УДК 004.051+519.612.2+532.5 doi 10.26089/NumMet.v16r456

ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ ПРИ МОДЕЛИРОВАНИИ ТЕЧЕНИЙ ВЯЗКОЙ НЕСЖИМАЕМОЙ СРЕДЫ МЕТОДОМ LS-STAG НА СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ

И. К. Марчевский[1], **В. В. Пузикова**[2]

Для моделирования течений вязкой несжимаемой среды методом погруженных границ LS-STAG и его модификациями разработан параллельный программный комплекс LS-STAG_ext. Комплекс позволяет моделировать обтекание движущихся профилей произвольной формы и систем из любого числа профилей, имеющих одну или две степени свободы. Комплекс LS-STAG_ext поддерживает использование таких технологий параллельного программирования, как Intel® Cilk™ Plus, Intel® Threading Building Blocks и OpenMP. Представлены результаты сравнения эффективности реализованных в программном комплексе LS-STAG_ext параллельных алгоритмов с аналогами из высокопроизводительной библиотеки Intel® Math Kernel Library. Описаны особенности реализации в разработанном программном комплексе алгоритма метода FGMRES для решения систем линейных алгебраических уравнений.

Ключевые слова: технология Intel® Cilk™ Plus, технология Intel® Threading Building Blocks, технология OpenMP, библиотека Intel® Math Kernel Library, разреженные линейные системы, метод FGMRES, метод BiCGStab, решатель PARDISO, вязкая несжимаемая среда, метод погруженных границ LS-STAG.

1. Введение. При решении сложных с вычислительной точки зрения задач, таких как сопряженные задачи гидроупругости, крайне важно обеспечить высокую точность расчета и эффективность программной реализации используемых численных методов. Важную роль играет тип используемой сетки: весьма эффективными оказываются сеточные методы, в которых сетка не связана с границей тела и не перестраивается при его движении. К таким методам относятся методы погруженных границ [1]: в них используются прямоугольные сетки, а при пересечении ячеек сетки с границей области течения образуются ячейки неправильной формы, называемые усеченными. Точность метода определяется прежде всего точностью решения задачи в этих ячейках.

Одним из наиболее эффективных методов погруженных границ считается метод LS-STAG [2]. Описание погруженной границы при помощи функций уровня [3] и однородность LS-STAG-дискретизации в прямоугольных и усеченных ячейках делают дополнительные затраты времени на обработку усеченных ячеек пренебрежимо малыми. К настоящему моменту построена LS-STAG-дискретизация двумерных уравнений Навье–Стокса для вязкой несжимаемой среды [2], уравнений RANS, LES и DES [4, 5], а также уравнений, входящих в модели турбулентности Спаларта–Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST [4]. Кроме того, разработана модификация метода для решения сопряженных задач гидроупругости [6]. Метод LS-STAG и его модификации реализованы в разработанном авторами настоящей статьи параллельном программном комплексе LS-STAG_ext. Комплекс позволяет проводить расчеты с использованием таких технологий параллельного программирования, как Intel® Cilk™ Plus [7], Intel® Threading Building Blocks [8] и OpenMP.

Целью настоящей работы является исследование эффективности распараллеливания вычислений в программном комплексе LS-STAG_ext с использованием различных технологий параллельного программирования. Для сравнения используются аналогичные алгоритмы из библиотеки высокооптимизированных математических алгоритмов Intel® Math Kernel Library (MKL) версии 11.2 [9].

2. Постановка тестовой задачи. В качестве тестовой задачи для исследования эффективности разрабатываемых параллельных алгоритмов рассмотрим двумерную задачу о моделировании обтекания

[1] Московский государственный технический университет им. Н. Э. Баумана, факультет фундаментальных наук, ул. 2-я Бауманская, д. 5, 105005, Москва; доцент, e-mail: iliamarchevsky@mail.ru

[2] Московский государственный технический университет им. Н. Э. Баумана, факультет фундаментальных наук, ул. 2-я Бауманская, д. 5, 105005, Москва; ассистент, e-mail: valeria.puzikova@gmail.com

жесткого кругового профиля диаметра D с границей K, совершающего в невозмущенном горизонтальном равномерном потоке вязкой несжимаемой среды постоянной плотности поперечные колебания по закону

$$\begin{cases} X_C = X_C^0, \\ Y_C = Y_C^0 + \begin{cases} A, & t < 10, \\ A\cos\bigl(2\pi S_e[t-10]\bigr), & t \geqslant 10, \end{cases} \end{cases}$$

где A — амплитуда колебаний профиля, S_e — кинематическое число Струхаля, t — безразмерное время, $\bigl(X_C^0, Y_C^0\bigr)$ и (X_C, Y_C) — координаты центра профиля в среднем положении и в текущий момент времени соответственно. Задача решается в расчетной области Ω с внешней границей $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$ (Γ_1 — входная граница, Γ_4 — выходная). Движение среды описывается уравнениями Навье–Стокса

$$\begin{cases} \nabla \cdot \boldsymbol{v} = 0, \quad \dfrac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{v} + \nabla p - \dfrac{1}{\mathrm{Re}}\,\Delta \boldsymbol{v} = 0, \\ \boldsymbol{v}(\boldsymbol{r},\ 0) = \boldsymbol{v}_0(\boldsymbol{r}), \\ \boldsymbol{v}\big|_{\Gamma_1\cup\Gamma_2\cup\Gamma_3} = \boldsymbol{v}_\infty, \quad \boldsymbol{v}\big|_K = \boldsymbol{v}^{ib} = \boldsymbol{v}^{ib}(\boldsymbol{r}, t), \quad \left.\dfrac{\partial \boldsymbol{v}}{\partial \boldsymbol{n}}\right|_{\Gamma_4} = \vec{0}, \quad \left.\dfrac{\partial p}{\partial \boldsymbol{n}}\right|_{\Gamma\cup K} = 0, \end{cases}$$

где x и y — безразмерные координаты; $\boldsymbol{r} = x\cdot\boldsymbol{e}_x + y\cdot\boldsymbol{e}_y$ — радиус-вектор точки; $p = p(\boldsymbol{r}, t)$ — безразмерное давление; $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{r}, t) = u\cdot\boldsymbol{e}_x + v\cdot\boldsymbol{e}_y$ — безразмерная скорость (в качестве характерной скорости выбрана скорость V_∞ набегающего потока); Re — число Рейнольдса, вычисленное по диаметру профиля. В тестовых расчетах будем моделировать обтекание профиля в течение 30 единиц безразмерного времени на сетке 240×296 (линейный размер ячейки сетки вблизи границы профиля $h = 0{,}03125$) с шагом по времени $\Delta t = 0{,}005$ при $V_\infty = 1{,}0$, $D = 1{,}0$, $\mathrm{Re} = 185$, $A = 0{,}2D$, $S_e/\mathrm{Sh} = 1{,}2$, где $\mathrm{Sh} \approx 0{,}201$ — число Струхаля, вычисленное при $\mathrm{Re} = 185$ для неподвижного профиля. Далее эту задачу будем обозначать VerOscTest.

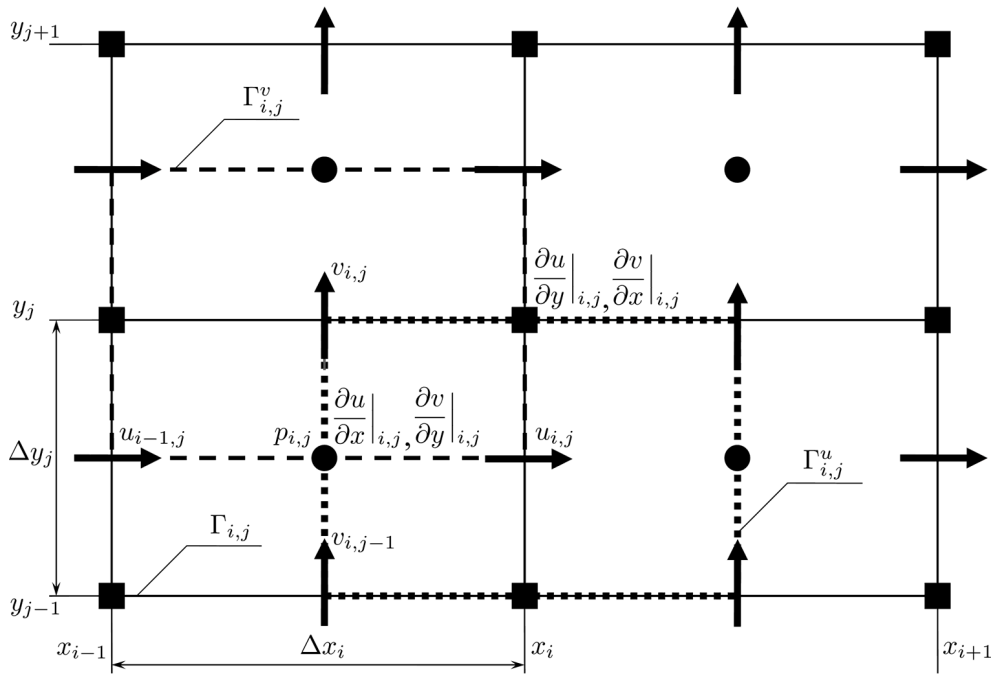

Рис. 1. Разнесенные сетки

Вычислительные эксперименты, представленные в данной работе, проводились на рабочей станции, построенной на платформе Intel H81 с использованием двухъядерного процессора Intel Core i3-4350T (Haswell) с поддержкой HyperThreading (4 логических ядра), работающего на частоте 3100 МГц. Станция оснащена 8 ГБайт оперативной памяти DDR3-1333, SSD-накопителем Crucial объемом 128 ГБайт и жестким диском Seagate объемом 1 ТБайт. Далее эту рабочую станцию будем обозначать PC1. Для исследования масштабируемости алгоритмов также применялась рабочая станция, построенная на платформе Intel Z97 с использованием 4-ядерного процессора Intel Core i7-4790K (Devil's Canyon) с поддержкой

технологии HyperThreading (8 логических ядер), работающего на частоте 4400 МГц. Станция оснащена 16 ГБайт оперативной памяти DDR3-1600 и двумя SSD-накопителями Crucial объемом 256 ГБайт и 1 ТБайт. Данную рабочую станцию будем обозначать PC2. Внешние графические карты не использовались.

3. Основные идеи метода LS-STAG. В расчетной области вводится прямоугольная сетка с ячейками $\Omega_{i,j} = (x_{i-1}, x_i) \times (y_{j-1}, y_j)$, радиусы-векторы центров которых будем обозначать $\boldsymbol{x}^c_{i,j} = \left(x^c_i, y^c_j\right)$, а границы — $\Gamma_{i,j}$ (рис. 1). Эти ячейки являются контрольными объемами, которые используются для дискретизации уравнения неразрывности.

Строятся смещенные сетки с ячейками $\Omega^u_{i,j} = (x^c_i, x^c_{i+1}) \times (y_{j-1}, y_j)$ и $\Omega^v_{i,j} = (x_{i-1}, x_i) \times (y^c_j, y^c_{j+1})$, границы которых обозначим $\Gamma^u_{i,j}$ и $\Gamma^v_{i,j}$ соответственно (рис. 1). Эти ячейки являются контрольными объемами для уравнения баланса импульса в проекциях на оси Ox и Oy.

Для описания положения границы Γ^{ib} твердого тела произвольной формы Ω^{ib} вводят знакопеременную функцию расстояния $\varphi(\boldsymbol{r})$ (функцию уровня [3]), такую, что

$$\begin{cases} \varphi(\boldsymbol{r}) < 0, & \boldsymbol{r} \in \Omega^f = \Omega \setminus \left\{\Omega^{ib} \cup \Gamma^{ib}\right\}, \\ \varphi(\boldsymbol{r}) = 0, & \boldsymbol{r} \in \Gamma^{ib}, \\ \varphi(\boldsymbol{r}) > 0, & \boldsymbol{r} \in \Omega^{ib}. \end{cases}$$

В простейших случаях функция уровня может быть задана аналитически: так, для кругового профиля диаметром D с центром в точке (X_C, Y_C) имеем $\varphi(x, y) = 0{,}5D - \sqrt{(x - X_C)^2 + (y - Y_C)^2}$.

Для построения функции уровня в случае профиля более сложной формы можно использовать алгоритм, предложенный в [10]. В каждой усеченной ячейке $\Omega_{i,j}$ LS-STAG-сетки (рис. 2) погруженная граница $\Gamma^{ib}_{i,j}$ представляется отрезком прямой, положения концов которого определяются линейной интерполяцией величины $\varphi_{i,j}$, принимающей значение функции уровня $\varphi(x_i, y_j)$ в правом верхнем углу ячейки $\Omega_{i,j}$. Для определения типа усеченной ячейки вводят коэффициенты заполнения ячеек $\vartheta^u_{i,j}$, $\vartheta^v_{i,j} \in [0, 1]$, показывающие, какая часть ячейки занята жидкостью на восточной и северной гранях ячейки соответственно [2]. Значения скоростей вычисляются в серединах жидких частей граней, а давление аппроксимируется кусочно-постоянной функцией на каждой ячейке.

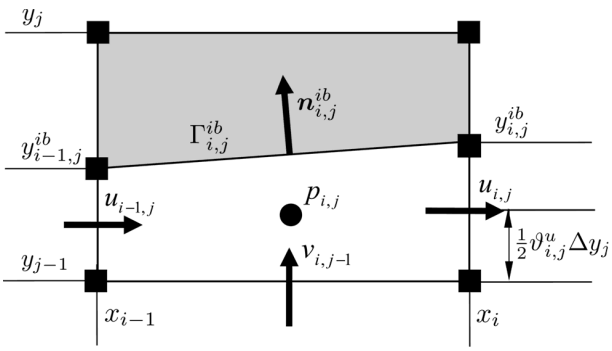

Рис. 2. Точки вычисления неизвестных на усеченной ячейке $\Omega_{i,j}$

На каждом шаге по времени при использовании метода LS-STAG выполняются следующие действия. В случае подвижных погруженных границ решаются разностные аналоги уравнений движения обтекаемого профиля или системы профилей и пересчитывается функция уровня, а также зависящие от нее характеристики сетки, матрицы предобусловливателей и разностных аналогов дифференциальных операторов. Затем происходит пересчет правых частей систем линейных алгебраических уравнений. После этого методом BiCGStab (методом бисопряженных градиентов со стабилизацией [11]) с ILU-предобусловливанием [12] решаются два разностных аналога уравнения Гельмгольца для прогнозов скоростей. Затем с учетом полученных прогнозов скоростей методом BiCGStab с многосеточным предобусловливанием [13] решается разностный аналог уравнения Пуассона для поправки давления. Значения прогнозов и поправки корректируются, в результате чего получаются значения скоростей и давления на текущем шаге по времени. После этого в случае использования моделей турбулентности решаются разностные аналоги уравнений из модели турбулентности и рассчитываются рейнольдсовы или подсеточные напряжения, которые необходимы для пересчета правых частей систем для прогнозов скоростей на следующем шаге по времени.

4. Параллельный программный комплекс LS-STAG_ext. Разработанный программный комплекс LS-STAG_ext позволяет с помощью метода LS-STAG и его модификаций решать различные задачи вычислительной гидродинамики, в том числе и сопряженные задачи гидроупругости (рис. 3).

В программном комплексе распараллелены следующие операции: умножение разреженной матрицы на вектор, решение систем линейных алгебраических уравнений с трехдиагональными матрицами методом прогонки при выполнении сглаживания в многосеточном предобусловливателе, операции с векторами, на выполнение которых приходится бóльшая часть затрат времени при решении систем линейных алгебраических уравнений методом BiCGStab (без учета затрат на работу предобусловливателей). Таким образом,

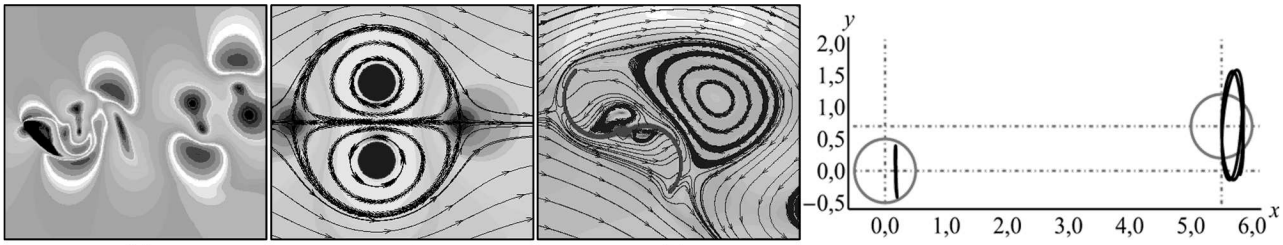

Рис. 3. Примеры результатов моделирования, полученные при помощи параллельного программного комплекса LS-STAG_ext: а) обтекание эллиптического профиля; б) стабилизация следа за расположенными рядом поперек потока круговыми профилями, вращающимися в противоположных направлениях; в) авторотация ротора Савониуса; г) резонанс системы двух круговых профилей с двумя степенями свободы (изображены траектории движения центров профилей)

бóльшая часть распараллеленных алгоритмов связана с работой решателя. При решении тестовой задачи VerOscTest в последовательном режиме на выполнение этих операций до оптимизации кода приходилось около 95% времени работы программы.

Для распараллеливания вычислений в программном комплексе LS-STAG_ext используются такие технологии параллельного программирования, как Intel® Cilk™ Plus, OpenMP (реализация из Intel® Parallel Studio XE 2015, стандарт 4.0), Intel® Threading Building Blocks (TBB). Эти технологии предполагают, что пользователю достаточно при помощи ключевых слов лишь обозначить задачи, выполняемые параллельно, а организация управления потоками и работы с ними определяются самой технологией. Таким образом, накладные расходы на поддержку многопоточности (расходы на создание задач и потоков, работу планировщика потоков, запуск и синхронизацию потоков и т.д.), а значит, и достигаемое ускорение, могут зависеть от используемой технологии параллельного программирования. По этой причине LS-STAG_ext поддерживает все перечисленные технологии параллельного программирования, а переключение между ними осуществляется при помощи директив препроцессора по определениям `LS_STAG_USE_CILK`, `LS_STAG_USE_OMP` и `LS_STAG_USE_TBB`.

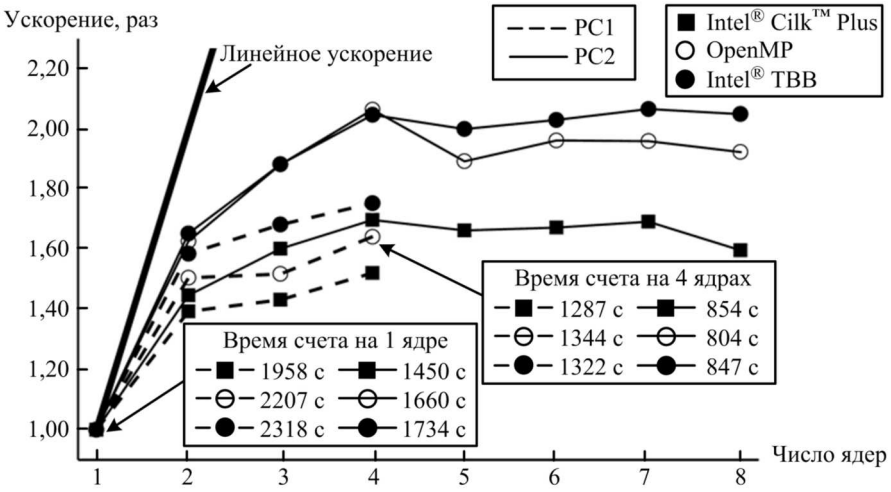

Рис. 4. Масштабируемость разрабатываемого программного комплекса LS-STAG_ext на тестовой задаче VerOscTest

Вычислительные эксперименты показали, что наименьшее время проведения расчета независимо от технологии параллельного программирования и на PC1, и на PC2 получается при использовании четырех ядер (рис. 4). Поскольку диспетчеры потоков Intel® Cilk™ Plus, OpenMP и Intel® TBB не отключаются при работе приложения на одном ядре, особенности их реализации напрямую сказываются на быстродействии даже в однопоточной версии: из рис. 4 можно сделать вывод о том, что диспетчер потоков Intel® Cilk™ Plus реализован эффективнее планировщиков потоков OpenMP и Intel® TBB. Однако приложение с Intel® Cilk™ Plus на обеих рабочих станциях масштабируется хуже приложения, в котором

используется OpenMP, а оно, в свою очередь, — хуже приложения, использующего Intel® TBB. Из-за этого при проведении расчета с использованием четырех логических ядер на PC1 приложение с Intel® TBB решает задачу быстрее приложения с OpenMP (при этом наименьшее время счета получается по-прежнему при использовании Intel® Cilk™ Plus), а на PC2 опережает приложение, использующее Intel® Cilk™ Plus.

5. Сравнение эффективности разработанных алгоритмов с аналогами из Intel® MKL. Сравним эффективность алгоритмов, реализованных в разработанном параллельном программном комплексе LS-STAG_ext, с аналогами из библиотеки Intel® Math Kernel Library (MKL) версии 11.2. Данная библиотека оптимизирована для работы с процессорами Intel и обеспечивает использование их расширенных возможностей. Многопоточная версия библиотеки Intel® MKL использует технологию OpenMP. Количество ядер, выделяемых для работы библиотеки, задается при помощи функции `mkl_set_num_threads`.

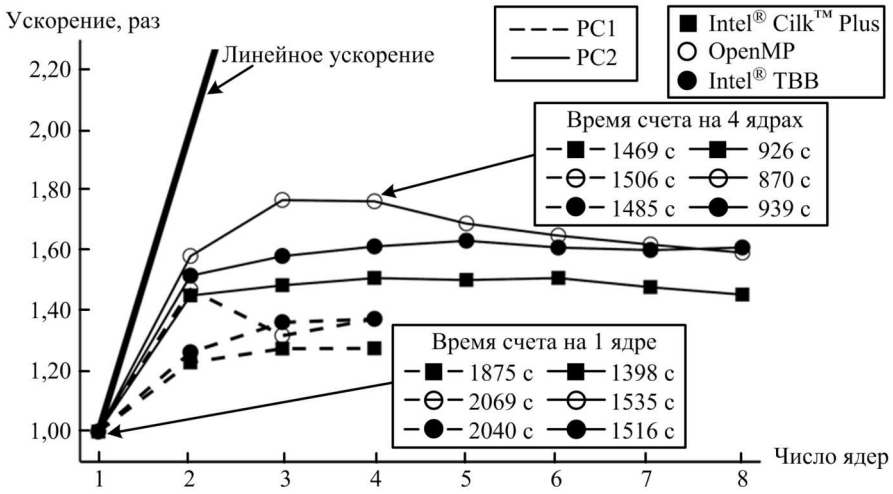

Рис. 5. Масштабируемость разрабатываемого программного комплекса при использовании матрично-векторного умножения из Intel® MKL

Используем функцию `mkl_cspblas_dcsrgemv` из Intel® MKL вместо разработанной реализации умножения разреженной матрицы, хранящейся в формате CSR [12] в массивах `Cell`, `Num` и `Col`, на вектор, хранящийся в массиве `multiplier`, с сохранением результата в массив `Vector` (рис. 5):

```
char n = 'N'; mkl_cspblas_dcsrgemv(&n, &size, Cell, Num, Col, multiplier, Vector);
```

Видно, что использование `mkl_cspblas_dcsrgemv` позволяет получить незначительное ускорение на одном ядре (в среднем на 8–9%), но полученный алгоритм хуже масштабируется, и на четырех ядрах время счета оказывается больше, чем при использовании исходного алгоритма.

При выполнении ILU-предобусловливания происходит решение двух систем линейных алгебраических уравнений с треугольными разреженными матрицами, тоже хранящимися в формате CSR, при помощи обратного хода метода Гаусса. Вместо реализации из LS-STAG_ext можно использовать функцию `mkl_cspblas_dcsrtrsv`:

```
char l = 'L', n = 'N', u = 'U'; mkl_cspblas_dcsrtrsv(&l, &n, &u, &s, LC, LN, LP, b, y);
                                mkl_cspblas_dcsrtrsv(&u, &n, &n, &s, UC, UN, UP, y, x);
```

Первый вызов функции соответствует решению системы с нижнетреугольной матрицей (`LC`, `LN`, `LP`) размера $s \times s$, а второй — решению системы с верхнетреугольной матрицей. Результаты вычислительных экспериментов представлены на рис. 6: видно, что использование Intel® MKL увеличивает время счета как на одном, так и на четырех ядрах.

Реализованные векторные операции можно заменить на такие функции Intel® MKL, как `vdAdd` (покомпонентное сложение векторов), `vdSub` (покомпонентное вычитание векторов), `vdMul` (покомпонентное умножение векторов), `vdSqr` (покомпонентное возведение в квадрат), `cblas_dscal` (покомпонентное умножение на скаляр), `cblas_dnrm2` (расчет евклидовой нормы), `cblas_dasum` (вычисление суммы элементов), `cblas_ddot` (вычисление скалярного произведения). Однако, как и в случае матрично-векторного умножения, такая замена приводит лишь к незначительному ускорению на одном ядре (в среднем на 3%)

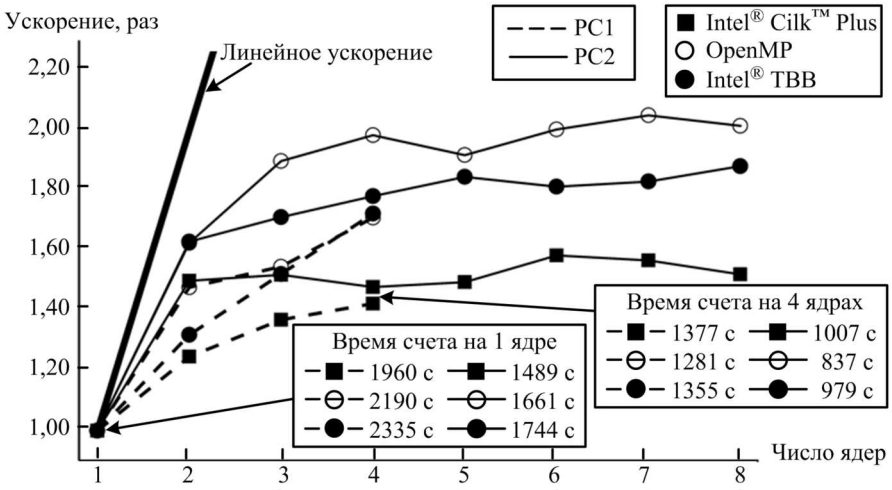

Рис. 6. Масштабируемость программного комплекса при использовании функции `mkl_cspblas_dcsrtrsv` из Intel® MKL

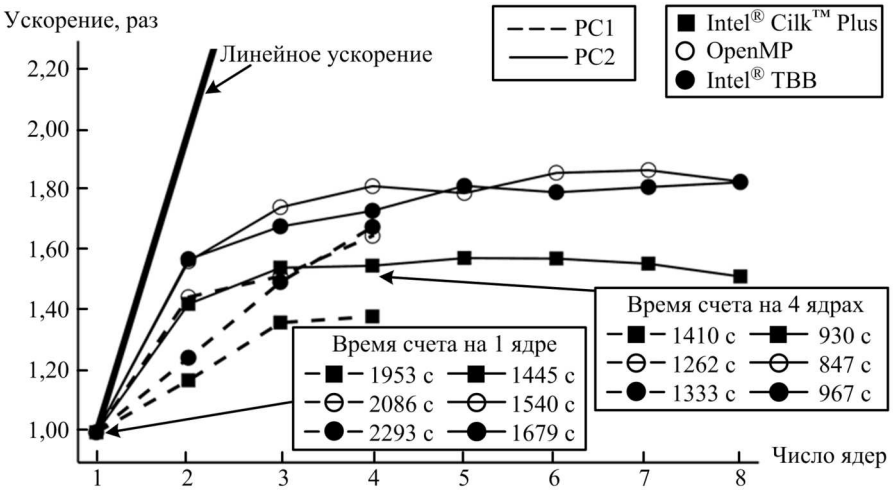

Рис. 7. Масштабируемость разрабатываемого программного комплекса при использовании операций над векторами из Intel® MKL

и ухудшению масштабируемости, из-за чего при расчете на четырех ядрах время счета оказывается примерно на 8% больше, чем при использовании исходного алгоритма (рис. 7).

Рассмотрим возможность замены всего решателя аналогом из Intel® MKL. Библиотека содержит прямой решатель PARDISO [14], позиционируемый как высокопроизводительный параллельный прямой решатель для систем с разреженными матрицами, который на системах с числом уравнений менее ста тысяч оказывается эффективнее итерационных решателей [9]. Поскольку возникающие при решении тестовой задачи VerOscTest системы линейных алгебраических уравнений содержат по 71040 уравнений, были проведены вычислительные эксперименты с использованием решателя PARDISO. Из-за индексации в C++ массивов с нуля параметры решателя необходимо задавать вручную, а затем поочередно вызывать четыре фазы решателя (анализ, построение разложения, решение и освобождение памяти):

```
void *t[64]; int ip[64]; int e = 0; int type = 11; int fct = 1; int lvl = 0;
for(int i = 0; i < 64; i++){ t[i] = 0; ip[i] = 0; }
int *f = &fct, *v = &lvl; int* p; p = new int[s]; ip[0] = 1;// не по умолчанию
ip[1] = 2; ip[9] = 13; ip[10] = 1; ip[12] = 1; ip[17] = -1; ip[23] = 1;// параллельный
int phase; ip[18] = -1; ip[20] = 1; ip[34] = 1;// индексация в массивах с 0
phase = 11; pardiso(t, f, f, &type, &phase, &s, Cell, Num, Col, p, f, ip, v, b, x, &e);
phase = 22; pardiso(t, f, f, &type, &phase, &s, Cell, Num, Col, p, f, ip, v, b, x, &e);
phase = 33; pardiso(t, f, f, &type, &phase, &s, Cell, Num, Col, p, f, ip, v, b, x, &e);
```

```
phase = -1; pardiso(t, f, f, &type, &phase, &s, Cell, Num, Col, p, f, ip, v, b, x, &e);
```

Использование PARDISO привело в данном случае к значительному увеличению продолжительности расчета (рис. 8): на одном ядре замедление составило в среднем 4,444 раза, а на четырех ядрах — 5,747 раза.

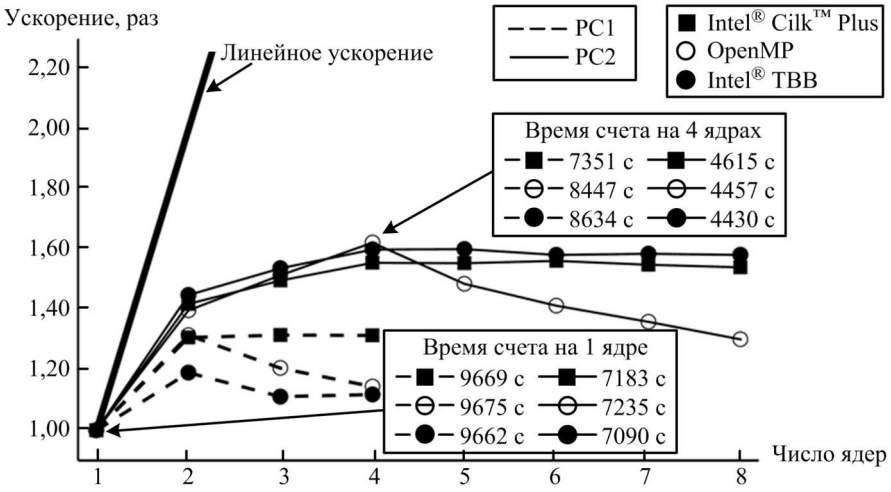

Рис. 8. Масштабируемость разрабатываемого программного комплекса при использовании решателя PARDISO из Intel® MKL для решения разностных аналогов уравнений Гельмгольца и Пуассона

Из предобусловливателей библиотека Intel® MKL содержит только ILU-предобусловливатель, но из-за индексации массивов с единицы в его реализации эту функцию можно использовать лишь в программах, написанных на языке Фортран. Используемый в текущей версии LS-STAG_ext метод BiCGStab в Intel® MKL не реализован: из итерационных методов доступны только CG (метод сопряженных градиентов) и FGMRES (гибкий метод обобщенных минимальных невязок). Метод CG не подходит для решения поставленной задачи, поскольку матрицы возникающих систем в общем случае являются несимметричными. Использование метода FGMRES без предобусловливания тоже не позволяет решить задачу: метод не сходится за 200 итераций при решении разностного аналога уравнения Пуассона, и примерно через четыре единицы модельного времени расчет “разваливается”. Воспользуемся тем фактом, что работа решателя FGMRES в Intel® MKL построена так, что становится возможным использовать предобусловливатель, реализованный вне библиотеки:

```
int it = 0, req, ip[128]; double dp[128], *t;
t = new double[1+(2*maxIt+1)*s+maxIt*(maxIt+9)/2]; dfgmres_init(&s, x, b, &req, ip, dp, t);
ip[4] = maxIt; ip[14] = maxIt; ip[10] = 1;// используется предобусловливатель
dfgmres_check(&s, x, b, &req, ip, dp, t);
while(true)
{ bool exit = false;
  dfgmres(&s, x, b, &req, ip, dp, t);
  switch(req)
  { case 0: exit = true; break; // решение получено
    case 1: w.setProduct(&t[ip[21]-1],&t[ip[22]-1], s); break; // умножение на матрицу
    case 2: if(dp[4] < eps) exit = true; break; // норма невязки мала
    case 3: preconditioner(&t[ip[21]-1], &t[ip[22]-1]); break; // предобусловливание
    case 4: if(dp[6] < PLUS_TINY) exit = true;  break; // норма обобщенного вектора
  } if(exit) break;
}
norm_r = dp[4]; dfgmres_get(&s, x, b, &req, ip, dp, t, &it); delete[] t;
```

При использовании метода FGMRES из Intel® MKL с реализованным в параллельном программном комплексе LS-STAG_ext ILU-предобусловливанием для решения как разностного аналога уравнения

Гельмгольца, так и разностного аналога уравнения Пуассона получаем ускорение по сравнению с исходным решателем в среднем на 20% при расчете на одном ядре. Однако масштабируемость полученного алгоритма оказывается очень низкой, поэтому при использовании четырех ядер он уступает исходному решателю (рис. 9). При этом решение разностного аналога уравнения Гельмгольца при использовании обоих методов получается за 2 итерации, тогда как решение разностного аналога уравнения Пуассона при использовании метода BiCGStab с многосеточным предобусловливанием получается за 7–20 итераций, а при решении методом FGMRES с ILU-предобусловливанием — за 25–140 итераций. Несмотря на такую разницу в числе итераций, метод FGMRES позволил существенно уменьшить время счета на одном ядре, что говорит о высокой эффективности метода.

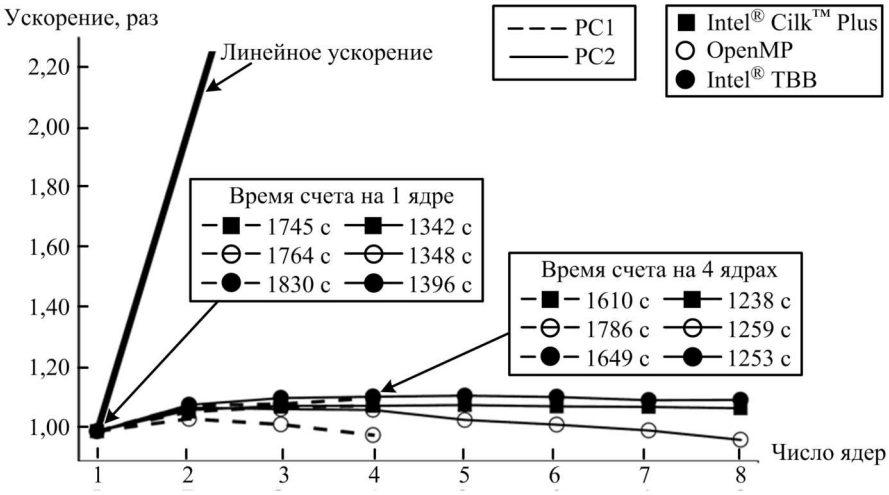

Рис. 9. Масштабируемость разрабатываемого программного комплекса при использовании метода FGMRES из Intel® MKL с ILU-предобусловливанием для решения разностных аналогов уравнений Гельмгольца и Пуассона

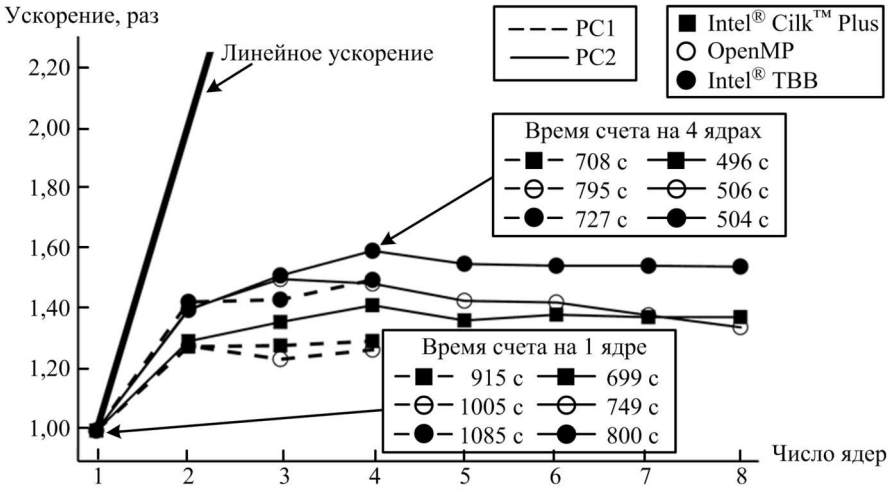

Рис. 10. Масштабируемость разрабатываемого программного комплекса при использовании метода FGMRES из Intel® MKL с ILU-предобусловливанием для решения разностного аналога уравнения Гельмгольца и с многосеточным предобусловливанием для решения разностного аналога уравнения Пуассона

Если при решении разностного аналога уравнения Пуассона методом FGMRES из Intel® MKL использовать многосеточное предобусловливание, реализованное в LS-STAG_ext, получаем ускорение по сравнению с решателем на базе метода BiCGStab на одном ядре в среднем в 2,155 раза, а на четырех ядрах — в среднем в 1,757 раза из-за худшей масштабируемости (рис. 10). При этом разностный аналог уравнения Пуассона решается за 4–7 итераций, что говорит о более быстрой сходимости метода FGMRES по сравнению с методом BiCGStab. Таким образом, представляется целесообразным разработать собственную реализацию метода FGMRES и сравнить ее эффективность с реализацией из Intel® MKL.

6. Реализация алгоритма метода FGMRES. Метод FGMRES, как и метод BiCGStab, относится к методам крыловского типа. Основное различие между ними заключается в способе построения базиса в подпространстве Крылова: в методе BiCGStab для этого используется биортогонализация Ланцоша, а в методе FGMRES — ортогонализация Арнольди [15]. Для системы линейных алгебраических уравнений

$$Ax = b, \quad x, b \in \mathbb{R}^n, \quad A \in M(\mathbb{R})_{n\times n},$$

которую требуется решить с точностью ε, алгоритм метода FGMRES с рестартами через каждые m итераций имеет следующий вид [15]:

$$\begin{aligned}
&\text{Start. } r^0 = b - Ax^0,\ \beta = \|r^0\|_2,\ v^1 = r^0/\beta;\\
&\quad \text{for } (j = 1, \ldots, m)\\
&\qquad z^j = M_j^{-1} v^j;\ w^j = Az^j;\\
&\qquad \text{for } (i = 1, \ldots, j)\Big\{\ h_{i,j} = \big(w^j, v^i\big);\ w^j = w^j - h_{i,j}v^i;\ \Big\}\\
&\qquad h_{j+1,j} = \big\|w^j\big\|_2;\ v^{j+1} = w^j/h_{j+1,j};\\
&\quad Z_m = \big[z^1 \ldots z^m\big];\ \overline{H}_m = \{h_{i,j}\};\ y^m = \operatorname*{argmin}_y \big\|\beta e^1 - \overline{H}_m y\big\|_2;\\
&\quad x^m = x^0 + Z_m y^m;\ \text{if}\Big(\big\|b - Ax^m\big\|_2 > \varepsilon\Big)\big\{\ x^0 = x^m;\ \text{goto Start.}\ \big\}
\end{aligned} \tag{1}$$

Здесь $x^0 \in \mathbb{R}^n$ — начальное приближение; $r^0 \in \mathbb{R}^n$ — начальная невязка; $M_j \in M(\mathbb{R})_{n\times n}$ — матрица предобусловливателя на j-й итерации; $y^m \in \mathbb{R}^m$; $e^1 = (1\,0\ldots0)^{\mathrm{T}} \in \mathbb{R}^{m+1}$; $Z_m \in M(\mathbb{R})_{n\times m}$ — матрица, столбцами которой являются векторы z^j; $\overline{H}_m \in M(\mathbb{R})_{(m+1)\times m}$ — матрица коэффициентов ортогонализации $H_m \in M(\mathbb{R})_{m\times m}$ (верхняя матрица Хессенберга), дополненная строкой $(0\ldots0\,h_{m+1,m})$; $w^j, z^j \in \mathbb{R}^n$ при $j = \overline{1,m}$; $v^j \in \mathbb{R}^n$ при $j = \overline{1,m+1}$; (u,p) — скалярное произведение векторов u и p; $\|u\|_2 = \sqrt{(u,u)}$ — евклидова норма вектора u. Отметим, что векторы $v^1, \ldots, v^m$ образуют ортогональный базис в подпространстве Крылова размерности m, порожденном вектором v^1 и матрицей A, т.е. в линейном пространстве

$$K_m = K_m(A, v^1) = \operatorname{span}\big\{v^1, Av^1, A^2v^1, \ldots, A^{m-1}v^1\big\}.$$

Алгоритм (1) требует решения линейной задачи наименьших квадратов $y^m = \operatorname*{argmin}_y \big\|\beta e^1 - \overline{H}_m y\big\|_2$, т.е. решения системы

$$\overline{H}_m y = \beta e^1. \tag{2}$$

Для этого используем QR-разложение, построенное при помощи метода вращений [15]: умножим (2) слева на матрицу

$$Q_m = \Omega_m \cdot \ldots \cdot \Omega_1, \quad Q_m, \Omega_i \in M(\mathbb{R})_{(m+1)\times(m+1)}, \quad i = \overline{1,m},$$

где Ω_i — матрица вращений Гивенса: i-й и $(i+1)$-й диагональные элементы этой матрицы равны $c_i = h_{i+1,i}/\Delta_i$, $\Delta_i = \sqrt{h_{i,i}^2 + h_{i+1,i}^2}$, остальные диагональные элементы — единицы; $(i+1)$-й элемент i-й строки равен $s_i = h_{i,i}/\Delta_i$, i-й элемент $(i+1)$-й строки равен $(-s_i)$, остальные элементы — нули. В итоге получаем систему

$$\overline{R}_m y = \overline{g^m}, \tag{3}$$

где $\overline{R}_m = Q_m\overline{H}_m \in M(\mathbb{R})_{(m+1)\times m}$, $\overline{g^m} = Q_m(\beta e^1) = (\gamma_1 \ldots \gamma_{m+1})^{\mathrm{T}} \in \mathbb{R}^{m+1}$. После удаления из (3) последнего уравнения получаем

$$R_m y^m = g^m, \tag{4}$$

где $R_m \in M(\mathbb{R})_{m\times m}$ — верхнетреугольная матрица, что позволяет решить эту систему при помощи обратного хода метода Гаусса.

Полученное решение y^m системы (4) тоже является решением задачи (2) и позволяет вычислить итерационное приближение x^m из (1). При этом для нормы вектора невязки из (1) справедливо равенство $\|b - Ax^m\|_2 = |\gamma_{m+1}|$ [15]. Благодаря этому свойству критерий останова можно проверять на каждой итерации без решения системы (4) и, если он выполняется, вычислять решение, не дожидаясь m-й итерации. При этом удобно хранить матрицы Z_m и H_m по столбцам (packed storage). Для того чтобы пересчитывать на j-й итерации элементы j-го столбца матрицы H_m и элементы g_j и $g_{j+1} = \gamma$ правой части системы (4),

необходимо хранить векторы косинусов и синусов $c, s \in \mathbb{R}^m$. Тогда алгоритм (1) можно переписать следующим образом:

$$
\begin{aligned}
&\text{Start. } r^0 = b - Ax^0,\ \beta = \left\|r^0\right\|_2,\ w^0 = r^0,\ \gamma = \beta,\ k = m;\\
&\quad \text{for } (j = 1, \ldots, m)\\
&\qquad v^j = w^{j-1}/\beta;\ z^j = M_j^{-1} v^j;\ w^j = Az^j;\\
&\qquad \text{for } (i = 1, \ldots, j)\Big\{\ h_{i,j} = \left(w^j, v^i\right);\ w^j = w^j - h_{i,j} v^i;\ \Big\}\\
&\qquad \text{for } (i = 1, \ldots, j-1)\ \Big\{\ \widetilde{h} = h_{i,j};\ h = h_{i+1,j};\ h_{i,j} = c_i\widetilde{h} + s_i h;\ h_{i+1,j} = -s_i\widetilde{h} + c_i h;\ \Big\}\\
&\qquad \widetilde{h} = h_{j,j};\ \beta = \|w^j\|_2;\ h_{j,j} = \sqrt{\widetilde{h}^2 + \beta^2};\ c_j = \beta/h_{j,j};\ s_j = \widetilde{h}/h_{j,j};\\
&\qquad g_j = \gamma c_j;\ \gamma = -s_j\gamma;\ \text{if } (|\gamma| < \varepsilon)\{\ k = j;\ j = m+1;\ \}\\
&Z_k = \left[z^1 \ldots z^k\right];\ H_k = \{h_{i,j}\};\ x^m = x^0 + Z_k H_k^{-1} g^k;\ \text{if } \left(|\gamma| > \varepsilon\right)\left\{x^0 = x^m;\ \text{goto Start.}\right\}
\end{aligned}
\tag{5}
$$

Алгоритм (5) реализован в программном комплексе LS-STAG_ext для решения разностных аналогов уравнений Гельмгольца и Пуассона с ILU- и многосеточным предобусловливанием соответственно. Разработанная реализация метода FGMRES позволила получить ускорение по сравнению с аналогичным решателем из Intel® MKL на одном ядре в среднем на 4%, а на четырех ядрах — на 6%, поскольку разработанный алгоритм лучше масштабируется (рис. 11). Наилучшее быстродействие при расчетах на одном ядре демонстрируют приложения, использующие технологию Intel® Cilk™ Plus, однако при расчете на PC2 с четырьмя ядрами благодаря хорошей масштабируемости наименьшая продолжительность расчета получается при использовании Intel® TBB.

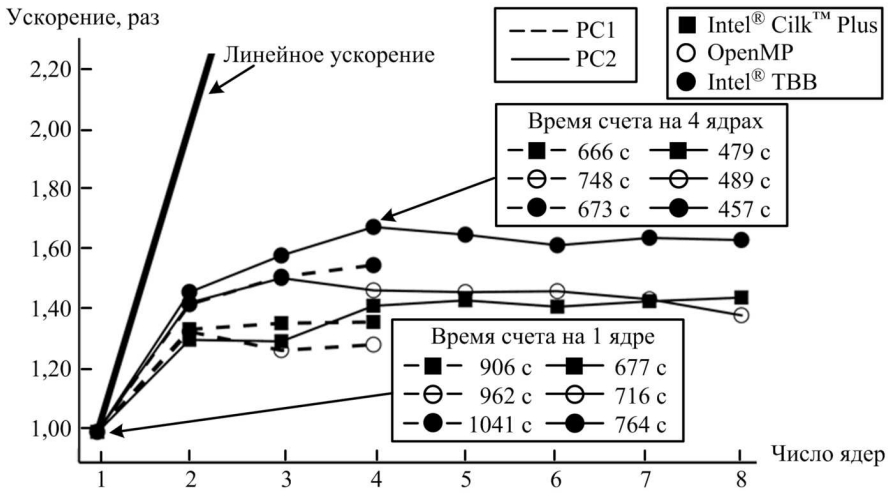

Рис. 11. Масштабируемость программного комплекса LS-STAG_ext при решении тестовой задачи VerOscTest с использованием разработанной реализации метода FGMRES

7. Заключение. Исследована эффективность параллельных алгоритмов, реализованных в разработанном программном комплексе LS-STAG_ext для моделирования движения профилей в потоке вязкой несжимаемой среды методом погруженных границ LS-STAG и его модификациями для решения сопряженных задач гидроупругости и моделирования турбулентности. Для сравнения эффективности использовались аналогичные алгоритмы, реализованные в библиотеке высокооптимизированных математических алгоритмов Intel® Math Kernel Library. По итогам сравнения была обнаружена необходимость замены метода BiCGStab, используемого в решателе комплекса LS-STAG_ext для решения систем линейных алгебраических уравнений, на метод FGMRES, показавший более быструю сходимость и обладающий менее затратными с вычислительной точки зрения итерациями. Оптимизированный алгоритм метода FGMRES реализован в параллельном программном комплексе LS-STAG_ext. Разработанный алгоритм демонстрирует более высокую эффективность по сравнению с использованием аналогичного решателя из библиотеки Intel® MKL как при расчетах на одном ядре, так и при использовании нескольких ядер.

Работа выполнена при финансовой поддержке гранта Президента РФ (код проекта МК–5357.2015.8).

СПИСОК ЛИТЕРАТУРЫ

1. *Mittal R., Iaccarino G.* Immersed boundary methods // Annu. Rev. Fluid Mech. 2005. **37**. 239–261.
2. *Cheny Y., Botella O.* The LS-STAG method: a new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties // J. Comput. Phys. 2010. **229**, N 4. 1043–1076.
3. *Osher S., Fedkiw R.* Level set methods and dynamic implicit surfaces. N.Y.: Springer, 2003.
4. *Puzikova V.V., Marchevsky I.K.* Extension of the LS-STAG immersed boundary method for RANS-based turbulence models and its application for numerical simulation in coupled hydroelastic problems // Proc. VI International Conference on Coupled Problems in Science and Engineering. Barcelona: Int. Center for Numer. Methods in Engineering, 2015. 532–543.
5. *Puzikova V.V.* On generalization of the LS-STAG immersed boundary method for large eddy simulation and detached eddy simulation // Advanced Problems in Mechanics International Summer School-Conference. St.-Petersburg: Inst. for Problems in Mechanical Engineering, 2015. 411–417.
6. *Puzikova V.V., Marchevsky I.K.* Application of the LS-STAG immersed boundary method for numerical simulation in coupled aeroelastic problems // Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dynamics. Barcelona: Int. Center for Numer. Methods in Engineering, 2014. 1995–2006.
7. Intel(R) Cilk(TM) Plus [Electronic resource] // https://software.intel.com/ru-ru/node/522579.
8. *Reinders J.* Intel threading building blocks: outfitting C++ for Multi-Core Processor Parallelism. Sebastopol: O'Reilly, 2007.
9. Intel® Math Kernel Library — Documentation [Electronic resource] // https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation.
10. *Пузикова В.В.* Построение функции уровня для профиля произвольной формы при моделировании его обтекания методом LS-STAG // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2012. Спец. выпуск № 2 "Математическое моделирование в технике". 163–173.
11. *Van der Vorst H.A.* Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for solution of nonsymmetric linear systems // SIAM J. Sci. Stat. Comp. 1992. **13**, N 2. 631–644.
12. *Баландин М.Ю., Шурина Э.П.* Методы решения СЛАУ большой размерности. Новосибирск: Изд-во НГТУ, 2000.
13. *Wesseling P.* An introduction to multigrid methods. Chichester: Willey, 1991.
14. *Schenk O., Gärtner K.* Solving unsymmetric sparse systems of linear equations with PARDISO // J. of Future Generation Computer Systems. 2004. **20**, N 3. 475–487.
15. *Saad Y.* Iterative Methods for Sparse Linear Systems. N.Y.: PWS Publ., 1996.

Поступила в редакцию
12.11.2015

Efficiency Study of Computation Parallelization for Viscous Incompressible Flow Simulation on Computing Systems with Shared Memory by the LS-STAG Method

I. K. Marchevsky [1] **and V.V. Puzikova** [2]

[1] *Bauman Moscow State Technical University, Faculty of Fundamental Sciences; ulitsa 2-ya Baumanskaya 5, Moscow, 105005, Russia; Ph.D., Associate Professor, e-mail: iliamarchevsky@mail.ru*

[2] *Bauman Moscow State Technical University, Faculty of Fundamental Sciences; ulitsa 2-ya Baumanskaya 5, Moscow, 105005, Russia; Assistant, e-mail: valeria.puzikova@gmail.com*

Received November 12, 2015

Abstract: The LS-STAG_ext parallel software package is developed for viscous incompressible flow simulation by the LS-STAG immersed boundary method and its modifications. This package allows one to simulate a flow around moving airfoils of arbitrary shape or around airfoil systems with one or two degrees of freedom. The LS-STAG_ext package supports Intel (R) Cilk (TM) Plus, Intel (R) Threading Building Blocks, and OpenMP parallel programming technologies. The efficiency comparison of parallel algorithms implemented in the LS-STAG_ext package LS-STAG_ext with similar software tools implemented in Intel (R) Math Kernel Library is discussed. In addition, the implementation features of the FGMRES method for solving systems of linear algebraic equations are described.

Keywords: Intel(R) Cilk(TM) Plus technology, Intel(R) Threading Building Blocks technology, OpenMP technology, Intel(R) Math Kernel Library, sparse linear systems, FGMRES method, BiCGStab method, PARDISO solver, viscous incompressible flow, LS-STAG immersed boundary method.

References

1. R. Mittal and G. Iaccarino, "Immersed Boundary Methods," Annu. Rev. Fluid Mech. **37**, 239–261 (2005).

2. Y. Cheny and O. Botella, "The LS-STAG Method: A New Immersed Boundary/Level-Set Method for the Computation of Incompressible Viscous Flows in Complex Moving Geometries with Good Conservation Properties," J. Comput. Phys. **229** (4), 1043–1076 (2010).

3. S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces* (Springer, New York, 2003).

4. V. V. Puzikova and I. K. Marchevsky, "Extension of the LS-STAG Immersed Boundary Method for RANS-Based Turbulence Models and Its Application for Numerical Simulation in Coupled Hydroelastic Problems," in *Proc. 6th Int. Conf. on Coupled Problems in Science and Engineering, Venice, Italy, May 18–20, 2015* (Int. Center for Numer. Methods in Engineering, Barcelona, 2015), pp. 532–543.

5. V. V. Puzikova, "On Generalization of the LS-STAG Immersed Boundary Method for Large Eddy Simulation and Detached Eddy Simulation," in *Advanced Problems in Mechanics International Summer School-Conference* (Inst. for Problems in Mechanical Engineering, St.-Petersburg, 2015), pp. 411–417.

6. V. V. Puzikova and I. K. Marchevsky, "Application of the LS-STAG Immersed Boundary Method for Numerical Simulation in Coupled Aeroelastic Problems," in *Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dynamics, Barcelona, Spain, July 20–25, 2014* (Int. Center for Numer. Methods in Engineering, Barcelona, 2014), pp. 1995–2006.

7. Intel(R) Cilk(TM) Plus [Electronic resource]. https://software.intel.com/ru-ru/node/522579. Cited December 3, 2015.

8. J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism* (O'Reilly, Sebastopol, 2007).

9. Intel® Math Kernel Library — Documentation [Electronic resource]. https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation. Cited December 3, 2015.

10. V. V. Puzikova, "Construction of the Level Function for a Profile of Arbitrary Shape by the LS-STAG Method when Modeling of Flow about this Profile," Vestn. Bauman Mosk. Tekh. Univ., Ser.: Natural Sci., No. 2, 163–173 (2012).

11. H. A. van der Vorst, "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for Solution of Nonsymmetric Linear Systems," SIAM J. Sci. Stat. Comp. **13** (2), 631–644 (1992).

12. M. Yu. Balandin and E. P. Shurina, *Methods for Solving Systems of Linear Algebraic Equations of High Dimension* (Novosibirsk Tekh. Univ., Novosibirsk, 2000) [in Russian].

13. P. Wesseling, *An Introduction to Multigrid Methods* (Willey, Chichester, 1991).

14. O. Schenk and K. Gärtner, "Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO," Future Gener. Comput. Syst. **20** (3), 475–487 (2004).

15. Y. Saad, *Iterative Methods for Sparse Linear Systems* (PWS Publ., New York, 1996).