Efficiency Experiments - Comparing NIFs to C++ Functions

Ellen Newcomb, en203

March 28, 2017

Abstract

An argument against the use of NIFs (*Native Implemented Functions*) over simply using the pre-existing C++ is the possible reduction in efficiency. These tests were devised to illustrate the possible effects of the latency involved with using and loading NIFs. This experiment is attempting to answer three questions with three tests; Are Erlang programs using NIFs inherently slower than their C or C++ counterparts, Is the overhead of using a NIF too great to reasonably allow use, and Could Erlang be considered a better alternative to C or C++ with the help of NIFs?

1 Method

Although it is not always possible to directly translate C or C++ functions into Erlang NIFs, this difference in implementation is potentially interesting, for example, it may be more efficient to write a function using Erlangs recursion rather than traditional C loops. The aims of this experiment are to deduce whether or not the potential latency is so high that it would impact a user writing larger programs with NIFs.

The experiment is split into three smaller tests; Comparing six increasingly complex like for like programs, comparing a factorial program with increasingly large inputs, and comparing a traditional C loop program with one using Erlang style recursion.

1.1 Comparing like for like programs

Six programs were written in C++ and their counterparts were written in Erlang and C++ using NIFs to communicate between the two languages. These programs were written in order to ensure that they remained as close to each other as possible, as the intention is to measure the effect of loading a shared object library, and using the *erl_nif* functions to transfer arguments and returned values.

The first of these programs is a simple "*Hello World*" style program which prints an output to the terminal. The second performs some simple arithmetic on an integer, and prints it to the terminal at the end of a string. The third takes a single parameter and adds it to the end of a string which is printed to the terminal. The fourth is a factorial function which is performed on a single parameter and prints a string. The fifth uses SDL functions to display a simple hello world BMP, but does not use the *SDL_Delay()* function as this could heavily skew the results. The sixth is similar to the fifth, but is considerably longer.

The sixth program set is written in a generally inefficient way, using code duplication rather than while loops or recursion. This is to avoid any possible side effects caused by the difference in efficiency in these different techniques.

Each program is run 25 times on either the university's *Dove* server to avoid conflicts with simultaneous, external interference, or Raptor. Unfortunately SDL programs wouldn't run on Dove.

1.2 Comparing factorial

The factorial programs from the first test take any integer input from the terminal. This test was run 25 times for each program (standard C++ functions, and Erlang NIFs) and for each input. The inputs were: 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 200000000, 300000000, 400000000, 500000000, 600000000, 700000000, 800000000, 900000000, & 1000000000. They were chosen experimentaly, by executing the function with increasingly large integers until it was producing outputs after some lag, in order to obtain results worth using.

1.3 Comparing loops with recursion

Two programs were written to compare the efficiency of C style while and for loops to using Erlangs recursion. The idea behind this is to illustrate why a developer might chose to use Erlang over C/C++, which is one of the main themes of the project.

The C++ program uses loops to rapidly load and switch BMPs on SDL surfaces, whereas the Erlang program uses recursion to achieve the same results. The inputs used to test the programs were: 10, 15, 20, 25, & 30. Each input was tested 25 times.

1.4 Executing the programs

The C++ tests were run from the terminal using the *time* command. Typically this would consist of:

```
/usr/bin/time -f \%eElapsed -o timerResults001.txt --append
    timerCpp001
```

The above command will append any results to the file timerResults001.txt, in the format "*Time in seconds to two decimal places* Elapsed". This is the time it takes for the function to run according to real time (rather than system or user time). Any parameters for timerCpp001 are tagged onto the end of the command.

The Erlang NIF tests were run from the erlang terminal using the *timer:tc* command. This would typically be of the format:

```
timer:tc(timerErl001, timeTest, []).
```

This command, unlike the above command for C++, does not store the results to a file, and instead prints them to the terminal in the format {*time elapsed in micro seconds, return value of the function*}. The command takes the module, the function, and a list of parameters for that function.

Please Be Aware: If recreating these tests with the code included in the corpus, certain programs (specifically those labelled 006 and 007) use rapid reloading of BMPs into an SDL surface. This causes the surface to flash rapidly, which could potentially cause epileptic seizures in certain individuals.

2 Results

In order to compare results accurately the raw data will be represented in set form (a,b,c,...), and will be averaged when shown in graph form, to remove outliers.

2.1 Comparing like for like programs

2.1.1 Hello World

The results for the Hello World style programs (001 in the corpus) are as follows:

C++: {0.00s, 0.00s}, Average = 0.00s

Erlang: {0.000007s, 0.000003s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000007s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000007s, 0.000001s, 0.000001s, 0.000004s, 0.000001s, 0.000001s} Average = 0.00000192s

In the interest of comparing the results available the results are rounded to two decimal places, making the average of the Erlang times 0.00s.

The results for this particular test are more or less inconclusive as the time function for C++ programs cannot measure to the same accuracy as Erlang's timer module. However, the Erlang function takes, on average, less than 2 microseconds, which can be considered negligible.

2.1.2 Simple Arithmetic

The results for the Simple Arithmetic programs (002 in the corpus) are as follows:

C++: {0.00s, 0.00s}, Average = 0.00s

Erlang: {0.000002s, 0.000003s, 0.000003s, 0.000003s, 0.000003s, 0.000001s, 0.000007s, 0.000001s, 0.000001s, 0.000004s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000003s, 0.000001s, 0.000001s, 0.000001s} Average = 0.0000018s

Once again, in order to remove extra accuracy the Erlang average is rounded to two decimal places, making it 0.00s.

The results of this test are also inconclusive, considering the test is too short to register anything greater than 0.01 seconds. Again, the run time could be considered negligible in everyday use.

2.1.3 Single Parameter

The single parameter programs (003 in the corpus) results are similar to the previous two programs, and are as follows:

C++: {0.00s, 0.00s}, Average = 0.00s

Erlang: {0.000002s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000005s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s} Average = 0.0000012s $\approx$ 0.00s

As per the above results, the time it takes for such a simple program to run in either language system in completely negligible in everyday use.

2.1.4 Factorial

The factorial programs (004 in the corpus) results are as follows:

C++: {0.00s, 0.00s}, Average = 0.00s

Erlang: {0.000002s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000005s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s} Average = 0.0000012s $\approx$ 0.00s

Surprisingly these results are identical to those from the previous program. Again, the time it takes for these programs to execute could be considered negligible.

2.1.5 Simple SDL

The SDL functions take a considerably longer time to execute than simple arithmetic or parameter parsing. The results are as follows for the simple sdl programs (005 in the corpus):

C++: {4.12s, 4.29s, 4.16s, 4.07s, 4.10s, 4.09s, 4.05s, 4.22s, 4.06s, 4.05s, 4.16s, 4.08s, 4.28s, 4.16s, 4.04s, 4.16s, 4.47s, 4.08s, 4.15s, 5.25s, 4.13s, 4.01s, 4.01s, 4.03s, 4.39s} Average = 4.1844s $\approx$ 4.18s

Erlang: {4.072634s, 3.991331s, 4.065723s, 4.041015s, 4.071194s, 4.040646s, 4.002148s, 4.049025s, 4.029042s, 4.020014s, 4.186578s, 4.023660s, 4.131931s, 4.099924s, 4.008859s, 4.119519s, 4.117680s, 4.084124s, 4.040599s, 3.981063s, 4.117753s, 4.003261s, 4.083563s, 4.091482s, 3.978826s} Average = 4.05806376s $\approx$ 4.06s

The results from this test show that the runtime of both programs is extremely close, with only a difference of 0.12s between the two rounded averages, with the Erlang NIF implementation surprisingly running slightly faster than it's C++ counterpart.

2.1.6 Extended SDL

The extended SDL programs swap two BMPs and loads them into the visible surface 10 times (resulting in 20 swaps), so the run times of these functions is considerably long. The interesting part of this test is that the Erlang file does call the NIF functions separately for every swap (and all of the SDL functions, like it does in the previous test, and in most uses of the SDL implementation written for the project). The results for the programs (006 in the corpus) are as follows:

C++: {5.44s, 5.44s, 5.47s, 5.46s, 5.63s, 5.34s, 5.35s, 5.46s, 5.31s, 5.77s, 5.75s, 5.24s, 5.45s, 5.29s, 5.44s, 5.91s, 5.48s, 5.59s, 5.27s, 5.86s, 5.36s, 5.34s, 5.33s, 5.46s, 5.49s} Average = 5.4772s $\approx$ 5.48s

Erlang: {5.197604s, 5.483497s, 5.531375s, 5.413344s, 5.490796s, 5.420796s, 5.618665s, 5.460986s, 5.436404s, 5.517628s, 5.235367s, 6.120621s, 5.583518s, 5.865637s, 5.586122s, 5.351303s, 5.396594s, 5.401040s, 5.464043s, 5.262781s, 5.484398s, 5.442589s, 5.324480s, 5.348019s, 5.402137s} Average = 5.473589760000001s $\approx$ 5.47s

Again the time taken to execute these programs is incredibly close, with an average difference of 0.01s, with Erlang, once again, narrowly edging out C++ as the fastest.

2.2 Comparing factorial

The factorial programs were both run with the same arguments (10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 200000000, 300000000, 400000000, 500000000, 600000000, 700000000, 800000000, 900000000, & 1000000000) 25 times each. The results are grouped by argument.

2.2.1 10

At this point the programs did not display any human-perceptible lag, or any recorded lag above 8 microseconds, therefore their results are negligible. They are, for reference, as follows:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.000003s, 0.000001s, 0.000001s, 0.000001s, 0.000002s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000008s, 0.000004s, 0.000004s, 0.000004s, 0.000001s, 0.000004s, 0.000003s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s, 0.000001s} Average = 0.00000192 $\approx$ 0.00s

2.2.2 100

As with the previous test, the argument was still too low to produce any usable results. They are as follows, for reference:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.000004s, 0.000005s, 0.000005s, 0.000006s, 0.000005s, 0.000005s, 0.000005s, 0.000006s, 0.000005s, 0.000006s, 0.000006s, 0.000006s, 0.000006s, 0.000005s, 0.000006s, 0.000006s, 0.000005s, 0.000006s, 0.000006s, 0.000006s, 0.000007s, 0.000006s, 0.000006s, 0.000025s} Average = 0.00000616 $\approx$ 0.00s

2.2.3 1000

There is still no human-perceptible lag at this stage, nor are the recorded results greater than 10 micro seconds, thus these results are negligible. The results are as follows, for reference:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.000009s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000008s, 0.000009s, 0.000010s, 0.000009s, 0.000009s, 0.000009s, 0.000008s, 0.000009s, 0.000008s, 0.000009s, 0.000009s, 0.000009s, 0.000010s, 0.000009s, 0.000009s, 0.000009s} Average = 0.00000864 $\approx$ 0.00s

2.2.4 10000

As before there is still no noticeable lag, although the Erlang results are starting to record some more interesting numbers. As they are still very low (around 42 micro seconds) they are incomparable with the C++ results. Results are as follows:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.000043s, 0.000043s, 0.000041s, 0.000043s, 0.000041s, 0.000041s, 0.000041s, 0.000040s, 0.000041s, 0.000043s, 0.000043s, 0.000041s, 0.000041s, 0.000043s, 0.000042s, 0.000043s, 0.000040s, 0.000041s, 0.000044s, 0.000040s, 0.000041s, 0.000043s, 0.000043s, 0.000042s, 0.000042s} Average = 0.00004184s $\approx$ 0.00s

2.2.5 100000

Still no usable results from the C++ but the Erlang results do show that the run time is increasing in an expected. Results are as follows:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.000376s, 0.000378s, 0.000377s, 0.000377s, 0.000377s, 0.000365s, 0.000364s, 0.000365s, 0.000365s, 0.000377s, 0.000378s, 0.000378s, 0.000375s, 0.000374s, 0.000376s, 0.000377s, 0.000375s, 0.000396s, 0.000377s, 0.000378s, 0.000376s, 0.000376s, 0.000374s, 0.000376s, 0.000373s} Average = 0.00037519s $\approx$ 0.00s

2.2.6 1000000

The results of this test reflect those of the previous tests, there are still no usable numbers from the C++, and the Erlang numbers are climbing as expected. The results are as follows:

C++: {0.00s, 0.00s} Average = 0.00s

Erlang: {0.003735s, 0.003739s, 0.003630s, 0.003735s, 0.003622s, 0.003623s, 0.003620s,0.003635s, 0.003701s, 0.003689s, 0.003707s, 0.003615s,0.003734s, 0.003625s, 0.003623s, 0.003630s, 0.003716s, 0.003711s, 0.003751s, 0.003744s, 0.003732s, 0.003720s, 0.003732s, 0.003879s, 0.003724s} Average = 0.00369488 $\approx$ 0.00s

2.2.7 10000000

This is the point where the recorded results start to display some lag in C++, although the lag is still almost imperceptible to a human user. The results are as follows:

C++: {0.05s, 0.04s} Average = 0.0496s $\approx$ 0.05s

Erlang: {0.037173s, 0.037418s, 0.037075s, 0.037173s, 0.037194s, 0.037164s, 0.037351s, 0.036486s, 0.037267s, 0.037315s, 0.038818s, 0.037448s, 0.037344s, 0.037303s, 0.036944s, 0.037231s, 0.037159s, 0.036078s, 0.036150s, 0.037285s,

0.037203s, 0.036058s, 0.036113s, 0.036226s, 0.036421s} Average = 0.03701588s ≈ 0.04s

The results here are somewhat comparable. They are incredibly close, with Erlang seemingly being the slightly faster of the two, edging out C++ by 0.01s on average.

2.2.8 100000000

The lag for this test is now human perceptible. The results are as follows:

C++: {0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.51s, 0.50s, 0.50s, 0.50s, 0.50s, 0.50s, 0.51s, 0.50s, 0.50s, 0.52s, 0.50s, 0.50s} Average = 0.5016s ≈ 0.50s

Erlang: {0.371474s, 0.371398s, 0.365854s, 0.361066s, 0.363139s, 0.368661s, 0.361234s, 0.361393s, 0.363321s, 0.368596s, 0.370578s, 0.370629s, 0.371119s, 0.363947s, 0.372043s, 0.370122s, 0.360489s, 0.360638s, 0.367078s, 0.371529s, 0.372209s, 0.365999s, 0.368652s, 0.371654s, 0.364780s} Average = 0.36710408s ≈ 0.37s

The C++ is starting to fall behind the Erlang here, but the numbers are still so close together the difference is negligible.

2.2.9 200000000

Instead of multiplying the input by 10, the last half of the arguments are incremented by 100000000, as their results are usable without increasing too quickly. The results for 200000000 are as follows:

C++: {0.97s, 0.88s, 0.82s, 0.84s, 0.86s, 0.88s, 0.85s, 0.89s, 0.89s, 0.87s, 0.80s, 0.82s, 0.87s, 0.90s, 0.94s, 0.95s, 0.97s, 0.78s, 0.89s, 0.99s, 0.78s, 0.89s, 0.95s, 0.96s, 0.78s} Average = 0.8808s ≈ 0.88s

Erlang: {0.729203s, 0.722091s, 0.740620s, 0.735113s, 0.738256s, 0.735489s, 0.729790s, 0.721444s, 0.737854s, 0.732616s, 0.724445s, 0.724898s, 0.744310s, 0.738539s, 0.739546s, 0.743648s, 0.737306s, 0.721735s, 0.732143s, 0.721755s. 0.720988s, 0.729810s, 0.727811s, 0.739940s} Average = 0.702774s ≈ 0.70s

The difference between the two sets of results, when averaged, is 0.18s. This is slightly greater than the last test and sees Erlang running faster again. However, this difference is still so little that it is negligible.

2.2.10 300000000

The results for the input of 300000000 are as follows:

C++: {1.08s, 1.23s, 1.08s, 1.22s, 1.25s, 1.13s, 1.26s, 1.08s, 1.24s, 1.08s, 1.24s, 1.11s, 1.13s, 1.25s, 1.10s, 1.26s, 1.15s, 1.26s, 1.09s, 1.22s, 1.25s, 1.17s, 1.16s, 1.25s, 1.09s} Average = 1.1752s ≈ 1.18s

Erlang: {1.083380s, 1.082177s, 1.082984s, 1.084784s, 1.095079s, 1.116867s, 1.116666s, 1.112014s, 1.108740s, 1.115855s, 1.117718s, 1.094367s, 1.104060s, 1.095380s, 1.081956s, 1.097126s, 1.101551s, 1.108507s, 1.112467s, 1.102020s, 1.107615s, 1.107172s, 1.111957s, 1.110791s, 1.112573s} Average = 1.10255224 ≈ 1.10s

As before, the Erlang program runs slightly quicker on average, but not enough to be non-negligible (0.08s).

2.2.11 400000000

The results for an input of 400000000 are as follows:

C++: {1.40s, 1.36s, 1.47s, 1.45s, 1.53s, 1.53s, 1.37s, 1.56s, 1.46s, 1.39s, 1.39s, 1.42s, 1.44s, 1.45s, 1.48s, 1.47s, 1.46s, 1.53s, 1.47s, 1.41s, 1.14s, 1.55s, 1.36s, 1.49s, 1.43s} Average = 1.44040s $\approx$ 1.44s

Erlang: {1.457455s, 1.470299s, 1.466735s, 1.467807s, 1.497536s, 1.453858s, 1.465369s, 1.446302s, 1.459514s, 1.473009s, 1.485310s, 1.469361s, 1.451831s, 1.454837s, 1.462330s, 1.461531s, 1.483328s, 1.459418s, 1.476368s, 1.470244s, 1.467959s, 1.459461s, 1.473712s, 1.452472s, 1.444121s} Average = 1.46520668 $\approx$ 1.47s

Unlike the previous tests, in this test the Erlang program was 0.03s slower than it's C++ counterpart, on average.

2.2.12 500000000

The results for an input of 500000000 are as follows:

C++: {1.73s, 1.73s, 1.65s, 1.78s, 1.67s, 1.70s, 1.75s, 1.75s, 1.80s, 1.83s, 1.82s, 1.84s, 1.76s, 1.80s, 1.83s, 1.67s, 1.76s, 1.69s, 1.79s, 1.80s, 1.64s, 1.64s, 1.71s, 1.79s, 1.81s} Average = 1.749600 $\approx$ 1.75s

Erlang: {1.831430s, 1.840523s, 1.808409s, 1.802687s, 1.805042s, 1.830444s, 1.810098s, 1.818603s, 1.816484s, 1.819633s, 1.813476s, 1.821229s, 1.820447s, 1.852798s, 1.810841s, 1.832683s, 1.812352s, 1.840807s, 1.827038s, 1.833999s, 1.828367s, 1.853055s, 1.820199s, 1.815670s, 1.805916s} Average = 1.8228892 $\approx$ 1.82s

The Erlang program was slower by 0.07s than the C++ program, on average for this input.

2.2.13 600000000

The results for an input of 600000000 are as follows:

C++: {2.07s, 2.11s, 1.95s, 2.04s, 2.12s, 2.02s, 2.07s, 2.00s, 1.95s, 2.06s, 1.93s, 2.12s, 1.93s, 1.98s, 1.98s, 1.95s, 2.13s, 2.05s, 1.98s, 2.05s, 2.01s, 1.99s, 1.96s, 1.96s, 2.11s} Average = 2.0208s $\approx$ 2.02s

Erlang: {2.230915 2.213646 2.209400 2.211899 2.219246 2.230320 2.208439 2.214860 2.224520 2.224502 2.228841 2.212129 2.222311 2.211982 2.200934 2.208808 2.188355 2.231259 2.209676 2.212508 2.246983 2.238952 2.230489 2.225013 2.230550 2.257115} Average = 2.30974608s $\approx$ 2.31s

The results for this test indicate that the Erlang program is falling behind it's C++ counterpart, here by 0.29s on average.

2.2.14 700000000

The results for an input of 700000000 are as follows:

C++: {2.41s, 2.23s, 2.25s, 2.40s, 2.28s, 2.30s, 2.40s, 2.39s, 2.32s, 2.31s, 2.39s, 2.37s, 2.35s, 2.33s, 2.34s, 2.25s, 2.24s, 2.37s, 2.31s, 2.26s, 2.41s, 2.37s, 2.27s, 2.31s, 2.30s} Average = 2.326400s ≈ 2.33s

Erlang: {2.580534s, 2.620625s, 2.689154s, 2.596227s, 2.612803s, 2.644210s, 2.589759s, 2.600134s, 2.566624s, 2.615701s, 2.655988s, 2.596650s, 2.654299s, 2.6095343s, 2.664970s, 2.630918s, 2.603128s, 2.611724s, 2.612906s, 2.604492s, 2.579124s, 2.583121s, 2.567946s, 2.555179s, 2.534818s} Average = 2.6072227s ≈ 2.61s

Like the most recent two tests the Erlang program was slower than the C++ program at this input, by 0.29s on average.

2.2.15 800000000

The results for an input of 800000000 are as follows:

C++: {2.56s, 2.61s, 2.65s, 2.69s, 2.51s, 2.75s, 2.69s, 2.62s, 2.68s, 2.56s, 2.69s, 2.54s, 2.68s, 2.58s, 2.59s, 2.70s, 2.69s, 2.56s, 2.70s, 2.59s, 2.69s, 2.55s, 2.53s} Average = 2.416400s ≈ 2.42s

Erlang: {2.889767s, 2.885877s, 2.887335s, 2.895206s, 2.902643s, 2.888964s, 2.885560s, 2.89017s, 2.885679s, 2.883900s, 2.884662s, 2.859572s, 2.844323s, 2.861317s, 2.873626s, 2.843742s, 2.782991s, 2.778402s, 2.820908s, 2.857153s, 2.860557s, 2.846620s, 2.880597s, 2.866318s, 2.880149s} Average = 2.8654415s ≈ 2.87s

2.2.16 900000000

The results for an input of 900000000 are as follows:

C++: {2.80s, 2.98s, 2.90s, 2.97s, 2.90s, 2.99s, 2.97s, 2.89s, 2.86s, 2.80s, 2.79s, 2.94s, 2.93s, 2.85s, 2.92s, 2.87s, 2.88s, 2.86s, 2.84s, 2.83s, 2.84s, 2.81s, 2.85s, 2.82s} Average = 2.76359s ≈ 2.76s

Erlang: {3.239644s, 3.135548s, 3.238031s, 3.138663s, 3.225993s, 3.245146s, 3.175455s, 3.110106s, 3.104300s, 3.192772s, 3.211830s, 3.117792s, 3.131220s, 3.146623s, 3.121459s, 3.208118s, 3.131979s, 3.177211s, 3.231641s, 3.167235s, 3.266262s, 3.152734s, 3.153420s, 3.245559s, 3.140877s} Average = 3.17638472s ≈ 3.18s

2.2.17 1000000000

The results for an input of 1000000000 are as follows:

C++: {3.21s, 3.19s, 3.24s, 3.08s, 3.18s, 3.27s, 3.07s, 3.32s, 3.29s, 3.10s, 3.08s, 3.10s, 3.17s, 3.08s, 3.16s, 3.21s, 3.27s, 3.25s, 3.15s, 3.20s, 3.16s, 3.21s, 3.22s, 3.25s, 3.26s} Average = 3.188800s ≈ 3.19s

Erlang: {3.437534s, 3.465171s, 3.507690s, 3.477691s, 3.466593s, 3.495241s, 3.45587s, 3.508733s, 3.422358s, 3.574212s, 3.505503s, 3.469876s, 3.461709s, 3.496081s, 3.499598s, 3.457680s, 3.464903s, 3.599175s, 3.410008s, 3.558394s, 3.521179s, 3.593183s, 3.451854s, 3.586535s, 3.574843s} Average = 3.4984646s ≈ 3.50s

2.3 Comparing loops with recursion

The two programs (007 in the corpus) used for these tests differ more than those used in the previous tests. The C++ program uses loops to rapidly load and display a BMP on an SDL surface, whereas the Erlang program uses recursion. The programs were tested with different inputs which denoted how many times the two BMPs were shown (an input of 1 would mean that each BMP was loaded once and shown once). The manner at which they were implemented is inefficient (it would be better to have three surfaces and flip between those rather than re-loading the BMPs into the same surface), but it was intended so for this test. The results are organised by their input argument.

2.3.1 10

As the programs in question display noticeable lag at inputs as low as 1, a much smaller section was taken for these tests. The results for an input of 10 are as follows:

C++: {5.27s, 5.34s, 5.22s, 5.26s, 5.17s, 5.24s, 5.14s, 5.33s, 5.53s, 5.28s, 5.50s, 5.28s, 5.25s, 5.33s, 5.29s, 5.52s, 5.23s, 5.19s, 5.34s, 5.22s, 5.13s, 5.18s, 5.29s, 5.28s, 5.20s, 5.27s, 5.87s} Average = 5.726000s $\approx$ 5.73s

Erlang: {5.210768s, 5.239860s, 5.492217s, 5.173358s, 5.287755s, 5.273984s, 5.116546s, 5.169300s, 5.405318s, 5.472248s, 5.260419s, 5.084197s, 5.486191s, 5.308423s, 5.294145s, 5.273074s, 5.235636s, 5.217542s, 5.287248s, 5.160886s, 5.184814s, 5.116996s, 5.058790s, 5.206066s, 5.269662s} Average = 5.2514177s $\approx$ 5.25s

As with most of tests prior to this set, the Erlang version of the program runs marginally quicker than the C++ program, despite having to call far more C++ and load libraries which aren't needed by the simpler program. The difference (0.48s), however, is far too small to be considered anything but negligible when compared to the runtime.

2.3.2 15

Considering the programs' high latency the inputs only increment by 5 for each test (rather than the much higher jumps of the previous set of tests). The results for an input of 15 are as follows:

C++: {5.79s, 6.10s, 5.78s, 5.85s, 5.85s, 5.82s, 5.73s, 5.97s, 5.72s, 5.75s, 5.83s, 5.80s, 6.06s, 6.04s, 6.12s, 5.72s, 6.01s, 5.89s, 5.73s, 5.79s, 5.78s, 6.01s, 5.82s, 6.02s, 6.26s} Average = 5.889600s $\approx$ 5.89s

Erlang: {5.656326s, 5.767737s, 5.806261s, 5.657844s, 5.546709s, 5.524563s, 5.691963s, 5.638994s, 5.629583s, 5.908933s, 5.807302s, 5.637159s, 5.789978s, 5.593515s, 5.808302s, 6.112678s, 5.798211s, 5.592811s, 5.742109s, 5.785664s, 5.902380s, 5.73489s, 5.630480s, 5.672904s, 5.607816s} Average = 5.7218044s $\approx$ 5.72s

For this Input Erlang remains marginally quicker then C++ at runtime. The difference is only 0.17s on average, which is negligible when compared to the actual run time.

2.3.3 20

The results for an input of 20 are as follows for each program:

C++: {6.68s, 6.44s, 6.35s, 6.13s, 6.42s, 6.45s, 6.76s, 6.50s, 6.20s, 6.44s, 6.29s, 6.31s, 6.19s, 6.52s, 6.46s, 6.35s, 6.67s, 6.56s, 6.45s, 6.36s, 6.43s, 6.23s, 6.56s, 6.11s, 6.25s} Average = 6.404400s ≈ 6.40s

Erlang: {6.363451s, 6.498057s, 6.063693s, 6.566908s, 6.532915s, 6.432372s, 6.152755s, 6.041723s, 6.339566s, 6.218376s, 6.117017s, 6.313924s, 6.328259s, 6.537618s, 6.357221s, 6.194826s, 6.077368s, 6.076106s, 6.162219s, 6.182674s, 6.200116s, 6.277479s, 6.14719s, 6.388148s} Average = 6.0227992s ≈ 6.02s

Erlang is still quicker than C++ at this input, by 0.38s on average.

2.3.4 25

An input of 25 yields the following results:

C++: {7.07s, 6.98s, 6.91s, 6.93s, 6.92s, 6.70s, 6.73s, 6.61s, 6.68s, 6.74s, 6.74s, 6.75s, 7.00s, 6.96s, 7.22s, 6.92s, 6.91s, 7.01s, 7.02s, 7.07s, 6.87s, 6.81s, 7.11s, 6.65s, 6.69s, 7.53s} Average = 7.181200s ≈ 7.18s

Erlang: {6.877998s, 6.843170s, 6.869665s, 6.863271s, 6.790267s, 6.805919s, 6.602797s, 6.750932s, 6.977972s, 6.775060s, 6.758403s, 7.040812s, 6.902265s, 6.792314s, 6.626248s, 6.994345s, 6.722562s, 7.188143s, 6.627187s, 6.767615s, 6.527603s, 6.595165s, 6.603223s, 7.030134s, 6.837249s} Average = 6.8068128s ≈ 6.81s

Erlang was, once again, slightly faster than it's C++ counterpart, with the average runtime being 0.37s quicker.

2.3.5 30

The final test was with an input of 30. At this point the run time of the programs was around 7 seconds. Below are the results for this input:

C++: {7.57s, 7.27s, 7.38s, 7.31s, 7.30s, 7.50s, 7.14s, 7.41s, 7.54s, 7.31s, 7.39s, 7.26s, 7.46s, 7.63s, 7.50s, 7.30s, 7.30s, 7.38s, 7.10s, 7.59s, 7.28s, 7.80s, 7.34s, 7.69s, 7.32s} Average = 7.402800s ≈ 7.40s

Erlang: {7.186412s, 7.135160s, 7.085418s, 7.238911s, 7.033844s, 7.577348s, 7.501673s, 7.246354s, 7.155420s, 7.229034s, 7.156600s, 7.385973s, 7.122461s, 7.671991s, 7.837022s, 7.516745s, 7.651781s, 7.427366s, 7.440862s, 7.290595s, 7.454447s, 7.635774s, 7.587689s, 7.278956s, 7.293623s} Average = 7.365658s ≈ 7.37s

The average runtimes of the two programs at an input of 30 are very close, with only a difference of 0.03s between them, making Erlang barely faster than C++ here.

3 Interpreting The Results

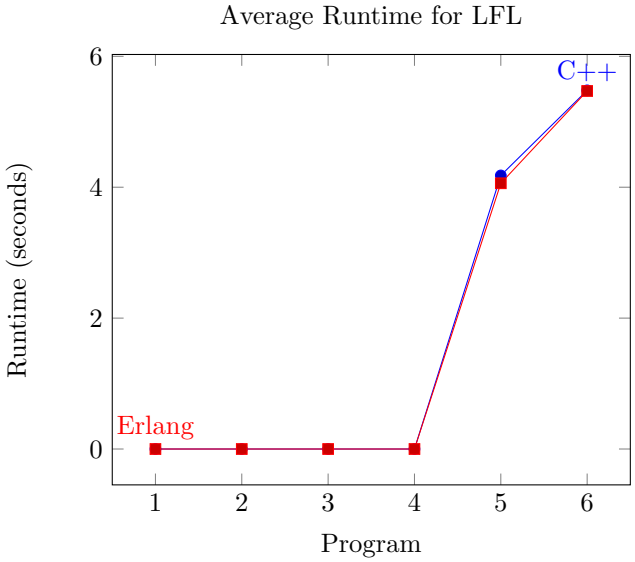
Although some of the tests yielded unusable results, especially for the smaller programs, they are included with the results that can be used. and have been

compiled into graph form below.

3.1 Like for Like programs

The average run times of the Erlang and C++ programs for each of the *Like for Like (LFL)* programs are below. The programs are referred to their numeric code within the corpus but for ease of reference they are:

*Hello World (001)
*Simple Arithmetic (002)
*Single Parameter (003)
*Factorial (004)
*Simple SDL (005)
*Extended SDL (006)



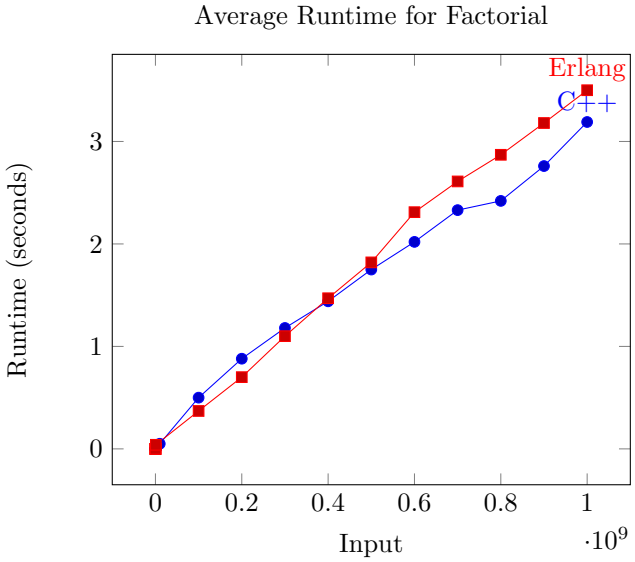
Test Program	C++ Average	Erlang Average
1	0	0
2	0	0
3	0	0
4	0	0
5	4.18	4.06
6	5.48	5.47

With the averages plotted above it is easy to see that the difference between runtime for the C++ and Erlang programs is barely noticeable in the data. For a human user these differences are imperceptible at this size of code base. These results could become more marked with a larger code base, but these tests can only show that, for simple programs, loading a shared object library

and executing code using NIFs does not impact the overall run time of these programs.

3.2 Factorial Programs

The *factorial* tests yielded some more interesting results, the averages for which are plotted on the graph and shown in the table below.



Input	C++ Average	Erlang Average
10	0	0
100	0	0
1,000	0	0
10,000	0	0
100,000	0	0
1,000,000	0	0
10,000,000	0.05	0.04
100,000,000	0.5	0.37
200,000,000	0.88	0.7
300,000,000	1.18	1.1
400,000,000	1.44	1.47
500,000,000	1.75	1.82
600,000,000	2.02	2.31
700,000,000	2.33	2.61
800,000,000	2.42	2.87
900,000,000	2.76	3.18
1,000,000,000	3.19	3.5

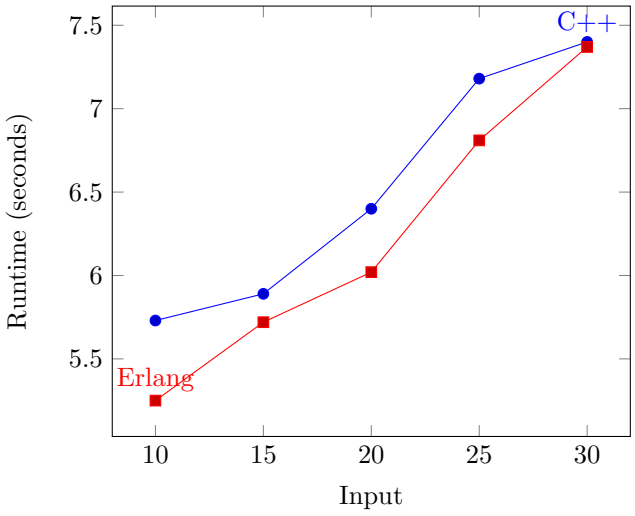
This time it is apparent that, for very large inputs ($>$ 400,000,000), the

Erlang (and NIF) implementation of the program ran slower than the pure C++ version. This phenomenon is reversed for smaller inputs. To be clear, the Erlang program takes an input and then calls a C++ factorial function through the use of a NIF, so the majority of the work is still done by the C++. A possible explanation of this pattern is the potential latency caused by running both the Erlang program inside the Beam virtual machine, and the shared object library containing the NIF at the same time. However, further investigation would be needed to confirm this.

3.3 Loops Vs. Recursion

The following graph and table show the results for the *Loops Vs. Recursion* test. This involved two programs written using SDL (the classic library for the C++ implementation and the wrapped library created for this project) to cycle between two BMP images x many times (x being the input). The C++ program used a while loop to do this, whereas the Erlang program used recursion.

Average Runtime for Loops Vs. Recursion



Input	C++ Average	Erlang Average
10	5.73	5.25
15	5.89	5.72
20	6.4	6.02
25	7.18	6.81
30	7.4	7.37

The above graph shows that, on average, the Erlang program was faster than it's C++ counterpart for every input (although they do come very close for the last input). Although further investigation is needed for this case, it is possible to draw a tentative conclusion that recursion may be faster than loops

for these types of applications, making Erlang a better suited language to write such things in.

4 Conclusion

Although the data in this document is very limited and could not be considered a decisive conclusion on any of the questions raised, it is possible to make some observations.

Are Erlang NIFs inherently slower than C++? In short, no. We have seen through some very simple examples that Erlang using NIFs can, in fact be quicker than almost identical C++. The margins are so slight though that this runtime difference makes a negligible impact.

Is the overhead of loading a shared object library too great to consider NIFs viable? Again, the results above indicate that is not the case. During the factorial test it emerged that there was some other factor at play affecting the change in results for higher inputs. This could be due to external factors, or possibly running both programs at the same time. The averages for lower inputs show that there does not appear to be any discernible overhead when loading a NIF.

Could Erlang using NIFs be considered a better choice than pure C++? The results of these experiments are contradictory on this particular point. Yes, Erlang generally performs better when being used to it's strengths. Recursion usually considered is more powerful than loops. Although it is not so easy to write recursion in C or C++, beginners are encouraged to get used to writing recursion in Erlang, thus it becomes second nature quickly. This leads to better programming habits, and generally more efficient and cleaner code. However we have seen that there may be complications with using Erlang for certain applications, such as with the high inputs in the factorial tests

These results are inconclusive, and more rigorous testing needs to be done surrounding this subject. These experiments were meant to simply illuminate the potential latency caused by using NIFs, and it is possible that the patterns seen in the data above could expand given bigger data and a much larger code base to work with.

All of the programs used in these tests, as well as the raw data, are available as part of the project corpus.