BioBlaze: Multi-Core SIMD ASIP for DNA Sequence Alignment

Nuno Neves, Nuno Sebastião, André Patrício, David Matos, Pedro Tomás, Paulo Flores and Nuno Roma

IST / INESC-ID
Rua Alves Redol, 9, 1000-029 Lisboa - Portugal

***Abstract*—A new Application-Specific Instruction-set Processor (ASIP) architecture for biological sequences alignment is proposed in this manuscript. This architecture achieves high processing throughputs by exploiting both fine and coarse-grained parallelism. The former is achieved by extending the Instruction Set Architecture (ISA) of a synthesizable processor to include multiple specialized SIMD instructions that implement vector-vector and vector-scalar arithmetic, logic, load/store and control operations. Coarse-grained parallelism is achieved by using multiple cores to cooperatively align multiple sequences in a shared memory architecture, comprising proper hardware-specific synchronization mechanisms. To ease the programming, a compilation framework based on an adaptation of the GCC back-end was also implemented. The proposed system was prototyped and evaluated on a Xilinx Virtex-7 FPGA, achieving a 200MHz working frequency. A sequential and a state-of-the-art SIMD implementations of the Smith-Waterman algorithm were programmed in both the proposed ASIP and an Intel Core i7 processor. When comparing the achieved speedups, it was observed that the proposed ISA achieves a 40x speedup, which contrasts with the 11x speedup provided by SSE2 in the Intel Core i7 processor. The scalability of the multi-core system was also evaluated and proved to scale almost linearly with the number of cores.**

I. Introduction

Bioinformatics applications represent one class of algorithms with particularly high performance and efficiency requirements. Among those, protein and Deoxyribonucleic acid (DNA) sequence alignment algorithms, whose optimal solutions are usually obtained by using Dynamic Programming (DP) methods, tend to present a large runtime when executed in current General Purpose Processors (GPPs).

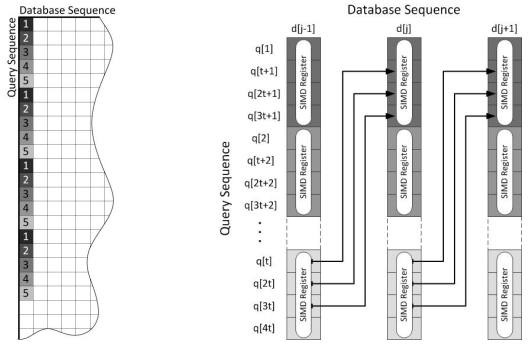
The Smith-Waterman (SW) algorithm [1], characterized by an $\mathcal{O}(nm)$ time complexity, is a widely established DP algorithm to obtain the local alignment between a query sequence (q) and a database sequence (d), of sizes m and n respectively. It operates in two distinct phases: it starts by filling a score matrix H, followed by a traceback phase over this matrix. The matrix is filled by using an *affine* gap penalty model [2] (see eq. 1), where α and β represent the cost of gap opening and extension, and $Sbc(q[i], d[j])$ denotes the substitution score value obtained by aligning character $q[i]$ against character $d[j]$. The initial conditions are given by $H(i,0) = H(0,j) = E(i,0) = F(0,j) = 0$.

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects HELIX (Ref. PTDC/EEA-ELC/113999/2009), TAGS (Ref. PTDC/EIA-EIA/112283/2009), THREadS (Ref. PTDC/EEA-ELC/117329/2010) and project PEst-OE/EEI/LA0021/2011.

$$
\begin{aligned}
H(i,j) &= \max \begin{cases} H(i-1,j-1) + Sbc(q[i], d[j]), \\ E(i,j), \\ F(i,j), \\ 0 \end{cases} \qquad (1) \\
F(i,j) &= \max\Big\{ H(i-1,j) - \alpha \ ; \ F(i-1,j) - \beta \Big\} \\
E(i,j) &= \max\Big\{ H(i,j-1) - \alpha \ ; \ E(i,j-1) - \beta \Big\}
\end{aligned}
$$

To speedup the alignment, several hardware accelerators were developed [4], but such solutions lack the adaptability and flexibility provided by GPPs and by Application Specific Instruction-set Processors (ASIPs). On the other hand, various SIMD parallelization approaches on GPPs have also been presented [3]. One of the fastest was proposed by M. Farrar [3], who adopted a pre-computed query profile for the entire database sequence, and optimized the processing scheme by using a *striped* access pattern, where the computations are carried out in several separate stripes that cover different parts of the query sequence (see Fig. 1(a)).

The query is divided into p equal length segments of size $t = \lfloor (m + p - 1)/p \rfloor$, where p denotes the number of vector elements that can be accommodated in a SIMD register (each SIMD vector element is assigned to one distinct segment). Each matrix column, corresponding to a database symbol $d[j]$, is processed in t iterations, where each iteration simultaneously processes p query symbols, separated by $t-1$ lines in the score matrix. Fig. 1(b) illustrates the data dependencies between the last segment and the first segment of the next column of the



(a) Striped processing scheme. (b) Data dependencies.

Fig. 1. Farrar's SIMD implementation of the SW algorithm [3]. The first five SIMD iterations were numbered and represented with different gray levels (for simplicity, only 4 data elements are shown in each SIMD register).

score matrix H. With this processing pattern, it is possible to move the conditional statements related to the commitment of the vertical dependencies to an independent *lazy loop*, executed outside the inner loop, where they have to be considered only once, before starting the processing of the next database symbol, thus reducing the impact of the vertical dependencies.

To further accelerate this class of algorithms, a new ASIP, specifically adapted for biological sequence alignment algorithms, is proposed in this manuscript. The attained processing throughput is achieved as a result of a two-fold improvement in the original architecture: *i)* extension of the processor ISA to support multiple specialized SIMD vector instructions, to extensively exploit fine-grained parallelism; and *ii)* implementation of an extensive multi-core computational structure, composed by multiple instantiations of the designed ASIP, in order to efficiently exploit coarse-grained parallelism.

II. Dedicated SIMD Instruction Set for Biological Sequences Alignment

Due to its higher performance and prevalence in most widely established bioinformatics applications, Farrar's SIMD implementation [3] will be herein adopted as the elected case-study. By analyzing the algorithm's pseudo-code (see Fig.2(a)), it is clear that the adoption of vector arithmetic instructions will potentially accelerate this algorithm. These instructions should not only speedup the operations between vectors, but they might also facilitate the several operations between vectors and scalars, which are particularly useful when subtracting the gap penalties. The shifting of the F and H vectors can also be efficiently implemented with a vector element shift instruction. Since all these new instructions will be dealing with SIMD vectors, it is also advantageous to include new memory access instructions, to handle vector-sized variables.

A special attention should be also devoted to the definition of optimized control instructions. This effort is justified by the significant predominance of loop procedures in these DP algorithms (generally implemented with conditional branch instructions), as well as the severe penalties that these control instructions generally impose on deep pipeline architectures. In particular, a new specialized branch instruction to simultaneously assert a branch condition in all vector elements, without any additional processing, will significantly increase the achieved performance (e.g.: execution of the *lazy loop*).

Although not limited at this respect, the proposed instruction set and the corresponding data-path (see section III) provides support for the same register and vector-element sizes as Intel SSE2 (used by Farrar [3]), i.e. 128-bit registers, with 8 or 16 elements. Furthermore, the vector elements of each register can take any size, starting from 8 bits to the limit imposed by the register size. However, to obtain a fair comparison with Farrar's [3] SSE2 implementation, only 128-bit registers with 8-bit vector elements will be considered in this particular case-study. On the other hand, any non-SIMD instruction will only operate over the least-significant part of the register (corresponding to a scalar word). Such solution confines the critical-path to the non-SIMD data-path, thus making it independent of the extended SIMD register size.

The proposed ISA extension defines 48 specialized SIMD instructions for arithmetic, logic, memory access and control operations. By comparing Farrar's [3] algorithm implementation based on Intel SSE2 ISA (see Fig.2(b)) with an implementation based on the proposed ISA (see Fig.2(c)), it can be observed that an immediate gain, regarding the number of instructions, is promptly achieved with the proposed ISA, with more visible advantages in the *lazy-loop*. The major contributor to this reduction is the new set of vectorized control instructions, that significantly reduce the control overhead.

It is also important to note that another significant advantage of the proposed ASIP arises from the fact that it adopts a strict Reduced Instruction Set Computer (RISC) paradigm based on a shallow pipeline structure, contrasting with Intel's Complex Instruction Set Computer (CISC) model. As a consequence, the observed difference in the number of instantiated instructions, together with the RISC single-cycle per instruction ratio (instead of CISC multiple-cycle per instruction), will significantly augment the processing gain, as it will be demonstrated in section V.

	Pseudo-code [3]	Intel SSE2	ASIP
Inner Loop	vH = vH + vProfile[i][j]	paddusb (r8,rcx,1),xmm1	**lv** r3, r9, r23 **addvv** r3, r8, r3
		psubusb xmm9,xmm1	
	vMax = max(vMax, vH)	pmaxub xmm1,xmm3	**maxvv** r19, r19, r3
	load vE[j]	movdqa (rax,rcx,1),xmm2	**lv** r5, r7, r11
	vH = max(vH, vE[j])	pmaxub xmm2,xmm1	**maxvv** r4, r3, r5
	vH = max(vH, vF)	pmaxub xmm0,xmm1	**maxvv** r4, r4, r6
	vHStore[j] = vH	movdqa xmm1,(r11,rcx,1)	sv r4, r7, r10
	vH = vH - gapOpen	psubusb xmm7,xmm1	**rsubvs** r4, r24, r4
	vE[j] = vE[j] - gapExtnd	psubusb xmm5,xmm2	**rsubvs** r5, r12, r5
	vE[j] = max(vE[j], vH)	pmaxub xmm1,xmm2	**maxvv** r5, r5, r4
	vF = vF - gapExtnd	psubusb xmm5,xmm0	**rsubvs** r3, r12, r6
	vF = max(vF, vH)	pmaxub xmm1,xmm0	**maxvv** r6, r3, r4
	store vE[j]	movdqa xmm2,(rax,rcx,1)	addik r11, r0, 13824 **sv** r5, r7, r11
	vH = vHLoad[j]	movdqa (r9,rcx,1),xmm1	**lv** r8, r7, r22
		add $0x10,rcx	addik r7, r7, 16
		cmp r13,rcx	xori r18, r7, 4
		jne 7be <start+0x14a>	bnei r18, -72
Lazy Loop		movslq ecx,r10	
		shl $0x4,r10	
		lea (rdi,r10,1),r8	
	vH = max(vH, vF)	pmaxub xmm0,xmm1	**maxvv** r3, r4, r5
	vHStore[j] = vH	movdqa xmm1,(r11,r10,1)	**sv** r3, r6, r10
	load vE[j]	movdqa (r8),xmm2	**lv** r4, r6, r11
	vH = vH - gapOpen	psubusb xmm4,xmm1	**rsubvs** r3, r7, r3
	vE[j] = max(vE[j], vH)	pmaxub xmm2,xmm1	**maxvv** r4, r4, r3
	store vE[j]	movdqa xmm1,(r8)	**sv** r4, r6, r9
	vF = vF - gapExtnd	psubusb xmm8,xmm0	**rsubvs** r5, r8, r5
			addik r18, r0, 63
	if (++j ≥ segLen)	add $0x1,ecx	addik r6, r6, 16
		cmp ecx,edx	cmp r18, r6, r18
		jg 87b <start+0x207>	bgei r18, 12
	vF = vF ≪ 8	pslldq $0x1,xmm0	**sllv** r5, r5
	j=0	mov $0x0,ecx	addk r6, r0, r0
		movslq ecx,r8	
		shl $0x4,r8	
		movdqa (r11,r8,1),xmm1	addik r11, r0, 13936
	vH = vHStore[j]	movdqa xmm1,xmm2	**lv** r4, r6, r10
	vT = vH - gapOpen	psubusb xmm4,xmm2	**rsubvs** r3, r7, r4
		movdqa xmm0,xmm11	
	vT = vF - vT	psubusb xmm2,xmm11	**rsubvv** r3, r3, r5
		movdqa xmm11,xmm2	
		pcmpeqb xmm6,xmm2	
		pmovmskb xmm2,r8d	
		cmp $0xffff,r8d	
		jne 83e <start+0x1ca>	**bgtiv** r3, -68
	(a)	(b)	(c)

Fig. 2. Farrar's SIMD implementation of the SW algorithm [3]: (a) Pseudo-code definition; (b) Intel SSE2 assembly code; (c) Proposed ISA assembly code. Instructions outlined in bold face belong to the specialized ISA. Shaded areas outline the blocks of identical operations that require a different number of instructions to complete in the two implementations. Only the inner and lazy loops are illustrated in this figure.

III. SIMD Processor Architecture

The MB-LITE [5] soft-core was used as the base architecture for the implementation of the proposed ISA, not only due to its simple and portable processing structure, but also because it is a compliant implementation of the well known MicroBlaze ISA. Furthermore, since the GNU Compiler Collection (GCC) already supports the MicroBlaze processor, adding the new instructions' mnemonics and opcodes was easily accomplished by conveniently adapting the corresponding back-end.

The MB-LITE design is highly configurable and is relatively easy to adapt to the proposed ISA. Accordingly, some groups of instructions were left out, including the multiplication and barrel shifter operations, as well as all floating point and special register operations. In fact, the reduced hardware resources that are required by this core were also taken into account, prospecting the bases for a scalable multi-core processing platform to exploit coarse-grained parallelism.

Despite being fully parameterizable, the configuration of the designed SIMD module that was adopted for this specific case-study uses 128-bit registers with 16 8-bit SIMD elements. To support the proposed extension of the ISA, the execution unit had to be modified, by extending its original Arithmetic and Logic Unit (ALU) to include a new SIMD module. As an example, the *addition* and *subtraction* operations require one adder per SIMD vector element, together with some extra multiplexing logic. Since different types of SIMD operations are supported (*vector-vector*, *vector-scalar* and *inner-vector*), the required vector elements have to be selected from the corresponding registers and only then does the execution unit perform all the parallel arithmetic operations. The results are then chosen based on appropriate control signals.

The new *maximum* instruction, particularly useful for this class of algorithms, deserved a special attention. It was based on the already existing *compare* instruction, comprising a subtraction followed by a signal evaluation. Therefore, the same logic can be used to implement these two instructions, requiring only a multiplexer to choose the maximum between the two operands. To avoid increasing the critical-path, the decision logic was moved to the next pipeline stage and to the pipeline forwarding lines. This new *maximum* instruction not only substitutes one *compare* and one *branch* instruction, but it also prevents the pipeline flush (gaining 3 or 4 clock cycles), depending on whether the branch has delay slots or not.

Whenever possible, the same opcode was assigned to the new SIMD instructions as their non-SIMD counterparts, by using unused fields to distinguish them in the processor control unit. With this option, it was possible to re-use most of the original decoding structures, except for a few control signals that had to be generated from such bit-fields.

IV. Multi-Core Processing Platform

Many High Throughput Short Read (HTSR) sequencing applications require the alignment of multiple query sequences to one or more database sequences. This requirement adds a thread-level parallelism to the computation, where multiple cores concurrently align multiple query sequences with one or more database sequences. To allow this parallel computation, a shared memory is used to store the database and query sequences [6]. The computation is controlled by a master core, which manages the sequence alignment queue and the multiple processing elements. To initiate the sequence alignment, the master core needs to communicate a minimal set of data to the target processing core, which consists of the address (in main memory) and the length of the query and database sequences.

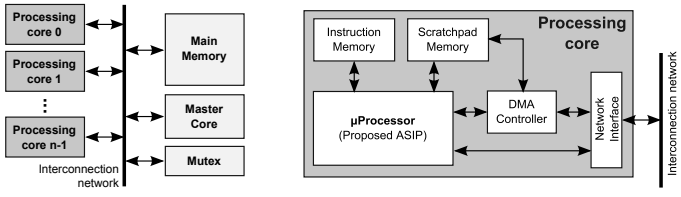
To compute the alignment score for multiple query sequences, the architecture ilustrated in Fig. 3 is now proposed. It is composed of: a memory element, to store both the biological sequences and the alignment scores; a master core, which is responsible for managing the sequence alignment queue; multiple processing cores based on the specialized SIMD ASIP; and a mutex circuit, to handle core synchronizations. All elements are interconnected by an AMBA 3 AHB-Lite compatible shared bus. To reduce the amount of data that is transferred between the master and the processing cores, it was considered the shared memory model studied in [7], specifically developed for this application domain.

V. Experimental Results

To evaluate the proposed multi-core processing framework, a thorough performance analysis of the complete system was conducted, by prototyping it in a Xilinx Virtex 7 FPGA (XC7VX485T). To synthesize the design and perform the place-and-route procedure, the Xilinx ISE 14.4 tool-chain was used. Accurate clock cycle measurements of the required time to execute the biological sequences analysis in the proposed platform was achieved by using Modelsim SE 10.0b.

Table I depicts an evaluation of the resources overhead introduced by the proposed extended ISA and of the attained maximum operating frequencies for both a single-core configuration of the proposed ASIP and for the multi-core system. The extension of the original MB-LITE ISA to support the new instructions led to a frequency decrease of about 27 MHz relative to the original MB-LITE implementation (not shown in the table). This demonstrates that the ISA extension had a reduced impact on the original critical path, which is now limited by the added multiplexing logic that is required to implement the SIMD instructions.

To demonstrate the advantages of the proposed SIMD ISA, the number of clock cycles required to execute a DNA sequence alignment procedure on both the proposed ASIP and on a state-of-the-art superscalar GPP, capable of multiple instruction issue, out-of-order and speculative execution (Intel Core i7 950 processor) was measured. For this test, both the sequential and the Farrar's SIMD versions of the SW algorithm were considered. The sequence alignment code was compiled with GCC 4.6.2, using flags -O2 (sequential case)



(a) General overview of the multi-core architecture.

(b) Schematic of the processing cores.

Fig. 3. Multi-core architecture for biological sequences alignment.

TABLE I. MULTI-CORE SYSTEM HARDWARE SPECIFICATIONS.

	Prop. ASIP	**Multi-core structure**		
	(μProcessor)	2 cores	16 cores	32 cores
Registers	1014 (<1%)	2354 (<1%)	18671 (3%)	37519 (6%)
LUTs	3943 (1%)	8725 (2%)	70430 (23%)	139309 (45%)
LUT-FF pairs	921 (22%)	2118 (23%)	15082 (20%)	30486 (20%)
RAM/FIFO	–	68 (6%)	432 (41%)	848 (82%)
Freq. (MHz)	199.1	181.3	167.6	151.7

TABLE II. CLOCK CYCLES($\times 10^6$) AND OBTAINED SPEEDUP VALUES.

Sequence	**Intel**			**Proposed ASIP**		
Size	Sequential	SSE2	Speedup	Sequential	SIMD	Speedup
128	0.124	0.027	5.90	0.651	0.016	40.69
256	0.242	0.032	7.56	1.302	0.032	40.69
512	0.481	0.053	9.08	2.604	0.068	38.29
1024	0.957	0.094	10.18	5.209	0.135	38.59
2048	1.914	0.181	10.57	10.416	0.270	38.58
4096	3.830	0.339	11.30	20.834	0.541	38.51
8192	7.668	0.675	11.36	41.665	1.081	38.54
16384	15.300	1.374	11.14	83.325	2.163	38.52
Average			9.64			39.05

and -O (SIMD case), corresponding to the most favorable parametrization for each case. On the Intel architecture, cycle-accurate measurements were obtained by using the PAPI library to read the processor performance counters. For the considered benchmark, a DNA data-set was used, which is composed of several database sequences ranging from 128 to 16384 elements and a query sequence of length 64. The database sequences correspond to a random selection of sub-sequences of the *Homo Sapiens* chromosome `Y` genomic contig (`NT_011875.12`), while the query sequence was generated by randomly combining reads from run `ERR004756` of study `ERP000053` (human DNA).

Table II presents the average number of clock cycles required to execute the DNA sequence alignment with the proposed ASIP and the Intel Core i7. As it can be observed in this table, the Intel Core i7 achieves a maximum speedup of 11.36x, while the ASIP, with the proposed ISA, achieves a maximum speedup of 40.69x, i.e., a value about 4 times higher. This result demonstrates that the proposed ISA extension is well tuned for operations commonly adopted in sequence alignment algorithms. Furthermore, it is important to recall that the ASIP SIMD register size was configured to a 128-bit width, for the single purpose of ensuring a fair comparison with the Intel processor, although it may be easily extended in order to increase the ASIP's performance.

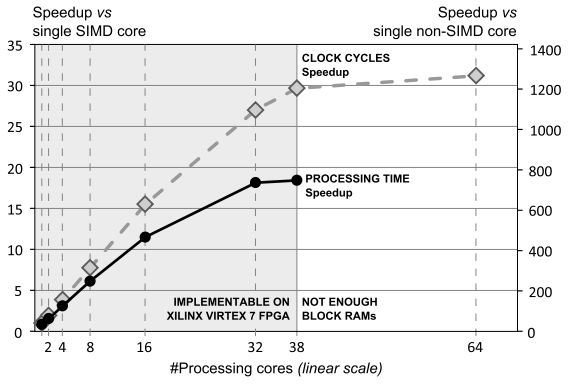


Fig. 4. Variation of the obtained speedup with the number of instantiated ASIP cores in the prototyped multi-core processing structure. The processing time speedup takes into account the decrease of the operating frequency in the multi-core implementations.

To analyze the scalability of the multi-core system, the obtained speedup was measured, when all the processing cores are executing Farrar's SIMD version of the sequence alignment algorithm. Fig. 4 presents the obtained speedup values in what concerns the clock cycles and the total processing time of the proposed multi-core structure. Such speedup values were obtained by using a single ASIP core as the reference. The observed speedup increases almost linearly for configurations up to 16 cores. With additional cores, the contention in the shared bus becomes a limiting factor [7], thus reducing the effectiveness of the extra cores and resulting in a sub-linear speedup increase. Still, when considering the initial non-SIMD sequential implementation as reference, the obtained results demonstrate that an 750x processing time speedup can be obtained with a 32-core parallel SIMD implementation of the proposed ASIP. It should be noticed that, due to the Block-RAM resource requirements of each core, a maximum of 38 processing cores can be instantiated on the prototyping FPGA.

VI. CONCLUSION

A new ASIP architecture, specifically adapted for biological sequence alignment algorithms, was proposed. The presented ASIP is able to achieve high processing throughputs through an optimized architecture that exploits both fine and coarse-grained parallelism. Fine-grained parallelism is achieved by expanding the MicroBlaze ISA to support multiple specialized SIMD instructions, and by conveniently adapting the pipeline architecture of the MB-LITE soft-core. This adaptation provided a speedup of about 40x, when compared to a non-SIMD implementation of the SW algorithm. In contrast, the same SIMD implementation executed in an Intel Core i7 only achieved a speedup of about 11x. Coarse-grained parallelism was also exploited by using a multi-core computational structure composed of multiple ASIPs. A functional prototype on a Xilinx Virtex-7 FPGA device demonstrated that a linear speedup can be achieved with up to 16 processing cores. Furthermore, experimental setups using more cores demonstrated that the proposed system is capable of achieving a cumulative speedup of 750x with 32 cores, despite the observable contention in the interconnection bus.

REFERENCES

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[2] O. Gotoh, "An improved algorithm for matching biological sequences," *J. of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982.

[3] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, p. 156, 2007.

[4] N. Sebastião, N. Roma, and P. Flores, "Integrated hardware architecture for efficient computation of the n-best bio-sequence local alignments in embedded platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 7, pp. 1262–1275, July 2012.

[5] T. Kranenburg and R. van Leuken, "MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 997–1000, March 2010.

[6] F. Sánchez Castaño, A. Ramirez, and M. Valero, "Quantitative analysis of sequence alignment applications on multiprocessor architectures," in *Proc. ACM Conference on Computing Frontiers*. ACM, 2009, pp. 61–70.

[7] N. Roma and P. Magalhães, "System-level prototyping framework for heterogeneous multi-core architecture applied to biological sequence analysis," in *IEEE Int. Symp. on Rapid System Prototyping (RSP'2012)*, Tampere - Finland, Oct. 2012, pp. 156–162.