Performance Impact of Addressing Modes on Encryption Algorithms

A. Murat Fiskiran and Ruby B. Lee
Department of Electrical Engineering
Princeton University
{fiskiran,rblee}@princeton.edu

Abstract

Encryption algorithms commonly use table lookups to perform substitution, which is a confusion primitive. The use of table lookups in this way is especially common in the more recent encryption algorithms, such as the AES finalists like MARS and Twofish, and the AES winner, Rijndael. Workload characterization studies indicate that these algorithms spend a significant fraction of their execution cycles on performing these table lookups, more specifically on effective address calculations.

This study considers the five AES finalists (MARS, RC6, Rijndael, Serpent and Twofish) and studies the effect of different addressing modes that can be used to calculate the effective addresses during the table lookups. We report our findings for four different addressing modes and on varying width EPIC processors. The results indicate that speedups exceeding 2x can be obtained when fast addressing modes are used.

1. Introduction

Diffusion and *confusion* are two cryptographic functions that are necessary to obscure the plaintext during encryption. Diffusion achieves this through mixing and reordering of data, such as in shifts or rotates. Confusion, on the other hand, relies on *substitution*, which means replacement of chunks of data by some other data, such as in a *table lookup.*

Table lookups in encryption algorithms have been used in this way for a long time. DES, for instance, is a very widely used algorithm whose security depends exclusively on table lookups. In DES, these tables are known as the S-boxes. More recent algorithms also rely heavily on table lookups for security. Of the five algorithms that were the finalists in the Advanced Encryption Standard (AES) effort (MARS [1], RC6 [2], Rijndael [3], Serpent [4], and Twofish [5]), all except RC6 used table lookups.

For some of these algorithms, table lookups are used for optimization purposes – beyond achieving confusion. Rijndael, for instance, is the AES winner and has a structure that involves various operations such as rotations and matrix multiplication following a series of table lookups. However, the algorithms is designed in such a way that the operations following the table lookups can be migrated into the table lookups by pre-modifying these tables, so that finally, the entire algorithm becomes only a series of table lookups. Similar optimizations are also possible for Twofish. For these algorithms, performing the table lookups efficiently has a direct and significant impact on the performance, since the algorithms are indeed a sequence of table lookups.

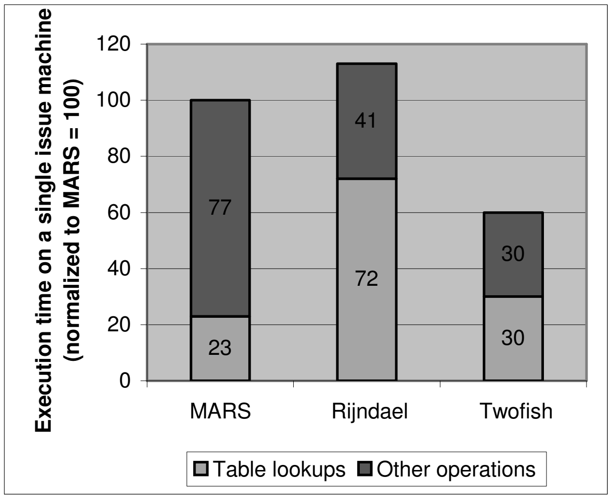

Figure 1. Distribution of execution cycles between table lookups and all other operations.

2. Table lookups in AES finalists

Problems in the encryption algorithms with many table lookups are twofold.

First, each table lookup involves operations other than loading of the data from the memory. Typically, this involves the *effective address* calculation. The *index* (that is the number specifying which entry of the table will be accessed) needs to be scaled and then added to the start address of the table (the base address) to get the effective address. The scaling is necessary whenever each entry of the table holds data that is larger than a single byte. This is quite often the case, since the most commonly used tables have 256 entries and each entry is 4 bytes. The addition of the scaled index value to the base address is usually a part of the `load` instruction. This is the *indexed* addressing mode. The scaling, however, usually requires a separate instruction (unless the scaling is also a part of the `load` instruction, in which case we have the *scaled-indexed* addressing mode).

Secondly, the index is not always readily available to begin with. Consider the optimized implementation of either Rijndael or Twofish. These algorithms use tables that have 256 entries and each entry is 4 bytes. The index, therefore, is 8 bits long ($2^8 = 256$ entries). However, the 8-bit index can be in any one of the four bytes of a 32-bit integer register. This means, to obtain and isolate the index, additional instructions are required. For instance, to obtain the 8-bit index which is in the third byte of a 32-bit integer register, the register first needs to be shifted to the right by 24 bits, and then anded with 0xFF to isolate the index. This requires at least two additional instructions, unless a special instruction such as `extract` is available.

2. Table lookups in AES finalists

This section will focus on how different addressing modes can be used to speed up the most common table lookups in the encryption algorithms. As an example, consider a table lookup from Rijndael or Twofish. These algorithms use tables with 256 entries, and each entry is 4 bytes. The indices into these tables are 8 bits (bytes), and these bytes are extracted from any one of the four byte locations in a 32-bit integer register. Assume that we perform a lookup using the third byte of the register `Ra` as the index, and that the table base address is in `Rb`. The result will be written to `Rd`.

3.1. Indexed addressing

Performing this table lookup using the indexed addressing mode will require four instructions.

```
1. shr Rc,Ra,24          # shift right
2. and Rc,Rc,0xFF        # bitwise and
3. shl Rc,Rc,2           # shift left
4. load Rd,Rc(Rb)        # load indexed
```

The first instruction performs a right shift to right-align the index bits and the second instruction is necessary to clear the upper order bits. The third instruction scales the index, and the fourth instruction performs the load. The indexed addressing mode calculates the effective address by adding the scaled index to the base address during the load operation.

A simple optimization, which is often missed by compilers is possible as follows.

```
1. shr Rc,Ra,22
2. and Rc,Rc,0x3FC
3. load Rd,Rc(Rb)
```

3.2. Scaled-indexed addressing

This addressing mode is found in some existing Instruction Set Architectures (ISAs) such as the PA-RISC 2.0 [6]. The index scaling is also migrated into the `load` instruction, and this permits a single instruction saving per table lookup.

```
1. shr Rc,Ra,24
2. and Rc,Rc,0xFF
3. load.4 Rd,Rc(Rb)
```

The '4' after the `load` indicates that a scaling for four bytes (that is two bits) will be applied to the index.

3.3. Scaled-indexed addressing + `extract`

An `extract` instruction picks an arbitrary continuous bit-field from the source register and writes it right-aligned to the target register, while clearing the remaining bits of the target register. `Extract` instructions are also found existing microprocessor ISAs, such as in IA-64 [7]. Using the `extract` instruction together with the scaled-indexed addressing mode permits a table lookup to be done in two instructions.

```
1. extract Rc,Ra,3b
2. load.4 Rd,Rc(Rb)
```

The '`3b`' in the `extract` instruction indicates that the third byte of `Ra` is extracted into `Rc`.

3.4. `Load.extract.scale (load.ex.sc)`

The final case we consider is a hypothetical instruction which combines the index extraction, index scaling and the memory access into one. This will be used to show the potential speedups if it was possible to eliminate all the overhead of the effective address calculations. This instruction will be called the `load.ex.sc.`

```
1. load.3b.4 Rd,Ra(Rb)
```

The '`3b`' in the '`ex`' field indicates that the third byte of `Ra` will be used as the index, and the '`4`' in the '`sc`' field indicates that the index will be scaled by `4`. The use of the `load.ex.sc` in this way completely eliminates any overhead associated with table lookups (Figure 2).

A similar, but more restrictive approach has previously been described in [8]. In that study, it was noted that if the tables were aligned to 1kB memory blocks, the effective address can very simply be calculated by a concatenation of the shifted index bits and the base address. Implementation of `load.ex.sc` is more complicated, however it is applicable to table sizes other than 1kB.

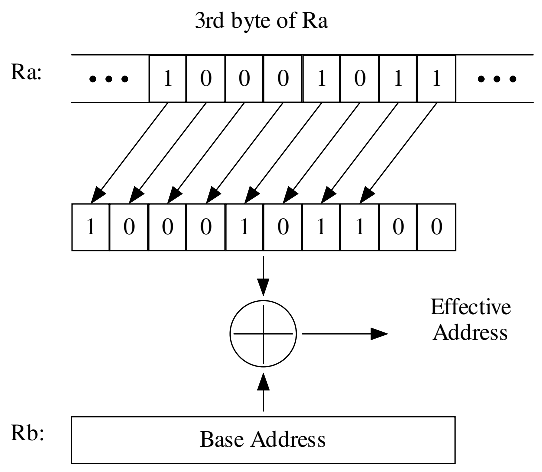

Figure 2. `load.3b.4 Rd,Ra(Rb)`

4. Simulations

We use the simulator of the IMPACT compiler [9] to evaluate the effect of different addressing modes on the encryption performance. Optimized C implementations of the encryption algorithms are used as the source. Findings for RC6 and Serpent are not reported. RC6 is the only AES finalist that did not use table lookups. Serpent, on the other hand, includes table lookups, however in its optimized implementations, these lookups are realized as a series of logical operations (ands and ors).

We simulate each of the algorithms using the four addressing modes explained previously, and perform an encryption of one hundred 128-bit blocks of data. For each of the addressing modes, we simulate the algorithms on 1,2,4, and 8-wide EPIC (Explicitly Parallel Instruction Computer) processors. To fairly compare the algorithms with different degrees of dependence on table lookups, the number of memory ports is also varied from one to the issue width, in multiples of two. We determine how the performance is affected by the following variables: a) the addressing mode in the architecture, b) the issue width of the processor, and c) the number of memory ports.

5. Conclusions

Table 1 is a partial summary of our results for one and two-wide processors with one memory port (also see Figure 3). All algorithms show speedups ranging from 20% (MARS) to over 200% (Rijndael). These figures verify MARS's relatively less use of table lookups, and Rijndael's heavy dependence on them. Twofish is in between with a speedup of 49% in a two-wide processor.

Table 2 summarizes the results for an eight-wide processor with one and two memory ports. Figure 4 shows the same data for up to eight memory ports. These results are also congruent with the previous and show that the performance of the AES winner, Rijndael, is very dependent on both the number of memory ports as well as the addressing mode used [10]. Other algorithms show a saturation beyond two memory ports, after which the addressing mode becomes more important than the number of memory ports. Best speedup in this case is for Rinjdael, with 41% for two memory ports.

As the data for the `load.ex.sc` indicates, the benefits of eliminating the overhead of effective address calculations are significant. `Load.ex.sc` achieves this by migrating the effective address calculations into the load operation. Since this requires selecting and shifting the index bits followed by the addition of the base address, `load.ex.sc` is likely to take two cycles in a high-performance processor with a high clock rate. In such a case, using this instruction would be similar to using scaled-indexed addressing and the `extract` instruction; hence the latter solution might be preferred. For a crypto-processor, however, the cycle times can be kept longer, and `load.ex.sc` can be implemented as a single-cycle instruction. This would allow both a low-power implementation due to the longer clock cycles, as well as significant speedups due to the elimination of the time overhead of effective address calculations.

6. References

[1] C. Burwick, D. Coppersmith, E. D'Avignon, et al, "MARS – a candidate cipher for AES," *1st AES Conference*, Ventura, CA, August 20-22, 1998.

[2] R.L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Lin, "The RC6 Block Cipher," *1st AES Conference*, Ventura, CA, August 20-22, 1998.

[3] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *1st AES Conference*, Ventura, CA, August 20-22, 1998.

[4] R. Anderson, E. Biham and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," *1st AES Conference*, Ventura, CA, August 20-22, 1998.

[5] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-bit Block Cipher," *1st AES Conference*, Ventura, CA, August 20-22, 1998.

[6] R.B. Lee, "Precision Architecture," *IEEE Computer*, Vol. 22, No. 1, pp. 78-91, January 1989.

[7] Intel, "IA-64 Architecture Software Developer's Manual, Vol. 3: ISA Reference," Rev. 1.1, ID 245319-002, July 2000.

[8] J. Burke, J. McDonald and T. Austin, "Architectural Support for Fast Symmetric-Key Cryptography," *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems,* Cambridge, MA, November 12-15, 2000.

[9] P.P. Chang, S.A. Mahlke, et al, "IMPACT: An Architectural Framework for Multiple Instruction Issue Processors," *Proceedings of the 18th International Symposium on Computer Architecture,* Toronto, Canada, pp. 266-275, May 28, 1991.

[10] J. Worley, B. Worley, T. Christian and C. Worley, "AES Finalists on PA-RISC and IA-64: Implementations and Performance," *Proceedings of the 3rd AES Conference*, New York, NY, pp. 57-74, April 12-14, 2000.

Table 1. Speedups for varying issue widths, w.r.t. indexed addressing. (a) Scaled-indexed (b) Scaled-indexed + `extract` (c) `load.ex.sc`.

Algorithm	Arch.	(a)	(b)	(c)
MARS	1G_1M[1]	1.09	1.16	1.21
	2G_1M	1.08	1.15	1.21
Rijndael	1G_1M	1.23	1.57	2.00
	2G_1M	1.26	1.63	2.05
Twofish	1G_1M	1.16	1.39	1.62
	2G_1M	1.17	1.39	1.49

Table 2. Speedups for one and two memory ports w.r.t. indexed addressing. (a) Scaled-indexed (b) Scaled-indexed + extract (c) `load.ex.sc`.

Algorithm	Arch.	(a)	(b)	(c)
MARS	8G_1M	1.05	1.10	1.10
	8G_2M	1.04	1.09	1.09
Rijndael	8G_1M	1.08	1.08	1.10
	8G_2M	1.19	1.37	1.41
Twofish	8G_1M	1.06	1.14	1.14
	8G_2M	1.07	1.14	1.21

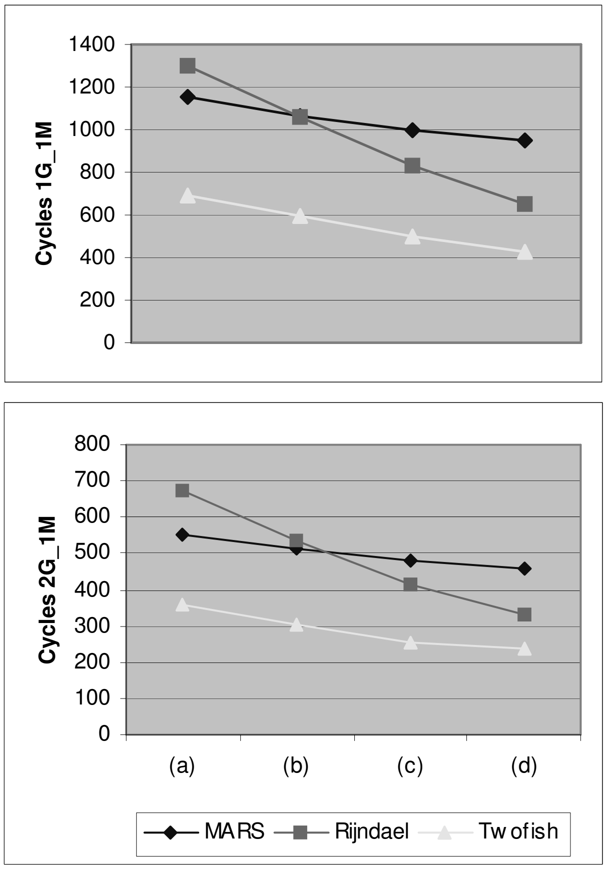

Figure 3. Execution cycles in one and two-wide processors with one memory port. (a) Indexed (b) Scaled-indexed (c) Scaled-indexed + `extract` (d) `load.ex.sc`.

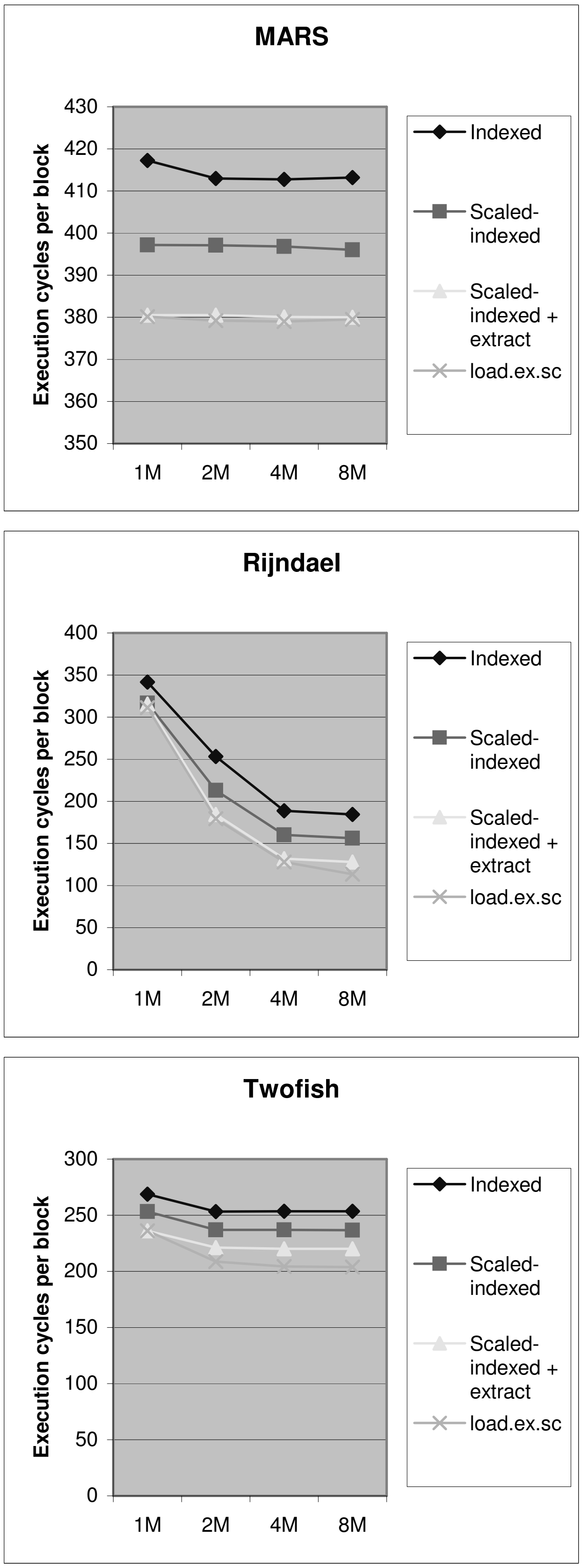

Figure 4. Execution cycles for one, two, four and eight memory ports in an eight-wide processor.

[1] 1G_1M indicates a one-wide (1G = 1 General ALU) processor with one memory port (1M = 1 Memory Port). Likewise, 8G_2M is an eight-wide processor with two memory ports.