Node (TESN) based on heterogeneous architecture.**

# INTRODUCTION

# PATHS TO ENERGY EFFICIENCY

[4] In IoT architectures, there are two ways to achieve energy efficiency by considering the balance between performance and cost. The first approach is to transform into a sleeping mode in a single high-performance processor when it is not used. However, it is still not energy-efficient because it takes a lot of energy to wake up a high-performance processor from sleep mode. As a result, the second strategy, in which a high-performance processor regulates a succession of low-power processors, is superior. So, in this paper, it presents a two-tiered heterogeneous and reconfigurable processor architecture that comprises of a high-performance host processor that controls a number of low-power interface processors and includes programmable computation and communication units. The two-tiered heterogeneous architecture allows for efficient energy management, while reconfigurability provides further flexibility and energy savings.

[5] Aside from underutilization, running a fast thread on a slow core will result in substantial power consumption because it will take a long time to complete. Because heterogeneous architectures are a new trend, they may obviate the need for existing schedulers. In comparison to other schedulers such as FIFO, the Completely Fair Scheduler (CFS) is extensively used because it avoids process starvation and ensures fairness. Because CFS is Linux's default scheduler, we'll concentrate on making it better for heterogeneous architectures.

[5] This study presents a dynamic thread scheduling technique called Fastest-Thread-Fastest-Core (FTFC), which bases its mapping choice on the conformity of running threads CPU consumption with the performance of available cores. It accomplishes this by dynamically and periodically monitoring the CPU use of running threads, with high CPU utilization threads being assigned to cores capable of delivering high performance, and low CPU utilization threads being assigned to low performance cores.

[5] This research also proposes a new method for modeling heterogeneity that is based on the disparity in core performance. We named it the heterogeneity measure (HM), which describes the system's heterogeneity. We can explore a wide range of heterogeneous combinations using this measurement technique.

[6] The computing performance of a single-core sink node for Internet of Things (IoTs) cannot keep up with the demand for massive data processing as the amount of data collected grows. As a result, in recent years, the sink node, which is built on embedded multi-core SoC for IoTs and maximizes its processing capability, has come into focus. In this research, we present the Weighted-Least Connection (WLC) task schedule technique to improve the efficiency of a multi-core Task-Efficient Sink Node (TESN) based on heterogeneous architecture.

[6] Master cores and slave cores are the two types of cores found in the sink node. The master core is in charge of task distribution, while the slave cores are in charge of data processing. The mailboxes connect all of the cores. The recommended WLC can balance core loads and alleviate network congestion by taking into account each core's real-time processing information and computing performance.

# DETAIL IMPLEMENTATIONS

[4] A core host processor with a communication unit and a high-performance optimized computing unit make up this design. A number of low-power efficient interface processors are connected to the host processor. Minor duties, such as gathering data from sensors and manipulating actuation elements, are handled by the interface processors. Because the interface processors need very little energy to operate, they have no impact on the battery life of IoT deployments. As a result, interface processors can be used in active mode at all times. The host processor, on the other hand, consumes a lot of energy during operation, therefore it is only used infrequently and for short periods of time. When compute-intensive operations such as data analysis, filtering, and complicated security protocols are required, the host processor is engaged.

[4] A computing unit, a communication unit, and a storage unit make up the host processor in our suggested design (s). Fault tolerance (FT) is supplied on a need-to-know basis for various activities carried out by the application and reconfigurable processors, based on the criticality of the function and IoT application. Selected computing and communication unit settings (e.g., core count, operating frequency, modulation power, baseband filtering, etc.) can be reconfigured using the host processor. The reconfiguration allows the host processor to add processing capabilities to IoT devices during mission-critical and/or emergency scenarios, and then remove those capabilities in idle and/or ordinary operation situations to transition to an energy-efficient configuration.

[5] The standard deviation between the normalized performances of particular cores is how we characterized it. Core performance is again modeled by total CPU utilization, which is obtained by serially performing four PARSEC benchmarks on that core, with core performance being directly proportional to average CPU utilization. Here is the formulation equation :

Where Ti is the ith core cpu performance. We employ HM as a key statistic to represent the heterogeneity of the settings we used in the experiment.

[5] Our method does not require modeling the power because it is based on the assumption that power consumption is reliant on the independent variable performance. As a result, we concentrate on performance optimization while leaving the dependent variable power to its own optimization.

[5] Six PARSEC benchmarks are initiated and randomly assigned to one of the six available cores before the FTFC scheduler takes over. Figure 2: Stage 1 offers an example of how to do it with four cores and four threads. The normalized CPU performance is denoted by the letter T. The throughput of the threads on their respective core is denoted by the letter U. The threads' lifespan are set arbitrarily, with a maximum of 30 seconds and a minimum of 5 seconds. Each thread was given the same priority as the previous one. When all threads have completed their execution, the scheduling procedure comes to a close. The CPU time was used to monitor CPU utilization. FTFC is similar to CFS, except it takes into account heterogeneity and does dynamic thread migrations. Because there are no thread migrations when the setup is homogeneous, HM = 0, FTFC is effectively CFS. In this way, FTFC might be considered a continuation of CFS.

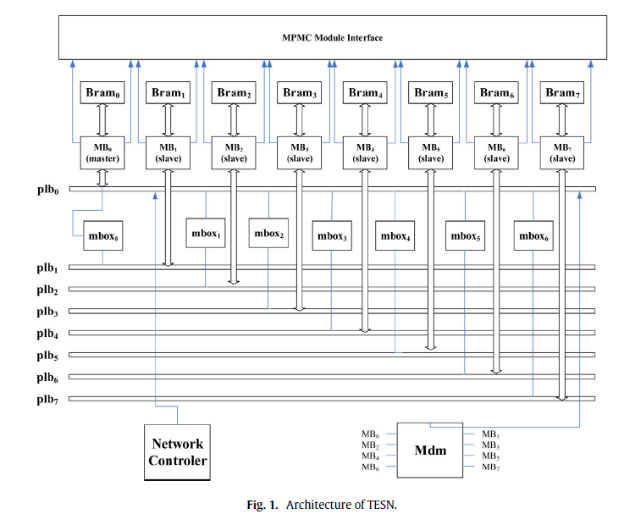
[6] IoTs require large-scale data collecting and processing. Multi-core systems have steadily become more popular as a way to enhance the low computing performance of single-core systems. The goal is to reduce processing time and allow the network to make its resources available faster. The collecting and processing of data is distributed across numerous computational cores in a multi-core system. Each core is capable of doing activities independently and is unaffected by the actions of others. All tasks can be run in parallel on multiple cores. Even if single-core performance is enhanced, the system's processing speed can be greatly increased to meet the demands of large-scale data.

[6] We propose a Task-Efficient Sink Node (TESN) for IoTs based on embedded multi-core SoCs in this paper. In the sink node, there are eight soft cores at work. One is the master core, while the others are slave cores. Each core is a stand-alone processing unit. The master core assigns work to slave cores and balances load, and slave cores execute the tasks that have been accepted by the master core.

# IMPLEMENTATION METHOD COMPARISONS

## Architecture

[6] Tasks are processed in parallel by TESN. It contains eight executing cores and meets the requirements for large-scale data processing. Soft cores are used to implement all of the cores in TESN. MicroBlaze is used to realize the eight executing cores, MB0 to MB7, as shown in Fig. 1. The master core is MB0, while the salver cores are numbered from MB1 through MB7. For salver cores, MB0 assigns jobs and balances the load. MB1 to MB7 are in charge of accomplishing tasks in collaboration. When a new job arrives, the master core will assign it to one of the seven slave cores based on the DWLC. Once a task is assigned to a core, it should be completed on that core. To reduce multi-core communication costs, there is no work migration among cores.



## Network communication

[4] In a heterogeneous IoT environment where devices employ diverse communication architectures and networking protocols, our suggested architecture enables reconfiguration in the communication unit to enable an IoT device to interact with other IoT devices. Modifications to radio settings (e.g., transmission power, antenna gain, modulation and frequency), data link layer parameters (e.g., channel monitoring and association schemes, transmission and sleep scheduling, transmission rate, and error checking), and network layer parameters are all part of the reconfigurability for communication units (e.g., routing, quality of service management, and topology control).

[6] TESN receives data from all of the sensor nodes via wireless transmission. TESN communicates with IoTs using the Network Controler wireless module. The wireless module sends duties to the master core, which it distributes to the appropriate slave core. TESN implements multi-core parallel execution of all jobs. Its computational performance has improved dramatically. The communication between cores, on the other hand, will consume a portion of the CPU's resources. The proportion of communication CPU resource use increases rapidly as the number of cores increases.

## Memory caching, distribution, and access

[5] The FTFC scheduler monitors the CPU utilizations of the running threads after all of the threads have been executed. When migrating between cores with private L1s and L2s and shared L3, some concerns to take into account include remembering the predictor state, permitting L2-L2 data transfers from inactive to active core, and keeping the tags of the various L2 caches coherent.

## Task mapping

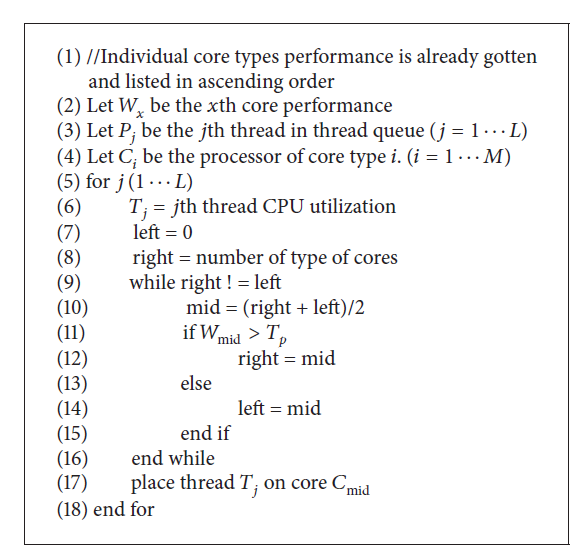
[5] Once the CPU utilization of a thread is

gotten, it is homogenized

[5] The mapping process takes over after the homogenizing and normalizing phases. To reduce temporal complexity and compute overhead, we propose using binary searching mapping (BSM). The thread mapping is done in a binary search, and the CPU usage value of the subject thread is found in the already sorted list of CPU performance. This ensures that the final item, at which the search ends, has the best CPU performance compared to the threads' use.

[5] The main idea of this algorithm is that

Here is the algorithm.



(in my words)

## Security and reliability

[4] Hardware acceleration for encryption/decryption (e.g., advanced encryption standard (AES)) and message authentication (e.g., hash-based message authentication code (HMAC)) are proposed. The use of hardware acceleration for these complicated security primitives increases performance and lowers energy consumption dramatically.

# PERFORMANCE QUANTIFICATIONS AND RESULTS COMPARISONS

# UNIQUNESS IN THE IMPLEMENTATIONS