Why a GHC Whole Program Compiler Mode Would Be Useful

Csaba Hruska

https://github.com/grin-compiler/ghc-whole-program-compiler-project

What is Whole Program Compilation?

WPC = access the whole program IR *(Intermediate Representation)*

WPC doesn't always mean Whole Program Analysis (WPA) & Optimization (WPO)

Incremental compilers do WPC at the linker level:

- Incremental Compiler: **Surface Language**
- Whole Program Compiler: **Object Code Linker**
  I.e. linker can remove dead sections **(WPO)**

It's always WPC for some program IR!

Why is WPC good? What is WPC good for?

Observability

Research

Static / Dynamic analysis
Optimization
Runtime system design
Compiler pipeline design, i.e. JIT
Language frontend / backend
Learn & understand

Industry

Debug / Profile
Test
Evidence based optimization
Tooling development
Measure
Dependency analysis

Why does GHC need WPC mode?

WPC mode *(whole program IR access)* unlocks new design and development methods

GHC DX ≠ Haskell DX
It's easy to write Haskell apps, but it's hard to develop GHC.

Many Haskell programmer is interested in compiler development,
yet there are orders of magnitude less GHC & tooling devs. Why?

Follow your favourite development method without the GHC conventions
Get the Haskell App Developer Experience (DX)

Non-GHC Haskell compilation history: *JHC, Intel Labs HRC/FLRC, Lambdachine JIT, UHC, LHC*

GHC WPC Patch Design

Principles

Simplicity

Maintainability

Use standard tech

No interference with GHC design

Requirements

GHC as WPC frontend, no new compiler CLI

Should work with tooling: Cabal, Stack

Export all information to run the program
or compile to executable

GHC WPC Patch Details

New GHC options: *-wpc, -no-wpc*

New file output in *-wpc* mode:

.modpak (zip) - compiler flags, pretty printed IRs, binary IRs (stg, core), FFI stubs
.ghc_stgapp (yaml) - Haskell deps, foreign deps, linker options
all_stubs.a, capi_stubs.a, cbits.a - C related object code from Cabal or GHC FFI generated

New GHC foundation lib:

external-stg-ir - depends on: base, binary, bytestring

Cabal installs *.modpak, cbits.a, stubs.a* files for packages

GHC Distribution:

GHC bindist tar.xz (~200MB) - original, not changed
WPCPak.zip (~1.2GB, only for wpc mode) - *.modpak, stub.a, cbits.a* files for foundation libs

FAQ & Concerns

Doesn’t GHC already support this via plugins? No.

Plugin API locks to GHC & Cabal pipeline design,
also it doesn’t expose enough info to generate executable later.

What about maintenance and compatibility costs?

STG to External STG conversion is a best effort approach.
It’s ok to have breaking changes between releases.

GHC had external core in the past, but was removed due to maintenance costs.

STG changes less frequently than Core.

Vision, Ideas, Experience

Haskell compiler & tooling should be developed by its users.

Users have different priority order:

production code:	*cpu/mem performance, simplicity*
research / observe / debug:	*simplicity, cpu/mem perf*

Let Haskell users to solve their own issues in their own way.

Increase compiler & tooling guru count to 1% of all Haskell programmers

1% of (30K-100K) = 300-1000

I.e Idris has several user developed backends, because it is easy to do

WPC use case: IR interpreter ideas

Projects

- External STG Interpreter (ExtSTGi), *talk: Why and How the External STG Interpreter is Useful*
- ExtSTGi based debugger & profiler
- Idris2 on External STG with zero-cost Haskell interop
- Unified Concrete & Abstract Interpreter for STG *(semantics definition + static analysis)*

Ideas

- ExtSTGi based Template Haskell evaluator GHC
- GHC Core interpreter, type lambda profiler
- JIT support for ExtSTGi
- New native backend reusing Cmm & RTS
- Staged compilation based RTS from ExtSTGi *(calculating correct compilers)*
- Debug Adapter Protocol for ExtSTGi

WPC use case: GHC remix

Build different compiler pipeline from unmodified GHC components
e.g. *analyses, core optimizer, codegen, RTS*

GHC is stuck in local optimum

Ideas to explore

- profile guided inliner (using Core/STG interpreter)
- reimplement GHC analyses in Datalog
- develop new analyses in Datalog
- profile guided hot code selection for code optimization & specialization
- detailed optimization effect visualization

Conclusion

Unleash the creative power
of all Haskell programmers in
Haskell compiler & tooling development

Compiler & Tooling DX = App DX

https://github.com/grin-compiler/ghc-whole-program-compiler-project