заменяется коэффициентом искусственной вязкости, а теплопроводность рассчитывается в предположении Pr = 3/4.

Для коэффициента искусственной вязкости подобрано выражение, согласующееся с искусственной вязкостью Неймана–Рихтмайера, но имеющее свои особенности (обобщение на многомерность; введение «порогового» ограничителя). Входящие в это выражение коэффициенты выбраны в результате проведения большого количества тестовых расчетов.

Литература

1. Peery K. M., Imlay S. T. Blunt body flow simulations // AIAA Paper 88-2924, 1988.

2. Quirk J. J. A contribution to the great Riemann solver debate // Int. J. Numer. Meth. Fluids. 1994. Vol. 18. P. 555–574.

3. Родионов А. В., Тагирова И. Ю. Искусственная вязкость в схемах типа Годунова как метод подавления «карбункул»-неустойчивости // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2015. Вып. 2. С. 3–11.

4. Родионов А. В., Тагирова И. Ю. Применение искусственной вязкости для борьбы с «карбункул»-неустойчивостью в схемах типа Годунова // Математическое моделирование. 2015. Т. 27, № 10. 47–641.

5. von Neumann J., Richtmyer R. D. A method for the numerical calculation of hydrodynamic shocks // J. Appl. Phys. 1950. Vol. 21. P. 232–237.

6. Pandolfi M., D'Ambrosio D. Numerical instabilities in upwind methods: analysis and cures for the «carbuncle» phenomenon // J. Comput. Phys. 2001. Vol. 166. P. 271–301.

7. Noh W. F. Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux // J. Comput. Phys. 1987. Vol. 72. 78–120.

ОПТИМИЗАЦИЯ ОПЕРАЦИЙ С МАТРИЦАМИ В ПАКЕТЕ ПРОГРАММ ЛОГОС

И. П. Рыжачкин

Российский федеральный ядерный центр –
Всероссийский НИИ экспериментальной физики, г. Саров

Введение

В настоящее время для расчета задач гидродинамики и тепломассопереноса используются неявные методы, где алгоритмы дискретизации, равно как и методы математического моделирования должны соответствовать некоторым критериям в силу разнообразия физических процессов и неоднородности типов уравнений вычислительной гидродинамики. Моделирование течений вязкого сжимаемого газа производится при помощи методик, которые основываются на решении полной системы уравнений Навье–Стокса [1]. При этом значительная часть вычислительных ресурсов тратится на решение систем линейных алгебраических уравнений (СЛАУ). Так, в модуле ЛОГОС TVD на решение СЛАУ тратится от 40 % времени счета задачи. Создание алгоритмов, повышающих эффективность решения СЛАУ, является одним из основных оптимизационных направлений. Векторизация вычислений является одним из перспективных методов оптимизации.

В модуле TVD пакета программ ЛОГОС [2] при решении задач аэродинамики используется алгебраический многосеточный решатель. При этом уравнения Навье–Стокса описываются блоч-

ной матрицей, коэффициентами которой являются матрицы небольших размерностей – 5×5. При решении данной СЛАУ интенсивно используются операции умножения матриц mul_mm, умножения матрицы на вектор MUL_MV и операция инверсии матриц INV_M.

В данном докладе описываются:

– методы оптимизации пакета программ ЛОГОС;

– используемый в ЛОГОС векторизованный метод вычисления умножения малоразмерной матрицы на матрицу (функция mul_mm), умножения малоразмерной матрицы на вектор (функция MUL_MV) и инверсии матриц INV_M;

– производительность различных реализаций mul_mm, MUL_MV, INV_M в реализованной автором библиотеке IALM.

1. Особенности оптимизации пакета программ ЛОГОС

Процесс оптимизации ЛОГОС, написанного на языке C++, осуществляется технологически различными способами или этапами, из которых можно выделить следующие:

– Оптимизация алгоритмов приложения. При проектировании структур использование однородных по методам обработки массивов данных определяет успех векторной оптимизации.

– MPI – оптимизация, суть – разделение задачи между узлами вычислительного кластера на фрагменты с учетом вычислительных производительности узлов и скорости обмена информацией.

– Поточная (OpenMP) – оптимизация, суть – преобразование структур данных и внутренних циклов для повышения независимости вычислений потоками.

– Оптимизация кода на уровне языка C++. Код программы выстраивается таким образом, чтобы компилятор мог построить максимально быстрый скалярный или векторизованный SIMD код.

– Оптимизация с использованием «интринсиков» [3]. Интринсики являются расширением C и оптимизируются компиляторами от Intel и Microsoft наравне с прочими операторами языка. Поскольку векторизация вычислений является долговременным общемировой тенденцией, в какой-то степени интринсики приблизились к тому, чтобы стать стандартом C, C++, Fortran.

– Оптимизация с использованием ассемблера. Используется в виде ассемблерных вставок в текст программы. Преимущество такой оптимизации заключается в том, что

- с его помощью можно использовать практически все многообразие команд процессора (интринсики не всегда покрывают все многообразие векторных команд процессора),
- ассемблерный код не оптимизируется компилятором. Соответственно, при этом сводятся к минимуму как вероятность ухудшения производительности, так и вероятность внесения ошибок компиляции.

2. Умножение матриц mul_mm

В ЛОГОС было реализовано несколько вариантов умножения малоразмерных плотных матриц mul_mm.

В реализации mul_mm общего вида (mul_mms) во внутреннем цикле вычисляется скалярное произведение вектора-строки i матрицы A и вектора-столбца j матрицы B. Нужно заметить, что данный код плохо оптимизируется всеми без исключения компиляторами и является наихудшим по производительности.

Более быстрый код (mul_mmo) получается при развертке циклов mul_mms в линейный код для конкретного размера матриц.

Данный код также не является лучшим по производительности, поскольку не векторизуем современными компиляторами.

3. Оптимизация mul_mm при помощи интринсиков

Векторная реализация функции умножения малых матриц сталкивается с проблемой отсутствия на современных ×86 – архитектурах команды горизонтального сложения чисел векторного регистра к единственному результату. Для архитектуры Intel®AVX256 команда горизонтального сложения hadd (интринсик _mm256_hadd_pd (__m256d, __m256d)) складывает по 2 соседних числа регистра.

Для получения единой суммы нужно добавлять еще команды перегруппировки в регистре и сложения, существенно замедляя, таким образом, алгоритм.

Поэтому для архитектур Intel® AVX и Intel® Xeon Phi используется строчный алгоритм умножения матриц, исключающий необходимость горизонтального сложения данных в регистрах. Суть его в том, что каждая строка матрицы B умножается на элемент строки матрицы A. Данный алгоритм удобен в реализации еще и потому, что в современных архитектурах ×86 присутствует команда загрузки в регистр числа с одновременным размножением его по всем элементам регистра.

Ниже приводится реализация на интринсиках mul_mmi для случая с матрицами 4×4 и векторного расширения Intel®AVX256, которое характеризуется наличием 256-битных векторных регистров. Согласно вышеприведенному алгоритму загружаем:

1) элементы первой строки матрицы A с размножением

$$A = \begin{matrix} a_{00}a_{01}a_{02}a_{03} \\ a_{10}a_{11}a_{12}a_{13} \\ a_{20}a_{21}a_{22}a_{23} \\ a_{30}a_{31}a_{32}a_{33} \end{matrix}$$

по элементам 256-битных векторных регистров AVX

$$ymm_0 = a_{00}a_{00}a_{00}a_{00}$$
$$ymm_1 = a_{01}a_{01}a_{01}a_{01}$$
$$ymm_2 = a_{02}a_{02}a_{02}a_{02}$$
$$ymm_3 = a_{03}a_{03}a_{03}a_{03}$$

2) элементы строки матрицы B без размножения

$$ymm_4 = b_{03}b_{02}b_{01}b_{00}$$
$$ymm_5 = b_{13}b_{12}b_{11}b_{10}$$
$$ymm_6 = b_{23}b_{22}b_{21}b_{20}$$
$$ymm_7 = b_{33}b_{32}b_{31}b_{30}.$$

Далее производится (SIMD) операция поэлементного умножения векторов с сохранением результата в регистре-сумматоре

$$ymm_s = ymm_0 * ymm_4 = a_{00} * b_{03}a_{00} * b_{02}a_{00} * b_{01}a_{00} * b_{00}.$$

При сложении

$$ymm_s += ymm_1 * ymm_5$$
$$ymm_s += ymm_2 * ymm_6$$
$$ymm_s += ymm_3 * ymm_7.$$

получаем элементы 0-й строки матрицы C.

$$ymm_s = c_{03}c_{02}c_{01}c_{00}.$$

Аналогично вычисляются оставшиеся строки матрицы C.

4. Умножение матрицы на вектор MUL_MV

Данная операция записывается как

$$C = AB,$$

где A – матрица, B – вектор, представимый матрицей размером $n\times1$.

Поэтому ей присущи все недостатки алгоритмов умножения матрицы на матрицу, представленные в разделе 2.

5. Оптимизация MUL_MV при помощи интринсиков

Векторная реализация алгоритма умножения малой матрицы на вектор сталкивается с той же проблемой отсутствия на современных ×86 – архитектурах команды горизонтального сложения чисел векторного регистра к единственному результату, что и в случае с умножением матриц (см. п. 3).

Применяемый для архитектуры AVX [4] и Mic [4] алгоритм аналогичен описанному в п. 3 за исключением того, что размножать по элементам регистра приходится уже элементы вектора B.

Каждый столбец матрицы $A[m][n]$ (m-строк) загружается в вектор-строку и умножается на элемент вектора $B[n]$:

$$t_i = a_{ij}b_i, \quad i = 0, m. \tag{1}$$

Суммируя все вектора t поэлементно, получаем результирующую вектор-строку C.

Ниже приводится реализация алгоритма (1) на интринсиках для случая с матрицами 4×4 и векторного расширения AVX [4]. Для этого загружаем с размножением элементы вектора B

$$ymm_0 = b_0b_0b_0b_0$$
$$ymm_1 = b_1b_1b_1b_1$$
$$ymm_2 = b_2b_2b_2b_2$$
$$ymm_3 = b_3b_3b_3b_3$$

без размножения – столбцы матрицы A

$$ymm_4 = a_{30}a_{20}a_{10}a_{00}$$
$$ymm_5 = a_{31}a_{21}a_{11}a_{01}$$
$$ymm_6 = a_{32}a_{22}a_{12}a_{02}$$
$$ymm_7 = a_{33}a_{23}a_{13}a_{03}.$$

Далее производится (SIMD) операция поэлементного умножения векторов с сохранением результата в регистре-сумматоре

$$ymm_s = ymm_0 * ymm_4 = b_0 * a_{30}b_0 * a_{20}b_{00} * a_{10}b_0 * b_{00}.$$

При сложении

$$ymm_s += ymm_1 * ymm_5$$
$$ymm_s += ymm_2 * ymm_6$$
$$ymm_s += ymm_3 * ymm_7$$

получаем элементы вектора C.

$$ymm_s = c_3c_2c_1c_0.$$

6. Инверсия матриц INV_M

Данная операция записывается как

$$C = A^{-1},$$

где A – квадратная матрица.

Обращение матриц занимает в неоптимизированном модуле ЛОГОС TVD около 3 % времени. При обращении матриц использовался метод Гаусса–Жордана. Реализация метода Крамера на интринсиках позволила сократить общее время работы инверсии в ЛОГОС в 4,6 раза.

7. Производительность оптимизированных вычислений

В результате работы в рамках пакета программ ЛОГОС создан оптимизированный набор функций mul_mm, MUL_MV, INV_M, ориентированный на современные векторные архитектуры процессоров.

Ниже приводятся лучшие результаты производительности mul_mm*, MUL_MV* и INV_M, полученные по результатам использования нескольких компиляторов Intel® C последних версий.

На архитектуре AVX [4] в серверном исполнении на умножении малых матриц получено ускорение от 1,3 до 2,6 на наборе инструкций SSE2 и от 1,2 до 3,7 на наборе инструкций AVX по отношению к оптимизированному *C* – коду mul_mmo.

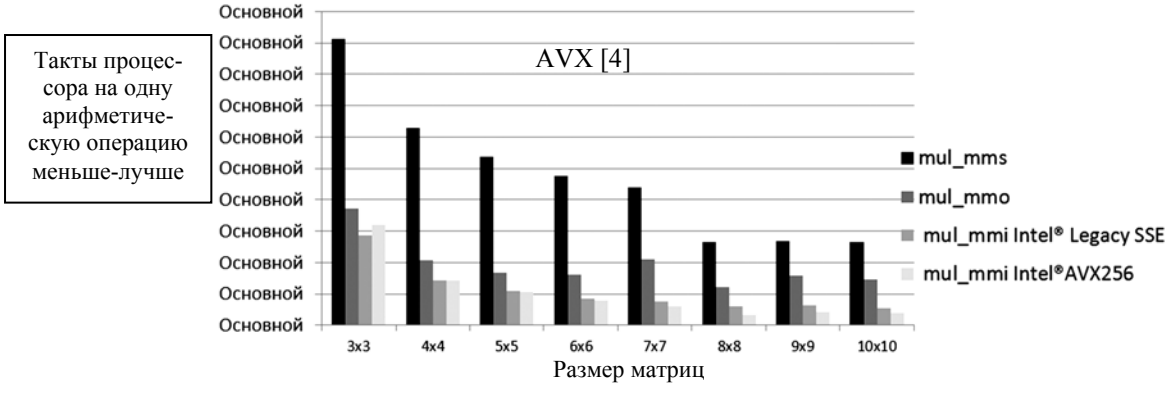

Рис. 1. Производительность mul_mm* на серверной архитектуре Intel®Sandy Bridge

На архитектуре Intel(R) Xeon(R) Phi 7120p @ 1.238 GHz (MIC[4]) ускорение составило от 1.2x до 3.7x относительно mul_mmo и 3x-8x относительно mul_mms.

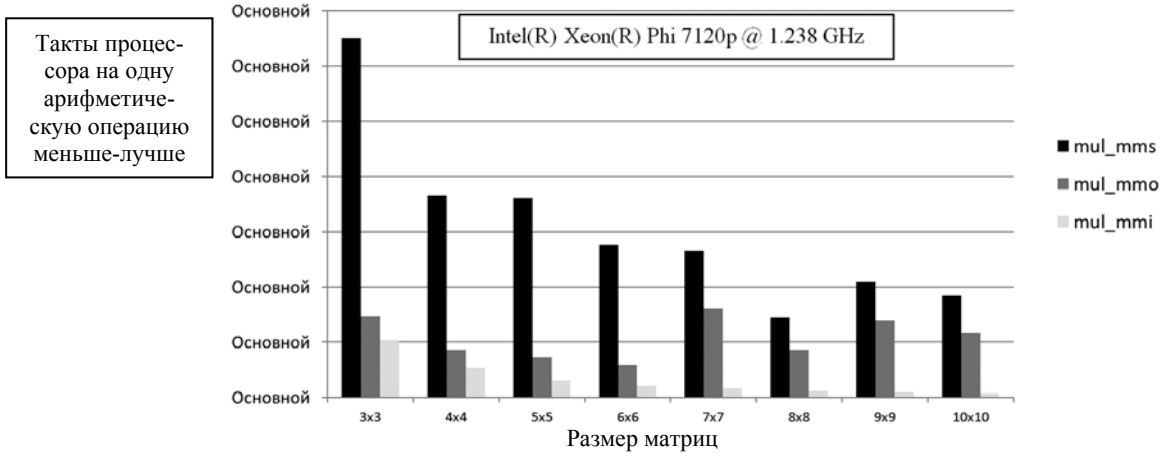

Рис. 2. Производительность mul_mm* на архитектуре Mic

На рис. 3 приводятся результаты сравнения производительности mul_mmi (функция ialm-Mul_mm_64f библиотеки IALM) и cblas_dgemm() библиотеки Intel®MKL на архитектуре AVX [4].

ialmMul_mm_64f() от 1.3 до 3-х раз быстрее cblas_dgemm() MKL[5] на размерностях от 3×3 до 10×10.

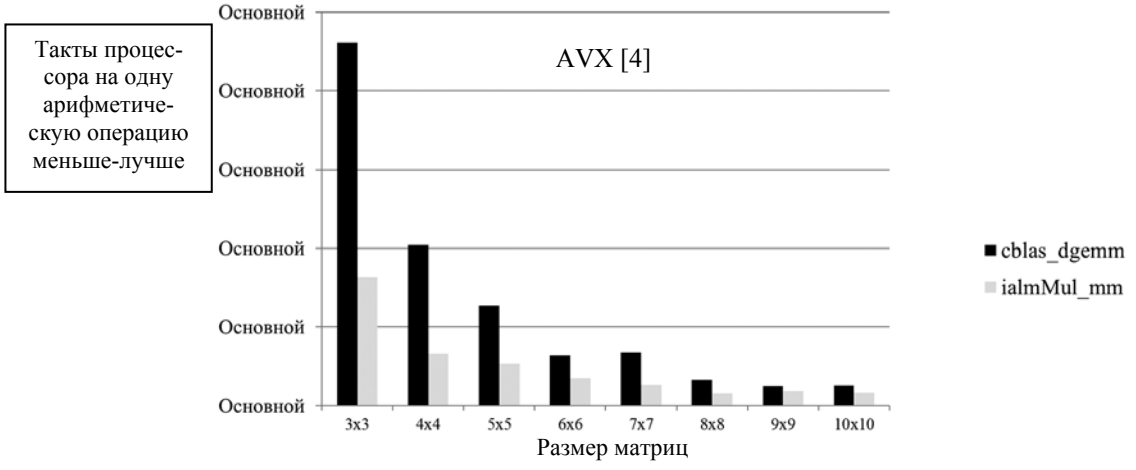

Рис. 3. IalmMul_mm_64f vs cblas_dgemm() библиотеки Intel®MKL

Результаты приведены в тактах процессора на одну арифметическую операцию mul_mm и рассчитывались по формуле:

$$N = \frac{o}{(2n-1)n \cdot n},$$

где o – число тактов, потраченное на вычисление mul_mm*, n – размерность квадратной матрицы, N – число тактов процессора, потраченное на одну арифметическую операцию при вызове функции mul_mm*.

Как видно из рисунка, код AVX начинает ощутимо выигрывать у SSE2, начиная с размерности 7×7. Небольшое снижение эффективности для нечетных размерностей 3,5,7,9 связано с необходимостью использовать векторные команды для обработки концевых неполных данных.

На рис. 3, 4 приводятся результаты оптимизации функции умножения малой матрицы на вектор ialmMul_mava_64f() библиотеки малых матриц IALM для архитектур AVX [4] и MIC [4] для матриц 5×5, которая производит поэлементное умножение массива матриц на массив векторов с сохранением результата в массив векторов.

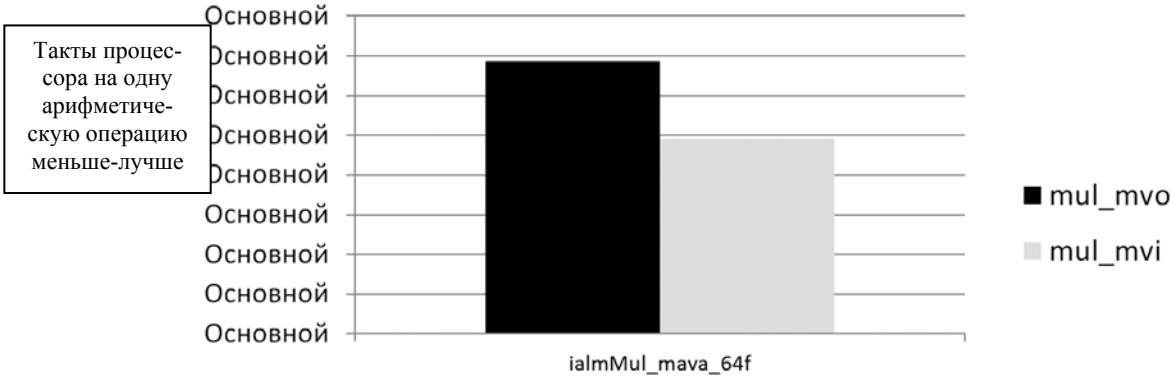

Рис. 4. Результаты оптимизации функции библиотеки IALM для Mic. Ускорение ~1,4×

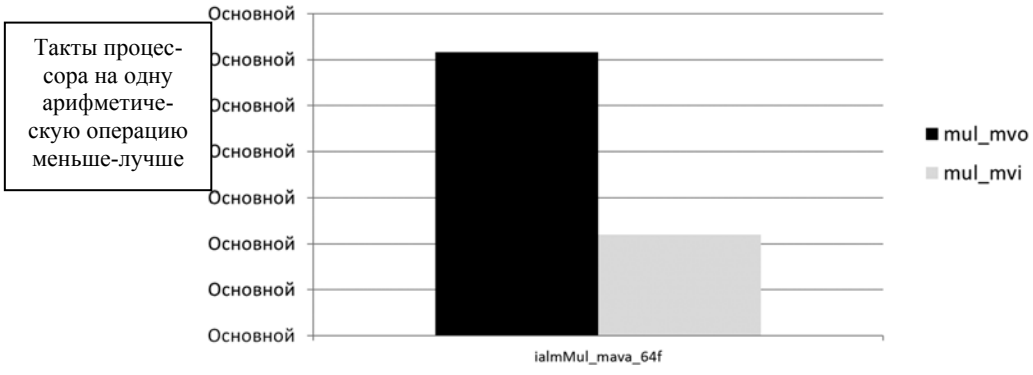

Рис. 5. Результаты оптимизации функции библиотеки IALM для AVX [4]. Ускорение ~3×

В библиотеке IALM реализована функция обращения матриц методом Крамера на *C* и интринсиках. На рис. 6 приводятся производительности вариантов обращения матриц.

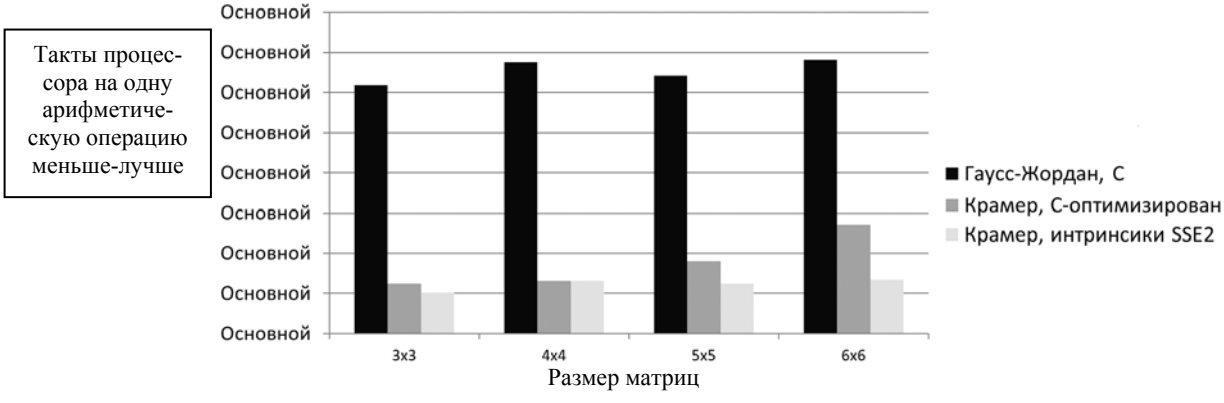

Рис. 6. Производительность инверсии матриц на архитектуре AMD Phenom. Ускорение 5-6×

Заключение

Рассмотренные оптимизированные варианты заметно ускорили решение мелкоблочных СЛАУ Навье–Стокса в ЛОГОС TVD, где на стадии построения агрегативного AMG на всех уровнях иерархии матриц выполняется умножение 5×5 блоков матриц при DILU-факторизации, а на стадиях сглаживания умножение 5×5 блоков матриц на блоки 5-ти компонент вектор (справедливо также для сглаживания Гаусса–Зейделя).

Применение функций библиотеки позволило распараллелить ранее не поддававшиеся (из-за ошибок компилятора) распараллеливанию участки кода LOGOS.

Разработанная оптимизированная библиотека IALM при решении задач ЛОГОС TVD ускоряет счет задачи «Трансзвуковое обтекание крыла профиля NACA0012» на 11 % в однопоточном и на 17 % в многопоточном режиме функционирования библиотечных функций при множественности процессов MPI.

В таблице приведены ускорения счета задачи о сверхзвуковом течении в плоском канале с клином (задача Klin), полученные в результате применения оптимизированной библиотеки IALM. Счет производился на 1 и 4 ядрах. В случае MIC 4 потока запускались на 1 ядре.

Ускорение LOGOS 6.0 на 1 и 4 OpenMP-потоках(п.) для AMD(R) SSE2, Intel(R)SSE2, Intel(R)AVX, MIC

AMD SSE2 (1п.)	Intel(R)SSE2 (1п.)	Intel(R)AVX (1п.)	Intel(R)AVX (4п.)	Mic (1п.)	Mic (4п.)
13 %	27 %	20 %	45 %	26 %	46 %

Нужно отметить, что для поддержания качества оптимизированных библиотек требуется мониторинг новых возможностей новых архитектур с последующей реализацией новых возможностей, если таковые будут востребованы. В данном контексте, «хорошей новостью» является как преемственность архитектур Intel, так и отмена закона Мура и модели «tic-toc» (первый год – новый технологический процесс, второй – новая архитектура). Можно предположить, что период рождения новых архитектур станет существенно дольше 2-х лет. К тому же множество команд процессора бесконечно расширяться не может, как и потребность во все более длинных регистрах (особенно при работе с малыми матрицами). И уже видна некоторая стабилизация в данной области. Реализация функции mul_mv на SSE2(128-битные регистры), например, существенно превосходит по производительности аналогичный код AVX(256-битные регистры). Данный недостаток AVX[4] и MIC [4] при работе с малыми матрицами будет преодолен только с появлением быстрых gather-команд, равно как и удобных команд горизонтального сложения.

Предполагаются следующие направления развития библиотеки IALM:

– реализовать диспетчеризацию кода, что позволит загружать целевой код для текущей архитектуры;

– оптимизировать основные операции до размерности 20×20;

– в перспективе оптимизация для архитектуры Эльбрус-8С.

Оптимизация при помощи функций библиотеки IALM покрывает около 20 % cpu-затратного (cpu expensive (CE)) кода ЛОГОС.

Оставшихся 80 % CE кода ЛОГОС планируется покрыть за счет создаваемой оптимизированной алгоритмической библиотеки IALL.

Литература

1. Волков К. Н., Емельянов В. Н. Вычислительные технологии в задачах механики жидкости и газа. М.: Физматлит, 2012.

2. Козелков А. С., Дерюгин Ю. Н., Зеленский Д. К. и др. Многофункциональный пакет программ ЛОГОС для расчета задач гидродинамики и тепломассопереноса на суперЭВМ. Базовые технологии и алгоритмы // XII Международный семинар «Супервычисления и математическое моделирование»:Труды XII Межд. семинара. Саров, 11 – 15 октября, 2010. С. 215 – 230.

3. «Intrinsics», USA [Electronic Resource] 2016. Mode of access: https://software.intel.com /

4. ООО «ЦКО», Саров, Россия [Electronic Resource] 2016. Mode of access: https://compcenter.org/

5. «MKL», USA [Electronic Resource] 2016. Mode of access: https://software.intel.com /