Ссылка на статью:
// Ученые записки УлГУ. Сер. Математика и информационные технологии. 2023, № 2, с. 104-113.

Поступила: 12.10.2023
Окончательный вариант: 19.11.2023

УДК 004.925

Компьютерное моделирование гидродинамики тонкоплёночных материалов

Тарасов Д.В., Юрьева О.Д. [*]***, Санкин Н.Ю.***

[*] yurjevaod@mail.ru
УлГУ, Ульяновск, Россия

В работе рассмотрено создание библиотеки, включающей в себя набор функций для моделирования гидродинамики концентрационного поля плёночного материала или покрытия. Разработан программный интерфейс приложения (API) для взаимодействия с этой библиотекой, создана демонстрационная программа, показывающая ее возможности. В результате достигнуто ощутимое ускорение вычислений.

Ключевые слова*: GPGPU (General-Purpose computing on Graphics Processing Units), CUDA (Compute Unified Device Architecture), моделирование, тонкопленочные материалы*

Введение

История развития вычислительной техники говорит о превосходстве графических процессоров над центральными процессорами по производительности при решении сложных математических задач. Ещё с начала 2000-х годов графические процессоры, благодаря активному развитию многопоточной архитектуры, достигли значительного отрыва по производительности от классических центральных процессоров. В настоящее время разрыв в производительности в задачах моделирования физических процессов, обработки больших данных, обучения моделей нейронных сетей и других задачах, требующих высокой производительности, может достигать двух порядков [1].

Учитывая закон Амдаля-Уэра [2], который описывает рост производительности с увеличением количества вычислительных потоков, можно отметить, что вычислительные задачи, связанные с моделированием пространства, обладают большим потенциалом для параллелизации. Процессы в каждой из множества точек пространства обрабатываются независимо друг от друга, что позволяет значительно ускорить вычисления при использовании многоядерных архитектур.

Современные центральные процессоры (CPU) обладают многоядерной архитектурой,

однако количество ядер, способных обрабатывать процессы параллельно, обычно сильно ограничено [1]. Напротив, графические процессоры (GPU) могут иметь неограниченное количество потоковых процессоров и кратно большее число потоков, что позволяет обрабатывать гораздо большее количество задач за единицу времени.

С течением времени скорость работы с памятью на GPU также заметно увеличилась, превышая аналогичные показатели у CPU. В настоящее время, даже несмотря на многоядерность и многопоточность, центральные процессоры все еще значительно отстают от графических процессоров по арифметической производительности.

Исходя из этого, подход GPGPU (General-Purpose computing on Graphics Processing Units) является очевидно лучшим выбором для решения задач, требующих высокой производительности и распараллеливания, включая моделирование физических процессов. Данный подход обеспечивает возможность применять мощность графических процессоров для решения научных и инженерных задач, обходя ограничения традиционных вычислительных платформ [3].

Лучшим на данный момент инструментом для реализации GPGPU вычислений является CUDA (Compute Unified Device Architecture), разработанный компанией NVIDIA [4]. Эта платформа предлагает низкоуровневый доступ к аппаратному обеспечению, что обеспечивает большую гибкость и контроль, а также предлагает библиотеки для выполнения часто используемых операций, таких как преобразования Фурье, работу с аппаратно-ускоренными операциями над тензорами, генерацию случайных чисел и другие.

Целью данной работы является создание библиотеки, включающей в себя набор функций для моделирования гидродинамики концентрационного поля плёночного материала или покрытия [5-7]. Кроме этого, в задачи входит разработка API для взаимодействия с этой библиотекой и создание демонстрационной программы, показывающей ее возможности. При этом ставилась задача достичь ощутимого ускорения вычислений.

Методы и результаты

Инструментом решения поставленной задачи должно было быть решение трёхмерного уравнения Навье-Стокса. Однако использование трёхмерного уравнения избыточно для данной задачи. Вместо этого использовалось модифицированное двухмерное уравнение Кана-Хиларда. Более подробное описание перехода от уравнения Навье-Стокса к уравнению Кана-Хиларда представлено в [8].

В данной работе реализовано численное решение модифицированного уравнения Кана-Хиларда, приведенного в форме [3]:

$$\eta \frac{\partial h}{\partial t} = \nabla \left\{ \frac{h^3}{3} \nabla \left[\frac{\partial V}{\partial h} - \sigma \nabla^2 h \right] \right\},$$

где η – вязкость, h – толщина пленки, t – время, σ – коэффициент поверхностного натяжения,

$$\frac{\partial V}{\partial h} = \frac{\alpha}{h^3} - \frac{\beta}{h^9}.$$

При численном решении данного уравнения были использованы три различных метода

аппроксимации оператора Лапласа. Лапласиан первого порядка точности вычислялся по следующей формуле [9]:

$$\Delta f(x,y) = \big(f(x+1,y) + f(x-1,y) - 2f(x,y)\big)/dx^2 + \big(f(x,y+1) + f(x,y-1) - 2f(x,y)\big)/dy^2 .$$

Однако такая численная аппроксимация оператора Лапласа приводила к проблемам со стабильностью и сходимостью решения.

Повышение точности вычисления оператора Лапласа достигалось с помощью полушаговой схемы [9], полушаги аппроксимировались как

$$f(x+1/2,y) = \big(f(x+1,y) + f(x,y)\big)/2.$$

Итоговая полушаговая разностная схема вычисления оператора Лапласа выглядит следующим образом:

$$\Delta f(x,y) = \big(f(x+1/2,y) + f(x-1/2,y) - 2f(x,y)\big)/dx^2 + \big(f(x,y+1/2) + f(x,y-1/2) - 2f(x,y)\big)/dy^2.$$

Такой подход позволил улучшить устойчивость уравнения, однако при шаге по времени больше $5\cdot10^{-4}$, уравнение теряет устойчивость, что сильно ограничивает скорость прохождения процессов системы. Необходимость дальнейшего увеличения точности всё ещё остаётся, однако данной точности решения уже достаточно для полноценного решения уравнения при заданных параметрах.

Лапласиан второго порядка точности вычислялся по следующей формуле:

$$\Delta f(x,y) = \Big((-1/12) * \big(f(x+2,y) + f(x-2,y)\big) + (4/3)\big(f(x+1,y) +$$

$$fx{-}1{,}y{+}{-}5/2{*}fx{,}y/dx2 {+}{-}1/12{*}fx{,}y{+}2{+}fx{,}y{-}2{+}4/3{*}fx{,}y{+}1{+}fx{,}y{-}1{+}{-}5/2{*}fx{,}y/dy2.$$

$$\begin{aligned}\Delta f(x,y) = \Big((-1/12) * \big(f(x+2,y) + f(x-2,y)\big) + (4/3)\big(f(x+1,y) + f(x-1,y)\big) \\ + (-5/2) * f(x,y)\Big)/dx^2 \\ + \Big((-1/12) * \big(f(x,y+2) + f(x,y-2)\big) + (4/3) * \big(f(x,y+1) + f(x,y-1)\big) \\ + (-5/2) * f(x,y)\Big)/dy^2.\end{aligned}$$

Использование Лапласиана второго порядка точности показало наилучшие результаты, позволяя достичь оптимального баланса между точностью и устойчивостью метода, а также ускорить эволюцию моделируемой системы.

Данное уравнения Кана-Хилларда решалось с периодическими граничным условиями.

Исходный код модели, представляющий собой численное решение приведенного уравнения, был написан на языке CUDA для эффективного использования параллелизма, доступного на современных графических процессорах (GPU). Однако, учитывая необходимость в совместимости решения с различными платформами, была также разработана версия программы для выполнения на центральном процессоре (CPU), написанная на C++17. API библиотеки также реализовано на C++17 для обеспечения гибкости и мобильности кода.

Код для решения уравнения был организован в виде класса, предоставляющего необходимые для работы с уравнением методы и атрибуты. Он включает в себя функции для инициализации начальных условий, проведения итераций, а также для записи и вывода результатов. Внутренние подклассы этого класса инкапсулируют различные аспекты функционала, включая выделение и освобождение памяти, генерацию начальных данных в виде случайного шума, а также передачу данных между центральным процессором и графическим процессором.

Далее приведено описание класса, организующего решение уравнения.

```
class Solver {
    private:
        double sigma;          // Параметр поверхностного натяжения в уравнении
        double eta;            // Коэффициент вязкости
        int level_of_accuracy; // Требуемый уровень точности решения
        double dt;             // Шаг по времени
        size_t Nx;             // Количество узлов сетки по оси X
        size_t Ny;             // Количество узлов сетки по оси Y
        size_t save_step;              // Частота сохранения промежуточных результатов вычислений
        double model_end_time;   // Время окончания моделирования

        // Указатели на память устройства
        double* devH;
        double* devH0;
        double* devBuff;

        // Методы для работы с памятью и вычислениями
        void AllocDeviceMemory();
        void FreeDeviceMemory();
        void ConfigKernel();
        void SaveResultsToHost();
        void CallKernelComputation();

    public:
        // Конструктор и деструктор
        Solver(size_t Nx, size_t Ny, double sigma, double eta, double dt, int level_of_accuracy, size_t save_step, double model_end_time);
        ~Solver();

        // Методы для запуска вычислений и генерации начального состояния
        void LaunchComputation(size_t iterations);
        void GenerateStartFluidField(double mean, double stddev);
```

};

В демонстрационной версии программы присутствует расширение основного класса, включающее в себя функции для построения и отображения графического интерфейса пользователя. Для этого использовались библиотеки OpenGL (Open Graphics Library), GLFW (Graphics Library Framework) и IMGUI (Immediate Mode Graphical User Interface). Open GL является кроссплатформенным стандартом для рендеринга 2D и 3D графики. GLFW - это библиотека для работы с окнами и вводом-выводом, которая используется в связке с OpenGL. IMGUI позволяет создавать графические интерфейсы с хорошей отзывчивостью и гибкостью. Итоговая реализация вычислительного процесса вне зависимости от точности, позволяет производить расчёты для пространства размером 16384x16384 точек, используя при этом менее 8GB памяти графического процессора.

Так как CUDA очень чувствительна к качеству кода, были предприняты методы по его оптимизации, не затрагивающие саму математическую модель, а улучшающие качество её реализации.

В ходе улучшения качества кода были проведены следующие виды оптимизации:

1) В CUDA использование памяти − один из основных факторов, влияющих на производительность, поэтому подход к организации памяти значительно влияет на скорость вычислений. Flattening – это преобразование многомерных массивов в одномерные. Такое преобразование улучшает локальность данных и повышает пропускную способность памяти за счёт разгрузки контроллера памяти. Вместе с этим обращения к памяти становятся ещё и последовательными, что тоже положительно влияет на производительность.

2) Выбор оптимального размера блока (или группы) потоков в CUDA существенно повысил производительность кода. Это связано с тем, что размер блока влияет на эффективность использования ресурсов GPU, включая регистры, shared memory и warp-ы.

Shared Memory (разделяемая память): В CUDA разделяемая память - это быстрая память, доступная всем потокам в пределах одного блока. Она используется для обмена данными между потоками в блоке, позволяя ускорить выполнение программы за счет сокращения обращений к глобальной памяти, которая медленнее. Разделяемая память имеет ограниченный объем, поэтому требует тщательного планирования её использования.

Warp: В архитектуре CUDA, warp представляет собой группу из 32 последовательно выполняемых потоков. Все потоки в warp выполняют одну и ту же инструкцию в любой момент времени, но могут работать с разными данными. Эффективное использование warp позволяет максимально загрузить вычислительные ресурсы GPU и повысить производительность программы.

3) Использование спецификатора restricted также внесло дополнительное повышение производительности. Этот спецификатор дает компилятору больше информации для проведения оптимизаций на уровне памяти.

4) Улучшение локальности памяти за счёт отказа от использования унифицированной памяти. Отказ от использования унифицированной памяти также улучшил производительность, поскольку управляемая память может вызывать некоторую накладную работу, связанную с автоматическим управлением данными между хостом и устройством.

5) Объединение вычислений в блок (batch) с помощью cuGraph позволило сократить время, затрачиваемое на переключение контекстов и вызовы функций, сгруппировав несколько итераций алгоритма в один большой пакет операций для последовательного выполнения. Такой "блок" представляет собой набор заранее определенных операций, которые выполняются вместе, что увеличивает эффективность вычислений за счет минимизации задержек. Увеличение размера блока (batch) обычно способствует повышению производительности, но его чрезмерное увеличение может отрицательно сказаться на времени запуска. Обычно размер batch подбирается экспериментально для достижения оптимального баланса между откликом и скоростью обработки данных.

6) Каждый мультипроцессор на GPU имеет ограниченное количество регистров. Эти регистры распределяются между всеми активными потоками на этом мультипроцессоре. Если функция использует больше 32 регистров, нехватка регистров будет компенсироваться за счет использования _global_ памяти, что существенно увеличивает время доступа к этим данным и снижает общую производительность. Ограничение числа регистров, используемых в функции, до 32 позволяет не только более эффективно распределить регистры между потоками, увеличивая количество активных warp'ов на мультипроцессор, но и предотвращает деградацию производительности из-за необходимости обращения к global памяти. Таким образом, эта оптимизация позволяет полнее использовать возможности GPU, улучшая общую производительность вычислений.

Явные указания компилятору для предсказания ветвления с помощью _builtin_assume и _assume_ не дали никакого статистически выявляемого прироста производительности. Такое поведение обусловлено тем, что данная оптимизация в новых версиях nvcc компилятора почти во всех случаях делается в автоматическом режиме.

Использование shared памяти, не дало, а напротив незначительно замедлило вычисления. Стоит отметить, что использование разделяемой памяти оправдано и даёт ускорение в случаях вычислений для небольшого размера сетки, однако для размера моделируемой области более чем 1024x1024 негативно сказывается на производительности в силу избыточной нагрузки на потоковые мультипроцессоры.

Итоговые замеры производительности были приведены для системы размером 2048x2048 и параметрами $\Delta t = 1 \cdot 10^{-4}$, $\sigma = 0.1$, $\Delta x = 1$, $\Delta y = 1$, $\eta = 1$, $h = 2{,}5$ с дисперсией 0,2.

Так как сложность вычисления изменяется с развитием системы, в качестве оценки времени выполнения функций приведены средние значения времени выполнения одной итерации для *batch* = 1000 (см. Таблицу 1).. Тестирование производилось на процессоре IntelCorei7-11800H с частотой 2.3GHz и графическом процессоре Nvidia RTX 3070.

Таблица 1. Сравнение быстродействия вычислительных схем

Время одной итерации, мс					
CPU, полушаговая схема	CPU, схема второго порядка точности	GPU, полушаговая схема	GPU, схема второго порядка точности	GPU, полушаговая схема, оптимизированное	GPU, схема второго порядка точности, оптимизированное

64,967	93,19	1,875	2,132	1,786	1,895
64,927	93,157	1,877	2,127	1,787	1,895
64,95	93,111	1,879	2,122	1,786	1,895
64,948	93,191	1,882	2,131	1,787	1,895
64,988	93,189	1,884	2,131	1,787	1,895
64,901	93,115	1,882	2,126	1,796	1,896
64,987	93,146	1,884	2,127	1,797	1,895
64,904	93,19	1,882	2,128	1,796	1,895
64,987	93,162	1,883	2,134	1,797	1,895
64,909	93,176	1,884	2,136	1,796	1,895

При сравнении среднего времени выполнения одной итерации вычислений на CPU для полушаговой разностной схемы и для разностной схемы второго порядка точности выяснилось, что при увеличении порядка точности схемы скорость вычислений на центральном процессоре снизилась на 43.44% (рис. 1).

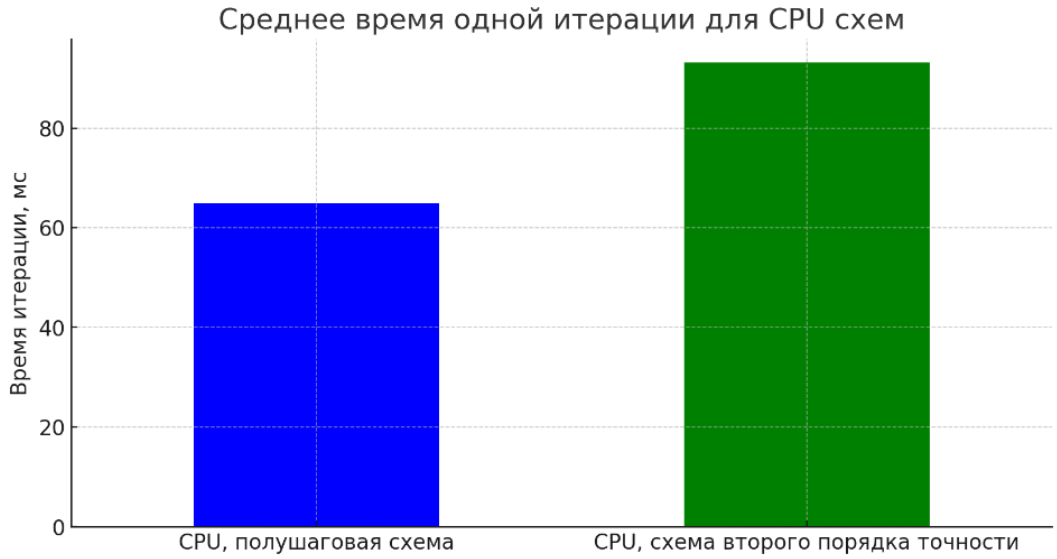

Рис. 1. Сравнение среднего времени выполнения одной итерации вычислений на CPU для полушаговой разностной схемы и для разностной схемы второго порядка точности

Время одной итерации с разностной схемой второго порядка точности после оптимизации на GPU почти сравнялось с временем выполнения одной итерации алгоритма с полушаговой разностной схемой без оптимизации (рис. 2).

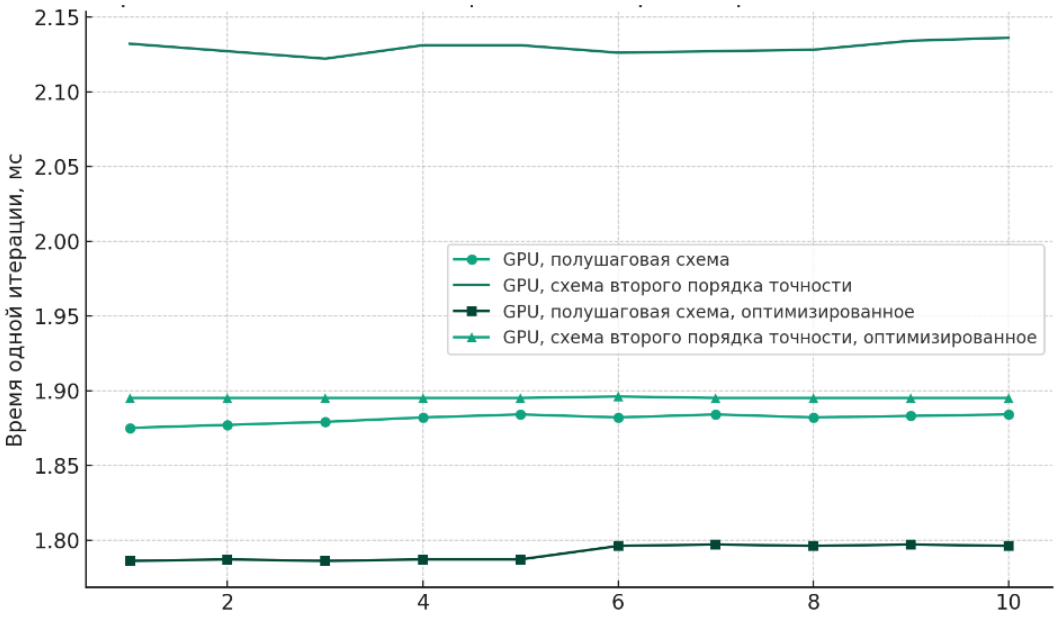

Рис. 2. Сравнение времени выполнения одной итерации вычислений с использованием GPU ускорения

Перенос вычислений с центрального процессора на графический сопроцессор позволило добиться ускорения вычислений до 35 раз.

Общая схема работы разработанной библиотеки представлена на рис. 3.

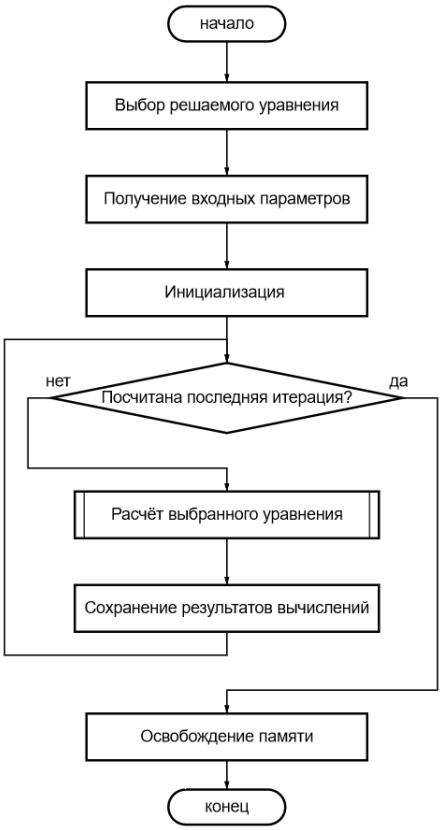

Рис. 3. Общая схема работы алгоритма

Заключение

Оптимизация кода оказывается неотъемлемой частью эффективной работы со сложными вычислительными задачами. Несмотря на все увеличивающиеся вычислительные мощности современной техники, затраты времени на реализацию оптимизационных стратегий оправдывают себя, особенно при длительных вычислениях.

Некоторые разработчики могут пренебрегать оптимизацией, не видя в этом необходимости, однако как показывают экспериментальные данные, даже небольшое улучшение производительности в результате оптимизации кода может привести к значимой экономии времени. Таким образом, оптимизация кода не является избыточной или второстепенной задачей. Это важная инвестиция в будущее, позволяющая обеспечить устойчивость и масштабируемость решений при работе с большими объемами данных или сложными вычислительными задачами.

Представленная в работе программа моделирования гидродинамики тонкоплёночных материалов может быть полезна при предсказании свойств наносимого покрытия и при определении необходимых условий для получения заданных характеристик покрытия, что может оказаться важным в промышленном производстве и исследованиях материалов. При ее разработке проведена оптимизация кода, позволившая существенно снизить затраты времени.

Список литературы

1. Filkenshtein I. *CPU, cloud virtual machines, and noisy neighbors: the limits of parallelism* [Электронный ресурс]. Режим доступа: https://pythonspeed.com/articles/cpu-limits-to-speed/ (дата обращения 21.09.2023).
2. Amdahl's law [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Amdahl%27s_law (дата обращения 21.09.2023).
3. L'vov P. E., Svetukhin V. V.. Phase-field simulation of radiation-induced phase transition in binary alloys // *Modelling and simulation in materials science and engineering*. 2021. V.29, no 3, 035013. DOI: 10.1088/1361-651X/abe177.
4. NVIDIA CUDA Toolkit Documentation. Электронный ресурс. Режим доступа: https://docs.nvidia.com/cuda/index.html (дата обращения 21.09.2023).
5. Becker J., Grün G., Seemann R., Mantz H., Jacobs K., Mecke K. R., Blossey R. Complex dewetting scenarios captured by thin-film models. *Nat Mater.* 2003, no 2(1), p.59-63. DOI: 10.1038/nmat788.
6. Thiele U. Recent advances in and future challenges for mesoscopic hydrodynamic modelling of complex wetting // *Colloids and Surfaces*. 2018. A 553, p. 487–495.
7. Alizadeh Pahlavan A., Cueto-Felgueroso L., Hosoi A. E., McKinley G. H., Juanes R. Thin films in partial wetting: stability, dewetting and coarsening // *Journal of Fluid Mechanics*. 2018. V. 845, p. 642–681. DOI: 10.1017/jfm.2018.255.
8. Bonn D., Ross D. Wetting transitions // *Reports on Progress in Physics*. 2001. V. 64, p.1085–1163. DOI: 10.1088/0034-4885/64/9/202.

9. Самарский А. А., Вабищевич П. Н., Самарская Е. А. *Задачи и упражнения по численным методам: Учебное пособие.* М.: Эдиториал УРСС, 2000. 208 с. ISBN: 5-8360-0158-8.

Computer modeling of hydrodynamics of thin-film materials

Tarasov, D.V., Yurieva, O.D.*[*], Sankin, N.Yu.

[*] yurjevaod@mail.ru

Ulyanovsk State University, Russia

The paper deals with the creation of a library containing a set of functions for modeling the hydrodynamics of the concentration field of a film material or coating. An application program interface (API) for interacting with this library has been developed, and a demonstration program showing its capabilities has been created. The result is noticeable computational acceleration.

***Keywords**: GPGPU (General-Purpose computing on Graphics Processing Units), CUDA (Compute Unified Device Architecture), modeling, thin-film materials*