KPP (Kinetic Pre-Processor) CUDA Enabled

Christos Kannas

The Cyprus Institute

February 2014

Introduction

The purpose of this document is to provide information about the enhancements in KPP version 2.2.3 that pave the road towards a new version of KPP (2.2.4) which will be able to generate CUDA and/or OpenACC enabled source code for atmospheric chemistry modelling.

The work currently available is this new version of KPP, 2.2.4, was focused in enhancing KPP version 2.2.3 with the capability of generating CUDA C code for the atmospheric modelling. The current development was focused on profiling and identifying the computational "hotspots" within the KPP and provides a solution that utilized the computational power of general purpose GPU infrastructure that can be found In almost any stat of the art HPC.

Disclaimer: This is the first version of the CUDA enabled KPP and its purpose is to pave the road for further development and enhancement of KPP.

Development Details

The first step of the development was to provide the necessary tools within KPP that will later help the developers to build a version of KPP that will offload as much work possible on the GPUs. This has been achieved by adding a CUDA C header file, which provides a set of functions with similar API covering the programming calls required by software that uses available CUDA devices. This

header file is located in *%KPP_HOME/util/* and it is named *cuda_utils.h*, this header fileshould be included in any generated CUDA C source code file. To do this during the generation of the final source code files, it is copied to the selected output directory and renamed to "*$ROOTName*_CUDA_Utils.h". The functions defined in this header file provide functions that display information about the available CUDA devices, select the best CUDA device available on the system, simple API to allocate memory that can be used by CUDA device(s), copy data to, from and between CUDA device(s).

The second step was to provide the required utilities within KPP to generate CUDA C source code files. This was achieved by adding in *%KPP_HOME/src/* the C source code file *code_cuda_c.c*. In this file there are the functions that let the developer define variables and functions using CUDA C API. Also is the main reference point with regards to what source code files will be generated. It is required due to the existence of special qualifiers for variables and functions. This file should be extended in the future in order to provide a complete set of functions that should be in line with the general source code generation paradigm KPP uses. Function *Use_CUDA_C* prepares the source code files to be created and maps the functions related to C source code generation to the KPP relative global calls. Function *CUDA_C_Declare* can be used to declare variables with the respective CUDA variable qualifier that will reside in CUDA device memory. Similarly function *CUDA_C_FunctionBegin* can be used to define a function with CUDA specific function qualifiers.

The third step was to update core KPP source code files responsible for the source code generation for all supported languages. In *code.h* added lines 80, 81 and 178. In gdata.h changed the versionof KPP from 2.2.3 to 2.2.4, at line 65 added *CUDA_C_LANG* in the *lang* variable, which lists the available languages that KPP can support. In file *code.c* updated function *OpenFile* with lines 169 – 180 where they add the required includes to header files required by files that have the extension *.cu*, update function *IncludeCode* so that in the case of CUDA C generated code, to include code from files with extension *.c* or *.cu* (lines 282 - 291), at lines 323 – 325 in the same function KPP is

instructed to substitute occurrences of *KPP_REAL* within a file with the corresponding C type for *KPP_REAL*. In file *gen.c*, in function *GenerateMonitorData* at lines 487 – 488 KPP is order to use any inline code that is written in C when it comes to generate CUDA C source code, in function *GenerateFun,* is where all the effort was given for this version of the code, lines 610 – 701 where added, in these lines the required source code for CUDA C is generated for running ODE Fun in a CUDA device. Variables that are to be shared among blocks are declared, the kernel function *d_Fun_A* is defined, this function is responsible to calculate equation rates; that is the vector *A* in the function each CUDA thread is responsible to calculate one element of A. Kernel function *d_Fun_Vdot* is defined, which using the previously calculate *A* vector to calculate the values of *Vdot* vector, each CUDA thread is responsible to calculate one element of Vdot. At line 707 – 772 the function *Fun* is defined, in the function the best CUDA device available is selected, the required memory allocations at the CUDA device are performed, then for each kernel the work distribution is performed, that is calculating how many blocks are required to do the calculation by using a fixed number of 256 CUDA threads per block, the calls to each kernel take place and finally the collection of results from the CUDA device and freeing of device memory used. At lines 2256 – 2292 the function *GenerateFortranInteface* was added, this function create a required Fortran interface in the case of CUDA C generated source code, in order to call the *Integrate* function. At line 3345 function Generate is located, this function has been updated accordingly at lines 3372 -3373 to generate the required files for CUDA C, and at lines 3477 – 3480 to generate the required Fortran interface.

Future Work

As this is the first version of the updated KPP, there are a lot of things that can be updated in order to make it more CUDA friendly.

In order to utilize the usage of available CUDA devices many parts of the generated source code should be ported to CUDA C code. In general you want to have as much of the computation to be done in the CUDA device, and minimize the data transfer between host and GPU.

One of the most important parts that have to be ported in CUDA are the integrators, for each integrator a new source code for CUDA C will be created where the selection of device, GPU memory allocation will take place, which will subsequently require to port further parts of the code structure in CUDA C, such us the various functions that are used to do calculations in various vectors required by the core ODEs *Fun* and *Jacobian*. The file *blas.c* will also need to be ported to CUDA.