Сравнени стратегий распараллеливания векторизованного римановского решателя с помощью OpenMP для микропроцессора Intel Xeon Phi KNL

М.Ю. Воробьев[1], А.А. Рыбаков[2], А.Д. Чопорняк[3]

[1]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nordmike@jscc.ru;

[2]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, rybakov@jscc.ru;

[3]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, adc@jscc.ru

Аннотация. Римановские решатели широко используются в численных методах, при решении задач газовой динамики. При этом во время проведения вычислений требуется решать задачу Римана о распаде произвольного разрыва на каждой итерации расчетов для каждой пары соседних ячеек расчетной сетки. Таким образом, требуется иметь эффективную реализацию римановских решателей. В данной статье рассматривается задача о распараллеливании с помощью OpenMP применения векторизованного точного римановского решателя к массивам входных данных (массивы газодинамических параметров слева и справа от разрыва). Рассматриваются три различные стратегии распараллеливания. Рассматриваемые стратегии распараллеливания были реализованы в программном коде и протестированы на 72-ядерных микропроцессорах Intel Xeon Phi KNL. В результате проведенных численных экспериментов для наиболее эффективной стратегии распараллелив было получено суммарное ускорение векторизованного римановского решателя в 368 раз (при использовании 139 потоков) на одном микропроцессоре Intel Xeon Phi KNL по сравнению с однопоточной невекторизованной версией решателя.

Ключевые слова: газовая динамика, римановский решатель, векторизация, AVX-512, распараллеливание, OpenMP, Intel Xeon Phi KNL

1. Введение

Точное или приближенное решение задачи Римана о распаде произвольного разрыва используется в численных методах для нестационарных задач с большими разрывами. На решении задачи Римана базируется метод Годунова решения систем нестационарных уравнений газовой динамики [1,2]. В методе Годунова на каждой итерации расчетов на каждой грани между соседними расчетными ячейками решается задача Римана для определения потоков через эту грань. Так как современные расчетные сетки могут содержать десятки миллионов ячеек и более, то для эффективного использования данного метода требуется иметь эффективную реализацию римановских решателей. В мире известно большое количество работ, посвященных оптимизации работы приближенных римановских решателей с помощью векторизации, а также распараллеливания при использовании OpenMP и MPI [3-6]. В частности в работе [5] описаны подходы, позволившие достичь ускорения приближенного римановского решателя за счет векторизации вычислений в диапазоне 2,5-6,5 раз для вещественных операций с двойной точностью.

Данная статья посвящена точному римановскому решателю, который одновременно обладает компактным вычислительным ядром, а с другой стороны, содержит достаточно сложный программный контекст, который не может быть автоматический векторизован современными оптимизирующими компиляторами. В числе особенностей программного кода точного римановского решателя можно отметить сложное управление (с уровнем вложенности условных конструкций, достигающим значения 4), циклы с нерегулярным количеством итераций и вызовы функций.

Вопросы векторизации точного римановского решателя для микропроцессоров Intel Xeon Phi KNL в одномерном случае подробно освещены в работе [7]. При этом для векторизации программного кода использовался набор AVX-512 [8] широких векторных инструкций. Инструкции набора AVX-512 встраивались в программный код с помощью специальных функций интринсиков [9]. Для векторизации сложного программного контекста применялся подход, основный на переводе программного кода в предикатное представление и замене скалярных инструкций на векторные аналоги, описанный в работах [10-12].

2. Описание римановского решателя

В статье рассматривается реализация точного римановского решателя [13], которая находится в свободном доступе в сети Интернет в составе библиотеки NUMERICA [14]. На первом этапе выполнялась векторизация одномерной версии решателя [7]. Рассмотрим краткое описание процесса векторизации. Изначально одномерный римановский решатель по значениям газодинамических параметров (плотность, скорость и давление) справа и слева от разрыва находит значение данных параметров на самом разрыве в нулевой момент времени после устранения перегородки. Это можно записать в виде формулы в следующем виде:

$$U_l = \begin{pmatrix} d_l \\ u_l \\ p_l \end{pmatrix}, \quad U_r = \begin{pmatrix} d_r \\ u_r \\ p_r \end{pmatrix}$$

$$U = \begin{pmatrix} d \\ u \\ p \end{pmatrix} = riemann(U_l, U_r)$$

В данных формулах буквами d обозначены соответствующие значения плотности, u – скорости, p – плотности слева, справа и на самом разрыве соответственно.

Для выполнения векторизации функция, принимающая скалярные входные параметры, была заменена на функцию с векторными входными параметрами следующим образом:

$$\overline{U}_l = \begin{pmatrix} \overline{d}_l \\ \overline{u}_l \\ \overline{p}_l \end{pmatrix}, \quad \overline{U}_r = \begin{pmatrix} \overline{d}_r \\ \overline{u}_r \\ \overline{p}_r \end{pmatrix}$$

$$\overline{U} = \begin{pmatrix} \overline{d} \\ \overline{u} \\ \overline{p} \end{pmatrix} = riemann(\overline{U}_l, \overline{U}_r)$$

После этого массивы входных данных были разделены на отдельные участки длины 16, которые при условии использования вещественных данных одинарной точности упаковываются в векторные регистры _m512. Таким образом, при описанной перекомпоновке входных данных с помощью векторизации стало возможным решать одновременно 16 экземпляров задачи Римана на одном процессорном ядре.

После выполнения полной векторизации кода римановского решателя было достигнуто ускорение 7 раз для одномерного случая [7] и в районе 6,7 раз для решателя, расширенного на трехмерный случай (для этого требуются минимальные изменения по добавлению еще двух составляющих скорости).

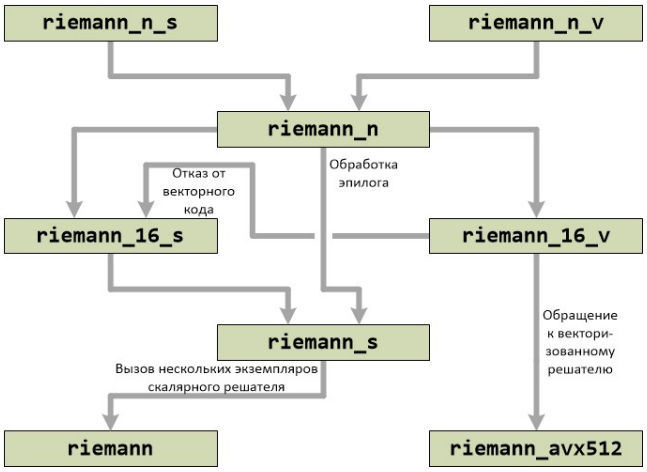

Рис. 1. Схема взаимодействия функций в реализации римановского решателя с поддержкой скалярной и векторизованной версий

Полная реализация векторизованного точного римановского решателя в трехмерном случае доступна в сети Интернет [15].

В числе основных функций в составе римановского решателя можно отметить следующие (см. рис. 1):

– riemann – оригинальная реализация решателя для скалярного набора входных данных.

– riemann_s – скалярная реализация для решения нескольких экземпляров задачи Римана.

– riemann_avx512 – векторная реализация с использованием инструкций AVX-512 решения 16 экземпляров задачи Римана.

– riemann_16_s – скалярная реализация решения 16 экземпляров задачи.

– riemann_16_v – реализация решения 16 экземпляров задачи, данная функция обращается либо к скалярной (riemann_16_s), либо к векторизованной (riemann_avx512) реализации в зависимости от флагов сборки.

– riemann_n – общий вызов функции для решения n экземпляров задачи, в которой можно выбирать, с помощью скалярного или векторного кода должна быть решена задача. Внутри данной функции выполняется реализация стратегий распараллеливания с помощью OpenMP. В частности выполняется разделение массивов входных данных на части по 16 элементов, обращение к функциям riemann_16_s и riemann_16_v, а также обработка эпилога цикла (если длина массивов входных данных не кратна 16, эпилог цикла обрабатывается с помощью функции riemann_s).

– riemann_n_s – решение n экземпляров задачи с помощью скалярного кода.

– riemann_n_v – решение n экземпляров задачи с помощью векторного кода.

3. Определение стратегий распараллеливания с помощью OpenMP

Микропроцессор Intel Xeon Phi KNL 7290, для которых проводилось исследование по распараллеливанию, содержат по 72 ядра, в каждом из которых возможно запустить по 4 потока, что дает суммарно 288 потоков для одного процессора. Ввиду этого применение распараллеливания с помощью OpenMP является ожидаемым, так как способно существенно ускорить исполняемый код. Были проанализированы 3 стратегии распараллеливания (см. рис. 2), описание которых приведено ниже.

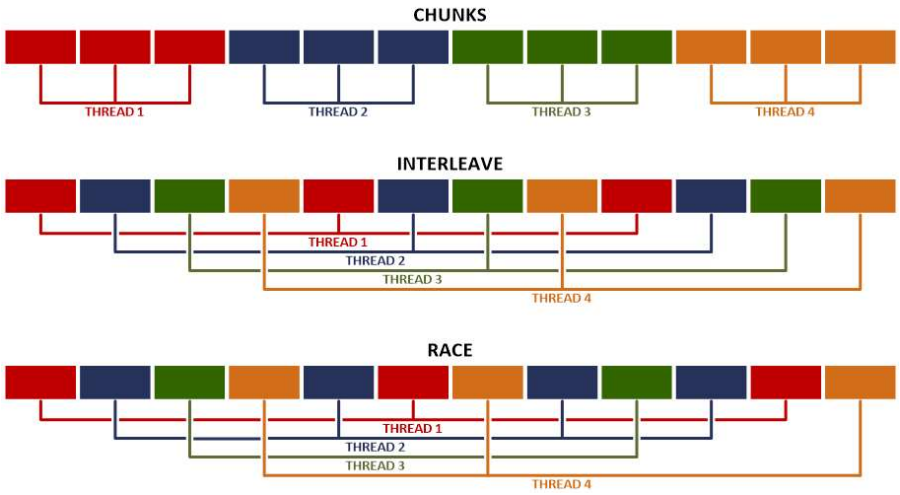

Рис. 2. Иллюстрация работы трех рассматриваемых стратегий распараллеливания вычислений: CHUNKS, INTERLEAVE, RACE

В качестве первой стратегии распараллеливания была использована стратегия CHUNKS, в которой массивы входных данных были разделены на равные непрерывные части по числу используемых потоков, каждый поток обрабатывал свои участки массивов входных данных (на листинге ниже nt – общее количество потоков, solver_16 – указатель на решатель для обработки 16 экземпляров задачи Римана).

```
// [15], riemann.cpp, 585-599
#pragma omp parallel
{
    int tn = omp_get_thread_num();
    int lb = (int)((c / FP16_VECTOR_SIZE) * ((double)tn / (double)nt));
    int ub = (int)((c / FP16_VECTOR_SIZE) * ((double)(tn + 1) / (double)nt));

    for (int i = lb * FP16_VECTOR_SIZE;
         i < ub * FP16_VECTOR_SIZE;
         i += FP16_VECTOR_SIZE)
    {
        solver_16(dl + i, ul + i, vl + i, wl + i, pl + i,
                  dr + i, ur + i, vr + i, wr + i, pr + i,
                  d + i, u + i, v + i, w + i, p + i);
    }
}
```

Во второй стратегии – INTERLEAVE – массивы входных данных были разделены на участки по 16 элементов и распределялись между потоками в шахматном порядке (на листинге ниже c_base – длина массивов входных данных без учета эпилога цикла, nt и solver_16 имеют тот же смысл, что и в листинге выше).

```
// [15], riemann.cpp, 603-615
#pragma omp parallel
{
    int tn = omp_get_thread_num();

    for (int i = tn * FP16_VECTOR_SIZE;
         i < c_base;
         i += nt * FP16_VECTOR_SIZE)
    {
        solver_16(dl + i, ul + i, vl + i, wl + i, pl + i,
                  dr + i, ur + i, vr + i, wr + i, pr + i,
                  d + i, u + i, v + i, w + i, p + i);
    }
}
```

Третья стратегия – RACE – основана на ведении глобального адреса следующего готового к обработке участка входных данных. Как только очередной поток освобождается, он приступает к обработке следующих свободных 16 экземпляров задачи. Таким образом была предпринята попытка избавиться от простоев потоков в случае разного времени выполнения отдельных экземпляров задачи.

```
// [15], riemann.cpp, 620-655
int g = 0;
#pragma omp parallel
{
    int i = 0;
    bool is_break = false;

    while (true)
    {
        #pragma omp critical
        {
            if (g >= c_base)
            {
                is_break = true;
            }
            else
            {
                i = g;
                g += FP16_VECTOR_SIZE;
            }
        }

        if (is_break)
        {
            break;
        }

        solver_16(dl + i, ul + i, vl + i, wl + i, pl + i,
                  dr + i, ur + i, vr + i, wr + i, pr + i,
                  d + i, u + i, v + i, w + i, p + i);
    }
}
```

На данном листинге g – глобальный счетчик следующей свободной партии экземпляров задачи Римана, доступный всем потокам. Для его проверки и продвижения требуется блокировка.

Для всех описанных стратегий были выполнены тестовые расчеты на суперкомпьютере МВС-10П.

4. Выполнение расчетов на суперкомпьютере

Тестирован различных стратегий распараллеливания обработки задачи Римана выполнялось на пакете [15]. Для сборки исполняемого файла под микропроцессор Intel Xeon Phi KNL использовалась строка компиляции следующего вида:

```
mpicc -I src test/*.cpp
src/*.cpp -DINTEL -
O3 -xmic-avx512 -qopt-re-
port=5 -DOPENMP_CHUNKS -DTEST_MODE=1 -
DREPEATS=3 -DINNER_REPEATS=100 -o rie-
mann.out -lm -fopenmp
```

При компиляции необходимо явно включить опцию использования инструкций AVX-512 (-xmic-avx512), также включен максимальный уровень оптимизации -O3. Данная строка предназначена для сборки исполняемого файла для стратегии распараллеливания CHUNKS. Для стратегий INTERLEAVE и RACE необходимо использовать флаги -DOPENMP_INTERLEAVE и -DOPENMP_RACE. Опция -DINNER_REPEATS=100 регулирует количество повторений тестовых сценариев при проведении вычислений (чем выше это количество, тем ниже погрешность замеров времени выполнения). Опция -DREPEATS=3 регулирует количество повторений, выполняемых для замеров времени исполнения теста.

После сборки теста его запуск выполняется с помощью команды

```
./riemann.out 288
```

Единственным параметров теста является максимальное количество потоков, на которых нужно выполнять прогон. При этом сначала тесты прогоняются на невекторизованном решателе с использованием одного потока (относительно этого запуска считается суммарное ускорение решателя). После этого выполняется запуск векторизованного решателя для различного количества потоков, начиная от 1 и заканчивая 288.

По результатам прогона выдается время работы как невекторизованного решателя на одном потоке, так и векторизованной версии для разного количества потоков. Отдельно выполнялись прогоны для стратегий распараллеливания CHUNKS, INTERELEAVE и RACE. По результатам прогонов были построены графики суммарного ускорения векторизованного и распараллеленного римановского решателя для всех трех стратегий, данные графики представлены на рис. 3.

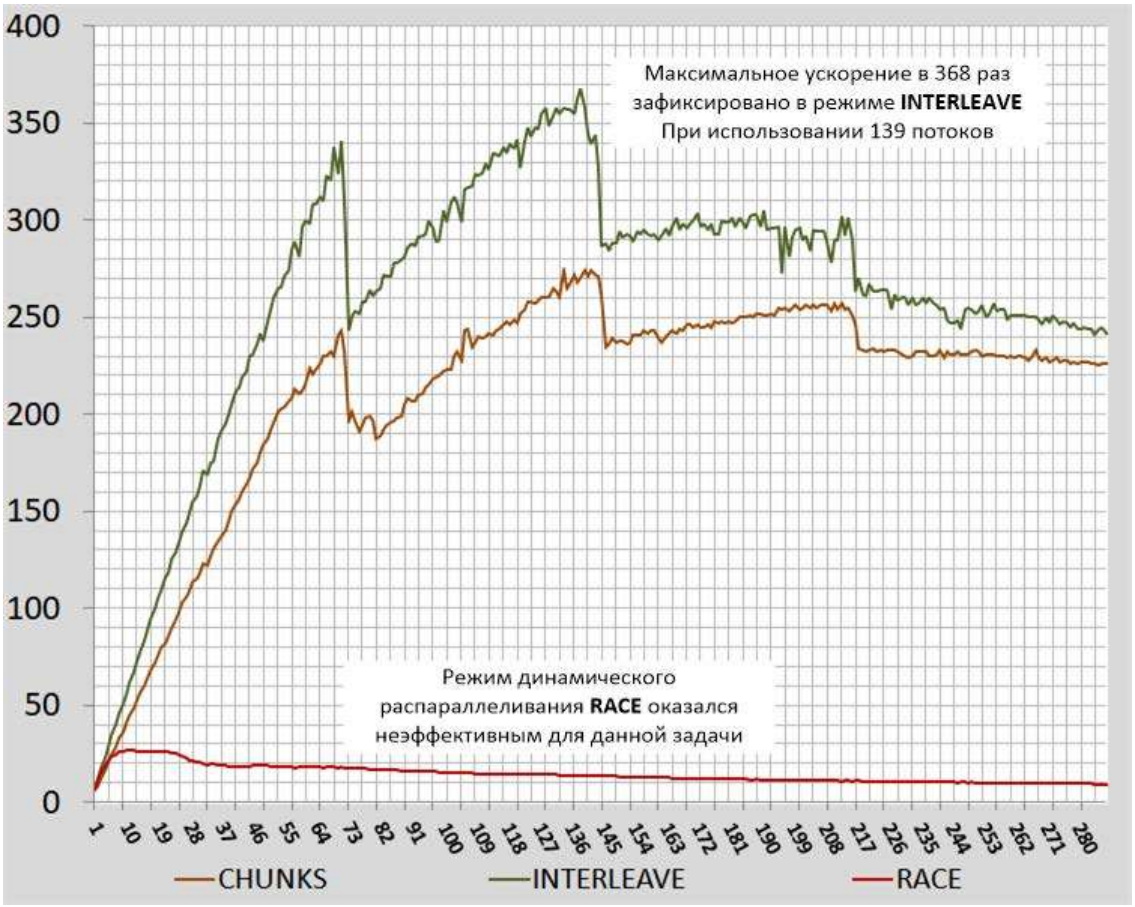

Рис. 3. График ускорения распараллеленного векторизованного римановского решателя по сравнению с нераспараллеленной невекторизованной версией

Из рис. 3 видно, что стратегия распараллеливания RACE является нежизнеспособной даже на сравнительно небольшом числе потоков. Блокировка глобального ресурса (счетчик следующей свободной партии задач), оказывается фатальной и приводит к простою потоков.

Эффективность двух других стратегий примерно одинакова, однако стратегия INTERLEAVE показала себя все же лучше, продемонстрировав максимальное ускорение 368 раз на 139 потоках.

На графиках явно просматриваются 4 характерные участка, длина которых совпадает с количеством ядер микропроцессора Intel Xeon Phi KNL 7290 (72 ядра). На первом участке масштабируемость распараллеливания близка к линейной. На втором участке наблюдается сначала некоторое снижение эффективности, но затем при дальнейшем увеличении количества потоков удается добиться дополнительного ускорения. На третьем и четвертом участках графиков наблюдается снижение производительности. Это ожидаемо, так как каждое ядро микропроцессора содержит два векторных исполнительных устройства и при запуске на ядре трех или четырех потоков начинается конкуренция за данные исполнительные устройства.

5. Заключение

Проведенные исследования по векторизации римановского решателя с помощью набора инструкций AVX-512 и распараллеливанию его с помощью OpenMP показали важность данной задачи, так как комбинирование данных методов позволило добиться ускорения решателя более чем на 2 порядка. Максимальное ускорение по распараллеливанию векторизованного трехмерного решателя составило 368 раз при использовании 139 потоков на микропроцессоре Intel Xeon Phi KNL.

Публикация выполнена в рамках проекта РФФИ «Векторизация римановского решателя для мультиядерных микропроцессоров» (№ 18-07-00638). В работе был использован суперкомпьютер МВС-10П, находящийся в МСЦ РАН.

Comparison of Strategies for Parallelizing a Vectorized Riemann Solver Using OpenMP for the Intel Xeon Phi KNL Microprocessor

M.Yu. Vorobyev, A.A. Rybakov, A.D. Chopornyak

Abstract. Riemann solvers are widely used in numerical methods, in the solution of gas dynamics problems. During the calculations, it is required to solve the Riemann problem on the decay of an arbitrary discontinuity at each iteration of the calculations for each pair of neighboring cells of the computational grid. Thus, an efficient implementation of Riemann solvers is required. This article discusses the problem of parallelization using OpenMP of applying a vectorized exact Riemann solver to arrays of input data (arrays of gas dynamic parameters to the left and right of the discontinuity). Three different parallelization strategies are considered. The considered parallelization strategies were implemented in software code and tested on 72-core Intel Xeon Phi KNL microprocessors. As a result of numerical experiments for the most efficient parallelization strategy, the total acceleration of the vectorized Riemann solver by a factor of 368 (using 139 threads) was obtained on one Intel Xeon Phi KNL microprocessor in comparison with the single-threaded unvectorized version of the solver.

Keywords: gas dynamics, Riemann solver, vectorization, AVX-512, parallelization, OpenMP, Intel Xeon Phi KNL

Литература

1. А.Г. Куликовский, Н.В. Погорелов, А.Ю. Семенов. Математические вопросы численного решения гиперболических систем уравнений. ФИЗМАТЛИТ, М., 2001, 608 с.
2. С.К. Годунов, А.В. Забродин, М.Я. Иванов, А.Н. Крайко, Г.П. Прокопов. Численное решение многомерных задач газовой динамики. Наука, М., 1976, 400 с.
3. I. Kulikov, I. Chernykh, V. Vshivkov, V. Prigarin, V. Mironov, A. Tatukov. The parallel hydrodynamic code for astrophysical flow with stellar equation of state. Proceedings of Russian Supercomputing Days 2018, 612-624.
4. M. Bader, A. Breuer, W. Höltz, S. Rettenberger. Vectorization of an augmented Riemann solver for

the shallow water equations. Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014, 2014, 193-201.

5. C.R. Ferreira, K.T. Mandli, M. Bader. Vectorization of Riemann solvers for the single- and multi-layer shallow water equations. Proceedings of the 2018 International Conference on High Performance Computing and Simulation, HPCS 2018, 2018, 415-422.

6. H.-Y. Schive, U.-H. Zhang, T. Chiueh. Directionally unsplit hydrodynamic schemes with hybrid MPI/OpenMP/GPU parallelization in AMR. International Journal of High Performance Computing Applications, 2011, 367-377.

7. A.A. Rybakov, S.S. Shumilin. Vectorization of the Riemann solver using AVX-512 instruction set. Program Systems: Theory and Applications, 2019, Vol. 10, № 3 (42), 41-58.

8. Intel 64 and IA-32 architectures software developer's manual. Combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4. Intel Corporation, 2019.

9. Intel Intrinsics Guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide/, дата обращения 31.08.2020.

10. B.M. Shabanov, A.A. Rybakov, S.S. Shumilin, Vectorization of high-performance scientific calculations using AVX-512 instruction set. Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 5, 580-598.

11. А.А. Рыбаков, С.С. Шумилин. Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций. Программные системы: Теория и алгоритмы, 2019, Т. 10, № 4 (43), 77-96.

12. А.А. Рыбаков, С.С. Шумилин. Векторизация сильно разветвленного управления с помощью инструкций AVX-512. Труды НИИСИ РАН, 2018, Т. 8, № 4, 114-126.

13. E.F. Toro. Riemann solvers and numerical methods for fluid dynamics. A practical introduction. 2nd edition, Springer, 1999.

14. NUMERICA, A Library of Sources for Teaching, Research and Applications, by E. F. Toro. https://github.com/dasikasunder/NUMERICA, дата обращения 31.08.2020.

15. https://github.com/r-aax/riemann_vec/releases/tag/metric, дата обращения 31.08.2020.