ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ГЕТЕРОГЕННЫХ СРЕДАХ

Научная статья

Максютов М.С.[1, *]

[1] Московский государственный строительный университет, Москва, Российская Федерация

* Корреспондирующий автор (mmaxutov[at]gmail.com)

Аннотация

В статье рассматривается хронология средств параллельных вычислений, проводится анализ существовавших средств и методов, а также специфика роста производительности компьютеров. Рассматривается эффективность применения оптимизирующих компиляторов в гетерогенных средах 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. Проводится анализ, как ранее существовавших методов, так и современных методов вычислений, приводятся примеры, как простых алгоритмов вычислений, так и вычислений с использованием математических библиотек для задач вычислительной линейной алгебры.

Ключевые слова: параллельный код, гетерогенная среда, intel data parallel c++, intel oneapi, onemkl, sycl, fcpga акселератор, графический процессор, openmp, mpi.

OPTIMISATION OF PARALLEL CALCULATIONS IN HETEROGENEOUS ENVIRONMENTS

Research article

Maxyutov M.S.[1, *]

[1] Moscow State Civilian Engineering University, Moscow, Russian Federation

* Corresponding author (mmaxutov[at]gmail.com)

Abstract

The article studies the chronology of parallel computing tools, analyses the existing tools and methods, as well as the specifics of computer performance growth. On the basis of the latest developments in the field of parallel computing, in particular at the SYCL abstraction level, the application of optimal parallel computing tools for building applications in the field of computational and applied mathematics is examined. Both earlier and modern methods of computations are analysed, examples of both simple algorithms of computations and computations using mathematical libraries for computational linear algebra problems are presented.

Keywords: parallel code, heterogeneous environment, intel data parallel c++, intel oneapi, onemkl, sycl, fcpga accelerator, GPU, openmp, mpi.

Введение

Сегодня, среди оптимизирующих компиляторов, компиляторы C++/ FORTRAN остаются наиболее эффективными средствами высокопроизводительных параллельных вычислений на уровне своих операторов. На пике развития суперкомпьютерных технологий они позволяют эффективно программировать многопроцессорные рабочие станции и кластеры, обладая самым продвинутым набором базисных параллельных операций и функций, по сравнению с другими языками программирования. Для реализации параллельных вычислений им уже не требуется внешних инструментов и директив, все это реализовано в самих языках.

В старых реализациях предполагалось использование только одного процессора, что и привело к образованию достаточно широкого круга программистов, разрабатывающих последовательные коды. Справедливости ради нужно отметить, что это касается и других языков программирования. С развитием технологий вычислительных систем, в ключе повышения их производительности, стало понятно, что очень скоро идеология «одного CPU» будет забыта, в пользу «суперкомпьютерных» технологий. Технологий, применявшихся ранее только в стенах академических учреждений и лабораторий, и теперь уже выходящих на потребительский рынок. Можно утверждать, что «суперкомпьютерный бум» случился с начала Второго тысячелетия. Считается, что переломный момент перехода на многопроцессорные/многоядерные системы приходится на 2004 год. Хотя уже в 1999 году появляются доступные рабочие станции и серверы под управлением Windows, содержащие несколько процессоров. В первую очередь, это напрямую коснулось FORTRAN, как основного потребителя таких систем, в смысле инструмента для создания высокопроизводительных, математических вычислений. Затем по пути высокопроизводительных вычислений пошел и язык программирования C++. Как следствие, появляется стандарт C++12, а затем более продвинутый C++17, использующие параллельные операции с циклами. В последнем, на момент написания этой статьи, стандарте C++20, появляется модуль. Технология «именованного препроцессора» в виде модуля, применявшаяся в FORTRAN последние тридцать лет, нашла теперь свое место и в C++. Фактически, заменив собой «устаревшие» методы использования заголовочных файлов, в которых приходится бороться с переопределениями.

Анализируя существующие решения на основе последовательного кода, следует отметить, что существует значительная разница между «параллельным кодом» и «параллельными вычислениями». Причем первый не обязательно приводит ко второму. Последовательный код на основе старой технологии, можно «выровнять» к новой, путем использования специальных директив. Фактически, «заставить» работать старые последовательные коды на

современных, многоядерных архитектурах и кластерах. Так работают хорошо известные, хотя тоже уже устаревающие, технологии OpenMP и MPI в виде директив.

Проследим эволюцию последовательного кода, который до сих пор реализуется, отчасти и по вине значительного числа программистов, не желающих осваивать новые технологии, в основном, по причине консерватизма. Добавление дополнительного числа ядер/процессоров в систему, на основе старой идеологии последовательных кодов не привело к значительному повышению их производительности. Хронология последовательного кода, по данным М Хоровица, Ф. Лабонте (2010), К. Руппа (2020) и других исследователей показана на рис. 1. Сегодня он работает в 12 раз медленнее, чем мог бы, если бы увеличение показателей могло продолжаться так же, как например и до 2004 года. Фактически, это равносильно утверждению о том, что код сегодня работает со скоростью, которая могла бы существовать от 8 до 10 лет назад, при сохранении текущей тенденции.

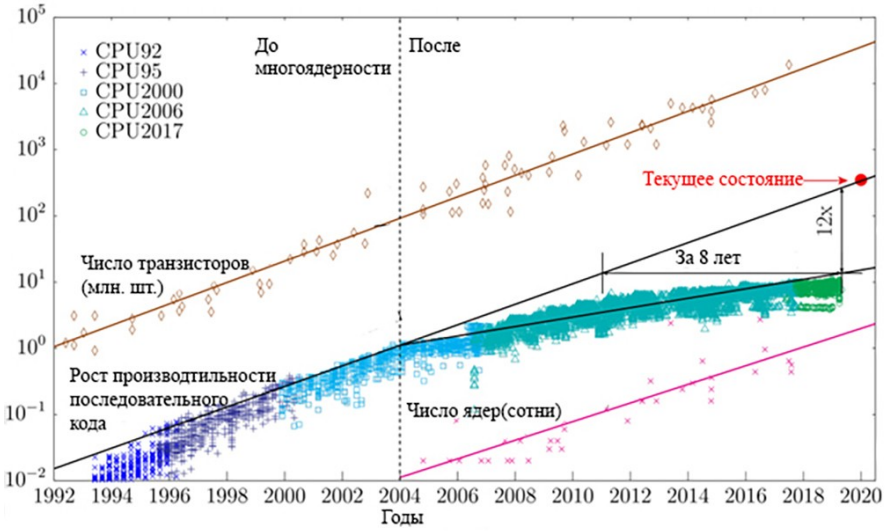

Рисунок 1 - Хронология последовательного кода

Примечание: по данным М Хоровица, Ф. Лабонте (2010), К. Руппа (2020) и др.

Дело в том, что высокопроизводительные системы пошли по пути наращивания ядер и процессорных гнезд, вместо более быстрых CPU. Дополнительным вкладом в повышении производительности стало применение графических акселераторов с большим количеством не программируемых, простых ядер, и оказавшихся более эффективными, чем дополнительные ядра CPU.

Если проследить тенденцию в развитии процессорных технологий за 48 лет (К. Рупп, 2018-2020), то можно заметить, что процессоры не становятся все быстрее и быстрее, как это было на рубеже 80-х и 90-х годов прошлого века. В частности, тактовая частота ядра, и вычислительная эффективность на нем, фактически постоянны, за последние 10 лет. Это отчетливо иллюстрируют графики на рис. 2

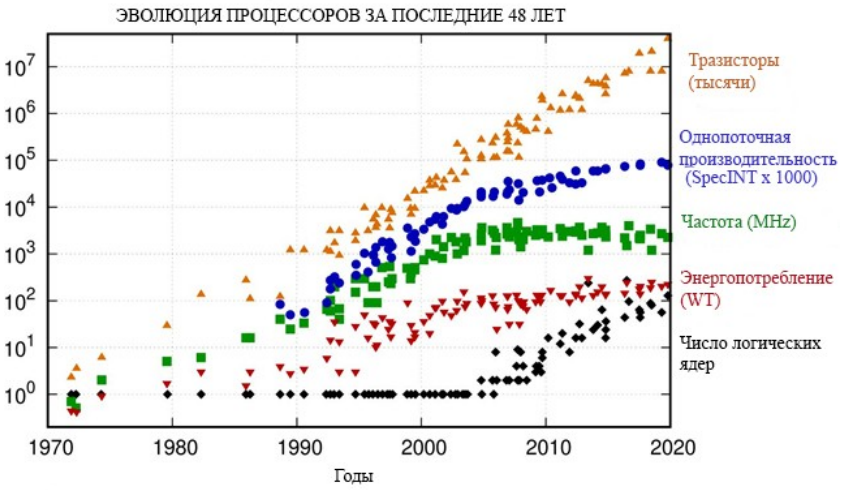

Рисунок 2 - Эволюция процессоров 1972-2020

Сегодня мы стоим перед той проблемой, что написанные за много лет, «тонны кода», нуждаются в распараллеливании для выполнения их на много ядерных системах. В противном случае их ждет не только отсутствие ускорения, но и существенная деградация в производительности вычислений. Написание же новых кодов, требует другой логики мышления учитывающей, как параллельность операций и функций, так и параллельность собственно вычислений. Идеально, когда оба этих представления совпадают.

Конечно, желательно, чтобы компьютер автоматически распараллеливал наш код, хотя бы для вычислений. Многие компиляторы предоставляли такую возможность, предлагая опцию «автопараллельности». Но те, кто использовал ее в реальности, хорошо знают: эта техника в большинстве случаев не только не работала, но была даже весьма опасна своей непредсказуемостью. Ситуация начала выправляться с появлением «параллельных образов» в компиляторе языка программирования Fortran 2008-2018, и в дальнейшем, в стандарте C17 языка программирования C++. В которых параллельные операции и параллельные вычисления слились в единое целое. Центральный процессор, в наше время, уже утрачивает роль высокопроизводительной «числодробилки». Чего не наблюдалось, каких-нибудь 20 лет назад. Все чаще вычисления переводят на различного рода «акселераторы», призванные «помочь», а в большинстве случаев и взять на себя всю трудоемкую вычислительную работу. Если число ядер центрального процессора исчисляется десятками, то число ядер графических и иных акселераторов, на основе «простых процессоров», уже приближается к сотням и тысячам. Фактически, центральному процессору отводится роль «английской королевы (или короля)» в высокопроизводительных вычислениях. Однако существовавшие до сих пор стандарты их программирования (например CUDA и др.) выглядят весьма громоздко и основываются на старых стандартах C/C++. Поэтому были разработаны различные интерфейсы программирования в гетерогенных средах, от различных вендоров, для различных платформ [1]. Однако единого решения, объединяющего их под одним стандартом, до последнего времени не существовало.

Методы вычислений

Группа компаний KHRONOS (www.khronos.org) предложила достаточно простой инструмент параллельных вычислений, в виде свободного кроссплатформенного уровня абстракции SYCL. Он позволяет программировать гетерогенные системы (микропроцессоры с ядрами различных типов) на основе современной реализации языка программирования C++, с соответствующим API, находя в системе соответствующие 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. Наиболее прозрачный и комплексный подход к программированию таких решений был реализован в новом инструменте разработчика Intel oneAPI, посредством компилятора Data Parallel C++. Intel удалось, основываясь на интерфейсе SYCL, использовать все возможности современных архитектур, в частности, основываясь на параллельных функциях стандарта C17, объединить использование графического акселератора и центрального процессора в одном коде для своих устройств. Основы их программирования в SYCL достаточно подробно изложены в книге Д. Риндерса, Б. Асбауха и Д. Бродмана «Data Parallel C++» [1].

Для начала, рассмотрим настройку вычислительной системы для использования Data Parallel C++ на примере рабочей станции HP ZBook Power G9, работающей в ОС Windows 11, сборка 22H2. В качестве микропроцессора в ней, выступает чип Intel Core i7-12700H с интегрированной графикой Intel Iris Graphics Xe. Оперативная память представлена набором модулей DDR5-4800, общим объемом 32 Гб, работающих в двухканальном режиме. Помимо графического адаптера Intel, в конфигурации рабочей станции присутствует еще и графический адаптер Nvidia A1000, с встроенным объемом памяти 4Гб DDR6. Для этого, рассмотрим простой программный код на Data Parallel C++, устанавливающий тип и количество доступных устройств,

```
#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";}
```

Результатом его работы будут сообщении о количестве и спецификации устройств поддерживаемых в данной системе (рис. 3).

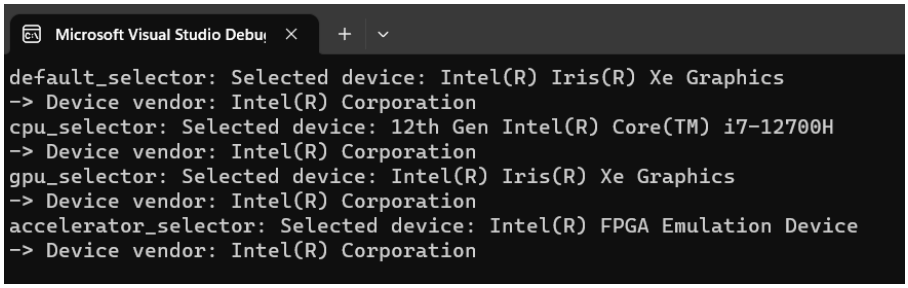

Рисунок 3 - Список доступных устройств

Как видно из списка на рис. 3, пока, напрямую поддерживаются только устройства компании Intel. Хотя необходимо отметить, что уже разработан плагин к oneAPI для некоторых видеоадаптеров Nvidia сторонними разработчиками (реализующий интерфейс CUDA для ОС Linux). Устройством по умолчанию в системе является графический адаптер, вторым устройством идентифицирован центральный процессор, и последним FPGA-акселератор. В последнем случае, конечно, никакого дополнительного устройства в системе нет. Однако этот акселератор эмулируется, на основе имеющихся центрального и графического процессоров.

В предыдущих версиях 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 Data Parallel C++ уже не совместимы с текущей.

Особенностью интегрированных графических адаптеров является возможность использовать оперативную память совместно с центральным процессором. Объем и тип которой ограничен только физически установленным, и поддерживается центральным процессором. Например, по умолчанию, для графического процессора Intel Iris Xe, в указанной рабочей станции, выделяется половина доступной оперативной памяти. Для того чтобы показать вычислительную эффективность инструмента разработки Data Parallel C++ достаточно сравнить их с современным C++, использующим лямбда-функции для параллельных циклов. Например, реализованные в компиляторе Visual C++, для среды разработки Visual Studio 2022.

Тестовые задачи и анализ результатов

В качестве первого примера, рассмотрим стандартную процедуру перемножения двух квадратных матриц, сначала на последовательном коде, затем на параллельном. В Visual C++ возможно использование мощностей только центрального процессора. Исходя из этого вычисления проведем, используя запрос на центральный процессор - *cpu_selector_v*. При этом, важно правильно выбрать размер матриц для того, чтобы выйти за пределы кэша процессора. Поэтому, во всех случаях, размерность матриц выбрана достаточно большой, и составляет 30000x30000 элементов. Для Data Parallel C++ уже возможно использовать графический процессор, используя запрос *gpu_selector_v*, что мы и сделаем для сравнения результатов. Результаты расчетов по обоим вариантам показаны на рис. 4.

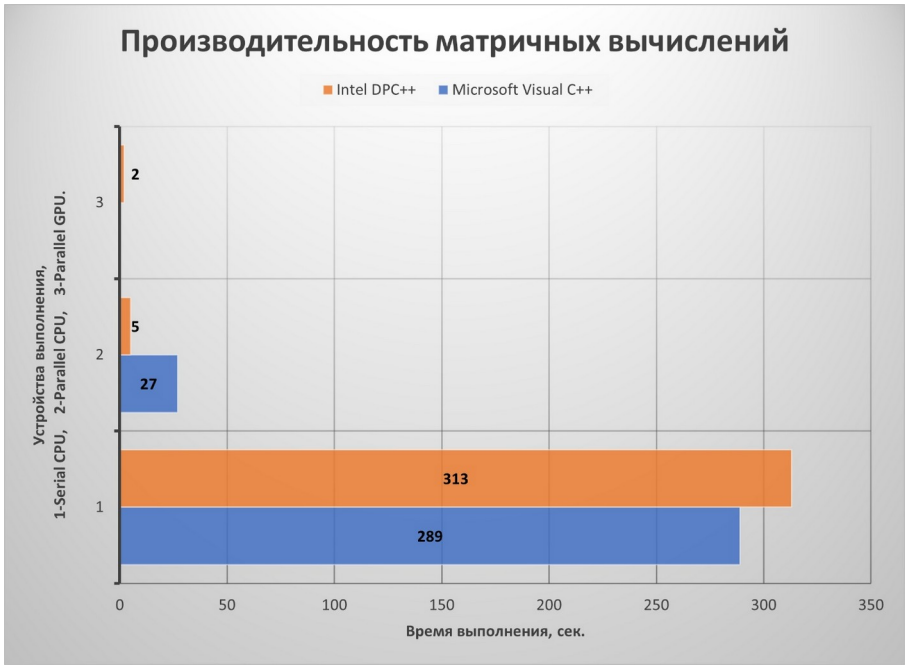

Рисунок 4 - Результат перемножения матриц размерностью 30000x30000 элементов

Как видно из результатов, Microsoft Visual C++ хорошо оптимизирован для выполнения однопоточных приложений (гистограмма 1), и незначительно опережает компилятор Intel. Но, результат выполнения его параллельной версии на 20-поточном процессоре Core i7 12700H(14 физических ядер), значительно уступает

компилятору Intel (гистограмма 2). Выполнение параллельного кода на графическом процессоре позволяет еще больше ускорить вычислений. Соответствующий показатель для Visual C++ в гистограмме 3 отсутствует, ввиду необходимости использования сторонних специализированных библиотек от других вендоров оборудования(Nvidia, AMD). Такой высокий показатель производительности объясняется тем, что графический акселератор, в данной конфигурации (Intel Iris Graphics Xe), содержит 96 исполнительных устройств (процессоров), каждый из которых, в свою очередь, содержит два вычислительных блока способных совершать 8 арифметических операций одинарной точности за такт. Эквивалентное графическим акселераторам Nvidia и AMD число шейдерных процессоров здесь равно 768.

Подводя итог, можно сказать, что графический акселератор позволил добиться повышения производительности, при распараллеливании вычислений, примерно в 150 раз (для двух компиляторов). По сравнению с последовательным кодом на центральном процессоре, в однопоточном режиме.

Однако не следует полагать, что использование графического процессора по умолчанию – это всегда наилучшее решение. Во-первых, графический процессор Intel Iris Xe напрямую не поддерживает данные с двойной точностью в ОС Windows. С двойной точностью можно работать только в режиме эмуляции в ОС Linux. Это довольно странно, так как в предыдущей версии интегрированной графики (Intel UHD Graphics) такое было возможно. Во-вторых, 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. Поэтому целесообразно использовать для вычислений, как центральный, так графический процессоры одновременно. Для этого нам придется использовать еще один механизм управления гетерогенными средами вычислений. В частности, речь пойдет об упоминаемых выше FPGA-акселераторах. В качестве примера одного из них, можно привести сопроцессор Intel Xeon Phi, MIC-архитектуры «Knights Landing», выпуск которого уже прекращен, но последняя версия еще поддерживается. Работа с акселератором происходит в режиме его эмуляции, если он не установлен, но это позволит нам использовать, как весь объем доступной памяти, так и возможность работы с данными двойной точности. Поможет это сделать запрос *accelerator_selector_v*. Применение запроса на акселератор, скорее всего, не улучшит производительность на простых задачах, вроде указанного выше перемножения матриц. 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 книге Д. Деммеля [4]. 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 СЛАУ этой библиотеки. Особенностью реализации этой версии для компилятора Data Parallel C++ является то, что теперь она поддерживает объектно-ориентированный интерфейс, для стандартных шаблонных классов C++(Vector, Array). Но самым главным преимуществом является возможность использования ее в гетерогенных средах. Исходная вещественная матрица для факторизации будет иметь такой же размер, 30000x30000 элементов. Результаты расчета для данных одинарной точности приведены на рис. 5.

Рисунок 5 - Результат QR-факторизации матрицы размерностью 30000x30000 элементов

Как видно из результатов расчета, наихудшим вариантом оказалось, как раз, использование графического процессора. Самым производительным оказался вариант эмуляции FPGA-акселератора. Как видно, библиотека хорошо оптимизирована для работы с FPGA-акселератором. Для указанной конфигурации рабочей станции, возможность вычислений в ОС Windows, с данными двойной точности, поддерживается только в режиме запроса центрального процессора, либо FPGA-акселератора.

Заключение

Приведенные примеры использования современных средств вычислений наглядно демонстрируют преимущество компилятора компании Intel – Data Paralell C++, в ее наборе инструментов разработчика oneAPI. Это особенно важно для наиболее распространенной архитектуры x86-64, 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. Сделать это сегодня, в рамках одного компилятора и целевой архитектуры, при условии свободно распространяемой лицензии, позволяет только набор инструментов Intel oneAPI.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы / References

1. Reinders J. Data Parallel C++ / J. Reinders, B. Ashbaugh, J. Brodman et al. — New York: Apress Open, 2021. — 512 p.
2. Aiichiro N. Data Parallel C++ for Heterogeneous Architectures / N. Aiichiro. — University of Southern California, 2022. — 25 p.
3. Kinsner M. Introducing Data Parallel C++ / M. Kinsner. — Intel Corporation, 2022. — 44 p.
4. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения / Дж. Деммель. — М.: Мир, 2001. — 429 с.
5. Lyashevsky A. SYCL Sparklers: Making the Most of C++ and SYCL / A. Lyashevsky, A. Titov. — New York: Apress Open, 2023. — 112 p.
6. Voss M. Pro TBB: C++ Parallel Programming with Threading Building Blocks / M. Voss, R. Asenjio, J. Rinders. — New York: Apress Open, 2019. — 312 p.
7. Reinders J. VTune Performance Analyzer Essentials / Reinders J. — New York: Intel Press, 2005. — 455 p.
8. Jeffers J. Intel Xeon Phi High Performance Programming / J. Jeffers, J. Reinders, A. Sodani. — Morgan Kaufmann Publishing, 2016. — 662 p.
9. Vladimirov A. Parallel Programming and Optimization with Intel Xeon Phi Coprocessors / A. Vladimirov, R. Asai, V. Karpusenko. — Colfax international, 2015. — 266 p.
10. Максютов М. Искусство вычислений в эпоху параллельности / М. Максютов. — М.: Изд-во URSS, 2021. — 208 с.

Список литературы на английском языке / References in English

1. Reinders J. Data Parallel C++ / J. Reinders, B. Ashbaugh, J. Brodman et al. — New York: Apress Open, 2021. — 512 p.
2. Aiichiro N. Data Parallel C++ for Heterogeneous Architectures / N. Aiichiro. — University of Southern California, 2022. — 25 p.
3. Kinsner M. Introducing Data Parallel C++ / M. Kinsner. — Intel Corporation, 2022. — 44 p.
4. Demmel J. Vychislitel'naja linejnaja algebra. Teorija i prilozhenija [Computational Linear Algebra. Theory and Applications] / J. Demmel. — M.: Mir, 2001. — 429 p. [in Russian]
5. Lyashevsky A. SYCL Sparklers: Making the Most of C++ and SYCL / A. Lyashevsky, A. Titov. — New York: Apress Open, 2023. — 112 p.
6. Voss M. Pro TBB: C++ Parallel Programming with Threading Building Blocks / M. Voss, R. Asenjio, J. Rinders. — New York: Apress Open, 2019. — 312 p.
7. Reinders J. VTune Performance Analyzer Essentials / Reinders J. — New York: Intel Press, 2005. — 455 p.
8. Jeffers J. Intel Xeon Phi High Performance Programming / J. Jeffers, J. Reinders, A. Sodani. — Morgan Kaufmann Publishing, 2016. — 662 p.
9. Vladimirov A. Parallel Programming and Optimization with Intel Xeon Phi Coprocessors / A. Vladimirov, R. Asai, V. Karpusenko. — Colfax international, 2015. — 266 p.
10. Maksjutov M. Iskusstvo vychislenij v jepohu parallel'nosti [The Art of Computing in the Age of Parallelism] / M. Maksjutov. — M.: Publishing House of URSS, 2021. — 208 p. [in Russian]