Performance of `CodornicesRq` software package

Second release performance for `X86-64 platform` and `ARM platform`

Michael Luby, Lorenz Minder, Pooja Aggarwal
International Computer Science Institute

May 8, 2019

1 Introduction

We provide performance results for the second release of the `CodornicesRq` software package.[1] Performance results for the following two test platforms are provided:

`X86-64 platform`: AMD Ryzen 5 2600 X86-64 processor @ 3.4 GHz.

`ARM platform`: Raspberry Pi 3 Model B+. The SoC is Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz, The tests were run with the platform in the 32-bit ARMv7 mode.

All tests are run sequentially using the single threaded `CodornicesRq` library compiled for the Linux OS, and thus only one core is used. All tests use a symbol size of $1,280$ bytes $= 10,240$ bits, which is a typical payload size for packets sent over the internet. For example, if $K = 1,000$ for a source block, the source block size is $1,280,000$ bytes $= 10,240,000$ bits.

We consider three different types of operations:

- **Encode:** generate K repair symbols from a source block of K source symbols.
  - Input symbols: source symbols with identifiers $\langle 0, \ldots, K-1 \rangle$
  - Output symbols: repair symbols with identifiers $\langle K, \ldots, 2K-1 \rangle$
- **Decode with no overhead:** generate the K source symbols of a source block from K random repair symbols for that source block.
  - Input symbols: K repair symbols with randomly chosen identifiers
  - Output symbols: K source symbols

[1] The second release performance is substantially improved compared to the first release performance, and we expect substantial performance improvements in future releases.

- **Decode with 5% overhead:** generate the K source symbols of a source block from $M = 1.05 \cdot K$ random repair symbols for that source block.
  - Input symbols: M repair symbols with randomly chosen identifiers
  - Output symbols: K source symbols

Decode performance improves as the overhead increases. Performance also improves as the amount of source symbols used to decode increases, as generally it is somewhat faster to decode when source symbols are available (in particular, there is no need to generate the already available source symbols). Thus, *Decode with no overhead* results are essentially worst case, whereas *Decode with 5% overhead* results are closer to what is expected in practice.

As expected, there are no decode failures in tests when decoding with 5% overhead, and there are less than 1% decode failures in tests when decoding with no overhead.

Compile time is defined as the time it takes to compile the program of symbol operations that will be used to generate output symbols from the input symbols for a source block. Thus, the compile time is the time to run `RqInterCompile` plus the time to run `RqOutCompile` for the source block, where `RqInterCompile` and `RqOutCompile` are described in [1]. (The time for the calls to `RqInterGetMemSizes`, `RqInterInit`, `RqInterAddIds`, `RqOutGetMemSizes`, `RqOutInit`, and `RqOutAddIds`, are essentially zero, and thus are not reported.)

Execute time is defined as the time is takes to execute the compiled program of symbol operations to generate the output symbols from the input symbols for a source block. Thus, the execute time is the time to run `RqInterExecute` plus the time to run `RqOutExecute` for the source block, where `RqInterExecute` and `RqOutExecute` are described in [1].

Execute speed, the ratio of the source block size to the execute time, is shown in Figure 1 for various values of K. Execute speed provides an indication of how much of the CPU is needed for execution on average, e.g., when a stream of data arriving at 10 Mbps, less than 0.6% of the CPU is used on average for encode execution if the stream is partitioned into source blocks using $K = 1,000$ and encoded to generate a 20 Mbps stream.

Execute speeds for *Encode* and *Decode with no overhead* are essentially identical, and execute speed for *Decode with 5% overhead* is somewhat faster. Generally, compile time is a small proportion of execute time, as shown in Figure 2. Furthermore, in many use cases, a program can be compiled once and executed multiple times on different source blocks, and thus the compile time can be amortized over many executions.

Figure 3 shows the compile time and execute time raw data from which Figure 1 and Figure 2 are derived for the `X86-64 platform`, and Figure 6 shows the compile time and execute time raw data from which Figure 4 and Figure 5 are derived for the `ARM platform`. In Figure 3 and Figure 6:

- The column labeled "Inter" in the "Compile Time" category is the time to execute `RqInterCompile` per source block.

- The column labeled "Out" in the "Compile Time" category is the time to execute `RqOutCompile` per source block.
- The column labeled "Inter" in the "Execute Time" category is the time to execute `RqInterExecute` per source block.
- The column labeled "Out" in the "Execute Time" category is the time to execute `RqOutExecute` per source block.

References

[1] Michael Luby, Lorenz Minder, Pooja Aggarwal. How to use the CodornicesRq software package. International Computer Science Institute, February 5, 2019.

List of Figures

1 `X86-64 platform` execute speed for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. 5
2 `X86-64 platform` compile time as a % of execute time for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. 6
3 `X86-64 platform` raw timing data (in milliseconds) from which the information in Figure 1 and Figure 2 can be deduced. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. . 7
4 `ARM platform` execute speed for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. 8
5 `ARM platform` compile time as a % of execute time for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. 9
6 `ARM platform` raw timing data (in milliseconds) from which the information in Figure 4 and Figure 5 can be deduced. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block. 10

Figure 1: `X86-64 platform` execute speed for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

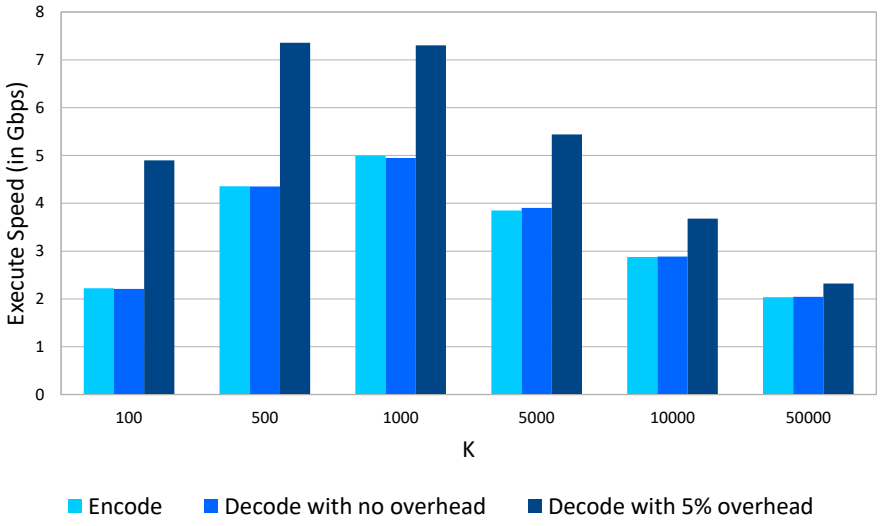

Figure 2: `X86-64 platform` compile time as a % of execute time for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

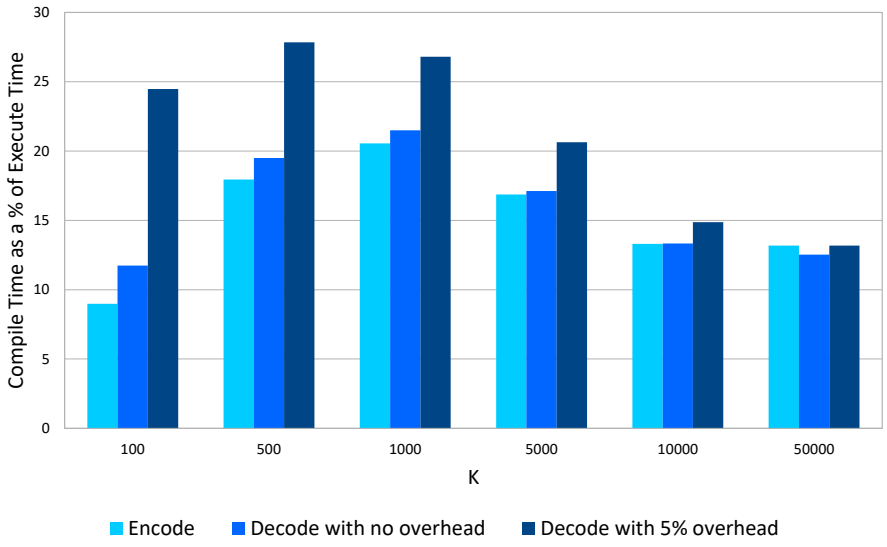

Figure 3: `X86-64 platform` raw timing data (in milliseconds) from which the information in Figure 1 and Figure 2 can be deduced. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

K		Compile Time (in ms)			Execute Time (in ms)		
		Inter	Out	Total	Inter	Out	Total
100	Encode	0.036	0.005	0.041	0.434	0.026	0.461
	Decode with no overhead	0.050	0.005	0.054	0.440	0.023	0.463
	Decode with 5% overhead	0.046	0.005	0.051	0.185	0.024	0.209
500	Encode	0.188	0.023	0.211	1.041	0.135	1.176
	Decode with no overhead	0.206	0.023	0.229	1.036	0.140	1.176
	Decode with 5% overhead	0.171	0.023	0.194	0.555	0.141	0.696
1000	Encode	0.375	0.046	0.421	1.769	0.280	2.049
	Decode with no overhead	0.400	0.046	0.445	1.778	0.292	2.071
	Decode with 5% overhead	0.330	0.046	0.376	1.109	0.293	1.402
5000	Encode	2.021	0.221	2.242	11.062	2.230	13.294
	Decode with no overhead	2.025	0.221	2.246	10.942	2.174	13.118
	Decode with 5% overhead	1.721	0.221	1.942	7.243	2.164	9.409
10000	Encode	4.301	0.439	4.740	28.393	7.212	35.607
	Decode with no overhead	4.291	0.439	4.730	28.173	7.292	35.468
	Decode with 5% overhead	3.701	0.439	4.139	20.955	6.868	27.827
50000	Encode	31.068	2.152	33.221	192.185	59.732	251.920
	Decode with no overhead	29.239	2.152	31.391	190.353	60.134	250.490
	Decode with 5% overhead	26.911	2.153	29.064	160.913	59.584	220.500

Figure 4: `ARM platform` execute speed for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

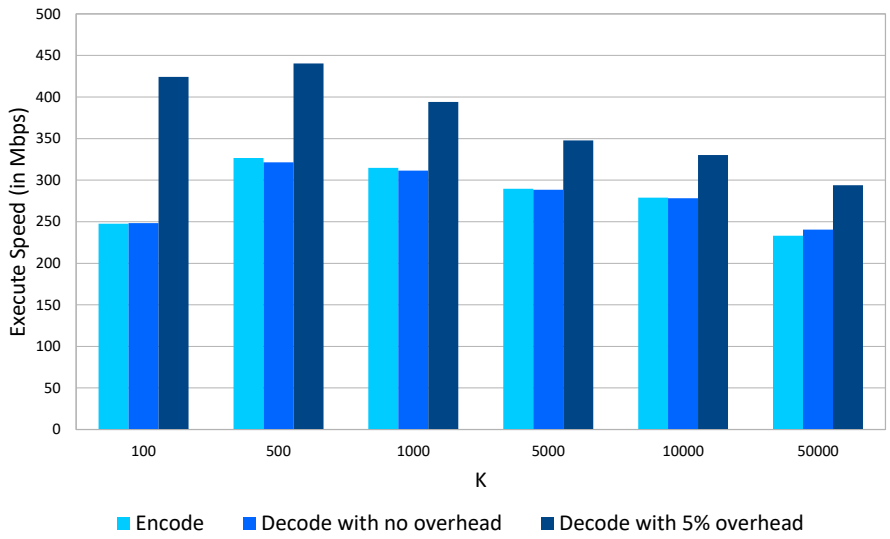

Figure 5: `ARM platform` compile time as a % of execute time for source blocks with K source symbols. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

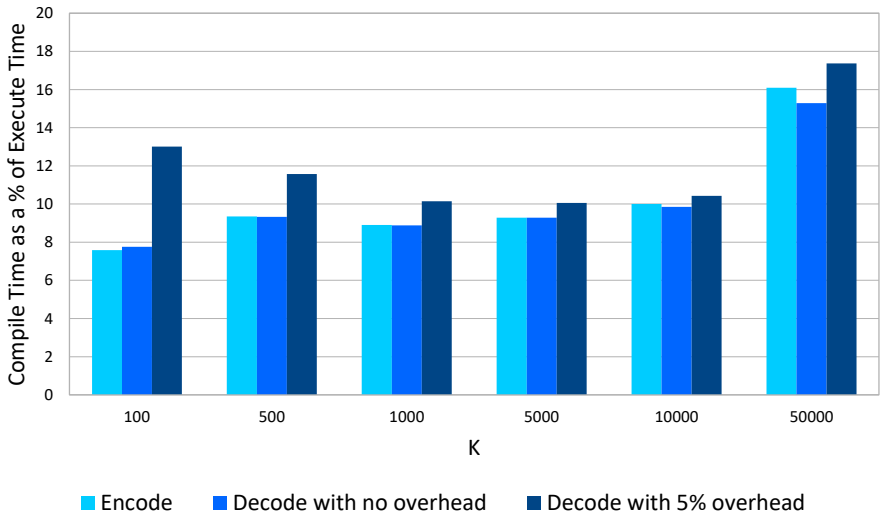

Figure 6: `ARM platform` raw timing data (in milliseconds) from which the information in Figure 4 and Figure 5 can be deduced. The symbol size is $1,280$ bytes $= 10,240$ bits for each source block.

K		Compile Time (in ms)			Execute Time (in ms)		
		Inter	Out	Total	Inter	Out	Total
100	Encode	0.271	0.043	0.314	3.699	0.433	4.138
	Decode with no overhead	0.278	0.042	0.320	3.742	0.377	4.125
	Decode with 5% overhead	0.272	0.042	0.314	2.029	0.379	2.413
500	Encode	1.246	0.219	1.465	12.283	3.379	15.673
	Decode with no overhead	1.264	0.221	1.485	12.383	3.529	15.924
	Decode with 5% overhead	1.126	0.219	1.345	8.173	3.441	11.626
1000	Encode	2.465	0.433	2.897	25.085	7.442	32.539
	Decode with no overhead	2.482	0.435	2.918	25.025	7.830	32.868
	Decode with 5% overhead	2.199	0.436	2.636	18.285	7.680	25.979
5000	Encode	14.312	2.103	16.415	135.492	41.264	176.769
	Decode with no overhead	14.361	2.124	16.486	135.427	42.116	177.557
	Decode with 5% overhead	12.696	2.101	14.796	105.642	41.518	147.174
10000	Encode	32.481	4.236	36.718	281.329	85.817	367.160
	Decode with no overhead	32.003	4.239	36.242	281.305	86.789	368.111
	Decode with 5% overhead	28.084	4.257	32.341	223.008	87.023	310.045
50000	Encode	332.597	20.950	353.547	1721.670	475.670	2197.350
	Decode with no overhead	304.741	20.878	325.618	1653.730	476.137	2129.880
	Decode with 5% overhead	281.779	20.975	302.754	1264.790	477.789	1742.590