Speeding up Preprocessor

Ivan Sorokin
JetBrains
Ivan.Sorokin@jetbrains.com

Problem statement

- In IDE, files need to be reparsed fast
  - For example in code completion, AST should be ready as soon as possible
- Different tricks are used
  - Caching parsed representation of header files
  - Doing incrementally as many things as possible
  - Doing lazily as many things as possible
- Still sometimes a complete reparse is needed
  - For example when some macro is changed

Idea

Can we compute the final result of the macro expansion and then reuse it?

```
#define A(x) (1 + B(x))
#define B(x) (2 * (x))

// ...
```

```
#define A(x) (1 + (2 * (x)))
#define B(x) (2 * (x))

// ...
```

Idea (2)

Macros can be redefined changing the meaning of other macros:

```
#define A(x) (1 + B(x))
#define B(x) (2 * (x))

// ...

#define B(x) (3 * (x))


// ...
```

```
#define A(x) (1 + (2 * (x)))
#define B(x) (2 * (x))

// ...

#define A(x) (1 + (3 * (x)))
#define B(x) (3 * (x))

// …
```

Idea (3)

- At first expansion, compute the final replacement tokens
    - Cache them for reuse
- Track dependencies: cached representation of which macro depends on which macros
    - Invalidate cached replacement tokens for all dependent macros
- Implemented a quick-and-dirty prototype on clang, the result:
    - Caching speeds up preprocessing of boost libraries by 20%.
    - Can not upstream, because source location tracking was disabled.
- Caching and tracking dependencies overhead can theoretically slow down preprocessing in some cases
    - Was not seen on real programs

Another approach

- Observation: only a small number of macros in boost are expanding thousand of times
- Can we just implement them as built-ins?

Another approach (2)

- Implemented built-in for common macros in boost

• BOOST_PP_CAT	8 LoC
• BOOST_PP_{IIF,IF}	11 LoC
• BOOST_PP_BOOL	10 LoC
• BOOST_PP_COMMA_IF	9 LoC
• BOOST_PP_REPEAT_n	~90 LoC

- Implementing just these reduced the total number of macros expanded from 752695 to 465823
- Got 20% speed-up on boost libraries

Another approach (3)

- Number of macros expanded

Before:

```
  58450 BOOST_PP_IIF_I
  58450 BOOST_PP_IIF
  43717 BOOST_PP_CAT_I
  43717 BOOST_PP_CAT
  40193 BOOST_PP_IIF_1
  32148 BOOST_PP_BOOL_I
  32148 BOOST_PP_BOOL
  18798 BOOST_PP_IF
  18257 BOOST_PP_IIF_0
  12785 BOOST_PP_COMMA_IF
   9498 BOOST_PP_COMMA
   8194 BOOST_PP_TUPLE_EAT_3
   8076 BOOST_PP_FOR_SR_P
   7697 BOOST_PP_ENUM_PARAMS_M
   7470 BOOST_PP_DEC_I
   7470 BOOST_PP_DEC
```

After:

```
  43717 BOOST_PP_CAT
  39652 BOOST_PP_IIF
  13350 BOOST_PP_BOOL
  12785 BOOST_PP_COMMA_IF
   8194 BOOST_PP_TUPLE_EAT_3
   8076 BOOST_PP_FOR_SR_P
   7697 BOOST_PP_ENUM_PARAMS_M
   7470 BOOST_PP_DEC_I
   7470 BOOST_PP_DEC
   6854 BOOST_PP_REPEAT_P
   6821 BOOST_PP_VARIADIC_SIZE_I
   6821 BOOST_PP_VARIADIC_SIZE
   6810 BOOST_PP_OVERLOAD
   6799 BOOST_PP_VARIADIC_ELEM
   6799 BOOST_PP_TUPLE_ELEM_O_3
   6799 BOOST_PP_TUPLE_ELEM_O_2
```

Another approach (4)

Pros:

- Much easier implementation than the caching
  - An implementation of each macro is localized in one function, doesn’t affect surrounding code
- Better error messages

```
BOOST_PP_IF(FOO, 1, 2)

$ pp --builtin-boost-pp 1.cpp
1.cpp:3:13: the 1st argument of macro BOOST_PP_IF must be a
number
$ pp 1.cpp
BOOST_PP_IIF_BOOST_PP_BOOL_FOO( 1, 2)
```

Another approach (5)

Pros:

- Much easier implementation than the caching
  - An implementation of each macro is localized in one function, doesn’t affect surrounding code
- Better error messages

Cons:

- Only speed up boost.

Question

- Is it feasible to implement these built-in macros in major compilers?
  - __builtin_pp_cat
  - __builtin_pp_if
  - __buitin_pp_repeat
  - …
- If it is so, we can prepare a patch.
- As a starting point, the semantics can be defined as if in BOOST_PP_xxx. Can be refined because inside the compiler more tools are available
- In theory we can make the preprocessor library better
  - What about replacing BOOST_PP_{ADD,SUB,MUL,DIV} with generic __builtin_pp_eval(pp-expr)?

Thank you!

- Any feedback is welcome!
    - Ivan.Sorokin@jetbrains.com