■ clang の CUDA 対応について調査

ちょっとソースを読んでみたり.
まず，lib/Lex/Lexer.cpp で

```
<<<
```

と

```
>>>
```

を，それぞれ toke::lesslessless と toke::greatergreatergreater として取得.
その後，lib/Parse/ParseExpr.cpp でパーズしている.

```
case tok::l_paren:         // p-e: p-e '(' argument-expression-list[opt] ')'
case tok::lesslessless: {  // p-e: p-e '<<<' argument-expression-list '>>>'
                           //   '(' argument-expression-list[opt] ')'
```

は共通の case になっていて，内部で処理が分岐している.
LLLoc と GGGLoc の間を取りだして，式を作っているのは次の部分.

```
ExprResult ECResult = Actions.ActOnCUDAExecConfigExpr(getCurScope(),
                                LLLLoc, move_arg(ExecConfigExprs), GGGLoc);
```

ここで，大事な中身は move_arg(ExecConfigExprs) で渡されている.
move_arge は,

```
include/clang/Sema/Ownership.h
```

で定義されている.

```
template <class T, unsigned N> inline
ASTMultiPtr<T> move_arg(ASTOwningVector<T, N> &vec) {
  return ASTMultiPtr<T>(vec.take(), vec.size());
}
```

Sema::ActOnCUDAExecConfigExpr は，lib/Sema/SemaExpr.cpp
に実装されている．で，ごにょごにょ何か処理したあと，真打は,

```
return ActOnCallExpr(S, ConfigDR, LLLLoc, execConfig, GGGLoc, 0);
```

で呼ばれている Sema::ActOnCallExpr.

```
if (Dependent) {
  if (ExecConfig) {
    return Owned(new (Context) CUDAKernelCallExpr(
        Context, Fn, cast<CallExpr>(ExecConfig), Args, NumArgs,
        Context.DependentTy, VK_RValue, RParenLoc));
  } else {
    return Owned(new (Context) CallExpr(Context, Fn, Args, NumArgs,
                                        Context.DependentTy, VK_RValue,
                                        RParenLoc));
  }
}
```

で，CUDAKernelCallExpr のインスタンスが作られている．
... と思ったら，この条件にマッチしないぞ？
Dpendent も ExecConfig も 0 だなあ ...
結局

```
return BuildResolvedCallExpr(Fn, NDecl, LParenLoc, Args, NumArgs, RParenLoc,
                             ExecConfig);
```

という最後の return 文で呼び出し元へ返っている．
ああ，Dependentって，定義部分なのか？と．
で，GPU カーネルの caller の場合，BuildResolvedCallExpr の中で，

```
if (Config) {
  TheCall = new (Context) CUDAKernelCallExpr(Context, Fn,
                                             cast<CallExpr>(Config),
                                             Args, NumArgs,
                                             Context.BoolTy,
                                             VK_RValue,
                                             RParenLoc);
} else {
  TheCall = new (Context) CallExpr(Context, Fn,
                                   Args, NumArgs,
                                   Context.BoolTy,
                                   VK_RValue,
                                   RParenLoc);
}
```

と, CUDAKernelCallExpr を生成している．ここで Args が, LLLLoc と GGGLoc の中身だと思うのだけど
dump しても一見ではわからないな ... また今度.
CUDAKernelCallExpr は ./include/clang/AST/ExprCXX.h に定義されている．

■一言メモ

- FSWiki の整形，プログラム系のメモを残すときには不便だな (Fri Mar 25 16:11:07 2011 +0900)
- 研究室の自分の端末 <-> 研究科のサーバ より さくらレンタルサーバ <-> 研究科のサーバの方が ping の round-trip が 1 桁くらい速い ... (Fri Mar 25 12:18:59 2011 +0900)
- Ruby スクリプト中の END 以降の行は DATA でアクセスできるのか．これは嬉しい♪ http://www.mapee.jp/ruby/data__end.html (Fri Mar 25 10:57:17 2011 +0900)
- みかけたら欲しいなあ．http://www.amazon.co.jp/exec/obidos/ASIN/4774138649/ (Fri Mar 25 03:45:47 2011 +0900)