Приложение Ж

МИНОБРНАУКИ РОССИИ
ФИЛИАЛ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО УЧРЕЖДЕНИЯ «ФЕДЕРАЛЬНЫЙ НАУЧНЫЙ ЦЕНТР НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ СИСТЕМНЫХ ИССЛЕДОВАНИЙ РОССИЙСКОЙ АКАДЕМИИ НАУК»
Межведомственный суперкомпьютерный центр
Российской академии наук
(МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН)

УТВЕРЖДАЮ
Директор МСЦ РАН – руководитель
КП вычислительными ресурсами
МСЦ РАН – филиала ФГУ ФНЦ
НИИСИ РАН
Б.М. Шабанов
«30» 09 2020 г.

Методики измерения производительности вычислительных систем

(методика измерения эффективности векторизации, методики измерения эффективности масштабирования высоконагруженных вычислений для систем с общей памятью)

Образовательные программы рассмотрены и одобрены на заседании Ученого совета МСЦ РАН – секции Ученого совета ФГУ ФНЦ НИИСИ РАН
«17» 09 2020 г., протокол № 4

Москва
2020

СВЕДЕНИЯ О РАЗРАБОТКЕ

ИСПОЛНИТЕЛЬ: Межведомственный суперкомпьютерный центр Российской академии наук – филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук».

Руководитель организации-разработчика: Шабанов Борис Михайлович.

ИСПОЛНИТЕЛИ:

Зам. директора МСЦ РАН _____ 31.08.2020 А. А. Рыбаков

Старший научный сотрудник _____ 31.08.2020 А. Д. Чопорняк

Младший научный сотрудник _____ 31.08.2020 С. С. Шумилин

Младший научный сотрудник _____ 31.08.2020 М. Ю. Воробьев

ВВЕДЕНИЕ

Повышение эффективности работы пользовательских приложений на вычислительных системах является актуальной задачей. При оптимизации суперкомпьютерных приложений возникает потребность повышения эффективности выполнения программного кода на разных уровнях распараллеливания: параллельность между узлами вычислительного кластера (например, распараллеливание приложения с помощью MPI), многопоточное распараллеливание приложений на общей памяти (например, распараллеливание приложений с помощью OpenMP), векторизация вычислений (например, использование векторных инструкций, поддержанных в современных микропроцессорах). Разработанные методики относятся к измерению эффективности распараллеливания приложений на последних двух уровнях (распараллеливание на общей памяти с помощью OpenMP и векторизация с помощью специального набора инструкций Intel AVX-512). В качестве программного кода, используемого для измерения эффективности распараллеливания и векторизации, был выбран функционал точного решения задачи Римана о распаде произвольного разрыва в трехмерной постановке. Реализация решения данной задачи обладает компактным вычислительным ядром и в то же время содержит программный контекст, отличающийся большим разнообразием: сложное управление, вложенные циклы, вызовы функций. Таким образом, реализация точного римановского решателя является представительным тестом для измерения эффективности распараллеливания и векторизации.

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].

ОБЛАСТЬ ПРИМЕНЕНИЯ

Настоящие методики измерений устанавливают процедуру измерения эффективности векторизации программного кода со сложным программным контекстом с помощью специального набора инструкций Intel AVX-512, а также процедуру измерения масштабируемости высоконагруженных вычислений для систем с общей памятью с применением технологии OpenMP.

Для применения настоящих методик требуется сборка тестовой задачи, написанной на языке программирования C, с помощью компилятора icc.

При реализации тестовой задачи была использована библиотека функций-интринсиков для поддержки набора инструкций Intel AVX-512 (immintrin.h). Настоящие методики применимы для вычислительных систем, использующих микропроцессоры Intel с поддержкой набора инструкций AVX-512. К таким микропроцессорам относятся, например, микропроцессоры семейств Intel Xeon Phi Knights Landing, Intel Xeon Skylake, Intel Xeon Cascade Lake.

При реализации тестовой задачи для обеспечения распараллеливания на общей памяти была использована технология OpenMP. Таким образом, для проведения измерений на вычислительной системе необходима поддержка данного инструмента (использовались средства поддержки OpenMP компилятора icc, заголовочный файл omp.h, директивы компилятора #pragma omp parallel и #pragma omp critical).

Исходный код тестовой задачи, используемой при проведении измерений, доступен сети Интернет в репозитории на ресурсе github.com [7]. Тестовая задача базируется на открытой реализации точного римановского решателя в одномерном случае, доступной в библиотеке NUMERICA [8], реализация которой основана на работе [9].

ОРГАНИЗАЦИЯ ВЕКТОРИЗАЦИИ ВЫЧИСЛЕНИЙ

Тестовая задача для замера эффективности векторизации сложного программного контекста содержит две реализации точного римановского решателя в трехмерной постановке: скалярная реализация и векторная реализация, позволяющая одновременно решать 16 экземпляров задачи на одном процессорном ядре. Для измерения эффективности векторизации обе реализации римановского решателя запускаются на исполнение на тестовых наборах данных, замеряются времена работы программы, после чего производится сравнение полученных результатов. При проведении векторизации программного контекста римановского решателя были использованы подходы к векторизации, описанные в работах [10-18].

Расчетное ядро задачи Римана о распаде произвольного разрыва представлено в четырех основных функциях (guessp, prefun, starpu, sample) для каждой из которых была выполнена векторизация с использованием функций-интринсиков для поддержки набора инструкций AVX-512. Ниже приведена реализация указанных функций в оригинальном и векторизованном виде.

guessp

```
static void
guessp(float dl, float ul  float pl, float cl,
       float dr, float ur  float pr, float cr, float &pm)
{
    float cup, gel, ger, pmax, pmin, ppv, pq, ptl, ptr, qmax, quser, um;
    quser = 2.0;
    cup = 0.25 * (dl + dr  * (cl + cr);
    ppv = 0.5 * (pl + pr) + 0.5 * (ul - ur) * cup;
    ppv = (ppv > 0.0) ? ppv : 0.0;
    pmin = (pl < pr) ? pl : pr;
    pmax = (pl > pr) ? pl : pr;
    qmax = pmax / pmin;

    if ( qmax <= quser) && (pmin <= ppv) && (ppv <= pmax))
    {
        pm = ppv;
    }
    else
    {
        if (ppv < pmin)
        {
            pq = pow(pl / pr, G1);
            um = (pq * ul / cl + ur / cr + G4 * (pq - 1.0)) / (pq / cl + 1.0 / cr);
            ptl = 1.0 + G7 * (ul - um) / cl;
            ptr = 1.0 + G7 * (um - ur) / cr;
            pm = 0.5 * (pow(pl * ptl, G3) + pow(pr * ptr, G3) ;

        else

            gel = sqrt((G5 / dl) / (G6 * pl + ppv));
            ger = sqrt((G5 / dr) / (G6 * pr + ppv));
            pm = (gel * pl + ger * pr - (ur - ul)) / (gel + ger);

    }
}
```

```
static void
guessp_avx512(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
              __m512 dr, __m512 ur, __m512 pr, __m512 cr, __m512 *pm)
{
    __m512 two, half, cup, ppv, pmin, pmax, qmax, pq, um, ptl, ptr, gel, ger, pqcr;
    __mmask16 cond_pvrs, cond_ppv, ncond_ppv;
    two = SET1(2.0);
    half = SET1(0.5);
    cup = MUL(SET1(0.25), MUL(ADD(dl, dr), ADD(cl, cr)));
    ppv = MUL(half, FMADD(SUB(ul, ur), cup, ADD(pl, pr)));
    ppv = MAX(ppv, z);
    pmin = MIN(pl, pr);
    pmax = MAX(pl, pr);
    qmax = DIV(pmax, pmin);
    cond_pvrs = CMP(qmax, two, _MM_CMPINT_LE)
              & CMP(pmin, ppv, _MM_CMPINT_LE)
              & CMP(ppv, pmax, _MM_CMPINT_LE);
    cond_ppv = _mm512_mask_cmp_ps_mask(~cond_pvrs, ppv, pmin, _MM_CMPINT_LT);
    ncond_ppv = ~cond_pvrs & ~cond_ppv;
    *pm = _mm512_mask_mov_ps(*pm, cond_pvrs, ppv);

    if (cond_ppv != 0x0)
    {
        pq = _mm512_mask_pow_ps(z, cond_ppv,
                                _mm512_mask_div_ps(z, cond_ppv, pl, pr), gl);
        pqcr = MUL(pq, cr);
        um = _mm512_mask_div_ps(z, cond_ppv,
                                FMADD(FMADD(SUB(pqcr, cr), g4, ur), cl, MUL(pqcr, ul)),
                                ADD(pqcr, cl));
        ptl = FMADD(_mm512_mask_div_ps(z, cond_ppv, SUB(ul, um), cl), g7, one);
        ptr = FMADD(_mm512_mask_div_ps(z, cond_ppv, SUB(um, ur), cr), g7, one);
        *pm = _mm512_mask_mul_ps(*pm, cond_ppv, half,
                                 ADD(_mm512_mask_pow_ps(z, cond_ppv, MUL(pl, ptl), g3),
                                     _mm512_mask_pow_ps(z, cond_ppv, MUL(pr, ptr), g3)));
    }

    if (ncond_ppv != 0x0)
    {
        gel = SQRT(_mm512_mask_div_ps(z, ncond_ppv, g5, MUL(FMADD(g6, pl, ppv), dl)));
        ger = SQRT(_mm512_mask_div_ps(z, ncond_ppv, g5, MUL(FMADD(g6, pr, ppv), dr)));
        *pm = _mm512_mask_div_ps(*pm, ncond_ppv,
                                 FMADD(gel, pl, FMADD(ger, pr, SUB(ul, ur))),
                                 ADD(gel, ger));
    }
}
```

prefun

```
static void
prefun(float &f, float &fd, float &p, float &dk, float &pk, float &ck)
{
    float ak, bk, pratio, qrt;

    if (p <= pk)
    {
        pratio = p / pk;
        f = G4 * ck * (pow(pratio, G1) - 1.0);
        fd =  1.0 / (dk * ck) * pow(pratio, -G2);
    }
    else
    {
        ak = G5 / dk;
        bk = G6 * pk;
        qrt = sqrt(ak / (bk + p));
        f = (p - pk) * qrt;
        fd =  1.0 - 0.5 * (p - pk) / (bk + p)) * qrt;
    }
}

static void
prefun_avx512(__m512 *f, __m512 *fd, __m512 p,
              __m512 dk, __m512 pk, __m512 ck, __mmask16 m)
{
    __m512 pratio, ak, bkp, ppk, qrt;
    __mmask16 cond, ncond;

    cond = _mm512_mask_cmp_ps_mask(m, p, pk, _MM_CMPINT_LE);
    ncond = m & ~cond;
```

```
    if (cond != 0x0)
    {
        pratio = _mm512_mask_div_ps(z, cond, p, pk);
        *f = _mm512_mask_mul_ps(*f, cond, MUL(g4, ck),
                                SUB(_mm512_mask_pow_ps(z, cond, pratio, g1), one));
        *fd = _mm512_mask_div_ps(*fd, cond,
                                 _mm512_mask_pow_ps(z, cond, pratio  SUB(z, g2)),
                                 MUL(dk, ck));
    }

    if (ncond != 0x0)
    {
        ak = _mm512_mask_div_ps(z, ncond, g5, dk);
        bkp = FMADD g6, pk, p);
        ppk = SUB(p, pk);
        qrt = _mm512_mask_sqrt_ps(z, ncond,
                                  _mm512_mask_div_ps(z, ncond, ak, bkp));
        *f = _mm512_mask_mul_ps(*f, ncond, ppk, qrt);
        *fd = _mm512_mask_mul_ps(*fd, ncond, qrt,
                                 _mm512_fnmadd_ps(_mm512_mask_div_ps(z, ncond, ppk, bkp),
                                                  SET1(0.5), one));
    }
}
```

starpu

```
static void
starpu(float dl, float ul, float pl, float cl,
       float dr, float ur, float pr, float cr, float &p, float &u)
{
    const int nriter = 20;
    const float tolpre = 1.0e-6;
    float change, fl, fld, fr, frd, pold, pstart, udiff;

    guessp(dl, ul, pl, cl, dr, ur, pr, cr, pstart);
    pold = pstart;
    udiff = ur - ul;
    int i = 1;

    for ( ; i <= nriter; i++)
    {
        prefun(fl, fld, pold, dl, pl, cl);
        prefun(fr, frd, pold, dr, pr, cr);
        p = pold -  fl + fr + udiff) / (fld + frd);
        change = 2.0 * abs((p - pold) / (p + pold));
        if (change <= tolpre) { break; }
        if (p < 0.0) { p = tolpre; }
        pold = p;
    }

    if (i > nriter)
    {
        cout << "divergence in Newton-Raphson iteration" << endl;
        exit(1);
    }

    u = 0.5 * (ul + ur + fr - fl);
}

static void
starpu_avx512(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
              __m512 dr, __m512 ur, __m512 pr, __m512 cr, __m512 *p, __m512 *u)
{
    __m512 two, tolpre, tolpre2, udiff, pold, fl, fld, fr, frd, change;
    __mmask16 cond_break, cond_neg, m;
    const int nriter = 20;
    int iter = 1;

    two = SET1(2.0);
    tolpre = SET1(1.0e-6);
    tolpre2 = SET1 5.0e-7);
    udiff = SUB(ur, ul);
    guessp_avx512(dl, ul, pl, cl, dr, ur, pr, cr, &pold );
    m = 0xFFFF;

    for (; (iter <= nriter) && (m != 0x0); iter++)
    {
```

```
        prefun_avx512(&fl, &fld, pold, dl, pl, cl, m);
        prefun_avx512(&fr, &frd, pold, dr, pr, cr, m);
        *p = _mm512_mask_sub_ps(*p, m, pold,
                                _mm512_mask_div_ps(z, m,
                                                   ADD(ADD(fl, fr), udiff),
                                                   ADD(fld, frd)));
        change = ABS(_mm512_mask_div_ps(z, m, SUB(*p, pold), ADD(*p, pold)));
        cond_break = _mm512_mask_cmp_ps_mask(m, change, tolpre2, _MM_CMPINT_LE);
        m &= ~cond_break;
        cond_neg = _mm512_mask_cmp_ps_mask(m, *p, z, _MM_CMPINT_LT);
        *p = _mm512_mask_mov_ps(*p, cond_neg, tolpre);
        pold = _mm512_mask_mov_ps(pold, m, *p);
    }

    if (iter > nriter)
    {
        cout << "divergence in Newton-Raphson iteration" << endl;
        exit(1);
    }

    *u = MUL(SET1(0.5), ADD(ADD(ul, ur), SUB(fr, fl)));
}
```

sample

```
static void
sample(float dl, float ul, float vl, float wl, float pl, float cl,
       float dr, float ur, float vr, float wr, float pr, float cr,
       const float pm, const float um,
       float &d, float &u, float &v, float &w, float &p)
{
    float c, cml, cmr, pml, pmr, shl, shr, sl, sr, stl, str;

    if (0.0 <= um)
    {
        v = vl; w = wl;
        if (pm <= pl)
        {
            shl = ul - cl;
            if (0.0 <= shl) { d = dl; u = ul; p = pl; }
            else
            {
                cml = cl * pow(pm / pl, G1);
                stl = um - cml;
                if (0.0 > stl)
                {
                    d = dl * pow(pm / pl, 1.0 / GAMA);
                    u = um; p = pm;
                }
                else
                {
                    u = G5 * (cl + G7 * ul);
                    c = G5 * (cl + G7 * ul);
                    d = dl * pow(c / cl, G4);
                    p = pl * pow(c / cl, G3);
                }
            }
        }
        else
        {
            pml = pm / pl;
            sl = ul - cl * sqrt(G2 * pml + G1);
            if (0.0 <= sl) { d = dl; u = ul; p = pl; }
            else
            {
                d = dl * (pml + G6) / (pml * G6 + 1.0);
                u = um; p = pm;
            }
        }
    }
    else
    {
        v = vr; w = wr;
        if (pm > pr)
        {
            pmr = pm / pr;
            sr = ur - cr * sqrt(G2 * pmr + G1);
            if (0.0 >= sr) { d = dr; u = ur; p = pr; }
            else
```

```
            {
                d = dr * (pmr + G6) / (pmr * G6 + 1.0)
                u = um; p = pm;
            }
        }
        else
        {
            shr = ur + cr;
            if (0.0 >= shr) { d = dr; u = ur; p = pr;
            else
            {
                cmr = cr * pow(pm / pr, G1);
                str = um - cmr;

                if (0.0 <= str)
                {
                    d = dr * pow(pm / pr, 1.0 / GAMA);
                    u = um; p = pm;
                }
                else
                {
                    u = G5 * (-cr + G7 * ur);
                    c = G5 * (cr - G7 * ur);
                    d = dr * pow(c / cr, G4);
                    p = pr * pow(c / cr, G3);
                }
            }
        }
    }
}

static void
sample_avx512(__m512 dl, __m512 ul, __m512 vl, __m512 wl, __m512 pl, __m512 cl,
              __m512 dr, __m512 ur, __m512 vr, __m512 wr, __m512 pr, __m512 cr,
              __m512 pm, __m512 um,
              __m512 *d, __m512 *u, __m512 *v, __m512 *w, __m512 *p)
{
    __m512 c, ums, pms, sh, st, s, uc;
    __mmask16 cond_um, cond_pm, cond_sh, cond_st, cond_s, cond_sh_st;

    cond_um = _mm512_cmp_ps_mask(um, z, _MM_CMPINT_LT)
    *d = _mm512_mask_blend_ps(cond_um, dl, dr);
    *u = _mm512_mask_blend_ps(cond_um, ul, ur);
    *v = _mm512_mask_blend_ps(cond_um, vl, vr);
    *w = _mm512_mask_blend_ps(cond_um, wl, wr);
    *p = _mm512_mask_blend_ps(cond_um, pl, pr);
    c = _mm512_mask_blend_ps(cond_um, cl, cr);
    ums = um;
    *u = _mm512_mask_sub_ps(*u, cond_um, z, *u);
    ums = _mm512_mask_sub_ps(ums, cond_um, z, ums);

    pms = DIV(pm, *p);
    sh = SUB(*u, c);
    st = _mm512_fnmadd_ps(POW(pms, g1), c, ums);
    s = _mm512_fmadd_ps(c, SQRT FMADD(g2, pms, g1)), *u);

    cond_pm = _mm512_cmp_ps_mask pm, *p, _MM_CMPINT_LE);
    cond_sh = _mm512_mask_cmp_ps_mask(cond_pm, sh, z, _MM_CMPINT_LT);
    cond_st = _mm512_mask_cmp_ps_mask(cond_sh, st, z, _MM_CMPINT_LT);
    cond_s = _mm512_mask_cmp_ps_mask(~cond_pm, s, z, _MM_CMPINT_LT);

    *d = _mm512_mask_mov_ps(*d, cond_st, MUL(*d, POW(pms, SETI(1.0 / GAMA))));
    *d = _mm512_mask_mov_ps(*d, cond_s, MUL(*d, DIV(ADD(pms, g6), FMADD(pms, g6, one))));
    *u = _mm512_mask_mov_ps(*u, cond_st | cond_s, ums);
    *p = _mm512_mask_mov_ps(*p, cond_st | cond_s, pm);

    cond_sh_st = cond_sh & ~cond_st;
    if (cond_sh_st != 0xC)
    {
        *u = _mm512_mask_mov_ps(*u, cond_sh_st, MUL(g5, FMADD(g7, *u, c)));
        uc = DIV(*u, c);
        *d = _mm512_mask_mov_ps(*d, cond_sh_st, MUL(*d, POW(uc, g4)));
        *p = _mm512_mask_mov_ps(*p, cond_sh_st, MUL(*p, POW(uc, g3)));
    }

    *u = _mm512_mask_sub_ps(*u, cond_um, z, *u);
}
```

ОРГАНИЗАЦИЯ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ НА СИСТЕМЕ С ОБЩЕЙ ПАМЯТЬЮ

В числе основных функций в составе римановского решателя, используемого в роли тестовой задачи, можно отметить следующие:

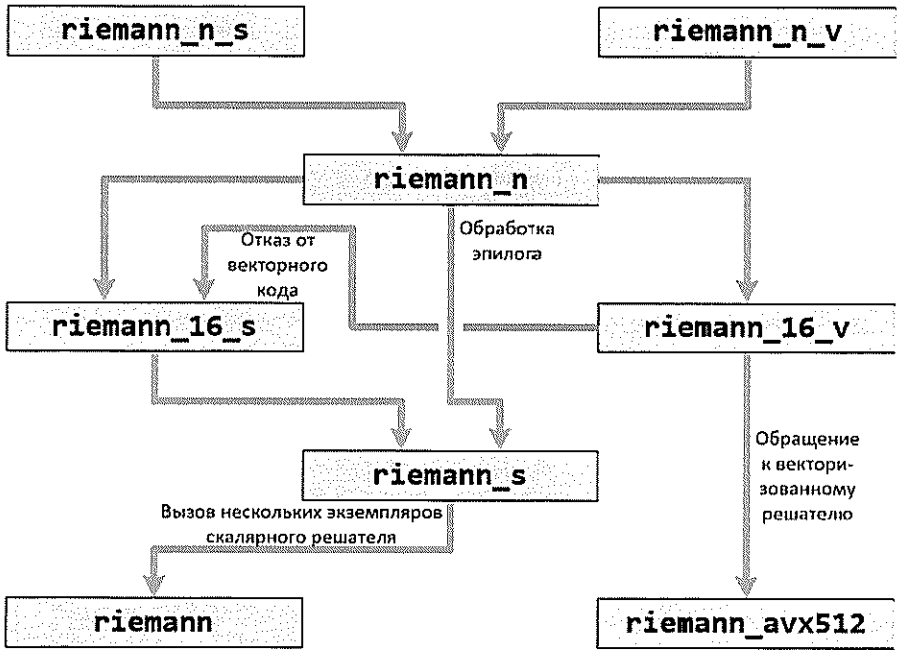

Рис. 1. Схема взаимодействия функций в реализации римановского решателя с поддержкой скалярной и векторизованной версий.

- `riemann` – оригинальная реализация решателя для скалярного набора входных данных.
- `riemann_s` – скалярная реализация для решения нескольких экземпляров задачи Римана (представляет собой цикл с вызовом функции `riemann`).
- `riemann_avx512` – векторная реализация с использованием инструкций AVX-512 решения 16 экземпляров задачи Римана.
- `riemann_16_s` – скалярная реализация решения 16 экземпляров задачи (вызов функции `riemann_s`).
- `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.

Для организации распараллеливания вычислений для систем с общей памятью с помощью OpenMP в тестовой задаче реализованы три стратегии разделения массивов входных данных между разными потоками.

В качестве первой стратегии распараллеливания используется стратегия CHUNKS, в которой массивы входных данных разделены на равные непрерывные части по числу используемых потоков, каждый поток обрабатывает 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 разделены на участки по 16 элементов и распределяются между потоками поочередно (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 предпринимается 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.

На рис. 2 представлена иллюстрация распределения входных данных согласно стратегиям CHUNKS, INTERLEAVE, RACE.

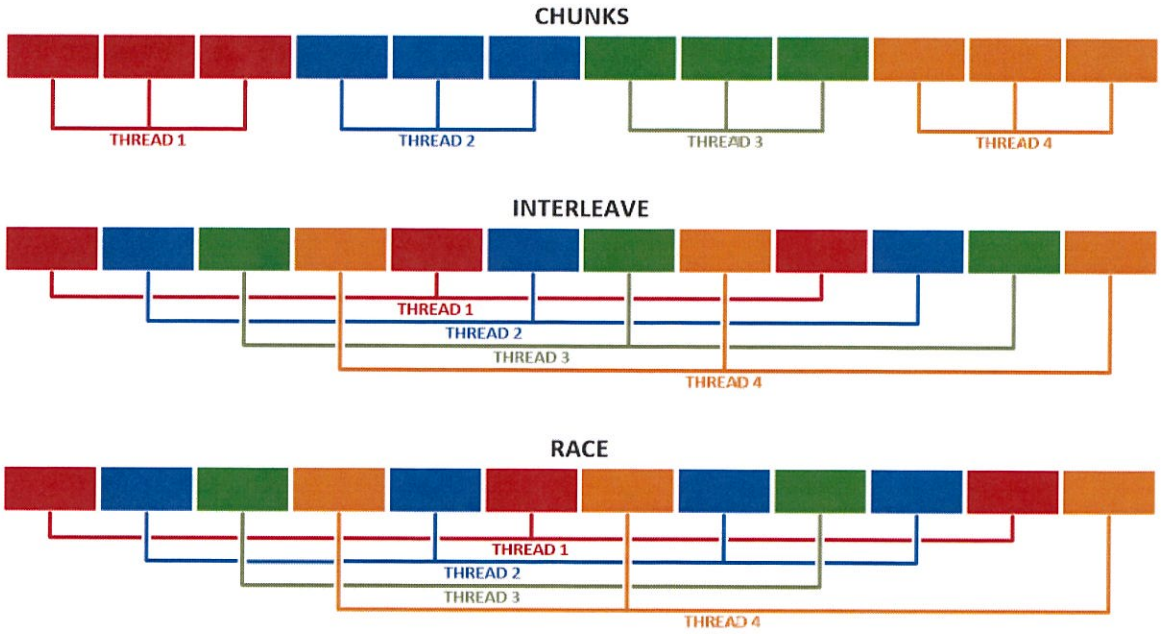

Рис. 2. Иллюстрация работы трех рассматриваемых стратегий распараллеливания вычислений: CHUNKS, INTERLEAVE, RACE.

ОПИСАНИЕ ПРОЦЕДУРЫ ИЗМЕРЕНИЙ

1. Получение исходных кодов тестовой задачи.
   Исходные коды тестовой задачи о распаде произвольного разрыва могут быть загружены из сети Интернет из репозитория github.com по адресу https //github.com/r-aax/riemann_vec/releases/tag/metric
   Исходные коды могут быть загружены в виде архива в формате *.zip или *.tar.gz.

2. Компиляция тестовой задачи.
   Для сборки тестовой задачи должен быть использован компилятор icc.
   В тексте программы пути к заголовочным файлам указываются относительно папки `src` (в сборке нужно указать опцию `-I src`).
   В сборке должны участвовать исходные тексты тестовой задачи, находящиеся в папках `test` и `src` (файлы для сборки должны быть описаны в виде `test/*.cpp src/*.cpp`).
   Для подключения функций-интринсиков для поддержки операций из набора инструкций AVX-512 требуется подать на компиляцию опцию `-DINTEL` (в противном случае векторный код создаваться не будет).
   Для разрешения компилятору создавать инструкции AVX-512 может потребоваться задание специальных опций (например, `-xmic-avx512`).
   Выбор конкретной стратегии распараллеливания с помощью OpenMP включается специальными опциями: опция `-DOPENMP_CHUNKS` включает стратегию CHUNKS, опция `-DOPENMP_INTERLEAVE` включает стратегию INTERLEAVE, опция `-DOPENMP_RACE` включает стратегию RACE.
   Для отладочных целей может быть подана опция `-DTEST_MODE=0`, которая подключает к тесту набор тестовых данных малого размера. Для проведения измерений следует использовать опцию `-DTEST_MODE=1`, по которой будет подключен набор входных данных большого размера.
   Опция `-DREPEATS=<N>` включает многократный замер времени прохождения теста (результатом замера будет считаться минимальное время). Рекомендуемое значение для данной опции – 3.
   Опция `-DINNER_REPEATS=<N>` сигнализирует, что тестовый набор входных данных должен быть обработан N раз для уменьшения флуктуаций времени выполнения теста. Рекомендуемое значение для данной опции – 100.

В сборку должны быть включены математическая библиотека и поддержка OpenMP (должны быть поданы флаги `-lm -fopenmp`).
Пример строки компиляции тестовой задачи для вычислительного узла на базе микропроцессора Intel Xeon Phi Knights Landing 7290 с использованием стратегии распараллеливания INTERLEAVE:

```
icc \
    -I src \
    test/*.cpp src/*.cpp \
    -DINTEL \
    -O3 \
    -xmic-avx512 \
    -DOPENMP_INTERLEAVE \
    -DTEST_MODE=1 \
    -DREPEATS=3 \
    -DINNER_REPEATS=100 \
    -o riemann.out \
    -lm -fopenmp
```

3. Запуск тестовой задачи.
Тестовая задача запускается с единственным аргументом – максимальным количеством потоков, для которого должны быть проведены замеры.
Пример строки запуска тестовой задачи для вычислительного узла на базе микропроцессора Intel Xeon Phi Knights Landing 7290 (данный микропроцессор имеет 72 ядра и 288 потоков):

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

4. Анализ выдачи исполняемого файла тестовой задачи.
Во время работы задачи на экран выдается текстовая информация (ниже в качестве примера приводятся выдержки из файла выдачи тестовой задачи, собранной с помощью строкой компиляции из п. 2).
В начале работы программы на экран выдается информация о количестве тестовых наборов входных данных:

```
test begin : 419996 test cases
```

Следующей строкой выдается информация, для какого количества потоков выполняются замеры (в строке ниже указано, что замеры проводятся для количества потоков от 1 до 288), также в строке указывается название стратегии распараллеливания с помощью OpenMP:

```
num_threads = 1 vs 288 (OPENMP_INTERLEAVE)
```

Вначале выполняются запуски скалярной версии тестовой программы на одном потоке:

```
n_s ~ 48.8827 seconds
n_s ~ 48.6517 seconds
n_s ~ 48.8367 seconds
```

После этого для каждого проверяемого количества потоков выполняются запуски, собирается и выводится на экран информация следующего вида: `nt` – количество потоков, `min_time` – минимальное время работы скалярной версии программы на одном потоке, `min_time_opt` – минимальное время работы векторизованной многопоточной версии программы, `time_reduce` – сокращение времени работы программы при переходе от скалярной однопоточной версии к векторизованной многопоточной версии, `speedup_x` – ускорение работы программы при переходе от скалярной однопоточной версии к векторизованной многопоточной версии.

```
                n_v ~ 7.40402 seconds
                n_v ~ 7.39298 seconds
                n_v ~ 7.31757 seconds
   test done : nt = 1,
                min_time = 48.6517,
                min_time_opt = 7.31757,
                time_reduce = 85%,
                speedup_x = 6.65
.....................................................
                n_v ~ 0.137 seconds
                n_v ~ 0.135 seconds
                n_v ~ 0.134 seconds
   test done : nt = 138,
                min_time = 48.7,
                min_time_opt = 0.134,
                time_reduce = 1e+02%,
                speedup_x = 363
   ----------
                n_v ~ 0.137 seconds
                n_v ~ 0.136 seconds
                n_v ~ 0.132 seconds
   test done : nt = 139,
                min_time = 48.7,
                min_time_opt = 0.132,
                time_reduce = 1e+02%,
                speedup_x = 368
```

5. Проведение анализа полученных результатов.

   Значение ускорения (`speedup_x`) полученное на одном потоке (`nt = 1`) соответствует *ускорению задачи, достигнутому исключительно за счет векторизации кода* (для рассматриваемого примера в печати в п. 4 видно, что данное значение равно 6,65).

   По значениям `speedup_x`, собранным для всех значений количества потоков может быть построен график *суммарного ускорения, достигнутого вследствие использования векторизации кода и распараллеливания с помощью OpenMP* при указанной в сборке стратегии распараллеливания.

   Разделив собранные значения суммарного ускорения на значение `speedup_x` при `nt = 1` можно получить отдельно *ускорение, достигнутое исключительно вследствие применения распараллеливания с помощью OpenMP* при указанной в сборке стратегии распараллеливания.

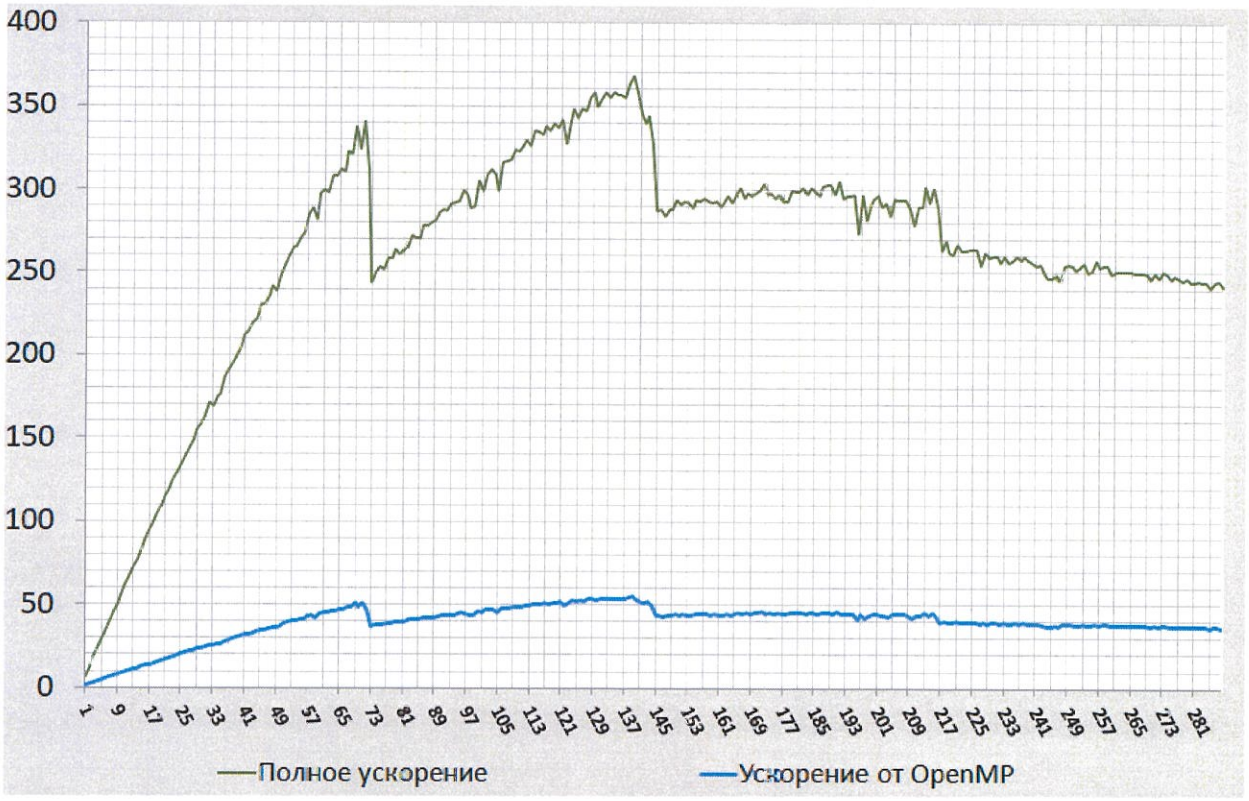

Рис. 3. Пример построения графиков полного ускорения и ускорения только с учетом OpenMP для вычислительного узла на базе микропроцессора Intel Xeon Phi Knights Landing 7290 для тестовой задачи, собранной с помощью строки компиляции из п. 2.

ЗАКЛЮЧЕНИЕ

В документе приведено описание следующих методик измерения эффективности векторизации и масштабирования высоконагруженных вычислений для систем с общей памятью:

- методика измерения эффективности векторизации сложного программного контекста с помощью специального набора инструкций Intel AVX-512;
- методика измерения эффективности распараллеливания вычислений с помощью OpenMP при разделении массивов входных данных между потоками на сплошные области;
- методика измерения эффективности распараллеливания вычислений с помощью OpenMP при разделении массивов входных данных между потоками с помощью чередования фрагментов;
- методика измерения эффективности распараллеливания вычислений с помощью OpenMP при динамическом разделении массивов входных данных между потоками и наличии конфликта по глобальному ресурсу;
- комплексная методика измерения эффективности оптимизации сложного программного контекста, достигаемой вследствие применения векторизации кода и распараллеливания с помощью OpenMP с фиксированной стратегией распараллеливания.

Применение рассмотренных методик позволяет оценить вычислительную систему с точки зрения ее пригодности для выполнения параллельных вычислений для задач, содержащих плотный векторизованный исполняемый код.

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, с. 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, с. 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, с. 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, с. 367-377.
7. https://github.com/r-aax/riemann_vec/releases/tag/metric
8. NUMERICA, A Library of Sources for Teaching, Research and Applications, by E. F. Toro. // https://github.com/dasikasunder/NUMERICA
9. E. F. Toro. Riemann solvers and numerical methods for fluid dynamics. // A practical introduction. 2nd edition, Springer, 1999.
10. A. A. Rybakov, S. S. Shumilin. Vectorization of the Riemann solver using the AVX-512 instruction set. // Program Systems: Theory and Applications, 2019, Vol. 10, № 3 (42), p. 41-58.
11. А. А. Рыбаков, С. С. Шумилин. Векторизация римановского решателя с использованием набора инструкций AVX-512. // Программные системы: Теория и приложения, 2019, Т. 10, № 3 (42), с. 59-80.
12. А. А. Рыбаков. Векторизация нахождения пересечения объемной и поверхностной сеток для микропроцессоров с поддержкой AVX-512. // Труды НИИСИ РАН, 2019, Т. 9, № 5, с. 5-14.
13. А. А. Рыбаков, С. С. Шумилин. Исследование эффективности векторизa-

ции гнезд циклов с нерегулярным числом итераций. // Программные системы: Теория и алгоритмы, 2019, Т. 10, № 4 (43), с. 77-96.

14. 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, p. 580-598.
15. Л. А. Бендерский, А. А. Рыбаков, С. С. Шумилин. Векторизация перемножения малоразмерных матриц специального вида с использованием инструкций AVX-512. // Международный научный журнал «Современные информационные технологии и ИТ-образование», 2018, Т. 14, № 3, с. 594-602.
16. А. А. Рыбаков, С. С. Шумилин. Векторизация сильно разветвленного управления с помощью инструкций AVX-512. // Труды НИИСИ РАН, 2018, Т. 8, № 4, с. 114-126.
17. А. А. Рыбаков, П. Н. Телегин, Б. М. Шабанов. Проблемы векторизации гнезд циклов с использованием инструкций AVX-512. // Электронный научный журнал: Программные продукты, системы и алгоритмы, 2018, № 3, с. 1-11.
18. Л. А. Бендерский, С. А. Лещев, А. А. Рыбаков. Векторизация операций над матрицами малой размерности для процессора Intel Xeon Phi Knights Landing. // Международный научный журнал «Современные информационные технологии и ИТ-образование», 2018, Т. 14, № 1, с. 73-90.