**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.

Jai compiler command-line options for performance:

**-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.

Options in a build metaprogram for a release build (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

LLVM-options:

(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.