RISC-Based Simulation of Longest Common Subsequence Algorithm in MIPS64 Simulators

Glenn Paul P. Gara* †, Mark Phil B. Pacot* ‡, Roger Luis T. Uy*
*College of Computer Studies, De La Salle University, Manila, Philippines
†Information Technology Education Program, University of the Immaculate Conception, Davao City, Philippines
‡College of Engineering and Information Technology, Caraga State University, Butuan City, Philippines
{glenn_gara, mark_phil_pacot, roger.uy}@dlsu.edu.ph

***Abstract*—The longest common subsequence (LCS) is an essential technique in the sequence alignment. By deleting zero or more symbols, it determines one of the longest subsequences in a sequence. This paper described a simulation of the algorithm using the EduMIPS64 and MIPSers simulators where the latter implements the most recent developments of MIPS64 instruction sets. The method applied to solve the LCS in this study was the first known solution invented by Wagner and Fischer. The authors programmed the LCS on a RISC architecture and evaluated the results of the test cases by observing the number of clock cycles performed through the simulators. Results show that the MIPSers executed the least number of clock cycles compared to the EduMIPS64 simulator.**

***Index Terms*—Longest common subsequence, MIPS64, EduMIPS, MIPSers, RISC architecture**

I. Introduction

The tools needed to interpret the enormous amount of biological data particularly biosequences is in the realm of computer science [1]. One of the most significant tasks of bioinformatics in analyzing the sequences is the searching of the longest common subsequence (LCS) [2]. Given the two strings, it determines the longest subsequences by deleting zero or more symbols [3]. Since the input size of biosequences grows exponentially, a high-performance implementation of LCS is necessary to increase throughput. Several implementations of the LCS were performed using sequential processing. This type of processing is commonly used to program instructions using a high-level programming language which can execute only one instruction at a time [4].

Meanwhile, research work on the area of pipelining processing particularly using MIPS64 instructions is scarce. Pipelining is a technique whereby several instructions overlapped in execution. It is a high-performance implementation, permitting multiple instructions to be in some phase of execution at the same time. MIPS is a RISC architecture that was designed originally to support pipelining with ease of implementation. It is based on the load/store architecture like the other RISC processors [5], [6]. MIPS64 Instruction Set Architecture (ISA) is commonly used in computer architecture programming due to its orthogonality and suitability in the real-world application. Currently, there are a couple of tools to simulate and visualize the execution of MIPS64 instructions where EduMIPS64 is the most used in the academia as a supporting tool for teaching. A newly developed simulator called MIPSers supports the MIPS release 6 and the only simulator today that can demonstrate the major changes in instruction sets [7].

In this study, the researchers simulated MIPS64 instructions in solving the longest common subsequence of the two sequences. It follows an assessment of the clock cycle results of the algorithm using the EduMIPS64 and MIPSers simulators..

II. Related Works

The longest common subsequence is a classical method based on the principle of dynamic programming [8]. It is a useful technique in aligning biological sequences that mostly implemented sequentially [9] . The study of [10] implemented it using Java with dual match index to reduce significant numbers of false positives in the match result. C# and VB.net were utilized by [11] in implementing the LCS to identify the succeeding movement of a user in a browser by classifying user navigation patterns.

The LCS has also employed in C/C++ and Perl as a library function for basic analysis toolkit for biological sequences and they named the software BATS [12]. Due to numerous sequential implementation of LCS, it is important to note that the performance of it should be given high regard especially when there is an increasing number of input size of sequences. Thus, there are few ways on how this kind of algorithm can be implemented to handle the inflation of input size, and one is through parallelism. A study of [8] uses a GPU to implement LCS algorithm. The technique exploited a large number of processing units and the unique memory-accessing properties to achieve high performance. A CUDA technology for GPU implementation of the algorithm shows that it is faster compared to its sequential implementation [13].

In this paper, the researchers make use of instruction-level parallelism which is known as pipelining to implement the algorithm in solving the longest common subsequence. They used MIPS64 instruction sets to program the LCS through a CPU simulator since the said type of parallelism is not well-explored. The researchers examined the number of clock cycles performed by EduMIPS64 and MIPSers simulators using various test cases.

III. Background

This section presents the method on how the LCS problem was solved using the technique developed by Wagner and Fischer. It also presents the MIPS64 simulators used to simulate the said technique.

A. *Longest Common Subsequence*

The longest common subsequence is not new in the field of computer science and biology. Given the two sequences $M = [m_1, m_2, \ldots, m_k]$ and $N = [n_1, n_2, \ldots, n_k]$, the output should be any one longest common subsequence having maximum possible length [14]. For example, **TUSDAY** is the longest common subsequence of **TUESDAY** and **ThUrSDAY** [3]. There are two common approach in solving the LCS, the recurrence and dynamic approach. The following is the recurrence relation for solving LCS Problem:

$$x = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ LCS[m-n, n-1] + 1 & \text{if } M[m] = N[n] \\ max(LCS[m-1, n], LCS[m, n-1]) & \text{if } M[m] \neq N[n] \end{cases} \tag{1}$$

Equation 1 shows the relation extending the LCS length for each prefix pair $(M[1...m], N[1...n])$. Here, $LCS[m, n]$ indicates the length of $LCS(M[1...m], N[1...n])$. The length of $LCS(M, N)$ is given by $LCS(k, k)$. $LCS(M, N)$ can be obtained by backtracking from $LCS(k, k)$.

The dynamic programming approach has two properties. The first property considers two sequences M and N of lengths m and n respectively. Let the elements of these sequences are denoted as $m_1, m_2 ... m_k$ and $n_1, n_2 ... n_k$ respectively. Then, $LCS(M_m, N_n) = LCS(M_{m-1}, N_{n-1}) + m_k$ if $m_k = n_k$. The $+$ refers to the final element of sequence M appended to the prefix of sequence. This property states that when the last element of sequence M and N are equal, it can be pruned to make the sequences shorter. Thus, the LCS is calculated on the remaining shorter sequences. Elements are pruned until the final element of M and N sequences are equal. LCS is computed on this shorter sequence and the last element is appended to this LCS to obtain the final LCS output. The second property states that if the last element of sequence M and N are not equal then the LCS is the maximum of the sequences $LCS(M_{m-1}, N_n)$ and $LCS(M_m, N_{n-1})$.

B. *EduMIPS64 and MIPSers CPU Simulator*

MIPS is an instruction set architecture (ISA) based on the reduced instruction set computer (RISC) formed by MIPS Technologies [15]. A simulator is necessary for simulating and visualizing a MIPS code.

In this study, EduMIPS64 and MIPSers simulators perform the simulation of the code. EduMIPS64 is popularly known and is used in the academia to teach students in assembly programming. It is also a visual debugger that allows the user to see the status of registers and memory, the behavior of instructions in the pipeline and how the CPU handled the stalls [16].

A newly developed simulator called MIPSers is based on the current release 6 version of MIPS. Only MIPSers support the said release is considered the most important feature of a simulator. The version consists of major changes such as branch without branch delay slots, multiplication and division without the use of HI/LO registers, selection operations, floating point comparison and bit swap operation. Similar to EduMIPS64, It was developed to supplement the learning of the students studying computer architecture. Such features would help the program to minimize the number of clock cycles necessary to improve its performance in terms of speed.

IV. Implementation

The authors developed a set of instructions to solve the LCS problem of the given two sequences using the MIPS64 instruction sets. This study applied the first technique to solve the LCS problem invented by [17] using the simulators that can demonstrate the MIPS64 code. To leverage the release 6 version of MIPSers, the implementation of instructions are different per simulator.

A. *MIPS64 Programs*

This section presents how the algorithm was implemented using MIPS64 instructions sets.

1) Identify the string if it is empty: The first thing to do is to check if the string is empty. Once the condition is true, the program will return only a zero value. See figure 1 for the MIPS64 implementation.

2) Comparing the last index of the strings: Based on the first property in solving the LCS problem, if the last element of the sequence M and N are equal, prune the last element to make the sequences short. The LCS can be calculated using the shorter sequences. The last elements are removed in M and N sequences until it becomes equal. To get the final LCS, the LCS is computed from the shorter sequence and append the last element to this LCS. See figure 1 for the MIPS64 implementation.

3) Getting the maximum of sequences: If the last element of the sequence M and N are not equal, then the LCS is the maximum of $LCS(M_{m-1}, N_n)$ and $LCS(M_m, N_{n-1})$ sequences. See figure 2 for the MIPS64 implementation.

V. Simulation and Results

The MIPS64 assembly code in solving the longest common subsequence problem has been successfully simulated using EduMIPS64 and MIPSers simulators. To know if the code returns the correct output, a test was conducted using the test cases seen in table I.

Table II shows the LCS results and its corresponding cycle count in EduMIPS64 and MIPSers. As the input strings

```
con1:
    dsubu r4,r4,r7
    lb r15,l1(r4)
    beq r15,r0,exit
    dsubu r5,r5,r7
    lb r16,l2(r5)
    beq r16,r0,exit
    daddiu r1,r0,#0000
    daddiu r6,r0,#0000
    dadd r12,r0,r4
    dadd r13,r0,r5
    lb r9,l1(r12)
    lb r10,l2(r13)
    beq r9,r10,saveLCS1
    j addUp1
    dadd r5,r0,r1 ; /*length of L2*/
```

(a) EduMIPS64

```
con1:
    dsubu r4,r4,r7
    lb r15,l1(r4)
    beq r15,r0,exit
    daddiu r31,r0,#1
    dsubu r5,r5,r7
    lb r16,l2(r5)
    beq r16,r0,exit
    daddiu r31,r0,#1
    daddiu r1,r0,#0000
    daddiu r6,r0,#0000
    daddu r12,r0,r4
    daddu r13,r0,r5
    lb r9,l1(r12)
    lb r10,l2(r13)
    beq r9,r10,saveLCS1
    daddiu r31,r0,#1
    j addUp1
    daddiu r31,r0,#1
```

(b) MIPSers

Fig. 1: Instructions to identify the string if it is empty and last index comparison

```
addUp1:
    slt r22,r4,r5
    beq r22,r7,up1.1
    dadd r4,r4,r7
    j loadA
up1.1:
    dadd r5,r5,r7
    j loadA
exit:
    NOP
```

(a) EduMIPS64

```
addUp1:
    slt r22,r4,r5
    beq r22,r7,up11
    daddiu r31,r0,#1
    daddu r4,r4,r7
    j loadA
    daddiu r31,r0,#1
up11:
    daddu r5,r5,r7
    j loadA
    daddiu r31,r0,#1
exit:
    NOP
```

(b) MIPSers

Fig. 2: Instructions on getting the maximum sequences

TABLE I: Test cases

Test Case	String 1	String 2
T1	NEMATODE	EMPTY
T2	KNOWLEDGE	BOTTLE
T3	AABAABAAB	ABAB
T4	ABCBDAB	BDCABDB
T5	MZJAWXU	XMJYAUZ

TABLE II: LCS and number of cycles

Test Case	LCS	EduMIPS64 Cycles	MIPSers Cycles
T1	EMT	506	367
T2	OLE	803	621
T3	ABAB	386	268
T4	BCBDB	934	639
T5	MJAU	949	652

increased its size, the clock cycles also increased. By observing the table, the two simulators performed differently with regards to the number of cycles for each test case. Test case 1 runs in EduMIPS64 with 506 cycles and 367 cycles in MIPSers respectively. The number of cycles in test case 2 is significantly higher than the first test case with 803 cycles in EduMIPS64 and 621 in MIPSers. Test case 3 executes the least number of cycles with 386 in EduMIPS64 and 268 in MIPSers. In test case 4, EduMIPS64 returns 934 cycles while MIPSers has 639 cycles only. Lastly, test case 5 executes the most number of cycles where EduMIPS64 returns 949 while MIPSers executes only with a cycle count of 652. Among the two simulators, EduMIPS64 executes the most number of cycles due to the difference of implementation of branch delay slot leading to such control hazards (pipeline 1 and pipeline 2).

VI. Conclusion

Using the technique of Wagner and Fischer to solve the LCS problem of the two sequences, the authors were able to program it as a MIPS64 assembly code and simulated it using the EduMIPS64 and MIPSers simulators. However, due to the difference of code implementation especially in branch delays, the results also has huge disparity. It turned out that the MIPSers performed better compared to the EduMIPS64 simulator due to the major updates of instructions of the former.

References

[1] J. Cohen, "Bioinformatics — an introduction for computer scientists," *ACM Computing Surveys (CSUR)*, vol. 36, no. 2, pp. 122–158, 2004.

[2] A. Gorbenko and V. Popov, "On the longest common subsequence problem," *Applied Mathematical Sciences*, vol. 6, no. 116, pp. 5781–5787, 2012.

[3] N. Nakatsu, Y. Kambayashi, and S. Yajima, "A longest common subsequence algorithm suitable for similar text strings," *Acta Informatica*, vol. 18, no. 2, pp. 171–179, 1982.

[4] A. Silver, "Rethinking CS101 [Resources_Education]," *IEEE Spectrum*, vol. 54, no. 4, p. 23, 2017.

[5] S. P. Dandamudi, *Guide to RISC processors: for programmers and engineers*. Springer Science & Business Media, 2005.

[6] I. Pantazi-Mytarelli, "The history and use of pipelining computer architecture: MIPS pipelining implementation," in *Systems, Applications and Technology Conference (LISAT), 2013 IEEE Long Island*, 2013, pp. 1–7.

[7] N. M. D. Kho and R. L. Uy, "MIPSers: MIPS extension release 6 simulator," *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–6, 2017.

[8] J. Yang, Y. Xu, and Y. Shang, "An efficient parallel algorithm for longest common subsequence problem on GPUs," in *Proceedings of the World Congress on Engineering*, vol. 1, 2010, pp. 499–504.

[9] A. Dhraief, R. Issaoui, and A. Belghith, "Parallel computing the Longest Common Subsequence (LCS) on GPUs: efficiency and language suitability," in *The 1st International Conference on Advanced Communications and Computation (INFOCOMP)*, 2011.

[10] T. S. Han, S.-K. Ko, and J. Kang, "Efficient subsequence matching using the longest common subsequence with a dual match index," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, 2007, pp. 585–600.

[11] M. Jalali, N. Mustapha, M. N. Sulaiman, and A. Mamat, "A recommender system approach for classifying user navigation patterns using longest common subsequence algorithm," *American Journal of Scientific Research*, vol. 4, pp. 17–27, 2009.

[12] R. Giancarlo, A. Siragusa, E. Siragusa, and F. Utro, "A basic analysis toolkit for biological sequences," *Algorithms for Molecular Biology*, vol. 2, no. 1, p. 10, 2007.

[13] S. Deorowicz, "Solving longest common subsequence and related problems on graphical processing units," *Software: Practice and Experience*, vol. 40, no. 8, pp. 673–700, 2010.

[14] A. N. Arslan and Ö. EĞECIOĞLU, "Algorithms for the constrained longest common subsequence problems," *International Journal of Foundations of Computer Science*, vol. 16, no. 06, pp. 1099–1109, 2005.

[15] I. S. MIPS IV, "Revision 3.2, MIPS Technologies," *Inc., entire publication submitted (Sep. 1995)*, 1995.

[16] A. Spadaccini, "EduMIPS64 Documentation," 2017.

[17] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.