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

EE5902 CA REPORT

Yue Zequn (A0129884M)
Luo Zijian (A0224725H)

Abstract – In this project, we explored six papers that aim to achieve energy efficiency in multicore and multiprocessor systems in the field of IoT applications. From reconfigurable instruction sets in cache to communication among cores and task mapping schemes that will enable those features to be optimized, this survey provided an outline of microprocessor characteristics that will promote the expansion of the IoT. We have also talked over some of advantages of achieving the abovementioned optimizations, as well as the limits that come with them.

I. INTRODUCTION

Over the last few decades, the progressive semiconductor manufacturing industry is slowly bringing us to the edge of the technological barrier in terms of silicon-based transistor size, the drastic computational improvements in computing power are gradually slowing down and the technical solutions started to branch into parallelism of processors/cores to improve/maintain the processing power trajectory. We are seeing more and more multicore and multiprocessor systems ranging from high-end processors to simple device controllers. With more cores, multiple processors and increasing die size, the power consumption is exponentially increasing as well. For small devices, especially in mobile and compact IoT applications which run on battery, there lies the strong need for an energy-efficient solution with appropriate computational power.

There are several different ways to achieve energy efficiency in a multicore or multiprocessor system. The direct way is to utilize near-threshold computing (NTC) by simply scaling down the supply voltage of part of the system to the optimal energy point (OEP) while maintaining an appropriate processing capability through parallel processing and proper thread scheduling and task mapping across multi-units in the system. More complicated ways of achieving energy efficiency include special software reconfigurable architectures, unique cache fetching strategies and communication techniques among cores or processors. In the field of IoT, there can be special ways of energy saving like synchronization schemes between sensors, master and slave distributed computing approaches. All the above-mentioned perspectives will be thoroughly discussed in this survey on the six chosen papers.

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 utilizes long microinstructions for better energy efficiency. In "***ECAP: Energy Efficient CAching for Prefetch Blocks in Tiled Chip MultiProcessors*****[2]**", the author explores 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 lightweight 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.

II. PATHS TO ENERGY EFFICIENCY

In this section, the technique used to achieve energy efficiency will be discussed briefly.

In **[1]**, the author mainly use 3 ways to save energy. Firstly, the overall control logic is simplified and deep pipelining is avoided. To still keep the same functionality, a more complex instruction set, namely long microinstructions is implemented. Here software complexity is used to bring simplicity to hardware, thus reducing intrinsic energy consumption for all operations. The author also explores ways to combine C and Java Instruction Set Architectures (ISAs) for better programmability. Secondly, with a multicore system, each core can be reconfigured to different working modes to achieve the best energy efficiency. A core can be reconfigured to function as an accelerator like an application-specific processor and take a specific task from the general cores. Thirdly, the system has an integrated router implemented to facilitate a short-range off-chip network for multi-processor systems. The most efficient data routing algorithm can be constructed with the integrated router. Data routing and transfer account for a very large portion of the total energy consumption for IoT application chips.

In **[2]**, the author dive into a better data prefetch technique in tiled chip multiprocessors. Each processor in the package has a private L1 cache and a shared L2 cache. By utilizing the L1 cache of nearby less used processors which are running fewer data-intensive applications as virtual L1 cache, we can avoid data prefetched onto critical data blocks which generate more L1 cache miss and increase the frequency of L2 cache accesses. This essentially saves a large amount of energy in terms of data movement over high-bandwidth buses.

In **[3]**, the author explores a similar domain like the previous article: it is on the basis of a shared cache processing element (PE) cluster. The author focus on establishing a hardware-accelerated synchronization and communication unit (SCU) to facilitate synchronization between all the PEs and perform power management for idling PEs at a lower granularity level.

In **[4]**, the author proposes a two-tiered heterogeneous and reconfigurable processor architecture, which consists of a high-performance host processor coordinating multiple low-power interface processors. To distinguish the characteristics between the host processor and the low-level processor, the author mainly detects their energy-consumption performance in the benchmarks of general IoT applications.

In **[5]**, this author proposes a new method for modelling heterogeneity that is based on the disparity in core performance and also presents a dynamic thread scheduling technique called Fastest-Thread-Fastest-Core (FTFC), which bases its mapping choice on the real-time condition of running threads consumption with the performance of available cores. Therefore, it can significantly improves the energy efficiency of overall system by reducing extra executing time while mapping threads.

In **[6]**, the author presents 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. In the sink node, there are two sorts of cores: master cores and slave cores. Task distribution is handled by the master core, while data processing is handled by

the slave cores. By taking into account each core's real-time processing information and computing performance, the proposed WLC can balance core loads and decrease network congestion.

III. DETAIL IMPLEMENTATIONS

In the field of architecture restructuring, **[1]** propose a reconfigurable multicore system that can adapt to the specific applications at run time. Assuming a system with dual identical cores, each core consists of an arithmetic logic unit (ALU), multiplier, shifter, accumulator etc. which support all normal operation functionalities. The two cores share an on-chip memory where all the programs are stored. There is a separate shared reconfigurable memory where all long micro codes are stored and used by the cores to decode the long micro coded instructions. Some of the instructions can be specifically created according to the need to execute more complex and application-specific workflows with simplified data and logic paths. The long micro instruction has a more complex functionality than conventional simplified instruction sets like Reduced Instruction Set Computer (RISC). Unlike RISC instructions which are general in purpose and very short in nature, the proposed long microinstruction has multiple functionalities, including controlling all functional elements like ALU and multiplication and accumulation unit (MAC); I/O and memory access; control logic to steer data path inside the functional units. For conventional RISC instructions, normally each instruction only handles one operation which means all functional units besides the one that is active at this clock cycle are all idle, this intrinsically increases the total program execution time thus increasing the static power drawn. One might argue that this could be improved by introducing deep pipelining stages to increase the utilization of all the functional units at each cycle. However, deep pipelining will come with additional hardware overhead and extra stages of data steering, this will increase the overall die size and additional dynamic power consumption which defects the purpose of energy efficiency in IoT applications. Owing to the use of micro coding, very complex operations can be done in one instruction[1]. This brings 2 main perks: with a more predictable operational combination inside one long instruction, less branching is needed and all parallelism is fully utilized amongst all the functional units; it also reduces the frequency of memory access and data block prefetching will be less necessary as data loading will be done in parallel with the computations. As a result, the dynamic power for branching and energy heavy memory access in high-speed buses, and the static power of idle functional units are saved, achieving significant energy efficiency improvement.

Another notable benefit of having a core with a reconfigurable instruction set is the flexibility of the functionality of a specific core that can be altered in the whole system. All cores can be either general-purpose processors or application-specific accelerators based on the need for different IoT usage scenarios. This is not achievable by conventional architectures as normal RISC instruction set cores cannot run an application-specific accelerator flow with acceptable speed and efficiency.

In the domain of efficient caching, the author of **[2]** described a new idea to place the prefetched data block inside a Tiled Chip Multiprocessor (TCMP) with an underlying Network on Chip (NoC). Prefetching is a commonly used technique to avoid logic units idling while the needed data is being accessed through the slow bus from the external cache or memory when a data miss happens in the core L1 cache. Due to the nature of the uneven distribution of tasks with large variations of data access behaviour and weightage. The ideal place to

place the prefetched block is definitely the home L1 cache of the core. However, in the case when a data-heavy program is running in a core with limited L1 cache space, placing a prefetched block may evict useful blocks of data. This may cause more cache misses and thus more delays and energy in the long run. A more clever way to place this prefetched block when the local cache of the core is fully utilized, as proposed in the article, is to place it in a nearby neighbouring tile L1 cache. The key question to answer, which will be illustrated in more detail in the next few paragraphs will be: how to define and search for a less used adjacent tile; how to map the metadata for a remote prefetch block so that it can be easily located when the data is needed; and what is the fundamental key for this Energy Efficient CAching for Prefetch Blocks (ECAP) system to reduce the number of network transactions and eventually, saving energy.

To track the satellite data blocks stored in the remote tile cache, metadata is defined for each data block. The metadata consists of two parts, namely the forward pointer and the backward pointer. The forward pointer is stored in the Prefetch Tag Array (PTA) inside each tile and map the details of all the remote prefetch data blocks for this tile. Each PTA index contains a set of Mapping Vectors (MV) which correspond to target sets of remote caches, and the number of MVs is equal to the number of target sets of a remote cache. Hence we do not need to store the matching target set number inside MVs, this significantly reduce the amount of data storage required and speed up the process of locating a remote satellite data block when the data block is on demand. The MV contains a valid bit to indicate data validity, a tag bit to indicate if the remote target set has a prefetched data block inside, and four flag bits to indicate the exact location that the remote prefetch block is stored. To limit the data transmission and avoid underlying network congestion, ECAP only stores satellite prefetch blocks in the adjacent 1-hop tiles. Hence a four-bit phrase can capture the four directions: N, S, E, W. [The following part is not inside the original article; it is a suggested implementation improvement.] In actuality, because the forward pointer already has a tag bit to indicate if the remote block is being used, we can directly use 2 bits to indicate which direction the remote data block is stored, i.e., 00 for N, 01 for S, 10 for E, and 11 for W. when the tag bit is false, the 2-bit direction would be invalid and unmeaningful. The backward pointer is a 4-bit flag, associated with each block inside the L1 cache to indicate the type of data stored. Each flag bit indicates one type of data: invalid block, demand block, local prefetch block, and satellite block. Likely for here, the author could also have used a 2-bit flag to indicate the type of data stored to save half of the overhead bits used for each backward pointer.

Another key algorithm for ECAP is the policy of replacement used when an original old data block needs to be evicted to make place for the new prefetch block. Compare to the conventional prefetch system, ECAP will have an additional type of data block type: the remote prefetch block from neighbouring tiles. This introduces more intricacy and complexity to the replacement techniques. A Confidence-Aware Replacement Policy (CARP) is introduced. It has an intrinsic preponderance over the conventional Least Recently Used (LRU) replacement policy as a block once promoted to MRU position requires a longer time to become LRU. LRU policy will have an inherent problem: when a used demand block resides in a very rarely used cache, it will never be pushed to the end of the recently used queue. This block is now considered a dead block, although it is no longer demanded, this space becomes a blind spot for the LRU replacement algorithm. The proposed CARP algorithm has a mechanism of auto insertion, promotion, and demotion. Each

block is assigned a confidence value which increments when the cache block experience a hit, and decrement by one after each 32-cycle window is passed. The confidence value for a lightly used set is set to be 3 for a demand block and 4 for a local or satellite fetch block. This is to decrease the likeliness of a prefetch block being the identified victim for replacement as a prefetch block is more likely to be demanded later, replacing a prefetch block will cause more cache misses and more network communications if the prefetched block is a remote satellite block. When the CARP system determine if a set is lightly or heavily used, it only considers the local cache demand and access rate as the remote access requests does not reflect the application traits running on the local tile currently. During a cache block replacement, CARP will identify the block with the least confidence value from a lightly used block. The cache controller will inform the owner of the remote data before evicting a satellite block by checking the backward pointer, to keep the data consistency in an ECAP system.

The author of **[3]** proposed a new way of further utilizing the architecture of multiprocessor clusters for IoT low power applications through an optimized PE-to-PE synchronization and communication with a new synchronization and communication unit (SCU). He also discussed the energy-saving fine-grain power management (PM) can bring.

The author mainly focused on the new SCU unit introduction which aims to accelerate the conventional synchronization in a processor cluster. The synchronization is a huge part that needs to be carefully considered as more programs are seeking ways to utilize parallelism. The conventional way of synchronization is normally software implementation variants of the Test-and-Set (TAS) methodology. TAS intrinsically cause core idling and consumes a significant amount of static energy. Specifically to IoT applications, due to limited shared memory space, only a small partition of the data can be loaded at one time, it requires synchronisation at a finer granularity level with small synchronisation free regions (SFRs). The SCU is a hardware solution that is superior in terms of fine-grain control of each PE for better power management.

The SCU consists of several instances of a base unit equivalent to the number of PE cores. It will manage the triggering events and wait-state through direct control of the clock enable signal. This saves a huge amount of power used for the core to consistently check on the atomic register status while waiting for the other PEs to complete the operation. The key of the base unit is a central finite state machine (FSM) which handles the state of the corresponding core: active, sleep and interrupt-handling states.

The SCU has several extension units which are used to generate core-specific events which help to synchronize between core executions. The main component consists of a Notifier, a Mutex, and an Event FIFO. These units will generate application-specific events which will control the status of the core through the event buses.

The SCU unit is integrated into the low-latency logarithmic interconnect (LINT). A private bus is added between the core and its corresponding SCU base unit to avoid bus congestion when multiple cores are communicating with the SCU.

The author of **[4]** proposes a new idea in two-tiered heterogeneous architecture that contains a host processor and some interface processors, which cleverly takes advantage of the computing difference between the processor at each level. The core host processor has a communication unit and a high-performance optimized computing unit. Then many low-power high-efficiency interface processors are

connected to the host processor. Some simple tasks, such as collecting data from sensors and actuation control the hardware elements, are conducted by the low-level interface professors owing to their advantage in less energy cost. But for some complex and high-quality computing operations such as filtering and executing AES protocols, the computing capability of interface processors cannot satisfy the requirement, so the host processor which have more cores and higher clock frequency are suitable to finish these tasks with a small energy cost at the same benchmarks.

Then turn the perspective to the specific units of the host processor. The computing unit, a communication unit, and a storage unit make up the host processor in our proposed design. In order to adjust the uncertainty of allocated workload, the computing unit is reconfigurable and scalable, which means it can select the setting of processing capabilities by changing the number of cores or operational frequency. As for the computing unit, it can communicate with data with other devices in IoT environment and even modify the parameters of some layers. When it comes to the storage unit, the part within computing unit is responsible to record the information from sensors and the analysis from the higher computing devices. On the other hand, the part within communication unit holds the real-time condition in the wireless network topology. In a word, due to its excellent adjusting capability, the overall system can regulate the executing performance to avoid extra power consumption.

In the field of task mapping in the heterogenized system, **[5]** propose a new method of modelling heterogeneity based on the discrepancy between the performances of each core and Fastest-Thread-Fastest-Core (FTFC) dynamic thread scheduling algorithm. Because of the popularity of multi-core systems in IoT systems, how to detect the computing performance in each core of a heterogenized system is very necessary when the multi-core system is scheduling the new-coming task to the proper core. The author formulates the heterogeneity degree by calculating the normalized CPU utilization of individual cores. It is called the heterogeneity measure (HM) which can describe how heterogeneous the system is.

$$\mathbf{HM} = \sqrt{\sum_{i=1}^{N} \left(\frac{T_i - T_{\mathrm{avg}}}{T_{max}}\right)^2} \qquad \mathbf{(1)}$$

where T_i is the i th core CPU performance

Therefore, this method can be used to investigate a large range of heterogeneous combinations. It means that this metric can show the total processing capability of each core by collecting the utilization conditions.

After receiving the real-time information in each core, the system takes FTFC dynamic thread scheduling algorithm as the key method in energy optimization. Firstly, each thread was settled same priority and all the tasks are assigned randomly to some cores. At the same time, the system monitors the CPU utilization of running threads. Secondly, the system updates the heterogeneity information that can reflect the relative energy consumption when some threads are finished. After homogenization (CPU utilization divided by the normalized core performance) in each core, the system can detect the relative threads utilization of each core in a fair metric. Lastly, after normalizing the CPU utilization of running threads by these configurations (CPU utilization divided by the maximum CPU utilization), we can periodically apply our FTFC scheduler that the relatively high utilizing thread combine with the relatively high utilizing core, which solves the underutilization condition by setting appropriate matching between cores and threads. As for the matching strategy, we use the binary searched mapping (BSM) algorithm to reduce the

time complexity owing to the less computation and better energy efficiency by comparisons with WED and MTS searching algorithms. This can guarantee that the selected core in such combinations can achieve the requirement of less power.

The author of **[6]** proposes an eight cores sink node architecture named TESN and a task scheduling strategy named WLC. Limited by the manufacturing process of current Soc, the executing capability of a single-core node is performing not very well when it meets large-scale data. Therefore, the improvement of processing speed in the sink node is very necessary. Obviously, adding more cores is more useful and realistic than adjusting in single-core. For this new node architecture, it adapts the master and slave structure to increase the computing performance. All the tasks can be executed in parallel because the collection and processing of data are allocated to separated computing cores, which means the duty of the master core is to allocate tasks dynamically based on the real-time conditions of the appropriate slave cores and the responsibility of each slave core is to execute its designated task immediately without task migration. At the same time, the central timer module (LogiCORE IP Processor Local Bus) controls global time to transfer instruction and information between these cores. When the visiting of memory from each executing core causes conflicts, the Mdm and MPMC modules are debugging the errors in independent register data. Owing to the synchronous processing and shared-bus design, the cost of multi-core communication is reduced. In other words, the energy efficiency of this architecture is improved by reducing multi-core communication costs.

To achieve the fast processing speed of TESN, how to make a perfect load balancing in task allocation is the pivotal design. The author proposes a task scheduling strategy named WLC, which contains three parts: Threads scheduling of slave cores, Multi-core communication strategy and Multi-core task allocation strategy. As for threads scheduling of slave cores, the communication thread of the slave core sends the finished task number and status of the data-processing thread to the master core. After balancing the processing performance of each slave core, the task allocation thread of the master core allocates the coming tasks to the data-processing thread in the proper slave core. The second part of WLC is a Multi-core communication strategy. Multi-core communication thread strategy can achieve synchronized executing because all the threads can have the right permissions to access and allocate resources. After updating the real-time state from slave cores from communication threads, the master core can allocate tasks by the WLC algorithm. When it turns to Multi-core task allocation, the master core collects real-time information on how many tasks have been executed in each slave core and modify the number of current waiting tasks. After dividing the clock frequency of each core, the weight can reflect the load situation of each core in this multi-processor.

$$\mathbf{weight}\,[\boldsymbol{i}] = \mathbf{task}_{\boldsymbol{c}}\,[\boldsymbol{i}]/\boldsymbol{cp}[\boldsymbol{i}] \qquad \textbf{(2)}$$

When a new task is coming, the master core chooses the salve core with the least weight as the proper core. In a word, designing dynamical scheduling in task allocation can achieve the best use of computing performance in each core. Because it may achieve less task migration in the overall executing process, it consumes less energy.

IV. IMPLEMENTATION METHOD COMPARISONS

A. Architecture

All the architecture modification for energy efficiency does not come free. There are different levels of complexity for the implementation and integration of current

systems. In **[1]** and **[4]**, the hardware complexity to implement the energy-efficient architecture is relatively minor. Yet in **[3]**, the introduction of the SCU comes with significant architecture changes.

In **[1]**, the reconfigurability of the micro coded cores requires minimum hardware integration change. As long as the application-specific is coded into the ISA, with similar hardware the program can execute accordingly. The working mode reconfigurability needs very limited overall structure modification. The only part that may need to be specially tailored is the connectivity between the cores, the ISA module, and the memory. Under different working modes, the overall bus routing and module functionality varies. In [4], the two-tier hierarchical structure requires some interconnect for the host-to-slave communications. The additional co-processor extensions on security features do not need any additional hardware architecture changes.

In **[3]**, the complexity of the SCU into any current existing system is rather high. It requires an addition of an all-new SCU unit and multiple reserved buses need to be in place for the core event to be routed to-and-from the SCU on time. Special units need to be slotted into the original processing cores to receive and generate the events for the energy-saving functionality.

There is always different levels of overhead incurred in each implementation. In **[2]** and **[6]**, the architecture change will require extra data overhead, but in **[3]** and **[4]**, although there is an architecture change, there is no data overhead as the change is more related to within core functional mode changes and a master-slave relation.

In **[2]**, there is a significant level of data overhead attributed to the metadata mapping on the ECAP architecture implementation. To locate the data on the satellite data blocks and keep track of the data set availability, multiple bits need to be added to indicate all the aforementioned information. Those extra bits incurred will be transmitted on top of the original data.

In **[6]**, to implement the TESN scheduling functionality, several new metrics need to be calculated for the algorithm, like the load and congestion value of each core. These values need to be stored, accessed, compared, and updated consistently. This holds true for almost any software implementation.

B. Network communication

For the communication part in the future IoT environment, the most efficient way to achieve energy efficiency is to reduce the extra communication between cores. In **[4]** and **[6]**, they all design a specious communication unit to satisfy the need of translating data. However, not all processers need a separate communication unit as required.

For example, in **[4]**, the author only designs a sophisticated communication unit for the host processor, but for the low-level interface processors, they are just plugged into the host processor by the inter-connection. Owing to the energy cost of sensing elements being heavily larger than that transforming data in interface processors, we can know that the energy consummated by communication is mainly from the host processor.

But for **[6]**, the communication units between master core and slave core are all designed. The aim of the communication unit in the master core is to receive the load situation from each slave core and allocate new tasks to slave cores by mailbox technology. After adjusting all the new-coming tasks from the timer module as desired, the communication unit of slave cores sends the current load congestion state to the master core. As a matter of that,

the energy cost in the communication part depends on the master core and slave cores.

C. Task mapping algorithm

For all multi-core processors in the future IoT implementations, it is obvious that more cores can improve the total computing speed more than the one-single core. Therefore, how to allocate tasks wisely to different executing performances of cores is very crucial. In **[5]** and **[6]**, they all focus on the threads but allocate them in different views.

The first difference is about the aim of the scheduling algorithm. In **[5]**, there is an innovation in the strategy of mapping tasks from threads and cores. Through detecting the CPU utilization from threads and cores at the first stage, and the system updates these metrics with the homogenization and normalization methods. Based on monitoring the extreme capacity of the different cores at that time, the system can output the processing speed order of threads and cores. And applying the Binary Search Mapping algorithm to map these threads and cores can reduce the extra energy consumption in task mapping. In a word, this FTFC scheduler allocates the tasks from the extreme utilization in a later time, which is the un-greedy algorithm to realize the maximum computing potentiality of a multi-core system.

But in **[6]**, the author proposes TESN architecture and WLC task scheduling strategy to realize the best use of threads at that time. Then to compare the ratio of the unfinished task and clock frequency, and allocate the new-coming task to the core with the least weight. As a result, this mapping strategy can achieve the best use of the load conditions and less task migration. It means that this system gets a better energy efficiency because fewer resources are consumed. Obviously, this mapping tasks algorithm is greedy owing to the consideration of the load situation at that time. Yet in **[5]**, it can map these threads and cores with a global optimization view.

Another difference between **[5]** and **[6]** is the function of threads. In **[5]**, the scheduling regards the threads as one kind and sort them dynamically by detecting the extreme computing performance. But in **[6]**, the system distinguishes the load conditions of threads from the master core and slave core, from computing and communication. As a fact that this WLC task scheduling technique assigns new-arriving tasks to the appropriate threads rather than treating them as a single type. It can greatly improve the steady performance between computing and communicating tasks.

V. PERFORMANCE QUANTIFICATIONS AND RESULTS COMPARISONS

Out of the six papers, paper **[2] [3] [5]** and **[6]** use software simulation for the proposed architecture change or algorithm for performance quantification, where paper **[1]** and **[4]** use actual hardware implementation to gauge the performance and energy savings. Paper **[5]** and **[6]** mainly talks about a software implementation of a unique task and thread mapping algorithm hence software simulation is rather accurate in representing the real performance and theoretical energy saving in an IoT multicore or multiprocessor system. For complex architectural changes mentioned in the other four papers, it is hard for a software simulation to represent the real-world performance of such complex signalling and data flow changes inside the circuitries.

The implementation of paper **[2]** is somewhat unique that it is not a pure software implementation. It uses a full RTL implementation of the whole circuit and place-and-routed all the detail modules with a 22nm CMOS library, following all the DRC rules. Another perk on an actual

physical simulation of the circuit gives us an idea of how big the circuit area is for the newly introduced SCU which roughly accounts for the static energy consumption of the whole implementation. This is more realistic in estimating the energy consumption which is utterly critical in IoT applications.

In terms of how the authors compare performance and energy saving, there is a wide range of metrics and benchmarks used throughout the papers.

In **[2]**, as the paper is memory-centric, the author mainly use cache-related benchmarks, including cache miss, network traffic, packet latency and average memory access time. All these metrics are good measurements that contribute significantly to the energy consumption of the whole system. From fig.11 in the article, we can see the cache miss rate under all ranges of workload is the best for ECAP compared to SCP, LRU or Pseudo-LRU prefetching. The author also used the number of flits to quantify the total network traffic in the underlying NoC. From fig. 12 in the article, we can see the gain and energy saving in a lightly loaded system is more compared to a heavily loaded system. This is due to the nature of the ECAP architecture where the additional extra remote L1 caches need to be from those unused satellite blocks of nearby tiles. In **[3]**, the author directly uses the cycle-based energy as a gauge of cycle wise energy saving through voltage scaling and clock signal disablement. Since synchronization also reduces the total execution time for a specific application, the total power consumption is used to quantify the actual energy saving as well. The author also establishes the instructions-per-cycle (IPC) to reflect the cycle saving as more instruction can be executed within a cycle with more efficient synchronization between cores.

In **[4]**, the simulation investigates the effects of different microarchitecture configurations of two-tier architecture on energy usage in each of the benchmarks. When it comes to high-performance processors, the requirements for settings are a little more stringent when they are used for complicated tasks like data processing and data mining. The advantage of a higher clock frequency is that it takes less time to execute. Meanwhile, it must consume 50 times the amount of power as a low-level processor in fair benchmarks. Because of its cheap energy cost, low-level processors were meant to serve as interface processors, performing just basic tasks such as data sensing. Another issue to consider is that when the workload is heavy, a bigger cache size can improve processing performance for both the host processor and the interface processor. In comparison to ARM-based architecture, this two-tier architecture can perform 47.93x implementations while consuming 2.4x less energy for completing one AES encryption.

In **[5]**, the simulation explores the impact of various degrees of heterogeneity on the FTFC scheduler and CFS scheduler. When the system is high configurations (HM=0.1), it is easy to notice that, compared to the CFS scheduler, the FTFC scheduler can save 2.22 per cent of power and boost computing performance by 52.62 per cent. But when the system is in low configurations (HM=0), there is no obvious difference between FTFC scheduler and CFS scheduler because FTFC operates as CFS in the condition of zero heterogeneity. Generally, this FTFC scheduler can provide quicker executing speed and consume less energy than the CFS scheduler in the same degree of heterogeneity.

In **[6]**, the simulation firstly shows that the change of load situation in RR scheduling and our proposed WLC strategy. Based on the result, it is easy to conclude that, with the growth of cores, WLC can keep a better load balance than RR because WLC has a higher computing speed than RR. And then it explores the effect of congestion in RR

and WLC. When the number of cores is little, RR can have less congestion owing to its reduction in information transformation and weight calculations. But when the number of cores increases, the computing capacity of WLC solves the problem of larger congestion, which means there is no difference in this condition. Combined with the results, it shows our proposed WLC scheduling algorithm is suitable to use in future multi-core IoT implementations.

The expansion of computing performance is limited when the number of cores reaches a particular threshold, thus the author detects the average load and congestion of each slave core in a real wireless network. When this TESN architecture is used for a growing number of tasks, simply increasing the number of cores does not appreciably improve the execution time; in fact, more cores result in higher extra energy costs due to inter-core communication. After weighing the overall results, the author proposes the 8-cored TESN architecture as the future IoT node structure since it successfully addresses the need for large-scale data sensing and processing.

VI. UNIQUENESS IN THE IMPLEMENTATIONS

In **[2]**, the author takes the conventional prefetching technique to another level. It is a too commonly used technique and people rarely think of the underlying problem where the eviction of a demand data block may cause even more data misses. In the context of a multiprocessor or multicore system, it may have more consequences when a critical data block in a parallel execution thread is evicted, a huge amount of energy will be used to fetch the data back to the core and other cores executing parallel threads of the same program will need to be stalled. The author takes advantage of the fact that all the cores or PEs are assigned with programs with various intensities and requirements of data storage. This allows a variable L1 cache size through exploiting the connected tiled core network. This is a brilliant idea that provides endless flexibility and the possibility of the prefetching algorithm.

In **[4]**, the author proposes a high-quality architecture by separating the processors through the difference of computing performance. As for the high-level host processor, it should be activated all the time to govern some low-level interface processors. Limited by the constraints of energy resources, the uniqueness of this design is to reduce the power cost by allocating the right processors with proper tasks and the interconnecting communication function.

In **[5]**, the author provides a general measure to express the CPU utilization of each core and explores the impact of different degrees of heterogeneous in throughput and energy cost in FTFC and CFS schedulers. As for the general measure, we can detect the extreme performance of this system after calculating the average performance of all the cores. It is very useful and convenient for the later task mapping algorithm if the system knows the load condition of each core. Therefore, the FTFC scheduler can avoid congestion conditions.

In **[6]**, the author provides an 8-core TESN architecture after testing the total computing performance with a different number of cores and proposes a WLC task scheduling algorithm. Unlike other multi-core scheduling algorithms, the master core firstly divides threads into four categories based on processors and functions and then assigns threads to the most appropriate slave core according to the needs of specific tasks and weights collected from the load situation of each slave core. This scheduling algorithm adopts the idea of greedy scheduling, which is based on the congestion of each core in the current time.

VII. CONCLUSION

Large volumes of data will be generated as a result of the obvious IoT's enormity and rapid expansion, resulting in increasing energy costs. Multi-core and multi-processor devices greatly minimize these energy expenses by equipping IoT devices with the right-provisioned microprocessors and algorithms that can run computations on the sink nodes.

This survey presented an overview of microprocessor characteristics that will support the growth of the IoT, from reconfigurable instruction sets in cache, communication between cores, some special functional units, and task mapping strategies that will enable those characteristics to be optimized. Especially for [5] and [6], we believe that the improvement of thread mapping can immediately reduce the energy consumption in the commercial application.

We have also discussed some of the benefits of attaining the aforementioned optimizations and their accompanying constraints. This study lays the groundwork for future research into application needs and microprocessor optimizations that will support next-generation IoT devices, as multi-core and multi-processors on the IoT are a burgeoning topic of research.

[1] N. Ma, Z. Zou, Z. Lu, L. Zheng, and S. Blixt, "A hierarchical reconfigurable micro-coded multi-core processor for IoT applications," in *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC 2014*, 2014, doi: 10.1109/ReCoSoC.2014.6861360.

[2] D. Deb, J. Jose, and M. Palesi, "ECAP: Energy-efficient caching for prefetch blocks in tiled chip multiprocessors," *IET Comput. Digit. Tech.*, vol. 13, no. 6, 2019, doi: 10.1049/iet-cdt.2019.0035.

[3] F. Glaser, G. Tagliavini, D. Rossi, G. Haugou, Q. Huang, and L. Benini, "Energy-Efficient Hardware-Accelerated Synchronization for Shared-L1-Memory Multiprocessor Clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, 2021, doi: 10.1109/TPDS.2020.3028691.

[4] P. Kansakar and A. Munir, "A two-Tiered heterogeneous and reconfigurable application processor for future internet of things," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2018, vol. 2018-July, doi: 10.1109/ISVLSI.2018.00130.

[5] T. M. Birhanu, Z. Li, H. Sekiya, N. Komuro, and Y. J. Choi, "Efficient Thread Mapping for Heterogeneous Multicore IoT Systems," *Mob. Inf. Syst.*, vol. 2017, 2017, doi: 10.1155/2017/3021565.

[6] T. Qiu, A. Zhao, R. Ma, V. Chang, F. Liu, and Z. Fu, "A task-efficient sink node based on embedded multi-core SoC for Internet of Things," *Futur. Gener. Comput. Syst.*, vol. 82, 2018, doi: 10.1016/j.future.2016.12.024.

	Section	Owner
0	Abstract	Zequn & Zijian
1	Introduction	Zequn
2	Paths to Energy Efficiency	Zequn & Zijian (each focus on own 3 papers)
3	Detail Implementations	Zequn & Zijian (each focus on own 3 papers)
4	Implementation Method Comparisons	
4.1	Architecture	Zequn
4.2	Communication	Zequn & Zijian
4.3	Task mapping	Zijian
5	Performance Quantifications and Results Comparisons	Zequn & Zijian
6	Uniqueness in The Implementations	Zequn & Zijian (each focus on own 3 papers)
7	Conclusion	Zijian

*paper [1][2][3] --- Yue Zequn;

*paper [4][5][6] --- Luo Zijian;