«Оптика атмосферы и океана», **24**, № 5 (2011)

УДК 004.94, 004.424, 519.633, 519.683, 535.12

Численное моделирование распространения оптических волн с использованием технологий параллельного программирования

П.А. Коняев, Е.А. Тартаковский, Г.А. Филимонов*

Институт оптики атмосферы им. В.Е. Зуева СО РАН
634021, г. Томск, пл. Академика Зуева, 1

Поступила в редакцию 20.12.2010 г.

Рассмотрены методы и особенности применения параллельных алгоритмов для численного моделирования распространения оптических волн. Скалярное параболическое уравнение для комплексной амплитуды поля монохроматической волны решалось численно методом Фурье в случае однородного уравнения и методом расщепления в сочетании с методом Фурье для неоднородного уравнения. Были разработаны два варианта параллельных алгоритмов — по технологии OpenMP для многоядерных процессоров с библиотекой MKL фирмы Intel и по технологии CUDA для графических ускорителей фирмы NVIDIA. Сравнение эффективности алгоритмов между собой и со стандартным последовательным двумерным алгоритмом БПФ из библиотеки FFTW проводилось путем вычисления среднего числа решений тестовых задач в секунду.

Показано, что скорость работы параллельных алгоритмов существенно выше (в десятки раз) по сравнению со стандартным последовательным алгоритмом, причем разница тем больше, чем больше размерность решаемой задачи. Сравнение параллельных алгоритмов между собой выявило следующую картину: в задачах с размерами расчетных сеток до величин 1024×1024 лидировал подход, основанный на технологии OpenMP, в то время как на больших сетках (1024×1024 и более) быстрее работал алгоритм, построенный по технологии CUDA.

Обсуждаются полученные результаты и приводятся рекомендации по переходу от последовательных алгоритмов к параллельным.

Ключевые слова: компьютерное моделирование, параллельные алгоритмы, скалярное волновое уравнение, численные методы; OpenMP, CUDA, parallel programming, wave propagation simulation, computational optics.

Введение

Со времени опубликования первых работ по компьютерному исследованию задач волновой физической оптики прошло уже более 30 лет [1–3]. За эти годы методы численного моделирования волновых процессов получили широкое распространение во многом благодаря прогрессу вычислительных процессоров, которые эволюционировали по эмпирическому «закону Мура» — удвоение мощности каждые 2 года [4]. Исследователи получили возможность изучать тонкие физические явления, в том числе и нелинейные, решая уравнения численными методами на многомерных сетках с большим числом измерений и узлов сеток [5].

В последнее время закон Мура, видимо, «перестает работать» — развитие процессорной техники идет по пути создания многоядерных процессоров и процессорных комплексов — кластеров. В таких вычислительных системах дальнейший рост эффективности происходит за счет того, что вычисления можно проводить параллельно, т.е. одновременно. Для реализации соответствующих алгоритмов интенсивно развиваются технологии параллельного программирования, такие как OpenMP [6], MPI и CUDA [7]. Они позволяют получать многократное ускорение вычислений в задачах, допускающих распараллеливание операций.

К таким задачам, которые допускают распараллеливание операций, безусловно, относятся численные решения скалярных уравнений волновой оптики — задачи дифракции и распространения волн. Однако существующие алгоритмы решения этих уравнений построены по принципам последовательного программирования и предназначены для выполнения одним процессором. Для применения в многоядерных процессорных комплексах и графических ускорителях эти алгоритмы необходимо модифицировать с учетом специфики решаемой задачи, а также конфигурации вычислительного оборудования.

В данной статье рассматриваются методы и особенности применения параллельных алгоритмов для численного моделирования распространения оптических волн в однородных и неоднородных средах. Представлены результаты расчетов, и приводится

* Петр Алексеевич Коняев (petrkonyaev@gmail.com); Евгений Александрович Тартаковский (flar@sibmail.com); Григорий Алексеевич Филимонов (fga@iao.ru).

2*.

сравнение эффективности трех подходов: стандартного последовательного и двух параллельных — OpenMP для многоядерных процессоров фирмы Intel и CUDA для графических ускорителей фирмы NVIDIA.

1. Однородное параболическое уравнение и его решение методом Фурье

Однородное параболическое уравнение для скалярной комплексной амплитуды $U(x, y, z)$ описывает процесс распространения монохроматической оптической волны вдоль оси z в приосевом приближении в среде без вариаций показателя преломления и имеет вид

$$2ik_0\frac{\partial U(x,y,z)}{\partial z}+\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)U(x,y,z)=0, \quad (1)$$

где $k_0 = 2\pi/\lambda$ — волновое число; λ — длина волны.

Аналитическое решение этого уравнения методом Фурье хорошо известно. Рассмотрим алгоритм получения частного решения поля в плоскости $z = Z$ по заданному полю в плоскости $z = 0$ в дискретном виде.

1) Вычислить двумерное дискретное преобразование Фурье (ДПФ) от комплексной амплитуды поля $U(x_k, y_l, 0)$, заданной в дискретных узлах сетки с шагом Δx в плоскости $z = 0$:

$$F(m,n,0)=\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}U(k,l)\exp\left[-i2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)\right], \quad (2)$$

где M и N — число узлов сетки; m и n — индексы пространственных частот.

2) Умножить полученный пространственный спектр поля на частотную характеристику однородной среды в параболическом приближении:

$$F(m,n,Z)=F(m,n,0)\exp\left\{-i\frac{Z}{2k_0}\left[\frac{4\pi^2}{MN\Delta x^2}(m^2+n^2)\right]\right\}. \quad (3)$$

3) Вычислить обратное двумерное ДПФ от пространственного спектра $F(m, n, Z)$ и получить комплексную амплитуду поля в плоскости $z = Z$:

$$U(k,l,Z)=\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}F(m,n,Z)\exp\left[i2\pi\left(\frac{km}{M}+\frac{ln}{N}\right)\right]. \quad (4)$$

Для вычисления ДПФ стандартным является алгоритм быстрого преобразования Фурье (БПФ). Число операций в двумерном БПФ пропорционально $\sim NM\log_2(NM)$, поэтому для больших значений M и N время вычислений может быть существенным. Несмотря на то что алгоритм БПФ с момента его появления [8] многократно улучшался различными авторами, наиболее существенный прирост в скорости достигнут для параллельных версий этого алгоритма.

Рассмотрим два варианта параллельной реализации решения задачи дифракции с использованием алгоритмов БПФ фирмы Intel из библиотеки Intel Math Kernel Library 10.3 и фирмы NVIDIA из библиотеки CUFFT CUDA Toolkit 3.2 и проведем сравнение их быстродействия со стандартным алгоритмом Fastest Fourier Transform in the West (FFTW) [9].

Библиотека Intel Math Kernel Library (MKL) представляет собой набор функций линейной алгебры, быстрого преобразования Фурье и векторной математики. В ней содержится несколько разных реализаций алгоритма БПФ, имеющих различную производительность и точность вычислений. Функции библиотеки являются версиями алгоритма БПФ, оптимизированными специально для микропроцессоров, совместимых с микропроцессорами Intel. На стадии выполнения MKL автоматически определяет модель процессора и запускает соответствующую версию вызываемой функции, что гарантирует максимальное использование возможностей процессора и максимально возможную производительность. Параллелизм достигается за счет работы в многопоточном режиме, при параллельном выполнении отдельным ядром отдельного потока.

Для функций преобразования LAPACK, BLAS (третьего уровня) и дискретных преобразований Фурье (DFT) также допускается распараллеливание по стандарту OpenMP. При распараллеливании алгоритма БПФ на центральном процессоре используются технология многопоточного программирования, реализованная в MKL, а также технология OpenMP для развертки многомерных циклов.

В библиотеке CUFFT, так же как и в MKL, применяются несколько различных алгоритмов БПФ, используемых в зависимости от свойств входной матрицы. Во всех случаях задача разбивается на подзадачи, представляющие собой вычисляемые параллельно элементарные преобразования над матрицей меньшего размера. Все алгоритмы используют разделяемую память (shared memory) для ускорения процесса вычислений. Наилучшая производительность достигается в том случае, если размер входной матрицы удовлетворяет двум критериям: возможность размещения матриц элементарных преобразований в разделяемой памяти CUDA и выбор размера матрицы из условия $N = M = 2^m$.

При соблюдении данных критериев также достигается наибольшая точность вычислений, благодаря эффективной реализации численного алгоритма. В противном случае ($N = M \neq 2^m$) задействованы более медленные и менее точные алгоритмы БПФ, использующие также более медленную глобальную память графического адаптера.

Рассмотрим более подробно реализацию алгоритма решения задачи дифракции с использованием CUDA Toolkit 3.2. На вход в процедуру поступают данные двух типов: матрица поля (field) — двумерный массив комплексного типа; матрица

фильтрации (filter) — одномерный массив комплексного типа. Из пользовательской программы вызывается CPU-часть алгоритма, в которой проводится инициализация GPU-части алгоритма:

```
Diffractor(field, filter, size);.
```

Далее выделяется необходимый объем графической памяти:

```
cudaMalloc((void**)&pDeviceField,
size*size*sizeof(cufftComplex));
cudaMalloc((void**)&pDeviceFilter,
size*sizeof(cufftComplex));.
```

В графическую память копируются необходимые данные из оперативной памяти:

```
cudaMemcpy(pDeviceField, field,
size*size*sizeof(cufftComplex),
cudaMemcpyHostToDevice);
cudaMemcpy(pDeviceFilter, filter,
size*sizeof(cufftComplex),
cudaMemcpyHostToDevice);.
```

Далее определяется размерность блоков BLOCK_SIZE и сетки блоков для kernel-функции фильтрации:

```
dim3 dimBlock (BLOCK_SIZE, BLOCK_SIZE);
dim3 dimGrid (size / dimBlock.x, size / dimBlock.y);.
```

Kernel-функция — это функция, вызываемая отдельно для каждого потока. В ней программируются действия, которые должен выполнять определенный поток, принадлежащий определенному блоку, над определенным элементом. Вычисляемый элемент индексируется через индекс блока в сетке и индекс потока в блоке:

```
int tx = threadIdx.x; // Индекс потока по x.
int ty = threadIdx.y;
int bx = blockIdx.x; // Индекс блока по x.
int by = blockIdx.y;
int bdx = blockDim.x; // Размер блока по x.
int bdy = blockDim.y;
int elemnum = BLOCK_SIZE*by + size*bdx*bx +
tx*size + ty;.
```

При таком способе программирования исчезает потребность в циклах по измерениям матриц. Итоговая kernel-функция описывает однотипные действия, которые можно выполнять параллельно произвольным числом потоков над произвольным числом элементов, кратным размерности блока.

После завершения этапа инициализации из CPU-части алгоритма вызывается функция прямого БПФ из библиотеки CUFFT:

```
cufftHandle plan;
cufftPlan2d(&plan, size, size, CUFFT_C2C);
cufftExecC2C(plan, pDeviceField, pDeviceField,
CUFFT_FORWARD);.
```

Управление передается GPU, на котором проходит исполнение по вышеописанной схеме. После исполнения прямого БПФ управление возвращается CPU.

Далее из CPU-части алгоритма вызывается kernel-функция фильтрации:

```
MxVcomplex_dot_kernel<<<dimGrid,
dimBlock>>>
(pDeviceField, pDeviceFilter, size);,
```

в которой производится параллельное поэлементное умножение полученного спектра на матрицу фильтрации.

После вызова kernel-функции управление передается графическому процессору. Во время исполнения потоком kernel-функции производится копирование блока матрицы, которому принадлежит поток, из глобальной памяти в разделяемую:

```
__shared__ cufftComplex
sharedField[BLOCK_SIZE][BLOCK_SIZE];
sharedField[ty][tx] = pDeviceField[elemnum];.
```

Дальнейшие вычисления каждого из потоков внутри блока проводятся над элементами в разделяемой памяти:

```
sharedField[ty][tx].x = . . . . . .;.
```

После окончания вычислений результат копируется обратно в глобальную память:

```
pDeviceField[elemnum] = sharedField[ty][tx];.
```

Разделяемая память работает намного быстрее глобальной, поэтому такой механизм позволяет ускорить процесс вычислений. После завершения исполнения управление возвращается CPU.

Далее из CPU-части алгоритма вызывается обратное БПФ:

```
cufftExecC2C(plan, pDeviceField, pDeviceField,
CUFFT_INVERSE);
```

управление передается на GPU, процедура исполняется, после исполнения управление возвращается CPU.

На этом этапе в графической памяти хранится итоговый результат, который копируется из графической памяти в оперативную:

```
cudaMemcpy (field, pDeviceField,
sizeof(cufftComplex)*size*size,
cudaMemcpyDeviceToHost);.
```

После копирования выделенная графическая память освобождается:

```
cudaFree(pDeviceField);
cudaFree(pDeviceFilter);.
```

Сравнение быстродействия всех трех алгоритмов проводилось в следующей конфигурации компьютерного оборудования: процессор Intel Core i7-860, видеокарта XFX GTX-285, материнская плата Gigabyte P55A-UD4. Число процессоров видеокарты равно 220, число вычислительных потоков процессора могло варьироваться от 1 до 8.

На рис. 1, *а* приведен график зависимости числа решений двумерной задачи дифракции в секунду (частота кадров) от размеров матрицы. Графики убедительно показывают существенный прирост

в скорости (в десятки раз!) при переходе от стандартного последовательного алгоритма к параллельным.

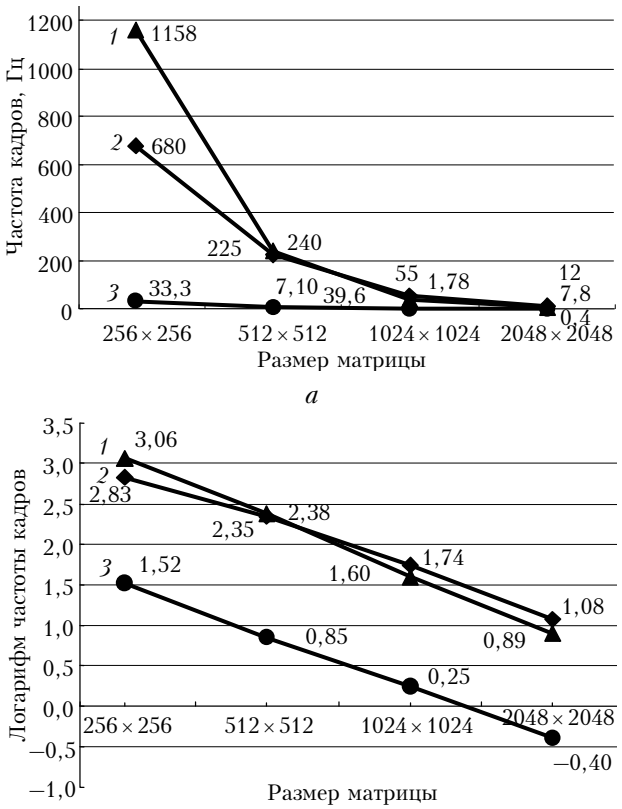

Рис. 1. Число решений задачи дифракции в секунду (частота кадров) в зависимости от размеров матрицы – линейный масштаб (*а*) и логарифмический масштаб (*б*): *1* – вычисления на MKL; *2* – вычисления на CUDA; *3* – последовательные вычисления

На рис. 1, *б* показаны те же зависимости в логарифмическом масштабе, при этом можно заметить, как по мере увеличения размера матрицы выше чем 512×512 элементов растет преимущество в скорости алгоритма на основе библиотеки CUFFT CUDA Toolkit.

2. Неоднородное параболическое уравнение и его решение методом расщепления

Неоднородное параболическое уравнение для комплексной амплитуды $U(x, y, z)$ описывает распространение монохроматической оптической волны вдоль оси z в приосевом приближении в среде с вариациями показателя преломления $n_1(x, y, z)$ и имеет вид

$$2ik_0 \frac{\partial U(x,y,z)}{\partial z} + \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + 2k_0^2 n_1(x,y,z) \right) U(x,y,z). \quad (5)$$

Для численного решения этого уравнения известен и широко применяется метод расщепления [2, 10]. Суть метода состоит в разбиении эволюционной переменной z на последовательность шагов $N_z = Z/\Delta z$ и решении на каждом шаге эквивалентной системы уравнений:

$$\frac{\partial U(x,y,z)}{\partial z} = \frac{i}{2k_0} \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) U(x,y,z),$$

$$\frac{\partial U(x,y,z)}{\partial z} = ik_0 n_1(x,y,z) U(x,y,z). \quad (6)$$

Такая схема расщепления (одноциклическая) является несимметричной и для некоммутирующих операторов имеет первый порядок аппроксимации по z. Более точная двухцикличная схема требует лишь одного дополнительного полушага на последнем цикле расщепления, при этом имеет второй порядок аппроксимации [2, 10]. Рассмотрим алгоритм получения решения уравнения (5) в дискретном виде.

1. Решить задачу дифракции (уравнение (1)) для комплексной амплитуды поля $U(x, y, 0)$ на полушаге $\Delta z/2$, далее последовательно выполнять следующие два пункта алгоритма для значений $j = 1, \ldots N_z$.

2. Умножить полученное решение на фазовый множитель, соответствующий решению задачи рефракции в слое неоднородной среды Δz:

$$\tilde{U}(x,y,z_j) = U(x,y,z_j) \exp\left\{ -ik_0 \int_{z_j}^{z_j+\Delta z} n_1(z)dz \right\}. \quad (7)$$

3. Решить задачу дифракции для комплексной амплитуды поля $U(x, y, z_j)$ на шаге Δz, если $j \neq N_z$, или на полушаге $\Delta z/2$, если $j = N_z$.

Вариант параллельной реализации решения задачи распространения с использованием библиотеки CUFFT CUDA Toolkit 3.2 выглядит следующим образом.

На вход алгоритма поступают данные трех типов: матрица поля (field) – двумерный массив комплексного типа; матрица фильтрации (filter) – одномерный массив комплексного типа; набор фазовых экранов для каждого шага рефракции (phase) – трехмерный массив вещественного типа.

Из пользовательской программы вызывается CPU-часть алгоритма, в которой проводится инициализация GPU-части алгоритма:

```
Propagator(field, filter, phase, size, steps);.
```

В дополнение к соответствующим шагам решения задачи дифракции выполняются следующие действия.

Выделяется дополнительно память под массив, содержащий набор фазовых экранов:

```
cudaMalloc((void **)&pDevicePhase,
    size*size*sizeof(cufftReal));.
```

В графическую память дополнительно копируются необходимые данные из оперативной памяти:

```
cudaMemcpy(pDevicePhase,phase,size*size*steps*size
    of(cufftReal), cudaMemcpyHostToDevice);.
```

Далее циклически выполняется последовательность шагов дифракции и рефракции.

Шаг рефракции: производится параллельное поэлементное умножение комплексной матрицы поля на экспоненциальную функцию соответствующего шага рефракции:

```
Refraction_kernel<<<dimGrid, dimBlock>>>
    (pDeviceField, pDevicePhase, size);.
```

Шаг дифракции: выполняется последовательность действий, соответствующая алгоритму дифракции, описанному ранее.

После выполнения алгоритма выделенная графическая память освобождается.

На рис. 2 приведена блок-схема алгоритма решения задачи распространения с использованием библиотеки Intel MKL.

Аналогичная блок-схема алгоритма на основе библиотеки CUFFT CUDA Toolkit изображена на рис. 3. Из сравнения этих диаграмм видно, что алгоритм, основанный на библиотеке CUFFT, имеет гораздо больше операций работы с памятью, на которые уходит достаточно много времени. Это объясняет поведение зависимостей на рис. 1 — при малых значениях размера сетки алгоритм MKL имеет преимущества в скорости по сравнению с CUDA.

На рис. 4 приведен график зависимости числа решений двумерной задачи распространения в секунду (частота кадров) от числа разбиений продольной переменной для фиксированного размера сетки 1024×1024. И снова графики показывают существенную разницу в скорости (почти в 30 раз) между стандартным последовательным и параллельными алгоритмами.

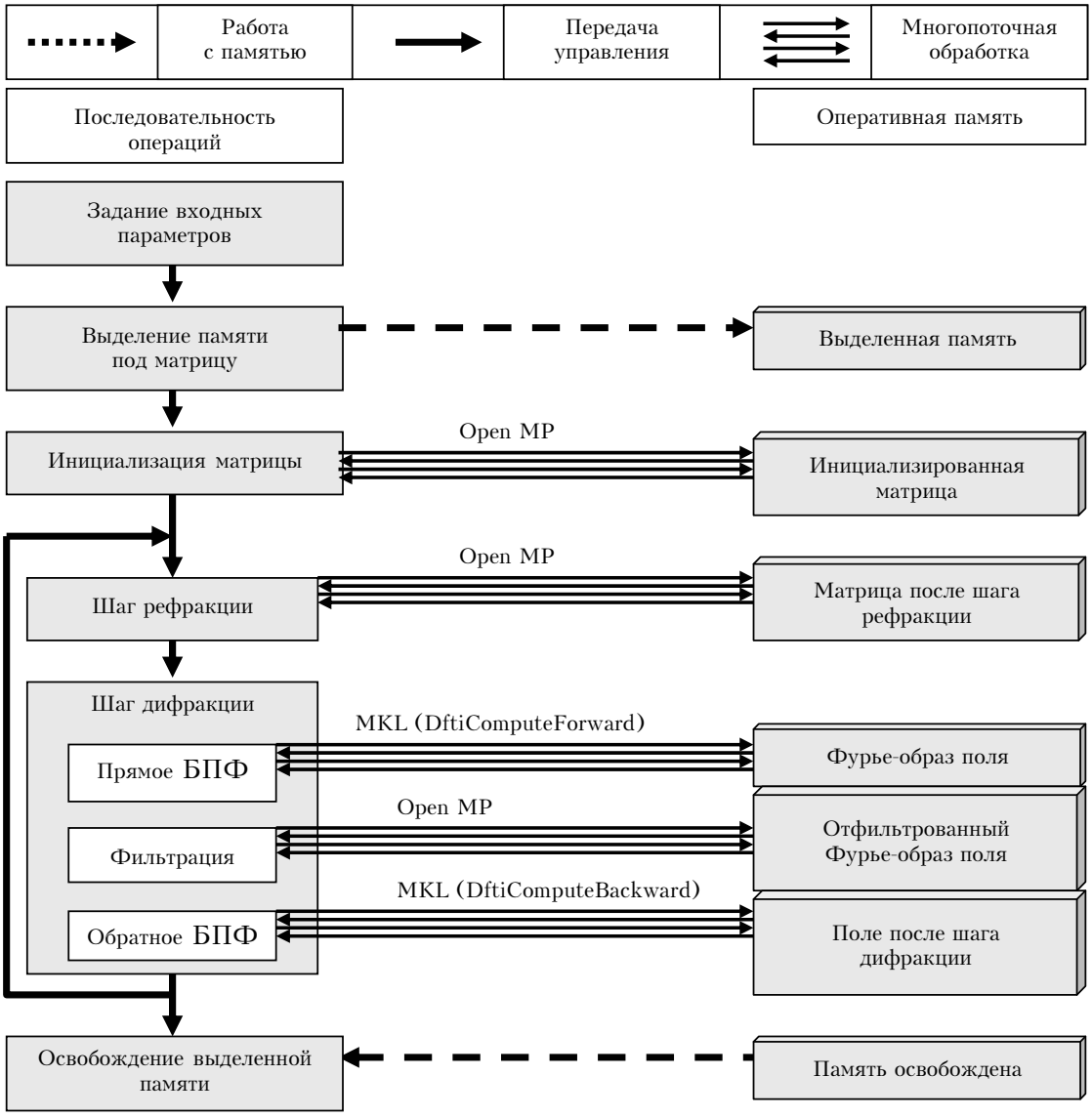

Рис. 2. Блок-схема алгоритма решения задачи распространения с использованием библиотеки Intel MKL

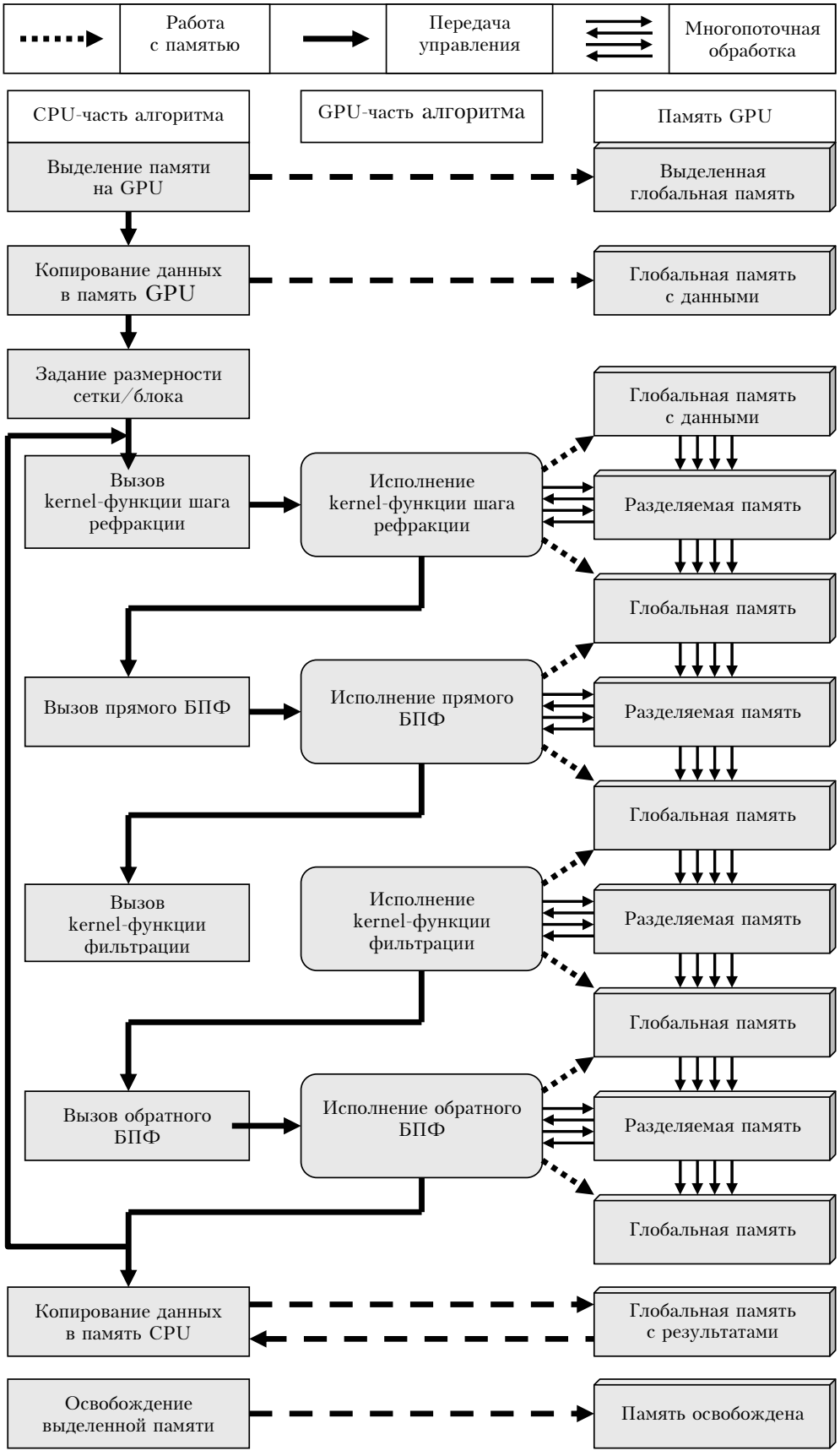

Рис. 3. Блок-схема алгоритма решения задачи распространения с использованием библиотеки CUFFT CUDA Toolkit

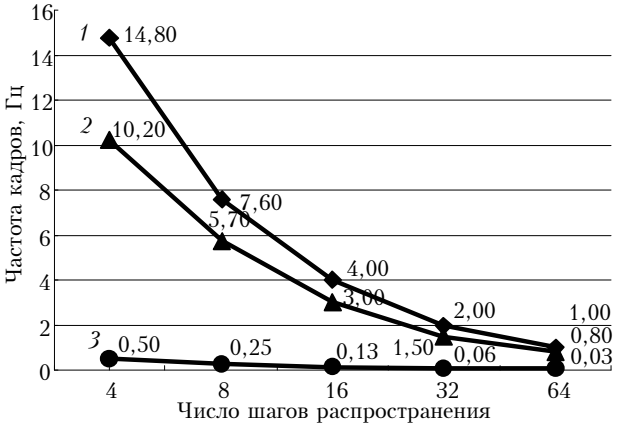

Рис. 4. Число решений задачи распространения в секунду (частота кадров) на сетке 1024×1024 для различного числа разбиений продольной переменной (шагов распространения): *1* — вычисления на CUDA; *2* — вычисления на MKL; *3* — последовательные вычисления

Заключение

В данной статье были рассмотрены методы и особенности применения параллельных алгоритмов для численного моделирования распространения оптических волн в однородных и неоднородных средах. Представлены результаты расчетов и сравнение эффективности трех подходов — стандартного последовательного (алгоритм FFTW), OpenMP для многоядерных процессоров Intel и CUDA для графических ускорителей фирмы NVIDIA. Эти результаты убедительно показывают преимущества в скорости параллельных алгоритмов (в десятки раз) по сравнению с традиционными методами численного решения задач дифракции и распространения волн. При этом использование графических ускорителей дает преимущество в скорости перед многоядерными процессорами в наиболее трудоемких задачах. Дальнейшее развитие методов параллельного программирования предполагается в объединении технологий на основе CPU и GPU в единую систему, усиливающую достоинства каждого из методов по отдельности.

Разумеется, переход от последовательного к параллельному программированию требует от разработчиков программного обеспечения достаточно много усилий. Особенно это касается применения технологии CUDA. Для того чтобы модификация уже написанных и отлаженных программ проходила как можно легче, возможна реализация основных трудоемких алгоритмов в виде библиотек динамической компоновки (DLL) с последующим включением вызовов необходимых функций в исходный текст программ. Такая технология позволяет минимизировать работу по модификации кода и тем самым ускорить внедрение методов параллельного программирования в практику научных исследований и компьютерного моделирования.

1. *Sziklas E.A., Siegman A.E.* Mode calculations in unstable resonators with flowing saturable gain. II. Fast Fourier Transform method // Appl. Opt. 1975. V. 14, N 8. P. 1874–1889.
2. *Fleck J.A., Morris J.R., Feit M.D.* Time-dependent propagation of high energy laser beams through the atmosphere // Appl. Phys. A. 1976. V. 10, N 2. P. 129–160.
3. *Коняев П.А.* Численное решение задачи дифракции на случайном фазовом экране // V Всесоюз. симпоз. по распространению лазерного излучения в атмосфере. Ч. II. Томск: ИОА СО АН СССР, 1979. С. 120–122.
4. *Konyaev P.A., Lukin V.P., Sennikov V.A.* Effect of phase fluctuations on propagation of the vortex beam // Proc. SPIE. 2007. V. 6731. C. 67311A1–A6.
5. *Moore G.* Litography and the future of Moore's law // Proc. SPIE. 1995. V. 2437. P. 2–17.
6. *Yan J., He J., Han W., Chen W., Zheng W.* How OpenMP Applications Get More Benefit from Many-Core Era // Proc. Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More: 6th Int. Workshop on OpenMP, IWOMP 2010. Tsukuba, Japan, June 14–16, 2010. C. 83–95.
7. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. 232 с.
8. *Cooley J.W., Tukey J.W.* An algorithm for the machine calculation of complex Fourier series // Mathem. Comput. 1965. V. 19, N 2. P. 297–301.
9. *Frigo M., Johnson S.G.* The design and implementation of FFTW3 // Proc. IEEE. 2005. V. 93, N 2. P. 216–231.
10. *Коняев П.А.* Модификация метода расщепления для численного решения квазиоптических задач // VI Всесоюз. симпоз. по распространению лазерного излучения в атмосфере. Ч. III. Томск: ИОА СО АН СССР, 1981. С. 195–198.

P.A. Konyaev, E.A. Tartakovskii, and G.A. Filimonov. **Computer simulation of optical waves propagation, using parallel programming technique.**

Methods and features of parallel algorithms for numerical simulation of optical waves propagation are considered. The scalar parabolic equation for a complex amplitude of monochromatic waves was solved numerically, using the Fourier transform method for homogeneous media and split-step Fourier method for inhomogeneous media. Two parallel algorithms, using OpenMP technology with MKL library for Intel multicore processors and CUDA technology for NVIDIA graphic accelerators have been created. The comparison of two approaches with each other and with a common sequential algorithm, using FFTW library, was performed by calculation of average number of test task solutions per second.

It is shown that parallel algorithms have a considerable advantage in speed (by tens times) to the common sequential algorithm in accordance with the grid size in computational task. When comparing the performance of the above two parallel techniques with each other the results were as follows: for grids up to1024×1024 the approach, using OpenMP technology, holds the lead, while for the large grids (from 1024×1024 and more) the approach, using CUDA technology, was faster.

The obtained results are discussed and recommendations about transition from sequential algorithms to the parallel ones are given.