LTO: History and work to be done

Rafael Ávila de Espíndola

World Wide Studios - Sony Computer Entertainment

2014-04-07

Very quick history overview

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)
- We wrote plugins for native tools (libLTO.dylib, LLVMgold.so)

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)
- We wrote plugins for native tools (libLTO.dylib, LLVMgold.so)
- We are writing new tools (new llvm-nm and llvm-ar, lld, . . .)

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)
- We wrote plugins for native tools (libLTO.dylib, LLVMgold.so)
- We are writing new tools (new llvm-nm and llvm-ar, lld, . . .)
- Are we going back and forth?!?

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)
- We wrote plugins for native tools (libLTO.dylib, LLVMgold.so)
- We are writing new tools (new llvm-nm and llvm-ar, lld, . . .)
- Are we going back and forth?!?
- Yes!

Very quick history overview

- We had llvm specific tools (llvm-nm, llvm-ar, llvm-ld, . . .)
- We wrote plugins for native tools (libLTO.dylib, LLVMgold.so)
- We are writing new tools (new llvm-nm and llvm-ar, lld, . . .)
- Are we going back and forth?!?
- Yes!
- Probably the correct choice at every point in time.

Early tools

Early tools

- They were llvm only.

Early tools

- They were llvm only.
- Easy to write. Important for a small (at the time) project.

Early tools

- They were llvm only.
- Easy to write. Important for a small (at the time) project.
- llvm-link was “95% persent working” in October 2001.

Early tools

- They were llvm only.
- Easy to write. Important for a small (at the time) project.
- llvm-link was “95% persent working” in October 2001.
- Combine with opt and we already had LTO!

Evolution of the early tools

Evolution of the early tools

- Adding llvm-link, opt and llc to a build is a pain.

Evolution of the early tools

- Adding llvm-link, opt and llc to a build is a pain.
- We added gccld to help in January 2002. It hides the extra build steps. Produces a shell script that runs lli.

Evolution of the early tools

- Adding llvm-link, opt and llc to a build is a pain.
- We added gccld to help in January 2002. It hides the extra build steps. Produces a shell script that runs lli.
- Libraries can be created with llvm-ar since August 2003.

Evolution of the early tools

- Adding llvm-link, opt and llc to a build is a pain.
- We added gccld to help in January 2002. It hides the extra build steps. Produces a shell script that runs lli.
- Libraries can be created with llvm-ar since August 2003.
- Native codegen added to gccld in September 2003).

Evolution of the early tools

- Adding llvm-link, opt and llc to a build is a pain.
- We added gccld to help in January 2002. It hides the extra build steps. Produces a shell script that runs lli.
- Libraries can be created with llvm-ar since August 2003.
- Native codegen added to gccld in September 2003).
- gccld is renamed llvm-ld in r16305 September 2004.

Some limitations remained

Some limitations remained

- They don't know about native object files. Have to be conservative.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.
- To assemble and link, llvm-ld runs gcc.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.
- To assemble and link, llvm-ld runs gcc.
- Semantics not exactly right.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.
- To assemble and link, llvm-ld runs gcc.
- Semantics not exactly right.
- Slow.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.
- To assemble and link, llvm-ld runs gcc.
- Semantics not exactly right.
- Slow.
- Build changes are still required.

Some limitations remained

- They don't know about native object files. Have to be conservative.
- The archive symbol table only lists llvm members.
- To assemble and link, llvm-ld runs gcc.
- Semantics not exactly right.
- Slow.
- Build changes are still required.
- Hacks everywhere.

The native tools

The native tools

- Native tools (ld, ar, ranlib . . .) know the command line args.

The native tools

- Native tools (ld, ar, ranlib . . .) know the command line args.
- They know all semantic corner cases.

The native tools

- Native tools (ld, ar, ranlib . . .) know the command line args.
- They know all semantic corner cases.
- They know the library format.

The native tools

- Native tools (ld, ar, ranlib . . .) know the command line args.
- They know all semantic corner cases.
- They know the library format.
- Build systems are already using them!

The native tools

- Native tools (ld, ar, ranlib . . .) know the command line args.
- They know all semantic corner cases.
- They know the library format.
- Build systems are already using them!
- They know all there is to know about LTO, except llvm IR.

The plugin era

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`
- gold got plugin support in September 2008.

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`
- gold got plugin support in September 2008.
- `tools/gold added in February 2009.`

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`
- gold got plugin support in September 2008.
- `tools/gold added in February 2009.`
- bfd (nm, ar) gets plugin support in May 2009.

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`
- gold got plugin support in September 2008.
- `tools/gold added in February 2009.`
- bfd (nm, ar) gets plugin support in May 2009.
- bfd (ld) gets plugin support in October 2010.

The plugin era

- Idea: Use the native tools, but have them ask llvm for help.
- First implemented in ld64.
- `tools/lto added in August 2006.`
- gold got plugin support in September 2008.
- `tools/gold added in February 2009.`
- bfd (nm, ar) gets plugin support in May 2009.
- bfd (ld) gets plugin support in October 2010.
- We can now do LTO by setting CFLAGS, CXXFLAGS and LDFLAGS!

The llvm project around 2009

The llvm project around 2009

- llvm had become a good compiler toolkit:

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of
  - Assemblers

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of
  - Assemblers
  - Linkers

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of
  - Assemblers
  - Linkers
  - Debuggers

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of
  - Assemblers
  - Linkers
  - Debuggers
  - ...

The llvm project around 2009

- llvm had become a good compiler toolkit:
  - High quality.
  - Liberal license.
  - Modular.
  - ...
- That is not enough!
- These are also desirabilities of
  - Assemblers
  - Linkers
  - Debuggers
  - ...
- We need a toolchain toolkit!

The project expands to other areas

The project expands to other areas

- `lib/MC added in June 2009.`

The project expands to other areas

- `lib/MC` added in June 2009.
- `libcxx` added in r103490 May 2010.

The project expands to other areas

- `lib/MC added in June 2009.`
- `libcxx added in r103490 May 2010.`
- `lldb added in r105617 June 2010.`

The project expands to other areas

- `lib/MC added in June 2009.`
- `libcxx added in r103490 May 2010.`
- `lldb added in r105617 June 2010.`
- `lib/Object added in r119107 November 2010.`

The project expands to other areas

- `lib/MC added in June 2009.`
- `libcxx added in r103490 May 2010.`
- `lldb added in r105617 June 2010.`
- `lib/Object added in r119107 November 2010.`
- lld added in r146598 December 2011 (was named lold).

The new tools and LTO

The new tools and LTO

- We are building a complete toolchain.

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!
- Very different tools form the original tools:

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!
- Very different tools form the original tools:
  - They are not llvm only. ELF, COFF and MachO work.

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!
- Very different tools form the original tools:
  - They are not llvm only. ELF, COFF and MachO work.
  - In archives, IR files go in the same symbol table.

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!
- Very different tools form the original tools:
  - They are not llvm only. ELF, COFF and MachO work.
  - In archives, IR files go in the same symbol table.
  - Implement the same semantics and options as the native tools.

The new tools and LTO

- We are building a complete toolchain.
- Lets make sure it supports LTO!
- Very different tools form the original tools:
  - They are not llvm only. ELF, COFF and MachO work.
  - In archives, IR files go in the same symbol table.
  - Implement the same semantics and options as the native tools.
  - Should still be able to do LTO by setting CFLAGS, CXXFLAGS and LDFLAGS.

Recent developments

Decided to start with llvm-ar and make it generic.

The first expected items

The first expected items

- lvm-ar was rewritten to use `lib/Object`.

The first expected items

- lvm-ar was rewritten to use `lib/Object`.
- Removed `lib/Archive`.

The first expected items

- llvm-ar was rewritten to use `lib/Object`.
- Removed `lib/Archive`.
- llvm-ar could then create native symbol tables!

The first expected items

- llvm-ar was rewritten to use `lib/Object`.
- Removed `lib/Archive`.
- llvm-ar could then create native symbol tables!
- And they include native object formats!

The first expected items

- llvm-ar was rewritten to use `lib/Object`.
- Removed `lib/Archive`.
- llvm-ar could then create native symbol tables!
- And they include native object formats!
- Pretty much done by July 2013.

The first expected items

- lvm-ar was rewritten to use `lib/Object`.
- Removed `lib/Archive`.
- llvm-ar could then create native symbol tables!
- And they include native object formats!
- Pretty much done by July 2013.
- Then I "only" needed to add IR support to `lib/Object`.

Yak shaving, part 1

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.
- llvm-ar should not depend on `lib/Target`.

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.
- llvm-ar should not depend on `lib/Target`.
- The mangler had to move from `lib/Target to lib/IR`.

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.
- llvm-ar should not depend on `lib/Target`.
- The mangler had to move from `lib/Target to lib/IR`.
- And the mangling had to be specified in DataLayout.

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.
- llvm-ar should not depend on `lib/Target`.
- The mangler had to move from `lib/Target to lib/IR`.
- And the mangling had to be specified in DataLayout.
- The DataLayout specs were out of sync in llvm and clang.

Yak shaving, part 1

- The symbol table in archives uses the final (mangled) name.
- llvm-ar should not depend on `lib/Target`.
- The mangler had to move from `lib/Target to lib/IR`.
- And the mangling had to be specified in DataLayout.
- The DataLayout specs were out of sync in llvm and clang.
- Move completed in Jan 2014.

Yak shaving, part 2

Yak shaving, part 2

- The ObjectFile interface is huge.

Yak shaving, part 2

- The ObjectFile interface is huge.
- Has to handle symbols, sections, segments, relocations, ...

Yak shaving, part 2

- The ObjectFile interface is huge.
- Has to handle symbols, sections, segments, relocations, ...
- For IR, only symbols make sense.

Yak shaving, part 2

- The ObjectFile interface is huge.
- Has to handle symbols, sections, segments, relocations, ...
- For IR, only symbols make sense.
- Added a new SymbolicFile with a simpler interface.

Yak shaving, part 2

- The ObjectFile interface is huge.
- Has to handle symbols, sections, segments, relocations, ...
- For IR, only symbols make sense.
- Added a new SymbolicFile with a simpler interface.
- An ObjectFile is a SymbolicFile.

Yak shaving, part 2

- The ObjectFile interface is huge.
- Has to handle symbols, sections, segments, relocations, ...
- For IR, only symbols make sense.
- Added a new SymbolicFile with a simpler interface.
- An ObjectFile is a SymbolicFile.
- Finally implemented IRObjectFile in February 2014.

Yak shaving, part 3

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!
- But DataLayout is an ImmutablePass.

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!
- But DataLayout is an ImmutablePass.
- And llvm-ar has no pass manager :-(

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!
- But DataLayout is an ImmutablePass.
- And llvm-ar has no pass manager :-(
- Split DataLayout into DataLayoutPass and DataLayout.

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!
- But DataLayout is an ImmutablePass.
- And llvm-ar has no pass manager :-(
- Split DataLayout into DataLayoutPass and DataLayout.
- Split finished in February 2014.

Yak shaving, part 3

- We could then mangle a GlobalValue's name using DataLayout!
- But DataLayout is an ImmutablePass.
- And llvm-ar has no pass manager :-(
- Split DataLayout into DataLayoutPass and DataLayout.
- Split finished in February 2014.
- The symbol tables are now correct and include IR files!

Work to be done

Finish llvm-ar

Finish llvm-ar

- We need to finish it so that it can be used on any build.

Finish llvm-ar

- We need to finish it so that it can be used on any build.
- Support for non-gnu format is missing.

Finish llvm-ar

- We need to finish it so that it can be used on any build.
- Support for non-gnu format is missing.
- BFD format should make it possible to align objects!

Finish llvm-ar

- We need to finish it so that it can be used on any build.
- Support for non-gnu format is missing.
- BFD format should make it possible to align objects!
- Thin archives are missing.

Finish llvm-ar

- We need to finish it so that it can be used on any build.
- Support for non-gnu format is missing.
- BFD format should make it possible to align objects!
- Thin archives are missing.
- File handling could use open, ftruncate, mmap.

Finish llvm-ar

- We need to finish it so that it can be used on any build.
- Support for non-gnu format is missing.
- BFD format should make it possible to align objects!
- Thin archives are missing.
- File handling could use open, ftruncate, mmap.
- Hopefully no yaks.

Some inefficiencies of IR handling in llvm-ar

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`
- `lib/Object doesn't know what it is being used for.`

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`
- `lib/Object doesn't know what it is being used for.`
- `lib/Object uses lib/IR to find out.`

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`
- `lib/Object doesn't know what it is being used for.`
- `lib/Object uses lib/IR to find out.`
- `lib/IR doesn't know what it is being used for.`

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`
- `lib/Object doesn't know what it is being used for.`
- `lib/Object uses lib/IR to find out.`
- `lib/IR doesn't know what it is being used for.`
- `lib/IR parses the entire file.`

Some inefficiencies of IR handling in llvm-ar

- llvm-ar needs the name of global symbols.
- llvm-ar uses `lib/Object to find out.`
- `lib/Object doesn't know what it is being used for.`
- `lib/Object uses lib/IR to find out.`
- `lib/IR doesn't know what it is being used for.`
- `lib/IR parses the entire file.`
- Using a lazy module helps, but still reads metadata.

Making `lib/Object` lazier

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.
- Implement it directly with `lib/Bitcode`.

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.
- Implement it directly with `lib/Bitcode`.
- llvm-ar would never even create a Module.

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.
- Implement it directly with `lib/Bitcode`.
- llvm-ar would never even create a Module.
- Code that needs more info can create a Module and query.

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.
- Implement it directly with `lib/Bitcode`.
- llvm-ar would never even create a Module.
- Code that needs more info can create a Module and query.
  - llvm-nm: Not performance critical.

Making `lib/Object` lazier

- Reading only the names and DataLayout is a special case.
- Implement it directly with `lib/Bitcode`.
- llvm-ar would never even create a Module.
- Code that needs more info can create a Module and query.
  - llvm-nm: Not performance critical.
  - LTO: Can delay creating a Module in some cases.

How linkers view LTO

LTO according to ld64

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.
    - C++ inline function whose address is not taken.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.
    - C++ inline function whose address is not taken.
  - Requires loading all function bodies.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.
    - C++ inline function whose address is not taken.
  - Requires loading all function bodies.
- Does symbol resolution.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.
    - C++ inline function whose address is not taken.
  - Requires loading all function bodies.
- Does symbol resolution.
- Decides which symbols llvm should keep.

LTO according to ld64

- ld64 uses libLTO to create Modules for all IR files.
- ld64 requests the list of all symbols in each module
  - Really all symbols, even uses and defs in inline assembly.
  - Asks whether a symbol can be hidden form the symbol table.
    - C++ inline function whose address is not taken.
  - Requires loading all function bodies.
- Does symbol resolution.
- Decides which symbols llvm should keep.
- libLTO creates a MachO file.

LTO according to gold

LTO according to gold

- Requests the list of global symbols of each file.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.
  - Symbol was preempted.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.
  - Symbol was preempted.
  - ...

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.
  - Symbol was preempted.
  - ...
- Plugin decides which symbols to keep and creates an ELF file.

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.
  - Symbol was preempted.
  - ...
- Plugin decides which symbols to keep and creates an ELF file.
- Only now do we need a Module!

LTO according to gold

- Requests the list of global symbols of each file.
  - Defined/Undefined/Weak/...
  - Visibility
  - Comdat Key
- Does symbol resolution. May close some files.
- Notifies the plugin of the result for each symbol.
  - Symbol is used from an ELF or BC file.
  - Symbol is used by the symbol table.
  - Symbol was preempted.
  - ...
- Plugin decides which symbols to keep and creates an ELF file.
- Only now do we need a Module!
- Can trim the Module before IR linking!

Reorganizing lib/LTO, tools/gold and tools/lto

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold
  - Implemented on top of lib/Object and lib/LTO.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold
  - Implemented on top of lib/Object and lib/LTO.
  - No longer uses a C api.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold
  - Implemented on top of lib/Object and lib/LTO.
  - No longer uses a C api.
  - Will serve to test LTO strategies.

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold
  - Implemented on top of lib/Object and lib/LTO.
  - No longer uses a C api.
  - Will serve to test LTO strategies.
- tools/lto

Reorganizing lib/LTO, tools/gold and tools/lto

- lib/LTO logic needed by any linker doing lto.
  - Codegen can add calls to memcpy, don't drop it.
  - MCStreamer to find asm symbols.
- tools/gold
  - Implemented on top of lib/Object and lib/LTO.
  - No longer uses a C api.
  - Will serve to test LTO strategies.
- tools/lto
  - Free to be just the api used by ld64 again.

Metadata issues

Metadata issues

- We have support for lazy loading function bodies.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.
- It is uniqued at the context level. Many nodes end up dead.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.
- It is uniqued at the context level. Many nodes end up dead.
- Making Module own metadata would help gold, hurt lld.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.
- It is uniqued at the context level. Many nodes end up dead.
- Making Module own metadata would help gold, hurt lld.
- Lazy loading it during IR linking should be possible.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.
- It is uniqued at the context level. Many nodes end up dead.
- Making Module own metadata would help gold, hurt lld.
- Lazy loading it during IR linking should be possible.
- Maybe do a GC pass? Slower than having the module own it.

Metadata issues

- We have support for lazy loading function bodies.
- Debug metadata is larger than code and not lazy loaded.
- It is uniqued at the context level. Many nodes end up dead.
- Making Module own metadata would help gold, hurt lld.
- Lazy loading it during IR linking should be possible.
- Maybe do a GC pass? Slower than having the module own it.
- Might still need to find a denser representation.

Making LTO useful

Making LTO useful

- One optimization decision can prevent other optimizations.

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.
  - Try to find loop bounds before unrolling.

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.
  - Try to find loop bounds before unrolling.
- LLVM is fairly well tuned for these.

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.
  - Try to find loop bounds before unrolling.
- LLVM is fairly well tuned for these.
- Under the assumption it will never see the code again :-(

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.
  - Try to find loop bounds before unrolling.
- LLVM is fairly well tuned for these.
- Under the assumption it will never see the code again :-(
- We should probably drop some passes with -flto (loop-unroll, vectorize).

Making LTO useful

- One optimization decision can prevent other optimizations.
  - Inlining C in B can prevent inlining B in A.
- Some optimization order is also important.
  - Try to find loop bounds before unrolling.
- LLVM is fairly well tuned for these.
- Under the assumption it will never see the code again :-(
- We should probably drop some passes with -flto (loop-unroll, vectorize).
- Others (inliner) should know that -flto is being used.

Profiling and LTO

Profiling and LTO

- We need to get basic LTO working well first.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.
  - Just cherry pick hot functions for LTO.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.
  - Just cherry pick hot functions for LTO.
  - And use the others from the ELF part (-ffunction-sections).

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.
  - Just cherry pick hot functions for LTO.
  - And use the others from the ELF part (-ffunction-sections).
- LIPO style build also possible.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.
  - Just cherry pick hot functions for LTO.
  - And use the others from the ELF part (-ffunction-sections).
- LIPO style build also possible.
  - In the second build, read some funtions from other IR files.

Profiling and LTO

- We need to get basic LTO working well first.
- Lots of possibilities, have to be careful to not over engineer.
- Should we support fat binaries?
  - Do a first build using the ELF half.
  - Do a sampling run.
  - Only LTO relink with the profile info.
- Partial LTO.
  - Just cherry pick hot functions for LTO.
  - And use the others from the ELF part (-ffunction-sections).
- LIPO style build also possible.
  - In the second build, read some funtions from other IR files.
  - Add them as available-externally to the current TU.