An Implementation of Parallel Accelerating System on Chip for DNA Sequence Matching

Like Yan, Dazhou Wan, Tianzhou Chen, Zhenbao Huang
College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, PRC

Abstract: - DNA sequence matching is one of the most crucial operations in molecular biology. Since huge volume of DNA sequences, fast DNA sequence matching is significant in research and some kinds of applications. In this paper, a heterogeneous multi-cores implementation in SoC is presented for DNA sequence matching, to accelerate the comparison of the abundant DNA sequences information. We design a multi-core SoC with a master-slave model where the master processor is the general purpose processor, which is responsible for the task scheduling for the four slave computing units which are design for the acceleration of DNA sequence matching. And the results of experiments show that the average speedup rate for sequence matching is 81.05 times compared to a software solution running on a quad-core PC on the same algorithm.

Key-Words: - accelerate, heterogeneous, SoC, DNA sequence matching, parallel, multi-core

1 Introduction

DNA turns out to be the most important biomacromolecule in the view of modern science and technology. It carries all the information that an organism needs in metabolism and reproduction. With the implement and rapidly development of the human genome project and microorganism genome project, people must understand the information on DNA-level to direct the research on medicine, biology and other related fields. And the development of the DNA detecting method (ex. DNA chip) cause the large increase of gene information. The number of the base pair has already reached 10-billion. So large amount not only offer an attractive market of gene field, but also demand a lot of analysis tools.

DNA matching is very widely used in laboratories and practice. The speed of matching is very limited due to the incalculability of DNA base pairs and the great complexity of DNA sequences. All these call for acceleration to DNA matching process. Fortunately, the feature of DNA sequence distribution and the dispersion in DNA sequence matching bestow it the possibility, and furthermore, the applicability for parallel processing.

There are two matching methods: exact matching and rough matching. Exact matching is to find out the given string P in the longer string T [1]. And rough matching is to find out the position of the DNA string where the similarity is higher than the standard similarity. The difference between the two matching methods is the precision of matching.

In general purpose processors in the desktop or workstation, the comparison is implemented using two registers at a time. Such as, load the data to the two registers in two times, compares the contents of two registers by subtracting one register from the other and save the result in condition registers. The problem with this mechanism is that only two registers is computed in a time. But in the comparisons of the DNA info string, abundant of data comparisons where started from deferent position are independent. It will be speed up the comparisons by parallel processing.

As the development of process technology, more and more transistor can be put in a single chip, which makes the implementation of complex system possible in a chip named SoC. In a SoC, there are will be one general purpose processing core and several cores to deal with special computing, this kind of SoC is called heterogeneous multi-core SoC.

Some kinds of heterogeneous multiprocessor systems have been designed and implemented such as the automotive real-time multi-core system [2] and heterogeneous system in a real-time video and graphics streams [3].

And some hardware accelerator had been designed to speedup the DNA sequence matching. For instance, [4] designs a hardware accelerator to implement the Needman- Wunsch algorithm, which will increase the performance of the DNA sequence matching. The DNA sequence matching unit in [5] has 3 simple instructions. But they are just used for do some simple configurations before the matching operations. In the two methods above, their functions are fixed and can not fit the flexible situations, when the task is changed.

With the multi-core SoC design method, the parallelism can largely improved, which it’s a great challenge to accelerate the DNA matching on a multi-core SoC. Task level parallelism by executing different task on separate cores simultaneously, Different schemes have been developed, such as a Static Scheduling Heuristic for Heterogeneous Processors Task scheduling algorithms for heterogeneous processors in [7], Master-slave tasking on heterogeneous processors in [8] and so on.

In this paper, a SoC with heterogeneous multi-cores with master-slave model for DNA sequences matching is proposed. In this system, a sequence matcher is designed to accelerate the DNA sequence matching. And four matchers are integrated in the system to speed up the matching parallel of DNA sequences. And a corresponding algorithm controlling and partition of the data and scheduling of task is explored.

The rest of the paper is organized as follow. Section 2 presents the design of the overall framework of the heterogeneous multi-cores SoC design for the system. Section 3 presents the design and implementation of the components and system. Section 4 reports the experiments and the results. Finally, Section 5 gives a conclusion.

2 Multi-core SoC Framework

2.1 Overall framework

The basic principle of DNA sequencing is that the four DNA base-pairs can be presented with 2 bits characters Table 1 shows the two bits representations for these four types of DNA base pairs. So It can be sequenced with character string matching algorithm on general purpose processor. However, the general purpose processor is more suitable for control but not the comparison because the general purpose design. And it is not optimized for this kind of application, which results in poor performance.

Table 1. Representation of DNA base pairs.

Base Pairs of DNA	Bits
Adenine	00
Cytosine	01
Guanine	10
Thymine	11

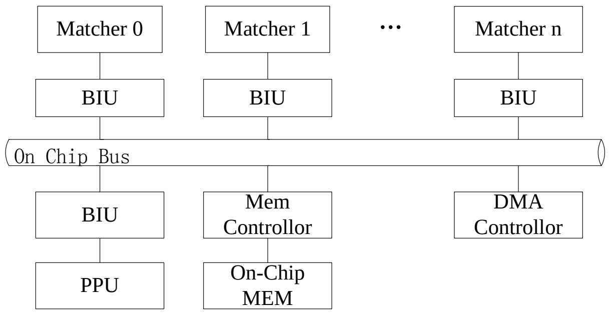

Figure 1. Overall Framework.

In order to accelerate the DNA sequence matching, a special purpose processing unit should be designed for better performance, which is named matcher. And for the independence of each sequence segments, the sequence matching can be done parallel. So more then one matchers should be integrated to improve the parallelism to further improve the performance. Besides, a general purpose processor should be adapted to control the matching, such as sequences dispatching, the results merging and so on.

Based on the previous discussion, the DNA matching SoC design should keep to the following design principles:

1) Heterogeneous: A general purpose processor core for controlling, dispatching, merging, and several matchers for sequence matching. So that the system can speed up the DNA matching by parallel processing.
2) High Agility: Our design is not against any specific matching algorithm. It's only optimized for the basic comparison operation.
3) High Integration: All the mentioned general purpose processor core and matchers are integrated on a single chip, so that the communication cost can be reduced sufficiently.

With these 3 principles, an overall framework is given in Figure 1. In this framework, a heterogeneous multi-core architecture is proposed with master-slave model: The master is a general purpose processor takes control of the operation, task dispatching and results merging. The slave cores are designed for the comparison operations. All the processors are connected to the on-chip-bus via the BIU. We also designed On-Chip-Memory to store the target and source DNA sequence and the matching result. Since the size of the on-chip-memory is limited, a DMA controller is required to transfer the data between the on-chip-memory and the extern storage with higher performance.

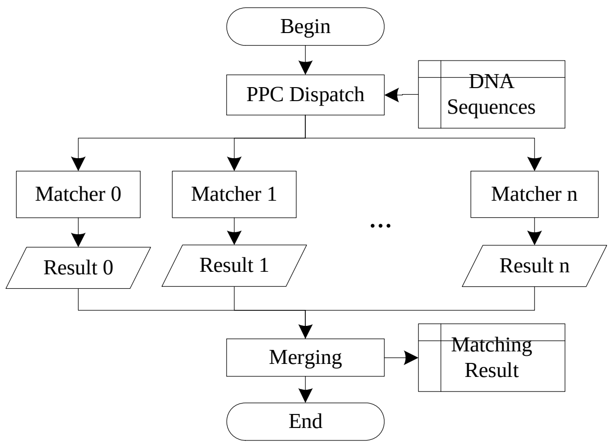

Figure 2. Parallel matching flow.

2.3 Parallel processing flow

With the architecture shows in Figure 1, the DNA sequences are matched parallel. The processing flow is shown in figure 2 on the proposed architecture. Because the exact matching is a extreme condition of rough matching, only rough matching method is explored in this paper.

The processing flow is shown below:

1) A task of DNA sequence matching arrives, the master gets the target and source DNA sequence. And it partitions the DNA sequence into several segments and dispatches them to each matcher via on-chip bus.
2) Every matcher will deal with the corresponding segments and got the similarity and the offset position in the sequence.
3) Then, the results are gathered from the output of all matchers by the master, and the master can elicit the final similarity and the offset position for the target sequence in source sequence.
4) The final result can be transferred to the memory or shown to the user.

3 Design and Implementation

3.1 The design of the matcher

The matcher is designed for the sequence matching. The structure of matcher is shown in Figure 3, in detail, it is constructed with Comparator, Controller, Local Storage (Source Mem, Result Mem), Local DMA, communication and configuration registers, while the bus interface is implemented by IPIF module, which is bus agnostic, provided some very basic services, such as slave attachment, address decoding, byte steering, and some optional services that can greatly simplify the task of creating the peripheral. Selecting the DMA service and the user-logic registers support, our MATCHER has the ability to load and store the data in the memory.

The controller reads the target sequence and source sequence from the Source Mem and forwards them to the comparator. The data are executed XOR operation to get which bits are the same. Then it works out the similarity by counting the number of bits at state “1”. The similarity and the address where the similarity presents will be transferred to the Result Mem.

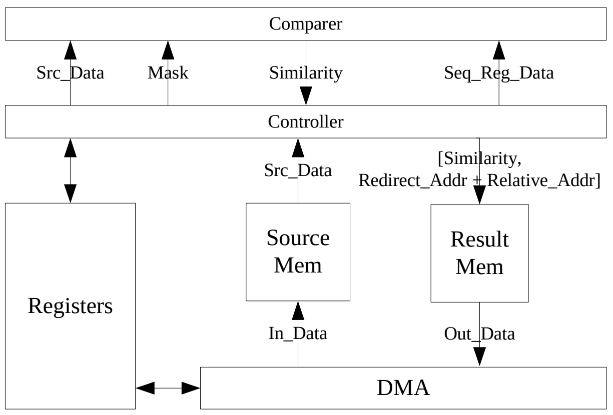

Figure 3. Matcher Hardware Structure

Table 2 describes all base, control, and status registers inside the matcher IP Core. The Address field indicates a relative address in hexadecimal. Width specifies the number of bits in the register, and Access specifies the valid access types to that register. RW stands for read and write access, R for read-only access.

3.2 Task Dispatching

The matcher can compare 128 bit data (64 DNA base-pairs) every clock. It compares the Sequence data and the data from database and provides the result to its local Result Memory when the similarity of the two exceeds the Similarity set by user. Then,

Table 2. All base, control, and status registers inside the Matcher.

Name	Address	Width	Access	Description
MODER	0x00	64	RW	MODE REGISTER
INT_SOURCE	0x08	64	RW	Interrupt Source Register
INT_MASK	0x10	64	RW	Interrupt Mask Register
Seq_Reg_Data_h	0x18	64	RW	The Data for Sequencing
Seq_Reg_Data_l	0x20	64	RW	The low 64bits data for sequencing
Seq_Reg_Data_Bits	0x28	64(8)	RW	The length of Sequencing Data
Seq_Similarity	0x30	64(8)	RW	The similarity
Seq_Mask_h	0x38	64	RW	High 32bits mask data for Sequencing
Seq_mask_l	0x40	64	RW	Low 32bits mask data for Sequencing
MATCHER_Mode	0x48	3	Write/Read	Mode of MATCHER execution
Src_mem_state	0x50	64	RW	The State of Source Mem
Redirect_Addr	0x58	64(24)	Write/Read	Redirect address for Source Mem
Receive_BD	0x1000	64	Write/Read	The first 32bits indicates the length and configure information; high 32bits indicates the memory address MATCHER fetches.
Transmit_BD	0x1000	64	Write/Read	The first 32bits indicates the length and configure information; high 32bits indicates the memory address MATCHER transmits

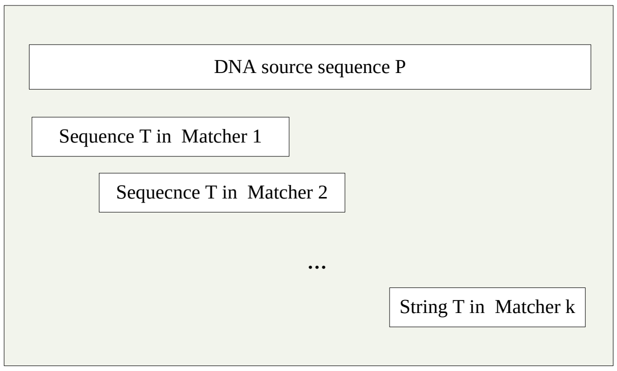

Figure 4. Parallel implementation in comparison.

these result data can be transferred to external memory through its local DMA.

In this SoC, the abundant of computation in the comparing of DNA sequences is speeded up by the matcher implemented in the FPGA. Figure 4 shows the comparison and the parallelism implementation. DNA sequence P is the source DNA data from the Gene Bank database while the sequence T is the target DNA sequence which wanted to comparison to find the similar sequence in P from different start point in different matcher.

The whole matching and task dispatching process is show in the Figure 2, the PPU dispatch is responsible for the whole data transmit and receive between the source DNA sequences database and the MEM IN M in each matcher, the PPU is also responsible for the gather and manage the result of the matching, such as the similarity and the offset position and so on.

In this paper, it is designed to be the compile-time task scheduling.

For one matcher, it is natural of doing the comparing tasks is to partition a long DNA sequence into segments of containing a certain length. Here it is assumed that the source DNA sequence P is partitioned to segments *P1*, *P2*, *P3* ….. *Pn*, while the target DNA sequence is partitioned to segments *T1*, *T2*, *T3* ….. *Tm*. And the result of the comparison of the *Pn* and *Tm* is *C(m, n)*, which contains the information of position and the corresponding similarity.

And the task scheduling algorithm can be described as follow: Assume T is partitioned with the length LT, and the P is partitioned with the length LP;

Initialize the result data group *G(C (i,j),LT, LP, k)* where the *C(i,j)* is the information of similarity and position to the comparison of *Ti* and *Pj* , and the k is the value to represent how much *C(i,j)* has been synthesized for this result data group. Here is the comparing process in C style:

```
For (i=0; i<m; i++)
{
  For (j=0;j<n; j++)
  {
    Execute the comparison of Ti and Pj and save the result C (i,j)
  }
  Update the result date group G(C(i,j),LT ,LP ,k+1)
  It is synthesizing the k* G(C(i,j),LT ,LP ,k+1) with the C(i,j), here it is also can been represent how long the matched sequence have been computed with the LP*k.
}
```

Finally, it will have the entirety information of the result of matching, which can be analyzed and picked up via the limit of the standard similarity value.

For the whole implemented system, it is designed with four matchers to archive the parallelism of the sequence matching.

They are comparing the sequences from different start point, while the *P2* used in Matcher 2 is the combined of left excursion x of *P1* used in Matcher 1 and the more x sequence after *P1*.

Here is the matcher operation in C style:

```
// Matcher task initiation
Void Matcher_Init(Unit_Seq * init_Seq,
Unit_Redirt_Addr * redirt_Addr, Unit_Matcher_Ctrl *
ctrl_matcher)
{
// set the initial data of sequencing task, including the
Seq_Reg_Data, //Seq_Reg_Data_Bits, Seq_Similarity,
Seq_Mask;
SetSeqData(init_Seq);
// set the redirect address of the local source memory0;
SetRedirtAddr0(redirt_Addr->addr0);
// set the redirect address of the local source memory1;
SetRedirtAddr0(redirt_Addr->addr1);
// set the Matcher control registers including
Matcher_Start and Matcher_Mode;
setMatcherControl(ctrl_matcher);
}
// Matcher DMA control initiation
Void Matcher_DMA_Init(Unit_DMA_Ctrl * ctrl_DMA);
// get Matcher local memory state including
Src_Mem0_State, Src_Mem1_State, Rsut_Mem0_State,
// Rsut_Mem1_State;
Void getMatcher_LocalMEM_State(Unit_LM_State *
state_LM)
```

3.3 Implementation

In this paper, the system is constructed with only one main core, and four slave matchers. The main core is the general purpose processor, the PowPC405 CPU on the Xilinx® ML403 development board. The development board contains the Virtex®-4 FPGA which include the hard PowerPC405 core, and also the normal hardware device such as the memory, the VGA interface, AC97 interface, UART, USB, and Ethernet and so on. We implement the SoC include the two kinds of processors and the normal device control interface all in the FPGA chip.

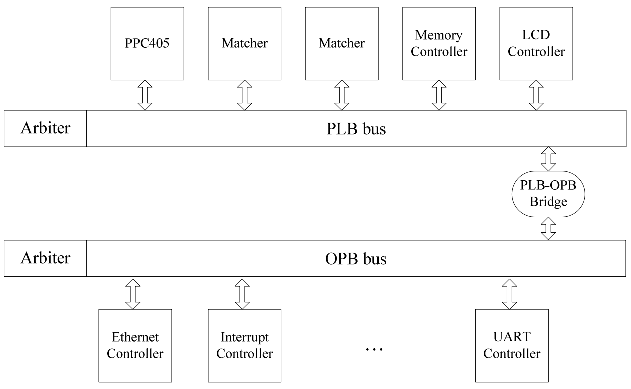

Figure 5. The implemented heterogeneous multi-core SoC architecture.

Figure 5 shows the implemented heterogeneous multi-cores system architecture, which is implemented in the FPGA chip. The IBM® CoreConnect® [9] on-chip bus is sufficiently flexible and robust to support such embedded system, in the Figure 6 we can see that the on-chip bus integrated the processors, peripheral controller cores into one system. All processors and the peripheral IP core are connected to the on-chip bus with the IPIF (Intellectual Property Interface). The Elements of CoreConnect® include the processor local bus (PLB), the on-chip peripheral bus (OPB), a bus bridge, and a device control register (DCR) bus. High performance peripherals connect to the high-bandwidth, low-latency PLB. Slower peripheral cores connect to the OPB, which reduces traffic on the PLB, resulting in greater overall system performance [10].

In the implemented system the main processor and the matchers are connected onto the PLB bus, the memory and the LCD controller are connected onto the PLB too, while the other device such as the UART, Ethernet, and interrupt controller is connected onto the OPB bus.

Table 3 shows the main detail information of the system. Cooperating with the SystemACE standalone software as the bootloader, the BSP and the application for those four matchers, a full system with the hardware and the software is implemented.

Table 3: The parameters of the implemented system.

Component	Attribute
Main core: PowPC405	Speed:100M Cache:32k
Slave: Matcher(four)	Speed:100M Mem in Matcher:64K+4K
PLB	Speed:100M Data width: 128bits
DDR memory	64M
Operation system	Linux kernel 2.4.26

4 Experiments and Results

On the implemented system, we validate the system in function and the performance:

First, input the test data:

1) registers Src_Mem_Stat, Rsut_Mem_State. These two register can combine into on char type register(8 bits), named group [Src_Mem_State, Rsut_Mem_State], initiated such as 8`b0000-0000, 8`b1000-0000, 8`b0011-0000, 8`b0011-1000, 8`b0011-1100 and so on;

2) The source DNA sequence and the target DNA sequence;

3) The target boundary of similarity.

Then the system starts working and results in form of [similarity, relative_addre_positon] will be out put.

Finally, we synthesize these result with the target data we expect, also in the format like [similarity, adder], and we will know, it is the certain data we expect.

Generally speaking, the processes are summarized in the following steps:

Set the four part necessary data;

Start comparison;

Estimate the result;

Change another group data, repeat step 2) and 3).

At last, the results looks like shown in Table 4.

Table 4. The result of matching DNA sequences.

Serial No.	Similarity	Position in the source sequence
1	87	286326800
2	98	286334466
3	81	286334512
…	…	….

To evaluate the performance of the implemented system, a software solution with the same algorithm is implemented on a PC equipped with quad-core CPU, each of whose cores is general purpose processing core. And the number of core is same to the number of matchers.

In detail, the performance comparison in group of different scale of DNA sequences. The running time consumed by the two systems is compared for each case. The data is measured in byte; while one byte can hold four DNA pairs. And the matcher is 128-bit in data width, while the width of the PLB is 128-bit. The PC we used to test has the 1.7G quad-core CPU, with a similar algorithm software solution.

The experiment results is shown in Figure 6, which shows that the parallel accelerating SoC is effectively improve the speed to the compared quad-core PC, and the average speedup is 81.05 times. And it also shown that the larger the sequence size is, the higher the speedup is.

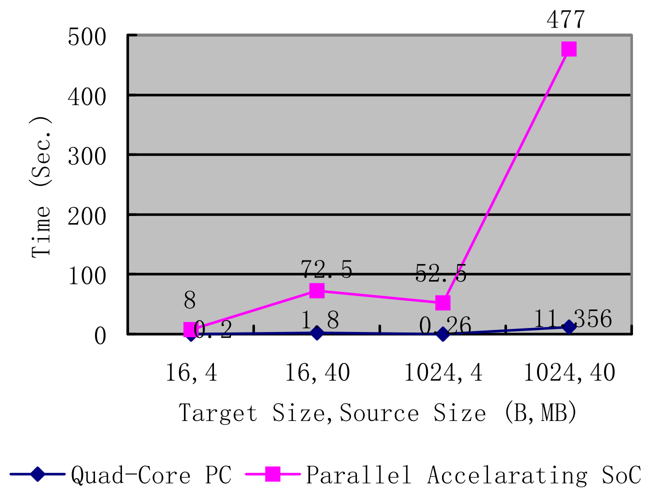

Figure 6. The comparison of matching time between SoC and PC

5 Conclusion

In this paper, a heterogeneous multi-core implementation of parallel accelerating SoC for DNA sequence matching is proposed, in order to accelerate the DNA sequence matching. This SoC is designed with the master-slave model where the master processor is a general purpose processor, which is responsible for the tasks dispatching and results merging, and for the four slave matchers which are design for the acceleration of DNA sequence matching. In this system, it is effectively improve the speedup to the matching of the DNA sequences. The results of experiments show that the average speedup rate for sequence matching is 81.05 times compared to a software solution on the same algorithm.

References

[1]. J. Kim, Computers Are from Mars, Organisms Are from Venus, *IEEE Computer*, Vol.35, No. 7, 2002, pp.25-32.

[2]. J. Axelsson. A Case Study in Heterogeneous Implementation of Automotive Real-Time Systems. *Proceedings of 6th International Workshop on Hardware/Software Co-Design*, 1998.

[3]. M. T. J. Strik, A. H. Timmer, J. L. vanMeerbergen, and G.-J. van Rootselaar. Heterogeneous multiprocessor for the management of real-time video and graphics streams. *IEEE Journal of Solid-State Circuits* , Vol.35, No.11, 2000, pp.1722-1731.

[4]. B Fagin, JGI Watt, R Gross. A Special-Purpose Processor for Gene Sequence Analysis. *Computer Applications in the Biosciences*, Vol.4, 1993.

[5]. Benjamin O. Brown, Dr. Meng-Lai Yin, Dr. Yi Cheng. DNA Sequence Matching Processor Using FPGA and JAVA Interface. *Proceedings of the 26th Annual International Conference of the IEEE EMBS.* 2004, pp.3043-3046.

[6]. Tien-Fu Chen,Chia-Ming Hsu, and Sun-Rise Wu, Flexible heterogeneous multicore architectures for versatile media processing via customized long instruction words. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 5, 2005, pp. 659-672.

[7]. Hyunok Oh and Soonhoi Ha, A Static Scheduling Heuristic for Heterogeneous Processors, *Proceedings of the Second International Euro-Par Conference on Parallel Processing* Vol.2, 1996, pp.573-577.

[8]. Pierre-Franc¸ ois Dutot, Master-slave tasking on heterogeneous processors. Proceedings of International Parallel and Distributed Processing Symposium, 2003.

[9]. CoreConnect Bus Architecture: http://www-306.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture.

[10].Xilinx® Corporation, ML40x EDK Processor Reference Design.