Знакомство с Altera OpenCL SDK

План

1 Необходимое программное и аппаратное обеспечение

Для выполнения данной работы, как и для использования Altera OpenCL в целом, необходимо наличие САПР Altera Quartus II (не Web Edition !) версии 13.1 и выше с поддержкой устройств FPGA семейства Altera Stratix IV и выше, а также ПО Altera OpenCL SDK. В данной работе используется версия Altera Quartus II 13.1, но рекомендуется версия 14 от июля 2014, поскольку в ней реализован ряд улучшений поддержки OpenCL и представлены новые инструменты.

Для тестирования работы решений и затем для выполнения собственно вычислений необходимо наличие аппаратного обеспечения, содержащего FPGA Altera Stratix IV и выше (для новой версии OpenCL SDK также поддерживаются новые семейства Cyclon) и обеспечивающего поддержку Altera OpenCL. В данной работе используется плата DE5-NET компании Terasic с FPGA Altera Stratix V (рисунок 1). Оборудование и программное обеспечение приобретено в рамках университетской программы Altera.

Для подготовки хост приложений используется Microsoft Visual Studio 2012.

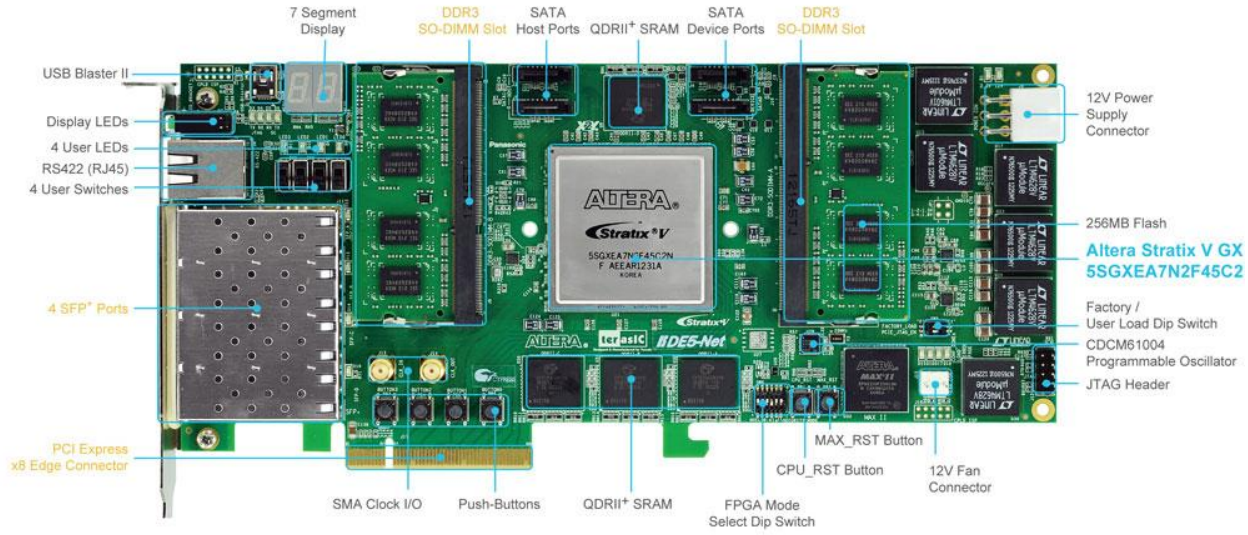

Рисунок 1 - Внешний вид платы DE5-NET с поддержкой OpenCL

2 Особенности проведения занятия – используем Remote Desktop

Поскольку в наличии имеются только 3 устройства, установленных на серверах **MS HPC 2012**, работа с ними ведется через удаленный рабочий стол **Remote Desktop**. Также необходимо отметить, что компиляция решений, и особенно – подготовка прошивок для FPGA требуют значительных ресурсов, в частности, не менее 20-24 Гб оперативной памяти, и занимают длительное время, что также объясняет использование мощного серверного оборудования и удаленного доступа к нему.

После получения доступа к машине, на которой будет вестись работа, рекомендуется создать папку со своей фамилией или номером группы на диске C:\ в каталоге Projects.

3 Подготовка решения

Как уже отмечалось выше, подготовка ядер OpenCL и особенно получение прошивок для FPGA по коду этих ядер требует значительного времени. В текущей версии Altera OpenCL эта проблема не решена, и в документации указывается, что в зависимости от сложности ядра и

возможностей компьютера, используемого для генерации прошивок, она может занимать от нескольких часов до суток и более (!).

По этой причине необходимо внимательно относиться к подготовке ядер и использовать режим отладочной компиляции компилятора Altera OpenCL (AOCL) :

```
aoc –c <имя файла ядра.cl> --report
```

Также в большинстве случаев можно использовать оценку пропускной способности (производительности) ядра с помощью ключа **--estimate-throughput :**

```
aoc –c <имя файла ядра.cl> --report --estimate-throughput
```

Этот режим не доступен при оценочной компиляции так называемых single work-item ядер (с атрибутом <<task>>).

В целом при подготовке решений необходимо придерживаться схемы, предлагаемой в документации Altera (рисунок 2) :

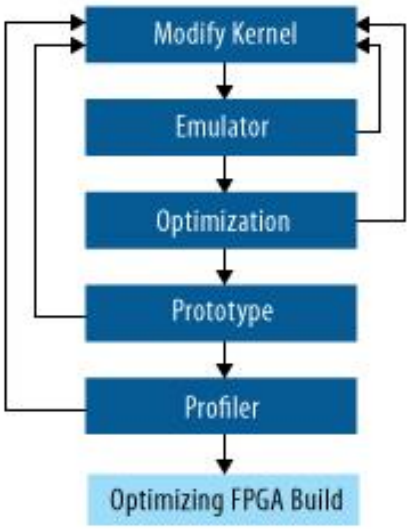

Рисунок 2 – Последовательность подготовки FPGA решений на Altera OpenCL SDK

3.1 Подготовка ядра OpenCL (kernel) и его оптимизация

Подготовку ядра OpenCL можно вести как в среде разработки, так и в текстовом редакторе, при подготовке необходимо пользоваться

документацией Khronos Group (khronos.org) по стандарту OpenCL и документацией Altera (http://www.altera.com/products/software/opencl/opencl-index.html) – руководством программиста AOCL и руководством по оптимизации кода AOCL.

Запуск компилятора AOCL нужно производить в командной строке, соответственно, нужно открыть Command Prompt и перейти в нужный каталог.

Оптимизация ядра включает как типичные для OpenCL приемы, такие как использование Private + Local памяти, работу с индексами NDRange, в целом выделение наиболее интенсивных с точки зрения вычислений, но при этом компактных функций и т.д., так и более специфичные для AOCL : разворачивание циклов (loop unroll), ядра с одним work-item (single work-item), атрибуты для размещения нескольких вычислительных блоков и SIMD – блоков в ядре и т.д.

Одним из вариантов оптимизации является автоматическая компиляторная оптимизация с ключом –O3 по аналогии с оптимизацией, например, компиляторов C/C++. В большей степени она позволяет максимально задействовать ресурсы FPGA (до 85%), но она не заменяет ручной оптимизации.

3.2 Создание образа для FPGA

Для создания бинарного образа прошивки FPGA (файл с расширением *.aocx) необходимо просто запустить компилятор с указанием файла с исходным кодом ядра :

```
aoc <имя файла ядра.cl>
```

Но придется запастись терпением…

3.3 Приложение для хоста

Хост – приложение на C представляет собой в целом обычное хост-приложение для OpenCL программ, в нем используются предусмотренные стандартом функции API для запроса системы, создания контекстов, очередей, буферов памяти, выделения памяти, пересылки буферов в память устройства и назад, собственно для вызова ядер. Необходимо отметить ряд

ограничений, например, в хост-приложениях для Altera OpenCL нельзя выполнять компиляцию исходных кодов ядра, а допускается только загрузка бинарного файла с прошивкой. При первом вызове ядра функцией `clEnqueueNDRangeKernel` происходит и загрузка прошивки в FPGA, при повторных вызовах этой функции, если прошивка не менялась, процесс программирования соединений FPGA выполняться не будет.

В рамках данной работы необходимо воспользоваться предоставленными примерами хост-приложений.

4 Пример 1

В данном примере рассматриваются некоторые возможности оптимизации кода ядра. В качестве задачи рассмотрено умножение 3х компонентного вектора на последовательность предварительно рассчитанных целочисленных матриц в экспериментальной схеме шифрования HW-QES (на базе кватернионов). В рамках данной работы сам алгоритм HW-QES не рассматривается, его описание можно найти в [1,2]. С точки зрения вычислительной схемы в данном случае необходимо для большого числа 3- компонентных векторов с компонентами-байтами выполнить их умножение последовательно на ряд матриц с предварительно рассчитанными целочисленными элементами. Матрицы передаются в ядро в качестве параметров, как и указатель на буфер, содержащий вектора. Собственно, это буфер для шифруемых данных. Каждая единица работы (work-item) будет независимо обрабатывать свой вектор.

Вначале рассматривается код без развернутого цикла.

Небходимо выполнить оценочную компиляцию и определить такие параметры ядра, как пропускную способность в work-item в секунду, требуемый темп работы с памятью, а также проценты использования кристалла FPGA, включая логические блоки, DSP блоки, блоки памяти, регистры микросхемы.

Затем можно выполнить компиляцию версии ядра, в которой Private память (по умолчанию) заменена на __local и определить те же параметры ядра.

Затем нужно выполнить разворачивание единственного цикла с помощью прагмы

```
#pragma unroll 8
```

и снова определить параметры ядра. Обратите внимание на то, как возросла сложность и одновременно – пропускная способность.

Попробуйте перейти к **single work-item** ядру, указав атрибут task :

```
__attribute__ ((task))
```

перед кодом ядра.

В заключение попробуйте, проанализировав процент занятости ресурсов FPGA в предпоследнем варианте, подобрать максимальное количество одновременно работающих блоков N с помощью атрибута

```
__attribute__((num_compute_units(N)))
```

Обратите внимание на то, чтобы не превышался максимальный темп выдачи данных из памяти (около 26 Гб / для данной платы).

К сожалению, из-за большого времени компиляции прошивки, поэкспериментировать с созданной прошивкой и хост-приложением не удастся.

5 Пример 2

В качестве второго примера рассматривается реализация FIR-фильтра низких частот с использованием плавающей запятой одинарной точности.

Данный пример приводится на сайте Altera в разделе Примеры (для OpenCL – см. отдельный документ в pdf). К сожалению, большинство примеров из разряда HPC, приведенных на сайте, используют **single work-item,** соответственно, не удастся поэкспериментировать в отличие от предыдущего примера, с разными вариантами ядра, так как оценка производительности для таких типов ядер компилятором не проводится.

Рассмотрите проект, выданный преподавателем.

Попробуйте выполнить оценочную компиляцию исходного ядра и с закомментированным атрибутом task. Сравните параметры используемых ресурсов FPGA.

Изучите хост-приложение. Скомпилируйте его, поместите ранее подготовленную прошивку в папку с исполняемым файлом и запустите

приложение. Запишите выдаваемую программой информацию о производительности. Попробуйте переместить файл с прошивкой в другое место, запустите программу, убедитесь, что она выдает ошибку. Верните прошивку в текущую папку. Поменяйте в программе флаг запуска расчета на ЦПУ, скомпилируйте и снова запустите, сравните показатели производительности с вариантом расчета на FPGA. Уменьшите до одного количество обрабатываемых фильтров и посмотрите, как изменится относительная производительность FPGA и CPU. Почему так происходит ?

Можно попробовать выполнить повторный вызов `clEnqueueNDRangeKernel` и замерить общее время их выполнения (оценить время прошивки и то, что она повторно не выполняется).

Сравните производительность с оценками для GPU, приведенными в pdf документе. Сравните производительность с производительностью для платы с FPGA, приведенной в документе pdf. Чем объясняется, на ваш взгляд, разница с проведенным вами замером ?