OP2-clang

December 11, 2017

Contents

1 Introduction

OP2 is a high level framework with associate libraries and preprocessors to generate parallel executables for unstructured mesh applications. This document describes the usage and implementation of the clang based C++ preprocessor for OP2.

2 Overview

OP2-clang is a clang based preprocessor using clang's LibTooling to generate parallel executables. Currently OP2-clang enables users to write single program which can be built into three executables:

- Single-threaded on a CPU
- Multi-threaded using OpenMP for multicore CPU systems
- AVX vectorized for x86 CPU systems

OP2-clang is a clang based "source-to-source translator" tool. It's purpose is to generate parallelised and optimised source code for unstructured-mesh applications using OP2's abstractions. OP2-clang provides a modular abstraction to perform the translation and gives a convenient interface to extend it with

new optimisations and parallelisation approaches. This tool is operating on the AST of the application. It collects data about the application from based on the OP2 library calls and with simple replacement on the main sources it generates copies for parallel execution. Then based on the collected data the tool generates optimised sources for the parallel loops. The kernel generation is operating on parallelisation approach specific simple skeleton codes. The generator parse the skeletons for each parallel loop and alter the source code through replacements to get the corresponding kernel used to get parallel executables.

3 Build

You will need a checkout of the llvm, clang and clang-tools-extra source code first (see `http://clang.llvm.org/get_started.html` for instructions). Check out the OP2-Clang repository and set `OP2_INSTALL_PATH` e.g. with:

```
git clone https://github.com/bgd54/OP2-Clang.git;
export OP2_INSTALL_PATH="/path/to/op2/"
```

Then in the OP2-Clang directory:

```
mkdir build
cd build
cmake ..
make
```

4 Usage

The tool can be run on the command-line with the command:

```
./op2-clang main.cpp sub1.cpp -- clang++ --app-spec-flags
```

Assuming that the application is split over several files. This will lead to the output of the following files `main_op.cpp, sub1_op.cpp, sub2_op.cpp` and a master kernel file (`main_xxxkernels.cpp`) for each approach generated where xxx is differ between executables, and an individual kernel file for each parallel loop.

- The `main_op.cpp` file is used for all versions (sequential, OpenMP and vectorized) executables.
- The `main_xxxkernels.cpp` files are including the individual kernel files (`<loop_name>_xxxkernel.cpp`).
- The individual kernel files (`<loop_name>_xxxkernel.cpp`) containing the efficient parallelised versions of the parallel loops. These files generated from the corresponding skeletons with modifications based on the information collected from `main.cpp` (and from other files parsed during the invocation of the tool) about the loop.

The code generation can be controlled through `optarget` flag:

```
./op2-clang -optarget=vec main.cpp -- clang++
```

In this case only the version specified by the value of `optarget` will be generated.

5 Implementation

4 layer arch layers: data collect version generators kernel generators userfunction modifiers

?dev-guide?