Appendix D - Performance

The text talks about performance in § 1.3, 2B, 4D, 6.2.4, 6B.2, 10 intro, 11.1, 17.6.3, 18 intro, 19.5, 20.1.2, 22.1.1, 26.5.4, 26.5.6, 28 intro, 29.1, 30.4.6, 33.4, 34.3.4.

Goal: Excellent performance, on par with C++,
similar to C, or sometimes even faster.

Benchmarks: see § 1.3
2023 Jan – Chess game: Jai was some 4-8 % slower than C

This is partly achieved by having:

- NO garbage collection (GC)
- NO automatic memory management
- NO exceptions (they are too complex, weighs too heavy on performance)
- NO RAII (Resource Acquisition Is Initialization), like: a struct has to have a copy constructor, move constructor, iterator, and so on, which leads to high friction
- temporary storage, which is much faster than malloc
- the context resides in cache

Jai has these characteristics to increase performance:
1) LLVM optimizations
2) boolean operators && and || are short-circuited.
3) strings are immutable, not '0'-terminated
4) arrays are built into the compiler (very efficiently (contiguously) stored, *on the stack* for small arrays so they are very fast
5) developer has complete control over memory allocation: packing (alignment/padding)
for example: struct memory layout: fields are contiguous, packed together or aligned

To increase performance you can make use of:
1) inlining procedures with `inline`
2) disable assertions: `#import "Basic"()(ENABLE_ASSERT=false);` (see § 6.1.3 and § 20.1.2)
3) use --- to avoid default initialization of variables.
4) turn off cast bound checks at runtime: `cast,`**`no_check`**`(type) var;`
5) use SOA (struct of arrays) instead of AOS (array of structs) (see § 26.10)
6) if you only need to print simple strings or numbers, use write_* procs instead of print. (see § 5.7.2)
7) use a String_Builder to accumulate a lot of strings. (see § 19.5)
8) turn off array bounds checking with **#no_abc** (see § 18.3.2)
9) disable stack-trace on crash: setting **Compiler.Build_Options.backtrace_on_crash** .**OFF** will turn off the crash handler (and cause less code to get imported when your program is built). (see § 30.4.5)
modules/Default_Metaprogram now handles the argument **-no_backtrace_on_crash**, which will cause the crash handler not to be loaded.
10) set build option `runtime_storageless_type_info` to true (see § 30.4.7)
11) use relative pointers (see § 10.6)
12) cast the index of a for (normally type s64) to a smaller integer type
13) alignment of fields in structs (see § 12.11)
14) when defining large arrays, use 64-bit cache alignment: (see § 18.3.4), for example:
array := NewArray(500, int, alignment=64);`
15) use e.g. enum u16 type instead of enum (which is 64bit)

16) for faster memory management: keep things on structs by value where possible.
17) it is better to return things from a procedure by value; this avoid having extra stack copies like in C.

<u>Jai compiler command-line options for performance:</u>

-release Build a release build, i.e., tell the default metaprogram to disable stack traces and enable optimizations.

-no_dce	Turn off dead code elimination.
-no_check	Do not import modules/Check and run it on the code.
-no_check_bindings	Disable checking of module bindings when running modules/Check.

<u>Options in a build metaprogram for a release build</u> (disabling checks and decrease output size):
(In a build program the full command will be like: target_options.optimization_level = .RELEASE;)

```
optimization_level = .RELEASE;
```
set_optimization_level(*target_options, 2, 0);

```
stack_trace = false;
backtrace_on_crash = .OFF;
array_bounds_check = .OFF;
cast_bounds_check  = .OFF;
math_bounds_check = .OFF;
null_pointer_check = .OFF;
runtime_storageless_type_info = true;
emit_debug_info=.NONE;                    //  (no .pdb files are created)
write_added_strings = false;
dead_code_elimination = .ALL;
 shorten_filenames_in_error_messages = true;
 lazy_foreign_function_lookups = true;
 enable_bytecode_inliner = true;
 enable_frame_pointers = false;
 relative_pointer_bounds_check = .OFF
```

<u>LLVM-options:</u>
(In a build program the full command will be like:
target_options.llvm_options.gen_optimization_level = 3;)

.gen_optimization_level = 3;
```
.enable_tail_calls = false;
.enable_loop_unrolling = false;
.enable_slp_vectorization = false;
.enable_loop_vectorization = false;
.reroll_loop = false;
.verify_input = false;
.verify_output = false;
.merge_functions = false;
.disable_inlining = true;
.disable_mem2reg = false;
.enable_split_modules = false;
```
(See also Llvm_options / X64_Options: § 30.4.8)

Choosing between a debug or release build:

See 30.4.9 / 30.10

How to measuring performance:

- using get_time: see § 6B.2, or with a macro: see § 26.5.6.