■ ASPLOS(4)

本会議 2 日目.
キーノートは量子コンピュータの話.
分野によって発表する人の雰囲気がにてるような気がするなあ，とか.

夜は懇親会．立食かと思いきやテーブル式.
隣に座っていた人が Mobile での発表があった著者だったので,
いろいろと話を聞くなど.

懇親会では，グラスモニカの演奏が.
とても澄んだいい音色でした.

■ Halide AOT **について調べる** (1)

Halide について，ちょっと調査．気になるのは AOT コンパイラまわり.
というわけで，とりあえずチュートリアルを動かしてみつつ，何やってるのか眺めてみる.
Halide/tutorial からみて，../../halide_build の下で一式をコンパイルした環境.

lesson_10_aot_compilation_generate.cpp をコンパイル

```
g++ lesson_10_aot_compilation_generate.cpp -g -std=c++11 \
    -I ../../halide_build/include \
    -L ../../halide_build/bin/ \
    -lHalide -lpthread -ldl -o lesson_10_generate
```

これで，lesson_10_generate が生成される

生成された lesson_10_generate を実行

```
LD_LIBRARY_PATH=../../halide_build/bin ./lesson_10_generate
```

すると，lesson_10_halide.a と lesson10_halide.h が生成される.
これは，lesson_10_aot_compilation_generate.cpp の

```
brighter.compile_to_static_library("lesson_10_halide", {input, offset}, "brighter");
```

で決められた名前．lesson_10_halide.h をみると,

```
 int brighter(struct halide_buffer_t *_input_buffer, uint8_t _offset, struct halide_buffer_t
*_brighter_buffer) HALIDE_FUNCTION_ATTRS;
```

が定義されている．lesson_10_halide.a を

```
objdump -d lesson_10_halide.a | grep \^000000
```

とダンプしてみると, いろいろと関数がまとまっていることがわかる (中を追うのはあとまわし)

実行ファイルの生成

```
g++ lesson_10_aot_compilation_run.cpp lesson_10_halide.a -g -std=c++11 \
    -I ../../halide_build/include \
    -L ../../halide_build/bin/ -lHalide -lpthread -ldl -o lesson_10_run
```

と，generate で生成した .a ファイルと一緒にコンパイル.

_generate と _run

_generate で，brighter を作っている.

```
    Func brighter;
    Var x, y;
    Param<uint8_t> offset;
    ImageParam input(type_of<uint8_t>(), 2);
    brighter(x, y) = input(x, y) + offset;
    brighter.vectorize(x, 16).parallel(y);
    brighter.compile_to_static_library("lesson_10_halide", {input, offset}, "brighter");
```

データの箱と操作を定義してライブラリにコンパイルする，と.
で，_run で,

```
    Halide::Runtime::Buffer<uint8_t> input(640, 480), output(640, 480);
    int offset = 5;
    int error = brighter(input, offset, output);
```

と呼び出している，と.

compile_to_static_library？

AOT コンパイラの本体は Func に定義された，compile_to_static_library のようなので,
Halide/src/Func.cpp を眺めてみると,

```
void Func::compile_to_static_library(const string &filename_prefix,
                                     const vector<Argument> &args,
                                     const std::string &fn_name,
                                     const Target &target) {
    pipeline().compile_to_static_library(filename_prefix, args, fn_name, target);
}
```

というのがある．pipline() とは？と，Func.cpp の中で検索してみると,

```
Pipeline Func::pipeline() {
    if (!pipeline_.defined()) {
        pipeline_ = Pipeline(*this);
    }
    internal_assert(pipeline_.defined());
    return pipeline_;
}
```

と，Pipeline のインスタンスを用意する関数のよう.
実際には，Pipeline のインスタンスの compile_to_static_libary を呼び出している，と.

Pipeline の compile_to_static_library の定義をみてみると

```
void Pipeline::compile_to_static_library(const string &filename_prefix,
                                         const vector<Argument> &args,
                                         const std::string &fn_name,
                                         const Target &target) {
    Module m = compile_to_module(args, fn_name, target);
    Outputs outputs = static_library_outputs(filename_prefix, target);
    m.compile(outputs);
}
```

となっている.

もっと AOT

lesson15 では，Generator を継承するチュートリアルなので，そっちもみてみよう.
src/Generator.h には,

```
 * Generator is a class used to encapsulate the building of Funcs in user
 * pipelines. A Generator is agnostic to JIT vs AOT compilation; it can be used for
 * either purpose, but is especially convenient to use for AOT compilation.
 *
 * A Generator explicitly declares the Inputs and Outputs associated for a given
 * pipeline, and (optionally) separates the code for constructing the outputs from the code from
 * scheduling them. For instance:
```

とか書いてある.