SHA: A Design for Parallel Architectures?

Antoon Bosselaers, René Govaerts and Joos Vandewalle

Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT
Kardinaal Mercierlaan 94, B-3001 Heverlee, Belgium

antoon.bosselaers@esat.kuleuven.ac.be

25 February 1997

Abstract. To enhance system performance computer architectures tend to incorporate an increasing number of parallel execution units. This paper shows that the new generation of MD4-based customized hash functions (RIPEMD-128, RIPEMD-160, SHA-1) contains much more software parallelism than any of these computer architectures is currently able to provide. It is conjectured that the parallelism found in SHA-1 is a design principle. The critical path of SHA-1 is twice as short as that of its closest contender RIPEMD-160, but realizing it would require a 7-way multiple-issue architecture. It will also be shown that, due to the organization of RIPEMD-160 in two independent lines, it will probably be easier for future architectures to exploit its software parallelism.

Key words. Cryptographic hash functions, instruction-level parallelism, multiple-issue architectures, critical path analysis

1 Introduction

The current trend in computer designs is to incorporate more and more parallel execution units, with the aim of increasing system performance. However, available hardware parallelism only leads to increased software performance, if the executed code contains enough software parallelism to exploit the potential benefits of the multiple-issue architecture.

Cryptographic algorithms are often organized as an iteration of a common sequence of operations, called a round. Typical examples of this technique are iterated block ciphers and customized hash functions based on MD4. In many applications, encryption and/or hashing forms a computational bottleneck, and an increased performance of these basic cryptographic primitives is often directly reflected in an overall improvement of the system performance.

To increase the performance of round-organized cryptographic primitives it suffices to concentrate the optimization effort on the round function, knowing that each gain in the round function is reflected in the overall performance of the primitive multiplied by the number of rounds. Typical values for the number of rounds are between 8 and 32.

This paper confronts one class of cryptographic primitives, namely the customized hash functions based on MD4, with the most popular computer architectures in use today or in the near future. Although only the MD4-like hash functions are considered in the sequel, much of it also applies to other classes of

iterated cryptographic primitives. Our main aim is to investigate the amount of software parallelism in the different members of the MD4 hash family, and the extent to which nowadays RISC and CISC processors are able to exploit this parallelism. This approach differs of the one in [BGV96] in that we now take the hashing algorithms as a starting point, and investigate the amount of inherently available parallelism, while previously we took a particular superscalar processor as starting point, and investigated to which extent an implementation of the hashing algorithms could take advantage of that architecture.

The next section considers the basic requirements a processor has to meet to enable efficient implementations of MD4-like hash functions. Section 3 gives an overview of currently available processor architectures, and lists their, for our purposes, interesting characteristics. Section 4 introduces the notion of a critical path. The available amount of instruction-level parallelism in the MD4-like algorithms is determined in section 5, and confronted with the available hardware of section 3. Finally, section 6 formulates the conclusions.

2 Basic hardware requirements

The customized hash functions based on MD4 include MD4 [Riv92a], MD5 [Riv92b], SHA-1 [FIPS180-1], RIPEMD [RIPE95], RIPEMD-128 and RIPEMD-160 [DBP96]. It are all iterative hash functions using a compression function as their basic building block, the input to which consists of a 128 or 160-bit chaining variable and a 512-bit message block. The output is an update of the chaining variable. Internally, the compression function operates on 32-bit words. The conversion from external bit strings to internal word arrays uses a big-endian convention for SHA-1 and a little-endian convention for all the other hash functions. Depending on the algorithm the compression function consists of 3 to 5, possibly parallel, rounds, each made up of 20 (SHA-1) or 16 (all other) steps. Finally, a feedforward adds the initial value of the chaining variable to the updated value. Every round uses a particular non-linear function, and every step modifies one word of the chaining variable and possibly rotates another. Definitions of the round and step functions can be found in Tables 1 and 2, respectively.

Multiplexer	$(x \wedge y) \vee (\overline{x} \wedge z)$, $(x \wedge z) \vee (y \wedge \overline{z})$
Majority	$(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
Xor	$x \oplus y \oplus z$
Or-Xor (OX)	$(x \vee \overline{z}) \oplus y$, $(x \vee \overline{y}) \oplus z$, $(y \vee \overline{z}) \oplus x$

Table 1. Definition of the Boolean round functions used in MD4-family algorithms.

This short overview allows us to conclude that an implementation of MD4-like hash functions will benefit from a processor that

Algorithm	Step function using Boolean function:	Mux	Maj	Xor	Or-Xor
MD4	$A := (A + f(B,C,D) + X_i + K)^{\lll s}$	1	2	3	
MD5	$A := B + (A + f(B,C,D) + X_i + K)^{\lll s}$	1,2		3	4
SHA-1	from step 17 onwards: $X_i := (X_i \oplus X_{i+2} \oplus X_{i+8} \oplus X_{i+13})^{\lll 1}$ $A := A + B^{\lll 5} + f(C,D,E) + X_i + K$ $C := C^{\lll 30}$	1	3	2,4	
RIPEMD	$A := (A + f(B,C,D) + X_i + K)^{\lll s}$	1	2	3	
RIPEMD-128	$A := (A + f(B,C,D) + X_i + K)^{\lll s}$	2L,3R 4L,1R		1L,4R	3L,2R
RIPEMD-160	$A := E + (A + f(B,C,D) + X_i + K)^{\lll s}$ $C := C^{\lll 10}$	2,4		1L,5R	5L,1R 3

Table 2. Definition of the step function used in MD4-family algorithms. Additions are modulo 2^{32}. Rotating x over s bits to the left is indicated as $x^{\lll s}$. A, B, C, D, E are the words of the chaining variable, K and s are constants, X_i is a message word or a combination thereof, and $f()$ is one of the functions defined in Table 1. The last 4 columns indicate in which rounds these functions are used, and, if different, whether in the left (L) or right (R) parallel line.

1. supports 32-bit operations.
2. can handle both little-endian and big-endian memory addressing.
3. has a rotate instruction, and, in addition to the standard logical instructions `and`, `or`, and `xor`, instructions like `nand`, `nor`, `nxor`, `and-not`, and `or-not`, where the latter two are defined as, respectively, the `and` and `or` of the first operand and the complement of the second. Remark that `xor-not` would be the same as `nxor`.
4. is able to keep all local variables in registers: 16 message words, 5 chaining words, and 2 auxiliary words. The RIPEMD-family, having two parallel lines, requires two copies of the last two items. So in total up to 30 registers are required.
5. supports parallel execution of arithmetic or logical (ALU) operations. This item will be further investigated in the next section.

3 Hardware parallelism

The basic implementation technique, applied by all nowadays processors, to improve CPU performance is pipelining. A pipeline is organized in a number of stages, each of which executes part of a CPU instruction. Multiple instructions can overlap in execution by letting each stage in the pipeline complete a part of a different instruction. Hence, this technique allows different parts of consecutive instructions to be executed in parallel. As a consequence, pipelining increases the CPU instruction throughput. The execution time of each instruction usually slightly increases due to pipeline control overhead, but this is more

than compensated for by the increase in instruction throughput. The net effect is a substantial decrease in the number of clock cycles per instruction, ideally resulting in a speedup equaling the number of pipeline stages.

To enhance performance even further two approaches are available: increase the number of pipeline stages, or use a number of parallel pipelines. The former architecture is called superpipelined and emphasizes temporal parallelism, while the latter relies on spatial parallelism and comes in two flavors: superscalar or very long instruction word (VLIW). The aim of these techniques is to further increase the throughput. A superpipelined architecture achieves this by reducing the clock cycle time, while a superscalar/VLIW architecture tries to issue more than 1 instruction per clock cycle. However, there is a limit to what can be gained in terms of performance. This limit is determined by two factors: a software one and a hardware one. The software factor is the amount of parallelism in the instruction stream, i.e., the amount of data dependencies between the instructions. In the next section the available instruction-level parallelism in an instruction stream will be characterized by the its critical path. The hardware factor is the impact of the increase in the number of pipeline stages or pipelines on the clock cycle time.

In case of a superpipelined architecture limited parallelism in the instruction stream will eventually lead to so-called pipeline stalls due to data dependencies: the execution of an instruction has to be stalled until the data needed to complete it become available. But even in the absence of dependencies superpipelining will eventually run out of steam. The clock cycle time can never be lower than the overhead pipelining incurs on each stage: clock skew and pipeline register overhead [HePa96]. Therefore, increasing the number of pipeline stages beyond a critical point will result in performance degradation rather than performance gain.

Further increase in performance can then only be obtained by either going superscalar or using VLIWs.

- A superscalar processor has dynamic issue capability: a varying number of instructions is issued every clock cycle. The hardware dynamically decides which instructions are simultaneously issued and to which pipelines, based on issue criteria and possible data dependencies.
- A VLIW processor has fixed issue capability: every clock cycle a fixed number of instructions is issued, formatted as one large instruction (hence the name). The software (i.e., the compiler) is completely responsible for creating a package of instructions that can be simultaneously issued. No decisions about multiple issue are dynamically taken by the hardware.

An advantage of a VLIW over a superscalar is that the amount of required hardware can be reduced: choosing the instructions to be issued simultaneously is done at compile-time, and not at run-time. However, the superscalar has two major advantages: its code density is little affected by the available parallelism in the instruction stream, and it can be object-code compatible with a large family of non-parallel processors. The major challenge in the design of a superscalar processor will be to limit the impact on the clock cycle time of issuing and

executing multiple instructions per cycle. This is illustrated by the fact that to date a factor of 1.5 to 2 in clock rate has consistently separated the highest clock rate processors and the most sophisticated multiple-issue processors [HePa96].

A final uniprocessor technique to exploit parallelism inherent in many algorithms is single-instruction, multiple-data (SIMD) processing, a term originally only used in the context of multiprocessor environments [Fly66]. A SIMD instruction performs the same operation in parallel on multiple data elements, packed into a single processor word. Tuned to accelerate multimedia and communications software, these instructions can be found in an increasing number of general-purpose processor architectures. Examples are Intel's MMX [PeWe96], UltraSPARC's VIS [TONH96], and PA-RISC 2.0 architecture's MAX [Lee96]. MMH [HaKr97] is an example of a cryptographic hash function taking advantage of this new technology. Remark that a combination of multiple-issue and SIMD techniques creates in effect a kind of multiple-issue, multiple-data (MIMD) parallelism, also called SIMD-MIMD parallelism [Lee95].

CPUs can be differentiated among based on the type of their internal storage: a stack, an accumulator, or a set of registers. Only the latter class of CPUs will be considered in the sequel, since virtually every processor designed after 1980 uses that architecture, called a (general-purpose) register architecture. A further division of this call can be made based on the way instructions can access memory and on the operands for a typical ALU instruction.

- In a register-memory architecture memory can be accessed as part of any instruction, while in a register-register architecture memory can only be accessed with load and store instructions, for which reason the latter is also called a load-store architecture.
- The maximum number of operands of an ALU instruction is either two or three. A three-operand instruction contains a destination and two source operands, while in a two-operand instruction one of the operands is both a source and a destination for the operation.
- The number of memory operands of an ALU instruction can vary from none to the maximum number of operands (2 or 3).

It turns out that two[1] combinations suffice to classify all the CPUs that will be considered:

class 1 - a tree-operand load-store architecture (no memory operands in ALU instruction): MIPS, Precision Architecture (PA-RISC), PowerPC, SPARC, Alpha.

class 2 - a two-operand register-memory architecture (at most one memory operand in ALU instruction): 80x86 (including Pentium and PentiumPro), 680x0.

Remark that the same division also distinguishes between RISC processors (class 1) and CISC processors (class 2).

[1] Three suffice to classify nearly all existing machines, see [HePa96, Section 2.2]

Table 3 summarizes the characteristics of these architectures with respect to the requirements formulated at the end of the previous section, including the available hardware parallelism for ALU instructions [Sta96, HePa96, Bha96]. The figures are for the most recent processors of each architecture. As far as RISC processors are concerned, these are all 64-bit, although compatibility with their 32-bit predecessors is retained. Since Alpha was designed as a 64-bit device, the support for 32-bit operations is limited. All RISC architectures include support for both little and big-endian addressing, but especially with PA-RISC and Alpha architectures an implementation is not required to implement both addressing modes. An Alpha implementation is not even required to support changing the convention during program execution, but only at boot time [Dig96]. The other RISCs can use either format, selectable in either software or hardware. Some architectures are more than 2-way superscalar, but none can issue more than 2 instructions in parallel of the ALU subset that interests us: add, logical operations, rotate/shift.

Architecture	MIPS IV	PA 2.0	PowerPC	SPARC V9	Alpha EV5	80x86	680x0
Word Size	64	64	64	64	64	32	32
Integer regs	31	31	32	31	31	7	8
Endianness	select.	select.	select.	select.	Little	Little	Big
AND	`and`	`and`, `and-not`	`and`,`nand`, `and-not`	`and`, `and-not`	`and`, `and-not`	`and`	`and`
OR	`or`,`nor`	`or`	`or`,`nor`, `or-not`	`or`, `or-not`	`or`, `or-not`	`or`	`or`
XOR	`xor`	`xor`	`xor`,`nxor`	`xor`,`nxor`	`xor`,`nxor`	`xor`	`xor`
ROT	No	Yes[a]	Yes	No	No	Yes	Yes
ALU pipe s	1[b]/2[c]	2	2	2	2	2	2
32-bit subset	Yes	Yes	Yes	Yes	No[d]	(Yes)	(Yes)
Processor	R4000, R10000	PA-8000	PowerPC 620	Ultra-SPARC	21164	Pentium PPro	68060

[a] The PA-RISC 2.0 instruction `shrpw r1,r2,x,t` shifts the concatenation of `r1` and `r2` to the right over `x` bits, and puts the result in `t`. By taking `r1` = `r2` = `t` it is in effect a rotate.

[b] The R4000 is superpipelined (but not superscalar) and its pipeline clock is twice the external clock frequency, so that 2 instructions can be issued per clock cycle.

[c] The R10000 is superscalar, but not superpipelined.

[d] The Alpha architecture has just 3 32-bit integer operations: add, subtract, multiply. In addition, it has a set of in-register manipulation instructions on 32-bit quantities, such as extract, insert, and mask.

Table 3. Overview of the latest designs of the most popular computer architectures. Only those characteristics are listed that are relevant when implementing MD4-like hash functions on these architectures.

From this table we can conclude that, with respect to the requirements of

section 2, all listed RISC architectures fulfill requirement 4, while all of the first 3 requirements are only met by the PowerPC, and to a varying degree by the other RISCs. The most serious problem for a number of RISCs is certainly the absence of a rotate instruction, while CISCs are severely restricted by their small register set. In section 5 it is investigated whether such a two-way superscalar architecture suffices to exploit all the parallelism available in the MD4-like algorithms, using the analysis restricted to MD5 in [Tou95] as a starting point.

So far only superpipelined and/or superscalar processors have been considered. Nowadays most multiple-issue processors are superscalar, but VLIW is experiencing a comeback in popularity. An example of the latter is the recently introduced 32-bit VLIW processor TM-1 of Philips Trimedia [SRD96]. Up to 5 operations can be packed into a single VLIW-instruction and executed in a single clock cycle. Although intended for multimedia processing, its ability to execute 5 ALU operations in parallel creates new opportunities for fast implementations of existing and for the design of new cryptographic algorithms [Cla97].

4 Critical path length

To determine the amount of available instruction-level parallelism in the MD4-like hash functions, a critical path analysis is applied. To that end the algorithms are represented as a so-called activity-on-edge network, which is a directed graph with weighted edges.

Geometrically a graph G is defined as a set $V(G)$ of vertices v_i interconnected by a set $E(G)$ of edges e_i. In a directed graph or digraph an edge e_i is a directed pair $\langle v_i, v_j \rangle$ and represented by an arrow from the tail v_i to the head v_j. A directed path from v_p to v_q is a sequence of vertices $v_p, v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_q$ such that $\langle v_p, v_{i_1} \rangle, \langle v_{i_1}, v_{i_2} \rangle, \ldots, \langle v_{i_n}, v_q \rangle$ are edges in $E(G)$.

A network is a graph with weighted edges, i.e., to each edge e a weight $w(e)$ is assigned. In an activity-on-edge network (AOE-network) tasks to be performed are represented by directed edges. The vertices in the network represent events, signaling the completion of certain activities. Activities represented by edges leaving a vertex cannot be started until the event at that vertex has occurred. An event occurs only when all activities entering it have been completed. The weight $w(e)$ assigned to an edge e represents the time required to complete the activity associated with e.

The length of a path is then defined as $\sum_e w(e)$, where e runs over all edges on the path. It is the time it takes to complete the task represented by the path. Assuming the activities in an AOE network can be carried out in parallel, the minimum time to complete the overall task is the length of the longest path from the start vertex to the termination vertex. Such a path is called a critical path.

The evaluation of an arithmetic expression can be modeled as an AOE network. The start vertex corresponds to the availability of the input data, the activities represented by the edges correspond to the arithmetic operations constituting the expression, and the termination vertex corresponds to the result of the expression. The weight of an edge represents the time it takes to complete

the corresponding arithmetic operation. Maximum performance in evaluating an arithmetic expression will therefore be obtained by making its critical path as short as possible, using, as much as possible, parallel execution of individual arithmetic operations. However, we must take into account that eventually the evaluation of the expression will take place on a multiple-issue architecture of the kind described in the previous section, i.e., all parallel execution units are pipelined, and all advance at the same rate. Unless out-of-order execution is supported, operations executed in parallel all deliver their result at the same moment, and therefore not faster than the time of the slowest operation. For this reason the critical path length will be expressed in terms of required pipeline stages, rather than in clock cycles. A measure similar to critical path length is depth, as used in the analysis of parallel algorithms [Ble96].

5 CPL analysis of the MD4-family

The critical path length (CPL) of the MD4-like compression functions is mainly determined by the CPL of the individual rounds: the CPL of the feedforward is at most 2. The CPL of each round is equal to the sum of the CPLs of each step, so that the CPL of the compression function is easily derived from the CPL of a step. Each step updates one of the chaining words, and this updated word is then input to the next step. It is this basic dependency between steps that will determine their CPL. An inspection of two consecutive steps of every MD4-family member (see Appendix A) learns us that, except for SHA-1, the chaining word updated in one step is input to the Boolean function of the next step. The chaining word updated in that step only becomes available after adding in the Boolean result, rotating the resulting sum, and, in case of MD5 and RIPEMD-160, adding in another chaining word. SHA-1, in contrast, inputs the updated chaining word to a simple rotate, and the next chaining word becomes available after only 1 more addition. These lower bounds on a step's CPL are summarized in Table 4.

Algorithm	Operations in CP	min. CPL
MD4, RIPEMD, RIPEMD-128	$f(), +, \lll$	3
MD5, RIPEMD-160	$f(), +, \lll, +$	4
SHA-1	$+, \lll$	2

Table 4. Lower bound on the CPL of a step for each of the MD4-family members, assuming that it takes a minimum of 1 stage to deliver the result of a Boolean function.

SHA-1 uses exactly the same kind and amount of operations as MD5 and RIPEMD-160 to update a chaining variable: 1 application of a Boolean function, 4 additions, and a rotate. However, the lower bound on a step's CPL is only half

that of MD5 and RIPEMD-160. This is due to the fundamentally different way SHA-1's step function is organized compared to all the others:

1. The rotate is not applied to a sum of intermediate results, but to an individual chaining variable.
2. None of the arguments of the Boolean function are, except for a rotate, updated in the previous step, but in the step before that.

This in itself might be a coincidence, but it turns out that the lower bound is also the actual CPL of each SHA-1 step, while this is not the case for any of the other hash functions, as will be shown in the sequel. This seeming coincidence might well be a design principle.

For the other hash functions the Boolean function is part of the critical path. This results in an increase of the CPL if the result cannot be delivered within the 1 stage assumed for the lower bound. This is, e.g., the case for the multiplexer $(x \wedge y) \vee (\overline{x} \wedge z)$ used in all MD4-like hash functions. It would seem that from the moment x becomes available, and only using `and`, `or`, and `xor`, it takes three more stages to deliver the multiplexer result [Tou95]. However, using the mathematically equivalent expression $((y \oplus z) \wedge x) \oplus z$ [McC94, NMVR95], it only takes two more stages. Since this is still 1 more than the value assumed in the lower bound, this multiplexer lengthens the CPL of all steps using it by 1, except for SHA-1, where the Boolean function isn't necessarily part of the critical path. Remark that, as far as CPL is concerned, it doesn't always pay off to use the equivalent multiplexer expression. Consider the alternative multiplexer $(x \wedge z) \vee (y \wedge \overline{z})$ used in MD5, RIPEMD-128, and RIPEMD-160, and where the critical path runs through y. Without rewriting it only takes 2 stages to deliver the result from the point y becomes available, but using the equivalent expression $((x \oplus y) \wedge z) \oplus y$ the CPL increases to 3.

The results of this CPL analysis for the MD4-family of hash functions is given in Table 5. The analysis is done using both 3-operand and 2-operand instructions. With the exception of the first and third round steps of SHA-1, the shortest possible critical path is the same for both operand formats. However, for the same CPL a realization on a 2-operand architecture requires more parallel execution units than on a 3-operand one. This information can be derived from the last 4 columns, where for both formats the required number of parallel units and their efficiency is given. The efficiency is defined as

$$\frac{\text{number of instructions in a step}}{\text{CPL} \times \text{number of execution units}},$$

and is a measure of the average usage of the parallel execution units. The closer the value is to 1, the higher the degree of occupancy of the parallel units.

Table 5 also shows that if 3-operand instructions are used the shortest possible critical path of all SHA-1 steps is equal to the lower bound of Table 4: 2 stages. This is illustrated for the most involved case in Figure 1: the step function of the third round using the majority function. As a result the CPL of SHA-1's compression function is the shortest of all the MD4-like hash functions, as shown

Algorithm	Step	CPL		Regs		3-op pipe		2-op pipe	
	function	min.	real	state	aux.	#	eff.	#	eff.
MD4	Mux	3	4	16+4	1	2	0.75	2	0.88
	Maj		4		2	2	1.00	3	0.83
	Xor		3		1	2	1.00	3	0.78
MD5	Mux1	4	5	16+4	1	2	0.80	2	0.90
	Mux2		5		2	2	0.90	3	0.73
	Xor		4		1	2	0.88	2	1.00
	Or-Xor		5		1	2	0.80	2	0.90
SHA-1	Mux[a]	2	2/3[b]	16+5	5	7	0.93	6	0.89
	Xor		2		4	6	1.00	7	1.00
	Maj		2/3[b]		5	7	1.00	6	1.00
	Xor		2		4	6	1.00	7	1.00
RIPEMD	Mux	3	4	16+8	2	4	0.81	4	0.94
	Maj		4		4	4	0.94	5	0.95
	Xor		3		2	4	1.00	5	0.93
RIPEMD-128	Xor/Mux2	3	3/4[c]	16+8	3	4	0.92	5	0.89
	Mux1/OX		4		2	4	0.88	4	1.00
RIPEMD-160	Xor/OX2	4	4	16+10	2	4	1.00	5	0.90
	Mux1/Mux2		5		3	4	0.95	5	0.88
	OX1/OX1		5		2	4	0.90	4	1.00

[a] The message expansion only starts at step 17. Therefore, the first 16 steps have only an efficiency of 0.64 and 0.61, respectively.
[b] 3-operand/2-operand figure
[c] Xor/Mux2 figure

Table 5. Results of the critical path analysis on the MD4-like steps. Listed are for each step the lower bound and the actual value of the CPL, the required number of state (message+chaining) and auxiliary registers, and the required number and efficiency of parallel ALU pipelines, both for 3-operand and 2-operand instruction formats. The figures for the last two rounds of RIPEMD-128 and RIPEMD-160 are not listed, since they are the same as those for the first two rounds.

Algorithm	CPL (stages)	#Regs	Pipes	
			#	Eff.
MD4	176	22	2	0.91
MD5	304	22	2	0.84
SHA-1	160	26	7	0.85
RIPEMD	176	28	4	0.91
RIPEMD-128	240	27	4	0.90
RIPEMD-160	368	29	4	0.96

Table 6. The shortest possible CPLs of the MD4-like compression functions (without feedforward), and the required resources in terms of registers and parallel execution units. A 3-operand instruction format is assumed.

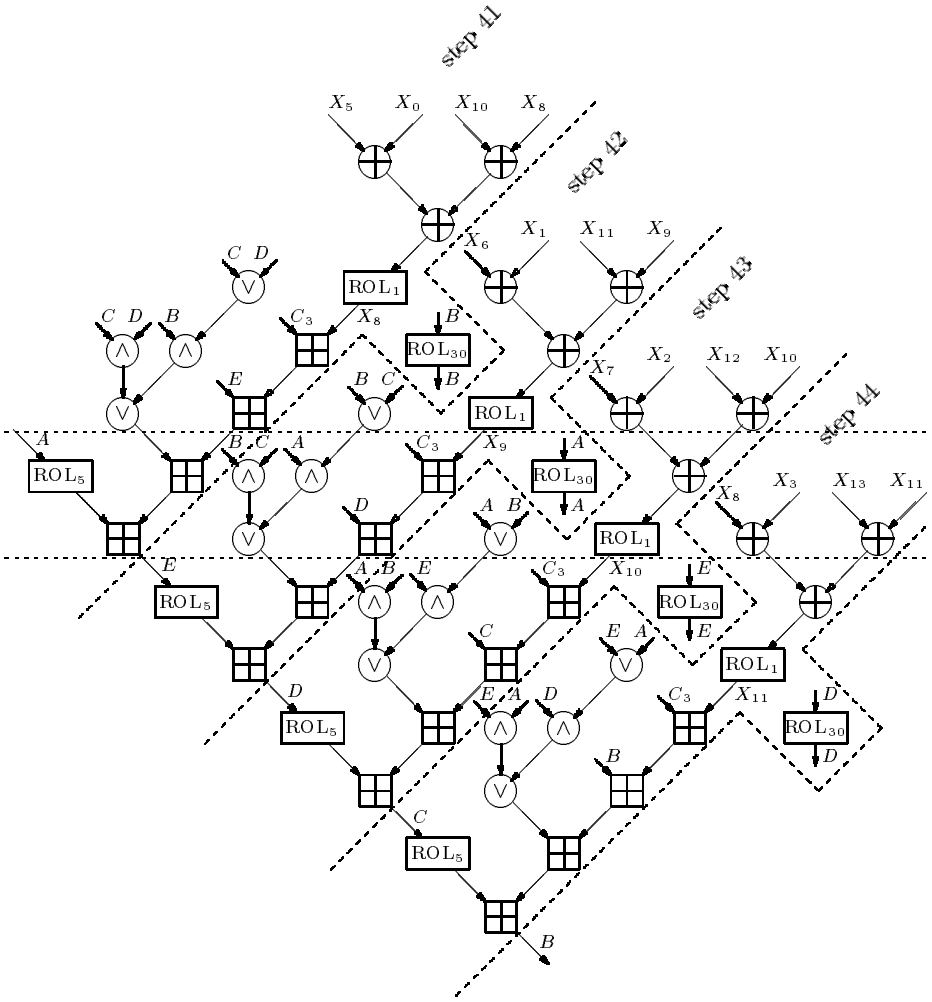

Fig. 1. The first 4 steps of SHA-1's round 3 on a 7-way multiple-issue architecture using a 3-operand instruction format. Instructions executed in parallel are drawn on the same horizontal level, while instructions belonging to the same step are shown between diagonal dotted lines. A CPL of 2 stages is realized by executing 7 instructions of up to 4 different steps in parallel, as shown between the 2 horizontal dotted lines.

in Table 6. To realize a CPL of 2 in round 1 and 3 of SHA-1, two parallel rotates of the same variable are required, see Figure 1. However, the rotate instruction is a unary operation, and hence its 2-operand format has equal source and destination, making a parallel execution on the same variable impossible. Comparing the requirements of Table 6 with the resources of Table 3 shows that current su-

perscalar architectures are only able to exploit all the available instruction-level parallelism of MD4 and MD5, two algorithms that as collision-resistant hash functions can no longer be considered as secure [Dob96a, Dob96b, Rob96].

The natural question to ask is: how realistic are the prospects for a general-purpose processor issuing one day 7 ALU instructions in parallel? Issuing many instructions per clock is difficult due to an increasingly complex issuing logic having a negative impact on the clock cycle time. Therefore, a high issuing rate will only pay off if the parallel execution units are kept sufficiently busy, so that the increase in cycle time will be more than compensated for by an enhanced throughput. The CPL analysis of SHA-1 shows that some algorithms certainly contain enough instruction-level parallelism to sustain such an increased issuing rate, but it is doubtful whether this will be the case for an average instruction sequence.

The RIPEMD-family has, in contrast to SHA-1, two completely independent lines, leaving room for exploiting parallelism on a different level: the use of a multiprocessor system where the multiple-issue capability of each processor is limited, rather than a uniprocessor system with a single, very sophisticated processor capable of offering all the required parallelism on its own. In this respect [HePa96, Section 4.10] states that 'to date, computer architects do not know how to design processors that can effectively exploit instruction-level parallelism in a multiprocessor configuration.' The capability of placing two fully configured processors on a single die, which should be possible around the turn of the century, might result in a new type of architecture allowing processors to be more tightly coupled than before, and at the same time allowing them to achieve very high performance individually. Therefore, exploiting the instruction-level parallelism of the RIPEMD-family in the near future seems much more likely, since each of the independent lines only requires a two-way superscalar architecture, which is already a standard feature of most processors today.

Algorithms with more instruction-level parallelism than the hardware they are executed on can provide, will inevitably see their CPL increase. This is illustrated by means of the first step of MD4's round 2. Using a 3-operand instruction format two parallel units suffice two exploit all available instruction-level parallelism, as illustrated in the left diagram of Figure 2. Remark that the efficiency is 100%. Using a 2-operand instruction format will increase the number of instructions, as operations of the form $A \leftarrow B$ op C will require two instructions: $A \leftarrow B$ and $A \leftarrow A$ op C. Due to the already 100% efficiency of the 3-operand instruction stream, 3 parallel units are now required to realize the same CPL of 4. Therefore, an implementation using only 2 parallel units will inevitably have a longer critical path. This is illustrated in the right diagram of the same figure, showing an increase in CPL of 1 stage. The left diagram is expected to be found on e.g., a PowerPC 604 [SDC94] or a PA 7100LC [BKQW95], while the right diagram resembles the situation on a Pentium processor, except that a Pentium cannot execute a rotate over more than 1 bit in parallel with any other instruction, resulting in a further increase of the CPL.

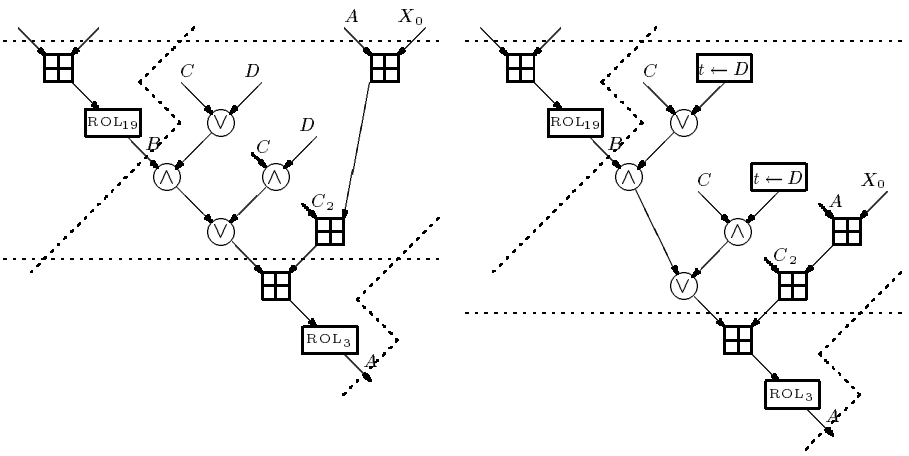

Fig. 2. The first step of MD4's round 2 implemented on a two-way superscalar architecture. Instructions executed in parallel are drawn on the same horizontal level, while instructions belonging to the same step are shown between diagonal dotted lines. The left diagram uses 3-operand instructions, and shows both instruction pipes already occupied for 100%. The use of 2-operand instructions increases the number of instructions by 2, either requiring an additional instruction pipe for the same CPL, or resulting in an increased CPL on the same architecture, as shown on the right.

6 Conclusion

The new generation of customized hash functions based on MD4 (RIPEMD-128, RIPEMD-160, SHA-1) contains more instruction-level parallelism than current general-purpose computer architectures are able to provide. The critical path of SHA-1 is shorter than any of the other MD4-like hash functions, but exploiting it would require a 7-way multiple-issue architecture. Exploiting the instruction-level parallelism of the RIPEMD-family in the near future seems more likely, due to their organization in two independent lines, each of which only requires a 2-way superscalar architecture. Opening up new perspectives is the recent introduction of a new 5-way VLIW processor, primarily intended for multimedia processing.

References

[BKQW95] M. Bass, P. Knebel, D.W. Quint, W.L. Walker, "The PA 7100LC microprocessor: a case study of IC design decisions in a competitive environment," *HP Journal,* Vol. 46, No. 2, April 1995, pp. 12–22.

[Bha96] D.P. Bhandarkar, *Alpha implementations and architecture,* Digital Press, Boston, MA, 1996.

[Ble96] G.E. Blelloch, "Programming parallel algorithms," *Communications of the ACM*, Vol. 39, No. 3, 1996, pp. 85–97.

[BGV96] A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 298–312.

[Cla97] C. Clapp, "Optimizing a fast stream cipher for VLIW, SIMD, and superscalar processors," *Fast Software Encryption, LNCS*, E. Biham, Ed., Springer-Verlag, 1997, to appear.

[Dig96] *Alpha architecture handbook, Version 3,* Digital Equipment Corp., Maynard, MA, 1996.

[Dob96a] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.

[Dob96b] H. Dobbertin, "The status of MD5 after a recent attack," *CryptoBytes,* Vol. 2, No. 2, 1996, pp. 1–6.

[DBP96] H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71–82. Final version available via ftp at `ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/ripemd/`.

[FIPS180-1] FIPS 180-1, "Secure hash standard," US Department of Commerce/NIST, Washington D.C., April 1995.

[Fly66] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, No. 12, 1966, pp. 1901–1909.

[HaKr97] S. Halevi and H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates," *Fast Software Encryption, LNCS*, E. Biham, Ed., Springer-Verlag, 1997, to appear.

[HePa96] J.L. Hennessy and D.A. Patterson, *Computer architecture: a quantitative approach, 2nd edition,* Morgan Kaufmann Publishers, San Francisco, 1996.

[Lee95] R. Lee, "Accelerating multimedia with enhanced microprocessors," *IEEE Micro,* Vol. 15, No. 2, April 1995, pp. 22–32.

[Lee96] R. Lee, "Subword parallelism with MAX-2," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 51–59.

[NMVR95] D. Naccache, D. M'Raïhi, S. Vaudenay, D. Raphaeli, "Can DSA be improved? Complexity trade-offs with the Digital Signature Standard," *Advances in Cryptology, Proceedings Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 77–85.

[McC94] K.S. McCurley, "A fast portable implementation of the secure hash algorithm, III," Technical Report SAND93-2591, Sandia National Laboratories, 1994.

[PeWe96] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 42–50.

[RIPE95] RIPE, *"Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)," LNCS 1007*, A. Bosselaers and B. Preneel, Eds., Springer-Verlag, 1995.

[Riv92a] R.L. Rivest, "The MD4 message-digest algorithm," Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.

[Riv92b] R.L. Rivest, "The MD5 message-digest algorithm," Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.

[Rob96] M. Robshaw, "On recent results for MD2, MD4 and MD5," Bulletin No. 4, RSA Laboratories, November 1996.

[SRD96] G.A. Slavenburg, S. Rathnam, H. Dijkstra, "The Trimedia TM-1 PCI VLIW media processor," *Hot Chips VIII Conference*, Stanford University, Palo Alto, CA, 1996.

[SDC94] S.P. Song, M. Denman, J. Chang, "The PowerPc 604 RISC microprocessor," *IEEE Micro,* Vol. 14, No. 5, October 1994, pp. 8–17.

[Sta96] P.H. Stakem, *A practitioner's guide to RISC microprocessor architecture,* John Wiley & Sons, New York, 1996.

[Tou95] J. Touch, "Performance analysis of MD5," *Proceedings of ACM SIGCOMM'95, Comp. Comm. Review,* Vol. 25, No. 4, 1995, pp. 77-86.

[TONH96] M. Tremblay, J.M. O'Connor, V. Narayanan, L. He, "VIS speeds new media processing," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 10–20.

A Dependencies between consecutive steps

This appendix lists, for each member of the MD4-family, the first two steps of an arbitrary round.

- MD4
  $A := (A + f(B,C,D) + X_i + K)^{\lll s_1}$
  $D := (D + f(A,B,C) + X_j + K)^{\lll s_2}$
- MD5
  $A := B + (A + f(B,C,D) + X_i + K)^{\lll s_1}$
  $D := A + (D + f(A,B,C) + X_j + K)^{\lll s_2}$
- SHA-1
  $X_i := (X_i \oplus X_{i+2} \oplus X_{i+8} \oplus X_{i+13})^{\lll 1}$
  $E := E + A^{\lll 5} + f(B,C,D) + X_i + K$
  $B := B^{\lll 30}$
  $X_{i+1} := (X_{i+1} \oplus X_{i+3} \oplus X_{i+9} \oplus X_{i+14})^{\lll 1}$
  $D := D + E^{\lll 5} + f(A,B,C) + X_{i+1} + K$
  $A := A^{\lll 30}$
- RIPEMD
  $A := (A + f(B,C,D) + X_i + K)^{\lll s_1}$
  $D := (D + f(A,B,C) + X_j + K)^{\lll s_2}$
- RIPEMD-128
  $A := (A + f(B,C,D) + X_i + K)^{\lll s_1}$
  $D := (D + f(A,B,C) + X_j + K)^{\lll s_2}$
- RIPEMD-160
  $A := E + (A + f(B,C,D) + X_i + K)^{\lll s_1}$
  $C := C^{\lll 10}$
  $E := D + (E + f(A,B,C) + X_j + K)^{\lll s_2}$
  $B := B^{\lll 10}$