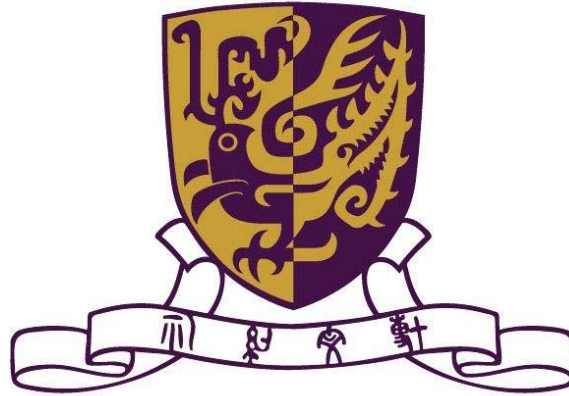


Department of Computer Science and Engineering  
The Chinese University of Hong Kong



**2007-2008**  
**Final Year Project Report (2<sup>nd</sup> Term)**

LYU0703

## **Electronic Advertisement Guide on PS3**

**Supervisor:**

Prof. Michael R. Lyu

**Prepared By:**

Wong Chung Hoi

05596742  
chwong5@cse.cuhk.edu.hk

**April 15<sup>th</sup> 2008**

## Abstract

---

This report covers our study and progress in implementing a LCS algorithm for Electronic Advertisement Guide (EAG) in this semester. It begins with some background information about commercial monitoring, the motivation and objectives of our final year project.

Then it is followed by an overview of our developing environment, the Cell Broadband Engine, which is the multi-core processor used in PlayStation®3. We will also have a brief introduction of the principal of parallel programming we have studied.

In the next topic, we will present our experience in designing and optimizing a video comparison algorithm based on LCS and running on PlayStation®3. We will show the advantages of using LCS approach rather than the minimum difference approach we did in the first term.

After that some experiment result will be demonstrated to show the accuracy and performance of our algorithm when finding similar video segments from the input data.

The last portion is a discussion about the project difficulties, project progress and possible future development. And finally a brief conclusion will be given.

# Table of Contents

---

<b>Abstract.....</b>	<b>2</b>
<b>Chapter 1 Introduction.....</b>	<b>6</b>
<b>1.1 Background Information.....</b>	<b>7</b>
<b>1.1.1 Market of Commercial Monitoring.....</b>	<b>7</b>
<b>1.1.2 Current Solution.....</b>	<b>8</b>
<b>1.2 Project Motivation.....</b>	<b>9</b>
<b>1.2.1 Increasing Need for TV Commercials Monitoring.....</b>	<b>9</b>
<b>1.2.2 Advances in Multi-Core Machines.....</b>	<b>10</b>
<b>1.2.3 Commercial Monitoring on Parallel Machines.....</b>	<b>11</b>
<b>1.3 Project Objectives.....</b>	<b>12</b>
<b>Chapter 2 Development Environment.....</b>	<b>14</b>
<b>2.1 Personal Computer.....</b>	<b>15</b>
<b>2.2 PlayStation®3.....</b>	<b>16</b>
<b>2.3 Cell Broadband Engine.....</b>	<b>17</b>
<b>2.3.1 Power Processor Element.....</b>	<b>18</b>
<b>2.3.2 Synergistic Processor Element.....</b>	<b>19</b>
<b>2.4 Software.....</b>	<b>20</b>
<b>2.4.1 Linux.....</b>	<b>20</b>
<b>2.4.2 IBM Cell Software Development Kit.....</b>	<b>20</b>
<b>Chapter 3 Principals of Parallel Programming.....</b>	<b>21</b>
<b>3.1 Parallel Algorithm vs. Serial Algorithm.....</b>	<b>22</b>

<b>3.2 SIMD Architecture.....</b>	<b>23</b>
<b>3.3 Shared-Memory System and Distributed-Memory System.....</b>	<b>24</b>
<b>3.4 Data Parallelism and Task Parallelism.....</b>	<b>26</b>
 <b>Chapter 4 Introduction of Problem and Solution.....</b>	 <b>27</b>
<b>4.1 Definition of Problem.....</b>	<b>28</b>
<b>4.2 High Level Description of the Solution.....</b>	<b>31</b>
 <b>Chapter 5 Low Level Implementation Details</b>	
<b>5.1 Converting Raw Video Data into Series of H13 Files.....</b>	<b>36</b>
<b>5.2 Processing H13 Files to Become the Final EAG.....</b>	<b>41</b>
<b>5.2.1 Further Data Conversion.....</b>	<b>41</b>
<b>5.2.2 The Main Program.....</b>	<b>46</b>
<b>5.2.2.1 Last Semester Approach – Review on Minimum Difference Algorithm.....</b>	<b>46</b>
<b>5.2.2.2 Introduction on Longest Common Subsequence (LCS) Problem.....</b>	<b>49</b>
<b>5.2.2.3 Why and How LCS Work on Video Stream Data..</b>	<b>54</b>
<b>5.3 Modification of LCS Algorithm.....</b>	<b>56</b>
<b>5.3.1 Equality of 2 Symbols.....</b>	<b>56</b>
<b>5.3.2 More Restriction on the EAG problem from R and T Frame.....</b>	<b>59</b>
<b>5.3.3 Logic System for “Equality” Comparison.....</b>	<b>67</b>
<b>5.3.4 Splitting Advertisements from LCS Result &amp; Analysis Strings.....</b>	<b>70</b>
<b>5.3.5 Synchronizing Flags.....</b>	<b>74</b>

5.3.6	Flags Propagation and Printing Result.....	76
5.3.7	Single Pass LCS to Multiple Passes LCS.....	78
5.3.8	Convert to EAG.....	81
5.4	Speeding Up in PlayStation®3 Platform.....	82
Chapter 6 Result Analysis.....		93
6.1	Experiment of Cross-Comparison of 7 Videos.....	94
6.2	Experiment of Comparing 2 One-Hour-Long Videos.....	102
6.3	Performance Comparison on PC and PlayStation®3.....	105
6.4	Cost Comparison on PC and PlayStaion®3.....	107
Chapter 7 Project Difficulties.....		108
Chapter 8 Future Development.....		110
Chapter 9 Conclusion.....		112
Chapter 10 Contribution of Work.....		113
Chapter 11 Acknowledgement.....		115
Chapter 12 Reference.....		116
Appendix A Result of Cross-Comparing 7 Days' TV Easy.....		117
Appendix B Result of Comparing 2 One-Hour-Long.....		125

# Chapter 1 Introduction

---

This chapter would briefly describe commercial monitoring. And discuss the reason why we are going to develop an automatic algorithm for this problem. The project motivation and objective are stated in this chapter also.

- Background Information
  - ◆ Market of Commercial Monitoring
  - ◆ Current Solution
- Project Motivation
  - ◆ Increasing Need for TV Commercials Monitoring
  - ◆ Advances in Multi-Core Machines
  - ◆ Commercial Monitoring on Parallel Machines
- Project Objectives

## 1.1 Background Information

### 1.1.1 *Market of Commercial Monitoring*

TV commercial broadcasting has been a big business. It is an effective way to promote and introduce products to the customers. Companies are spending a lot of money on TV commercials to have their products being exposed to the public.

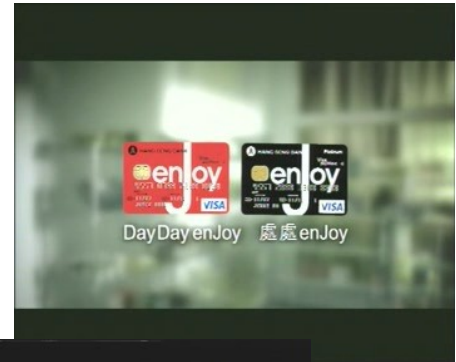


Since the cost for broadcasting an advertisement on TV is very high, companies would like to verify that their commercials are broadcasted through the TV channels as stated in the contract, this include verifying the number of broadcasts, broadcasting time and the commercial duration.

Companies are willing to pay for this kind of service too, e.g. for a large scale advertisement campaign which cost HKD\$ 10,000,000, they would spend about 5%, i.e. HKD\$ 500,000, for commercials monitoring. Thus there is a great market in this industry.

### 1.1.2 Current Solution

Nowadays there are companies providing this kind of commercial monitoring service. The approach they use is mainly monitoring the TV channels manually. Their staffs will need to take shift and sit in front of a TV, recording and timing the TV commercials that have been broadcasted. Then they can check the start time and end time of an advertisement. It is a hard work. Besides, verifying the commercials in this manual way is very inefficient.





## 1.2 Project Motivation

### ***1.2.1 Increasing Need for TV commercials monitoring***

Considering the situation in Hong Kong, the number of advertisement being broadcasted everyday is huge. From the channel list of licensed broadcasting services in Hong Kong published by the Hong Kong Broadcasting Authority, we can see that there is hundreds of TV channels receivable in Hong Kong. These channels are provided by free TV services, pay TV services and satellite TV etc.



If multiple channels are required to be monitored, this kind of service provider would need to recruit a lot of staffs to do this job. As a result, there is an increasing need for an automatic TV commercials monitoring system.

### 1.2.2 Advances of Multi-Core Machine

There are more and more applications requiring large amount of data manipulation and computation, such as advanced graphics, virtual reality, simulation and multimedia processing. Due to the limitation of single-core processor, including memory latency, wire delays, power consumption and heat problem, the development of increasing the performance of a processor has been shifted to multi-core one. Dual-core and quad-core chips for desktop machines are becoming popular today.

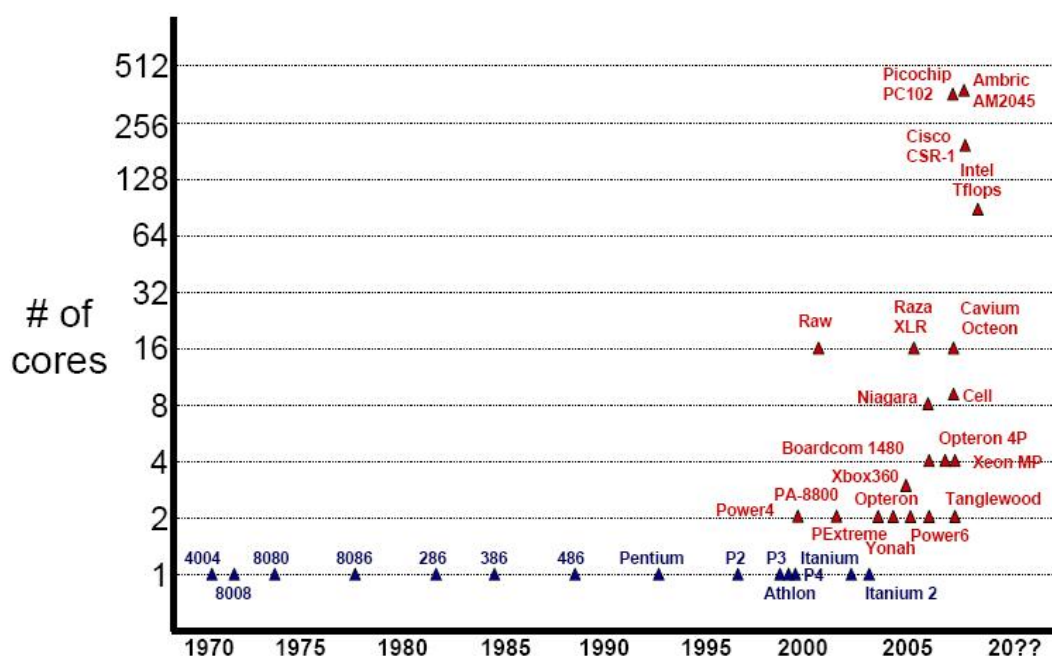


Fig. 1.1 Growth of No. of Cores in Processors

Although applications that run on a single-core machine can still run on a multi-core one, in order to take advantages of multi-core architecture, software engineers have to optimize and parallelize them. This would allow them to design applications with better performance.

### ***1.2.3 Commercial Monitoring on Parallel Machine***

As we mentioned in the previous part, there is a great market for commercials monitoring. We would like to design an algorithm to do this job automatically. We choose PlayStation®3 as our development platform since it is a cheap parallel machine with high processing power. We believe it is suitable for running application which is computation intensive, such as the proposed commercials monitoring algorithm.

Combining the problem we target to solve and the parallel machine we want to utilize, it gives rise to the topic of our final year project.

## 1.3 Project Objectives

The goal of our project is to develop a fast and accurate parallel algorithm for commercial monitoring, making it feasible to provide a Electronic Advertisement Guide (EAG).



Fig. 1.2 Electronic Advertisement Guide (EAG)

The first part of the project is to study the features of parallel programming and have some hands-on experience on it. Comparison and analysis of the performance difference between sequential and parallel programs are also needed. This part is being done in the 1<sup>st</sup> semester.

In the second part, an algorithm for commercial monitoring with high accuracy is implemented. The algorithm needs to be able to find similar video segments between two videos, with the assumption that the segment has appeared at least twice in both videos.

There are advertisements aired in the TV channels every minute. To make the algorithm practical and applicable, it needs to have high performance. E.g. spend less than an hour to identify advertisements that have appeared in a one-hour-long video.

One solution to this is to modify the algorithm from sequential approach to parallel approach. We optimize the program to the largest extent, showing that great improvement can be made with parallel programming and a multi-core machine.

In the following chapters, we will demonstrate the findings and the works that we have done through the year in order to achieve the above objectives.

## Chapter 2 Development Environment

---

In this chapter we will introduce the development environment of our project, including both hardware and software.

For hardware, we have two PCs and a PlayStation®3 running Windows XP and Linux respectively.

For software, we use the IBM Cell Software development Kit, which provide the compiler and libraries for parallel programming on the Cell processor.

We will introduce one by one in this list:

- Personal Computer
- PlayStation®3
- Cell Broadband Engine
  - ◆ Power Processor Element
  - ◆ Synergistic Processor Element
  - ◆ Element Interconnect Bus
  - ◆ Memory Management
- Linux
- IBM Cell Software Development Kit

## 2.1 Personal Computer

We are provided with two PCs for our project use. Although we do our parallel programming on the PlayStation®3, the configuration of PCs is still worth mention. We also run programs on PCs as a reference, to compare the performance of a sequential program and its parallel version on PlayStation®3.

Table 2.1 Major Specification of the PC	
CPU	Intel Pentium 4 3.0 GHz
Main Memory	1 GB RAM
Operating System	Windows XP Professional Edition

## 2.2 PlayStation®3

PlayStation®3 is the multi-core machine we used in this project. It uses the Cell Broadband Engine (Cell BE), which has great computation power, as the processor, giving high quality of game and graphics performance. The following is the basic hardware configuration of the PlayStation®3 we used.



Fig. 2.1 PlayStation®3  
produced by Sony

Table 2.2 Major Specification of the PlayStation®3	
CPU	3.2 GHz Cell Broadband Engine
Main Memory	256 MB XDR DRAM
Hard Disk	60 GB 2.5" SATA hard drive
PlayStation® System Software (i.e. the game OS)	Version 1.94
Operating System	Fedora 7 (Linux Kernel 2.6.21)

Although the major reason for designing PlayStation®3 is to be the next generation of video game console, its strong computation power and relatively low cost for a multi-core machine (about HK \$3,000) making it possible to have different applications and development. Sony also opened the platform to allow other OS (typically Linux) to be installed on PlayStation®3.



## 2.3 Cell Broadband Engine

As mentioned above, the Cell processor is the soul of the PlayStation®3. Cell, with full name Cell Broadband Engine Architecture, is jointly designed by Sony, Toshiba and IBM, started in 2001.

An overview of the Cell architecture is shown as follow:

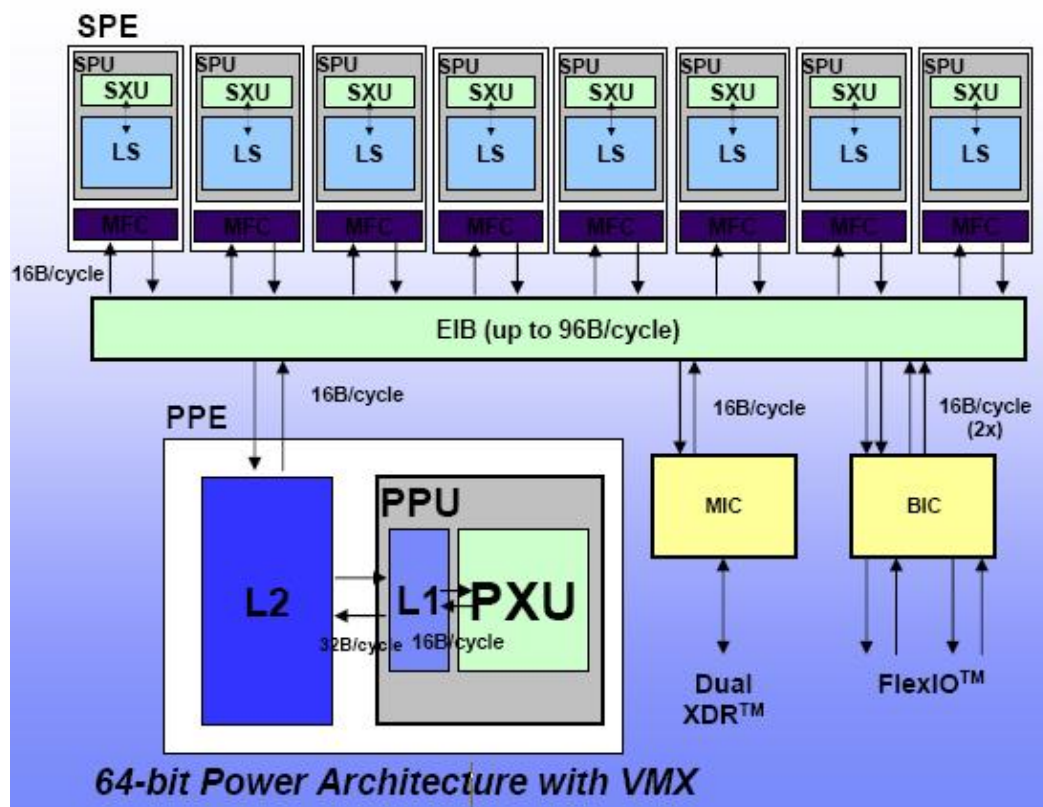


Fig. 2.2 Overview of Cell Architecture

Figure 2.4a and 2.4b has already shown the architecture and some of the major components of Cell BE. We are going to give more details about the PPE and SPE.

### **2.3.1 Power Processor Element (PPE)**

This is a general purpose, 64-bit PowerPC architecture based and two-way multi-threaded processor. It has 32 KB L1 cache and 512KB L2 cache. The PPE acts like a 64-bit PowerPC processor, allowing the execution of both operating system and applications.

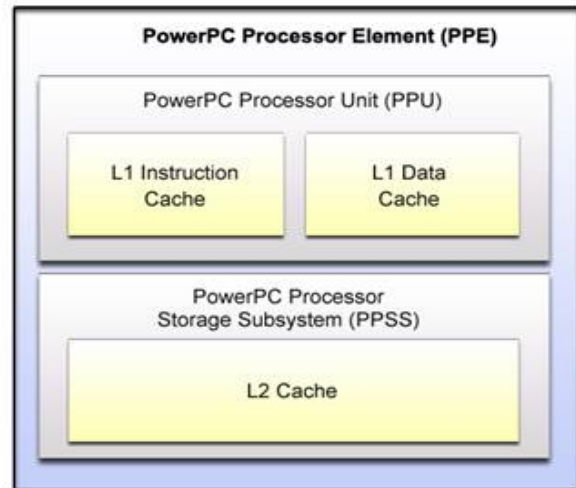
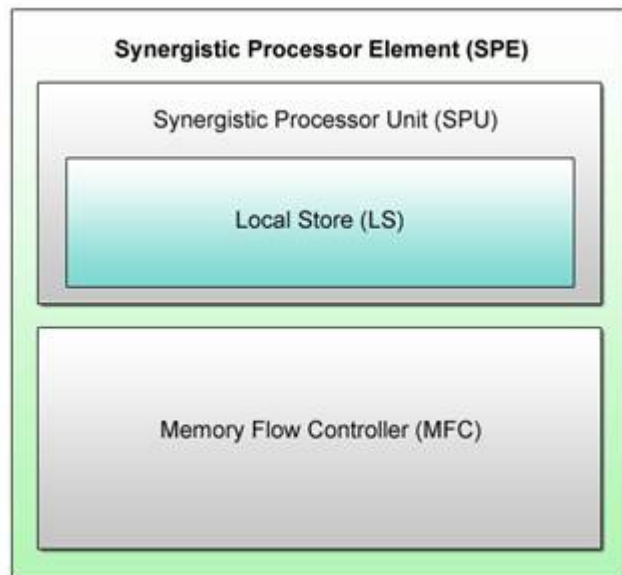


Fig. 2.3 Design of PPE

The PPE is designed as a control-intensive processor in Cell which mainly process control, including:

1. The I/O of accessing the main memory and other external devices requested by the operating system
2. The control over all 8 SPEs

### 2.3.2 Synergistic Processor Element (SPE)



The SPE is less complicated than the PPE, since it is target to provide computation performance, not control-intensive one. Every SPE consists of three main parts:

Fig. 2.4 Design of a SPE

1. A Synergistic Processor Unit (SPU), to perform its allocated task.
2. A 256KB Local Store (LS), which is the only memory accessible by the SPU.
3. A Memory Flow Controller (MFC), to control data transfer between the SPE's LS and the main memory or other SPEs.

There totally 8 SPEs in the Cell BE. For the Cell BE inside PlayStation®3, 1 SPE is disabled and 1 is reserved for the system software (i.e. the game OS), meaning that only 6 SPEs are accessible for programming under Linux.

## 2.4 Software

### 2.4.1 *Linux*

Although PlayStation®3 has its own system software, Sony has also opened the platform for third-party OS. The most common one is Linux.



Among the available Linux

distributions, we finally chose Fedora 7, with kernel updated to 2.6.23, as our developing environment. This is mostly because IBM officially declared that Fedora 7 is compatible with the Cell SDK, which is an important tool for our parallel programming.

### 2.4.2 *IBM Cell Software Development Kit*

The IBM Cell Broadband Engine SDK provides a complete Cell BE development environment. The SDK contains important tools for our parallel programming development, including:

- Libraries (SPE Run-time Management Library, SIMD math library)
- Samples source code
- IBM XL C/C++ Alpha Edition for Cell BE Processor
- GNU GCC compilers for PPU and SPU etc.

We use the version 2.1 to do our project in the first term. While in this term, we shift to use version 3.0.

## Chapter 3 Principals of Parallel Programming

---

We have already introduced the details of the principals of parallel programming in last term. Therefore in this chapter we will only review some important principals that we apply in the implementation of this term project. And that would include:

- Parallel Algorithm vs. Serial Algorithm
- SIMD Architecture
- Shared-Memory System and Distributed-Memory System

### 3.1 Parallel Algorithm vs. Serial Algorithm

Parallel algorithm is different from traditional serial algorithm. Traditional serial algorithm makes use of 1 CPU, executing command one by one and computes the final result. Parallel algorithm tries to make use of more than 1 processing unit for computing at a time. The result from each processing unit has to be put back together for the final result.

<b>Parallel algorithm</b>	<b>Serial algorithm</b>
multiple processing units	single processing unit
communication overhead	no communication overhead
higher complexity in code	straight forward code
ensure load balance between PU	everything is done by CPU

Table 3.1 Different between parallel algorithm and serial algorithm

The above table shows some different between parallel algorithm and serial algorithm. It is useful when we have to decide whether a problem should be solved by parallel algorithm or serial algorithm. Simply speaking, for problem which required heavy computation, parallel algorithm would be ideal as the communication overhead becomes negligible. For problem which has simple algorithm, it is not necessary to parallelize it as it increases the complexity of the code. Besides, the overhead in communication also becomes dominant.

## 3.2 SIMD Architecture

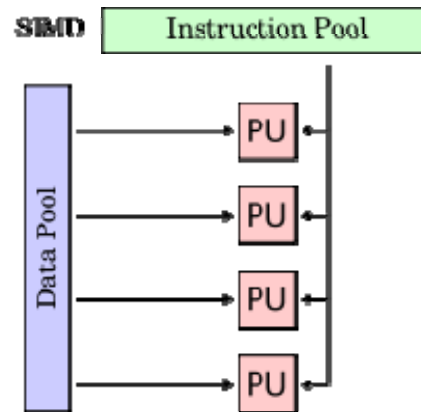


Fig. 3.1 SIMD architecture

In the case of single instruction multiple data (SIMD), the instruction stream of a computer is concurrently broadcast to multiple processors. Different stream of data are processed by different processors with the same instruction. For example, the SPE intrinsic functions provided by CellSDK are SIMD. They can apply same operation on every element of the input vector at the same time.

### 3.3 Shared-Memory System and Distributed-Memory System

Systems that are used to implement parallel program are classified into 2 types according to their distribution of physical memory, namely shared-memory system and distributed-memory system.

Shared-memory system refers to a system with large block of random access memory that can be access by several different central processing units in a multiple-processor computer system.

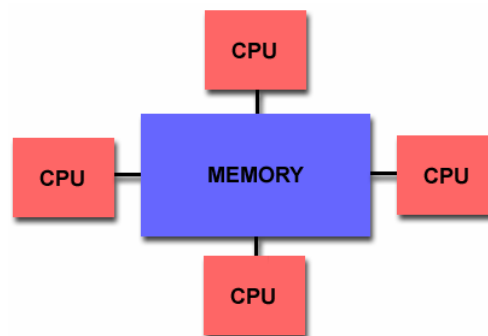


Fig. 3.2 Shared-memory system architecture

This type of system is relatively easy to program as every process access to same piece of data. They need a fast method to access central memory. Also, the memory coherence has to be maintained carefully.

In PlayStation®3, each SPE can access the main memory via Memory Flow Controller (MFC) by issuing Direct Memory Access (DMA) command. Therefore PlayStation®3 can be regard as a shared-memory system.



On the other hands, distributed-memory system refers to a multiple-processor computer system in which each processor has its own private memory. Data can be distributed to different processors for processing. After that, data has to be reassembled to generate a meaningful output.

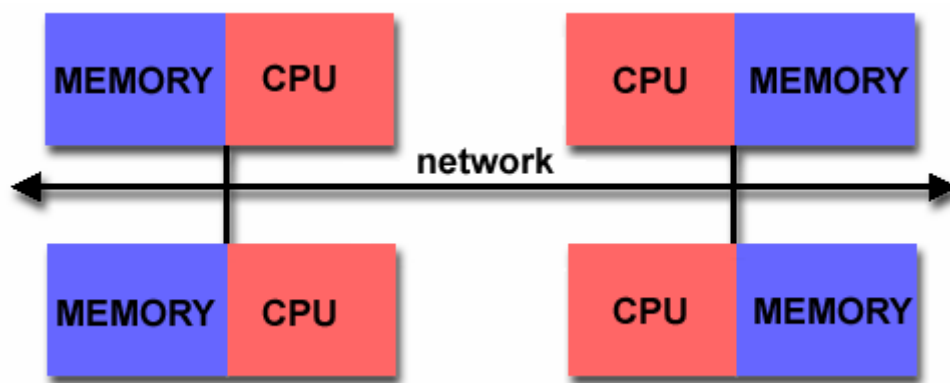


Fig 3.3 Distributed-memory system architecture

In PlayStation®3, each SPE has a local store (LS) of size 256 KB. LS are the working space for SPE. With LS, SPE can be assigned with task or data by the PPE, achieving parallel programming.

Actually, PlayStation®3 has both shared-memory and distributed-memory features. Thus, it can be classified as hybrid distributed-shared memory architecture.

This gives programmer a high degree of freedom when trying to parallelize a serial program or designing a parallel algorithm in PlayStation®3 platform. It makes PlayStation®3 an ideal environment for parallel program development.

### 3.4 Data Parallelism and Task Parallelism

Data parallelism is achieved by splitting data into smaller parts. Then distribute the smaller parts to different processing unit for computation. Usually, it happens within SIMD system where single set of instruction is used to operate on multiple set of data.

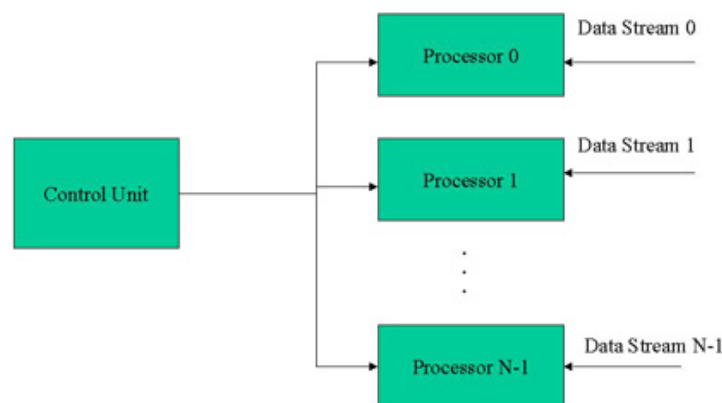


Fig 3.4 Data parallelism

Task parallelism is achieved by splitting program into different task. These tasks are assigned to different processing units. They can operate on same or different data. They can also communicate with each other and passing data around. Task parallelism usually occurred in MIMD system.

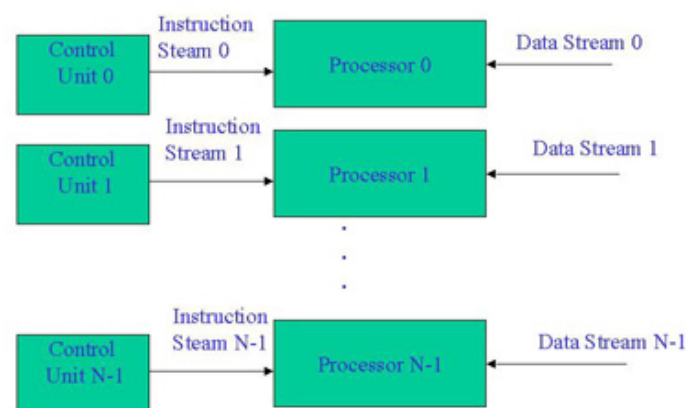


Fig.3.5 Task parallelism

## Chapter 4 Introduction of Problem and Solution

---

In this chapter, we will give a definition of the problem we target to solve, i.e. to produce Electronic Advertisement Guide (EAG). Then a high level description of our proposed solution will be demonstrated.

- Definition of Problem
- High Level Description of the Solution

## 4.1 Definition of Problem

Without many changes as last semester, we are given 2 streams of video segment. In which they contain different commercial advertisements. We have to distinguish all the repeated commercial advertisements out of the 2 video streams by some sort of comparison.

The final objective is to produce EAG (Electronic Advertisement Guide), which is something similar to EPG (Electronic Program Guide) that shows the starting and ending time of TV programs. There are differences between EAG (Fig. 4.1) and EPG (Fig. 4.2). EAG focus on commercial advertisements that have been shown on TV. EPG focus on TV programs that have yet been shown on TV.












02:20-02:25				
	ID: 00000013 (022143_022213) 30 sec	ID: 00000027 (022213_022243) 30 sec		
02:30-02:35				
	ID: 00000055 (023444_023504) 20 sec			
02:35-02:40				
	ID: 00000117 (023504_023524) 15 sec	ID: 00000023 (023524_023538) 15 sec	ID: 00000118 (023540_023554) 15 sec	ID: 00000014 (023555_023604) 10 sec
02:50-02:55				

Fig. 4.1 EAG example showing advertisements shown during different time period

The EAG example above shows the commercial advertisements appeared from 2:20-2:55 which include their corresponding ID in the database and the length of the advertisement. EAG company can provide service to their customer base on the above timetable. EAG only have records of commercial advertisements that have been shown on TV.

Week beginning 07. April 2008 to 13. April 2008, Thu. 10. April 2008

Mon Tue Wed **Thu** Fri Sat Sun ◀ This Week ▶ Help ?

Main Channels Freeview TopUp TV Entertainment Films Kids News and Business Sport Music Regional Net

BBC one BBC One	BBC TWO BBC Two	itv 1 ITV1 London	4
			
19.30 EastEnders <a href="#">SERIES</a>	21.00 Live Golf: The Masters	19.00 Emmerdale <a href="#">SERIES</a>	22.00 <a href="#">SERIES</a>
▶ BBC One - Morning	BBC Two - Morning	ITV1 London - Morning	Chann
▶ BBC One - Afternoon	BBC Two - Afternoon	ITV1 London - Afternoon	Chann
<b>BBC one Evening</b>	<b>BBC TWO Evening</b>	<b>itv 1 Evening</b>	<b>4</b>
18.00 BBC News	18.00 Eggheads	18.00 London Tonight	18.00 <a href="#">SERIES</a>
18.30 Regional News Programmes	18.30 Great British Menu	18.30 ITV Evening News	18.30 <a href="#">SERIES</a>
19.00 The ONE Show	19.00 Live Swimming	19.00 Emmerdale <a href="#">SERIES</a>	19.00 <a href="#">SERIES</a>
19.30 EastEnders <a href="#">SERIES</a>		19.30 Tales from the Country	19.55
20.00 HolbyBlue <a href="#">SERIES</a>	20.00 Coast	20.00 The Bill <a href="#">SERIES</a>	20.00
21.00 Who Do You Think You Are?	21.00 Live Golf: The Masters	<a href="#">SERIES</a> Sins of the Father - Part Two	21.00
		21.00 New Homes from Hell	21.00

Fig.4.2 EPG from tvtv.co.uk

The EPG example above is a screenshot example from tvtv.vo.uk. It has detail listing of program title together with their starting time and screenshot. EPG also contains program listing of different TV channel in the near future. Therefore user can also choose to view the program guide of a different date.

## 4.2 High Level Description of the Solution

In this part we will talk about our proposed solution to solve the EAG problem. We hope to let reader first have a brief understanding of our proposed solution to this problem before we are going deeply into the algorithm and implementation.

The whole procedure can be divided into 2 parts. Part 1 is about given raw video streams, how we convert them into series of H13 files which is suitable for video comparison. Part 2 is about how we make use of the H13 files captured from part 1 to do the comparison and produce the final EAG. The overview of the whole procedure is illustrated below.

## 1. Converting raw video data into series of HI3 files

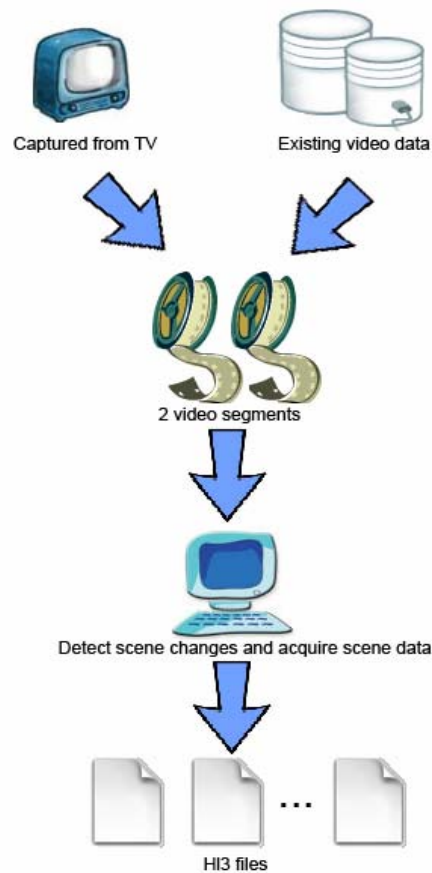


Fig. 4.3 Procedure to obtain HI3 files from raw video data

In our problem, we focus on 2 video segments. They are either captured in real time on TV or obtained from existing database. In which they contain only commercial advertisements.

These videos are processed in a PC with a program which could generate a series of video frame digest (in HI3 format). The way of doing this is black boxed to us. However, we will talk about some basic concept of doing this in later part of our report. Recently a new format called HI4 is under testing which should be able to improve the accuracy of the final result as well as reducing the file size.



## 2. Processing HI3 files to become the final EAG

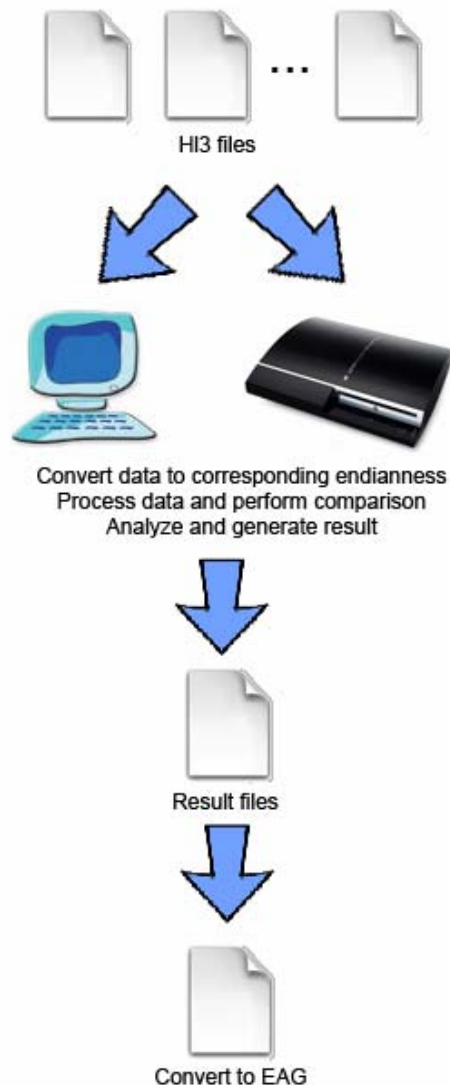


Fig. 4.4 Procedure of processing HI3 to become final EAG

PC with X86 structure uses little-endian format while PlayStation®3 with PowerPC structure uses big-endian format to represent data.

Therefore after obtaining HI3 files data, depends on the platform we are going to use. We have to first convert data to the corresponding endianness. We have written a program to do the job.

After data are in correct format, we can feed them to our program which will then perform video comparison. The detail implementation of this program is illustrated in the following chapter. This is the part we were focusing on in this semester.

We repeat the same job on 2 platforms just to compare the processing power between two. In practical, any one of the platform can have the job done. The result of performance comparison is demonstrated at chapter 6. From our experience, using parallel algorithm with PlayStation®3 gives us a speed up of about 570% comparing with serial algorithm on PC.

Result file is generated by our program. One more step is needed to convert the result file into the EAG. We make use of the macro function of Microsoft Excel to generate the final EAG. The result file is imported into the excel file and EAG is generated. Here is a screenshot of the debug version of the EAG, which contains even more information.

	A	B	C	D	E	F	G
100	2008.02.12.01.20.56.0875.R.25.h13	1157			2008.02.13.01.25.00.0531.R.25.h13	1172	
101	2008.02.12.01.20.56.0921.T.1.h13	46			2008.02.13.01.25.00.0578.T.1.h13	47	
102	2008.02.12.01.20.58.0812.R.40.h13	1891			2008.02.13.01.25.02.0390.R.39.h13	1812	
103	End of adv. 7				End of adv. 7		
104	Start of adv. 8				Start of adv. 8		
105	2008.02.12.01.24.06.0421.T.1.h13	187609			2008.02.13.01.23.30.0046.T.1.h13	-92344	
106	2008.02.12.01.24.16.0312.R.213.h13	9891			2008.02.13.01.23.40.0000.R.217.h13	9954	
107	End of adv. 8				End of adv. 8		
108	Start of adv. 9				Start of adv. 9		
109	2008.02.12.01.23.16.0406.T.1.h13	-59906			2008.02.13.01.23.40.0062.T.1.h13	62	

Fig. 4.5 Final EAG produced using the result file

Now reader should have a brief idea on how we do the video comparison to locate common video segments (commercial advertisement in our case) in 2 video segments. The actual and detail implementation and some key changes to the algorithm are presented in the next chapter.

## Chapter 5 Low Level Implementation Details

---

This chapter includes the details of the raw data, implementation and the modification we have made:

- Converting raw video data into series of HI3 files
  - ◆ Endianness Conversion
- The Main Program
- Processing HI3 Files to Become the Final EAG
  - ◆ Further Data Conversion
  - ◆ The Main Program
- Modification of LCS algorithm
  - ◆ Equality of 2 Symbols
  - ◆ More Restriction on the EAG problem from R and T Frame
  - ◆ Logic System for “Equality” Comparison
  - ◆ Splitting Advertisements from LCS Result & Analysis Strings
  - ◆ Synchronizing Flags
  - ◆ Flags Propagation and Printing Result
  - ◆ Single Pass LCS to Multiple Passes LCS
  - ◆ Convert to EAG
- Speeding Up in PlayStation®3 Platform

In this chapter we will present to our reader everything in detail including our algorithm design, reasoning for doing in such way, difficulties we face and further improvement suggestion. We recommend reader to finish the previous chapter to have an overview of the problem first before hopping into this chapter.

## **5.1 Converting raw video data into series of H13 files**

In this project, we target on some data which is captured from advertisement-intensive period, e.g. TV Easy (宣傳易). We are provided with data to do the video comparison, which include MPEG videos, JPEG files and H13 files. Thanks our project manager Edward Yau for providing these data. So basically we don't have to worry about how to obtain these data. However, some basic knowledge and concept on how these data are generated is still necessary for coming up with a better design of our program.

What is more, starting from this semester, our project manager did some changes on the H13 files being generated. This has great impact to our algorithm used in part 2. That's why some detail explanation is needed here.

The original approach of this part is to compute a series of video frame digest, which is in form of H13 format, out of the video segments base on a constant frequency.

However, if a high frequency is used, there will be too much output H13 files. Moreover, among these frame digest, most of their contents are duplicate. This results in a waste of both processing time and memory spaces.

For example if a frequency of 25 frames per second (the frequency of PAL standard for TV) is used. In a 1-hour-long video there would be:  
 $3,600 \text{ seconds} \times 25 = 90,000 \text{ frames}$



Fig. 5.1 Frames captured with a high frequency

On the other hand, if a small frequency is used, the final result may not be accurate enough as intermediate key frames may be missing.



Fig. 5.2 Frames captured with a low frequency

In most common video sources, such as TV programs, consecutive frames in the video usually have only small changes. Drastic changes occur when there is scene change, e.g. a new shot. Base on this fact, storing the key frames and discarding the intermediate one would be sufficient to represent the information of a video.

Advertisements recognizing can be satisfied by comparing only the key frames and not all frames. If two advertisements are the same, they would have the same or very similar key frames throughout the videos. Since intermediate frames between key frames can be ignored. This greatly reduces the number of frames which is needed to be processed and compared. The number of frames out of a 1-hour-long video is reduced from 90,000 to around 8,000. As a result, both computation time and memory spaces are saved.



Fig. 5.3 Key frames captured whenever scene change is detected

So how exactly key frames are captured? Whenever a scene change is detected, the frame just before it, called 'R frame' and the frame just after it, called 'T frame' are captured (Fig. 5.4). These frames are then outputted as JPEG files first. JPEG file stores the screenshot of that frame. The time information of the frame is used as the filename (Fig. 5.4).

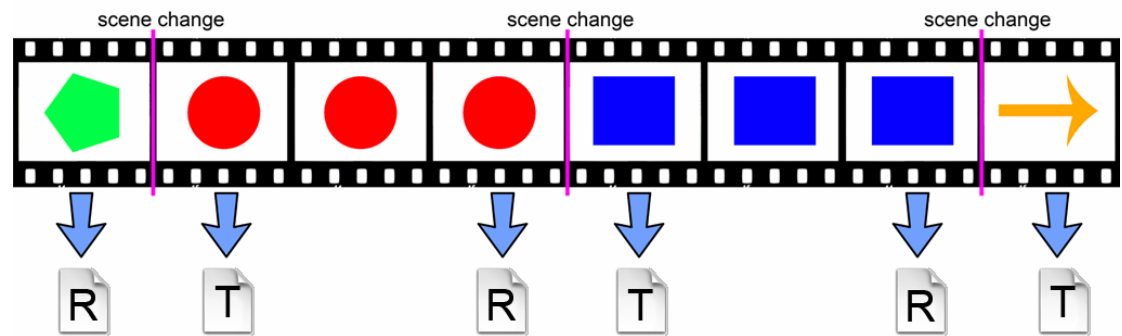


Fig. 5.4 The R and T frame outputted when scene change is detected.

In order to do comparison on the frames, the JPEG files are further converted into HL3 files. In a single HL3 file, it contains a 32 x 32 array, storing 1,024 integers. These numbers represent a JPEG file. For similar frames, their array would not vary by much.

With these HL3 files, we are able to determine the similarity between two frames by computing their difference in the integer array. The filename of HL3 files are the same as the JPEG files except the file name extension. It contains time information of that frame.

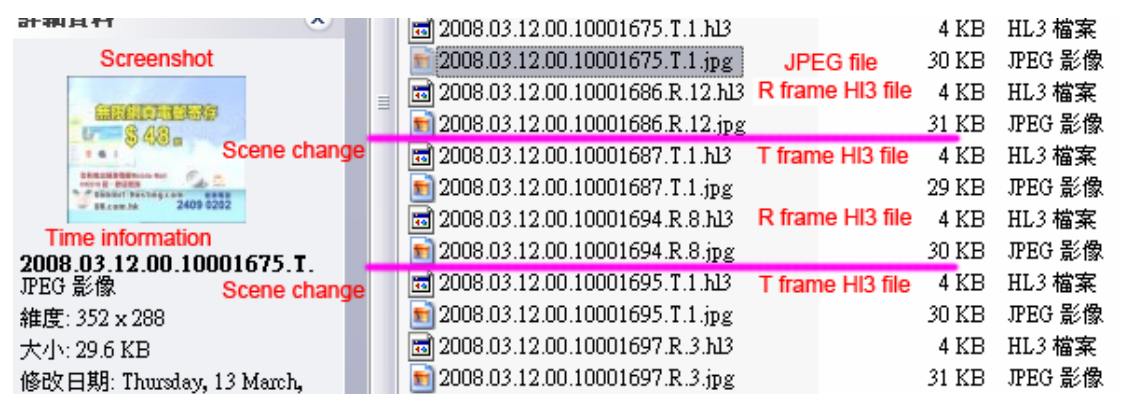


Fig. 5.5 List of outputted file generated



## 5.2 Processing HL3 Files to Become the Final EAG

Not like last semester, when we were focusing on how to speed up our program using parallel computation. This semester we were focusing on applying the knowledge we have learnt during last semester to implement a better approach to solve the problem.

This part is rather complicate as it involves a lot of low level implementation. Therefore it is further broken down into several parts in details.

### 5.2.1 Further Data Conversion

Mora data conversions are needed after the HL3 files are generated from the raw video. These include:

#### **Endianness Conversion**

In this project, we do the implementation on two different platforms, the PC with Intel x86 processor, and the PlayStation®3 with Cell processor, which is in PowerPC architecture.

The program running on both platforms are using the same set of data, i.e. the HL3 files. As we mentioned, the HL3 files contains 1024 integers, and they are stored in binary format, as sequences of bytes. However, the two platforms represent the byte values in different order, i.e. different endianness.

For PC (x86 architecture), it uses little-endian format, which means increasing numeric significance with increasing memory addresses.

For PlayStation®3 (PowerPC architecture), it uses big-endian format, which means decreasing numeric significance with increasing memory addresses.

Consider an example of storing a 32-bit integer 0x0A0B0C0D

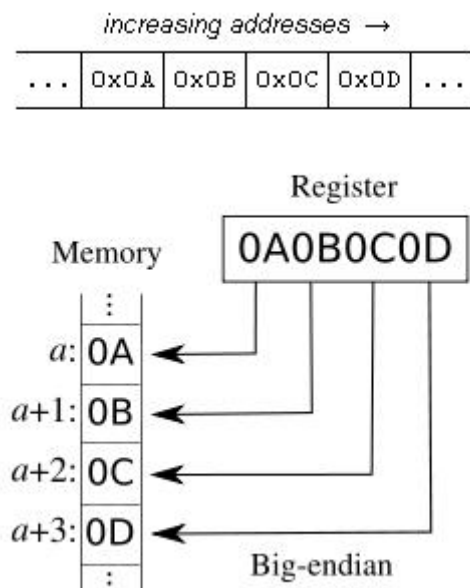


Fig. 5.6 PC Integer Representation

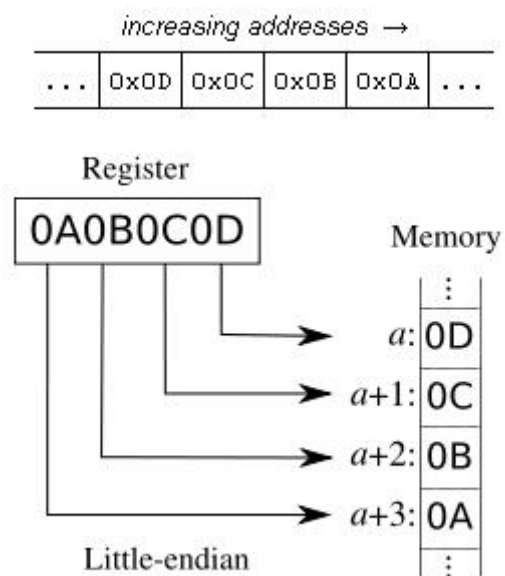


Fig. 5.7 PS3 Integer Representation

Therefore, we need to convert the endianness of HL3 data when moving them from PC to PlayStation®3. This is done by shifting and rearranging the 4 bytes of an integer.

```
int convertEndianInt (int i)
{
    unsigned char c1, c2, c3, c4;
    c1 = i & 255;
    c2 = (i >> 8) & 255;
    c3 = (i >> 16) & 255;
    c4 = (i >> 24) & 255;
    return ((int)c1 << 24) + ((int)c2 << 16) + ((int)c3 << 8) + c4;
}
```

### **Type Conversion**

As we found out in last semester, using SIMD (single instruction multiple data) intrinsic instruction, which is provided by the CellSDK, would improve arithmetic computation time a lot. These SIMD instructions take 128 bits register as input. Then apply operation on every element of the vector at the same time. This means that we can pack 4 integer values from the HL3 files in the 128 bits register.

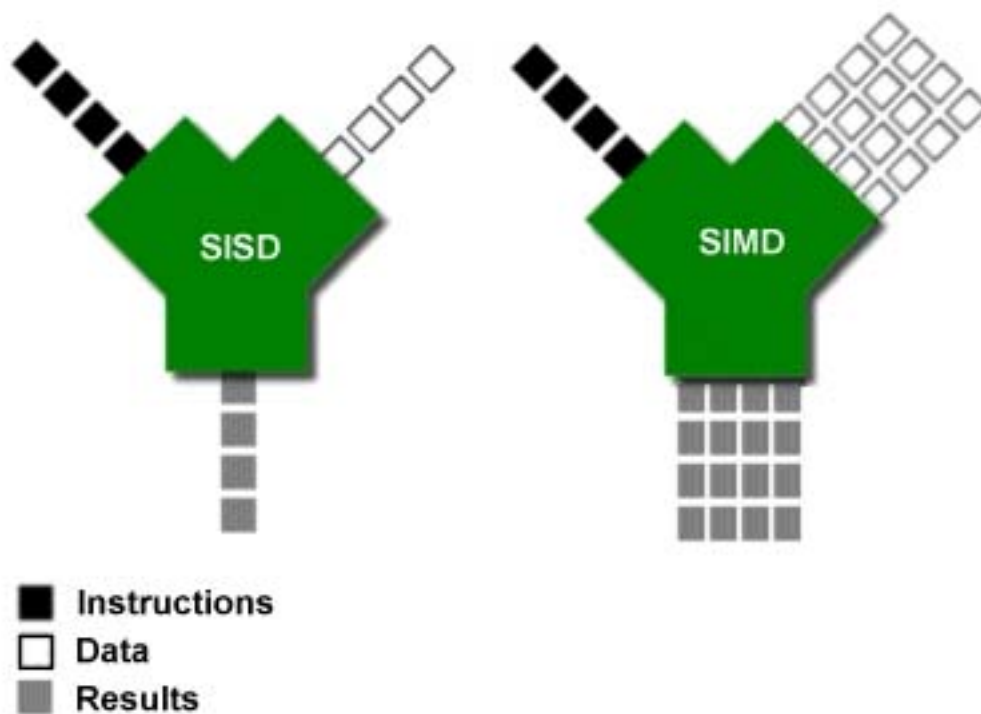


Fig. 5.8 Difference in Operation using SISD and SIMD

However, the original design of SPE is to handle floating point number. There are nearly no SIMD instructions for integer data type. Hence in the PlayStation®3 platform, we have to convert the type of the HL3 files from integer array to floating point number array, while still keeping the precision of the numbers.

### **T-R Combination**

To implement the algorithm in a Longest Common Subsequence approach, we need to have a sequence of alphabet. We find that the best way to map our HL3 data to alphabets for LCS would be combining a HL3 of T frame to the next HL3 of R frame.

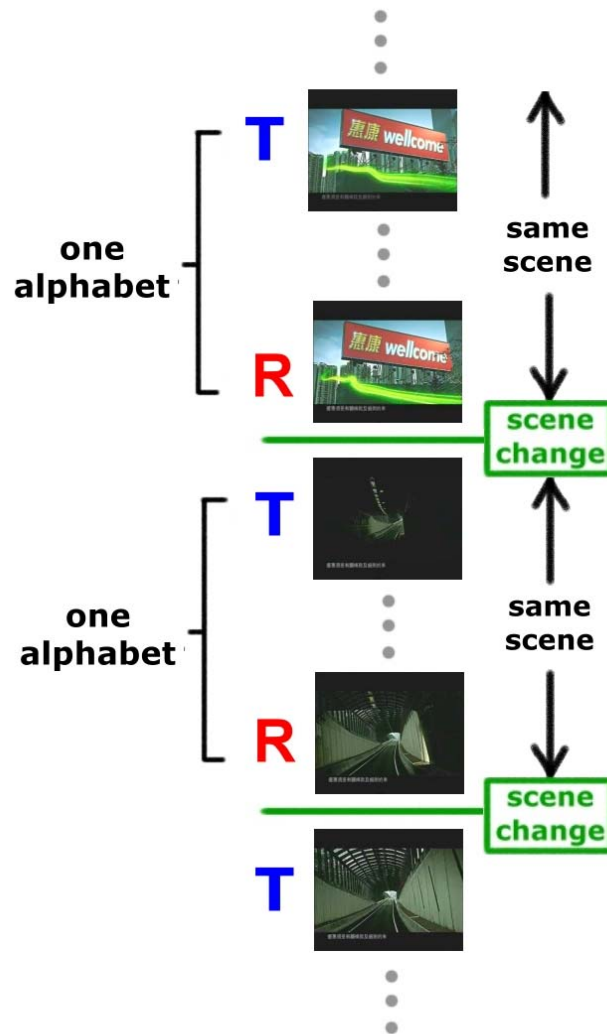


Fig. 5.9 Mapping T-R pair to be 1 alphabet

As a result, a pair of T-R frames is regarded as one alphabet and is computed together. The input data has to be start with a T frame and end with a R frame so that all frames can form pairs.

To conclude, we are processing little-endian integers in PC and big-endian floating point numbers in PlayStation®3. Both start with a HL3 file of R frame and end with one of T frame

## 5.2.1 The Main Program

### ***5.2.2.1 Last Semester Approach – Review on Minimum Difference Algorithm***

We will first have a short review on the minimum difference algorithm that we adapted during last semester. We used such a simple algorithm since we were focusing on how to speed up of the program, not much on the accuracy of the result. After reading this part, reader should have a refreshed memory on what algorithm we have used during last semester. Then we will briefly talk about the weaknesses of this algorithm in solving the video comparison problem.

The basic idea of minimum difference algorithm: Given two directories ("Repository" and "Target") representing two video segments, for each HI3 file in "Target" folder, we need to map it to another HI3 file in "Repository" directory such that square of their Euclidean distance is smallest.

Mathematically: Given a "Target" file  $P(p_1, p_2, \dots, p_k)$ , where  $p_i$  is the  $i^{\text{th}}$  integer in the file, we try to look for another file  $Q(q_1, q_2, \dots, q_k)$ , where  $q_i$  is the  $i^{\text{th}}$  integer in the file, in "Repository" such that the value is minimum. By doing this, each frame in the "Target" folder is mapped to the most mathematically resemble frame in the "Repository" folder.

Assume there are  $m$  files in the "Target" directory and  $n$  files in the "Repository" directory, the above algorithm is  $O(m \times n)$ .

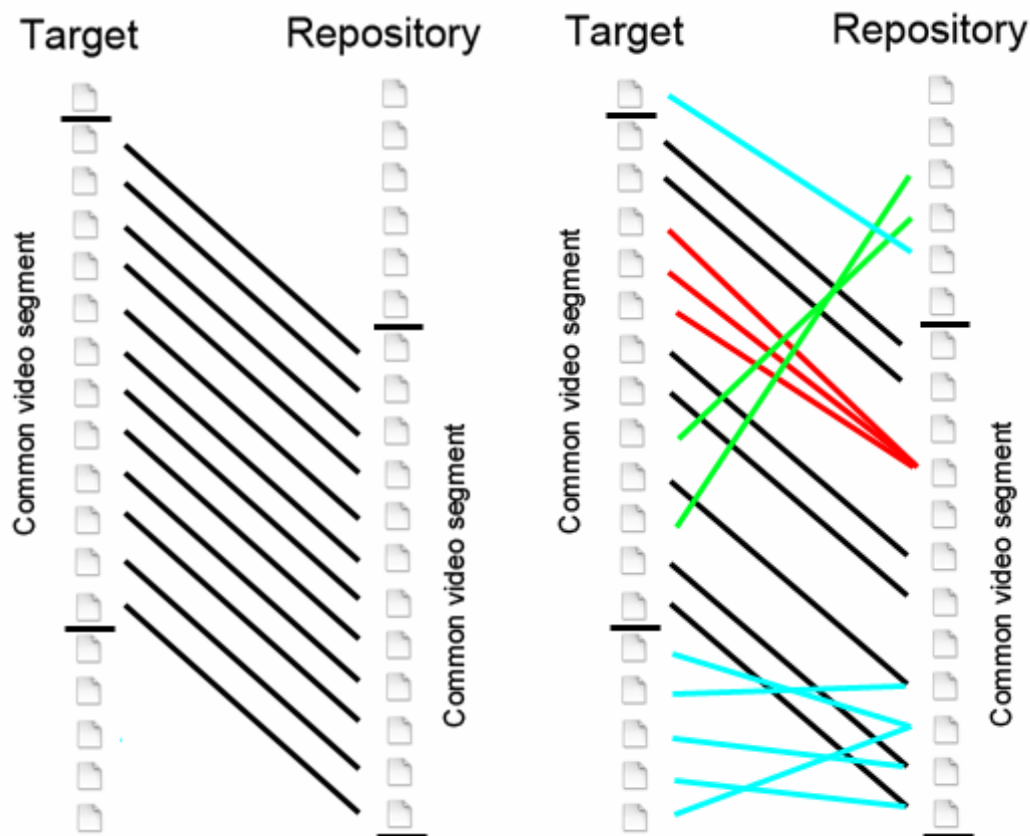


Fig. 5.10 Ground truth result and possible matching result generated with minimum difference algorithm

We will now illustrate the problem of the minimum difference algorithm. Consider the case in Fig. 5.10 above. The left part shows the ground truth. In which only the common video segment are matched with each other. The right part shows the matching result generated with minimum difference algorithm. It shows three type of matching problem of this algorithm with different colors. The main weakness of minimum difference algorithm can be seen from above.

Problem 1: As illustrated in “red” lines, minimum difference algorithm allows different “Target” file matched to the same “Repository” file. In ideal case like the ground truth result, this simply won’t happen as each file in the common video segment of “Target” should be matched with the corresponding file in the common video segment of “Repository”. This is because we assume that scene change and HI3 file generated with the same video segment should be exactly the same. Of course in practice, noise in video segments may make this assume wrong. However, we believe that in most case, matching multiple “Target” file to single “Repository” file won’t give us optimal result. That’s why we need a need algorithm to fix this problem.

Problem 2: As illustrated in “green” lines, there are out of phrase matching. We can think about it this way. When we are matching 2 common video segments, the relative position of the matched file should not vary by many. For example, in the ground truth result, 2<sup>nd</sup> file matches with 7<sup>th</sup> file, 3<sup>rd</sup> file match with 8<sup>th</sup> file and etc in a consistent way. While in minimum difference algorithm, since each “Target” file is considered as independent, it is possible for an HI3 file in the common video segments matches with a file that have totally no relation to the common video segments. Result in an out of phrase matching which is obviously a mismatch.



Problem 3: As illustrated in “blue” lines, showing another limitation of minimum difference algorithm. It is the problem that for this algorithm, every file in the “Target” has to find a match. This is wrong as shown in ground truth matching. In concept, we want to do a match if and only if the same frame appears in both video streams. So, a new algorithm that can solve the above 3 problems is definitely needed. Here we introduce our approach – multi-pass LCS algorithm.

### ***5.2.2.1 Introduction on Longest Common Subsequence***

Reader may wonder, what is the relation of the longest common subsequence problem and the EAG problem? Actually, we will introduce how we apply the longest common subsequence algorithm to the EAG problem using an analog approach. But first, let's have a review on the longest common subsequence problem first.

Longest common subsequence (LCS) problem <sup>[1]</sup> according to the definition is the problem of finding a maximum length subsequence of two or more strings. While The definition of subsequence <sup>[2]</sup> is as follows: Any string that can be obtained by deleting zero or more symbols from a given string. LCS is used in bioinformatics for comparing the longest common DNA sequence between 2 DNA sequences.

For example given 2 DNA sequences:

ACGGT  
AGCTC

Their longest common subsequence is AGT (in red) or ACT (highlighted in yellow). In this simple example, we know that it is not rare that there is more than one solution.

The most common method to solve this problem is by using dynamic programming approach. The formula and explanation quoted from Wikipedia <sup>[3]</sup> is given as follows:

Given the sequences  $X_{1...m}$  and  $Y_{1...n}$

$$\text{LCS}(X_{1...i}, Y_{1...j}) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ \text{LCS}(X_{1...i-1}, Y_{1...j-1}) + x_i & \text{if } x_i = y_j \\ \max(\text{LCS}(X_{1...i}, Y_{1...j-1}), \text{LCS}(X_{1...i-1}, Y_{1...j})) & \text{otherwise} \end{cases}$$

Here + denotes concatenation, and max gives the longest sequence.

The rationale for this recurrence is that, if the last characters of two sequences are equal, they must be part of the LCS. You could never get a larger LCS by matching  $x_m$  to  $y_j$  where  $j < n$ , and vice versa. If they are not equal, check what gives the largest LCS of keeping  $x_m$  and  $y_n$ , and use that.

Let see how implementation is done for this dynamic programming problem. Using the example above again, basically we want to fill in a table as following:

i\j	A	G	C	T	C	
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
G	0	1	2	2	2	3
G	0	1	2	2	2	4
T	0	1	2	2	3	5
	0	1	2	3	4	5

Table 5.1 Result table comparing 2 strings ACGGT and AGCTC

The sample code for serial computation of the result table is given below:

```

for i := 0..m
    C[i,0] = 0
    for j := 1..n
        C[0,j] = 0
    for i := 1..m
        for j := 1..n
            if X[i] = Y[j]
                C[i,j] := C[i-1,j-1] + 1
            else
                C[i,j] := max(C[i,j-1], C[i-1,j])

```

After some initialization, it starts from (1, 1), filling up the table up to (m, n). When filling the cell (i, j), information from (i-1, j-1), (i, j-1) and (i-1, j). This actually gives some limitation on the design of parallel algorithm but we will talk about it later. The above algorithm is in  $O(m \times n)$ . After obtaining this table, we have to do backtracking in order to retrieve the final LCS. The table and code below illustrated how we do the backtracking.

i\j	A	G	C	T	C	
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
G	0	1	2	2	2	3
G	0	1	2	2	2	4
T	0	1	2	2	3	5
	0	1	2	3	4	5

Table 5.2 Backtrack result of LCS, red = AGT, yellow highlight = ACT

```
function backTrack(C[0..m,0..n], X[1..m], Y[1..n], i, j)

    if i = 0 or j = 0
        return ""
    else if X[i] = Y[j]
        return backTrack(C, X, Y, i-1, j-1) + X[i]
    else
        if C[i,j-1] > C[i-1,j]
            return backTrack(C, X, Y, i, j-1)
        else
            return backTrack(C, X, Y, i-1, j)
```

This piece of code actually gives only one of the results. For simplicity, we will only handle the result given by this algorithm but not all. Starting from  $(m, n)$ , it finds its way back to  $(0, 0)$ . It output a character and jump to the diagonal cell when it which  $(i, j)$  and  $X_i$  equals  $Y_j$ , else it walks toward a cell with larger value. This worst case of this algorithm takes  $m + n$  steps. Thus it is  $O(m + n)$ .

Combining with the previous part, the whole algorithm is  $O(m \times n)$ , which is the same as the minimum difference algorithm.

### 5.2.2.1 Why and How LCS Work on Video Stream Data

So what is the relation of LCS with the EAG (video comparison) problem? Consider we treat each video data (H13 file) as a character in the character string of the LCS problem. Fig. 5.11 shows an analog of the LCS problem on DNA sequence with the EAG problem. The idea of LCS is to look for the longest common segments. This is applicable to our EAG problem as TV commercial advertisement is like a continuous sequence of files. If this sequence of files appears at both “Target” and “Repository” side, we can then recognize them using the LCS algorithm.

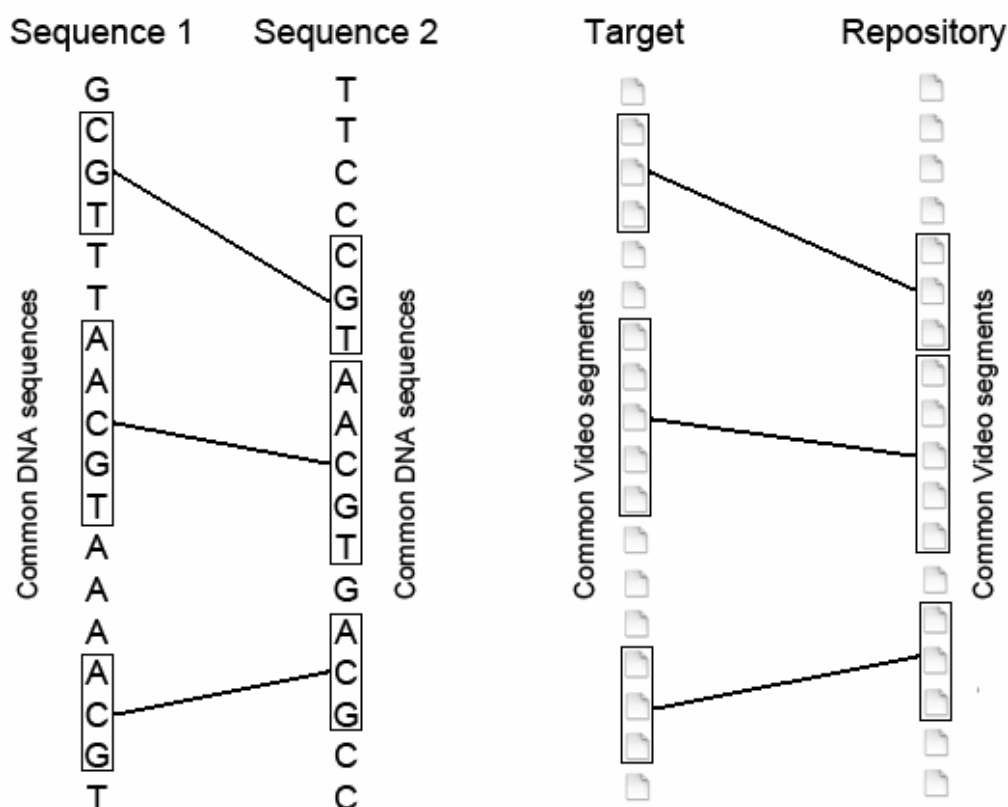


Fig. 5.11 An analog of the LCS on DNA sequence problem with the EAG problem

Refer back to 5.2.2.1, the 3 problems of minimum difference algorithm that we have come across are:

1. It matches multiple "Target" files to single "Repository" files.
2. It allows out of phrase matching within a common video segment.
3. It forces all "Target" files to find a match to a "Repository" file.

Now we can observe from Fig. 5.11 above that LCS algorithm actually solve all of these problem.

Firstly, by definition of "common" in LCS, it is just impossible for multiple "Target" files to single "Repository". This is because if multiple files are match to single file, then it means the "Repository" file is common to the "Target" files, but the "Target" files is NOT common to "Repository" files.

Secondly, by definition of "subsequence" in LCS, it is also impossible for out of phrase matching to occur. If out of phrase matching is allows, then the output sequence will not be a "subsequence" of the original 2 sequences.

Thirdly, this LCS algorithm won't force all the character to find a match. Therefore it solves the 3<sup>rd</sup> problem and conceptually it is much more accurate.

Everything sounds perfect now. Can we immediately solve the EAG problem by using LCS algorithm? The answer is no, but it can be done with some modification to the LCS problem.

## 5.3 Modification of LCS Algorithm

We are now applying LCS algorithm working which works for 2 characters string to 2 video stream data (HI3 files). Some modifications are needed and are stated in the following parts.

### 5.3.1 Equality of 2 symbols

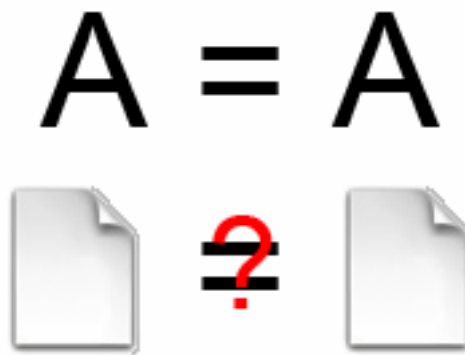


Fig. 5.12 We can compare 2 characters easily, but how about HI3 files?

In LCS algorithm (in both computing result table and backtracking, please refer to 5.2.2.2), the only comparison between 2 input character string is this statement:

**if**  $X[i] = Y[j]$

That means if we apply LCS algorithm to video data, we only need to handle the “Equality” of 2 HI3 files. It is simple right? An HI3 file is basically just an array of 1024 integer/float (please refer to 5.2.2.1) digits. So they can be considered as equal if all the digits in both HI3 files are identical. This is correct, ideally.



Practically, it is nearly impossible for 2 HI3 files to be identical.

Remember how HI3 files are generated? (Refer to 5.1) 2 JPEG files are captured whenever scene change is detected. One is the frame right before scene change (R frame); another is the frame right after scene change (T frame). Then each JPEG file's information is extracted and converted to an HI3 files.

Notice a few problems here related to the video source:

1. Things like brightness, contrast gives variation in optical view to same frame, but they should be treated as "equal".



Fig. 5.13 Brightness gives different optical view, but should be treated as equal

2. There may be noise in the source video. These noises will vary the content of JPEG as well as HI3 files by little. Sometimes even affect the scene change detection.
3. The frequency of the video sources may not be the same. For example, 1 may play at 25 fps; another may play at 30 fps. This will vary the scene change occurrence time by a little.

4. The videos, either recorded from TV, or obtained from database, are compressed by unknown number of time already from the raw sources. This would surely affect the output files extracted from these sources.
5. JPEG is a compressed format. Some information in the scene is lost during compression, leading 2 similar frames to compress to same frame. This is unlikely to happen though, but still possible.

Actually, this is why we use H13 file format for comparison. The idea of H13 file is that it can be act as an indicator to show how resemble 2 frames are. The smaller the square of Euclidean distance between 1024 dimension vectors in the 2 files, the more resemble the 2 files are, in term of optical viewing. There the formula to compute the "equality" between 2 H13 files is given below.

Given H13 file P  $(p_1, p_2, \dots, p_{1024})$

H13 file Q  $(q_1, q_2, \dots, q_{1024})$

P equals Q if and only if  $\sum_{i=1}^{1024} (p_i - q_i)^2 < \text{equality threshold}$

So it is done? No, this is only part of the story.

### 5.3.2 More Restriction on the EAG Problem from *R* and *T* Frame

If we use the algorithm to generate a result, there is a problem.

Remember we have 2 types of frames? That's right, the R frame and T frame. Please look at the result in Fig. 5.14 and Fig.5.15 below. If the LCS algorithm is applied directly, it is possible for an R frame matching with T frame and T frame matching with R frame. Why? This is because the system cannot distinguish between R and T frame in a sequence of files. It treats them both as a HI3 files.

5	2008.02.12.01.24.39.0718.T.1.M3	47		2008.02.13.01.22.21.0208.T.1.M3	47	
6	2008.02.12.01.24.39.0906.T.1.M3	188		2008.02.13.01.22.21.0875.R.4.M3	172	
7	2008.02.12.01.24.45.0968.R.42.M3	6062		2008.02.13.01.22.24.0812.R.48.M3	2987	

Fig.5.14 A T frame matched with R frame

2008.02.12.01.21.13.0875.T.1.M3	63		2008.02.13.01.22.52.0890.T.1.M3	47	
2008.02.12.01.21.16.0609.R.4.M3	2734		2008.02.13.01.22.55.0208.T.1.M3	2813	
2008.02.12.01.21.20.0609.R.82.M3	4000		2008.02.13.01.22.59.0187.R.70.M3	3984	

Fig. 5.15 A R frame matched with T frame

Ideally, again! We want only R frame matched with R frame, T frame matched with T frame. The reason is this should be the way for an ideal matching to be happened. The frame right before scene change happened should always matches with a frame before scene change. And so do the frame right after scene change.

This is the restriction of the EAG problem due to the introduction of R and T frame which aims at saving storage and computation time.

We thought of a 4 ways to encounter this restriction, actually only the last method works. We will give reader an introduction on the methods we have thought about.

### 1. Bypassing T-R and R-T comparison

The first trial is to try to skip the comparison between T-R or R-T pairs when computing the result table.

i\j		R	T	R	T	R	T	
	0	0	0	0	0	0	0	0
R	0	1						1
T	0							2
R	0							3
T	0							4
R	0							5
T	0							6
	0	1	2	3	4	5	6	

Table 5.3 Skipping the comparison between T-R or R-T pairs

This method does not work if you remember the algorithm to compute the result table for LCS. "When filling the cell  $(i, j)$ , information from  $(i-1, j-1)$ ,  $(i, j-1)$  and  $(i-1, j)$ ." As can be observed above, skipping comparison of R-T and T-R frame makes R-R and T-T frame comparison not possible.

## 2. Splitting frames into R group and T group

The second trial is to split frames into R and T groups. Then treat them as 2 separate LCS problem as shown below.

i\j	T	T	T	
	0	0	0	0
T	0	1	1	1
T	0	1	2	2
T	0	1	2	3
	0	1	2	3

i\j	R	R	R	
	0	0	0	0
R	0	1	1	1
R	0	1	2	2
R	0	1	2	3
	0	1	2	3

Table 5.4 Splitting frames into R group and T group

There is an advantage about this method. It reduces the problem size from  $m \times n$  into  $m/2 \times n/2 + m/2 \times n/2 = 1/2 (m \times n)$ . Therefore we save half the computation time with this restriction!

However, this method leads to a similar out of phrase matching problem as observed from the minimum difference algorithm. This is because the action to split the sequence into T and R groups destroys the continuity property between T and R frame. Now we are like matching T and R frame independently with each other.

Another problem is that: How are we going to combine the 2 LCS results into 1? If we try to combine them, we have to destroy the "subsequence" property of LCS algorithm. As mentioned above, the 2 LCS result generated maybe out of phrase with each other.

### 3. R-T combination

From the experience above, we know that the restriction of R and T frame matching can actually half the computation time. However we still want to preserve continuity of the data during the matching stage. Here we introduce our third approach, R-T combination approach.

The basic idea is that we try to group an R-T frame pairs in the sequence into a single symbol. Analogy, these 2 frames (R and T HI3 files) now become a character in the LCS character string (Fig. 5.16).

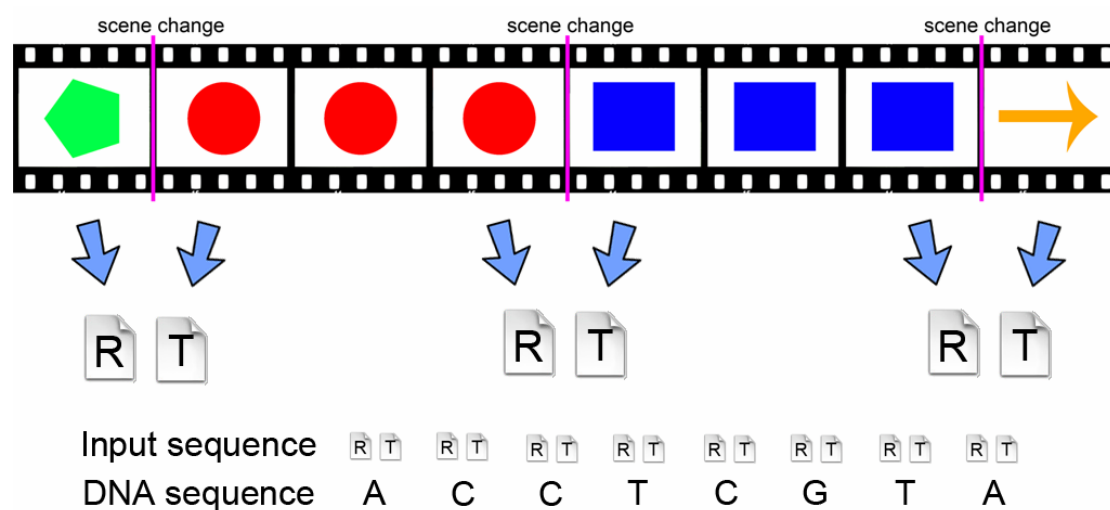


Fig. 5.16 R-T combination analog to DNA sequence in LCS algorithm

This method will maintain continuity in the input sequence, as well as reducing the problem size by half. Table below shows the result table that is expected to be generated.

i\j	R-T R-T R-T				
	0	0	0	0	0
R-T	0	1	1	1	1
R-T	0	1	2	2	2
R-T	0	1	2	2	3
	0	1	2	3	

Table 5.5 Result table using R-T combination method

It seems that we actually reduce the problem size into  $m/2 \times n/2 = 1/4(m \times n)$ . Actually, we doubled the computation in each of the cell. This is because each R-T symbol contains 2 HI3 files. When we are comparing their "equality", we are doing double amount of the job.

Did you spot any possible error of this method? The error arises at the matching stage.

Consider the following case in Fig. 5.17. Advertisement A (red circle) is followed by advertisement X (blue square) in the first video stream. In the second video stream, advertisement A (red circle) is followed by advertisement Y (purple triangle). Obviously there is scene change between A -> X and A -> Y. Thus the R-T pairs in the middle are generated. However, soon we will find that only R matches with R but T does not match with T.

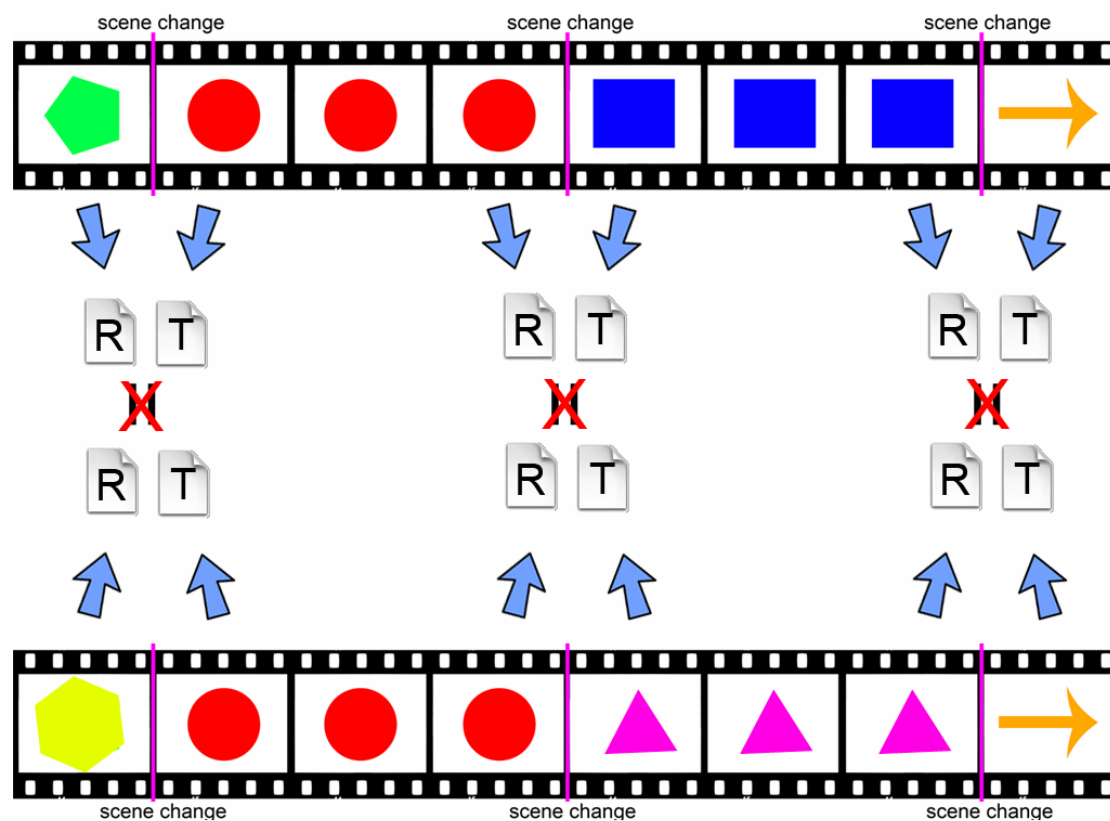


Fig. 5.17 Advertisement A (red circle) cannot be recognized because of the scene before and after are different





This method solves the problem induced by R-T combination method. Consider the following case below in Fig. 5.19. Being the same as before, advertisement A (red circle) is followed by advertisement X (blue square) in the first video stream. In the second video stream, advertisement A (red circle) is followed by advertisement Y (purple triangle). Obviously there is scene change between A -> X and A -> Y. Thus the R-T pairs are generated and grouped in T-R manner. Now advertisement A can be matched successfully.

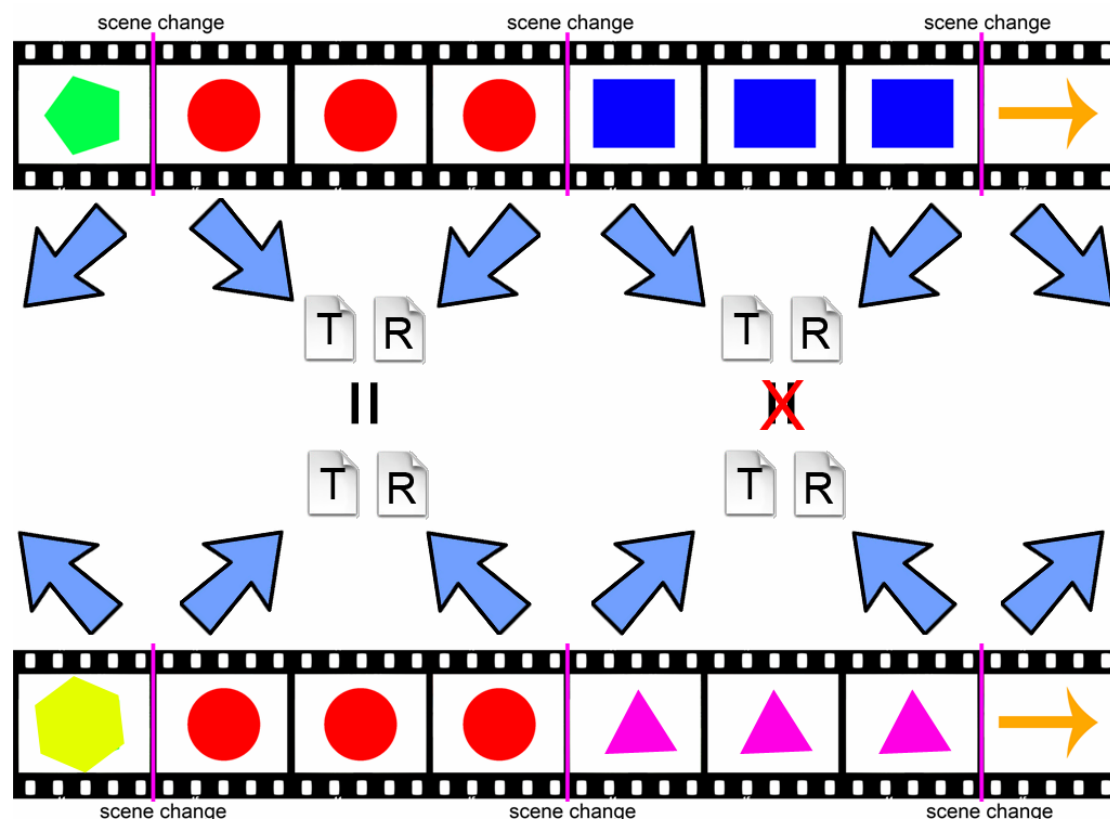


Fig.5.19 Advertisement A (red circle) can now be recognized

### 5.3.3 Logic System for “Equality” Comparison

Now after dramatically changing the LCS algorithm as mentioned in part 5.3.2, we are back to the problem mentioned in part 5.3.1. That is, with the new T-R pairs as a basic symbol in the data stream, how are we doing the equality comparison between them?

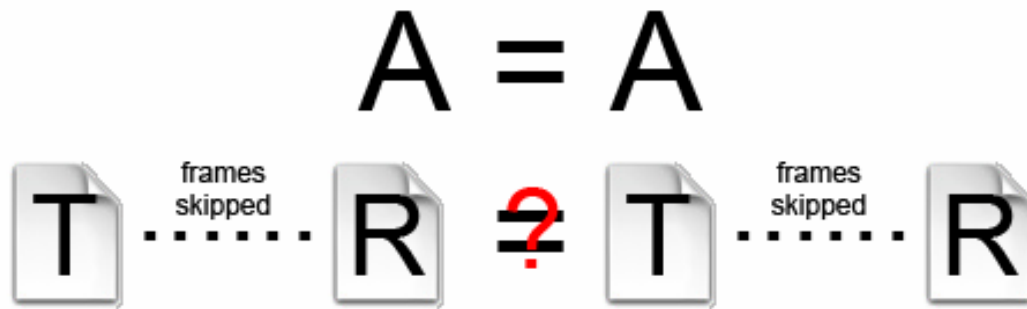


Fig. 5.20 How do we compare the equality of 2 T-R pairs?

Fortunately, the method we mentioned in 5.3.1 is still valid. We can still compare a T file with another T file, an R file with another R file. In addition, notice that the skipped frame number between the T-R pairs varies. That's why we can make use of this piece of information during the comparison too.

In our implementation, after several trials we use a procedure like the following to deduce the equality between 2 symbols:

We have 2 thresholds HL3\_THRESHOLD (30,000) and FRAME\_THRESHOLD (5).

From our experience, the value of 30,000 and 5 gives the best result. The Fig. 5.21 below shows the logic flow to determine whether 2 T-R pairs are matched.

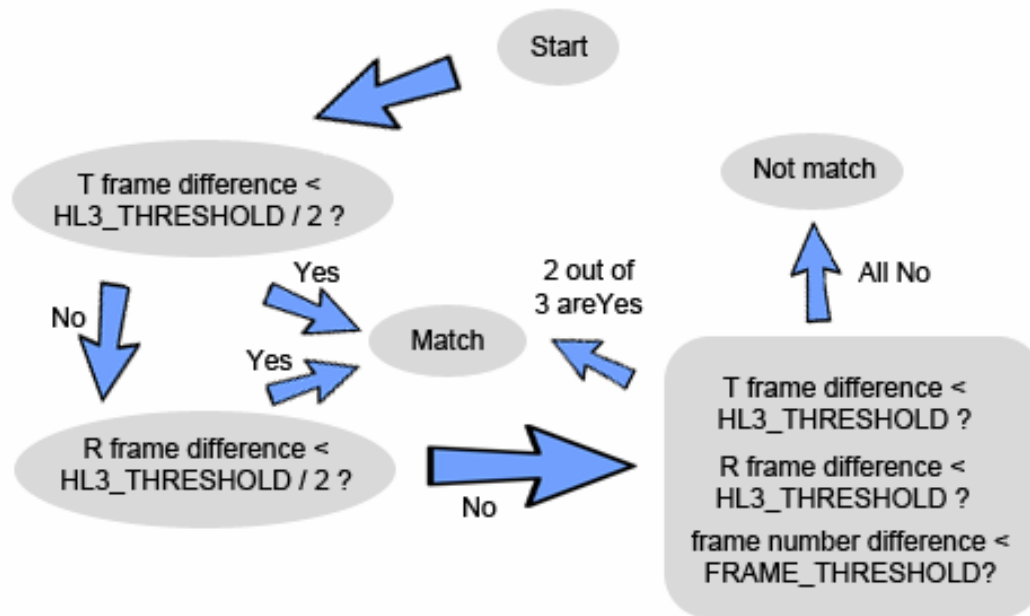


Fig. 5.21 The logic diagram to determine whether 2 T-R pairs are matched

Pseudo-code is like this:

1. Given 2 T-R pairs
2. Compute the square of Euclidean difference between T files
3. If difference between T files < HL3\_THRESHOLD / 2
4.     Return as match
5. Compute the square of Euclidean difference between R files
6. If difference between R files < HL3\_THRESHOLD / 2
7.     Return as match
8. Compute the frame number difference in the 2 T-R pairs
9. Initialize Score = 0

10. If difference between T files < HL3\_THRESHOLD
11.     Score = Score + 1
12. If difference between R files < HL3\_THRESHOLD
13.     Score = Score + 1
14. If difference between frame number < FRAME\_THRESHOLD
15.     Score = Score + 1
16. If Score >= 2
17.     Return as match
18. Else
19.     Return as not match

Meaning of this code: If one of the T frame or R frame is really alike (with difference < HL3\_THRESHOLD / 2), then they are consider to be matched. If not, further testing is carried out. If 2 out of 3 of the parameters (frame number difference, T frame difference, R frame difference) are within threshold, then it is consider a match too, else it is consider not match.

Comment on this logic system: Actually with 3 parameter and 2 thresholds, there are just too many way to create a logic system or a scoring system to determine whether 2 symbols are equal or not. It is difficult to say which procedure or system is the best. It is really up to the designer to create their own procedure. In our implementation, we try to keep it as simple as possible which could still obtain a satisfactory result.

### 5.3.4 Splitting Advertisements from LCS Result & Analysis Strings

So finally we can compute the LCS result for 2 video streams. It may look like Fig. 5.22 below.



Fig.5.22 LCS result after comparing 2 video data streams

Notices that the LCS result gives no information on how many commercial advertisements are detected, where the commercial advertisements start and end. Basically, it just uses a T-R pairs as a unit to represents the longest common video segment sequence detected among 2 video data stream. Therefore extra works have to be carried out to split apart different commercial advertisements in the LCS result.

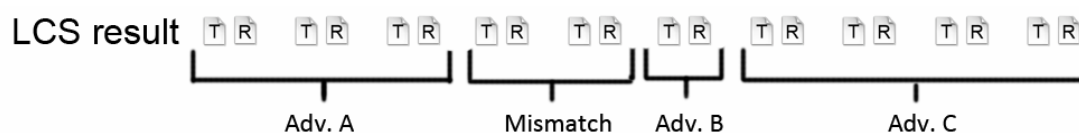


Fig. 5.23 1 possible way of the LCS result representing

Above shows an example on 1 of the possible way that a LCS result may be representing. It is important to recognize that 1 T-R pairs in the LCS is sometimes able to represent an advertisement already. For example those still picture advertisement which last for about 10 seconds. Below shows one example.

Start of adv. 8			Start of adv. 9		
2008.02.12.01.24.06.0421.T.1.hl3	187609		2008.02.13.01.23.30.0046.T.1.hl3	-92344	
					
2008.02.12.01.24.16.0312.R.213.hl3	9891		2008.02.13.01.23.40.0000.R.217.hl3	9954	
End of adv. 8			End of adv. 8		
Start of adv. 9			Start of adv. 9		
					
2008.02.12.01.23.16.0406.T.1.hl3	-59906		2008.02.13.01.23.40.0062.T.1.hl3	62	
					
2008.02.12.01.23.26.0343.R.214.hl3	9937		2008.02.13.01.23.49.0953.R.215.hl3	9891	
End of adv. 9			End of adv. 9		

Fig. 5.24 Advertisements that can be represented by 1 T-R pair

So what information we can make use of to determine the start point and the end point of a commercial advertisement? The only useful information here is the time information from the files name.

In our implementation, we take 1 assumption. That is we assume the minimum length of a commercial advertisement recognized is at least 5 seconds long. Therefore we have a constant MIN\_ADV\_LENGTH in our program code.

Then we apply an  $O(n)$  algorithm to split apart the advertisement.

1. Given a LCS result *result[]* with length *l*
2. Set *start\_time* = time of T frame of *result[0]*
3. For *i* = 1 to *l* do
4.     If *i* == *l* OR time elapsed between R frame of *result[i-1]* and T frame of *result[i]* > MIN\_ADV\_LENGTH Then
5.     If time elapsed between *start\_time* and R frame of *result[i-1]* > MIN\_ADV\_LENGTH Then

6.                      From *start\_time* to R frame of *result[i-1]* can be marked as "an advertisement"
7.                      Else
8.                      From *start\_time* to R frame of *result[i-1]* can be marked as "not an advertisement"
9.                      End if
10.                     If  $i < I$  Then
11.                     Set *start\_time* = time of T frame of *result[i]*
12.                     End if
13.                     End if
14.                     End for

In our implementation, we save down the states of the analysis result in form of a character string. We predefine 5 type of characters used in our analysis string.

1. START\_FLAG    'S' indicating it is a start point of an advertisement
2. END\_FLAG    'E' indicating it is an end point of an advertisement
3. MIDDLE\_FLAG    '>' indicating it is part of the advertisement
4. SINGLE\_FLAG    'A' indicating it is an advertisement itself (Start + End)
5. ISOLATE\_FLAG    'X' indicating it does not belong to any advertisement



These flags are defined just for debugging visually. Using the example above in Fig. 5.23 (second pic in this part), we want the algorithm to generate a character string like the following.

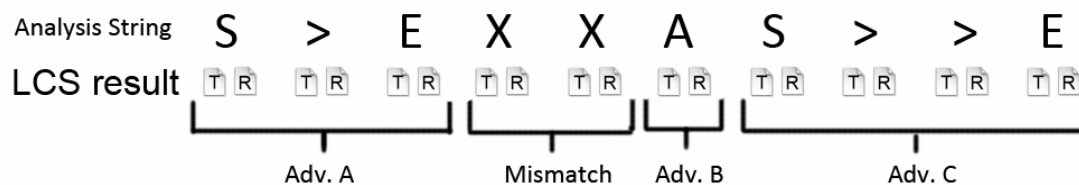


Fig. 5.25 Analysis String generated according

With the analysis string, can we print out the recognized commercial advertisements in order yet? Not really, remember Fig. 5.22? There are actually 2 sets of LCS result, 1 corresponding to video data stream 1, and another corresponding to video data stream 2. We have to apply the analysis algorithm twice on 2 video data streams. It is always possible that the 2 analysis string to be not equal due to many factor such as noise in video stream, scene change detection algorithm etc. That's why we will jump to next procedure, synchronizing the analysis string.

### 5.3.5 Synchronizing Flags

Below shows one possible case that after the analysis algorithm in part 5.3.4 is apply once to the 2 video data stream. It is clearly shown that it is possible for same LCS result gives different analysis result on 2 video data stream.

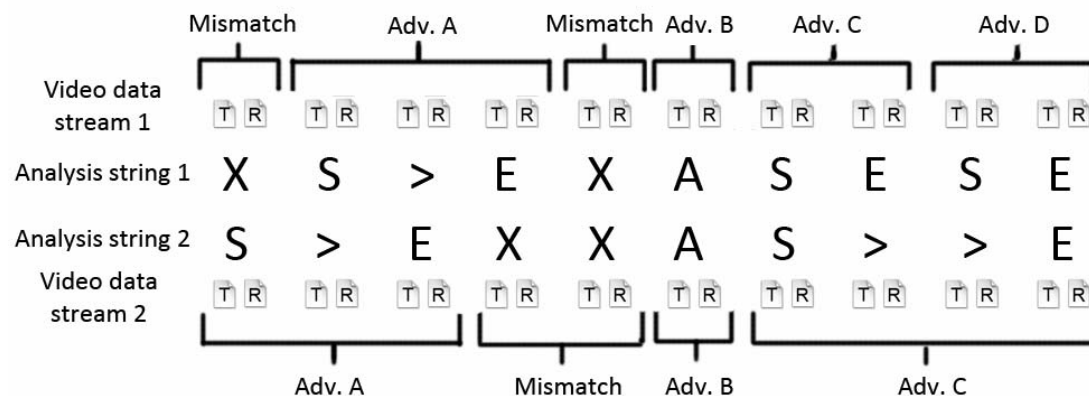


Fig. 5.26 Same LCS result gives different analysis result on 2 video data stream

There are many way to synchronize the flags in analysis string. Just like the logic procedure to determine whether 2 symbols is equal mentioned in part 5.3.3. It is really up to the designer to come up with their own design. It is difficult to say which one is best or one is better than another.

In our implementation, we use an  $O(n)$  algorithm similar to applying logical AND operation to the 2 analysis string. It is not totally the same but idea is similar. That is we try to recognize a particular position to be an advertisement if and only if both analysis string agrees that it is an advertisement. Using the example in Fig.5.26 above, below shows the result after synchronization.



### 5.3.5 Flags Propagation and Printing Result

After synchronization of the analysis flags, it is finally the time to print the result. But before that, we do one more step. We try to propagate the analysis result from the LCS back to the file list. What does that means? Please look at the Fig.5.30 below.

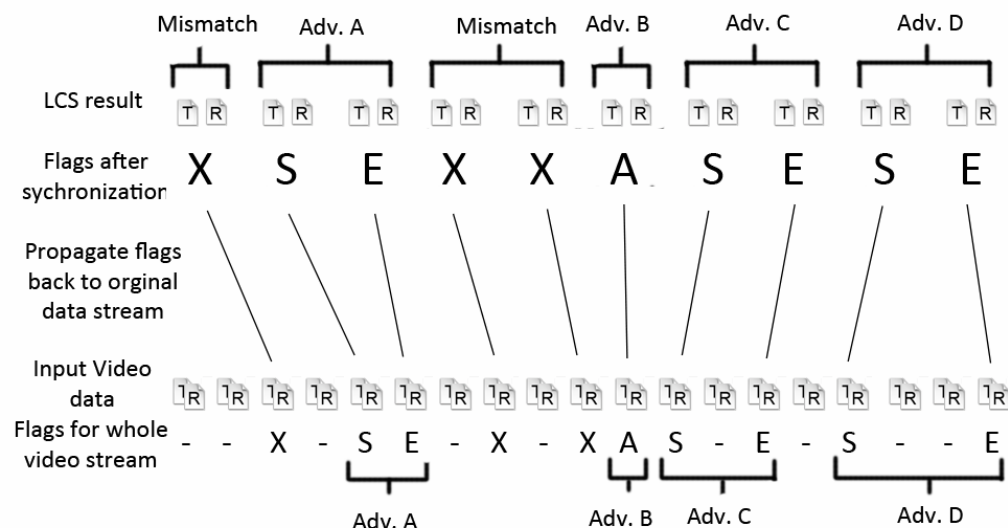


Fig. 5.30 Flag propagation from LCS result back to original input files stream

The advantage of doing this is that we can print out the unmatched T-R pairs which should also be part of the advertisement into the final result. For example, take a look at advertisement C and advertisement D in Fig. 5.30 above. In the synchronized LCS result, both advertisement C and advertisement D are recognized to be composed of 2 T-R pairs. However, after propagating the flags back to the original in file stream, we can spot that there is an unmatched T-R pair between start point and end point of advertisement C. On the other hand, we are missing 2 T-R pairs between starting point and end point of advertisement D. These are extra information we can obtain by doing flags propagation back to original input data stream.



### 5.3.6 Single Pass LCS to Multiple Passes LCS

There is one problem from LCS algorithm. Consider the case shows below.

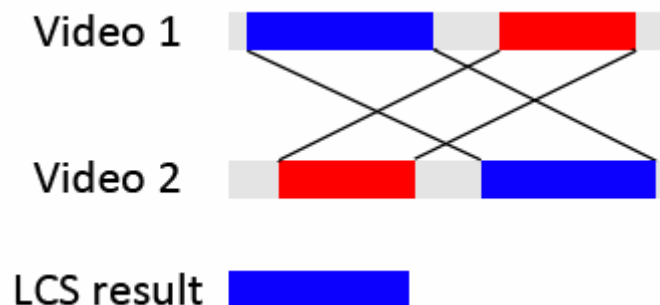


Fig. 5.33 LCS algorithm fail to recognize both advertisement

The diagram above shows that there are 2 common advertisements in both video streams. In video 1, advertisement A (blue) appears before advertisement B (red) while in video 2, advertisement B appears before advertisement A. This case is fairly common in reality. However LCS algorithm can only recognize the longer advertisement (advertisement A) because of “longest” and “subsequence” property of this algorithm.

Since LCS gave us so many nice properties as mentioned before, we want to keep using it. Therefore we use another approach to fix this problem. That is we don't do single pass LCS algorithm as we will be missing some advertisement. We do multiple passes LCS.

The idea of multiple passes LCS algorithm is as following. After we obtain result from the first pass of LCS, we print out the result to the output file. Then we delete the recognized advertisement segments from the original input video data stream. Afterwards, we apply LCS algorithm again to discover more repeated advertisements in both video stream. This procedure is repeated until no more new advertisements are recognized. Fig. 5.34 shows an example of using multiple passes LCS to recognized all advertisements.

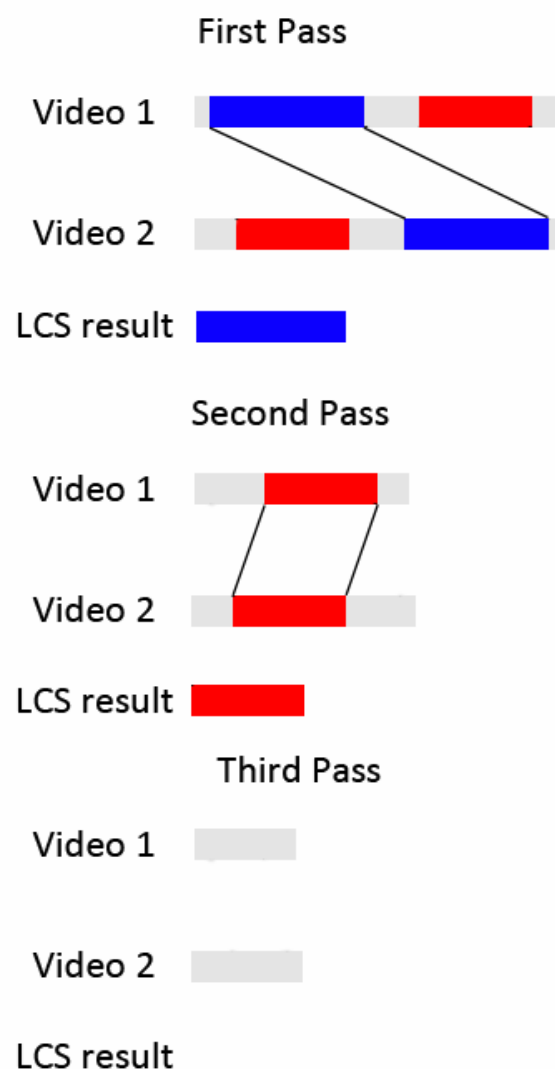


Fig. 5.34 An example of using multiple passes LCS to recognized all advertisements

The number of passes required really depends on complexity of the video streams. Usually, 2 video streams with 8000 frames each take about 7 to 8 passes to complete the recognized.

Reader may worry about the increase in computation time because of the increased 7 to 8 passes. Actually, we have a speed up approach base on this problem as stated in later part. After applying that method, the computation is reduce from  $O(1024 \times m \times n)$  for all passes to  $O(1024 \times m \times n)$  for first pass,  $O(m \times n)$  for remaining passes.



### 5.3.7 Convert to EAG

After the data has gone through multiple pass and computation is completed, a list of matching HL3 pairs is obtained. In addition, they are already grouped in different segments.

A Excel program, which is written in VBA, is implemented to retrieve information from the result and display them as a EAG. It will display the matching video segment separately, and show the start time and end time of the segment. A screen capture of the result display looks like:

	A	B	C	D	E	F	G	H
6	HL3 Match Threshold:	100000						
7	Number of Matched Frames:	294						
8	Number of Matched Ads:	9						
9	File Names of Folder 1	Interval	Image		File Names of Folder 2	Interval	Image	
10	Start of adv. 1				Start of adv. 1			
11	2008.03.07.00.10010054.T.1.hB	601054			2008.03.12.00.10004138.T.1.hB	604138		
12	2008.03.07.00.10010527.R.126.hB	473			2008.03.12.00.10004611.R.126.hB	473		
13	End of adv. 1				End of adv. 1			
14	Start of adv. 2				Start of adv. 2			
15	2008.03.07.00.10011153.T.1.hB	626			2008.03.12.00.10004612.T.1.hB	1		
16	2008.03.07.00.10011527.R.154.hB	374			2008.03.12.00.10004985.R.153.hB	373		
17	End of adv. 2				End of adv. 2			
18	Start of adv. 3				Start of adv. 3			
19	2008.03.07.00.10011571.T.1.hB	44			2008.03.12.00.10007530.T.1.hB	2545		
20	2008.03.07.00.10012027.R.95.hB	456			2008.03.12.00.10007986.R.95.hB	456		
21	End of adv. 3				End of adv. 3			
22	Start of adv. 4				Start of adv. 4			
23	2008.03.07.00.10012554.T.1.hB	527			2008.03.12.00.10008452.T.1.hB	466		
24	2008.03.07.00.10012555.R.2.hB	1			2008.03.12.00.10008610.R.159.hB	158		
25	End of adv. 4				End of adv. 4			

Fig 5.35 Screen Capture of the EAG result display

## 5.4 Speeding Up in PlayStation®3 Platform

The main focus of our project during last semester was how to use PlayStation®3 to speed up the computation. The reason why PlayStation®3 can outperform other platform like PC is because of its multiple cores SPU. These cores are specialized for computation intensive problem. Despite the lack of random access memory and computation power of PPU, PlayStation®3 can perform better than PC by a lot.

Since we changed the algorithm to solve the problem from minimum difference algorithm to multiple LCS algorithms, we have to redesign the program for this problem.

In this chapter, we will introduce our procedure in parallelizing the LCS algorithm as well as those speed up approach that we have applied in our program.

### **A. Parallelization**

First let's recall the LCS algorithm. LCS algorithm composes of 2 parts, computing result table and backtracking. Computing result table is  $O(m \times n)$ , where  $m$  and  $n$  are the length of input strings. Backtracking is  $O(m + n)$  which is negligible comparing with  $O(m \times n)$ . Therefore our focus will be put on how we speed up the computation of result table.

At first, we try to search for existing approach in doing parallelization of LCS algorithm. We have found one parallel algorithm called FAST\_LCS <sup>[4]</sup>. This method was used to speed up the procedure of finding LCS in bioinformatics field. However, part of the memory requirement for this algorithm is  $O(\text{alphabet set size} \times \text{length of input sequence})$ . This is not applicable to our case for the reason that we are not using A, C, G & T for our alphabet set. We are using pairs of T-R HI3 files as our alphabet set. Our alphabet set size will be very large if we are comparing 2 long video streams. With the limited random access memory available in PlayStation®3, it is simply impossible.

We look for a few more parallel methods but none seems applicable. Therefore we decide to come up with our own design as LCS algorithm isn't as difficult as we thought.

Recall the algorithm to compute the result table, the most important point is that when filling the cell  $(i, j)$ , information from  $(i-1, j-1)$ ,  $(i, j-1)$  and  $(i-1, j)$ . This is the one and only one restriction on the problem. Visually, it is like Fig.5.36 below.


Fig. 5.36 Computing the orange cell requires accessing to the red cell

We can view it the problem in the other way. Please look at Fig. 5.37 below.

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9

Fig. 5.37 Computable cell at each step

The figure shows cells that are computable at different steps. For example, at start, only the upper left cell labeled '1' can be computed. Then after '1' is computed, the 2 cells labeled '2' is computable. After '2' is computed, the 3 cells labeled '3' is now computable and so on. Consider we have 3 cores in our machine available, we can do parallel computation for the result table in the following ways.

Core 1	1	2	3	4	5
Core 2	2	3	4	5	6
Core 3	3	4	5	6	7
Core 1	6	7	8	9	10
Core 2	7	8	9	10	11

Fig. 5.38 Parallel strategy for 3 cores on 5 x 5 result table

Fig. 5.38 above shows a parallel strategy. The 3 cores become fully utilize starting from step 3 and up to step 7. Consider in PlayStation®3 where 6 cores are available. Also we are computing a table of size that is very large, like 4000 x 4000. So the only time 6 cores are not fully utilized will be the first 5 steps and last 5 steps, which are really negligible. We can do some simple mathematics:

During the first 5 and last 5 steps, we computed

$$(1+2+3+4+5) * 2 = 30 \text{ cells}$$

So we are fully utilizing all the cores

$$(1 - 30/4000*4000) * 100\% = 99.999\% \text{ of the time}$$

There is one limitation in PlayStation®3 though, that is the local storage of each cores are limited. Therefore when we use the cores to compute the values in the table, we have to use the direct memory access (DMA) mechanism to fetch data from main memory or put computed result back to main memory. The speed of DMA is so fast that it's like instant access. The limitation comes from the direct memory access size. It has to be at least 16 bytes or multiple of 16 bytes for DMA to work efficiently. For the data type of the result table, we are using integer type, which is just 4 bytes. As a result, we have to DMA at least 4 cells in the result table. Therefore, a newer approach will be like Fig. 5.39.

Core 1	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
Core 2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5
Core 3	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6
Core 1	5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8
Core 2	6	6	6	6	7	7	7	7	8	8	8	8	9	9	9	9

Fig. 5.39 3 cores computing a result table of size 5 x 16

This approach let each cores to compute 4 cells (16 bytes) at a time, such that the computed result can be put back to main memory immediately. To compute 4 cells, we have to access to 6 cells shown in the following:

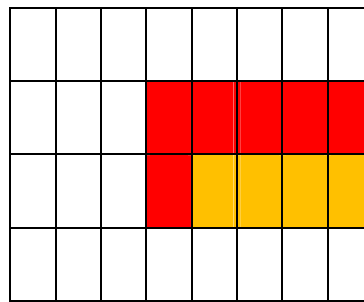


Fig. 5.40 Computing the orange cells require access to the red cells

The easiest approach will be to save the result of red cells on the same row, which is computed by the same core previously. Then DMA into local storage for the data in upper row.

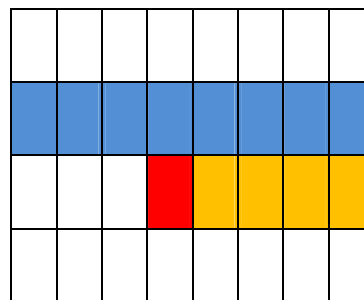


Fig. 5.41 DMA the upper 8 blue cells for computing the orange cell

In conclusion, each computation requires DMA into local storage for 32 bytes, and DMA out for 16 bytes. Reader does not have to worry about the time consumed by DMA for the following reasons:

1. DMA is operated by a separated device other than the core itself, therefore it is possible to do computation and DMA at the same time.
2. DMA is much faster than the computation steps, thus if they are done in parallel manner, DMA won't slow down the computation.
3. Double buffering technique we are going to mention in next part allows us to pre-fetch data needed for next computation. Therefore nearly no time is consumed by DMA data.

## **B. Double buffering**

Since there is a Memory Flow Controller (MFC) in each SPE for DMA, therefore DMA commands can be execute simultaneously while SPE is doing computation. Double buffering is a technique that makes use of this property.

Originally, there is only one buffer for storing the data to be fetched in. Therefore the flow of program is like:

1. Fetch in required data via DMA.
2. Do the computation for the result table of LCS algorithm
3. Put data back to main memory

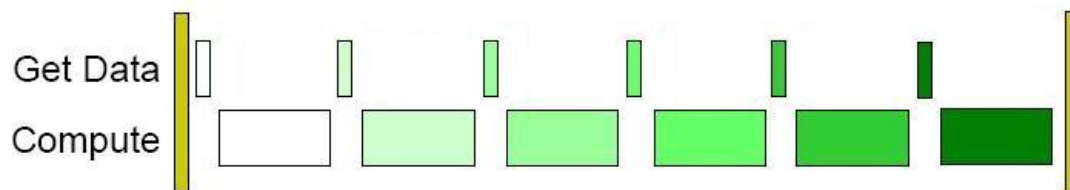


Fig. 5.42 The total time required for computation before applying double buffering

By allocating one more buffer in SPE's local store, it allows data to be fetched into local storage first and get ready in the buffer.



The new flow of the program becomes:

1. Fetch in required data to buffer 1 via DMA.
2. Do the following simultaneously
  3. Do the computation for the result table using data from buffer 1.
  - 1.
4. Fetch in next required data to buffer 2 via DMA.
5. Put the result back to main memory.
6. Loop back to 2 with usage of buffer 1 swapped with buffer 2.

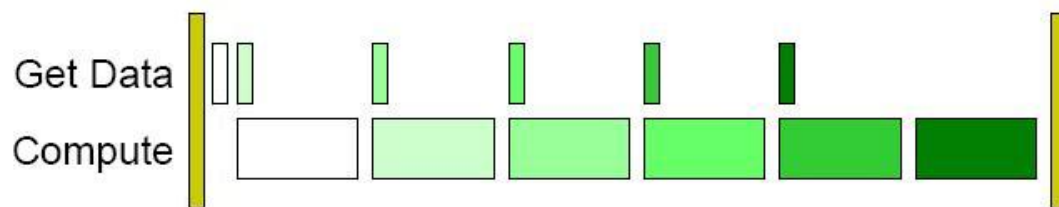


Fig. 5.43 Total computation time reduced after applying double buffering

As can be observe from the figure, if communication is smaller than computation, there won't be much speed up. This is the case in our program unfortunately.

### C. SIMD function

SIMD is short for single instruction multiple data. It means applying a single operation on multiple operands at the same time. CellSDK provides SIMD intrinsic instructions for SPE and PPE. These SIMD instructions take a 128-bit register as input. Then apply an operation on every element of the vector at the same time.

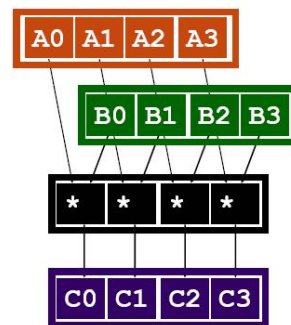


Fig. 5.44 SIMD operation on 4 elements at a time

SIMD function provided to SPE is mostly working on float or double data type. That is the reason why we change the data type of HI3 file from int to float type. Float type being 32 bits in size, we can pack 4 float values into a 128-bit register. If we operate all the float values at once, we are speeding up our program by 4 times.

#### **D. Loop unrolling**

From the experience we have during last semester, loop unrolling gives a great skill up if most the time the core is working within a loop. Loop unrolling simply removes the addition computation for the loop counter. Computation time for loop counter can have great contribution if the number of statements within the loop is small. This is true in our case. That's why we can apply loop unrolling method to speed up the computation.

The For-loop to compute square of Euclidean distance is fully unrolled by repeating the statements inside for 256 times. It is done by writing another program to generate the code.

```
int main(){
    int i;
    for (i=0;i<HL3_SIZE/INT_VEC_ELE;i++)
    {
        printf("\ttemp = spu_sub(hl3_a[%d].vec, hl3_b[%d].vec);\n", i, i);
        printf("\tdiff.vec = spu_madd(temp, temp, diff.vec);\n");
    }
    return 0;
}
```

Fig. 5.45 Simple program written to generate codes for loop unrolling

Then the code generated by the above program are copied and pasted into the source code. Now we have completely get rid of the loop counter i. The speed up becomes more obvious. For example, a multiple pass LCS algorithm running in 2 minutes can be speeded up to 1 minute 37 seconds.

### **E. Caching equality comparison result**

Multiple passes LCS algorithm take  $2048 \times m/2 \times n/2 \times \text{No. of passes}$   
 $= 512 \times m \times n \times \text{No. of passes}$ . The number of passes depends on the relative complexity between the 2 input video. For example, comparing between 2 video streams which are both 1 hour long, 8000 x 8000 frames, usually it takes about 7 to 8 passes. This cause the computation time increase by 7 to 8 times comparing go single pass LCS algorithm.

Actually, many redundant computations are done in multiple passes LCS algorithm. Consider in the first pass, we are computing a result table of  $m \times n$ . That's mean we have done all the equality comparison between any pairs of T-R pairs already. In the second pass and onwards, we have to do the equality comparison again to compute the final result table.

We can simply cache our equality comparison result between any T-R pairs in first pass using memory of  $O(1/4m \times n)$ . Then from second passes and onward, we use the equality comparison result. Therefore the total computation time is reduced from  $O(512 \times m \times n)$  for all passes to  $O(512 \times m \times n)$  for 1<sup>st</sup> pass and  $O(1/4 m \times n)$  for remaining passes. Meaning there is a speed up of 2048 times for 2<sup>nd</sup> pass and onward comparing with 1<sup>st</sup> pass.

## Chapter 6 Result Analysis

---

We have carried out experiments to verify whether our objectives have been achieved, i.e. an accurate and fast algorithm. The details of the experiment result and the follow-up analysis are presented in the following parts.

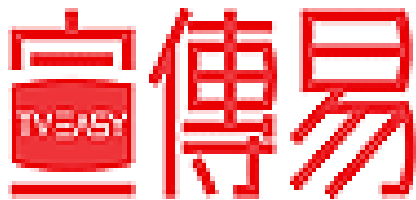
- ◆ Experiment of Cross-Comparison of 7 Videos
- ◆ Experiment of Comparing 2 One-Hour-Long Videos
- ◆ Performance Comparison on PC and PlayStation® 3
- ◆ Cost Comparison on PC and PlayStaion® 3

## 6.1 Experiment of Cross-Comparison of 7

### Videos

#### *Introduction of the Experiment*

In this experiment, we use data of the TV program “TV Easy(宣傳易)” from 7 different days as input to our LCS algorithm.



“TV Easy” is a classified advertising TV program. It lasts about 5 minutes.

Within this period, only commercials are broadcasted. Weather reports, TV programs promos etc. are excluded. Therefore, in each video segment there are less noise, only commercials are available.

#### *Objectives*

The aim of this experiment is to show that we can find all the advertisements of a single video by cross-comparing it to the others, with the assumption that the advertisements have appeared twice in the 7 days. Since 2 videos are compared each time, this requires us to run the LCS algorithm in  $C_2^7 = 21$  times in order to finish the cross-comparison for each video.

***Data Information***

Date	Total No. of HL3 Files	Total No. of Ad.	No. of Ad. appear only once in the 7 days	No. of Ad. appear at least twice in the 7 days
0304	558	14	0	14
0305	642	15	2	13
0306	522	13	1	12
0307	576	15	2	13
0310	718	20	3	17
0312	558	15	1	14
0313	574	14	3	11

For the advertisements that appeared once only, they are not being handled by our algorithm yet. Our algorithm finds video segments that have appeared at least twice in the data. Thus we ignore them at this point.

## Experiment Result

### Individual Result – March 4<sup>th</sup>

In this part we choose the result of March 4<sup>th</sup> TV Easy as an example to demonstrate the experiment and have a detailed result analysis.

1) March 4<sup>th</sup> is first compared to March 5<sup>th</sup>. After manual check, we find 7 advertisements appear in both TV Easy. The LCS algorithm also finds 7 matching commercials.

March 4<sup>th</sup> TV Easy



March 5<sup>th</sup> TV Easy



Fig 6.1 Comparison between 3/4 and 3/5 TV Easy



2) March 4<sup>th</sup> is then compared to March 6<sup>th</sup>. After manual check, 8 advertisements are found appearing in both TV Easy. The LCS finds 7 matching commercials and misses 1 (i.e. 1 false negative).



Fig 6.2 Comparison between 3/4 and 3/6 TV Easy

3) The procedure continues in a similar way for the remaining days.

Detail Result:

(百源堂) = Ad. appears only once in all 7 days, i.e. a unique Ad.

Man = Compared Manually

LCS = Compared by our LCS algorithm

Y = A Matched Pair is Found

	030400 TVEASY	0305		0306		0307		0310		0312		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	儲存易迷你倉							Y						Y	
2	Crisis Credit			Y	Y							Y	Y	Y	Y
3	e-print					Y	Y			Y	Y			Y	Y
4	Physical A	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
5	環保袋							Y	Y					Y	Y
6	sling	Y	Y	Y	Y			Y	Y	Y	Y			Y	Y
7	捷達移民			Y	Y			Y	Y					Y	Y
8	apple 迷你倉							Y	Y					Y	Y
9	uwants	Y	Y									Y	Y	Y	Y
10	得成			Y	Y							Y	Y	Y	Y
11	unlimit hosting	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
12	Physical B	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
13	Banner Shop	Y	Y			Y	Y	Y	Y			Y	Y	Y	Y
14	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	7	7	8	7	6	6	8	7	6	6	8	8	14	13
	False Negative		0		1		0		1		0		0		1
	False Positive		0		0		0		0		0		0		0
	Matching Rate (%)		100		87.5		100		87.5		100		100		92.9

Above is a table showing the commercial matching result of March 4<sup>th</sup> to 6 other days, with manual ground truth and LCS result.

The Result of March 4<sup>th</sup> after Cross-Comparison:

Date	No. of Ad. appeared at least twice in the 7 days	No. of Ad. Found after Cross-Comparison	False Negative	False Positive	Matching Rate (%)
0304	14	13	1	0	92.90%

For the 14 advertisements that have appeared twice in the 7 days, our LCS algorithm find 13 of them and miss 1, giving a matching rate of 92.9%.

Individual Result – March 12<sup>th</sup>

Detail Result:

(百源堂) = Ad. appears only once in all 7 days, i.e. a unique Ad.

Man = Compared Manually

LCS = Compared by our LCS algorithm

Y = A Matched Pair is Found

	031200 TVEASY	0304		0305		0306		0307		0310		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	Angel			Y	Y			Y		Y	Y			Y	Y
2	(星輝)														
3	Physical A	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
4	Umart							Y	Y	Y	Y			Y	Y
5	unlimit hosting	Y	Y	Y	Y	Y	Y	Y		Y		Y		Y	Y
6	葉謝鄧律師行					Y		Y		Y	Y	Y		Y	Y
7	e-print	Y	Y					Y	Y					Y	Y
8	時昌迷你倉			Y				Y	Y	Y	Y			Y	Y
9	Physical B	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
10	FIA							Y	Y					Y	Y
11	slings	Y	Y	Y	Y	Y	Y			Y	Y			Y	Y
12	海港錦鯉			Y	Y			Y	Y	Y	Y			Y	Y
13	Primada			Y	Y									Y	Y
14	NOD32					Y	Y			Y	Y	Y	Y	Y	Y
15	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	6	6	9	8	7	6	11	8	9	7	6	4	14	14
	False Negative		0		1		1		3		2		2		0
	False Positive		0		0		0		1		0		0		1
	Matching Rate (%)		100		88.8		85.7		72.7		77.8		66.7		100

Above is a table showing the commercial matching result of March 12<sup>th</sup> to 6 other days, with manual ground truth and LCS result.

The Result of March 12<sup>th</sup> after Cross-Comparison:

Date	No. of Ad. appeared at least twice in the 7 days	No. of Ad. Found after Cross- Comparison	False Negative	False Positive	Matching Rate (%)
0312	14	14	0	1	100%

For the 14 advertisements that have appeared twice in the 7 days, our LCS algorithm find 14 of them and misidentify 1 noise segment as advertisement, giving a matching rate of 100%.

The result is worth mentioning because it gives an example of false positive. That is, some frames are being mismatched and are regarded as an advertisement.

Start of adv. 4			Start of adv. 4		
2008.03.07.00.10012554.T.1.hl3	527		2008.03.12.00.10008452.T.1.hl3	466	
2008.03.07.00.10012555.R.2.hl3	1		2008.03.12.00.10008610.R.159.hl3	158	
End of adv. 4			End of adv. 4		

Fig 6.3 Example of a False Positive Advertisement

This kind of false positive may occur when there is some blank frames appear. These blank frames have a low HL3 difference with other frames which only have simple patterns, causing the algorithm to treat them as a match.

In addition, using the approach of cross-comparing, even the matching rate to a particular day is lower than 100%, all advertisements can still be found finally. It is because some advertisement may appear 3 or 4 times within the 7 days, missing a pair of commercials in comparison to one day can still be compensated by finding in comparison with the other day.

### Overall Result

The detail result of individual days is listed in Appendix A. The overall result of cross-comparing 7 days' TV Easy is shown here.

Date	Total No. of Ad.	No. of Ad. appear only once in the 7 days	No. of Ad. appear at least twice in the 7 days	No. of Ad. Found after Cross Comparison	False Negative	False Positive	Matching Rate (%)
0304	14	0	14	13	1	0	92.90%
0305	15	2	13	12	1	0	92.30%
0306	13	1	12	11	1	0	92%
0307	15	2	13	13	0	2	100%
0310	20	3	17	15	2	1	88%
0312	15	1	14	14	0	1	100%
0313	14	3	11	11	0	0	100%
							Average: 95%

The LCS algorithm has achieved a satisfactory matching rate of 95%. The total running time is 26 sec. To conclude, we are able to identify an advertisement by cross-comparing data from several days, if it has appeared at least twice in that period.

## 6.2 Experiment of Comparing 2 One-Hour-Long Videos

### *Introduction of the Experiment*

In this Experiment, we input two one-hour-long videos to our LCS algorithm to identify the advertisements that have appeared in both videos. Since the videos are captured in real-time, sometimes it may jitter or freeze for a second, making some noise in the data. The two videos record all program within that hour, including commercials, news reports and drama programs, so our LCS algorithm need to handle frames other than those from advertisements only.

### *Objectives*

The aim of the experiment is to:

- 1) Show the matching rate of our LCS algorithm when processing data other than just containing advertisements only.
- 2) Verify the result of inputting data in different order, i.e. compare March 4<sup>th</sup> to March 5<sup>th</sup> versus compare March 5<sup>th</sup> to March 4<sup>th</sup>.
- 3) Show the performance of our program in the case of large amount input data.

### *Data Information*

Date	Time	Duration	Total No. of HL3 Files
0304	12am – 1am	1 hour	8060
0305	12am – 1am	1 hour	7920

No. of Ad. appeared in both days =42

## ***Experiment Result***

Detail result is listed in the Appendix B. The overall result is shown as follow:

	Comparing 0304 to 0305	Comparing 0305 to 0304
No. of Ad. Found Manually	42	42
No. of Ad. Found by LCS	40	39
False Negative	2	32
False Positive	2	1
Matching Rate (%)	95%	93%
Time Elapsed	1 min 36 sec	1 min 39 sec

Comparing the data of March 4<sup>th</sup> to March 5<sup>th</sup> yields a matching rate of about 95% while comparing the data of March 5<sup>th</sup> to March 4<sup>th</sup> yields a lower matching rate of about 93%.

### False Negative

The matching rate for different input order is not the same. It is because comparing in the later order leads to a false negative of one of the advertisements, the "Physical". This commercial actually appears 4 times, 2 in each day. This commercial contains about 10 scenes. However, the scene may appear in different order, forming 2 patterns "Physical A" and "Physical B". Since the 2 patterns still sharing the same scene, it may confuse our algorithm and treat it as several shorter commercials. Like the condition in the above experiment.

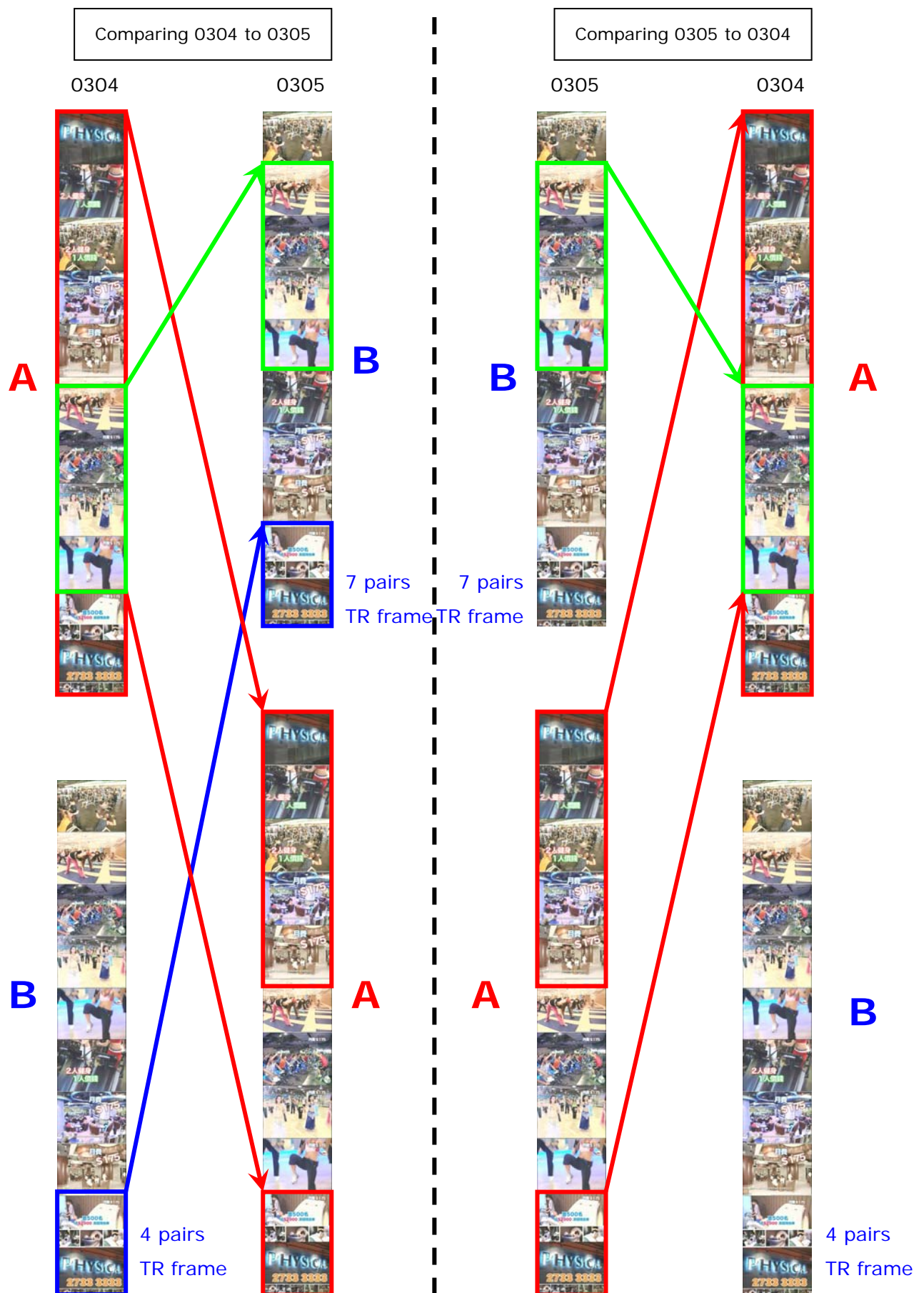


Fig. 6.4 False-negative condition cause by using same scenes in 2 Ad.

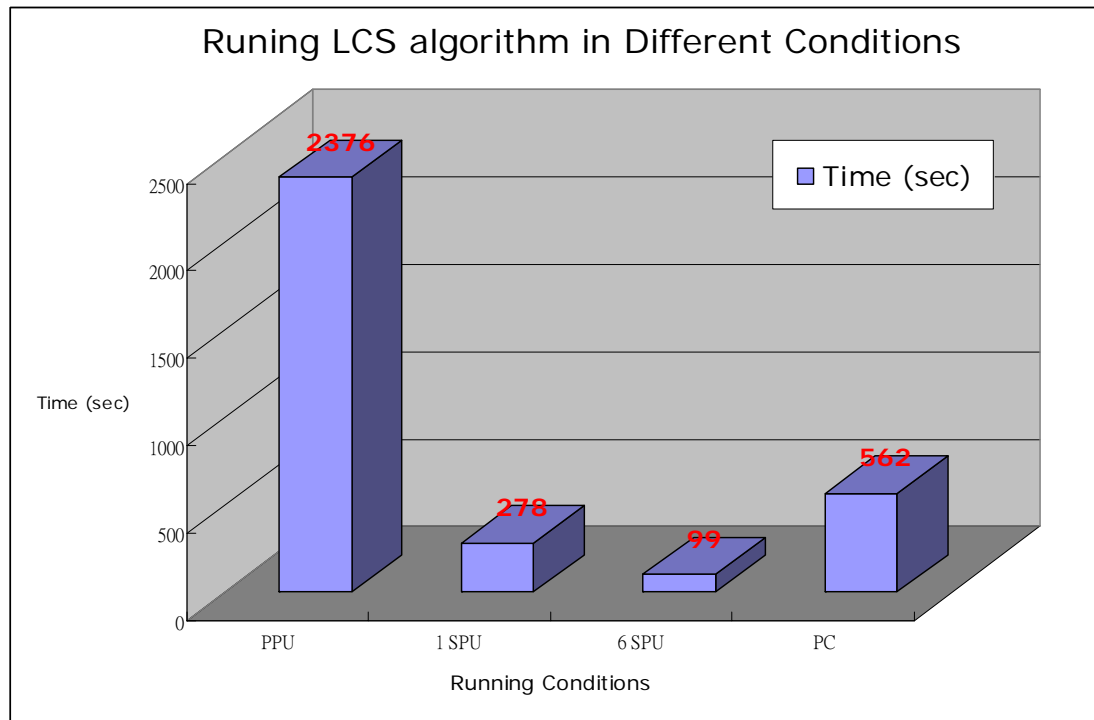


## 6.3 Performance Comparison on PC and PlayStation®3

In order to make our algorithm practical and applicable to solve the automatic commercial monitoring problem, it needs to have high performance. Otherwise it may lag behind since there are new video data recording every minute. The following gives a performance comparison of running the LCS algorithm in different conditions:

Data being processed	Date	Time	Duration	Total No. of Frames
	0304	12am – 1am	1 hour	8060
	0305	12am – 1am	1 hour	7920

Time (sec)	PlayStation®3			PC
	PPU	1 SPU	6 SPUs	
Test 1	2376	273	96	555
Test 2	2378	281	105	543
Test 2	2373	279	97	587
Average	2376	278	99	562



As a result, executing the LCS algorithm on PlayStation®3 and utilizing all 6 SPUs will be:

24 times faster than using only PPU on PlayStation®3

2.8 times faster than using 1 SPU on PlayStation®3

5.7 times faster than running on a PC

This shows that our parallel LCS algorithm works much faster than sequential one. It is also fast enough to process in semi-real-time.

## 6.4 Cost Comparison on PC and PlayStation®3

As we proposed, during real application, 1 hour of video has to be compared with 6 hour videos. In order to have a complete commercial monitoring of 1 hour video, the LCS algorithm has to be executed for 6 times. This takes:

$6 \times 99 \text{ sec} = 10 \text{ mins}$  on PlayStation®3

$6 \times 562 \text{ sec} = 56 \text{ mins}$  on PC

Therefore, using a PlayStation®3 is fast enough to monitor around 5 TV channels together in semi-real-time, without lagging behind, while using one PC can at most monitor 1 TV channel. 5 PCs are required to attain same performance with 1 PlayStation®3.

The cost for 1 PlayStation® is about HKD\$3000. The cost for 5 PCs is about HKD\$20000. The cost for hiring 3 staffs to monitor commercials manually may cost about HKD\$20000 per month. Obviously, running our LCS algorithm on PlayStation®3 would be the cheapest and fastest way to achieve automatic commercial monitoring.

## Chapter 7 Project Difficulties

---

In this part we will talk about the difficulties we have come across with. The interesting thing about them is that they are all related to the PlayStation®3.

### 1. PlayStation®3 not working again!

Last semester, our PlayStation®3 has broken down once due to plugging a USB web cam to it. This semester, we have our PlayStation®3 not loading the operation system because of the firmware update to version 2.1 disable the access to certain hardware, make the Linux crash once boot up.

There are no methods for us to downgrade the firmware back to version 2.0. Therefore, the PlayStation®3 was not available for about 2 month. It was fixed finally by using a new Linux boot loader just release at that time.

During the period when PlayStation®3 not available, we manage to keep our project going by using the PlayStation®3 owned by one of us which has not yet upgrade the firmware to version 2.1. Also we put more focus on the modification of the LCS algorithm and result comparison. Therefore, it doesn't affect us too much.

## 2. Migration to new SDK 3.0

The new CellSDK 3.0 becomes available. We installed CellSDK 3.0 and try to run our code which was written with CellSDK 2.1. Unfortunately, we found that we cannot even compile the program. There are many major changes in API and library in CellSDK 3.0. For example, the procedure to create a SPU thread is different and becoming more complicate. More codes are needed to perform the same job.

Therefore we have to spend some effort to modify our code in order to migrate from CellSDK 2.1 to CellSDK 3.0.

## 3. sprintf() function is slow!

At one moment, we have done all the parallelization implementation and optimization for the video comparison algorithm. However, we found that the paralleled program run even slower than the PC version program which uses only single core. We were so confused that maybe our parallel algorithm is not good enough. After a few weeks of keep trying, we finally realized that the sprintf() function implemented for the SPU is extremely slow. This function doesn't affect the run time of PC version by many though. After getting rid of it from the codes, our paralleled program on PlayStation®3 can finally outperform serial program on PC.




## Chapter 8 Future Development

---

Our final year project is completed, while there is still some improvement for this project. We are going to describe some unsolved problems in our project and some possible modification.

### ***Removing TV Promo***

Consider the experiment of comparing 0304 to 0305, there are some matching we found that are not commercials.

badge of the TV channel	
opening theme of a TV drama	
intermediate theme of a TV drama	

Since our algorithm can only identify similar video segments that have appeared twice, to determine whether the segment is a commercial would need further computation.

We may need to look into the content of the frames to eliminate these errors. For example, we may carry out logo recognition to remove those non-commercial TV programs. Since we only use 10 minutes to identify advertisement in 1 hour of TV program, time is still available for further computation without lagging behind the real-time TV program too much.

***Improve Accuracy by Removing Noise***

As shown in the result, our accuracy is not 100% yet. The minimum HL3 difference contributes to part of the error. As the videos are being captured from analogue signal, it would jitter and have noise. Thus even the HL3 files of the same scene from 2 videos may have a large minimum difference.

In 2008, the Digital TV Broadcast has launched in Hong Kong, providing digital signal for TV programs with no jittering and noise. This would give better data as there is no noise. Same scene would probably have a zero minimum difference, reducing the error rate.

***Use Knowledge from Electronic Program Guide***

From the Electronic Program Guide, we can find the approximate start time and end time for a TV programs. Using its knowledge may help to exclude TV dramas, news reports etc. from the data.

***Identifying New Commercials***

We are now able to identify an advertisement if it appears twice in the videos under comparison. Further analysis can be done to locate new commercials in between two known one.

## Chapter 9 Conclusion

---

To conclude, we have achieved the followings in our final year project:

- 1) Using the LCS approach, which is different from the one in last semester (minimum difference), to solve the video comparison problem.
- 2) The LCS is further modified for 2 major reasons:
  1. Attain a more accurate matching rate
  2. Utilize the parallel processing power of PlayStation®3
- 3) Finally, for comparing 1 hour video to 1 hour video, our parallel LCS algorithm takes 99 seconds to finish, and having about 95% matching rate
- 4) An Electronic Advertisement Guide can be provided from the result.
- 5) The algorithm we designed and implemented introduces a more accurate, faster and cheaper way to monitor commercials.



## Chapter 10 Contribution of Work

---

In this part I will talk about the work and contribution of myself in doing this final year project.

Before this final year project

After I look at the list of final year project title, I decide to choose this topic "Parallel Distributed Programming on PS3" without much doubt.

Being interested in parallel programming for a long time, and our department has no courses about parallel programming. I want to take up the chance in doing this project as well as learning parallel programming.

I even bought myself a PlayStation®3 just for this project. It does save us from many troubles when the PlayStation®3 in the viewlab has broken down or not working for both semesters. I also borrowed books and review course material from other university about parallel programming topics before the semester start.

First semester

In first semester, I first try to get myself familiar with PlayStation®3 development environment, includes OS, CellSDK by writing several programming making use of the principle I have studied.

Then our project manager gave us an algorithm which they used in video comparison problem, which is the minimum difference algorithm. I spend most of my time in implementing the program, and doing all type of optimization that I have learnt. My partner also gave me many valuable ideas and helps me in doing most of the measurement.

### Second semester

Starting at second semester, our project manager came up with another algorithm for video comparison, which is the LCS algorithm. I first try to look for existing paralleled LCS algorithm, but failed to find a applicable one to our problem. Then we try to parallelize the LCS algorithm by ourselves. After brainstorming with my partner, we come up with a parallelization method for the LCS algorithm.

After that, I focus on implementation and modification on the LCS algorithm to suit the video comparison problem. I also write codes to apply the optimization method we learnt during last semester. While at the same time, my partner works on a macro program, which can convert the result into a well formatted EAG, and take all the result data.

### Conclusion

It is really an enjoyable experience to work on this project. I have learnt so many skill in writing parallel program, as well as those optimization strategy.

## Chapter 11 Acknowledgement

---

We would like to thank our project supervisor, Professor Michael R. Lyu. He gives us useful advices and provides the resources we need for our project. In addition to this, he also reminds us the importance of concrete statistics and good scheduling.

Besides, we would like to thank Mr. Edward Yau and Mr. Un Tze Lung, who are the research staff in VIEW Lab. They give valuable advices, both conceptual and technical, to our project. They also provided the HL3 raw data to us for our project.

## Chapter 12 Reference

---

1. National Institute of Standards and Technology - longest common subsequence  
<http://nist.gov/dads/HTML/longestCommonSubsequence.html>
2. National Institute of Standards and Technology – subsequence  
<http://nist.gov/dads/HTML/subsequence.html>
3. MIT Multicore Programming Primer: PS3 Cell Programming  
<http://cag.csail.mit.edu/ps3/index.shtml>
4. Wikipedia - Longest common subsequence  
[http://en.wikipedia.org/wiki/Longest\\_common\\_subsequence](http://en.wikipedia.org/wiki/Longest_common_subsequence)
5. A fast parallel algorithm for finding the longest common sequence of multiple biosequences  
<http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1780122>
6. Patterns for Parallel Programming. Mattson, Sanders, and Massingill (2005)
7. IBM Cell Broadband Engine resource center  
<http://www-128.ibm.com/developerworks/power/cell/>
8. A List of Licensed Broadcasting Services in Hong Kong  
[http://www.hkba.hk/en/doc/channel\\_list\\_eng.pdf](http://www.hkba.hk/en/doc/channel_list_eng.pdf)
9. Digital TV Broadcasting officially launched  
<http://www.cedb.gov.hk/ctb/eng/press/pr31122007.htm>
10. Basic concepts on Endianness  
<http://www.codeproject.com/KB/cpp/endianness.aspx>

11. SIMD architectures

<http://arstechnica.com/articles/paedia/cpu/simd.ars>

12. "A Visual Feature based Video Identifying System for the TV Commercial's Monitoring" by Sung Hwan Lee, Won Young Yoo, and Young Suk Yoon.

## Appendix A Result of Cross-Comparing 7 Days'

### TV Easy

#### 1) Overall Result

Date	Total No. of Ad.	No. of Ad. appear only once in the 7 days	No. of Ad. appear at least twice in the 7 days	No. of Ad. Found after Cross-Comparison	False Negative	False Positive	Matching Rate (%)
0304	14	0	14	13	1	0	92.90%
0305	15	2	13	12	1	0	92.30%
0306	13	1	12	11	1	0	92%
0307	15	2	13	13	0	2	100%
0310	20	3	17	15	2	1	88%
0312	15	1	14	14	0	1	100%
0313	14	3	11	11	0	0	100%
							Average: 95%

Fig. A.1 Overall Result

#### 2) Individual Result

Reference:

(百源堂) = Ad. appears only once in all 7 days, i.e. a unique Ad.

Man = Compared Manually

LCS = Compared by our LCS algorithm

Y = A Matched Pair is Found

	030400 TVEASY	0305		0306		0307		0310		0312		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	儲存易迷你倉							Y						Y	
2	Crisis Credit			Y	Y							Y	Y	Y	Y
3	e-print					Y	Y			Y	Y			Y	Y
4	Physical A	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
5	環保袋							Y	Y					Y	Y
6	sling	Y	Y	Y	Y			Y	Y	Y	Y			Y	Y
7	捷達移民			Y	Y			Y	Y					Y	Y
8	apple 迷你倉							Y	Y					Y	Y
9	uwants	Y	Y									Y	Y	Y	Y
10	得成			Y	Y							Y	Y	Y	Y
11	unlimit hosting	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
12	Physical B	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
13	Banner Shop	Y	Y			Y	Y	Y	Y			Y	Y	Y	Y
14	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	7	7	8	7	6	6	8	7	6	6	8	8	14	13
	False Negative		0		1		0		1		0		0		1
	False Positive		0		0		0		0		0		0		0
	Matching Rate (%)		100		87.5		100		87.5		100		100		92.9

Fig. A.2.1 Result of 3/4 TV Easy comparing to 6 other days

	030500 TVEASY	0304		0306		0307		0310		0312		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	Angel					Y		Y		Y	Y			Y	Y
2	Primada									Y	Y			Y	Y
3	Physical A	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
4	slings	Y	Y	Y	Y			Y	Y	Y	Y			Y	Y
5	時昌迷你倉					Y		Y		Y				Y	
6	黃嘉錫律師行							Y	Y					Y	Y
7	Banner Shop	Y	Y			Y	Y	Y	Y			Y	Y	Y	Y
8	(百源堂)														
9	Physical B	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
10	康和堂					Y	Y					Y	Y	Y	Y
11	uwants	Y	Y									Y		Y	Y
12	海港錦鯉					Y	Y	Y	Y	Y	Y			Y	Y
13	unlimit hosting	Y	Y	Y	Y	Y		Y		Y	Y	Y		Y	Y
14	(傷亡權益)														
15	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	7	7	5	5	9	6	8	5	9	8	7	5	13	12
	False Negative		0		0		3		3		1		2		1
	False Positive		0		0		0		0		0		0		0
	Matching Rate (%)		100		100		66.6		62.5		88.9		71.4		92.3

Fig. A.2.2 Result of 3/5 TV Easy comparing to 6 other days



	030600 TVEASY	0304		0305		0307		0310		031		031		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	NOD32							Y	Y	Y	Y	Y	Y	Y	Y
2	葉謝鄧律師行					Y		Y		Y		Y	Y	Y	Y
3	Physical A	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
4	sling	Y	Y	Y	Y			Y	Y	Y	Y			Y	Y
5	髮再生														
6	捷達移民	Y	Y					Y	Y					Y	Y
7	Owell							Y	Y					Y	Y
8	unlimit hosting	Y		Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y
9	Physical B	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
10	Crisis Credit	Y	Y									Y	Y	Y	Y
11	得成	Y	Y									Y	Y	Y	Y
12	(鏗鏘駕駛)							Y						Y	
13	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	8	7	5	5	5	4	8	5	7	6	8	8	12	11
	False Negative		1		0		1		3		1		0		1
	False Positive		0		0		0		0		0		0		0
	Matching Rate (%)		87.5		100		80		62.5		85.7		100		91.7

Fig. A.2.3 Result of 3/6 TV Easy comparing to 6 other days

	030700 TVEASY	0304		0305		0306		0310		0312		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	時昌迷你倉			Y				Y	Y	Y	Y			Y	Y
2	Banner Shop	Y	Y	Y	Y			Y	Y			Y	Y	Y	Y
3	FIA									Y	Y			Y	Y
4	Physical A	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
5	(Naomi 護髮)														
6	unlimit hosting	Y	Y	Y		Y	Y	Y	Y	Y		Y	Y	Y	Y
7	Umart							Y	Y	Y	Y			Y	Y
8	海港錦鯉			Y	Y			Y	Y	Y	Y			Y	Y
9	Angel			Y				Y	Y	Y				Y	Y
10	(寶富麗)														
11	Physical B	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
12	康和堂			Y	Y							Y	Y	Y	Y
13	葉謝鄧律師行					Y		Y	Y	Y		Y		Y	Y
14	e-print	Y	Y							Y	Y			Y	Y
15	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	6	6	9	6	5	4	8	8	11	8	7	6	13	13
	False Negative		0		3		1		0		3		1		0
	False Positive		0		0		0		1		1		0		2
	Matching Rate (%)		100		66.7		80		100		72.7		85.7		100

Fig. A.2.4 Result of 3/7 TV Easy comparing to 6 other days

	031000 TVEASY	0304		0305		0306		0307		0312		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	儲存易迷你倉	Y												Y	
2	(金寶綠水)														
3	(保嬰丹)														
4	Umart							Y	Y	Y	Y			Y	Y
5	環保袋	Y	Y											Y	Y
6	黃嘉錫律師行			Y	Y									Y	Y
7	apple 迷你倉	Y	Y											Y	Y
8	Banner Shop	Y	Y	Y	Y			Y	Y			Y	Y	Y	Y
9	葉謝鄧律師行					Y		Y	Y	Y		Y		Y	Y
10	Angel			Y				Y	Y	Y	Y			Y	Y
11	捷達移民	Y	Y			Y	Y							Y	Y
12	鏗鏘駕駛					Y								Y	
13	NOD32					Y	Y			Y	Y	Y	Y	Y	Y
14	unlimit hosting	Y	Y	Y		Y		Y	Y	Y		Y	Y	Y	Y
15	Owell					Y	Y							Y	Y
16	海港錦鯉			Y	Y			Y	Y	Y	Y			Y	Y
17	sling	Y	Y	Y	Y	Y	Y			Y	Y			Y	Y
18	(APE 寵物)														
19	時昌迷你倉			Y				Y	Y	Y	Y			Y	Y
20	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	8	7	8	5	7	5	8	8	9	7	5	4	17	15
	False Negative		1		3		2		0		2		1		2
	False Positive		0		0		0		1		0		0		1
	Matching Rate (%)		87.5		62.5		71.4		100		77.8		80		88.2

Fig. A.2.5 Result of 3/10 TV Easy comparing to 6 other days

	031200 TVEASY	0304		0305		0306		0307		0310		0313		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	Angel			Y	Y			Y		Y	Y			Y	Y
2	(星輝)														
3	Physical A	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
4	Umart							Y	Y	Y	Y			Y	Y
5	unlimit hosting	Y	Y	Y	Y	Y	Y	Y		Y		Y		Y	Y
6	葉謝鄧律師行					Y		Y		Y	Y	Y		Y	Y
7	e-print	Y	Y					Y	Y					Y	Y
8	時昌迷你倉			Y				Y	Y	Y	Y			Y	Y
9	Physical B	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
10	FIA							Y	Y					Y	Y
11	sling	Y	Y	Y	Y	Y	Y			Y	Y			Y	Y
12	海港錦鯉			Y	Y			Y	Y	Y	Y			Y	Y
13	Primada			Y	Y									Y	Y
14	NOD32					Y	Y			Y	Y	Y	Y	Y	Y
15	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	6	6	9	8	7	6	11	8	9	7	6	4	14	14
	False Negative		0		1		1		3		2		2		0
	False Positive		0		0		0		1		0		0		1
	Matching Rate (%)		100		88.8		85.7		72.7		77.8		66.7		100

Fig. A.2.6 Result of 3/12 TV Easy comparing to 6 other days

	031300 TVEASY	0304		0305		0306		0307		0310		0312		Total	
		Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS	Man	LCS
1	unlimit hosting	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y		Y	Y
2	Physical A	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
3	Crisis Credit	Y	Y			Y	Y							Y	Y
4	得成	Y	Y			Y	Y							Y	Y
5	uwants	Y	Y	Y										Y	Y
6	康和堂			Y	Y			Y	Y					Y	Y
7	(田生集團)														
8	NOD32					Y	Y			Y	Y	Y	Y	Y	Y
9	葉謝鄧律師行					Y	Y	Y		Y		Y		Y	Y
10	(Beauty Pro)														
11	(0101 Hosting)														
12	Physical B	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y
13	Banner Shop	Y	Y	Y	Y			Y	Y	Y	Y			Y	Y
14	景鴻移民	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	No. of Ad. Found	8	8	7	5	8	8	7	6	5	4	6	4	11	11
	False Negative		0		2		0		1		1		2		0
	False Positive		0		0		0		0		0		0		0
	Matching Rate (%)		100		71.4		100		85.7		80		66.7		100

Fig. A.2.7 Result of 3/13 TV Easy comparing to 6 other days

## Appendix B Result of Comparing 2

### One-Hour-Long Videos

	0304	0305	
		Man	LCS
1	晚間新聞		Y
2	TVB (宣傳易 酒店風雲)	Y	Y
3	攻劇策略 (最美麗的第七天)		
4	LOREAL		
5	異度見鬼 A		
6	Gaviscon	Y	Y
7	Canon 相片打印機	Y	Y
8	牙醫選 (牙膏)		
9	惠氏 (奶粉)		
10	FedEx	Y	Y
11	司各脫 枝裝		
12	鼻舒樂 A		
13	Red Bull	Y	Y
14	TVB Sign A	Y	Y
15	宣傳易 Begin	Y	Y
16	儲存易迷你倉		
17	Crisis Credit		
18	e-print		
19	Physical A	Y	Y
20	環保袋		
21	sling	Y	Y
22	捷達移民		
23	apple 迷你倉		
24	uwants	Y	Y
25	得成		
26	unlimit hosting	Y	Y
27	Physical B	Y	Y
28	Banner Shop	Y	Y
29	景鴻移民	Y	Y
30	宣傳易 End	Y	Y
31	勁歌金曲 預告	Y	Y

	0305	0304	
		Man	LCS
1	晚間新聞		Y
2	TVB (宣傳易 酒店風雲)	Y	Y
3	和味無窮 預告		
4	Red Bull	Y	Y
5	Gaviscon	Y	Y
6	港鐵		
7	Olympus	Y	Y
8	Dettol 沐浴露	Y	Y
9	司各脫 粒裝		
10	Canon		
11	3T		
12	專業旅運	Y	Y
13	Canon 相片打印機	Y	Y
14	方大同 B		
15	TVB Sign A	Y	Y
16	宣傳易 Begin	Y	Y
17	Angel		
18	Primada		
19	Physical A	Y	Y
20	sling	Y	Y
21	時昌迷你倉		
22	黃嘉錫律師行		
23	Banner Shop	Y	Y
24	百源堂		
25	Physical B	Y	
26	康和堂		
27	uwants	Y	Y
28	海港錦鯉		
29	unlimit hosting	Y	Y
30	傷亡權益		
31	景鴻移民	Y	Y

32	NBA 預告			32	宣傳易 End	Y	Y
33	SAMSUNG TV			33	太極 預告 A		
34	亮視點			34	獎門人 預告	Y	Y
35	專業旅運	Y	Y	35	OLAY (definity)	Y	Y
36	捐血			36	FedEx	Y	Y
37	SK-II A	Y	Y	37	交通規則		
38	Prudential			38	10,000 BC B		
39	幸福傷風咳素			39	Rejoice		
40	TVB Sign B (+PG 家長指引)	Y	Y	40	新聞透視 預告		
41	酒店風雲 主題曲	Y	Y	41	比華利山		
42	酒店風雲 15-1			42	勁歌金曲 預告	Y	Y
43	酒店風雲 Before Ad	Y	Y	43	幸福傷風感冒素		
44	邦民	Y	Y	44	TVB Sign B	Y	Y
45	獎門人 預告	Y	Y	45	酒店風雲 主題曲	Y	Y
46	星期二檔案 預告			46	酒店風雲 16-1		
47	FANCL	Y	Y	47	酒店風雲 Before Ad	Y	
48	雀巢咖啡			48	邦民	Y	Y
49	Dettol 消毒藥水			49	古靈精偵		
50	Olympus	Y	Y	50	市場策劃獎 A		
51	BEA Funds			51	JobsDB.com		
52	嘉士伯 (飛機)	Y	Y	52	方大同 C		
53	Y.E.S.			53	心理痛		
54	白兔牌快眠精	Y		54	康泰旅行	Y	Y
55	eye (PCCW)	Y	Y	55	市場策劃獎 B		
56	異度見鬼 B	Y	Y	56	嘉士伯 (飛機)	Y	Y
57	使立消			57	碧蓮 去漬霸		
58	Dettol 沐浴露	Y	Y	58	SK-II A	Y	Y
59	鼻舒樂 B			59	eye (PCCW)	Y	Y
60	TVB 週刊			60	異度見鬼 B	Y	Y
61	酒店風雲 After Ad	Y	Y	61	TVB 奧運		
62	酒店風雲 15-2			62	酒店風雲 After Ad	Y	Y
63	酒店風雲 Before Ad	Y		63	酒店風雲 16-2		
64	攻劇策略 (太極)			64	酒店風雲 Before Ad	Y	Y
65	McDonald			65	太極 預告 B		
66	方大同 A	Y	Y	66	奧運 (兵兵)		
67	Canon 相片打印機		Y	67	FANCL	Y	Y
68	Dettol 沐浴露			68	方大同 A	Y	Y
69	京都念慈庵			69	Pizza Hut		

70	嘉士伯 (降落傘)		
71	天上野 A		
72	Neway	Y	Y
73	OLAY A		
74	酒店風雲 After Ad		
75	酒店風雲 15-3		
76	酒店風雲 Before Ad	Y	Y
77	Music Link	Y	Y
78	Gaviscon		
79	NOW TV	Y	Y
80	10,000 BC A		
81	OLAY (definity)	Y	Y
82	Coca Cola (眼 劑 x2)	Y	Y
83	康泰旅行	Y	Y
84	enJoy Card	Y	Y
85	酒店風雲 After Ad	Y	Y
86	酒店風雲 15-4		
	No. of Ad. Found	42	<b>40</b>
	False Negative		<b>2</b>
	False Positive		<b>2</b>
	Matching Rate (%)		<b>95%</b>

70	Coca Cola (眼 劑 x2)	Y	Y
71	潘婷		
72	samhelp (版權法)		
73	天上野 B		
74	Music Link	Y	Y
75	AMTD		
76	法國雙飛人		
77	異度見鬼 C		
78	香港筆跡 預告		
79	酒店風雲 After Ad	Y	Y
80	酒店風雲 16-3		
81	酒店風雲 Before Ad	Y	Y
82	NOW TV	Y	Y
83	10,000 BC C		
84	Nikon		
85	白兔牌快眠精	Y	
86	Neway	Y	Y
87	enJoy Card	Y	Y
88	SK-II B		
89	碧蓮 洗衣粉		
	No. of Ad. Found	42	<b>39</b>
	False Negative		<b>3</b>
	False Positive		<b>1</b>
	Matching Rate (%)		<b>93%</b>