Инженерный вестник Дона, №8 (2023)
ivdon.ru/ru/magazine/archive/n8y2023/8599

Применение графических адаптеров Intel UHD в математических вычислениях

М.С. Максютов

Московский государственный строительный университет

Аннотация: На основе последних разработок в области параллельных вычислений, в частности, на уровне абстракции SYCL, рассматривается применение оптимальных средств параллельных вычислений для построения приложений в области вычислительной и прикладной математики. Приводятся примеры, как простых алгоритмов вычислений, так и вычислений с использованием математических библиотек для задач вычислительной линейной алгебры.
Ключевые слова: параллельный код, гетерогенная среда, openmp, intel data parallel c++, oneapi, onemkl, sycl, fpga, cpu, gpu.

Когда мы говорим о повышении производительности вычислений, то, в первую очередь, речь заходит о параллельных вычислениях, первое знакомство с которыми нам дает интерфейс OpenMP, и примеры его использования в работах [1,2]. Механизмы сопровождения и отладки таких приложений были хорошо освещены в работе [3]. Если раньше мы могли ограничиваться, в основном, только ресурсами многоядерного центрального процессора (CPU) или набора таких CPU, то сегодня речь уже идет о системах, содержащих различные типы вычислительных устройств: от центрального процессора до различного типа акселераторов массивно параллельных вычислений (FPGA), и графических картах (GPU), классификация которых хорошо представлена в работах [4,5]. Такие системы и среды, содержащие микропроцессоры с ядрами различных типов, принято называть гетерогенными. Поэтому возникла необходимость в разработке единого инструмента программирования для гетерогенных сред, на фоне существования разрозненных специализированных программных интерфейсах (API) для них. Несогласованная поддержка инструментов на разных платформах еще больше усиливает этот разрыв. Дело в том, что для

каждой платформы требуется уникальное программное обеспечение и его дальнейшая поддержка.

Группа компаний CHRONOS (khronos.org) предложила достаточно простой, в тоже время очень эффективный интерфейс, устраняющий все эти трудности, путем организации единого средства программирования, в виде свободного кроссплатформенного уровня абстракции SYCL. Он позволяет программировать гетерогенные системы на основе современной реализации языка программирования C++, с соответствующим API, находя в системе необходимые устройства для выполнения на них кода. Компания Intel обобщила этот опыт и применила его для выполнения на различных устройствах собственной архитектуры, от центрального и графического процессоров до FPGA-акселераторов, в наборе библиотек и инструментов разработчика oneAPI [6]. Основным в oneAPI являяется компилятор Data Parallel C++ (DPC++), позволяющий напрямую программировать, как CPU, так и GPU устройства. На основе API-библиотек предоставляется возможность также использовать различные FPGA-акселераторы. В одном коде позволяется смешивать устройство, используемое по умолчанию в системе, вместе с ядрами гетерогенных акселераторов.

Например, рассмотрим настройку вычислительной системы для использования DPC++ на примере лэптопа DELL Latitude 5310. В качестве микропроцессора, в нем выступает Intel Core i5-10210U с интегрированной графикой Intel UHD Graphics. Мы можем обратиться к центральному процессору компьютера, и узнать его подробную спецификацию, использовав следующий код:

```
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;
using namespace std;
```

```
int main()
{
//
 queue Q{ cpu_selector_v };
//
 cout << "Selected device: " <<
 Q.get_device().get_info<info::device::name>() << std::endl;
 cout << " -> Device vendor: " <<
 Q.get_device().get_info<info::device::vendor>() << std::endl;

 return EXIT_SUCCESS;
}.
```

Результат его выполнения представлен на рис.1.

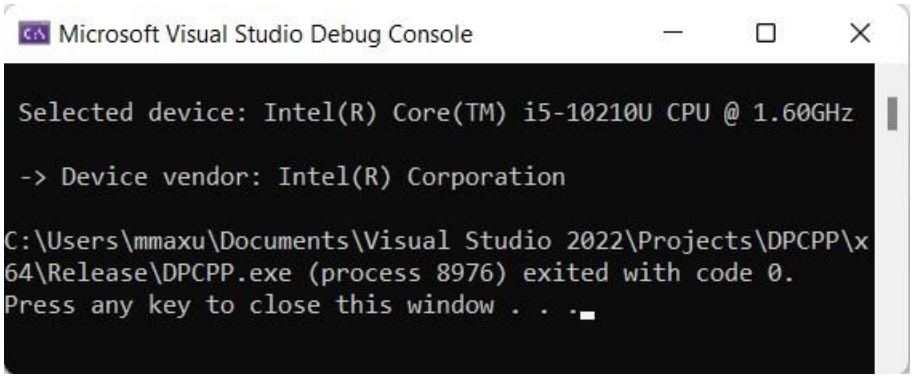

Рис. 1. - Спецификация CPU в системе.

В предыдущих версиях DPC++ центральный процессор являлся устройством по умолчанию. С появлением версии 2023 года, ситуация изменилась в корне, и теперь устройством по умолчанию является графический процессор. Это необходимо учитывать, так как коды, реализованные в двух предыдущих версиях DPC++ уже не совместимы с текущей!

Полную информацию об устройствах, доступных в системе (на примере лэптопа DELL Latitude 5310) теперь можно получить, используя следующий код:

```
#include <sycl/sycl.hpp>
#if FPGA || FPGA_EMULATOR
#include <sycl/ext/intel/fpga_extensions.hpp>
#endif
#include <iostream>
#include <string>
using namespace sycl;
using namespace std;
void output_dev_info(const device& , const std::string& );
int main()
{
  output_dev_info( device{ default_selector_v},"default_selector" );
  output_dev_info( device{ cpu_selector_v},"cpu_selector" );
  output_dev_info( device{ gpu_selector_v},"gpu_selector" );
  output_dev_info( device{ accelerator_selector_v},"accelerator_selector" );
//
  return EXIT_SUCCESS;
}
void output_dev_info(const device& dev,const std::string& selector_name)
{
  cout << selector_name << ": Selected device: " << dev.get_info<info::device::name>() << "\n";
  cout << "-> Device vendor: " << dev.get_info<info::device::vendor>() << "\n";
}.
```

Результат его работы представлен сообщениями, показанными на рис. 2. Как видно из рис.2, устройством по умолчанию является графическая подсистема, в виде процессора Intel UHD 620. Его особенность - возможность

использовать оперативную память в составе центрального процессора Intel Core i5-1021U (4 физических и 4 логических ядра).

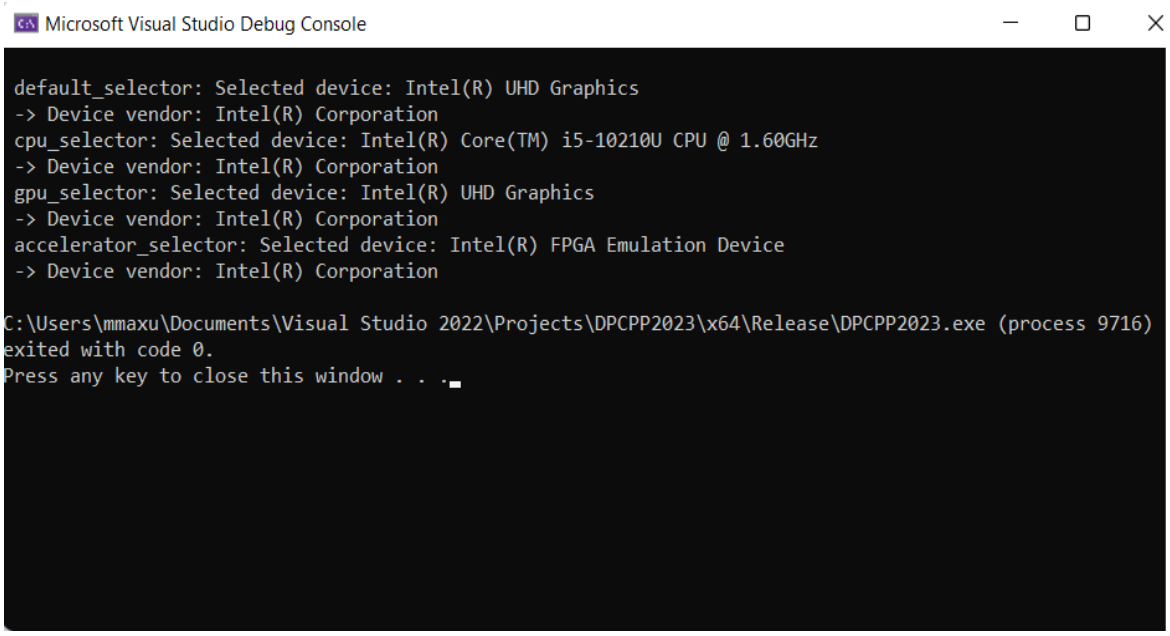

Рис. 2. – Информация о доступных устройствах.

Кроме того, он позволяет работать с данными двойной точности, в отличии, например, от графики следующего поколения - Intel Iris Xe. Это обстоятельство особенно важно для задач вычислительной математики.

По умолчанию, для него выделяется половина объема оперативной памяти (ОЗУ), установленной в системе, максимально до 32 Гб. В составе графической подсистемы находится 24 исполнительных устройства (процессоры), с эквивалентным числом шейдерных процессоров – 192. Помимо графического процессора, предоставляется возможность использования FPGA-акселератора, и если он не установлен в системе, то он эмулируется ресурсами CPU и GPU. Несмотря на то, что данное CPU - это низковольтный процессор для ноутбука, его графическая подсистема позволяет добиться высокой производительности в вычислениях. Конечно, его эффективность достигается только при использовании соответствующего ПО.

Рассмотрим простой код на DPC++, в стандарте C++20, который формирует одномерный целочисленный массив и записывает в него значения его индексов, используя акселератор:

```
#include <CL/sycl.hpp>
#include <iostream>
#include <cstdlib>
using namespace sycl;
using namespace std;
const int N = 10;
int main()
{
 device dev(accelerator_selector_v);
 queue q(dev);
 auto R = range<1>{ N };
 buffer<int > A{R};
 q.submit([&](handler& h)
        {
         auto out = A.get_access<access::mode::write>(h);
             h.parallel_for(R,[=](id<1> idx){out[idx] = idx[0];});
        });
 auto result = A.get_access < access::mode::read>();
 for (int i = 0; i < N; ++i)cout << result[i] << std::endl;
 return EXIT_SUCCESS;
}.
```

Если выбрать в качестве целевого устройства акселератор массивно-параллельных вычислений, например, Intel Xeon Phi [7], то он будет вызван запросом accelerator_selector_v. За организацию массива будет отвечать шаблонный класс buffer. При этом, часть кода, которая будет выполняться

на ядрах акселератора, определяется только в строке 15, объявляющей лямбда-функцию в параллельном цикле parallel_for, остальной код выполняется ресурсами CPU. Для избежания конфликтов ресурсов race condition, при распараллеливании вычислений, доступ к массиву A объявляется только для записи, а соответствующий вывод - только для чтения.

Для того, чтобы показать вычислительную эффективность инструмента разработки Data Parallel C++, достаточно сравнить его со стандартным C++, например, реализованном в среде разработки Visual Studio 2022. Для этого, в качестве первого примера, возьмем стандартную процедуру перемножения двух вещественных квадратных матриц, в арифметике двойной точности. Сначала на последовательном коде, затем на параллельном. При этом, размер матриц выбран достаточно большим, чтобы выйти за пределы кэша процессора. В данном случае, размерность матриц определена, как 4000x4000. Результаты сравнительного теста приведены в гистограмме, на рис. 3. Как видно из рис. 3, для 8-ядерного процессора (4 физических ядра), результат выглядит не слишком впечатляюще, хотя эффективность загрузки ядер в параллельном цикле дает 100-процентную загрузку всех ядер. Скорость вычислений повысилась всего лишь в три раза (синяя гистограмма). Решим аналогичную задачу с помощью Intel Data Parallel C++, используя мощности только центрального процессора (запрос **cpu_selector_v**). В случае использования компилятора DPC++ (оранжевая гистограмма), результат перемножения для последовательного цикла практически не изменился, с небольшим преимуществом Visual C++, что объясняется приоритетом оптимизаций однопоточных вычислений для Visual C++. Однако оптимизированный для параллельных вычислений, в параллельном цикле, код DPC++ выполнился гораздо быстрее, и «лучше» подходит для восьмиядерного процессора. Скорость вычислений выросла в 13 раз!

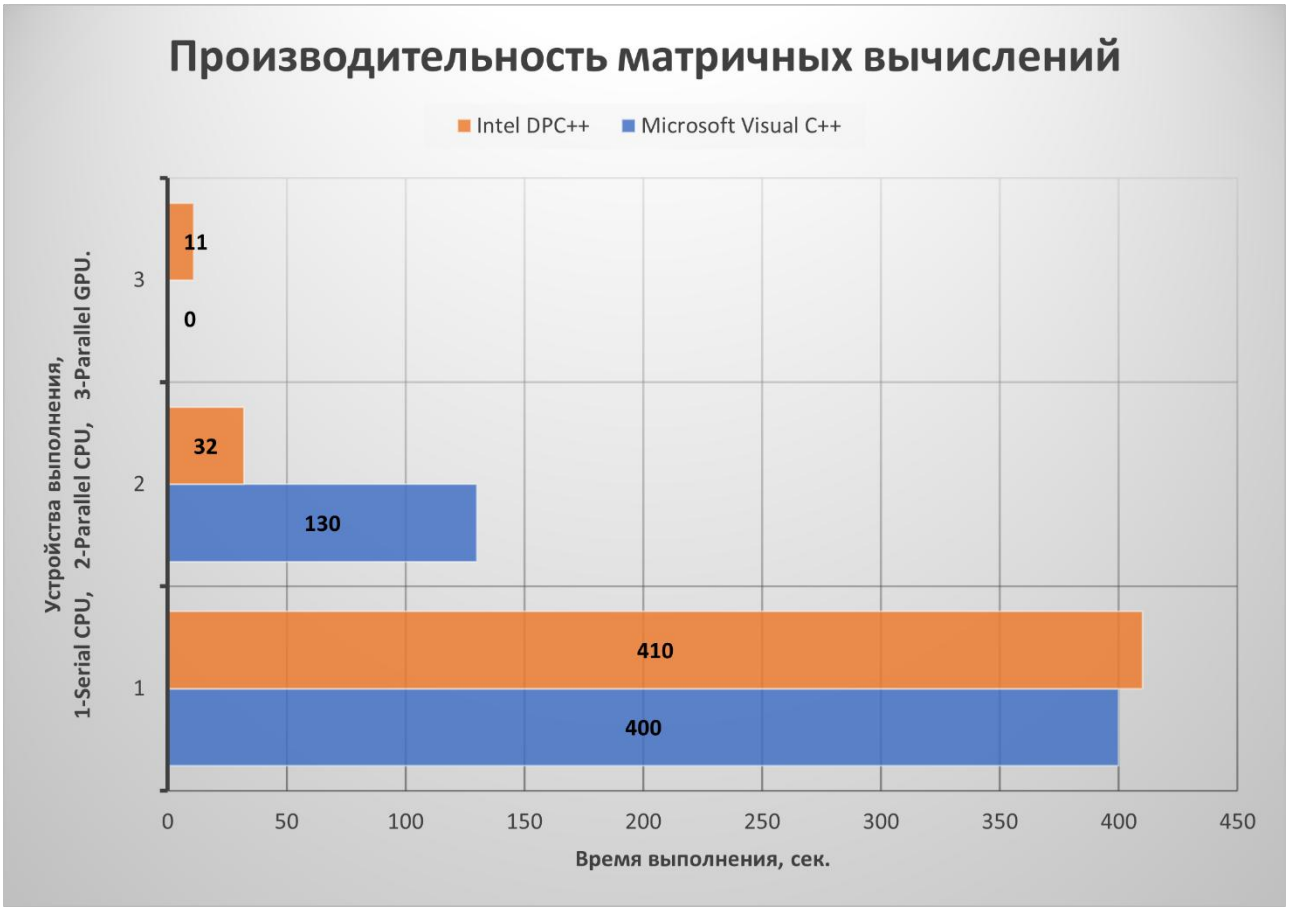

Рис. 3. -Результат перемножения матриц 4000x4000 элементов

Теперь, пожалуй, самое интересное, как изменятся результаты при использовании графического акселератора (запрос **gpu_selector_v**). Использование графического акселератора наиболее эффективно сказалось на выполнении именно параллельного кода, фактически, прирост производительности оказался примерно в 37 раз быстрее, чем при использовании последовательного кода, только на центральном процессоре! В Visual C++ без использования сторонних специализированных библиотек, возможно использование мощностей только центрального процессора, поэтому его результат в этом тесте отсутствует и обнулен.

Однако не следует полагать, что использование графического акселератора по умолчанию — это всегда наилучшее решение. Если графический процессор интегрирован, то необходимо помнить, что для успешной работы необходимо следить за размером выделяемой оперативной

памяти, объем которой не может превышать половины установленного объема ОЗУ. Более того, в зависимости от специфики задач, применение только графического акселератора может оказаться неэффективным, так как мощности CPU окажутся незадействованными. Поэтому целесообразно использовать для вычислений, как центральный, так и графический процессоры одновременно. Для этого нам придется использовать еще один механизм управления гетерогенными средами вычислений, а именно: FPGA-акселератор. В качестве одного из примеров такого акселератора, можно привести уже упоминавшийся сопроцессор Intel Xeon Phi, MIC-архитектуры “Knights Landing”, выпуск которого прекращен, но последняя версия еще поддерживается oneAPI. Работа с акселератором, в случае его физического отсутствия, происходит в режиме его эмуляции, но это позволит нам использовать весь объем доступной памяти и ресурсы CPU. Поможет это сделать запрос **accelerator_selector_v**. Применение запроса на акселератор, скорее всего, не улучшит производительность при простых задачах, вроде указанного выше перемножения матриц. Если говорить более конкретно, она останется на уровне центрального процессора. Однако для более сложных методов, включающих в себя, как прямые, так и итерационные алгоритмы, его применение может оказать существенное влияние на производительность вычислений.

В качестве примера, рассмотрим метод QR-факторизации для нахождения собственных чисел вещественных несимметричных матриц, применяемый в вычислительной линейной алгебре. Для полного изложения сути метода, можно обратиться к работе [8]. В нашем случае, мы ограничимся математической библиотекой MKL (oneMKL), также предлагаемой компанией Intel, в качестве компоненты для инструментов разработчика oneAPI. Параллельная версия QR-факторизации, на основе пакета LAPACK, входит в модули решения систем линейных алгебраических

уравнений (СЛАУ) этой библиотеки. Особенностью реализации этой версии для компилятора DPC++ является то, что теперь она поддерживает объектно-ориентированный интерфейс, для стандартных классов C++. Но самым главным преимуществом является возможность использования ее в гетерогенных средах. Поэтому в качестве второго примера, возьмем задачу QR-факторизации вещественной несимметричной матрицы, размерностью 10000x10000 элементов. Результаты расчета для данных двойной точности приведены в гистограмме, на рис.4. Как видно из результатов вычислений, наихудшим вариантом оказалось, как раз, использование графического процессора. Самым производительным оказался вариант эмуляции FPGA-акселератора. Очевидно, что библиотека oneMKL хорошо оптимизирована для работы с FPGA-акселераторами.

Рис. 4. - Результат QR-факторизации матрицы, размерностью 10000x10000 элементов

Приведенные примеры использования современных средств вычислений наглядно демонстрируют преимущество компилятора компании Intel – Data ParalelI C++, в ее наборе инструментов разработчика oneAPI. Следует отметить также, что их применение возможно в комбинации с другими

инструментами параллельных вычислений компании Intel [9,10]. В частности, Intel Threading Building Blocks (ITBB), являющегося собственным расширением Intel языка C++ для параллельных вычислений, на основе директив OpenMP. Это особенно важно для наиболее распространенной архитектуры систем с общей памятью, требующих эффективных методов программирования в задачах, ориентированных на использование гетерогенных сред в вычислительной и прикладной математике.

Использование мощностей только центрального процессора уже недостаточно. Становится необходимым привлечение акселераторов вычислений и графических процессоров для повышения производительности математических вычислений. Преимущество инструмента разработки компании Intel – Data Paralell C++, заслуживают пристального внимания со стороны научных организаций и соответствующих факультетов физико-математических специальностей ВУЗов. Сегодня мы стоим перед той проблемой, что написанные за много лет, «тонны кода» нуждаются в распараллеливании для выполнения их не только в многопроцессорных, но и в гетерогенных средах. В противном случае их ждет не только отсутствие значимого ускорения, но и фактическая потеря актуальности для высокопроизводительных вычислений! Написание же новых кодов уже требует другого подхода, учитывающего особенности распараллеливания вычислений на процессорных ядрах различных типов: многоядерных центральных процессоров, графических процессоров, и акселераторов вычислений различного типа.

Литература

1. Баклагин В.Н. Реализация распараллеливания алгоритмических структур, моделирующих экосистему озерных объектов, на

многоядерные процессоры // Инженерный вестник Дона, 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1750.

2. Максютов М. С. Искусство вычислений в эпоху параллельности. М.: URSS, 2021, 208 с.
3. Rinders J. VTune Performance Analyzer Essentials. New York: Intel Press, 2005, 455 p.
4. Reinders J., Ashbaugh B., Brodman J. Data Parallel C++. New York, Apress Open, 2021, 512 p.
5. Aiichiro N. Data Parallel C++ for Heterogeneous Architectures. University of Southern California: 2022, 25p.
6. Kinsner M. Introducing Data Parallel C++. New York: Intel Corporation, 2022, 44 p.
7. Vladimirov A., Asai R., Karpusenko V. Parallel programming and optimization with intel xeon phi coprocessors // Colfax International: 2015, 266 p. URL: eecs.umich.edu/courses/eecs570/hw/phi_intro.pdf
8. Деммель Д. Вычислительная линейная алгебра. Теория и приложения. М.: Мир, 2001, 429 с.
9. Lyashevsky A., Titov A. SYCL Sparklers: Making the Most of C++ and SYCL. New York: Apress Open, 2023, 112 p.
10. Voss M., Asenjio R., Rinders J. C++ parallel programming with threading building blocks. New York: Apress Open, 2019, 312 p.

References

1. Baklagin V.N. Inzhenernyj vestnik Dona, 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1750.

2. Maksyutov M. S. Iskusstvo vy`chislenij v e`poxu parallel`nosti. [The Art of Computing in the Age of Parallelism]. M.: URSS, 2021, p. 208.
3. Rinders J. VTune Performance Analyzer Essentials. New York: Intel Press, 2005, 455 p.
4. Reinders J., Ashbaugh B., Brodman J. Data Parallel C++. New York: Apress Open, 2021, 512 p.
5. Aiichiro N. Data Parallel C++ for Heterogeneous Architectures. University of Southern California: 2022, 25p.
6. Kinsner M. Introducing Data Parallel C++. New York, Intel Corporation, 2022, 44 p.
7. Vladimirov A., Asai R., Karpusenko V. Parallel programming and optimization with intel xeon phi coprocessors. Colfax International, 2015, 266 p. URL: eecs.umich.edu/courses/eecs570/hw/phi_intro.pdf.
8. Demmel` D. Vy`chislitel`naya linejnaya algebra. Teoriya i prilozheniya. [Computational linear algebra. Theory and applications]. M.: Mir, 2001, 429 p.
9. Lyashevsky A., Titov A. SYCL Sparklers: Making the Most of C++ and SYCL. New York: Apress Open. 2023, 112 p.
10. Voss M., Asenjio R., Rinders J. C++ parallel programming with threading building blocks. New York: Apress Open, 2019. 312 p.