Kludging The editor with The compiler

Andrea Corallo
<akrl@sdf.org>
What GNU Emacs is

- It's a Lisp implementation (Emacs Lisp)
- Its task is to slurp unstructured text from the OS
- Lisp programs can represent, manipulate, and share these data
Emacs Lisp

- Continuously improving
- Capable of (almost) any task
- Surprisingly spread
  - Emacs 1.794.561 LOC
  - emacsmirror.net 9.888.547 LOC!!
  - 21th in push number (https://madnight.github.io)
Lisp

- Dynamic
- Homoiconic
- Maxwell’s equations of software (Alan Kay)

- Easy to learn... easy to implement!
~1500
Implementation

- ~30% is C
- Lisp Interpreted or byte-compiled
- Byte compiler is written in Elisp
- Must bootstrap!
- Byte-code runs on a stack-based VM
Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debuggability and compile time errors
Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debbuggability and compile time errors
Where to improve

- Namespace
- Extensibility
- Performance
  - Garbage Collector
  - Execution Engine
  - Real multi-threading
- Debuggability and compile time errors
How to improve

Improving the Lisp performance allow for:

- Less C to be written and maintained
- Ease write of performance critical extensions
Lisp Objects

Lisp_Object
Tagged pointer

Fixnum

Header
Real Lisp Object

Lisp_Object
Tag bits

Tagged pointer
Tag bits
Elisp VM

A stack base push and pop VM

• Lisp
  
  \[(\ast \ (+ \ a \ 2) \ 3)\]

• LAP
  
  (byte-varref \ a)
  (byte-constant \ 2)
  (byte-plus)
  (byte-constant \ 3)
  (byte-mult)
  (byte-return)
Elisp VM

- LAP
  (byte-varref a)
  (byte-constant 2)
  (byte-plus)
  (byte-constant 3)
  (byte-mult)
  (byte-return)
Elisp VM

- LAP
  (byte-varref a) <=
  (byte-constant 2)
  (byte-plus)
  (byte-constant 3)
  (byte-mult)
  (byte-return)
Elisp VM

- LAP
  (byte-varref a)
  (byte-constant 2) <=
  (byte-plus)
  (byte-constant 3)
  (byte-mult)
  (byte-return)
Elisp VM

- LAP
  (byte-varref a)
  (byte-constant 2)
  (byte-plus) <=
  (byte-constant 3)
  (byte-mult)
  (byte-return)
Elisp VM

• LAP
  (byte-varref a)
  (byte-constant 2)
  (byte-plus)
  (byte-constant 3) <=
  (byte-mult)
  (byte-return)
Elisp VM

• LAP
  (byte-varref a)
  (byte-constant 2)
  (byte-plus)
  (byte-constant 3)
  (byte-mult) <=
  (byte-return)
Elisp VM

- LAP
  (byte-varref a)
  (byte-constant 2)
  (byte-plus)
  (byte-constant 3)
  (byte-mult)
  (byte-return) <=
Byte-compiler pipeline

- Macro expansion
- Closure conversion
- Source level optimizations
- Single pass byte-compiler => LAP (Lisp Assembly Program)
- Peephole LAP optimizations
- Assembled into byte-code
Elisp → LAP → Byte-code (.elc)
Elisp → LAP → ? → Libgccjit IR → GCC...
libgccjit

• Added by David Malcolm in GCC 5
• Describe programmatically a C-ish semantic
• Good for Jitters or AoT compilers
A simple translation

**LAP**

- (byte-varref a)
- (byte-constant 2)
- (byte-plus)
- (byte-constant 3)
- (byte-mult)
- (byte-return)

**C**

```c
Lisp_Object local[2];

local[0] = varref (a);
local[1] = two;

local[0] = plus (local[0], local[1]);
local[1] = three;

local[0] = mult (local[0], local[1]);
return local[0];
```
Optimizing outside GCC

- Generate code effectively optimizable
- Provide user feedback
  - warning
  - errors
  - optimizations hints
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Lisp

C
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

LAP

LIMPLE

Libgccjit IR
Native compiler pipeline

Run the byte-compiler infrastructure to obtain LAP

- spill-lap
- llimplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final
Native compiler pipeline

Convert LAP into LIMPLE
- LIMPLE as tribute to GIMPLE
- CFG based
- SSA

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final
Native compiler pipeline

- Forward propagate types and values
- Execute in the run-time pure functions

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Elisp

foo

Emacs core

funcall

message
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Emacs core

funcall

message

Elisp

foo
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Elisp

foo

bar

Emacs core

funcall
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Elisp

foo

bar

Emacs core

funcall

Allow GCC IPA logic
Native compiler pipeline

- Infer function purity

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
-fwprop
- remove-type-hints
- final
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Tail Recursion Elimination
- Pattern match and replace recursive calls in tail position
Native compiler pipeline

- spill-lap
- limplify
- fwprop
- call-optim
- ipa-pure
- fwprop
- dead-code
- tco
- fwprop
- remove-type-hints
- final

Convert LIMPLE into libgccjit IR
- Define inline functions to access fundamental data types
- Use type information for code generation
Extending the language

- speed [0-3]
- Compilation unit
- Compiler hints

Borrowed from CL

- Allow for some cheating at 3
Extending the language

- First class object
  - Allow for the GC to handle loaded functions

- speed [0-3]
- Compilation unit
- Compiler hints
Compilation Unit

- Symbol foo1
  - Function 1
  - CU1
    - Function 1
    - Function 2

- Symbol foo2
  - Function 2
  - CU2
    - Function 3
    - Function 4

- Symbol bar1
  - Function 3

- Symbol bar2
  - Function 4
Compilation Unit

- GC root
  - Symbol foo1
    - Function 1
  - Symbol foo2
    - Function 2
  - Symbol bar1
  - Symbol bar2

CU1
  - Function 1
  - Function 2
Extending the language

- Suggest a type for an expression
  - Assertion or hint for removing the type check

(setf x (1+ x))

(setf x (comp-hint-fixnum (1+ x)))
Jit vs AoT

- Born as an Ahead of Time compiler
- Moved to a Hybrid approach
Async compilation

- Jit like triggered
- Parallel
- Output reused between different sessions
- Definitions hot-swap
Async compilation

byte-code load

native compilation

definition hot-swap

t0

native compilation

t1

t
Async compilation
## Performance

<table>
<thead>
<tr>
<th>test</th>
<th>runtime (s)</th>
<th>runtime (s)</th>
<th>perf uplift</th>
</tr>
</thead>
<tbody>
<tr>
<td>fibn-rec</td>
<td>7.24</td>
<td>0.00</td>
<td>---</td>
</tr>
<tr>
<td>fibn-tc</td>
<td>6.40</td>
<td>0.01</td>
<td>---</td>
</tr>
<tr>
<td>fibn</td>
<td>11.91</td>
<td>0.00</td>
<td>---</td>
</tr>
<tr>
<td>listlen-tc</td>
<td>8.81</td>
<td>0.21</td>
<td>42.0x</td>
</tr>
<tr>
<td>nbody</td>
<td>17.10</td>
<td>2.41</td>
<td>7.1x</td>
</tr>
<tr>
<td>inclist</td>
<td>15.01</td>
<td>2.39</td>
<td>6.3x</td>
</tr>
<tr>
<td>bubble</td>
<td>12.13</td>
<td>2.55</td>
<td>4.8x</td>
</tr>
<tr>
<td>pcase</td>
<td>11.93</td>
<td>2.70</td>
<td>4.4x</td>
</tr>
<tr>
<td>flet</td>
<td>12.15</td>
<td>3.58</td>
<td>3.4x</td>
</tr>
<tr>
<td>bubble-no-cons</td>
<td>10.81</td>
<td>3.64</td>
<td>3.0x</td>
</tr>
<tr>
<td>dhrystone</td>
<td>8.78</td>
<td>3.63</td>
<td>2.4x</td>
</tr>
<tr>
<td>pidigits</td>
<td>14.51</td>
<td>9.47</td>
<td>1.5x</td>
</tr>
<tr>
<td>map-closure</td>
<td>9.33</td>
<td>9.32</td>
<td>1.0x</td>
</tr>
<tr>
<td>total</td>
<td>146.11</td>
<td>39.92</td>
<td>3.7x</td>
</tr>
</tbody>
</table>

[https://elpa.gnu.org/packages/elisp-benchmarks.html]
libgccjit take aways

Works for us!

- Compile time is okay
- Leaks memory
- How to expose easily more accessors? LTO?
- Distros may fix their packages
Project Status

• Exists and it is usable
  emacs.git feature/native-comp
  ./configure --with-nativecomp
  M-x report-emacs-bug
• Focusing on integration and consolidating
• Maybe in Emacs 28?
  <http://akrl.sdf.org/gcemacs.html>
  <emacs-devel@gnu.org>
Project Status

- **Exists** and it is usable
  
  `emacs.git feature/native-comp`
  
  ```
  ./configure --with-nativecomp
  ```
  
  `M-x report-emacs-bug`

- Focusing on integration and consolidating
- Maybe in Emacs 28?


<emacs-devel@gnu.org>