electronics

Article

An Extended Instruction Set for Bioinformatics' Multiple Sequence Alignment †

Anargyros Gkogkidis [1,2,*,‡], Vasileios Tsoukas [1,2,‡] and Athanasios Kakarountas [1,2]

1 Department of Computer Science and Biomedical Informatics, University of Thessaly, 35131 Lamia, Greece
2 Intelligent Systems Laboratory, University, 35131 Lamia, Greece
* Correspondence: agkogkidis@uth.gr
† This paper is an extended version of our paper published in Gkogkidis, A.; Kakarountas, A. Exploration Study on Configurable Instruction Set for Bioinformatics' Applications. In Proceedings of the 2019 Panhellenic Conference on Electronics & Telecommunications (PACET), Volos, Greece, 8–9 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
‡ These authors contributed equally to this work.

Abstract: Multiple Sequence Alignment (MSA) is one of the most fundamental methodologies in Bioinformatics and the method capable of arranging DNA or protein sequences to detect regions of similarity. Even on cutting-edge workstations, the MSA procedure requires a significant amount of time regarding its execution time. This paper demonstrates how to utilize Extensa Explorer by Tensilica (Cadence) to create an extended instruction set to meet the requirements of some of the most widely used algorithms in Bioinformatics for MSA analysis. Kalign showed the highest acceleration, reducing Instruction Fetches (IF) and Execution Time (ET) by 30.29 and 43.49 percent, respectively. Clustal had acceleration of 14.2% in IF and 17.9% in ET, whereas Blast had 12.35% in IF and 16.25% in ET.

Keywords: ASIP; hardware accelerator; re-configurable instruction-set; bioinformatics; multiple sequence alignment

check for updates

Citation: Gkogkidis, A.; Tsoukas, V.; Kakarountas, A. An Extended Instruction Set for Bioinformatics' Multiple Sequence Alignment. *Electronics* **2022**, *11*, 2550. https://doi.org/10.3390/electronics11162550

Academic Editor: Wojciech M. Zabołotny

Received: 16 July 2022
Accepted: 7 August 2022
Published: 15 August 2022

1. Introduction

In comparison to their predecessors from the 1970s and 1980s, modern microprocessors have undergone a radical transformation due to breakthroughs in device technologies, design techniques, and programming paradigms. In modern processing device and packaging technologies, architectures, and programming interfaces, heterogeneity has grown prevalent, yielding unprecedented performance, power efficiency, and functionality improvements.

Various health monitoring applications have been developed over the past decades; however, the current trend in personal health monitoring is toward wearable devices. As an illustration, the insulin pumps described in [1] need the processing of vast volumes of data or decision-making regarding various crucial aspects. Even in football, players wear sensor-equipped vests so that health-related team experts can monitor their physical condition and identify possible health issues [2]. As a result, it has become essential to build microprocessors that have a low impact on the amount of energy they consume while they are capable of meeting adequate data processing capacity. This is the case with Reduced Instruction Set Computer (RISC) processors, which execute algorithms swiftly while consuming less energy than traditional CPUs.

Each year, the sectors of health monitoring and health care release brand-new mobile applications designed with customers' specific needs in mind. This field's methodology is derived from the Bioinformatics domain, which combines biological research techniques and methods from applied information technology. Today, bioinformatics is a rapidly developing field crucial for creating phylogenetic trees, predicting protein structures, and discovering new medications; thus, its significance cannot be overstated. Multiple sequence alignment (MSA) is the essential stage in carrying out the aforementioned activities [3]. The MSA is a process in which more than two sequences are aligned, in contrast to pairwise sequence alignment, in which only two sequences are matched. The objective of both pairwise and MSA is to identify areas of similarity [4]. MSA has a far greater computational cost than pairwise sequence alignment.

With the advent of high-sequencing technology, both the sequences and the data are growing exponentially. The analysis of these sequences is a considerable challenge regarding its complexity and is widely identified as a demanding task. There are several computer strategies for the MSA issue resolution, including the approach of dynamic programming, which is slow but highly accurate. Initially, the dynamic programming approach was only employed for pairwise sequencing; later, it was also explored for MSA. Achieving the ideal alignment is proved by the research community to be a computationally challenging task [3]. The information utilized as algorithm input substantially determines the algorithms' needs and computational complexity. As will be detailed in further depth in the following sections, the input for this method might range from a few hundred to several thousand distinct DNA or protein sequences. Therefore, the amount of time necessary to finish the execution in each scenario is directly influenced by the input.

Recent breakthroughs in experimental methods have substantially improved the ability to identify protein structures experimentally, yet the gap between the number of protein sequences and the number of known protein structures continues to grow. Prediction of the computational protein structure is one technique to fill this void. The performance of AlphaFold2 in the most recent Critical Assessment of protein Structure Prediction (CASP) demonstrates that Deep Learning (DL)-based techniques have recently facilitated significant progress in the field of protein structure prediction (CASP14). In work [5], the authors emphasize significant advancements and milestones in the field of protein structure prediction due to DL-based approaches, as shown in CASP studies. In addition, they discuss advancements in protein contact map prediction, protein distogram prediction, protein real-valued distance prediction, and Quality Assessment/refinement stages of the protein structure prediction pipeline. Furthermore, the authors present several end-to-end DL-based techniques for protein structure prediction. Moreover, given that there have been recent DL-based advancements in protein structure determination via Cryo-Electron (Cryo-EM) microscopy, some of the significant advancements in the field are highlighted. In conclusion, they offer an overview and potential future research objectives for DL-based techniques in the field of protein structure prediction. DL-based approaches for MSA are outside the scope of this study, however, they are highlighted as a foundation for future research.

The purpose of this study is to investigate the possibility of discovering frequently executed code kernels in bio-algorithms. This may then assert the development of a new instruction, or instructions, that could extend the Instruction Set of a re-configurable CPU. As a result, it would allow the integration of a subset of instructions tailored for Bioinformatics algorithms. The anticipated outcome is the creation of a low-power, low-cost RISC processor capable of data processing and can be utilized in wearable or portable health monitoring/care systems. The new processing intellectual property (IP) core with the extended instruction set architecture (ISA) can be included in any very-large-scale integration VLSI system embedding other components for the latter applications.

In this work, a demonstration of how to construct an extended instruction set for the most commonly used MSA algorithms is provided. The rest of the paper is organized as follows: Section 2 provides a brief description of the algorithms under examination. Section 3 provides an overview of various re-configurable processor platforms. Section 4 introduces the Xtensa tool from Tensilica. Section 5 provides an analytical overview of the acceleration process and the experiments' findings. Section 6 concludes with the results and presents future plans.

2. Bioinformatics Algorithms

The objective of MSA methods is to construct alignments that represent the biological relationships of different sequences. In reality, approximation techniques are employed to align sequences by maximizing their similarity, as it is computationally virtually unfeasible to compute accurate MSAs [6]. These approaches may be applied to DNA, RNA, or protein sequences and account for evolutionary events such as mutations, insertions, deletions, and rearrangements under specific conditions [7]. MSA is a fundamental modeling tool since its creation involved resolving various computational and biological challenges. It has been known for decades that the calculation of an accurate MSA is an NP-complete challenge, which explains the fact that the scientific community is actively researching the field, and the reason behind that more than 100 different approaches have been devised in the previous three decades. Moreover, improvements in precision, the extension of MSA approaches' applicability, and large-scale alignments are among the ongoing research fields [8]. In addition, alignment precision is critical for an extensive array of analyses, frequently in difficult-to-evaluate ways. Several benchmarking methodologies have been pursued for the purpose of comparing the performance of different aligners and detecting systematic mistakes in alignments [9]. Clustal, Blast, and Kalign are three of the most utilized MSA algorithms. In the following paragraphs, a brief description of the aforementioned algorithms is provided.

2.1. Clustal Family

The Clustal family of algorithms is a group of frequently used tools for MSA in Bioinformatics. Clustal W and Clustal X have been the most popular programs for generating MSAs during the past 30 years, while Clustal Omega, the last stable version, is capable of making larger sequence alignments accurately and fast [10]. An overview of the aforementioned algorithms follows.

2.1.1. Clustal W

Clustal W is a publicly accessible application with many adjustments integrated into the procedure of MSA when compared to other similar methods. The most notable ones are the following. Regarding the alignment of divergent protein sequences, the sensitivity of the frequently employed progressive MSA approach has been significantly increased. First, separate assigned weights to each sequence in a partial alignment are applied to deweight nearly identical sequences and increase the weight of the most dissimilar sequences. At successive phases of alignment, amino acid substitution matrices are modified based on the variance of the sequences to be related. Additionally, in hydrophilic locations, new gaps are stimulated in potential loop regions rather than the regular secondary structure. Finally, to promote the development of fresh gaps at these points, local gap penalties are decreased for positions in initial alignments where gaps have been created [11].

2.1.2. Clustal X

Clustal X is an updated version of the commonly utilized progressive MSA software Clustal W. A new variable sequence coloring scheme integrated into the systems enables the user to emphasize conserved characteristics in the alignment. Additional new features include the ability to cut-and-paste sequences to modify the alignment's order, the ability to realign the selected subset of sequences, and finally, it can also reinsert a selection of a subrange of the alignment back into the original alignment. Additionally, it is possible to do an alignment quality study and identify low-scoring regions or unusual residues. Quality analysis and realignment of specified residue ranges offer the user a potent instrument for enhancing and refining challenging alignments and detecting inaccuracies in input sequences [12].

2.1.3. Clustal Omega

Clustal Omega is the most recent member of the Clustal family of MSA generators. It was created about a decade ago in response to the rapidly growing amount of accessible sequences and the necessity to generate large alignments rapidly and precisely. Clustal Omega's code has been maintained, and various bugs have been fixed throughout the years. Nucleotide sequences are now supported as a new fundamental aspect of Omega's functionality. Furthermore, several additional input and output features, such as the ability to read compressed files and multi-byte characters in sequence names, were introduced. Printing of residue numbers and control over line lengths and sequence order are among the new output features. The runtimes of Clustal Omega scale effectively with the number of sequences, which is achievable due to the usage of the mBed method for guiding trees and the parallelization of the distance matrix calculation exploiting several threads. Additionally, the progressive phase has been primarily parallelized, allowing for increased scalability regarding large numbers of sequences [10].

2.2. BLAST

BLAST attempts to discover homologous proteins and DNA sequences based on sequence similarity that is excessive. If two sequences have significantly more similarity than predicted by chance, the most concise overview for the high similarity is similar ancestry or homology. Effective similarity searches analyze protein sequences, not DNA sequences, for sequences that encode proteins and utilize expectation values. The BLAST software enables the comparison of sequences regarding protein and DNA to protein databases. In addition, it can either be executed on popular websites but it may also be run locally. Target databases can be tailored to the sequence data being described using the local installation. Nowadays, colossal protein databases demand enhanced search sensitivity, which can be achieved by scanning less comprehensive databases, such as a complete protein set from an evolutionarily related model organism. In conclusion, BLAST gives extremely precise statistical estimates that may be used to correctly identify protein sequences that originated more than 2 billion years ago [13].

2.3. Kalign

Kalign is a highly effective MSA tool that can align thousands of protein or nucleotide sequences. Nonetheless, current alignment difficulties involving several sequences surpass the initial design specifications of Kalign. The program currently employs a SIMD (single instruction multiple data) accelerated version of the bit-parallel Gene Myers [14] method to estimate pairwise distances, a sequence embedding technique, and the bisecting K-means algorithm to generate guide trees for thousands of sequences swiftly. The new version is capable of scaling more effectively when compared to other existing MSA tools and retains good alignment accuracy for both protein and nucleotide alignments. Compared to other programs such as Clustal Omega, in two out of the six Bralibase alignment categories, the mean performance of Kalign is considerably superior. Nevertheless, it must be emphasized that each approach's performance might vary significantly based on the particular alignment scenario [15].

3. Re-Configurable Processor Platforms

In order to keep up with the rapidly developing research and technology industries in recent years, more robust computer systems have become necessary. This is due to the necessity of keeping up with rapidly evolving research and technical domains. As a result, several research groups and firms have been motivated to develop creative solutions with the objective of decreasing the amount of time required for a computer system to respond, the amount of energy it consumes, its size, and, finally, its cost.

Expansion of a central processing unit's instruction set is a prime example of this tactic. Therefore, it is possible to design a processor for a particular application that reduces the time required for response and data processing, as well as the amount of the required energy.

It is feasible to design new instruction sets for processors, and once this is accomplished, the code will need to be refactored by relatively easy modifications in order to utilize the newly developed instruction set. For example, executing a calculation involving the addition of three numbers will require N cycles for each integer included in the computation. Due to the fact that the translation into assembly language will now use the new instruction that executes the addition of three integers at the lowest possible cost, the extension allows the reduction of the number of required execution cycles. The aforementioned process is made feasible as a direct result of the extension's enhanced functionality, and also, it is accomplished due to the fact that the instruction adds the three numbers using as few resources as possible.

3.1. Field Programmable Gate Array Designs

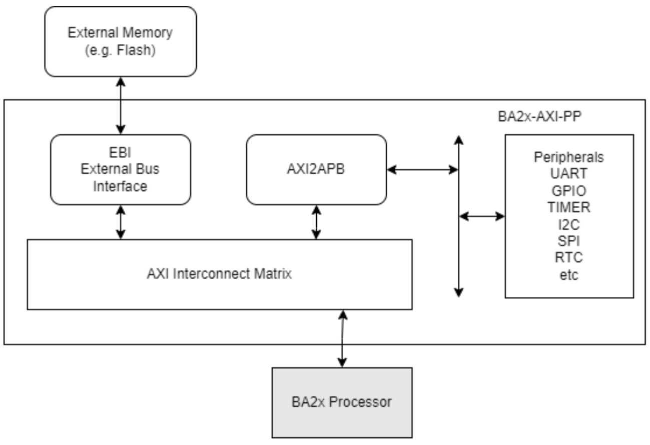
The Field Programmable Gate Array (FPGA) platform is utilized extensively in embedded systems and applications requiring a specialized computer system. FPGAs are silicon devices that can be electrically programmed to become virtually any digital circuit or system. They offer several appealing advantages over fixed-function Application Specific Integrated Circuit (ASIC) technologies. A fundamental aspect of FPGA is that it contains a very large number of standard gates and other digital operations such as counters and memory registers. Nonetheless, it lacks a processor ready to run the software, and it is the user's responsibility to construct the circuit.

3.2. CAST BA2x IP Core

The IP Core BA2x series from CAST is yet another option that can be used in the process of designing and manufacturing an application-specific processor [16], as depicted in Figure 1. The main differentiating feature of this product is its PipelineZero architecture [17], which eliminates the waiting time that would typically occur between the execution of individual code fragments. Thus, this family of IP Cores manages to:

- Reduce risks of structural hazards, control hazards, and data hazards, which results in better performance.
- Require a limited number of flip-flops and has fewer pipeline registers, which results in a processor that has small form factor.
- Have a small energy consumption footprint.

Furthermore, hardware designers can increase the capabilities of the CPU by multiplying several blocks, implementing the IEEE-754 standard for floating point units [18], and implementing multipliers and divisions at the hardware level. The process, as stated above, is possible by simply multiplying many blocks together.



External Memory (e.g. Flash)
BA2x-AXI-PP
EBI External Bus Interface
AXI2APB
Peripherals UART GPIO TIMER I2C SPI RTC etc
AXI Interconnect Matrix
BA2x Processor

Figure 1. Layout of the BA2x family of processors, depicting all accessible communication lines and peripheral processing units [16].

3.3. Tensilica—Cadence

It was determined that the Xtensa from Tensilica (Cadence) would be the optimal choice for the reconfigurable central processing unit. The decision on whether to choose this platform or not was heavily influenced by the fact that the entire development platform is cloud-hosted. The repute of the licensing scheme among university research teams was another crucial aspect. In addition, the tools (compiler included) and the methodology used to extend the Instruction Set of Xtensa are user-friendly, dependable, and well-described in their respective documentation.

Tensilica has created a two-way approach in order to use its platform. The user can either focus on choosing a predefined IP core that can be configured or a software extension by defining new instructions, as shown in Figure 2. The former option allows the use and modification of existing IP cores in a library accessible from the Integrated Development Environment (IDE), which supports various categories, such as HiFi audio, Imaging, AI, and others [19]. The latter option was developed to enhance the CPU's capabilities by extending the instruction set. Thus, Tensilica has developed the Tensilica Instruction Extension (TIE) language, similar to Verilog, which allows the creation of new application-specific instructions. In this work, a blank configuration was utilized to illustrate the capabilities of the TIE programming language.

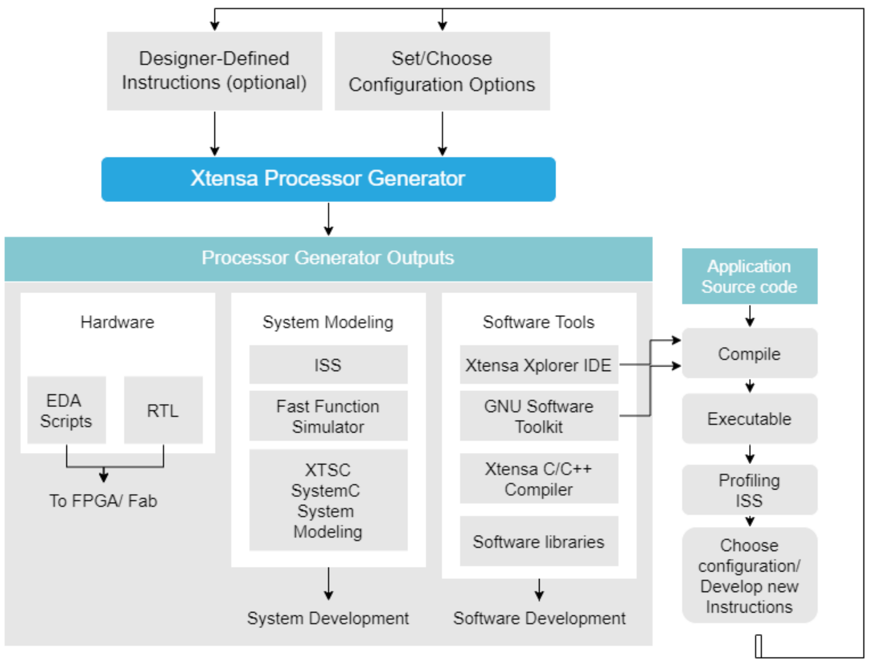


Figure 2. Tensilica design schema [20].

In order to accelerate the speed of a targeted algorithm, it is necessary to study the implementation code. Profiling the code execution with typical inputs is the required procedure. Based on the number of repetitions, the most demanding code kernels are then sorted. If the amount of instructions in a kernel (often a loop) is sufficient, the kernel may be used to create a new instruction. On this basis, the number of instruction fetches from memory is drastically decreased, along with the total number of clock cycles in the decoder unit. As a result, utilizing the TIE programming language enables the development of new instruction. All the above operations are carried out in a cloud simulation provided by the Xtensa platform at Cadence premises. Using a cycle-correct SystemC-compatible simulation model and instruction set simulator, refs. [21,22] enable developers to work without a physical development board. This ensures that all testing and simulations are accurate. Consequently, Electronic Design Automation (EDA) synthesis scripts can be used to transfer the design to a hardware-based board.

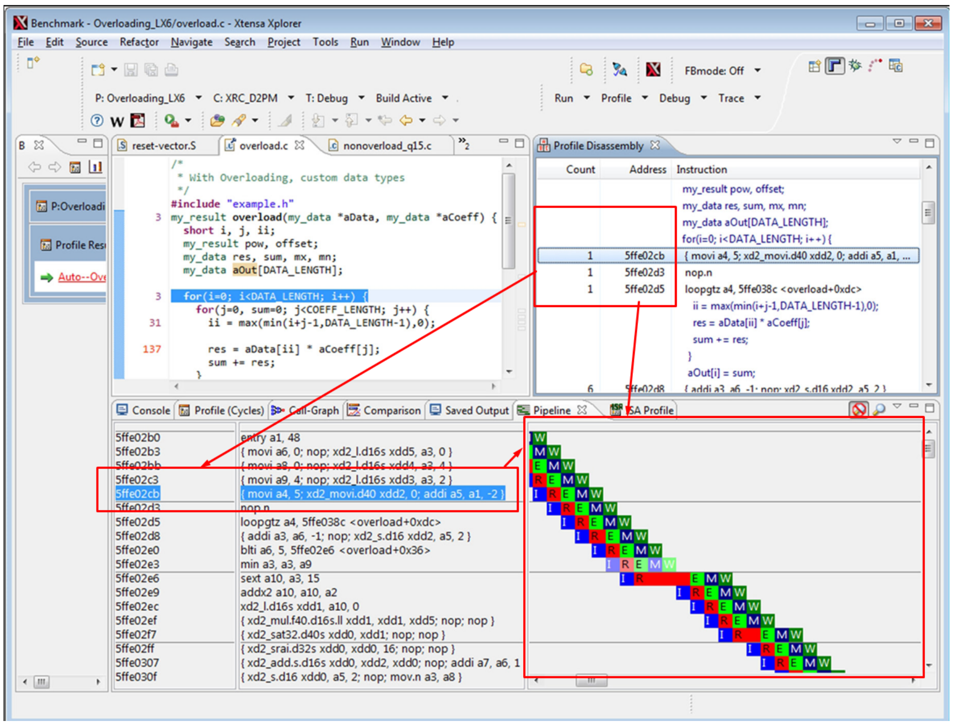
4. Extended Instruction Set

4.1. Profiling

Profiling is a dynamic program analysis that measures, for example, the amount of space (memory) or time used by a program, the frequency and length of function calls, or the use of specific instructions. Profiling also checks individual instruction utilization. In most instances, profiling information is utilized to aid in the optimization of programs and, more specifically, performance engineering.

In order to perform profiling, it is necessary to instrument the program's source code or binary executable with a piece of software known as a profiler (or code profiler). Profilers may employ numerous methodologies, such as event-based, statistical, instrumented, and simulation approaches, among others.

With the assistance of the Xtensa Xplorer tool [20], it is feasible to conduct a realistic simulation of the pipeline. In Figure 3, the pipeline output of a for-loop code kernel is depicted, and the generated assembly for that segment of code. This allows the characterization of the method by employing the Instruction Set Simulator (ISS). Utilizing the profiler, generated data such as the collection of execution cycles of a command, the execution of subroutines, the execution cycles of each subroutine, the performance of the quick access memory (cache), and similar data could be produced. Concurrently, the code in assembly language generated while it is being translated may be inspected, as well as the number of instructions necessary to execute each command. This is feasible due to the translation of the code into another language.


Benchmark - Overloading_LX6/overload.c - Xtensa Xplorer
File Edit Source Refactor Navigate Search Project Tools Run Window Help
P: Overloading_LX6 C: XRC_D2PM T: Debug Build Active
FBmode: Off
Run Profile Debug Trace
reset-vector.S overload.c nonoverload_q15.c
Profile Disassembly
Console Profile (Cycles) Call-Graph Comparison Saved Output Pipeline ISA Profile

```
    /*
     * With Overloading, custom data types
     */
    #include "example.h"
3   my_result overload(my_data *aData, my_data *aCoeff) {
      short i, j, ii;
      my_result pow, offset;
      my_data res, sum, mx, mn;
      my_data aOut[DATA_LENGTH];

3     for(i=0; i<DATA_LENGTH; i++) {
        for(j=0, sum=0; j<COEFF_LENGTH; j++) {
31        ii = max(min(i+j-1,DATA_LENGTH-1),0);

137       res = aData[ii] * aCoeff[j];
          sum += res;
        }
```

```
Count   Address    Instruction
                   my_result pow, offset;
                   my_data res, sum, mx, mn;
                   my_data aOut[DATA_LENGTH];
                   for(i=0; i<DATA_LENGTH; i++) {
1       5ffe02cb   { movi a4, 5; xd2_movi.d40 xdd2, 0; addi a5, a1, ...
1       5ffe02d3   nop.n
1       5ffe02d5   loopgtz a4, 5ffe038c <overload+0xdc>
                     ii = max(min(i+j-1,DATA_LENGTH-1),0);
                     res = aData[ii] * aCoeff[j];
                     sum += res;
                    }
                    aOut[i] = sum;
6       5ffe02d8   { addi a3, a6, -1; nop; xd2_s.d16 xdd2, a5, 2 }
```

```
5ffe02b0   entry a1, 48
5ffe02b3   { movi a6, 0; nop; xd2_l.d16s xdd5, a3, 0 }
5ffe02bb   { movi a8, 0; nop; xd2_l.d16s xdd4, a3, 4 }
5ffe02c3   { movi a9, 4; nop; xd2_l.d16s xdd3, a3, 2 }
5ffe02cb   { movi a4, 5; xd2_movi.d40 xdd2, 0; addi a5, a1, -2 }
5ffe02d3   nop.n
5ffe02d5   loopgtz a4, 5ffe038c <overload+0xdc>
5ffe02d8   { addi a3, a6, -1; nop; xd2_s.d16 xdd2, a5, 2 }
5ffe02e0   blti a6, 5, 5ffe02e6 <overload+0x36>
5ffe02e3   min a3, a3, a9
5ffe02e6   sext a10, a3, 15
5ffe02e9   addx2 a10, a10, a2
5ffe02ec   xd2_l.d16s xdd1, a10, 0
5ffe02ef   { xd2_mul.f40.d16s.ll xdd1, xdd1, xdd5; nop; nop }
5ffe02f7   { xd2_sat32.d40s xdd0, xdd1; nop; nop }
5ffe02ff   { xd2_srai.d32s xdd0, xdd0, 16; nop; nop }
5ffe0307   { xd2_add.s.d16s xdd0, xdd2, xdd0; nop; addi a7, a6, 1
5ffe030f   { xd2_s.d16 xdd0, a5, 2; nop; mov.n a3, a8 }
```

Figure 3. The pipeline viewer facilitates comprehension of instruction delays and latency problems [20].

4.2. Tensilica Instruction Extension—TIE

This study employed the TIE language to extend the processor's fundamental ISA in order to speed up the algorithm described above. TIE's syntax resembles that of C/C++. In the Cadence-supplied TIE language reference handbook, there are other different approaches for achieving the same outcomes; nevertheless, the purpose of this article is to present a basic and uncomplicated method that even novice designers may exploit. As an alternate and more straightforward technique to produce the new instruction, this research focuses on the production of new instructions that integrate the functionality of opcodes, operands, and other statements of kernel code instructions. Only a few lines are required to add a custom operation instruction to the Xtensa CPU, as shown in Listing 1.

The operation listed below defines an instruction with two parameters in the argument list and performs a sum operation on those parameters. The first parameter, a, is an operand that reads and writes, inout statement, the preconfigured Xtensa processor address register file called AR. The second parameter, b, is an AR register file-reading operand, in statement. The Xtensa compiler is responsible for adding this new instruction to a library file so the software can use it.

Listing 1. A simple instruction in TIE language.

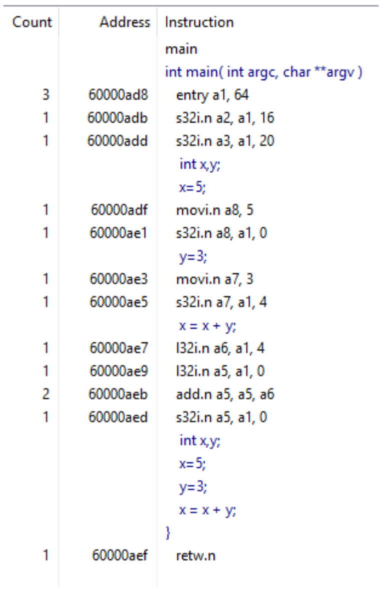
```
operation ADDACC {inout AR a, in~AR b} {}
{
    assign a = a + b;
}
```

With this new operation *ADDACC*, the following C code in Listing 2, which performs an addition between the two integer variables x and y, regarding the new operation *ADDACC*, may be refactored to use this new instruction rather than the traditional sum function operand +.

Listing 2. Example code.

```
int main( int argc, char **argv )
{
    int x,y;
    x=5;
    y=3;
    x = x + y;
}
```

The preceding code has two integer variables (x, y), performs a sum, and assigns the result to variable x . The profiling tool is launched from Xtensa Xplorer, in which one of the most crucial pieces of information provided is the produced Assembly code. Figure 4 showcases the generated code as produced via the Xtensa Xplorer's profiler.



```
Count    Address   Instruction
                   main
                   int main( int argc, char **argv )
    3   60000ad8     entry a1, 64
    1   60000adb     s32i.n a2, a1, 16
    1   60000add     s32i.n a3, a1, 20
                      int x,y;
                      x=5;
    1   60000adf     movi.n a8, 5
    1   60000ae1     s32i.n a8, a1, 0
                      y=3;
    1   60000ae3     movi.n a7, 3
    1   60000ae5     s32i.n a7, a1, 4
                      x = x + y;
    1   60000ae7     l32i.n a6, a1, 4
    1   60000ae9     l32i.n a5, a1, 0
    2   60000aeb     add.n a5, a5, a6
    1   60000aed     s32i.n a5, a1, 0
                      int x,y;
                      x=5;
                      y=3;
                      x = x + y;
                   }
    1   60000aef     retw.n
```

Figure 4. Xtensa Xplorer generates assembly code from the given profiler.

As stated previously, the TIE compiler will produce a library file which will be loaded into the source code. By modifying the basic source code and importing the required library (xtensa/tie/TieFile.h), the *ADDACC* operation is now accessible as a new instruction. The revised source code will appear as follows in Listing 3.

Listing 3. Example code with TIE.

```
#include <xtensa/tie/CustomTie.h>

int main( int argc, char **argv )
{
    int x,y;
    x=5;
    y=3;
    ADDACC(x,y);
}
```

5. Acceleration Process & Results

Using Xtensa Xplorer tools, the aformentioned Bioinformatics algorithms were investigated. The initial parameters for all subsequent experiments follow:

- The IDE allocates 24 GB of RAM for each simulation.
- ExtHomfam dataset was utilized as the input file [23].

The first phase is the analysis and characterization of the algorithm. Using the laboratory's infrastructure (ParICT_CENG), several tests were executed on each of the aforementioned algorithms in order to identify the potential kernels. With the aid of the Xtensa profiler, all of the data obtained throughout the process of profiling using the laboratory's infrastructure was analyzed and cross-checked, ultimately leading to defining three kernels that were most common among the three algorithms.

Initially, special purpose registers were constructed for the local variables of the kernels under examination, reducing the time but also the number of execution cycles required to manage the related segments.

The first kernel, cond_add (conditional add), is responsible for checking the value of a register and, if necessary, increasing it by one. Thus, all *for* and *while* loops with a known iteration count were refactored to utilize the aforementioned new instruction. The aforementioned modification affects the algorithm itself, as the variable *i*, which was before used in the loop, is now a register to which the instruction has direct access. Thus, the number of instruction fetches required to execute a loop has lowered.

The second kernel, array_data, was designed to enable the retrieval, assignment, and modification of array values. Since the aforementioned methods deal with DNA and RNA data, array structures are frequently employed. Although TIE language offers the possibility to generate an array in a single instruction, there are typically no static data to utilize this feature. Therefore, the objective of this kernel was to support the existing actions and work in conjunction with the first kernel. An extended version of this kernel was explicitly designed for Kalign and is described in the following subsection.

The processes as described above were utilized for the algorithms described in Section 2. Table 1 displays the three identified kernels and their contribution to each of the aforementioned algorithms.

Table 1. Participation of the kernel in every algorithm.

Algorithm/Kernels	*Kalign*	*BLAST*	*Clustal*
cond_add	✓	✓	✓
array_data	partially	✓	✓
array_data_ex	✓	–	–

5.1. Extended Array_Data Kernel

It was discovered that a function is being utilized sixty percent of the time during algorithm execution. The function into consideration is void update_gaps (int old_len, int *gis, int new_len, int *newgaps), and as shown in Listing 4, it has two dynamic arrays as input, gis and newgaps, where the former is a table which maintains the distance of the gaps, while the later is used to assign the new gaps on the sequence. The parameters of old_len are used to initiate the first loop, while new_len is a parameter that is never used. This effectively implies that its execution is always dependent on the input. Additionally, the function under examination is responsible for filling in sequence gaps in order to achieve alignment. Finally, the aforementioned process is accomplished through a layered iteration structure in which the number of iterations relies on the current value of position *I* in one of two dynamic arrays.

Listing 4. Initial Kalign function.

```
void update_gaps(int old_len,int*gis,int new_len,int *newgaps)
{
        unsigned int i,j;
        int add = 0;
        int rel_pos = 0;
        for (i = 0; i <= old_len;i++){
                add = 0;
                // CORE 1
                for (j = rel_pos;j <= rel_pos + gis[i];j++){
                        if (newgaps[j] != 0){
                                add += newgaps[j];
                        }
                }
                rel_pos += gis[i]+1;
                gis[i] += add;
        }
}
```

The primary aim was to decrease the number of instruction execution cycles for the portion of the function responsible for creating the final output. Consequently, the read_data kernel was deployed; however, the results were not even close to what was anticipated. The result was an enhanced version of the kernel for this algorithm, as shown in the Listing 5. For each of the parameters used in the operations, a specific state register variable was created, and these variables can be accessed from the C code using the following two functions, *RUR_stateVariableName* and *WUR_stateVariableName*, to read and set the value respectively.

The first section initializes the specific purpose registers precisely as it did with the function's primary local variables. Then, the newly built TIE instructions and repetition structures, along with the processing portion, are employed to identify the gaps in each sequence. Notably, the *innerForLoop* command accepts the memory address of the *gis* dynamic table as an input. At this point, a reasonable question may be regarding how this command understands which table location it needs to access to process the data. The answer lies in the special purpose registers and the capability offered by the Xtensa Xplorer software, utilizing the TIE programming language, in which the user may access a memory location using another register, *out VAddr, in MemDataIn32*. With the *out VAddr* and *in MemDataIn32* statements, the user may access the memory location and the data present at that place, respectively. The fully refactored function, taking advantage of the extended array_data kernel, is shown in Listing 6.

Listing 5. Extended array_data kernel.

```
operation core1 {in AR var1} {inout add}
{
        assign add = var1!=0? add + var1: add;
}

operation outerForLoop {out AR result}
{inout loopi, in~loopiInit,out add,out loopj,in rel_pos}
{
        assign result = loopi <= loopiInit;
        assign loopi = loopi+1;
        assign add=0;
        assign loopj=rel_pos;
}

operation innerForLoop {out AR result, in~AR *var1}
{out VAddr, in~MemDataIn32,inout loopj, in~loopi,
                        inout acc1,in rel_pos}
{
        assign VAddr = (var1+((loopi -1)*4));
        assign acc1 = MemDataIn32;
        assign result = loopj <= rel_pos +acc1 ;
        assign loopj = loopj + 1;
}

operation updateStates {inout AR var1} {inout rel_pos, in~add}
{
        assign rel_pos = rel_pos +var1 +1;
        assign var1 = var1+add;
}
```

Listing 6. Refactored Kalign function using extended read_data kernel.

```
void update_gaps(int old_len,int*gis,int new_len,int *newgaps)
{
        WUR_rel_pos(0);
        WUR_loopi(0);
        WUR_loopj(0);
        WUR_loopiInit(old_len);
        while(outerForLoop()){

                // CORE 1
                while(innerForLoop(gis)){
                        core1(newgaps[RUR_loopj() -1]);
                }
                updateStates(gis[RUR_loopi() -1]);
        }
}
```

The subsection that follows contains the outcomes of the aforementioned modification and a summary of the overall acceleration of all tested algorithms.

5.2. Results

During the testing in Kalign, it was discovered that regarding the files greater than 100 KB, the speedup is also apparent during the testing, where the hemopexin file (293 KB) was employed, and the final speedup result was 30.29 percent. Acceleration refers to the reduction in the number of needed execution cycles and the overall execution time of the algorithm, with the entire adjustments resulting in a 43.49 percent time acceleration.

Without employing the newly created instructions, the original number of execution cycles was 105,940,161,745, whereas the number of execution cycles with the developed code is 73,845,557,474, as depicted in Figure 5.

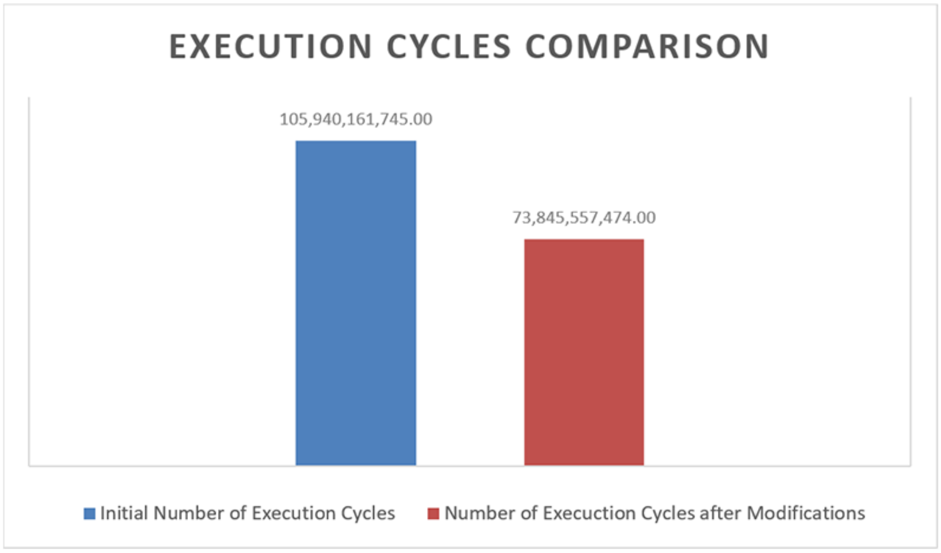


Figure 5. Kalign execution cycles acceleration.

Without the newly created commands, the method described in Listing 4 needed 123,861 s to execute, but with the newly constructed TIE commands, the execution time is now 69,992 s, as shown in Figure 6. On the basis of the above information, a result that may be extracted is that the acceleration gained is not constant but somewhat varies depending on the size of the input file. This is because, as stated previously, the *update_gaps* method accepts arrays of dynamically sized numbers as parameters. According to the conducted experiments, the more the input file size, the greater the acceleration that could be achieved. During the testing, it was also noted that when the algorithm had processed 80 to 82 percent of the sequences, its speed dropped significantly, even while employing TIE instructions.

Table 2 summarizes the acceleration results on the algorithms that have been tested. Kalign exhibited the highest acceleration overall, demonstrating a reduction in Instruction Fetches (IF) and Execution Time (ET) with values of 30.29 percent and 43.49 percent, respectively. Blast accelerated by 12.35 percent in IF and 16.2 percent in ET, whereas Clustal was accelerated by 14.2 percent in IF and 17.9 percent in ET.

Table 2. Acceleration results.

Algorithm/Kernels	*Kalign*	*BLAST*	*Clustal*
Instruction Fetches	30.29%	12.35%	14.2%
Execution time	43.49%	16.2%	17.9%

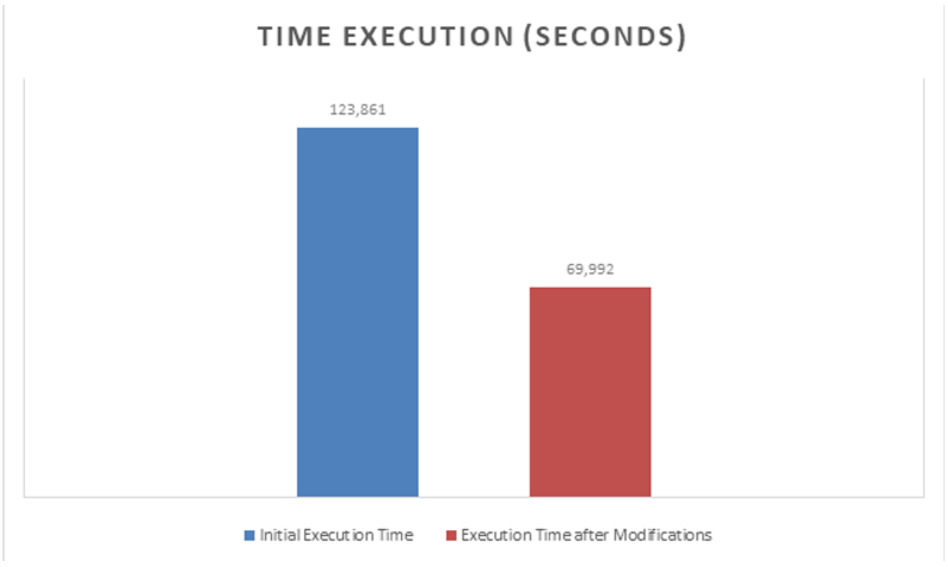


Figure 6. Kalign time execution acceleration.

6. Discussion and Conclusions

In this work, the capability of a reconfigurable RISC processor to incorporate an extended Instruction Set dedicated to the efficient implementation of BioInformatics algorithms, in terms of performance, is investigated. The typical acceleration techniques, reported in other works, are based either on the development of a custom hardware IP dedicated to one application or the selection of architecture offering massive parallel processing. In the first case, the limitation of the IP core to one application increases costs as a result of the additional hardware (area requirements) introduced to the system. The second case reflects the modern MPSoC solutions, including multi-core CPUs, many-core GPUs, and FPGA parallel processing. Both approaches offer reasonable solutions and report high performance, presenting, however, increased costs and demanding design complexity that affects time-to-market. Additionally, the characterization of its approach is capable only on a post-synthesis level, meaning that the additional hardware should be integrated and then characterized.

The proposed approach is based on a reconfigurable RISC processor, namely the Cadence Xtensa processor. The benefit of this scalable processing core is its flexible management of the ISA, allowing easy extension of the Instruction Set, either via micro-programming (using the TIE language), or the access of IP cores as accelerator peripherals. The first approach is especially suitable for rapid prototyping of a custom processor based on Xtensa since there is no need to alter the pre-characterized core. The result is correct-by-construct, and its performance may be accurately reported by the web-based reconfiguration tool offered by Cadence.

This work explored the design flow that is based on the TIE language to accelerate well-known Bioinformatic algorithms. Thus, the Xtensa Xplorer, a comprehensive IDE that includes the tools required to extend a RISC Instruction Set, was utilized. The profiling tool that comes with Xtensa Xplorer was exploited to monitor code usage, repetition, memory calls, cache misses, etc., which assisted in determining subroutines and code segments that could be transformed to single new instructions. This, as expected, minimized the instruction fetches and efficiently exploited streams of data from memory. Consequently, a few lines of TIE commands (Xtensa capabilities) were sufficient to add new instructions to the ISS and restructure the algorithms, which has decreased execution cycles by lowering the number of instruction executions and minimizing the memory fetches. This work highlights a design flow for efficiently developing custom eXtended Instruction Sets for Biomedical Informatics algorithms, without the need to design complex custom hardware and increase the cost, area, and weight of the proposed solution. Specifically, this work was developed for wearable devices requiring low-power dissipation, high autonomy, and low cost. The proposed solution offered the aforementioned requirements without affecting the overall system, tweaking its operation for biomedical informatics applications, and exploiting the reconfigurability features offered by the Xtensa processor.

The results are promising since they reported that even the worst acceleration time is equal to 16.2% compared to the reference implementation. The best acceleration time is equal to 43.49% compared to the reference implementation and presents the best savings in instruction fetches. The approach of treating every frequently used part of code as one critical code kernel allows the selection of the appropriate kernels to become new Instructions for the Extended Instruction Set. The native compiler of the Xtensa processor easily maps the code to the new Instructions and exploits the benefits of micro-programming via the TIE language. The proposed approach proved to be efficient for all tested algorithms.

In the future, this work will serve as the reference approach to evaluate more reconfigurable CPU platforms in order to compile a repository of benchmarks, with the procedure described above serving as the basis for this analysis. Future iterations of this project will entail the development of hardware accelerators in order to create a hybrid architecture with a new extended Instruction Set and hardware components to achieve higher margins of acceleration. The main goal in future work will be the minimization of design complexity based on the reconfigurability features explored in this work, limiting the development of custom hardware accelerators to the least needed.

Author Contributions: This work was developed by the authors as follows: Conceptualization, A.G. and A.K.; Data curation, A.G. and V.T.; Funding acquisition, A.K.; Investigation, A.G. and V.T.; Methodology, A.G. and V.T.; Project administration, A.K.; Resources, A.G., V.T. and A.K.; Software, A.G. and V.T.; Supervision, A.K.; Validation, A.G. and V.T.; Visualization, A.G. and V.T.; Writing—original draft, A.G., V.T. and A.K.; Writing—review and editing, A.G., V.T. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work has received funding by the project "ParICT_CENG: Enhancing ICT research infrastructure in Central Greece to enable processing of Big data from sensor stream, multimedia content, and complex mathematical modeling and simulations" (MIS 5047244) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014–2020) and co-financed by Greece and the European Union (European Regional Development Fund).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IP	Intellectual Property
VLSI	Very Large Scale Integration
RISC	Reduced Instruction Set Computer
MSA	Multiple Sequence Alignments
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
TIE	Tensilica Instruction Extension
IDE	Integrated Development Environment
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
IF	Instruction Fetch
ET	Execution Time
EDA	Electronic Design Automation
CASP	Critical Assesment of protein Structure Prediction

DL	Deep Learning
Cryo-EM	Cryo-Electron

References

1. French Multicentre Experience of Implantable Insulin Pumps—ScienceDirect. Available online: https://www.sciencedirect.com/science/article/abs/pii/S0140673694914621 (accessed on 15 July 2022).
2. Yang, G. *Body Sensor Networks*; Springer: London, UK, 2006.
3. Reddy, B.; Fields, R. Multiple sequence alignment algorithms in bioinformatics. In *Smart Trends in Computing and Communications*; Zhang, Y.-D., Senjyu, T., So-In, C., Joshi, A., Eds.; Lecture Notes in Networks and Systems; Springer: Singapore, 2022; pp. 89–98. [CrossRef]
4. Reddy, B.; Fields, R. Multiple Anchor Staged Alignment Algorithm—Sensitive (MASAA—S). In Proceedings of the 2020 3rd International Conference on Information and Computer Technologies (ICICT), San Jose, CA, USA, 9–12 March 2020; pp. 361–365. [CrossRef]
5. Pakhrin, S.C.; Shrestha, B.; Adhikari, B.; KC, D.B. Deep Learning-Based Advances in Protein Structure Prediction. *Int. J. Mol. Sci.* **2021**, *22*, 5553. [CrossRef]
6. Notredame, C. Recent Evolutions of Multiple Sequence Alignment Algorithms. *PLoS Comput. Biol.* **2007**, *3*, e123. [CrossRef] [PubMed]
7. Chatzou, M.; Magis, C.; Chang, J.-M.; Kemena, C.; Bussotti, G.; Erb, I.; Notredame, C. Multiple Sequence Alignment Modeling: Methods and Applications. *Brief. Bioinform.* **2016**, *17*, 1009–1023. [CrossRef]
8. Kemena, C.; Notredame, C. Upcoming Challenges for Multiple Sequence Alignment Methods in the High-Throughput Era. *Bioinformatics* **2009**, *25*, 2455–2465. [CrossRef] [PubMed]
9. Iantorno, S.; Gori, K.; Goldman, N.; Gil, M.; Dessimoz, C. Who watches the watchmen? An appraisal of benchmarks for multiple sequence alignment. In *Multiple Sequence Alignment Methods*; Russell, D.J., Ed.; Methods in Molecular Biology; Humana Press: Totowa, NJ, USA, 2014; pp. 59–73. [CrossRef]
10. Sievers, F.; Higgins, D.G. Clustal Omega for Making Accurate Alignments of Many Protein Sequences. *Protein Sci.* **2018**, *27*, 135–145. [CrossRef] [PubMed]
11. Thompson, J.D.; Higgins, D.G.; Gibson, T.J. Clustal W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Res.* **1994**, *22*, 4673–4680. [CrossRef] [PubMed]
12. Thompson, J.D.; Gibson, T.J.; Plewniak, F.; Jeanmougin, F.; Higgins, D.G. The Clustal_X Windows Interface: Flexible Strategies for Multiple Sequence Alignment Aided by Quality Analysis Tools. *Nucleic Acids Res.* **1997**, *25*, 4876–4882. [CrossRef] [PubMed]
13. Pearson, W.R. BLAST and FASTA similarity searching for multiple sequence alignment. In *Multiple Sequence Alignment Methods*; Russell, D.J., Ed.; Methods in Molecular Biology; Humana Press: Totowa, NJ, USA, 2014; pp. 75–101. [CrossRef]
14. Myers, G. A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming. *J. ACM* **1970**, *46*, 395–415 . [CrossRef]
15. Lassmann, T. Kalign 3: Multiple Sequence Alignment of Large Datasets. *Bioinformatics* **2020**, *36*, 1928–1929. [CrossRef] [PubMed]
16. CAST. 32-Bit BA2X Processors. Available online: https://www.cast-inc.com/processors/32-bit (accessed on 28 June 2022).
17. CAST. BA20 | PipelineZero 32-Bit Embedded Processor IP Core. Available online: https://www.cast-inc.com/processors/32-bit/ba20 (accessed on 28 June 2022).
18. IEEE SA—IEEE 754-2019. IEEE Standards Association. Available online: https://standards.ieee.org/ieee/754/6210/ (accessed on 16 July 2022).
19. Tensilica Processor IP. Available online: https://www.cadence.com/ko_KR/home/tools/ip/tensilica-ip.html (accessed on 5 August 2022).
20. Tensilica Software Development Toolkit (SDK). Available online: https://ip.cadence.com/uploads/103/SWdev-pdf (accessed on 28 June 2022).
21. Technologies. Available online: https://www.cadence.com/ko_KR/home/tools/ip/tensilica-ip/technologies.html (accessed on 5 August 2022).
22. Robinson, N. Ten Reasons to Optimize a Processor. Available online: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ip/tensilica-ip/tip-wp-10reasons-customize-final.pdf (accessed on 5 August 2022).
23. Gudys, A. *ExtHomFam Benchmark*; Harvard Dataverse: Cambridge, MA, USA, 2016. [CrossRef]