In the field of architecture restructuring, ***“A Hierarchical Reconfigurable Micro-coded Multi-core Processor for IoT Applications[1]***” propose a reconfigurable multicore system which can adapt to the specific applications at run time. Assuming a system with dual identical core, each core consist of 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 program are stored. There is a separated 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 need to execute more complex and application specific workflows with simplified data and logic path. 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 function 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 handle one operation which means all functional units beside the one that is active at this clock cycle are all idle, this intrinsically increase the total program execution time thus increase the static power drawn. One might argue that this could be improved by introduce deep pipelining stage 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 bring 2 main perks: with 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 reduce 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 is saved, achieving significant energy efficiency improvement.

The author of ***“A Two-Tiered Heterogeneous and Reconfigurable Application Processor for Future Internet of Things******[4]”*** propose a new idea in two-tiered heterogeneous architecture contains a host processor and some interface processors, which cleverly takes advantage of the computing difference between processor in 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. For some secondary tasks, such as collecting data from sensors and manipulating drive elements, they are processed by the interface processor. Because interface processors require very little energy to run, they have no impact on the battery life of IoT deployments. Therefore, the interface processor can always be used in active mode. On the other hand, for the host processor, it consumes a lot of energy during operation, so it is rarely used and only used for a short time. When computing-intensive operations such as data analysis, filtering, and complex security protocols are required, the host processor is involved.

Then turn the perspective to the specific unit part of the host processor. The computing unit, communication unit, and storage unit make up the host processor in our proposed design. Fault Tolerance (FT) is based on the functionality and criticality of IoT applications, and provides for various activities performed by applications and reconfigurable processors on a need-to-know basis. The host processor can be used to reconfigure selected computing and communication unit settings. Reconfiguration allows the host processor to add processing capabilities to IoT devices in mission-critical and/or emergency situations, and then remove these capabilities in idle and/or normal operating situations to transition to an energy-saving configuration.

In the field of task mapping in heterogenized system, ***“Efficient Thread Mapping for Heterogeneous Multicore IoT Systems[5]”*** propose a new method of modeling of heterogeneity based on the discrepancy between the performances of each cores and Fastest-Thread-Fastest-Core (FTFC) dynamic thread scheduling algorithm. Because of the popularity of multi-core system in IoT system, 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 proper core. The author formulates the heterogeneity degree by calculating the normalized CPU utilization of individual cores. It means that this metric can shows 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 it can reflects 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 ***“A task-efficient sink node based on embedded multi-core soC for Internet of Things[6]”*** propose 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 single-core node is performing not very well when it meets large-scale data. Therefore, the improvement of processing speed in 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 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 in order to transfer instruction and information between these cores. When the visiting of memory from each executing cores causes the conflicts, the Mdm and MPMC module 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 cost.

In order to achieve 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, contains three parts: Threads scheduling of slave cores, Multi-core communication strategy and Multi-core task allocation strategy. As for the threads scheduling of slave cores, the communication thread of slave core sends the finished task number and status of data-processing thread to the master core. After balancing the processing performance of each slave core, the task allocation thread of master core allocates the coming tasks to the data-processing thread in proper slave core. The second part of WLC is 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 WLC algorithm. When it turns to Multi-core task allocation, the master core collects the real-time information how much tasks have been executed in each slave cores 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. When new task is coming, the master core chooses the salve core with the least weight as the proper core. In a word, designing a 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.

All the architecture modification for energy efficiency does not come free. There are different level of complexity for the implementation and integration to current systems. In [1] and [5], 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 is able to execute accordingly. The working mode reconfigurability need very limited overall structure modification. The only part that may need to be specially tailored is the connectivity between the cores, ISA module, and the memory. Under different working modes, the overall bus routing and module functionality varies. In [5], 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 in order to receive and generate the events for the energy saving functionality.

There is always different levels of overhead incurred in each implementations. In [2] and [6], the architecture change will require extra data overhead, but in [3] and [5], although there is a 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 meta data mapping on the ECAP architecture implementation. In order to locate the data on the satellite data blocks and keep track on the data set availability, multiple bits needs 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 updates consistently. This is true for all types of software implementations.

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

First difference is about the aim of 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 different core 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 allocate the tasks from the extreme utilization in later time, which is the un-greedy algorithm to realize the maximum computing potentiality of 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 less resources are consummated. Obviously, this mapping tasks algorithm is greedy owing to the consideration on 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 master core and slave core, from computing and communication. As a fact of 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.

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

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 to the host processor by the inter-connection. Owing to the energy cost of sensing elements is 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 communication unit in 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 send the current load congestion state to the master core. As s matter of that, the energy cost in the communication part depends on the master core and slave cores.

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 for the purpose of governing 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 right processors with proper tasks and the inter-connecting communication function.

In [5], the author provides a general measure to express the CPU utilization of each core and explores the impact of different degree of heterogeneous in throughput and energy cost in FTFC and CFS scheduler. 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.

In [6], the author provides an 8-core TESN architecture after testing the total computing performance with 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 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 period.

In [4], the simulation investigates the effects of different microarchitecture configurations of two-tierd architecture on energy usage in each of the benchmarks. When it comes to high-performance processor, 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-tierd architecture can perform 47.93x implementations while consuming less 2.4x the energy for completing one AES encryption.

In [5], the simulation explores the impact of various degrees of heterogeneity on 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 percent of power and boosts computing performance by 52.62 percent. 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 CFS scheduler in 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 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 a 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 the 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.

Due to the obvious IoT's magnitude and quick expansion, large volumes of data will be generated, resulting in higher energy costs. By equipping IoT devices with right-provisioned microprocessors and algorithms that can execute computations on the sink nodes, multi-core and multi-processor significantly reduce these energy costs.

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. We have also discussed some of the advantages with achieving the discussed optimizations and corresponding limits. Since multi-core and multi-processors on the IoT is a growing area of research, this study provides a foundation for further research into application requirements and microprocessor optimizations that will support next-generation IoT devices.