Abstract

Due to the needs of high performance computer and embedded systems, processors are widely applyingmulticore technology.Parallel computing becomes increasingly important in current IT technology. However, the theory of parallel computing is not quite satisfied because the multicoresystems develop rapidly.

Previous research has shown that parallel computing will introduce extra cost; for a specific program, high parallelism may not increase the efficiency of the program. There are various parallel systems. It is difficult to find a patternin the relationship between parallelism and efficiency.

The project will focus onimplementing sort algorithmsmerge sort and radix sort on the Cortex-A9 MPCore. Research will be done about the Cortex-A9 MPCore, how the sort algorithms are implemented on these parallel systems will be introduced, and analyze the efficiency of the sort algorithms.

The project will explore the time consumed in parallel computing, for example in splitting and passing data while doing parallel computing.The model of the Cortex-A9 MPCore will be studied to research the efficiencythat is contributed by parallel computing.Two algorithms merge sort and radix sort, will be implemented on the ARM Cortex-A9 MPCore as a research objects and the model that contains profiling data, the costs in computing and the efficiency will be established. And the final conclusion will be made that under the ARM MPCore environment and with specific sort algorithms, a dual core systemis more efficient when it is used inparallel, or it is more efficient when it is in serial.The relationship with parallelism and efficiency will be investigated.

Acknowledgement

I would like to express my sincerely thanks to my supervisorDr.SimonHollis, who give me a lot of help and guides in the project design.

I would also thank my parents, who support my AMSE courses. They give me the chance and support the environment for studying in MSc.

Thank my girl friend for helping me with my project, for review and feedback on my dissertation.

Thank my friends and classmates Luo Wen, Wang Chen, Zhan Na, Lu Jun Chen,GaneshKuppan. Thank them for their advice and encouragement during my project

Really thanks all of them for their support and help in my study.

Table of Content

Abstract I

Acknowledgement II

Table of Content III

Chapter 1 Introduction and Motivation 1

1.1Aim and Objective 2

1.2Thesis Outline 2

Chapter 2 Background and Previous Work 3

2.1 Parallel Processing 3

2.1.1 Cortex A9 MPCoreParallel Configuration 4

2.2 ARM Processors 5

2.2.1 Cortex™-A9 MPcore 6

2.3Background Techniquesof Parallel Computation 8

2.3.1 Virtual Memory and Scatter Load 8

2.3.2 Caches 9

2.3.3 The Interrupt 11

2.3.4 The Mode of ARM 12

2.4. Algorithms and Parallel Theory 13

2.4.1 Parallel Design 13

2.4.2Merge Sort 16

2.4.3Radix Sort 17

2.4.4 The Theory of Parallel Computation 19

2.4.5 Performances of the TwoAlgorithms in Parallel Computing 19

2.4.6Cost and Benefits Analysis 20

Chapter 3Project Implementation 23

3.1 The Design of the Program 23

3.1.1 RealView Development Suit 4.1 Professional 23

3.1.2 Configure the Platform 25
3.1.3 Set the Stack and Clear the Invalid Data 26
3.1.3 Implement the Memory Map 26
3.1.4 Prime Core Starts 30
3.1.5 GIC Configuration 32
3.1.6 Dividing Data 33
3.1.7 The Synchronization of Two Cores 35
3.1.8The Sort Process 36
3.1.9 The Ending Step of the Program 36
3.1.10thePerformance Monitor Unit 37
3.2 Result Analysis 39
3.2.1 Program Analysis 39
3.2.2Simple Divide and Divide with Sort in Merge Sort 40
3.2.3 Simple Divide and Divide with Dort in Radix Sort 45
3.3 Result Summary 50
Chapter 4Project Conclusion and Evaluation 52
4.1Critical Evaluation 52
Chapter 5Future Work 54
5.1Future Work 54
References 55
Appendix A the Design Flow of the Code 57
Appendix B Code Example 60

Chapter 1 Introduction and Motivation

Nowadays, the technology of embedded systems is booming. From the cars we drive to the mobile phones we use, embedded systems are applied in many aspects of our life. High-performance embedded systems are increasing needed in current life.

However, the clock speed of the processor in embedded systems is restricted to the physical laws for a single core.So applying multicore processors has become the trend of embedded systems. Multicore processors can provide better speed performance, while at the same time having low clock speeds and power consumption. The first dual core for commercial use appeared in 2004, and now it has become mainstream for big companies such as INTEL, AMD and ARM.

But in parallel theory, parallel computing is not always more efficient thanserial. Engineers have already done some research in this area. William D. Gropp stated that there were costs additional brought by parallel computing, so parallel computing is not always more efficient than sequential computing [1]. This is because doing parallel will inevitably introduce additional steps, for example deciding which part of the application is done by a specific core.

In order to avoid this additional cost, it is necessary to investigate the model of parallel systems and parallel algorithms. For a specificalgorithm, how it is implemented in parallel system is crucial to the efficiency of the program.

ARM is a well-known international company that is famous for its processor design. The Cortex-A9 MPCore is its recent product, and it is applied on many embedded systems. This project will conduct research based on this processor. The main concernis that Cortex-A9 MPCore has high performance and advanced architecture compared with other processors, and it is mainly applied on embedded systemsthat require it tobe sensitive to power consumption and code efficiency.

For a general idea of parallel computation, data feeding into a parallel system will be distributed into several processors. There will be additional steps in distributing these data, and there will also be time saved because tasks in one processor are reduced. So, if these parts of cost and benefits can be modeled, the overall evaluation of the system can be modeled.

In further work, the whole system can execute the program according to the model to decide the most efficient and least power cost method. For example, if a specific program is modeled that it is most efficient at running on three processors, so the program should be run on three parallel processor systems.

1.1Aim and Objective

In this project, the parallel system will be the Cortex-A9 MPCore, and the programs running on this system will be merge sort and radix sort. The development tool will be RealView Development Suite 4.1 Professional.

In this project, the aim is to study the model of Cortex-A9 MPCore to research the efficiencythat parallel computing contributes, and explore the time consumed in parallel computing, for example in splitting and passing data while doing parallel computing. The main objectives are as below:

- Research some algorithmsand analyze if they can be applied to parallel systems.
- Research Cortex-A9 MPCoreand understand the general structures.
- Configure the Cortex-A9 MPCoreand prepare to safely implement the algorithms on this system.
- Implement the algorithms on the Cortex-A9 MPCore.
- Analyse different parts of the algorithms and record the time(in cycles) consumed in each stage of the program.
- Model the cost and time saved of the parallel computation.
- Model the parallel system and estimate the condition of applying parallel computing.

1.2Thesis Outline

This report contains 5 chapters.The first chapter is introducing the motivation and general introduction of the project. The second chapter is introducing the background technology and previous researches. The chapter three introduces whatis implemented in this project and the analysis of the result. The chapter four gives the conclusions and the critical evaluation of the project. The last chapter looks the drawbacks in this project and offers an improvement in future work.

Chapter 2 Background and Previous Work

Parallelism is increasingly popular in current computer science. It is due to the appearanceof mutli core systems. Intel, AMD and ARM all released their latest products with two to ten cores integrated. The rapid development of parallel systems will demand the developments of theory in parallel computation.

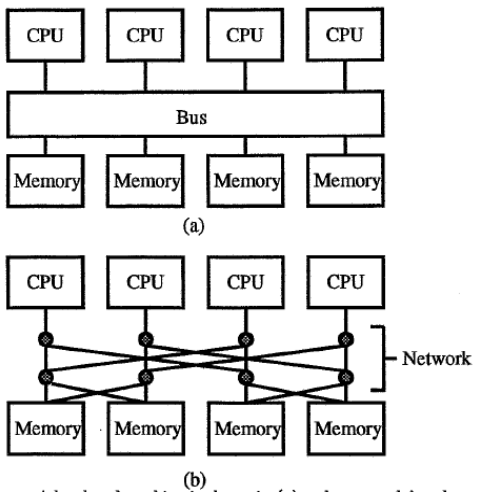
The theory of parallel computation has been developed for decades. They are introduced in this chapter. They are mainly in the aspect of hardware and structure, and the aspect of parallel algorithms

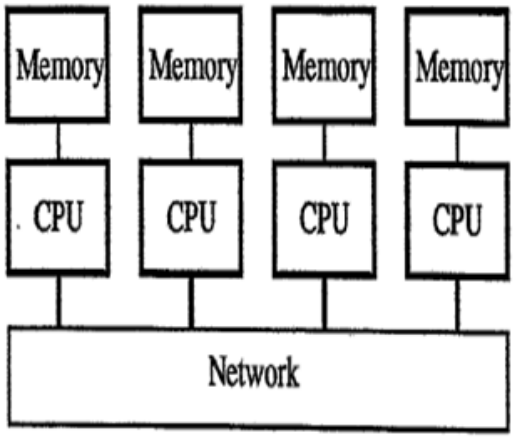
This section will introduce the aspect of hardware in parallel computation. They are basic structures of parallel computing, ARM Cortex A9 MPCore structure, and special technique in parallel computation.

2.1 Parallel Processing

Parallel processing is the solution of a single problem by dividing it into a number of sub-problems, each of which may be solved by a separate agent [2]. From this concept, we can find that the core of parallelism are splitting the program, running separately and combining them correctly.

Replicating the processor is the approach which works in parallel computing. [2](page8) The architecture of muticore processor computers is shown in figure2. The figure (a) and (b) is presenting the shared memory systems and the figure (c) shows the distributed memory systems. The difference here is that in share memory systems, all cpus can access the memory.





(c)

Figure2 (a) is shared memory and bus based systems. Figure (b) is shared memory and network based systems. Figure (c) is distributed memory systems. [2]

2.1.1 Cortex A9 MPCoreParallel Configuration

The Cortex A9 MPCore is naturally the shared memory system. However, it is also possible to apply mechanism to make it like a distributed memory system. This depends on how you manipulate the configurations. From the ARM documentation, in fact the Cortex A9 MPCore is able to configure the two below parallel systems.

- The Symmetric multi-processing (SMP) which means each core has the same view of memory and of sharedhardware. [16] It works as shared memory system. Since each core can access the whole memory and the hardware, any core can be assigned any tasks and even more tasks can be migrated from one core to others.
- The Asymmetricmulti-processing (AMP) which means each core has different view of memory and hardware. [16] It works as distributed memory system.So the core in AMP generally will be assigned different tasks even more maybe different Operating system. Between each core, it may need special mechanism to cooperative with each other.

SMP system is the mainstream in parallel computing, for it can dynamically assign the tasks. For example, if the operating system finds one core runs a time consuming task, it is convenient for OS to migrate the tasks from the busy core to other cores.

In the Cortex A9 MPCore system, the configurations of parallel computing can be implemented both into two types.However, the SMP would be suitable for the research.

2.2 ARM Processors

ARM has been a very outstanding company in the world. The recently developed multicore processor Cortex A9 MPCoreis a highly efficient and low power design. In my project, the Cortex A9 mpcore processor will be applied to implement the algorithms.

Since it will be a big topic to discuss a whole architecture of arm, here I just introduce some important aspects of ARM.

First, ARM processor applies the architecture of RISC, which represents Reduced Instruction Set Computer [3]. The ARM processor has the features as belows:

- A load –store architecture where instructions that process data operate only on registers and are separate from instructions that access memory [3]. This feature will reduce the complexity of the processor, but loss the convenience in code writing.
- Compared with non-fixed instructions, fixed-length 32-bit instructions also can make the processor simpler to implement [3]. 3-address instruction formats, ARM use it instead of using extra bits to record the next instruction address [3]. It reduces the bits of instructions so as to reduce the complexity of the processor.
- For another, ARM does not use the features like register windows, delayed branches and single-cycle execution of all instructions. Some are for the reasons as above to make the processor simpler, others are they are too expensive to a practice chip.

All of above features reduce the complexity of the processor, which will increase its performance in frequency. According to Steve Furber, ARM can be simpler in instructions which he refer to simper instructions are executed faster and do improvements in other devices (like cashes and bus) [3]. This is also the trick that ARM can maintain the low power consumption while execute in high frequency.

Second, as a parallel computing system, ARM processor applies both pipelines and replicates the devices.

ARM has 4 types of pipelines:

- 3-pipelines is fetch, decode and execute, for example ARM7TDMI.[4]
- 5-pipelines is fetch, decode, execute, access data and write back , for example ARM9TDMI.[4]
- 6-pipelines is fetch, decode, register read, execute, access data and write back, for example ARM10 core.[4]

- 8-pipelines is fetch1,fetch2, decode, register read, execute, access data1, access data2 and write back, for example ARM11 core, the Cortex-A9.[4]

Pipelines make ARM processor can compute the instructions at several stages in parallel. And 8 stage pipelines is the latest technique in ARM, which I will be performed on in the project.

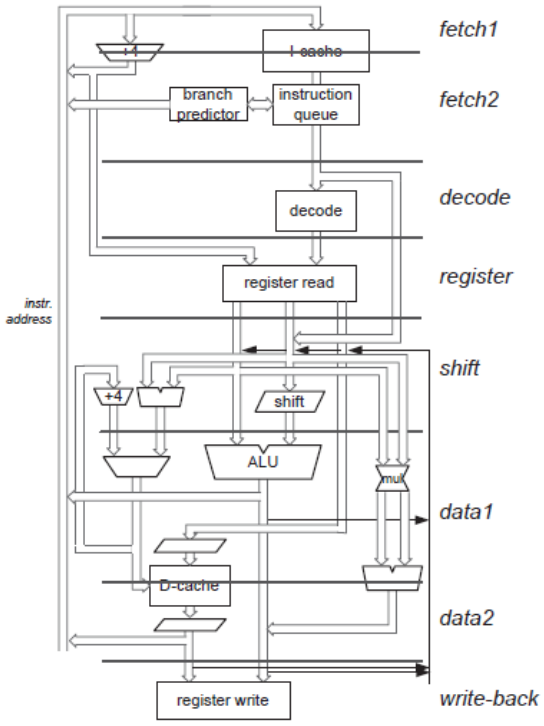


Figure4: pipeline of ARM architectures [5]

ARM also uses more than one processor to replicate its processors, which can achieve the work on parallel computing. The processor as Cortex-A9 MPcore applies the multiple cores, which I will discuss in later section.

2.2.1 Cortex™-A9 MPcore

The Cortex™-A9 MPcore which I will work on in this project is shown as bellows. In this project, the Cortex™-A9 MPcore will be used as a platform to implement the algorithms, and the performance and the costs of parallel computing will be profiled.

However, it is used as a bare core in this project, which means the configurations should be done for the better interface between the applications (which refers to the sort algorithms) and hardware. In real situations, these steps should be done by the Operating Systems. In this section, the hardware is introduced, and in Chapter 3, the configuration will be introduced.

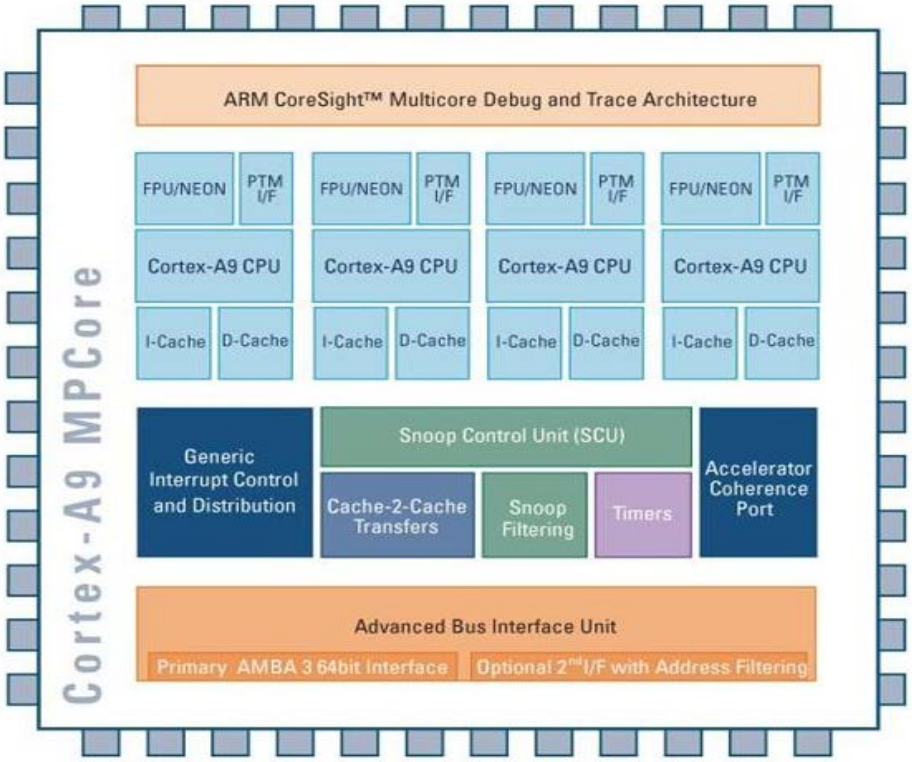


Figure5: The architecture of Cortex™-A9 MPcore[12]

The Cortex-A9 MPcore processor consists of from one to four Cortex-A9 processors in a cluster. [5] From the architecture, we can find:

- Each core in Cortex-A9 MPcore processor has its data cashes and instructions cashes, the size of cashes can be 16kb, 32kb, or 64kb. [5]
- ASnoop Control Unit(SCU) that can be used to ensure coherency within the cluster.
- The interrupt controller (GIC) is used to control the interrupt in each core.[5]
- The Accelerator Coherency Port (ACP) is used for transferring the coherent memory. From the figure.[5]
- The timer is the block which works as register incremented with the clock cycles.

From this figure, it indicates the following elements in the Cortex-A9 MPcore processor, which are helpful to the configuration of parallel computing.

1. Each core has the local instruction caches and the data caches. Cashes are introduced in section 2.3.2.
2. The SCU device is shared for all four cores. The SCU can maintain data cache coherency between Cortex A9 MPCores.
3. The GIC is a bit complicated, it contains two parts, the distributor is shared for four cores which is shown as in the figure, and the interface for four cores is local to each cores, which will be discussed later.

2.3Background Techniquesof Parallel Computation

In this project, it is complicated to configure the Cortex A9 MPCore so that the parallel algorithms can safely be processed. It needs carefully understand the structures and some techniques used in computer architecture.

Several techniques applied to the configuration for parallel computing will be introduced in next several parts.

2.3.1 Virtual Memory and Scatter Load

In this project, in order to configure the platform which the parallel algorithms will run on, there will be several requirementsforthe configuration of the memory. So, two techniques will be applied to manipulate memory, they are the scatter load and the virtual memory.

The scatter load is kind of a notification to the Cortex A9 MPCore which tells the processors how the image (which is compiled program) located in memory when in loading time and executing time.It is edited in txt file, and will join to the project through Realview Development Suit. Take the following figure as example:

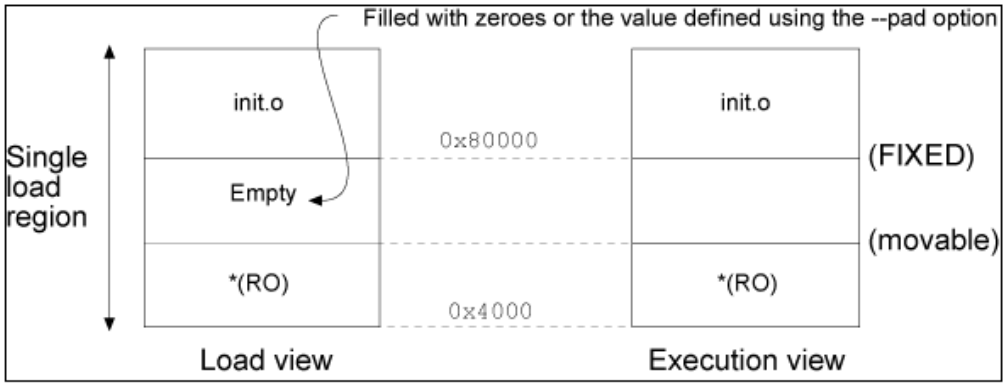


Figure 6: Scatter File example [17]

The scatter file will do the operation that when the program is loaded or executed, put the code and data as the scatter file described.

Thenthe design of where to put the code in memory, where to declare a heap for the application, where to put the area for each core private configurations and private data is implemented by virtual memory. The specificeach memory part will be analyzed in later parts with the code. Now a general technique of organize this memory is widely applied, which is called virtual memory. It is not only used in RISC architecture (ARM), but also widely applied in computers to manage the memory management.

Virtual memory is a very useful technology. In ARM Cortex A9 MPCore, It ismanipulated by the devicememory management unit (MMU).Basically, virtual memory is doing the memory mapping, which means get the address form the processor then map to a related address (which same or not the same as the original address) inphysical address. By applying memory address, it contributes lots of conveniences, for example, all the programs would be able to run on same virtual address without conflicts. The example is as follows:

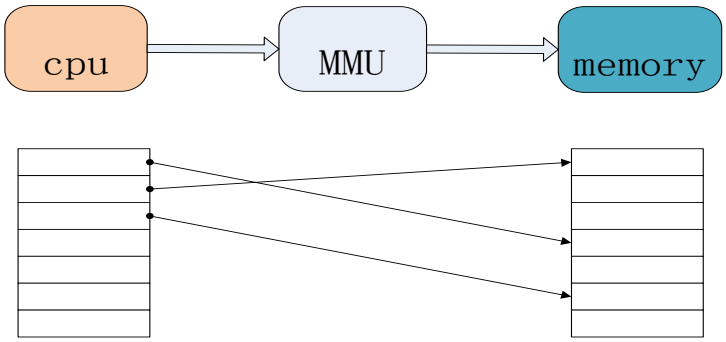


Figure 7: Virtual Memory Map to Physical Memory

In parallel systems, the CPU0 want to write data to a block of memory 0x0100-0x0200. However, the memory block 0x0100-0x0200 is used for CPU1 and currently unavailable for CPU 0. Sothe MMU can redirect the block to 0x0300-0x0400. Every time, the CPU0 ‘believe’ that it is accessing the 0x0100-0x0200, while actually the MMU is automatically translating the address to 0x0300-0x0400.

From the example above, the advantage here is that, the CPUs can use memory without the consideration of memory conflictions. So, doing this virtual memory, the memory can be used very convenient.

2.3.2 Caches

In ARM processer or current computers, the caches are very important device for fast computing. It is helpful but also easy to encounter errors, especially in parallel computing.

Nowadays, the calculation frequency of CPUs is much faster than the frequency of accessing the external memory. So accessing the external memory is becoming the bottle neck for the efficiency of the fast computation.

A cache is a small, fast block of memory which (conceptually at least) sits between theprocessor core and main memory. It holds copies of items in main memory. [18]

The basic idea of caches is that it is try to avoid using the external memory by using integrate fast memory. So there comes the technology that integrates small but fast blocks of memory on the CPUs, and then usesthese memories instead of external memory.

Similarly, the ARM caches is doing the task that when you runs instructions, it extracts memory from the external main memory, which means there are the original instructions in main memory and copies of the instructions in the caches.

When a CPU reads an instruction, the first thing it will do is checking if the instruction exists in the caches. If the instruction exists in the caches, then the CPU reads from the caches. If the instruction does not exist in caches, then the CPU will check it in the main memory and fill the caches with the instructions.

In write procedure, it will show the similar steps, but different processor or different configurations may show differences. For Cortex A9 MPCore, it applies two write policies.

- Write-through. With this policy writes are performed to both the cache and main memory.[20]
- Write-back. In this case, writes are performed only to the cache, and not to main memory.[20]

In the previous parts, it has been discussed that the Cortex A9MPCore has local caches for each core. So the configuration is as figure 8.

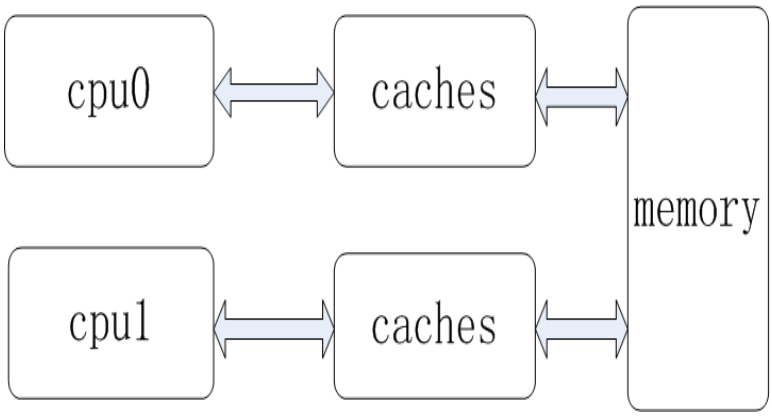


Figure 8: two cores caches system

However, for a mutlicore system, for example two cores systems, There will be a problem called caches coherence in the computations.

This problem happens when CPU0 and CPU1 all extract some memory from the main memory to the caches, and by accidently the two caches hold the same variable. So when CPU0 changes the variable in its caches or even in main memory, at that time

the variable in the caches of CPU1 is ‘stale’(mean the old data). The error will be encountered when the variable is used.

In mutli core systems, for a safely parallel computation an important problem is that the system should keep caches coherent. It means that changes to data held inone processor’s cache are visible to the other processors, making it impossible for processors tosee ‘stale’ copies of data. [20]

Another problem in cortex a9 mpcore systems is that the initialization of the caches, it should be pointed out that the caches use flags to indicate the states of the caches. For example, the ‘valid’ marks the line as containing data that can be used, the ‘dirty’ marks thecache (or part of it) holds data which is not the same as (newer than) the contents of mainmemory. [20]

The problem comes that when the arm processor is power on, the caches may contain the contents remains from last execution and the flag may even stay valid, so it should invalid all the caches when power on the processors unless the processors can do it automatically.

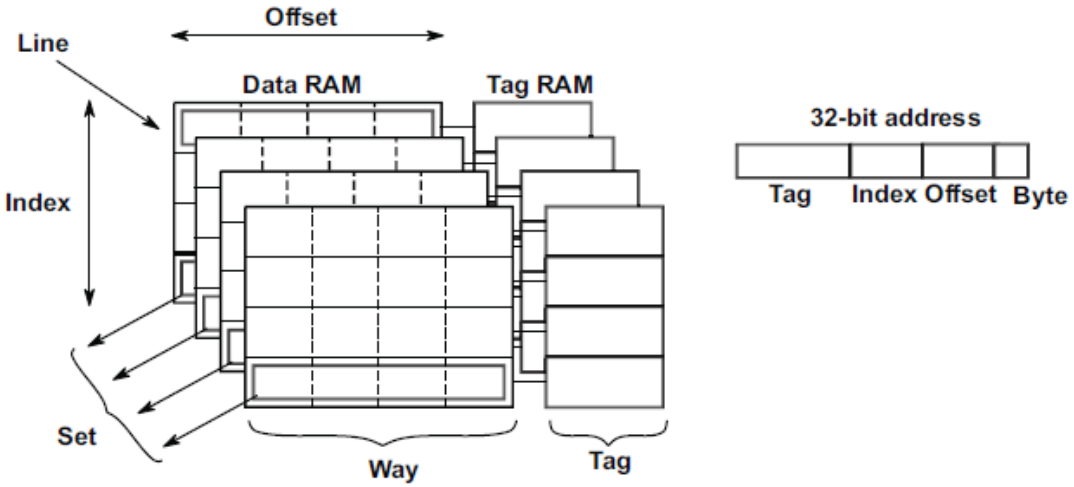


Figure 9: The Structure of Caches in ARM [20]

2.3.3 The Interrupt

In this project, the interrupt is introduced because the needs to response to some external signals.For example in a mutli core system, some of the communications cannot be finished by a singlecore, a signal between cores is needed. In this project, there is a situation that one core is power down for reducing the energy cost, so it cannot awake itself. An external signal, which may from the other core, is needed to wake the sleeping core.

ARM uses two types of interrupts, the IRQ and FIQ. The FIQ is usually specially configured to implement a fast interrupt. And it is always reserved to system-specific device. So in this project, the IRQ will be applied.

The Cortex A9 MPCore system has 32 interrupts, 16 of which are software interrupts.The software interrupts can be used between cores, which are used in this project.

The interrupts are managed by the device generic interrupt controller (GIC).Which is shown in figure 5.

The ARM interrupt are implemented with priority. All of the interrupt applied will have a priority number. Lower priority number means higher priority level and the core receives the interrupt can decide only interrupt that below certain priority number will be carried out.

2.3.4 The Mode of ARM

This part is introducing the modes of the ARM, which is relevant to the program state. The ARM Cortex A9 MPCore support seven modes for processors, which are FIQ, IRQ, SUPERVISOR,ABORT,UNDEFINRD, SYSTEM and USER. Six of the former modes are privileged and the last is unprivileged.

The ARM architecture provides these modes that support normal software execution and handleexceptions,the current mode determines the set of registers that are available and the privilege of theexecuting software. [53] The mode in fact is defining different states for different programs, for example, if the current programs is executing in the User mode, which give the program the ability to access certain the registers and forbidden the program to do some operations.

Application level view	User mode	System mode	Supervisor mode	Monitor mode ‡	Abort mode	Undefined mode	IRQ mode	FIQ mode
R0	R0_usr							
R1	R1_usr							
R2	R2_usr							
R3	R3_usr							
R4	R4_usr							
R5	R5_usr							
R6	R6_usr							
R7	R7_usr							
R8	R8_usr							R8_fiq
R9	R9_usr							R9_fiq
R10	R10_usr							R10_fiq
R11	R11_usr							R11_fiq
R12	R12_usr							R12_fiq
SP	SP_usr		SP_svc	SP_mon ‡	SP_abt	SP_und	SP_irq	SP_fiq
LR	LR_usr		LR_svc	LR_mon ‡	LR_abt	LR_und	LR_irq	LR_fiq
PC	PC							
APSR	CPSR							
			SPSR_svc	SPSR_mon ‡	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Figure 10: Registers in Different Mode [54]From the figure 10, it is showing that almost every mode has its own SP register. All these registers are needed to be initialized first. It is because if the stacksare not carefully set, the stacks of mutli modes may overwrite each other or overwrite the code. This will causeproblems in executing.

2.4. Algorithms and Parallel Theory

In this project, two algorithms merge sort, radix sort are chosen to be as case study. The principle of choosing these algorithms is that they can be applied in parallel computing. These algorithms all can be split easily and safely, because their data have the features that no dependence between each other.

2.4.1 Parallel Design

The design methodology of a parallel computing system comprises four steps [7].

- Partitioning: correctly understand the problem, and initialize the program to classify the tasks [7]
- Communication: make sure all processors locate all the data, if not, communicate with each other [7].
- Agglomeration: if necessary recombine small tasks into larger tasks, this is for reduce the communication cost and code effort. It is based on the real applications whether should be more parallel or more sequential [7].
- Mapping: at last, place the tasks to the processors [7].

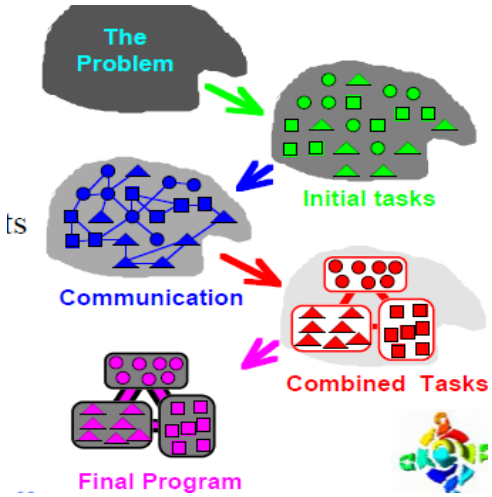


Figure 11: the steps for software parallel design [7].

We can study the parallel computing by using different Partitioning.

One method used functional decomposition [15]. It is the method to split the instructions into several parts [15]. The functional decomposition offers us a different way to consider a problem. It split the problem in different stages, and each stage can work independently. The common example of Functional Decomposition is pipelines[15]. The drawback of it is the program cannot be split in to large number of parts due to the limited number of pipelines in processors.

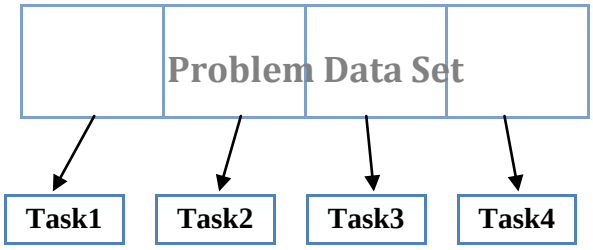


Figure 12: The Domain decomposition [15].

Another method which will be applied in the project is Domain Decomposition [15]. It is a simple and effective method in data decomposition, and used in large range of applications [15]. The procedure is shown as figure11. Each separate processor is in charge of a domain.

In this project, the method of parallel computing is that the data is split into different sub-tasks and send to the separate processors. When looking for algorithms which are suitable to apply these parallel methods, the constraint is that the data should not be dependent with each other. There are two examples, which can explain the differences.
First is

1. Get the input data a and b
2. X=a, Y=b
3. Then X=X+1,Y=Y+1

Second is

1. Get the data input a and b
2. X=a,Y=b
3. Then X=X+Y

The first example, the data has no dependence with each other, while in the second example data is dependent between X and Y. When a common algorithm is adapted to parallel computation with the data decomposition, the consideration is the relationships of the data.

In this project, the merge sort and radix sort are very similar, their data is not dependent in sort processes, but an additional step is needed to combine the result.

For example the data 1, 4, 7,9,3,4 is feed into sort algorithms. The most common idea is that the data is equally divided into two groups: the 1, 4, 7 and the 9, 3, 4. However, these groups will end up in the 1, 4, 7 and 3, 4, 9 if it is sorted in separate cores. So, in last step, the merge together is needed to combine the result from the different processors, the result will be 1, 3, 4,4,7,9. Alternatively, the divide step can be improved, if a smart way can be introduced to split the data into group 1,3,4 and 4,7,9, then the last merge step is not necessary.

Overall, the merge sort and radix sort may not the perfect parallel algorithms. But, they can be implemented on these parallel systems. And as a matter of fact, most of the program is not perfectly parallel, and it is also a very interesting topic that how the not well parallel program will influence the whole efficiency.

In this project, these two algorithms will be introduced on the Cortex A9 MPCore platform. Take the merge sort as example, in the project, different data will be tested, assuming the data is an array of ten integers. So, the domain decomposition can be applied as Figure 12. And the last merge step may be taken on the condition of how split happens.

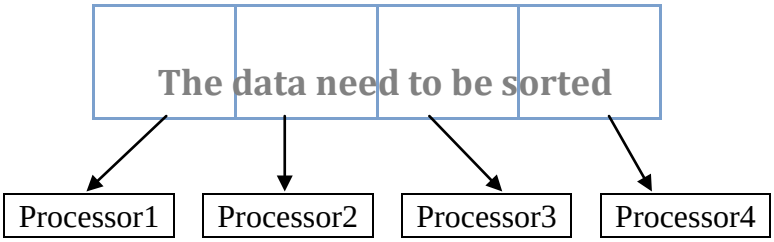


Figure13: The Domain Decomposition in theProject.

2.4.2Merge Sort

The merge sort is a sorting algorithm which follows the approach of divide-and-conquer. The methods divide and conquer usually mean that the main problem breaks into several sub-problems, using recursive methods, and then combines the results finally. [6] The approach here can be described as follows:

1. Divide: In execution of each pack, divide the data into 2 sub pack of data.
2. Conquer: In each sub pack, the data will be sorted recursively.
3. Combine: Combine each 2 sub pack data into a correct order of data pack.

There are the steps to implement the merge sort.

In general, we should divide the data for parts, although after each division, the combine will be checked and performed. It can be assumed that the combine function is called merge (), the divide function is merge-sort (). The general steps of the function should be:

MERGE_SORT (*Array*, *p*, *r*)

1 if p< r

2 q ← [(p + r)/2]

3 MERGE_SORT(Array, p, q)

4 MERGE_SORT(Aarray, q + 1, r)

5 MERGE(Aarray, p, q, r)

Here, Array means the array need to be sorted, p and q are variables initially point to the first element and last element of the left pack; r is the last element of the right pack.

Then we can deal with the combine steps, they are performed that creating two arrays, left and right. Beginning with left, if the element of left array is less than right then sort left, update the element, otherwise, do the same to the right. These are performed as follows:

merge(*Array*, *p*, *q*, *r*)
1 *num*1 ← *q* - *p* + 1
2 *num*2 ← *r* - *q*
3 arrays *L*[*num*1 + 1] and *R*[*num*2 + 1]
4 **for** *n* ← 1 **to** *num*1

5 **do** $L[n] \leftarrow Array[p + n - 1]$
6 **for** $m \leftarrow 1$ **to** $num2$
7 **do** $R[m] \leftarrow Array[q + m]$
8 $L[num1 + 1] \leftarrow \infty$
9 $R[num2 + 1] \leftarrow \infty$
10 $n \leftarrow 1$
11 $m \leftarrow 1$
12 **for** $k \leftarrow p$ **to** r
13 **do if** $L[n] \leq R[m]$
14 **then** $Array[k] \leftarrow L[n]$
15 $n \leftarrow n + 1$
16 **else** $Aarray[k] \leftarrow R[j]$
17 $m \leftarrow m + 1$

Following isa example of merge sort, the steps can be described as the figure shows:

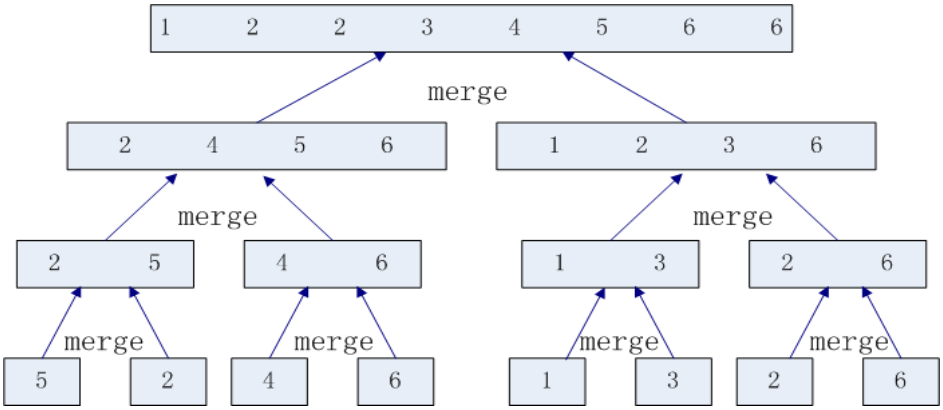


Figure 14: The procedure of merge sort [6].

The merge sort is analgorithms which will be implemented in a recursive way. The feature of the recursive is that it will be good code density, which will save the memory. However, the drawback is that it will use a lot of the function call. These massive function calls will demand the large stacks to store information when calling functions, And Calling function is also timeconsuming.

2.4.3Radix Sort

The radix sort is another sort algorithm. The radix sort begins with sorting from the least significant digit to the most significant digit, which is similar as a cord sorting. Following is an example of radix sort, which is sorting number of three digits, such as 329 and 457. The procedures arediscussed as sort cards by using three bins.

Assuming there are three bins, the cards have a mark of number, which is the 3-digit number. First we put the cards in bin0, in the order of the least significant digit, then we process the cards in bin0, sort it in the order of the second significant digit and put

it in the bin1, at last process the cards in bin1, sort it in the order of the most significant digit and put it in the bin2, finally we get the result, the sorted cards in bin2. For each column represent a bin, figure is shown as follow:

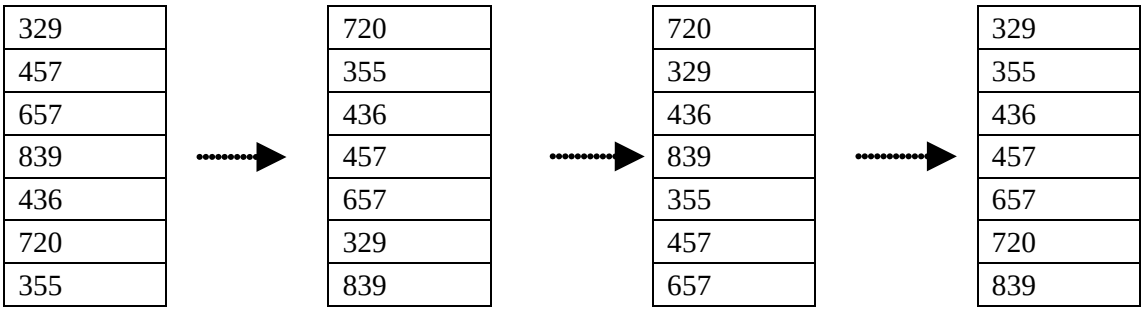


Figure 15: The procedure of radix sort [13].

The circumstance can be extended, for example more digits means more bins, and we can replace some marks with the digits, letters can be consider as the number 0 to 25.

The radix sort is easy to implement. The steps can be described as following procedures. Assuming that each element in the *n*-element array *Array*has*3* digits, and the variable n represents 3 [13].

RADIX-SORT (*A*, *n*)
1 **for** *i* ← 1 **to** *n*
2 **do** use a stable sort to sort array *A*on digit *i.*

In this project, the stable sort is referring to create ten blocks for each digit, because a digit ranges from 0 to 9. So the mainprocedures are creating ten blocks which means an array with 10elements, place the digit in these blocks means link the digit into the array. And totally there will be three arrays and 30 link list table created. The actual configuration is as below:

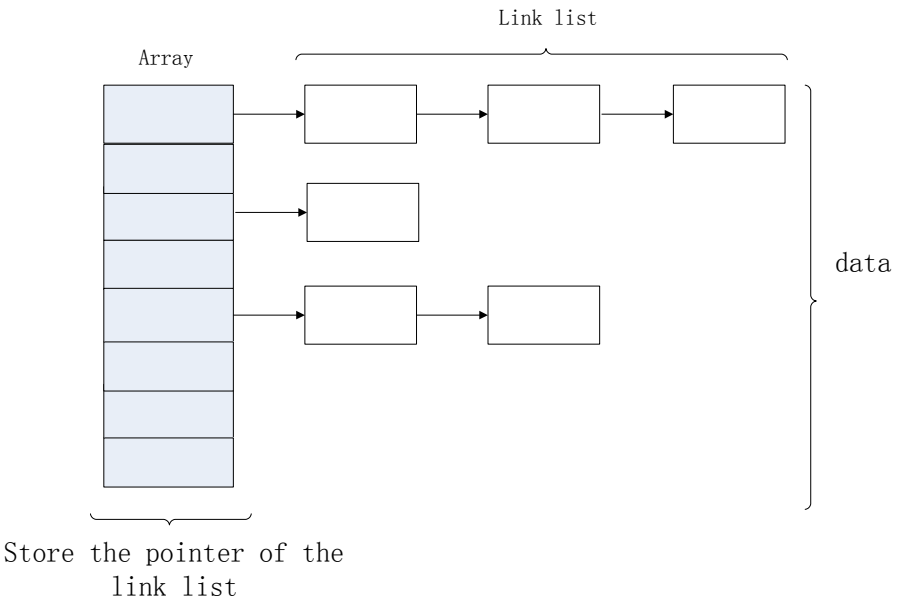


Figure 16: Data Structure of Radix Sort

So this algorithm will use a lot of memory location operations which is also time consuming.

2.4.4 The Theory of Parallel Computation

For many years, engineers try to find a best solution for implement software in parallel. With the development of the arts of VLSI, the fabrication continues to improve and it makes cost little to increase the transistors on chip. The mutli core systems hence to become dominate in current market of processors. The trend of parallel computing will be on multicore computers. So the theory of parallel computation is increasingly important.

2.4.5 Performances of the TwoAlgorithms in Parallel Computing

In parallel theory, the parallel computing is not always more efficient thanserial, because the parallelcomputing will introduce additional steps in order to make the program parallel. In a parallel system, a program which is more efficient in serial will be waste time and energy to be processed in parallel. This has become an increasingly important problem when the mutli core system is widely applied.

Some research has been done in this area.The figure 13 is from the theory of William D. Gropp. The model is based on distributed memory machines.[1] This model indicates that a parallel system is increasing the efficiency when running algorithm in small number of cores. When the number of cores increases, the efficiency will be reduced. This is because when more cores are applied, the more cost it will be introduced.

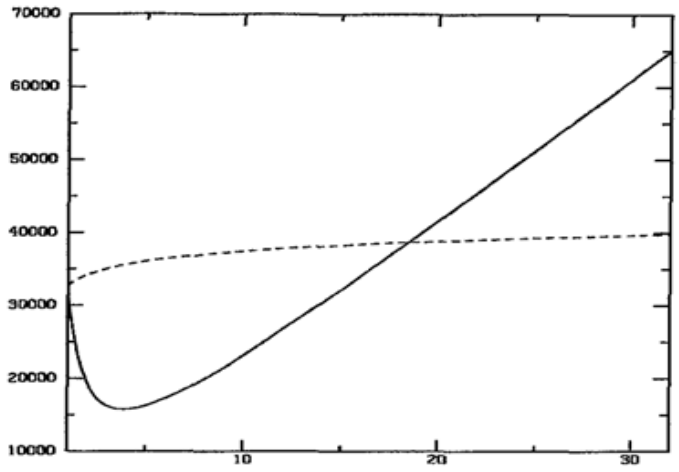


Figure 17: Comparison of serial and parallel computing, the solid line is the time for parallel algorithm; the dashed line is the time for uni-processor algorithm [1].

In my project, the ARM Cortex-A9 MPCore processor is a bus based shared memory system, and the software model only contains model of two cores. But, the basic parallel theory is much the same. Because the cost will be introduced by the parallel, and the efficiency of algorithms on two core system will be influenced by the program data input. If using the model above, applying small number of input data the cost of the parallel computing is much larger and applying more data will cause large cost, the graph will be as Figure 18. When input data increases, the more data are processed in parallel, overall time consumed is reduced compared with the cost.

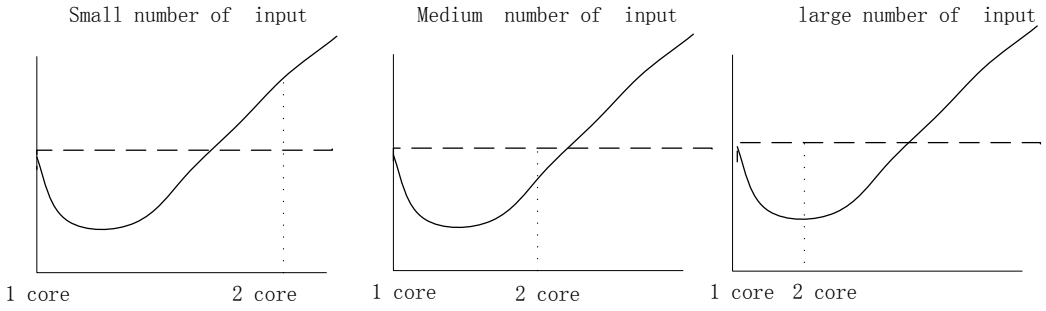


Figure 18: Core systems with different number of input.

2.4.6Cost and Benefits Analysis

The performance of the algorithms is complicated in real applications, it depends on many aspects.

- First, from the aspect of the software, how the program is organized and implemented in the processors will affect the performance. The sequential parts

of the program will reduce the overall achievements of parallel computing. And the interactions between the multicores will cost extra time for execution.

- Second, from the aspect of the hardware, the limitation of accessing the memory will cause extra time in computing.

This project will focus on the cost caused by using method of data decomposition. There are different ways to implement the approach of domain decomposition. It depends on the structure of the domains and the structure of the parallel computers, the best situation is the domains are able to split that they can exactly map on to the computers.[1]

The metrics we used to evaluate the performance of parallel computing is various. A important metric is efficiency.

The equation of efficiency E_p [1] can be defined as

$$E_p=T_1/pT_p \tag{2}$$

The equation of speedup S_p [1] can be defined as

$$S_p=T_1/T_p \tag{3}$$

The T_1 is the time that the algorithm is executed on a single processor, the T_p the time that the algorithm is executed on p processors.

So, we try to find the point that the speedup will no longer continue with the number p, which can be represented by the equation $(\partial S_p/\partial p)>0$.[1]

In this model, it should be pointed out that testing the time T_1 and $T_p$should remain the same algorithm.[1] Otherwise the result will be pointless. And the algorithm should be as concurrent as possible. [1]Here it represents that the E_p is better to be close to 1. In conclusion, it is trying to find a domain decomposition, which the E_p is close to1, and the $(\partial S_p/\partial p)>0$.[1]

From the model, we can conclude a theoretic performance of a parallel computing. [8]

$$S_p = \frac{1}{(1-f)+\frac{f}{n}} \tag{4}$$

The f represents the ideal parts of program which can be executed in parallel, then is the number of processors.

However, in the real applications of shared memory systems, it is not easy to get exact model for the desired solution [10] . First, the cost in parallel computing is caused by various aspects. In the aspects of software, the safe access to the memory will cost less time. For example, the terms of costs is performed as barriers and critical sections [10]. The hardware cost may lie in the bus limitation and the speed limitation of communications among each core [10].

So, it can be shown that not always parallelism is more efficient to execute a specific program. For a shared memory system, William D. Gropp proposes the model, which may helpful to estimate the best solution [10].

Chapter 3Project Implementation

The project implementations will be introduced in this chapter. In general, the project contains two main parts. First, it configures the Cortex-A9 MPCore for safe parallel computations. It then researches some algorithms and implements them successfully on this parallel system. Second, it will test the time information of these parallel algorithms, and investigate the models of the cost and benefits of the parallel algorithms.

3.1 The Design of the Program

In this section, the ARM RealView Development Suite 4.1 Professional (RVDS 4.1 Pro) will first be introduced. Second,the details of the platform configured on the Cortex-A9 MPCorewill be discussed. Finally, the details of the two sort algorithms and the running flow of the algorithms on the platforms will be examined.

The project is based on mixed C and assembler. This assembler is chosen because the configuration of Cortex A9-MPCore in fact is manipulating the hardware. So it is needed to enable or disable lots of registers on Cortex A9 MPCore, which is only accessed using ARM assembler. The C language is chosen is because the C is high level language and easy to program complex structure program.

3.1.1 RealView Development Suit 4.1 Professional

This whole project revolves around implementing sort algorithms on a Cortex-A9 MPCore system, and then analyzing the results. However, because of the limitations of equipment, this project is not working with an actual Cortex-A9 MPCore board.This project is instead applying a software simulation, which is based onReal-Time System Model (RTSM). The RTSM is integrated into RVDS 4.1 Pro.

The RealView Development Suite 4.1 Pro is a software development that supports almost all the ARM cores. It contains parts includingan ARM compiler, an ARM profiler, an ARM workbench IDE and an ARM Debugger. It supports editing, compiling, profiling and debugging the source code.

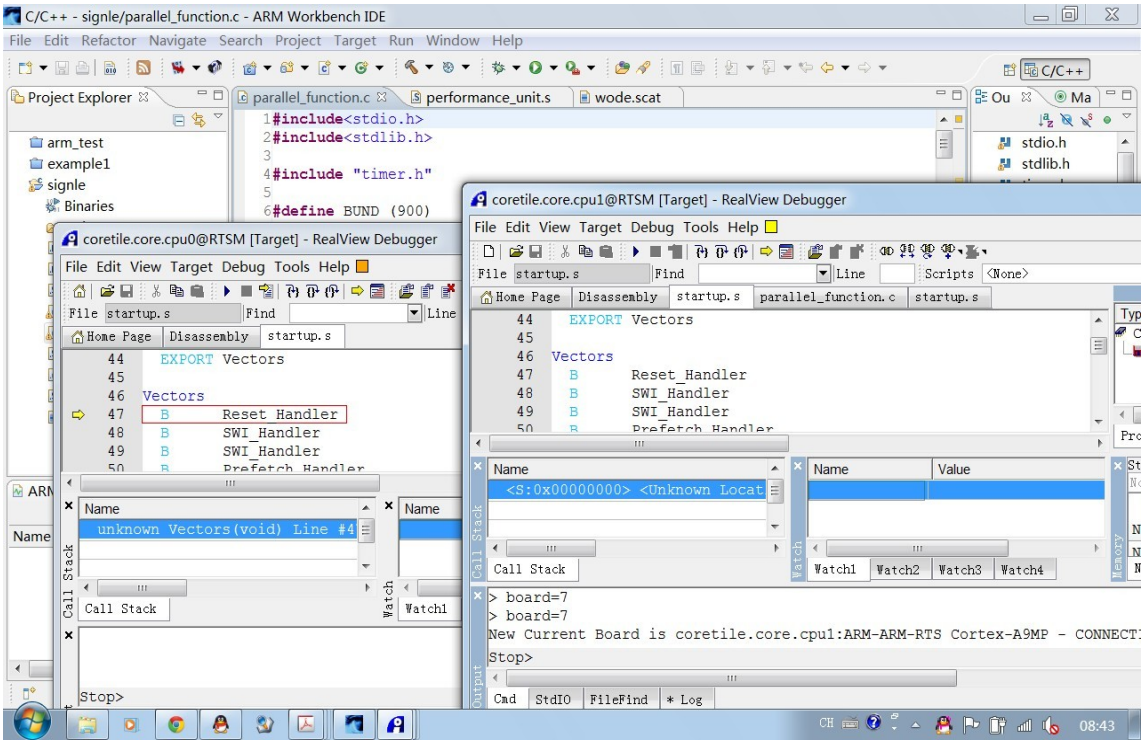


Figure 19:ARM RVDS 4.1 Pro.

Additionally, the RealViewdebugger provides several groups of target configurations. When connected to these configurations, the code can run on these target configurations. RTSM is a kind of target library.

In this project, although the Cortex-A9 MPCore has four cores, the RVDS 4.1 Pro only supports the target configuration that debugs two cores. So the model applied in this project is a Cortex-A9 MPCore with two cores. The configuration is software Real-Time System Model, which will be discussed later for its differences with real systems. The actual target connection is as follows:

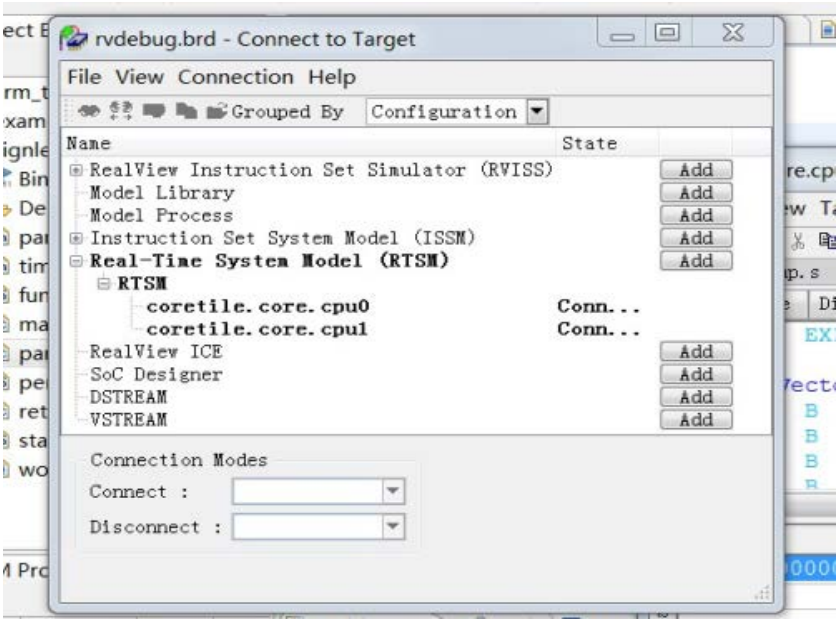


Figure 20: Cortex-A9 MPCore target connection.

3.1.2 Configure the Platform

The platform is built on RealView RTSM Cortex-A9 MPCore X2, which is a two-core software model of a RealView debugger. However, the model needs to be configured suitably for parallel computing.

From the background introduction, there are several aspects that need to be considered while doing the configurations.

1. The SMP configurations;
2. The memory configurations;
3. The caches coherence;
4. The interrupt configuration;
5. The other environment settings, such as stack, heap, private memory for data or private configurations.

Some of these problems are independent, some are not. The solutions will be anaylsed in this project.

As discussed in previous chapter, the general idea of thesort algorithms is introduced, and later the design flow will be analyzed with the actual code.

3.1.3 Set the Stack and Clear the Invalid Data

First, for running an ARM system, the pre-configuration defines every stack for each mode. In this project, the SYS mode and the IRQ mode will be used, so the stack address and capacity will be set.

As is introduced in the section about caches, some of the devices have the same problems: random garbage remaining in the processors may prevent the processors from correctly working. So they should be invalidated. The design flow is as below.

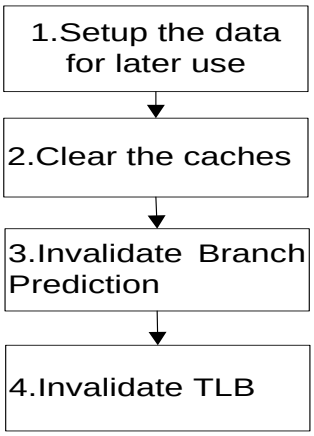


Figure 21:Set Stack and Clear Invalid Data.

3.1.3 Implement the Memory Map

In this step, the memory should be configured. As discussed in section 2.5.1, the main procedures implemented are the virtual memory and scatter load.

The memory design is as follows:

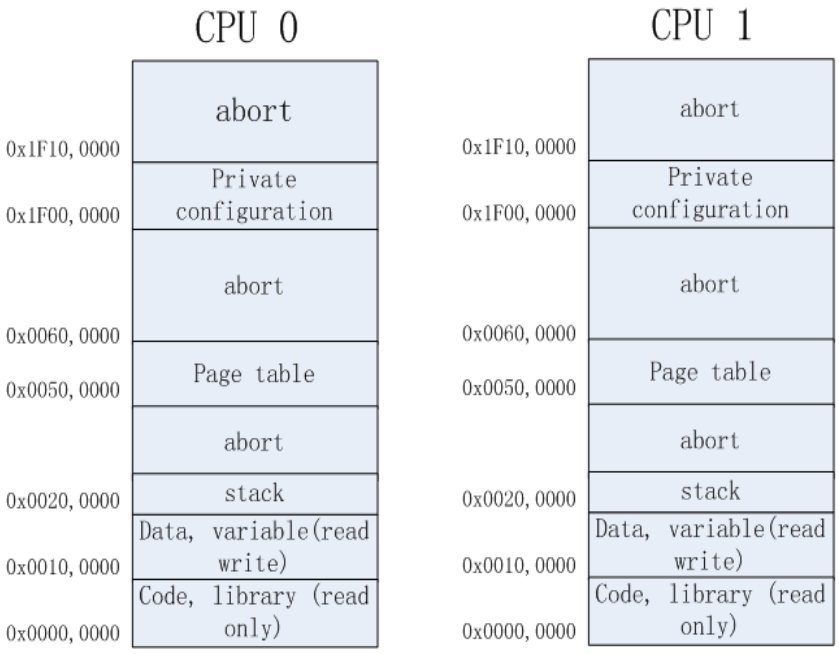


Figure 22: Expected CPU Memory Map.

So, the first 0x0000, 0000 – 0x0010, 0000 puts the code and library, which is a read only code. The 0x0010, 0000 – 0x0020, 0000 is the section to store data, variables, which is writable code. The page table section is for storing page tables, and the private configuration area is for storing the configurations of the cores, for example, the GIC configurations.

Then the scatter load should be applied to inform the Cortex-A9 MPCore of the memory configuration. This step is done by setting the path of the scatter file to the RVDS. The scatters file and set path procedure are as follows:

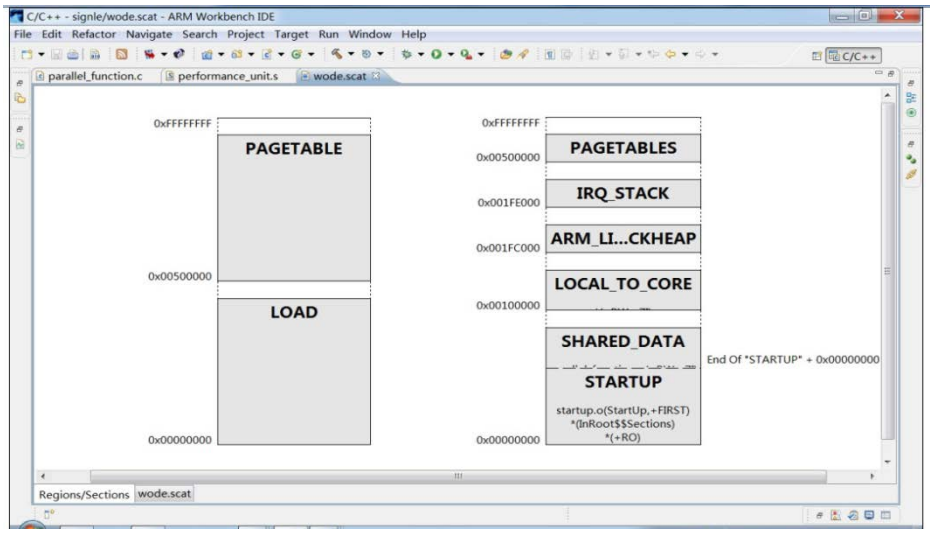


Figure 23: Introduce the Scatter File.

The scatter file should be introduced by setting the path:

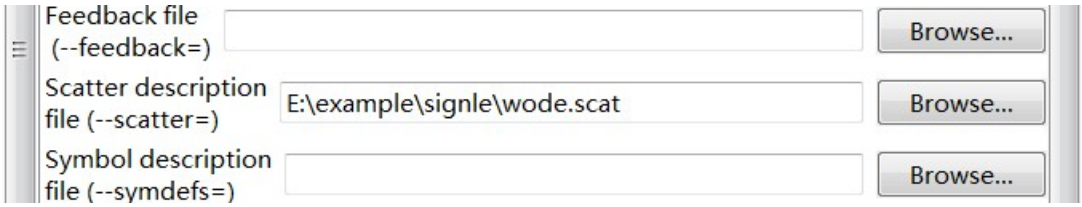


Figure 24: Set the Path for Scatter File.

Another problem here is implementing the virtual memory.In the ARM Cortex-A9 MPCore system, only one main memory exists. Concerning figure 22, the two cores want the same view of the memory;however, the memory of the different cores will overlap each other.

In this project the code and library part could be shared because they are not changed when executing. But the data and variables should not be overlapped because each core has its own data that should not be overwritten. Virtual memory will be needed to reorganize the memory. Here, CPU0 and CPU1 are the virtual memory, but the MMU will automatically translate the addresses to the physical memory.

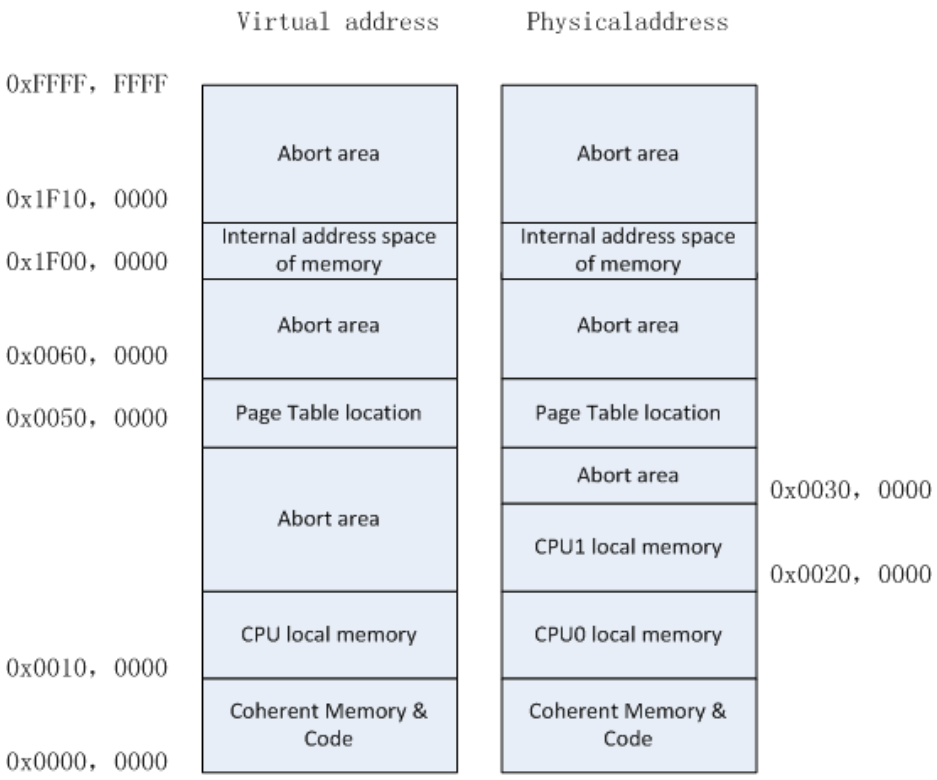


Figure 25: Virtual memory to physical memory.

The way to implement this is described in the procedures below.

This project is using 1 MB block memory mapping. It is also called first level virtual memory, which indicates it only maps once. As shown by figure 25, the ARM memory is a total of 4 GB. It can be divided into 4096 blocks; the idea of the first level virtual memory is to reorganize these 4096 blocks and the correlation of addresses inside the block does not change.

The translation procedure is as follows in figure 26.

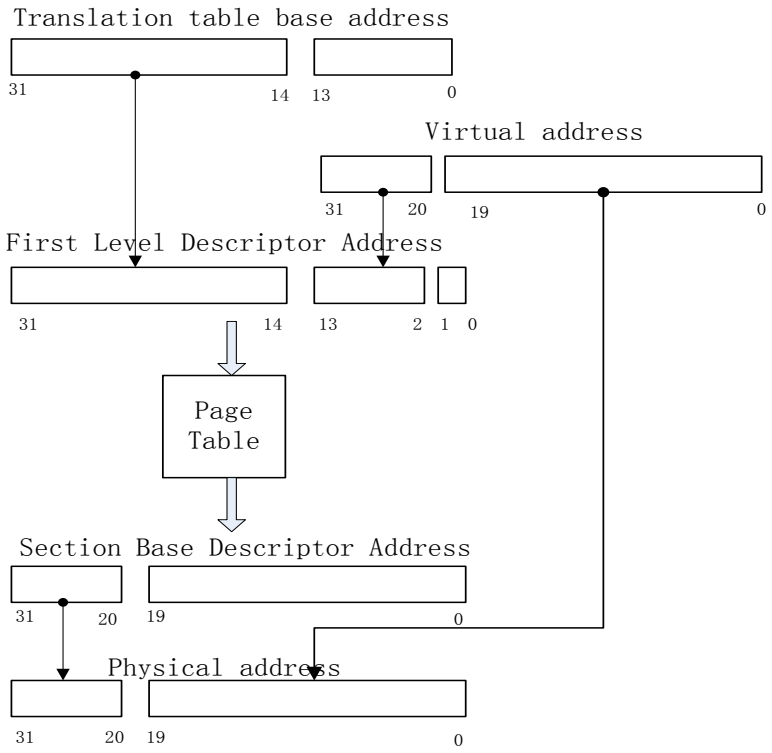


Figure 26: Virtual address to physical address.

For example, the virtual address is 0x0020, 0001.According to its 1MB mapping, it indicates the address between 0x0020, 0000-0x002F, FFFF will behave the same.

First the base address of the table is needed, which is predefined as 0x0050, 0000. Then a page table is needed during the process of virtual memory. The table is a block of memory thatcontains the mapping information. It is like a collection of pointers.

Then, as in figure 26, bit 14 to 31 of the base address, plus bit 20 to 31 is used for the index to search in page tables. It is very smart that the 0 and 1 bit is used as a signal bit to indicate that it is level 1 virtual memory, and at the same time the 2 bits are also offset because one page in a page table is 4 bytes.

So we get 0x0050, 000A (the last two bits are 1 and 0) to search the page table. From the page table, the section base descriptor will be searched.

This is the important mapping step; in this step the virtual address is used and the mapped section address is found (in section base descriptor). All the information needed is in this section base descriptor. The section base descriptor is defined by the programmer, who can choose which section to map to. The section base descriptor is combined with two parts. The first part is 31 to 20 bit, which can represent from 0 to 4096. These bits define which block is mapped to.The second part is controlling the bits that define the access policy and the caches policy.

The last step is getting the mapped block address bit plus the remaining part of the virtual address, then the physical address is found.

In this project the mapping principle is that for shared memory, just map to the same address; private memory is mapped to (same address + CPU number *1MB). The 0x0020, 0001 is in private memory. For CPU 0, it will map to the same block, and the result will be 0x0020, 0001. For CPU 1, it will get 0x0030, 0001.

By applying the virtual memory, the purpose is that the CPU0 and CPU1 can use the memory as in figure 22, while the actual memory map is shown in figure 25.

By knowing the procedures above, the configuration in the program is as follows:

- Firstly, build the page tables for each core, fill it with the pages and write the base page table address to a special register.
- Secondly, write the 0 to 19 bits of the section base descriptor (page); it is all about setting bits for cacheable and access permissions, which require setting the domain access first.
- Finally, enable MMU, and then the virtual memory is implemented.

From now on all addresses formed by the processor will be treated as virtual addresses and have to be translated by the MMU. If the translation failed, it will show data abort fail. The steps in the code are as follows:

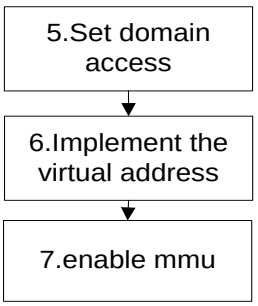


Figure 27: Code Flow for Memory Configuration.

3.1.4 Prime Core Starts

In this step, the configurations for the two cores show differences.

In the SMP system of ARM, usually a prime core is needed to do the tasks before parallel computing. According the parallel theory in section 2.4.1, the first step for parallel computing is partitioning and deciding the tasks of each core. This step starts from here. In this project there is a prime core CPU 0, which does the partitioning and sends the data to the cores, which are CPU 0 and CPU 1.

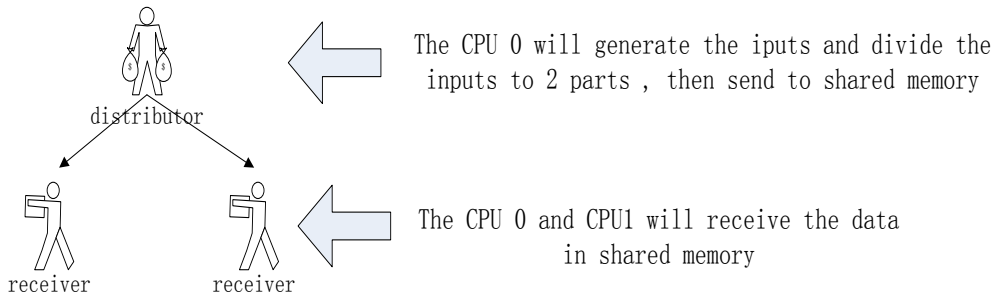


Figure 28: the Task Distribution in SMPSystem

The CPU 0 should be configured to work before the parallel computation starts, while the CPU 1 will do some configuration and temporarily ‘sleep’ (power down).

In the previous two parts, the memory configurations and the other environments setting are finished, and there are still the problems remaining to be finished for the configuration. They are the SMP configurations, the caches coherence and the interrupt configuration. These configurations will be almost finished in this step.

First, the SMP configuration is applied by setting the SMP bit in a special register in ARM.

The caches coherence is introduced by four steps:

1. Memory is marked as Normal Shared in the translation table;[19]
2. Data cache is enabled;[19]
3. MMU is enabled;[19]
4. SCU is enabled;[19]
5. The SMP bit is set in the SCU. [19]

The first condition is done in writing the 0 to 19 bits of the section base descriptor. The MMU is activated in the step of implementing virtual memory. The SCU is enabled by accessing a special register. The data caches enabled will be finished later, right before the jump to the main function in c.

The interrupt configuration is needed to configure the interrupt distributor and the interrupt interface. These partsare complicated and will bediscussed independently.

Another concept to be aware of is that some of the devices are local, which means they should be both initialized for two cores; some of the devices are external which means they only need to be initialized once.

The CPU 0 and CPU 1 should do the configuration respectively as figure below:

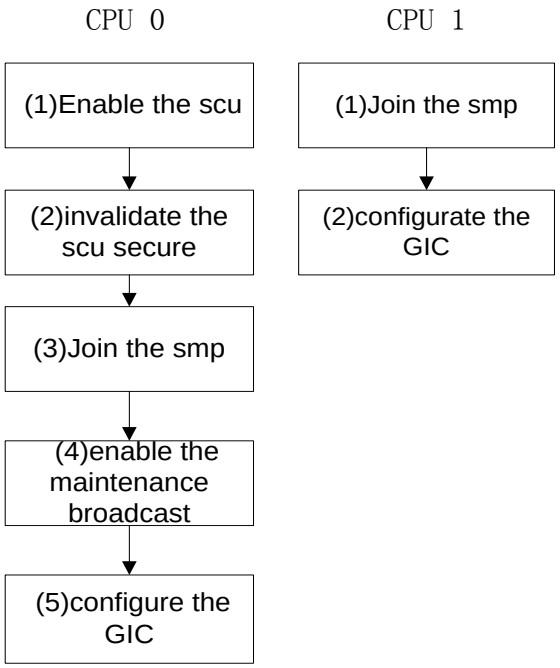


Figure 29: CPU 0 and CPU 1 initialization.

With the exception of enabling the data caches, all of the requirements are reached. These above steps except the configuration of GIC are all easy to achieve. The GIC will be introduced in the next part, and after these steps CPU 1 falls asleep until the signal is given. CPU 0 starts to divide the data and send them to two cores.

3.1.5 GIC Configuration

In Cortex-A9 MPCore, the interrupt is controlled by the device generic interrupt controller (GIC). From the last section, CPU 1 is falling asleep. Falling asleep in fact means CPU 1 is continuing to execute the instruction NOP. CPU 1 cannot do anything to wake itself when it is stuck executing NOP, so interrupt is needed.

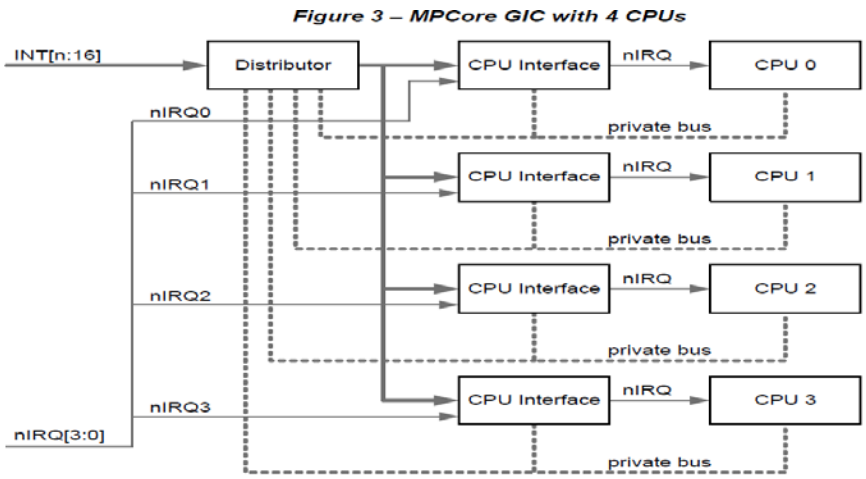


Figure 30: MPCore GIC with 4 CPUs [22].

At this step the GIC and configuration should be done. The Cortex-A9 MPCore GIC structure is shown in figure30.

The GIC contains the distributor and the CPU interface. A GIC allows you to configure an interrupt's priority, enable or disable it, disable all interruptsbelow a given priority, configure whether it can be pre-empted (interrupted by anotherinterrupt), and optionally what priority is required to pre-empt. [22]

From section 2.3.4, it is known that the IRQ interrupt will be used. Here it must be decided which interrupt is activated and which level priority of interrupt can be accepted by the processor. The design flow:

1. Enable the GIC distributor by setting the Distributor Control Register. [24]
2. Enable and set priority level bysetting and Priority Level registers. [24]
3. Enable the CPU Interface by setting the CPU Interface Control Register.[24]
4. Clear the Priority Mask by setting Priority Mask Register.[24]
5. Set priority of IRQ Interrupts by setting the Enable Set and Priority Level registers.[24]

3.1.6 Dividing Data

In this step, the algorithms start to run on this platform. The first step of the program is that the program will generate the data and the prime core will divide the data. The input data are all generated by using the random function in C. However, the program can apply two methods of dividing approaches. The two approaches are shown in figure 31.

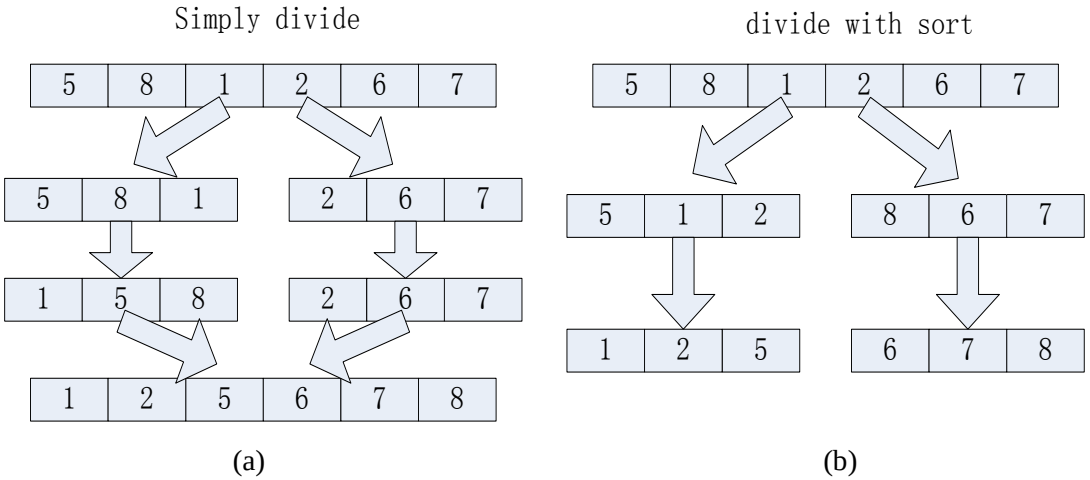


Figure31: Two DivideApproaches for the Parallel Algorithms.

- The first approach is simply dividing the data.After that the prime core divides them and sends them to separate cores.In this circumstance, the two results in the two cores will be random, so the last step should be a merge step, which is an additional sorting step operating in one core. This is called “simple divide” in this thesis.
- The second approach is that the dividing part is carefully undertaken. The prime core divides the data into two groups with data in one group larger than the other group. In this circumstance, the last step is not necessary. This is called “divide with sort” in this thesis.

In fact, these two approaches each have advantages and drawbacks. While due to the independence of the input data, they all can be implemented on the parallel system.

The simple divide method is more complicated from the aspect of program flow, it splits the merge step and the divide step. Compared with divide with sort, it has more store steps. (Store is actually the most time consuming operation in executing, especially with large data.)

However, the divide with sort method has drawbacks in that it is difficult to decide how to implement the first sort, because it should balance the complexity and the accuracy. The First sort need to be not very complicated but can divide as the requirements.

Taking figure 31 as an example, the simple divide will need more steps than divide with sort. As it will need a store step both in its first step and fourth step, it will take time. The divide with sort is combining them in one step. In divide with sort, the first sort is to divide the data into two groups based on the reference data 5. But if the reference number is not chosen correctly, the divide with sort will send more data to a specific core, and this will increase the whole execution time.

In this project, the two methodsare all implemented and compared. In this step, the program is generating the input data, and the prime core CPU 0 is dividing this input data with one of the methods. The code flow in the program is shown in figure 32.

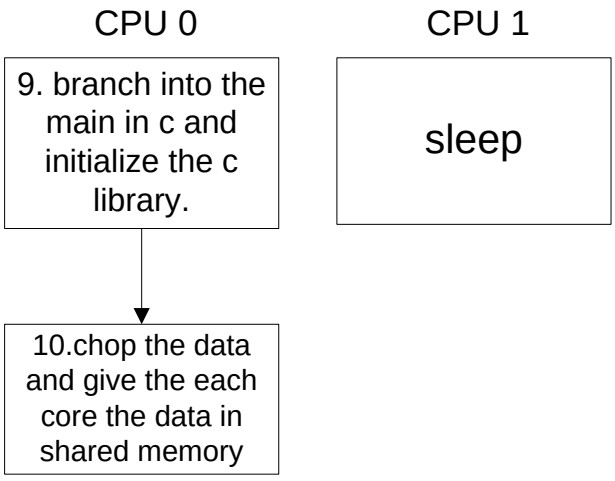


Figure 32: The Divide Procedure in Prime Core.

3.1.7 The Synchronization of Two Cores

In this step, the two cores need a synchronization step, because it is the first time CPU 1 jumps into C code. In last section CPU 1 isasleep, which means it has not done the step of initializing the C library. It is a time consuming step. In a real situation, this C library initialization is finished by the operating system or there may not be this step. Synchronization is introduced in order to make sure the two cores can begin to start the parallel computing step. It is also for the convenienceof studying the behavior of the program running on a parallel platform.

The procedure in the code is that CPU 0 sends the interrupt to wake CPU 1, and then CPU 0 goes to sleep until CPU 1 sends the interrupt to wake CPU 0.

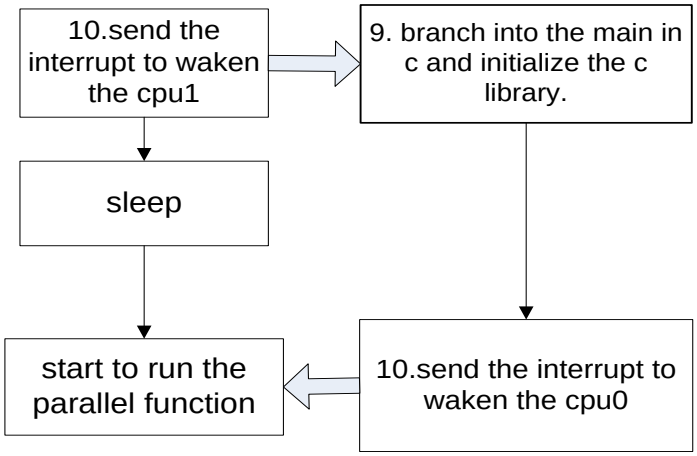


Figure 33: The Synchronization Step.

From this step on, the two cores are back in the same start line to run parallel computing.

3.1.8The Sort Process

In this step, CPU 0 and CPU 1 are ready to do the sort algorithms.

Here, two sort algorithms and two different parallel methods are applied. The two different parallel methods are referred to as the simple divide and the divide with sort in figure 31. The sort algorithms are brieflydiscussed earlier in the chapter.

- The merge sort chops the data, and then sorts the data when merging them together. This algorithm is written in a recursive way, which is improved in code density.
- The other is the radix sort, which is like a bin sort processthat puts the data into the bin according to the conditions. This algorithmcreateslot data structures, which means it will use the system stack a lot.

The procedure can refer to figure 14 and figure 15. In the result analysis, the code flow will be discussed again. Here, just a simply code flow:

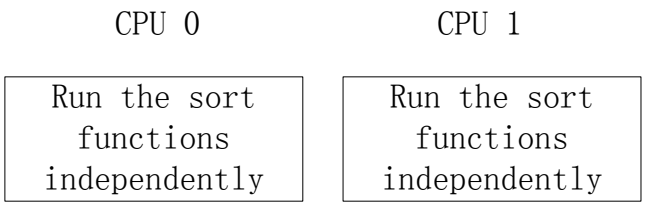


Figure 34: Sort Process

3.1.9 The Ending Step of the Program

In this Cortex-A9 MPCore system, CPU 1 and CPU 0 are not perfectly working at the same speed, and the input data for each core is also not always the same. Therefore, the problem will happen in the final step of parallel computing. The core that is faster than the other will end first, and the whole system will shut down with the faster core. In the code, a flag is set to decide if the whole system should shut down. If faster core finishes, the program willmake it sleep. The code procedure is as follows.

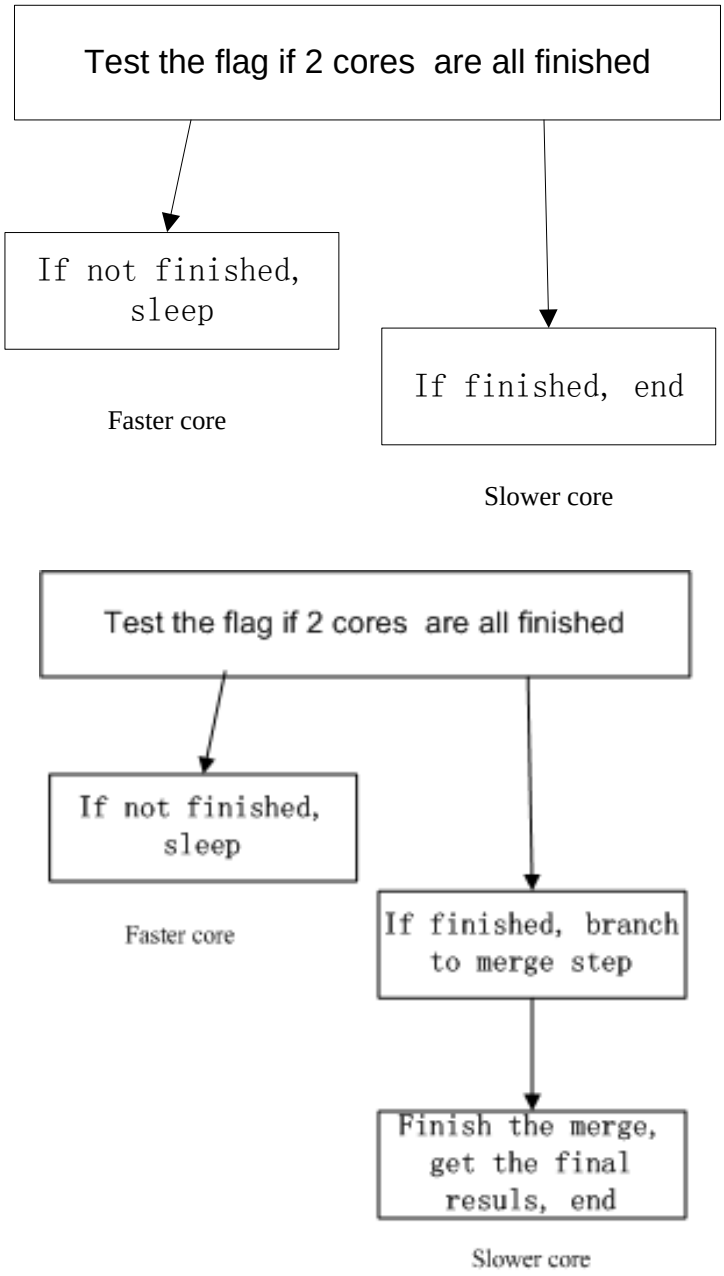


Figure 35:The Ending Step of the Code

3.1.10thePerformance Monitor Unit

After the platform and algorithms are implemented, the last thing is to add the test tools. The Cortex-A9 MPCore integrates the performance unit on its processor. It is a very powerful tool.The performance monitor contains several parts:

- A cycle counter.
- A number of event counters.
- Controls for enabling and resetting the counters, flagging overflows, and enabling interrupts on counter overflow. [23]

However, the model used in this project is a real time system model (RTSM), which only supports the cycle counter. The cycle counter counts cycles when the program is executing, and this can be used to identify the time.

The cycle counter is activated by manipulating the special registers in Cortex-A9 MPCore. In this project, these operations are written with dedicated functions. When testing is needed, there are two ways to monitor the instruction cycles have been passed.

A function call can be inserted to do a test. The result can be printed on output console. The result is as below:

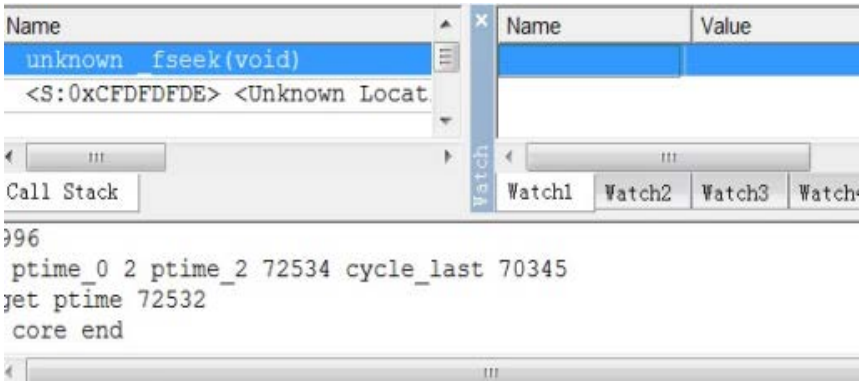


Figure 36: Example of Testing Cycles in Code.

Or it can be directly accessed in a performance unit plane in the RVDS debugger.

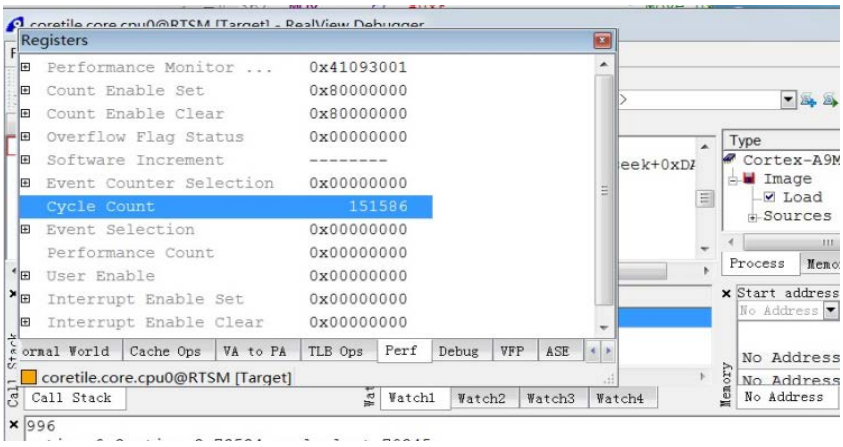


Figure 37: Example of Testing Cycles in Code

3.2 Result Analysis

In this section, these sorting algorithms running on Cortex-A9 MPCore are evaluated and some results of the running time are observed and analyzed. There are a total of four sort algorithms implemented, tested and analyzed in this section. They are simple divide and divide with sort in merge sort, and simpledivide and divide with sort in radix sort.

3.2.1 Program Analysis

In general, the code flow of this project is as follows:

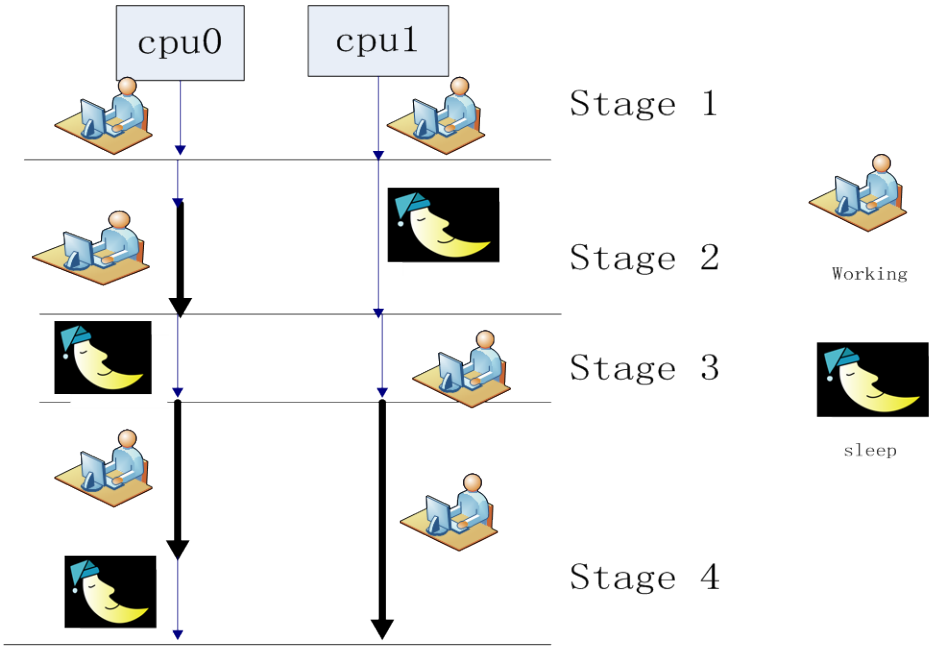


Figure 38: Code Flow of Sort Algorithms.

In this graph, the thin line indicates the code is not related to the executions of sort algorithms. The wider line is the opposite.

- In stage 1, the two cores are doing the boot steps, which isinitialization and it is not related to the executions of sort algorithms.
- In stage2, first is synchronization, and then core 0 chops the data.
- In stage3, core 0 is waiting for core 1 to do the setup of the C library.
- In stage4, the two cores will wait for each other to finish the task. The faster core will go tosleep, and the slower core will finish its work or finish the last merge step.

In this analysis, the idea is to identify the overhead that is brought by parallel computing and the time saved by using parallel computing. The overhead and time

saved are all presented in the running time of the programs, which is the wider line shown in figure 38.

For the overhead part of the program, the program is based on RSTM, which is a software model. Some multicore systems will encounter the problem that multicores access the same memory, which does not happen here. So, the overhead mostly lies in the program structure.

The overhead in this program is mainly the data chopping part, which means it divides the data and sends the data to the two cores. Indeed,this step is not applied when the program is in serial. However, if doing the divide with sort, the algorithms will save time in the merge. So in sort with merge, the time overhead will be subtracted from the last merge step.

For the time saved part of the program, the truth is that time is saved by sending data to the other core. For example, if CPU 0 needs to sort 10 pieces of data, it will save time by sendingtwo pieces of data to CPU 1.

3.2.2Simple Divide and Divide with Sort in Merge Sort

In this step, different input data will be supplied to the merge sort under the methods of simple divide and divide with sort. Then different stages of the time cost will be tested, and further investigation will be applied based on these results.

The procedure of simple divide is that the program will generate a different number of input data. They are all supplied to the merge sort. In simple divide, the sort algorithm can be considered in three parts:

- The simple divide part. This simple divide divides the input data into two groups. It is not needed in serial computing.
- The parallel computing part. This is actually parallel computing the data.
- The last merge part. This step merges the two groups of data from two cores in the correct order. And this step can be considered the part of program that cannot be parallel computed.

All these steps are calculated in execution cycles.

number of input	divide_cycle	CPU0_p_cycle	CPU1_p_cycle	last_m_cycle
10	2857	5029	5036	174
20	3042	10219	10307	377
30	3227	15244	15541	533
50	3597	26273	26029	909
100	4716	52850	53091	1821
150	5641	79959	80153	2697
200	6566	107088	107723	3621

Table 1: Instruction Cycles of Merge Sort with Simple Divide.

In this table, it is convenient to find out that the waste of parallel computing is part of the divide cycles. The divide part is shown in the figure below:

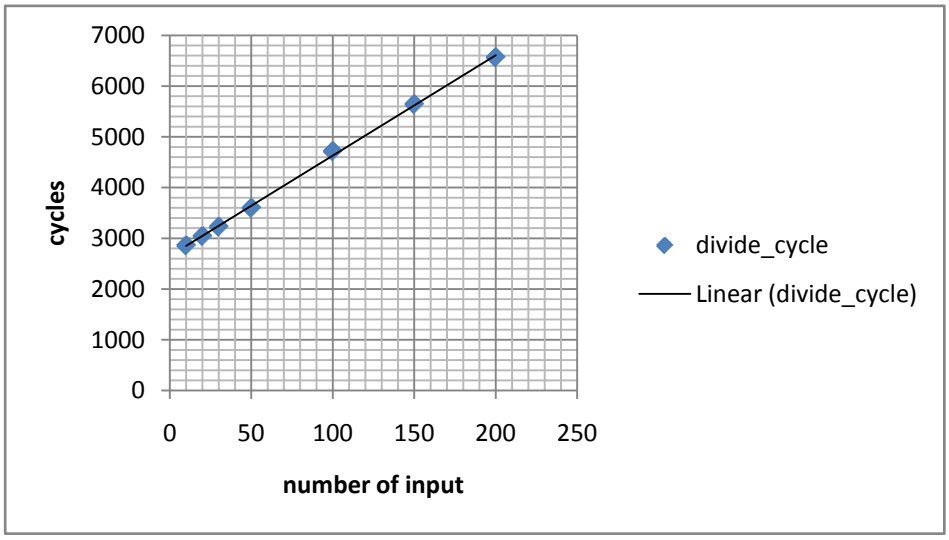


Figure 39: The Cycles of Divide Part.

From the chart above, the divide cycle can be estimated as

$$y = 20x + 2627$$

The y represents the divide cycles; the x represents the number of inputs.

Associated with this equation, the cost of parallel computing can be estimated in this project. And the time saved in this project is by applying 2 cores in parallel stage.

From table 1, the parallel stage can be described as follows:

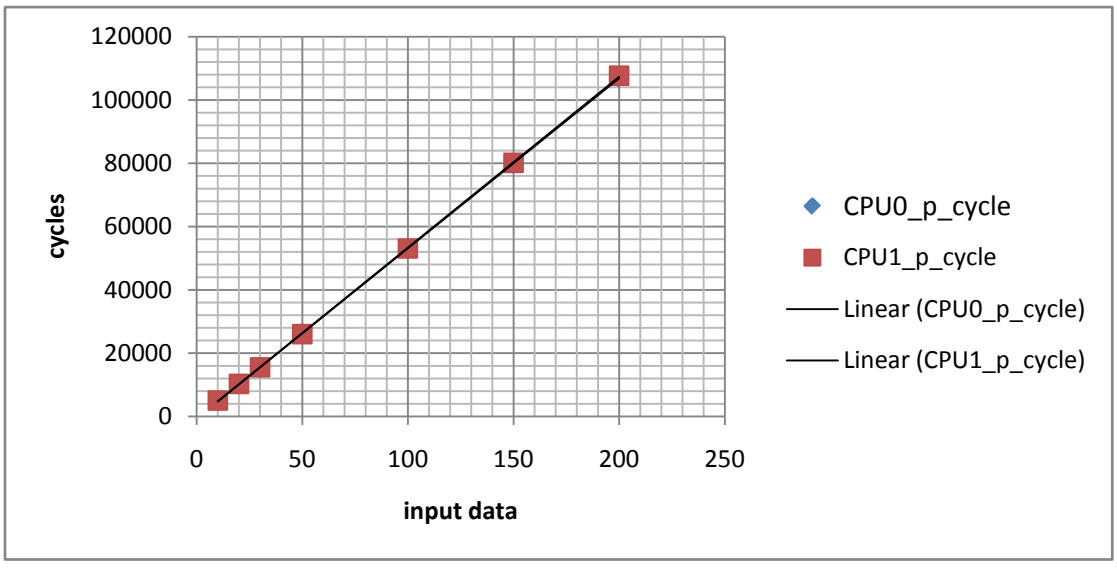


Figure 40:Cycles ofParallel Stage of Merge Sort with Simple Divide.

From figure 40, in the parallel stage the time consumed is proportional to the input data. So the time saved for a CPU is proportional to the reduced input data. In this merge sort with simple divide, the number of reduced input data is half of the total inputs.

From the chart above, the saved cycles can be estimated as

$$y = 560(\frac{x}{2})$$

The y represents the time saved cycles; the x represents the number of inputs.

After the above analysis, the cost of parallel computing and the benefits of parallel computing are identified. When parallel computing is applied, the consideration is whether parallel computing increases the efficiency. The evaluation of the cost and the benefits in applying parallel computing can show the efficiency of parallel computing. If the cost is greater than the time saved, parallel computing is increasing the efficiency. If not, the result is reversed.

The results can be used in further investigations. If the time cost is greater than the time saved:

$$20x + 2627 > 560(\frac{x}{2})$$

Parallel computing is decreasing the efficiency, and the result is that:

$$x < 10.1$$

This shows that when the input data is smaller than 10, parallel computing is in fact decreasing the efficiency.

The procedure of divide with sort in merge sort is that first the program will generate a different number of input data, then CPU 0 will divide the input data into two groups based on a reference number, for example, the input larger than 50 in one group and smaller in another group.

In divide with sort, the sort algorithm can be considered as two parts:

- The divide with sort part.
- The parallel computing part. This step actually parallel computes the data

The steps can be calculated in execution cycles:

number of input	divide_cycle	CPU0_input	CPU0_p_cycle	CPU1_input	CPU1_p_cycle
10	1478	3	3282	7	7449
20	1663	8	8285	12	12659
30	1849	12	12292	18	18981
50	2219	22	22842	28	29837
100	3239	49	51693	51	54731
150	4160	78	82910	72	77668
200	5087	101	107871	99	107418

Table 2: Cycles of Divide with Sort in Merge Sort.

The difference with the simple divide is that the divide with sort reduces the last merge sort step, but the divide with sort cannot guarantee the perfect parallel of input data. This is because the input data is divided according to the reference data. In this sort algorithm, the input data ranges from 0 to N, and the reference is N/2.

In this program, the input data is not ideally parallel, so the degree of parallelism can be defined as the parallel part of the input divided by the total part of the program. The parallel part of the input is twice the size of the smaller input. For example, the first total input is 10, which has 3 executed in CPU0 and 7 executed in CPU1, so 6 out of the input 10 is actually parallel computing, and the other 4 are actually running serial. In this example the degree of parallelism is 3, which is a smaller input; multiply it by 2 then divide by the total input. In simple divide, the degree of parallelism is 100%.

number of input	divide with sort	P _stage_cycle	simple divide	P _stage cycle
10	6/10=60%	7449	100%	5036
20	16/20=80%	12659	100%	10307
30	24/30=80%	18981	100%	15541
50	44/50=88%	29837	100%	26273
100	98/100=98%	54731	100%	53091
150	144/150=96%	82910	100%	80153
200	198/200=99%	107871	100%	107723

Table 3: Parallel Stage Cycles of Divide with Sort and Simple Divide.

Compared with the simple divide, a conclusion that can be made is that the algorithms are more efficient because they are highly parallel.

For the same process, the cost and benefits of parallel computing can be evaluated for this merge sort.

First, the cost should be modified for this divide with sort; indeed, divide with sortreduces a last merge step. This can be subtracted from table 1. So the actual cost is the divide cycle in table 2 subtractsthe last merge step in table 1.

The table and the line chart are as follows:

number of input	divide_cycle	last_m_cycle	cost_cycle
10	1518	174	1344
20	2958	377	2581
30	4398	533	3865
50	7278	909	6369
100	14422	1821	12601
150	21510	2697	18813
200	28654	3621	25033

Table 4: The Cost of Merge Sort in Divide with Sort.

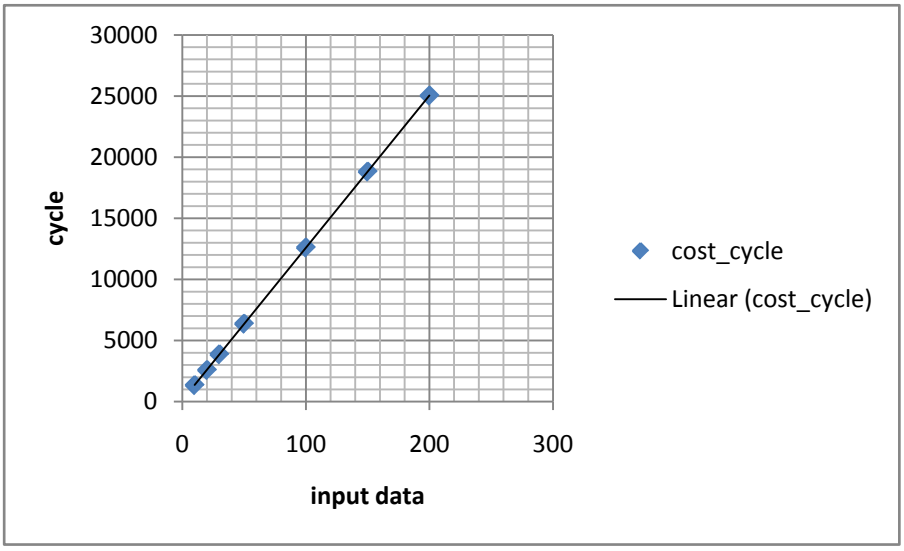


Figure 41: The Cost Cycles of Merge Sort in Divide with Sort.

The cost can be estimated:

$$y = 124x + 169$$

The y represents the divide cycles, and the x represents the number of inputs.

When listing all the input data and related cycles in parallel stages, the chart will be as follows.

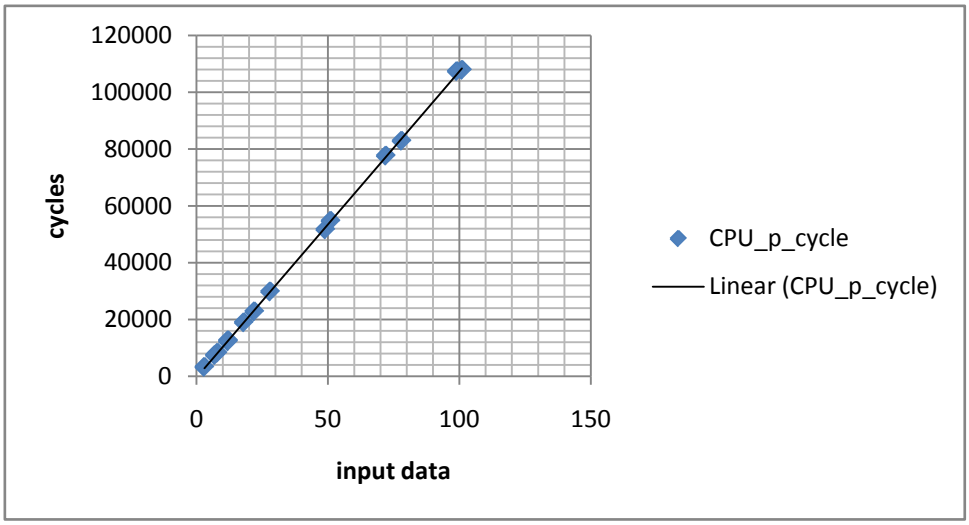


Figure 42: Cycles of CPU consumed in parallel stage.

From the chart, it is obvious the time consumed is proportional to the number of inputs. So the time saved is estimated as in simple divide:

$$y = 1034(x/2)$$

The y represents the time saved cycles; the x represents the number of inputs.

So, the result applies in this circumstance. If the time cost is greater than the time saved:

$$124x + 169 > 1034(\frac{x}{2})$$

Parallel computing is decreasing the efficiency, and the result is that:

$$x < 0.43$$

This shows that whatever the input data is, parallel computing increases the efficiency.

3.2.3 Simple Divide and Divide with Dort in Radix Sort

As in merge sort, different input data will be supplied to the radix sort under the methods of simple divide and divide with sort. Then different stages of the time cost will be tested, and more investigations will be applied based on these results.

The procedure of simple divide is that the program will generate a different number of input data to the radix sort. The sort algorithm can be considered as three parts:

- The simple divide part. This simple divide divides the input data into two groupsIt is not needed in serial computing.
- The parallel computing part. This actually parallel computes the data.
- The last merge part. This step merges the two groups of data from two cores in the correct order. This step can be considered the part of the program that cannot be parallel computed.

All these steps are calculated in execution cycles.

number of input	divide_cycle	cpu0_p_cycle	cpu1_p_cycle	last_m_merge
10	1720	6414	6320	197
20	1885	11848	12133	366
30	2050	18358	17640	569
50	2380	30288	29113	925
100	3302	60298	58340	1829
120	3632	72532	70222	2189

Table 5: Instruction cycles of radix sort with simple divide.

From the table above, the instruction cycles of the divide part can be shown in figure43.

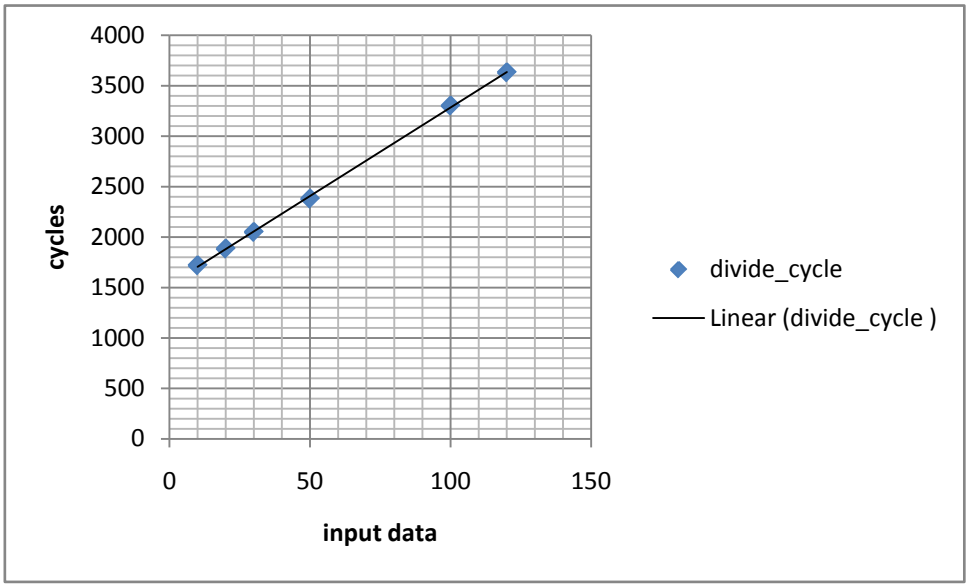


Figure 43: The chart of the divide part.

From the chart above, the divide cycle can be estimated as:

$$y = 17.6x + 1522$$

The y represents the divide cycles. The x represents the number of inputs.

Associated with this equation, the cost of parallel computing can be estimated in this project and the time saved in this project by applying two cores in parallel stage.

From the table 5, the parallel stage can be described as follows:

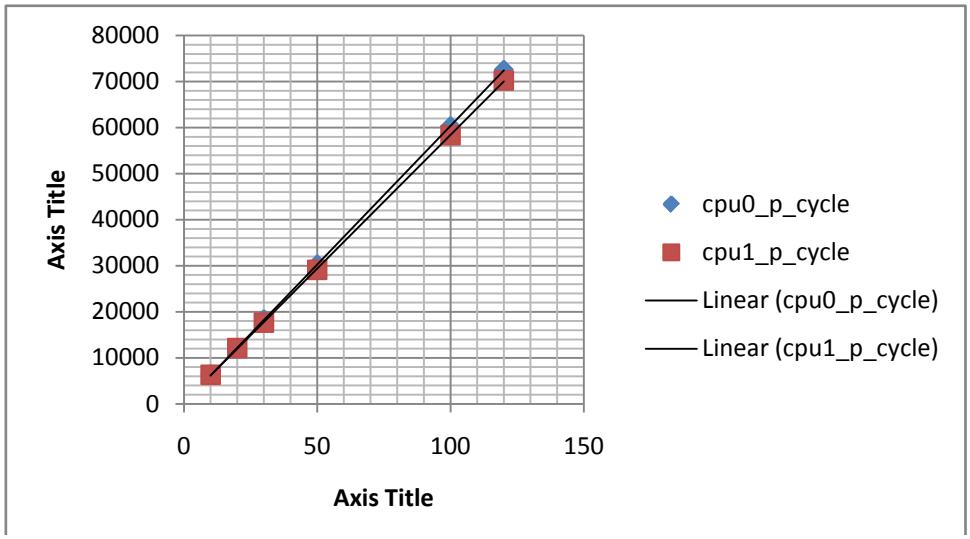


Figure 44: Cycles in parallel stage of Radix Sort with Simple Sort.

From figure 44, in parallel stage the time consumed is proportional to the input data. So the time saved for a CPU is proportional to the reduced input data. In this radix sort with simple divide, the number of reduced input data is half of the total inputs. In this chart, CPU0 and CPU 1 have slight differences, and the proportion can be chosen to the average of the cpu0_p_cycle and cpu1_p_cycle line.

From the chart above, the saved cycle can be estimated as:

$$y = 588(\frac{x}{2})$$

The y represents the time saved cycles; the x represents the number of inputs.

Now the problem is that when parallel computing is applied, whether parallel computing increases the efficiency. If the cost is greater than the time saved, parallel computing is increasing the efficiency. If not, the result is reversed.

So, in further investigation,using the equation below to represent the time cost is greater than the time saved:

$$17.6x + 1522 > 588(\frac{x}{2})$$

The result reveals thatparallel computing is decreasing the efficiency, under the condition below:

$$x < 5.5.$$

This result shows that when the input data is smaller than 5, parallel computing in fact decreases the efficiency.

The procedure of dividing with sortin radix sort is that first the program will generate a different number of input data,and then CPU 0 will divide the input data intotwo

groups based on a reference number; in this radix sort, the input larger than 550 is in one group and smaller is in another group.

In divide with sort, the sort algorithm can be considered in two parts:

- The divide with sort part. The divide with sort divides the input data intotwo groups based on a reference number.
- The parallel computing part. This step actually parallel computes the data

The steps can be calculated in execution cycles:

number of input	divide_cycle	CPU0_input	CPU0_p_cycle	CPU1_input	CPU1_p_cycle
10	2868	5	6308	5	6447
20	3062	11	12917	9	11020
30	3159	13	16280	15	19094
50	3650	23	26566	27	31909
100	4818	49	56754	51	60681
120	5206	61	71023	58	70491

Table 6: Cycles of Divide with Sort in Merge Sort.

The difference between the simple divide and divide with sort is that the divide with sort divides data according to reference data, so the last merge sort step is reduced.

In this program, the input data is not ideally parallel, so the degree of parallelism can be defined as the parallel part of the input divided by the total part of the program. For example, the first total input is 10, which has 3 executed in CPU0 and 7 executed in CPU1, so 6 out of the input 10 is actually parallel computing, and the other 4 are actually running serial. So the degree of parallelism is 3, which is the smaller input number; multiply it by 2 then divide by the total input:

$$(3 * 2)/10 = 60\%$$

In simple divide, the degree of parallelism is 100%.

number of input	divide with sort	dws_P_stage_cycle	simple_divide	sd_P_stage_cycle
10	100%	6447	100%	6414
20	90%	12917	100%	12133
30	87%	19094	100%	18358
50	92%	31909	100%	30288
100	98%	60681	100%	60298
120	98%	71023	100%	72532

Table 7: Different Parallel Stage Cycles of Divide with Sort and Simple Sort.

Compared with the simple divide, a conclusion that can be made is that the algorithms are more efficient because they are highly parallel.

For the same process, the cost and benefits of parallel computing can be evaluated for this radix sort.

First, the cost should be modified for this divide with sort; indeed, the divide with sort reduces the last merge step. This can be subtracted from table 5. So the actual cost is the divide cycle in table 7 subtractsthe last merge step in table 5.

number of input	divide_cycle	last_m_merge	cost_cycle
10	2868	197	2671
20	3062	366	2696
30	3159	569	2590
50	3650	925	2725
100	4818	1829	2989
120	5206	2189	3017

Table 8: The Cost of Merge Sort in Divide with Sort.

From the table, a line chart can be drawn.

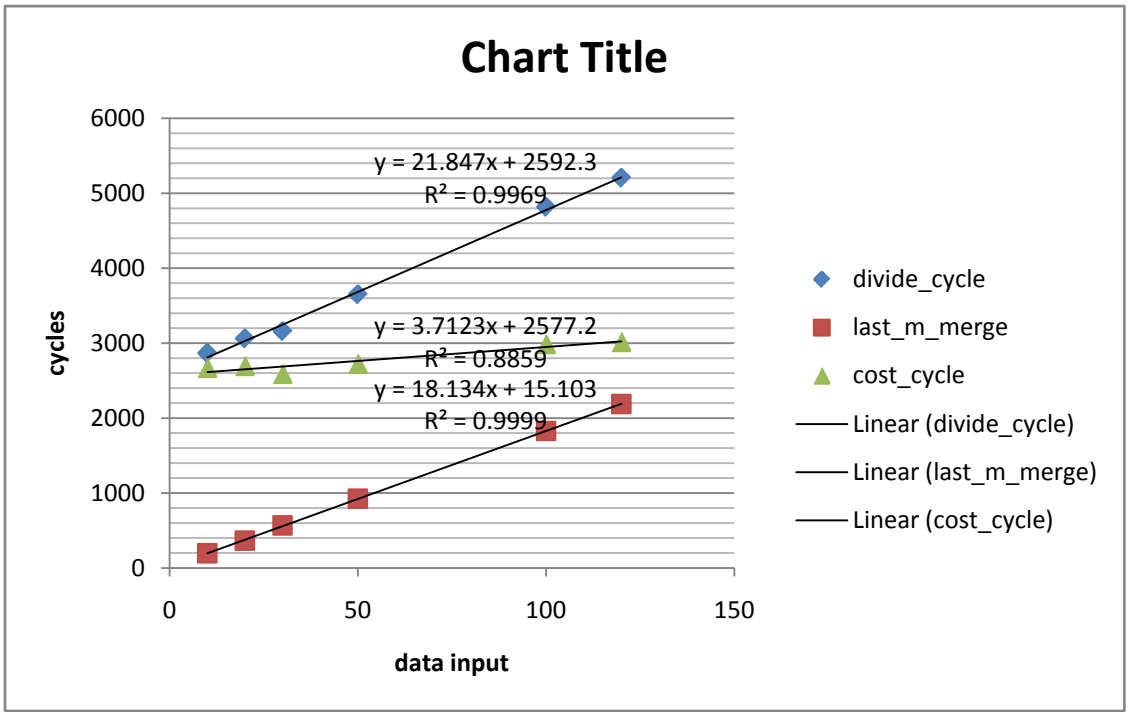


Figure 46: The cost cycle of radix sort in divide with sort.

The cost can be estimated:

$$y = 3.7x + 2577$$

The y represents the cost cycles. The x represents the number of inputs.

When listing all the input data and related cycles in parallel stages, the chart will be as follows.

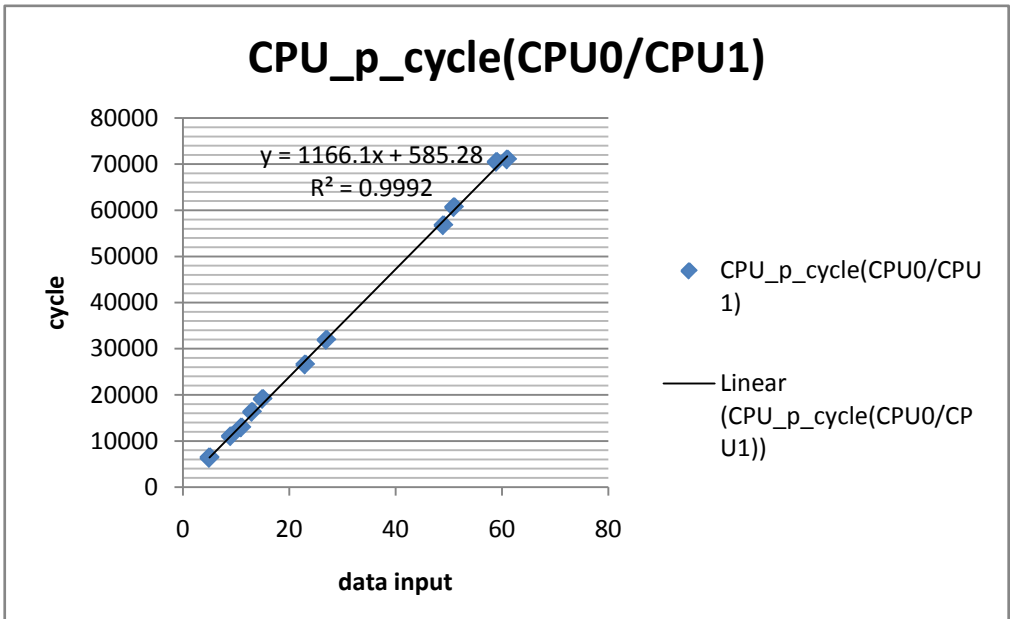


Figure 47: Cycles of CPU consumed in parallel stages.

From the chart, it is obvious the time consumed is proportional to the number of inputs. So the time saved is estimated as in simple divide:

$$y = 1166(x/2)$$

The y represents the time saved cycles; the x represents the number of inputs.

So, the result applies in this circumstance. If the time cost is greater than the time saved:

$$3.7x + 2577 > 1166(x/2)$$

Parallel computing is decreasing the efficiency, and the result is that:

$$x < 4.4$$

This shows that when the input data is greater than 4.4, parallel computing increases the efficiency.

3.3 Result Summary

In this project, all the objectives below were achieved:

- The platform Cortex-A9 MPCore software model is configured for applying parallel computation. All the potential problems that may cause faults in parallel computing are considered and fixed.
- The radix sort and merge sort algorithms are implemented using both simple divide and divide with sort.
- All the sort algorithms are successfully tested under different input data.
- The information tested from the algorithms in cycles is recorded and compared. Graphs, tables and charts are made based on this information.
- From the tables, graphs and charts, the cost and time saved model of parallel computing in this project were established.

- Applying the idea that cost in parallel computing is larger than time saved, along with the statistic model, find out under different inputs whether parallel computing increases efficiency
- Comparing the table and results of divide with sort and simple divide, the conclusion is that the highly parallel program will increase the efficiency.

Chapter 4Project Conclusion and Evaluation

The development of hardware has rocketed in the past decade.Multicore processors are becoming increasingly popular in computers and embedded systems. The software should keep pace with the trend of parallel computing.

Parallel computation is very powerful in increasing the efficiency of programs, but it is not always true that more parallelism will increase efficiency. As figure 17 shows William D. Gropp's theory, additional steps are introduced by parallel computing. [1]There must be certainty thatparallelism is the most efficient solution, and over-parallelism may significantly decrease the efficiency.

This project implemented sort algorithms on ARM Cortex-A9 MPCore, and investigated the program execution time under different input data. Models of the cost and benefits in parallel computation are established. Some researchis done based on information from the program, and some of the conclusions can be derived from the results of the project.

In searching and implementing the algorithms on parallel systems, the conclusion would be the program needs no inner dependency to be implemented on parallel systems.

In comparing the two divide methods, the conclusionis a highly parallel program will be more efficient on the parallel system. It also indicates that the program should be designed parallel to make better use of parallel systems.

In establishing these four models of cost and benefits of sort algorithms, the conclusion is that if the model of cost and the benefit of a parallel computation can be established, the efficiency of parallel computing can be predicted. If this can be applied in real systems, it may prevent parallel systems from decreasing the efficiency of some programs.

In the project, a multi core Cortex™-A9 MPCore developed by ARM is used. The processor is powerful and provides high performance, but it needs to be carefully configured for safe parallel computing. The initialization, some caches and memory problems should be considered to ensure parallel computation is correct.

4.1Critical Evaluation

In computer science, the efficiency and energy saving are always hot topics. It is also the same in parallel computations. Researchers have proven that in current parallel systems, the parallelism is not always efficient and energy saving. So, when the parallelism should be applied, what is the best parallel algorithm to a parallel system and what is the cost and benefits in parallelism are really concerned in current study of computer science.

In this project, based on the ARM Cortex A9 MPCore and two algorithms merge sort and radix sort, the following objectives are achieved.

- The platform Cortex-A9 MPCore software model is configured for applying parallel computation. All the potential problems that may cause faults in parallel computing are considered and fixed.
- The radix sort and merge sort algorithms are implemented using both simple divide and divide with sort.
- All the sort algorithms are successfully tested under different input data.
- The information tested from the algorithms in cycles is recorded and compared. Graphs, tables and charts are made based on this information.
- From the tables, graphs and charts, the cost and time saved model of parallel computing in this project were established.
- Applying the idea that cost in parallel computing is larger than time saved, along with the statistic model, find out under different inputs whether parallel computing increases efficiency
- Comparing the table and results of divide with sort and simple divide, the conclusion is that the highly parallel program will increase the efficiency.

However, due to the restriction of hardware, it is based on two core systems and software models. To obtain an accurate result, it will need the real processors to be tested. In future works, the models should be extended to mutli cores to get a more general result.

But, there is one idea to improve the efficiency as the project shows. Locate the cost and benefits in parallel computing, model them and predict the other situations in parallel computing, in a programmer view, it is improving the parallelism of the programs.

Chapter 5Future Work

5.1Future Work

Mutli core technology is becoming more and more popular, and lots of systems appear with even hundreds of cores integrated on one chip. The trend in mutli core systems is integrated more cores on a system. The theory of parallel computation should continue in the following aspects.First, the research of parallel computation should extend to general mutli core systems. As the Grop referred in figure17 , if to a specific class of processors, the model of efficiency and number of processors can be identified, it will helps the parallel systems increasing the efficiency and reduce the energy cost.

Second, it is helpful to locate the cost and benefits real systems. Indeed, there are more characteristics need to be tested, some of them cannot be modeled in RTSM, some of them cannot be tested on software. As it is mentioned in model limitations, the RTSM is not exactly the same as the hardware, and the performance management unit is not implemented. The future work can be based on the real Cortex A9 MPCore, which can get more accurate result, and more detail about the parallel computations

Third, more algorithms need to be explored to test the models. In fact, these results got in the project highly depend on the structure of the program; different algorithms should be tested to get better and general results.

Fourth, the project applies the Cortex A9 MPCore, it is a very outstanding design of RISC architecture. However, there is another type of processors CISC, which is widely applied on PCs. Parallel computation researches should also extended on that processors.

References

[1] Xiao-ChuanCai, William D. Gropp, and David E. Keyes. A comparison of some domain decomposition algorithms for non symmetric elliptic problems.In Tony F. Chan, et al, Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Philadelphia, PA, USA, 1992. SIAM, chapter 29.

[2] Alan Chalmers, Jonathan Tidmus. March 1996.Practical Parallel Processing: An introduction to problem solving in parallel. International Thomson Computer Press. ISBN 1-85032-135-3, page 5-8.

[3]Furber, Stephen B. ARM System-on-Chip Architecture. Harlow, England: Addison-Wesley, 2000, page 35-39.

[4] Leonid Ryzhyk. The ARM Architecture. Chicago University, Illinois, U.S.A. http://www.cse.unsw.edu.au/~cs9244/06/seminars/08-leonidr.pdf

[5] Cortex™-A9 MPCoreRevision:r2p**0** http://infocenter.arm.com/help/topic/com.arm.doc.ddi0407f/DDI0407F_cortex_a9_r2p2_mpcore_trm.pdf [Online; accessed 7-May-2011].

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. MIT Press, Cambridge, MA, 3nd edition (2009), page 30-40.

[7]. Ian Foster (1995)"Designing and Building Parallel Programs" Addison-Wesley, 1995 ISBN 0-201-57594-9, section 2.1.

[8] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. IEEE Computer, 41(7):33–38, 2008.

[9]The ARM Cortex-**A9 Processors** http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf [Online; accessed 7-May-2011].

[10] Xiao-ChuanCai, William D. Gropp, and David E. Keyes. A comparison of some domain decomposition algorithms for non symmetric elliptic problems.In Tony F. Chan, et al, Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Philadelphia, PA, USA, 1992. SIAM, chapter 22.

[11] Alan Chalmers, Jonathan Tidmus. March 1996.Practical Parallel Processing: An introduction to problem solving in parallel. International Thomson Computer Press. ISBN 1-85032-135-3, page 12-16.

[12] The ARM Cortex-A9 MPcore. http://www.arm.com/images/Cortex-A9-MP-core_Big.gif

[Online; accessed 11-May-2011].

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. MIT Press, Cambridge, MA, 3nd edition (2009), page 190-200.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. MIT Press, Cambridge, MA, 3nd edition (2009), page 250-270.

[15] Ian Foster (1995)"Designing and Building Parallel Programs" Addison-Wesley, 1995 ISBN 0-201-57594-9, section 2.1.

[17] The ARM Linker User Guide. http://www.keil.com/support/man/docs/armlink/armlink_Cchfgafb.htm [Online; accessed 25-september-2011].

[18] Cortex A Series programmer Guide Version: 1.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0041c/Chddbefb.html[Online; accessed 25-september-2011]. Page 96-99

[19] Cortex A Series programmer Guide Version: 1.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0041c/Chddbefb.html[Online; accessed 25-september-2011]. Page 280-290

[20] Cortex A Series programmer Guide Version: 1.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0041c/Chddbefb.html[Online; accessed 25-september-2011]. Page 105-110

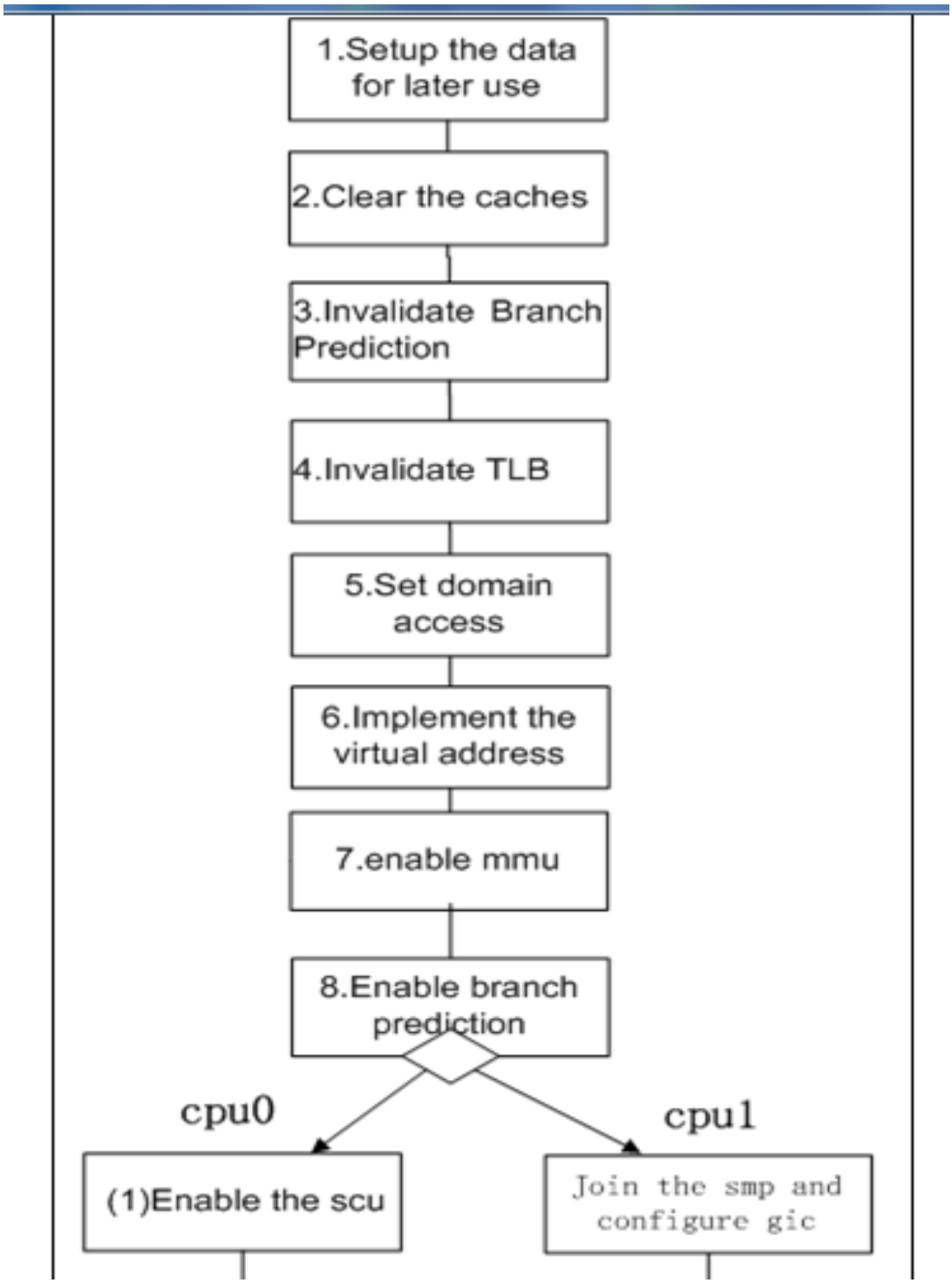
[21]ARM® ArchitectureReference ManualARM®v7-A and ARM®v7-REditionhttps://login.arm.com/login.php?cams_login_config=http&cams_original_url=https%3A%2F%2Fsilver.arm.com%2Fdownload%2Fdownload.tm%3Fpv%3D1104739&cams_security_domain=syste [Online; accessed 25-september-2011]. Page 1150-1160

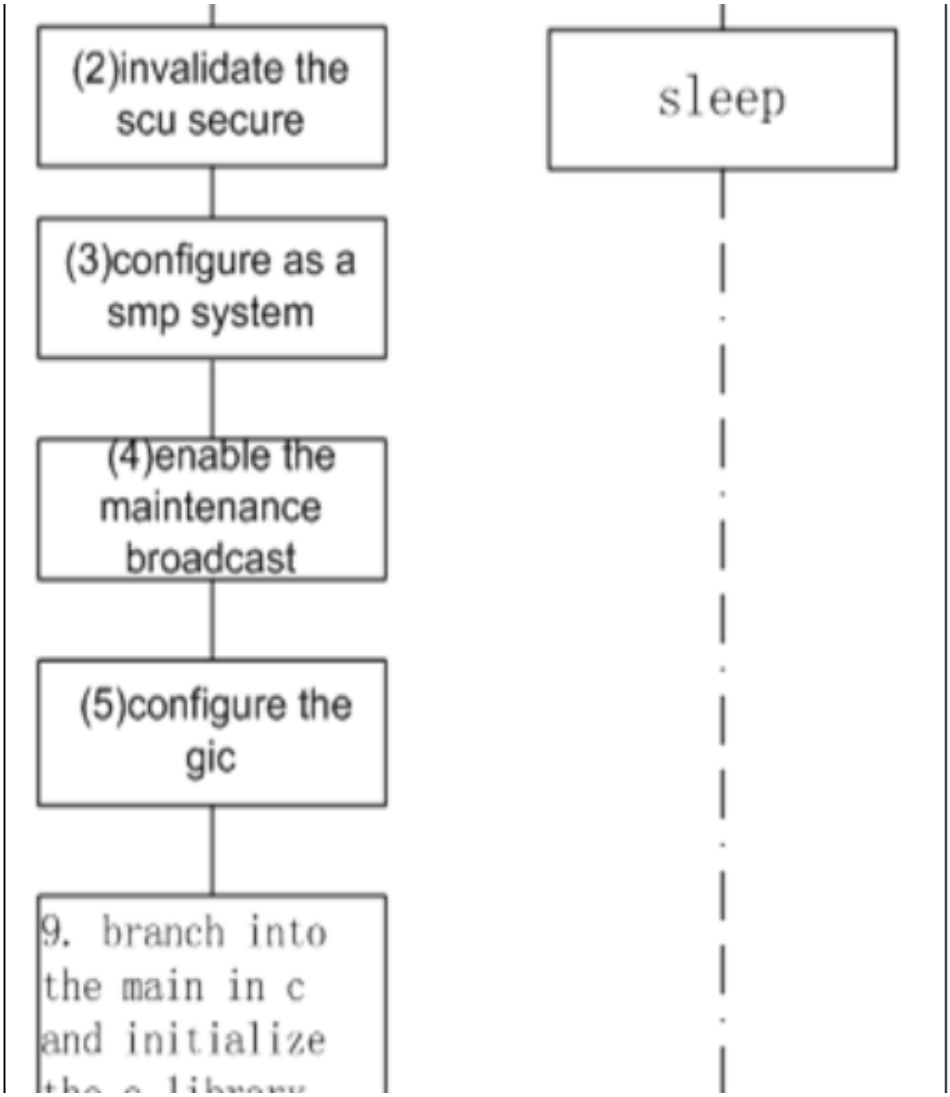
[22]Application NoteInterrupts on MPCore developmentboardshttp://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dsi0033l/index.html [Online; accessed 25-september-2011]. Page 9

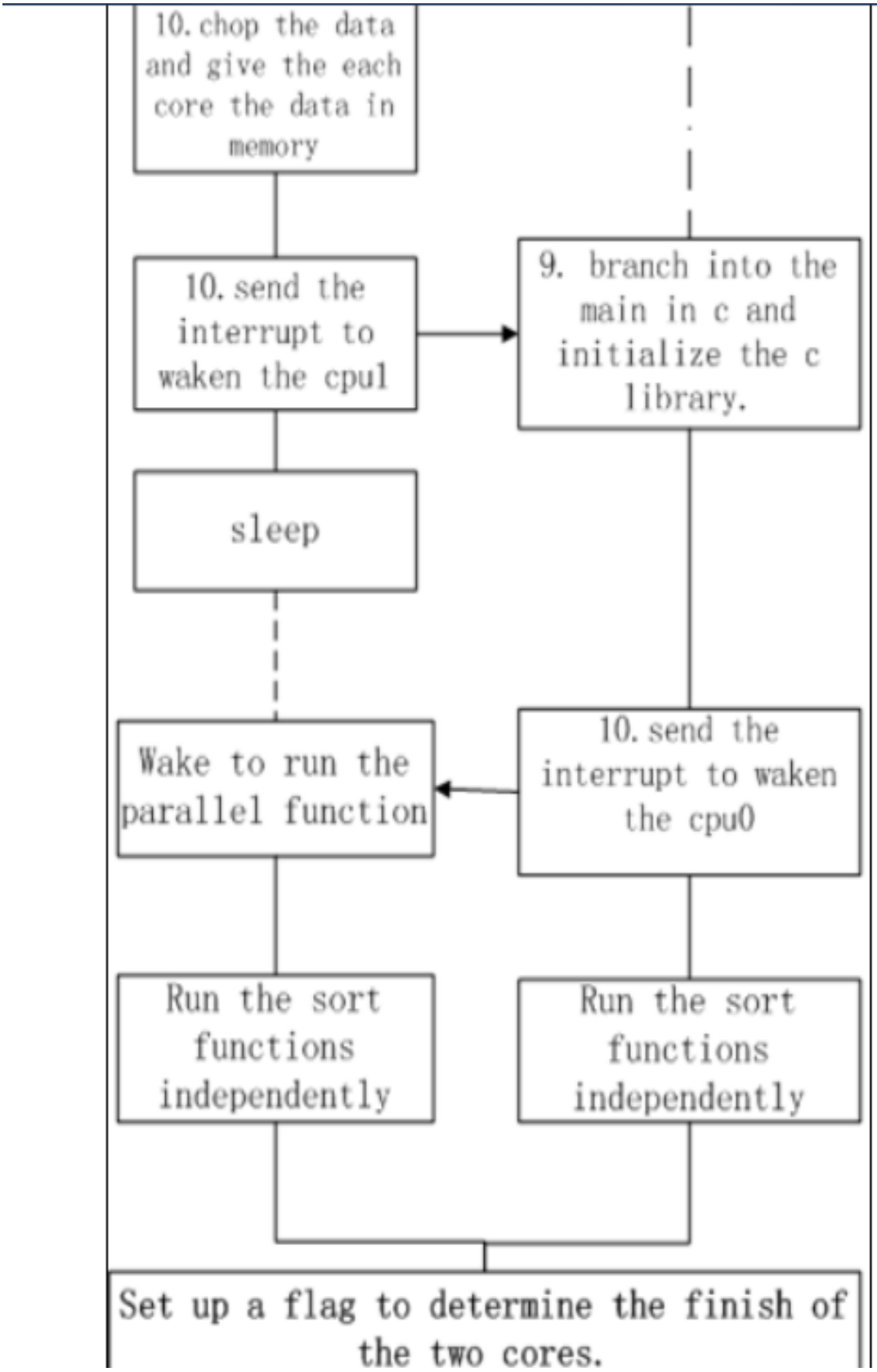
[23]ARM® ArchitectureReference ManualARM®v7-A and ARM®v7-REditionhttps://login.arm.com/login.php?cams_login_config=http&cams_original_url=https%3A%2F%2Fsilver.arm.com%2Fdownload%2Fdownload.tm%3Fpv%3D1104739&cams_security_domain=syste [Online; accessed 25-september-2011]. Page 1751

[24] Cortex A Series programmer Guide Version: 1.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0041c/Chddbefb.html[Online; accessed 25-september-2011]. Page 65-75

Appendix A the Design Flow of the Code




(2)invalidate the scu secure
(3)configure as a smp system
(4)enable the maintenance broadcast
(5)configure the gic
9. branch into the main in c and initialize
sleep


10. chop the data and give the each core the data in memory
10. send the interrupt to waken the cpu1
9. branch into the main in c and initialize the c library.
sleep
Wake to run the parallel function
10. send the interrupt to waken the cpu0
Run the sort functions independently
Run the sort functions independently
Set up a flag to determine the finish of the two cores.

Appendix B Code Example

Due to large size of the code, this is the part of code related to the projects. The full code should contain 4 projects, each project contains startup.s, function.s, main.s, parallel.h, parallel_function.c, performance_unit.s, retarget.c, timer.h, wode.scat. The four projects are all packs with a file, with a code notification“code_notification.txt” in the files.

```
//*****************************************************************
//                  The head function of the ‘parallel.h
//*****************************************************************
#ifndef PARALLEL_H_
#define PARALLEL_H_
voidsend_sgi(unsignedintID,unsignedintcore_list,unsignedintfilter_list);
unsignedintget_id(void);
voidiniti_parallel_computing(void);
voidparallel_computing(void);
#endif/*PARALLEL_H_*/
//*****************************************************************
//                  The head function of the ‘timer.h
//*****************************************************************
#ifndef TIMER_H_
#define TIMER_H_
voidenable_cycle(void);
voidenable_count(void);
voidclear_count(void);
unsignedintget_count(void);
#endif/*TIMER_H_*/

//*****************************************************************
//                              The main function in c
//*****************************************************************
// C standard Libary
#include <stdio.h>
#include "parallel.h"
#include "timer.h"
// Include project header
unsignedintchange_wait(void);
intmain(void)
{
unsignedintcpu_id;
unsignedintrc=0;
intcycle=0,cycle_1=0;
enable_cycle(); // turn on cycle counter
```

```
clear_count();
enable_count();
        // printf("for test");
cpu_id=get_id();//CPU 0 and CPU 1 branch to their own code part
if(cpu_id==0){
                // cycle=get_count();
initi_parallel_computing();//divide function
send_sgi(0x0,0x0F,0x01);//send waken signal through interrupt
cycle_1=get_count();
printf("get the test cycle, cycle_1 %d\n",cycle_1);
                        //do the synchronization
while(rc==0){
rc=change_wait();
}
}
if(cpu_id==1){

send_sgi(0x0,0x0F,0x01);
}

parallel_computing();

return0;
}

//*************************************************************************************
//                          The parallel function in c for merge sort
//*************************************************************************************
*
#include<stdio.h>
#include<stdlib.h>
#include "timer.h"
#define BUND (100) // define the bound of the generate data
#define finished (2)// finish flag
unsignedintget_id(void);
voidmerge_sort(unsignedint*array,unsignedint*array_sort,intstart,intend);
voidcombine(unsignedint*array,unsignedint*array_sort,intstart,intend,intmid);
unsignedint*a;
unsignedintdata_0[200];//this is in shared memory, the data is send to CPU 0
unsignedintdata_1[200];//this is in shared memory, the data is send to CPU 1
unsignedint*array_sort_0;
unsignedint*array_sort_1;
intnum_0,num_1;// tell the number of data is sent
```

```
intflag_finish;
void__wfe(void);
voidiniti_parallel_computing(void){// the initialization part of the parallel computation, generate the input data and    divide the data
inti=0,num=200;// num is the number of input data
intj=0,k=0;
unsignedintb=0,c=100;
intd=0,e=0;
volatileunsignedinttime_0=0,time_1=0,time_2=0,time_3=0;
unsignedintn=10;
flag_finish=0;

a=(unsignedint*)malloc(num*sizeof(unsignedint));
for(i=0;i<num;i++){//generate data

a[i]=1+rand()%BUND;
}
time_1=get_count();
printf("time_1%d\n",time_1);
for(i=0;i<num;i++){// divide data
if(a[i]>BUND/2){
data_1[j++]=a[i];
num_1=j-1;
}
else{
data_0[k++]=a[i];
num_0=k-1;
}
}
return;
}

voidparallel_computing(void){// the distribute data to CPU0 and CPU 1
                        //int time_1=0;
inti=0;
intcycle=0,cycle_f=0,cycle_s=0;

unsignedintcpu_id=get_id();
clear_count();
cycle=get_count();
printf("\n");// receive data according to the ID of cpu
if(cpu_id==0){
array_sort_0=(unsignedint*)calloc(num_0,sizeof(unsignedint));
```

```
merge_sort(data_0,array_sort_0,0,num_0);
for(i=0;i<num_0+1;i++){
printf("sort %d ",array_sort_0[i]);
}
printf("\n");
flag_finish++;
}
if(cpu_id==1){

array_sort_1=(unsignedint*)calloc(num_1,sizeof(unsignedint));
merge_sort(data_1,array_sort_1,0,num_1);
for(i=0;i<num_1+1;i++){
printf("sort %d ",array_sort_1[i]);
}
printf("\n");
flag_finish++;
}
while(flag_finish<2){
cycle_f=get_count();
printf("the sort data number is num_0 %d, num_1 %d",num_0,num_1);
printf("get the test cycle, cycle %d, cycle_f %d\n",cycle,cycle_f);
__wfe();
}
cycle_s=get_count();
printf("the sort data number is num_0 %d, num_1 %d",num_0,num_1);
printf("get the test cycle, cycle %d, cycle_s %d\n",cycle,cycle_s);
return;
}

voidmerge_sort(unsignedint*array,unsignedint*array_sort,intstart,intend)// using recurisive way to
chop the data for merge sort
{
intmid;
if(start<end){
mid=(start+end-1)/2;

merge_sort(array,array_sort,start,mid);
merge_sort(array,array_sort,mid+1,end);
combine(array,array_sort,start,end,mid);

}
return;
}
```

```
voidcombine(unsignedint*array,unsignedint*array_sort,intstart,intend,intmid)// combine the data in the correct order
{
intleft=start;
intright=mid+1;
inti=start;
while(left<=mid&&right<=end){
if(array[left]<=array[right]){
array_sort[i]=array[left++];
i++;
}
else{
array_sort[i++]=array[right++];
}
}
while(left<=mid){
array_sort[i++]=array[left++];
}
while(right<=end){
array_sort[i++]=array[right++];
}
i=end;
while(i>=start){
array[i]=array_sort[i];
i--;
}
}
//*******************************************************************************************
//                              Thelibrary code of performance monitor unit in assembler
//*******************************************************************************************
PRESERVE8

AREA    performance_unit, CODE, READONLY

   EXPORT enable_cycle
;there is a pmu unit through cp15 to help
enable_cycle PROC
   MRC         p15, 0, r0, c9, c12, 0    ; E, bit [0],1 All counters are enabled.
   ORR         r0, r0, #0x01
   MCR         p15, 0, r0, c9, c12, 0
   BX          lr
   ENDP
```

```
    EXPORT enable_count
;there is a pmu unit through cp15 to help
enable_count PROC

    MRC         p15, 0, r0, c9, c12, 0    ; D, bit [3] Clock divider.0 When enabled, PMCCNTR counts every clock
cycle
    AND       r0, r0, #0xFFFFFFF7
    MCR         p15, 0, r0, c9, c12, 0
    MRC         p15, 0, r1,c9,c12,1
    ORR         r1,r1,#0x80000000
    MCR         p15, 0, r1,c9,c12,1
    BX          lr
    ENDP

     EXPORT clear_count
;there is a pmu unit through cp15 to help
clear_count PROC


    MRC         p15, 0, r0, c9, c12, 0    ;C, bit [2] Clock counter reset.
    ORR         r0, r0, #0x4
    MCR         p15, 0, r0, c9, c12, 0
    BX          lr
    ENDP

    EXPORT get_count
get_count PROC

    MRC         p15,0,r0 ,c9,c13,0    ;C, bit [2] Clock counter reset.
    BX          lr
    ENDP

    END
//*****************************************************************************
//                          The parallel function in c for radix sort
//*****************************************************************************
*

#include<stdio.h>
#include<stdlib.h>
#include "timer.h"
#define BUND (900)
```

```
#define finished (2)
unsignedintget_id(void);
voidradix_sort(unsignedint*array,unsignedint*array_sort,intnum);
intget_temp_0(unsignedint,int);
structnode{
unsignedintdata;
structnode*next;
};
structboard{
structnode*board[10];
structboard*next;
};
typedefstructboardBoard;
typedefstructnodeNode;
unsignedint*a;
unsignedintdata_0[200];
unsignedintdata_1[200];
unsignedint*array_sort_0;
unsignedint*array_sort_1;
intnum_0,num_1;
intflag_finish;
void__wfi(void);
voidiniti_parallel_computing(void){
inti=0,num=100;
intj=0,k=0;
unsignedintb=0,c=100;
intd=0,e=0;
volatileunsignedinttime_0=0,time_1=0,time_2=0,time_3=0;
unsignedintn=10;
flag_finish=0;
a=(unsignedint*)malloc(num*sizeof(unsignedint));
for(i=0;i<num;i++){

a[i]=100+rand()%BUND;
}
time_0=get_count();
```

```
for(i=0;i<num;i++){
if(a[i]>(BUND/2+100)){
data_1[j++]=a[i];
num_1=j-1;
}
else{
data_0[k++]=a[i];
num_0=k-1;
}
}
time_1=get_count();
printf("\n time_0 %d time_1 %d",time_0,time_1);
//                printf("\nget the cycle %d", time_1-time_0);
return;
}

voidparallel_computing(void){

inti=0;
intptime_0=0;
intptime_1=0;
intptime_2=0;
unsignedintcpu_id=get_id();
printf("test the num %d %d\n",num_0,num_1);
clear_count();
ptime_0=get_count();
if(cpu_id==0){
array_sort_0=(unsignedint*)calloc(num_0,sizeof(unsignedint));
radix_sort(data_0,array_sort_0,num_0+1);

flag_finish++;
}
if(cpu_id==1){
array_sort_1=(unsignedint*)calloc(num_1,sizeof(unsignedint));
radix_sort(data_1,array_sort_1,num_1+1);
flag_finish++;
}
```

```
while(flag_finish<2){
ptime_1=get_count();
printf("\n ptime_0 %d ptime_1 %d ",ptime_0,ptime_1);
printf("\ngetptime %d \n ",ptime_1-ptime_0);
__wfi();

}
ptime_2=get_count();
printf("\n ptime_0 %d ptime_2 %d",ptime_0,ptime_2);
printf("\ngetptime %d \n ",ptime_2-ptime_0);
return;
}

voidradix_sort(unsignedint*array,unsignedint*array_sort,intnum){
inti=0;
intj=0;
intk=0;
unsignedinttemp=0,temp_0=0;
Node*list[10];
Node*current,*current_data;
Board*start;
for(i=0;i<10;i++){
list[i]=NULL;
}
for(j=0;j<num;j++){

temp=array[j]%10;
for(i=0;i<10;i++){
if(temp==i){
current=list[i];
if(current==NULL){
list[i]=(Node*)malloc(sizeof(Node));
list[i]->data=array[j];
list[i]->next=NULL;
}
else{
while(current->next!=NULL){
```

```
current=current->next;
}
current->next=(Node*)malloc(sizeof(Node));
current->next->data=array[j];
current->next->next=NULL;
}
}
}
}
start=(Board*)malloc(sizeof(Board));
for(j=0;j<10;j++){
start->board[j]=list[j];
}
start->next=NULL;
for(j=0;j<2;j++){
start->next=(Board*)malloc(sizeof(Board));
for(i=0;i<10;i++){
start->next->board[i]=NULL;
}
for(i=0;i<10;i++){
current_data=start->board[i];
while(current_data!=NULL){
temp=current_data->data;
current_data=current_data->next;
temp_0=get_temp_0(temp,j);
for(k=0;k<10;k++){

if(temp_0==k){

current=start->next->board[k];
if(current==NULL){
start->next->board[k]=(Node*)malloc(sizeof(Node));
start->next->board[k]->data=temp;
start->next->board[k]->next=NULL;
}
else{
while(current->next!=NULL){
```

```
current=current->next;
}
current->next=(Node*)malloc(sizeof(Node));
current->next->data=temp;
current->next->next=NULL;
}
}
}
}
}
start=start->next;
start->next=NULL;
}
for(i=0;i<10;i++){
current=start->board[i];
while(current!=NULL){
printf("%d\n",current->data);
current=current->next;
}

}
return;
}
intget_temp_0(unsignedinttemp,intj){

intnum;
intremain;
if(j==0){
remain=temp%10;
num=(temp-remain)%100;
num=num/10;
returnnum;
}
if(j==1){
remain=temp%100;
num=(temp-remain)%1000;
num=num/100;
```

```
returnnum;
}
return0;
}
```