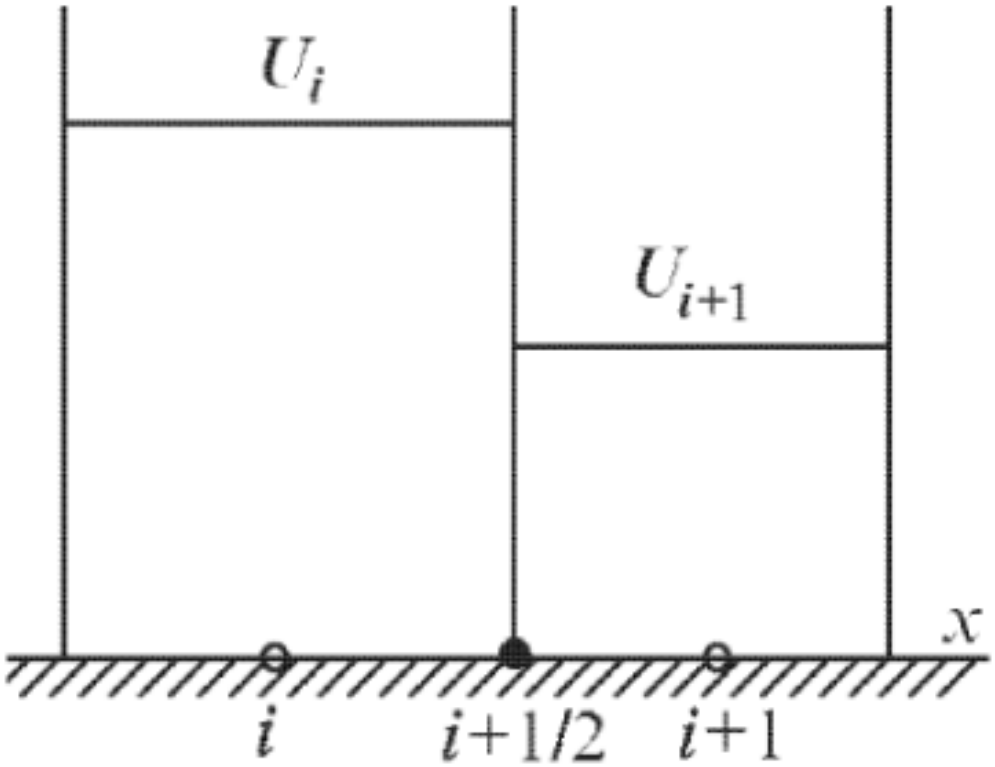
Межведомственный суперкомпьютерный центр Российской академии наук – филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН)

Рыбаков Алексей Анатольевич, Шумилин Сергей Сергеевич

Векторизация римановского решателя с использованием набора инструкций AVX-512

Национальный Суперкомпьютерный Форум (НСКФ) 2018
27-30.11.2018, Институт программных систем имени А. К. Айламазяна Российской академии наук, Переславль-Залесский, Россия

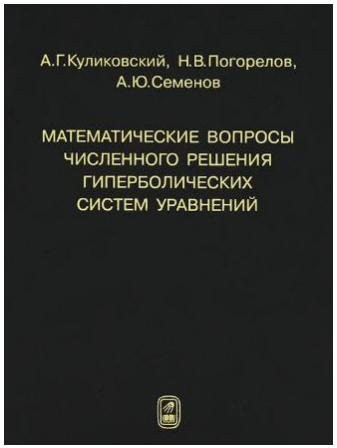
Метод Годунова
численного решения задач газовой динамики



$$\frac{\mathbf{U}_i^{k+1} - \mathbf{U}_i^k}{\Delta t} + \frac{\mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2}}{\Delta x} = \mathbf{0}$$

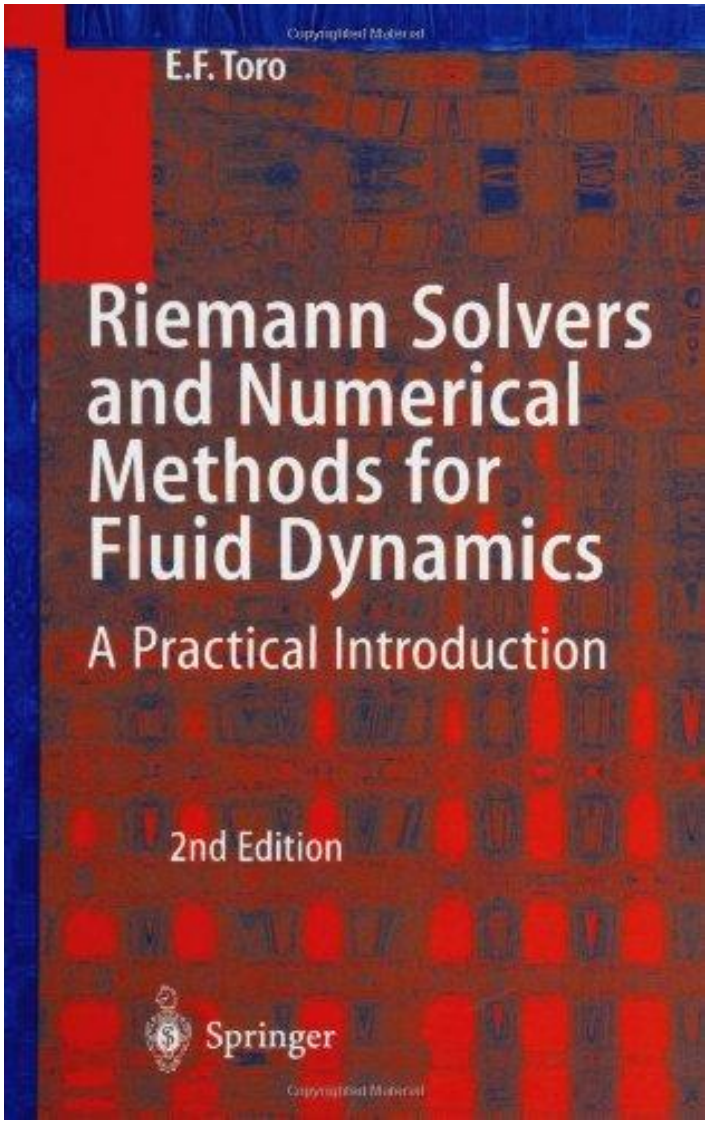
$$\mathbf{F}_{i\pm1/2} = \mathbf{F}(\mathbf{U}_{i\pm1/2})$$

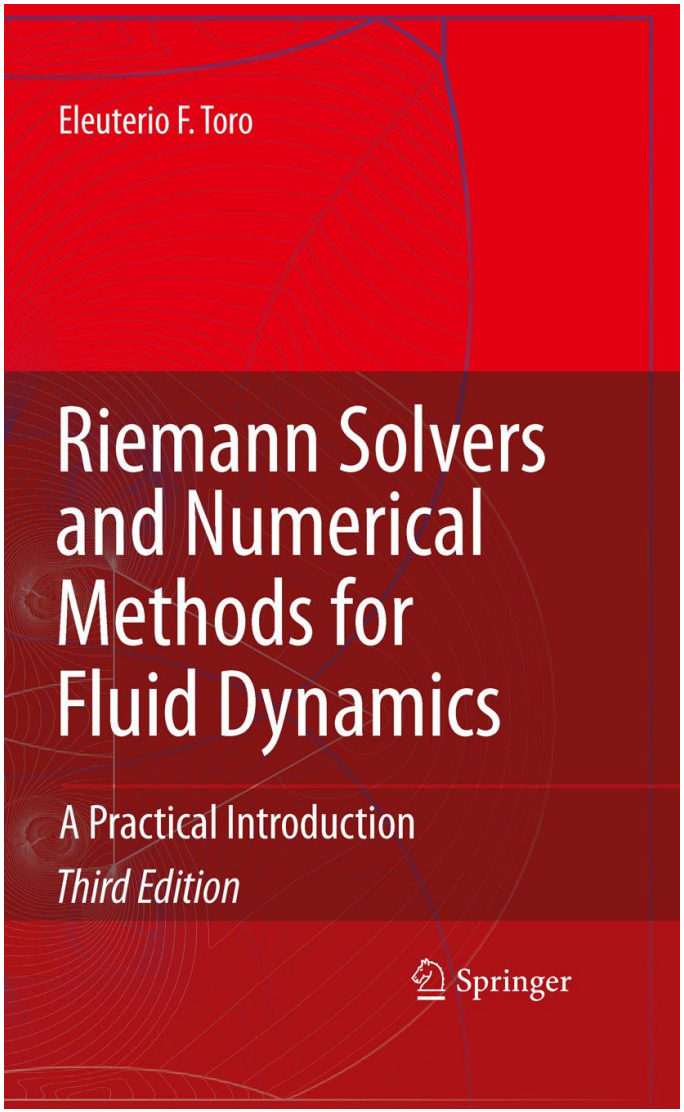
$$C = \max_p |C_p| \le 1, \quad C_p = \lambda_p \frac{\Delta t}{\Delta x}$$



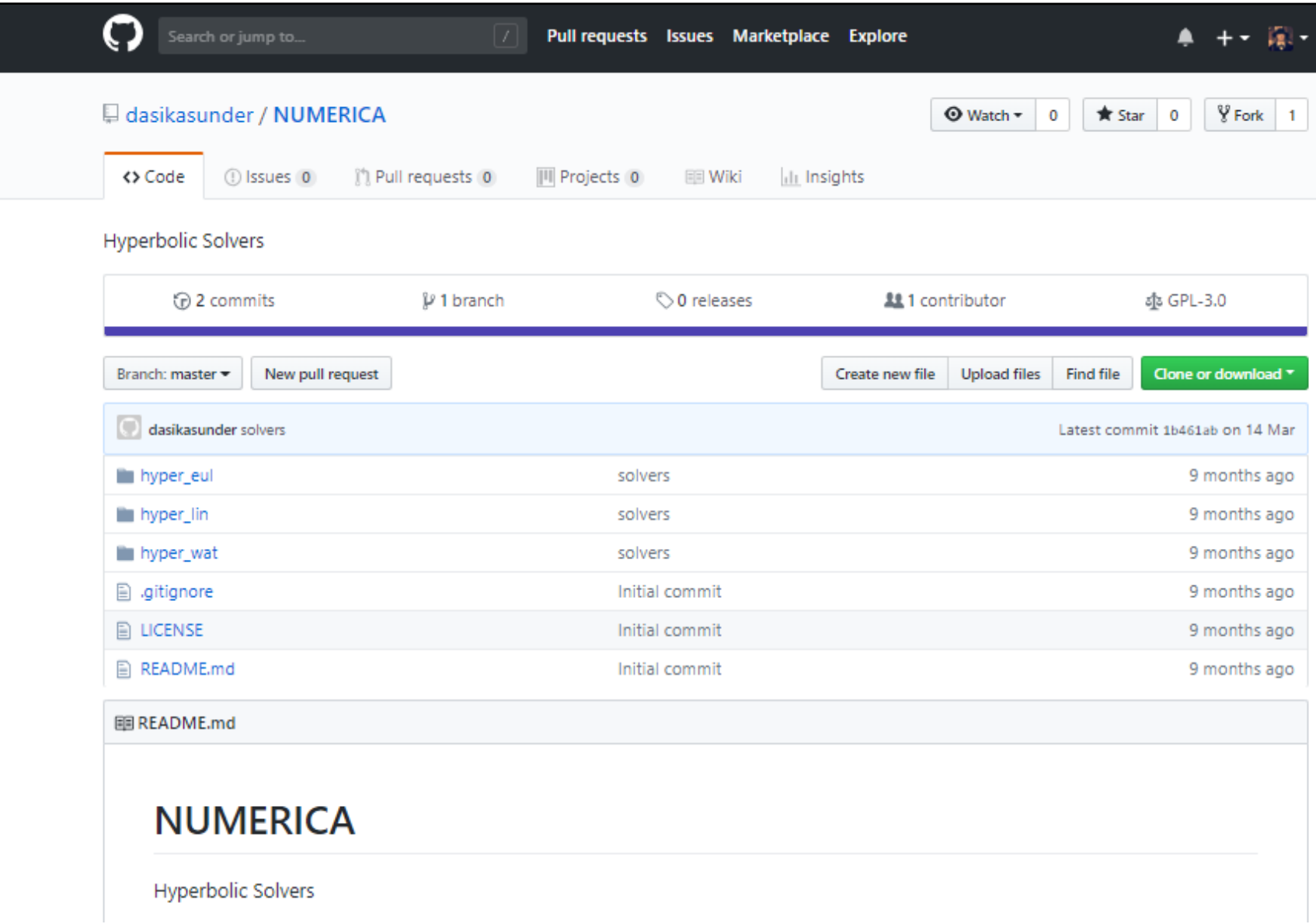
Куликовский А. Г.; Погорелов Н. В.; Семенов А. Ю. – *Математические вопросы численного решения гиперболических систем уравнений* (2001)

Теоретическое описание и практическая реализация римановского решателя

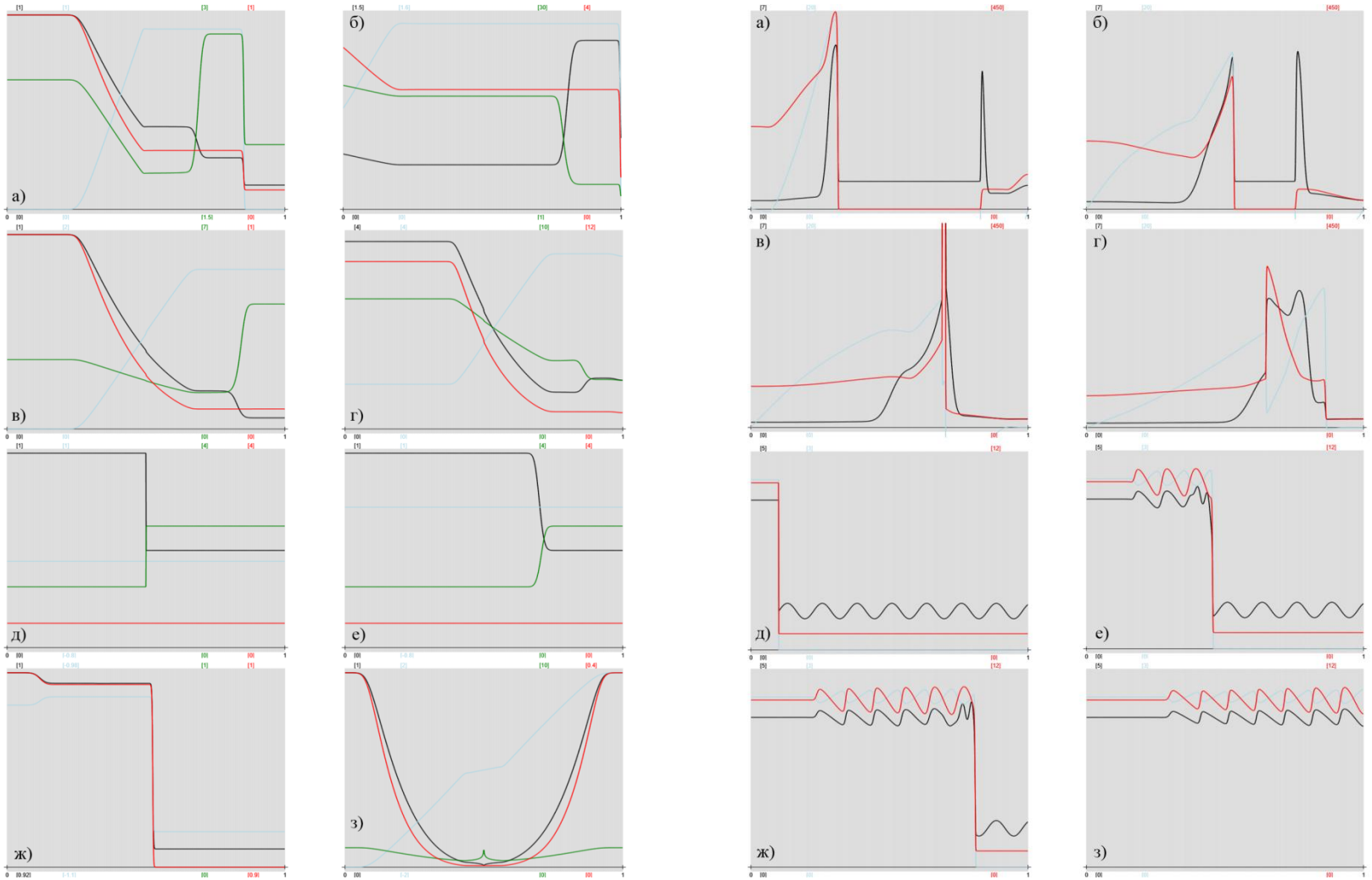




Библиотека NUMERICA
программная реализация расчетных кодов



Характерные задачи, использовавшиеся для сбора профиля римановского решателя



Задача Сода, задача Лакса, задача о неподвижном контактном разрыве, задача о слабой ударной волне, задача Эйнфельдта, задача Вудворда-Колелла, задача Шу-Ошера...

Схема обработки массивов данных римановским решателем

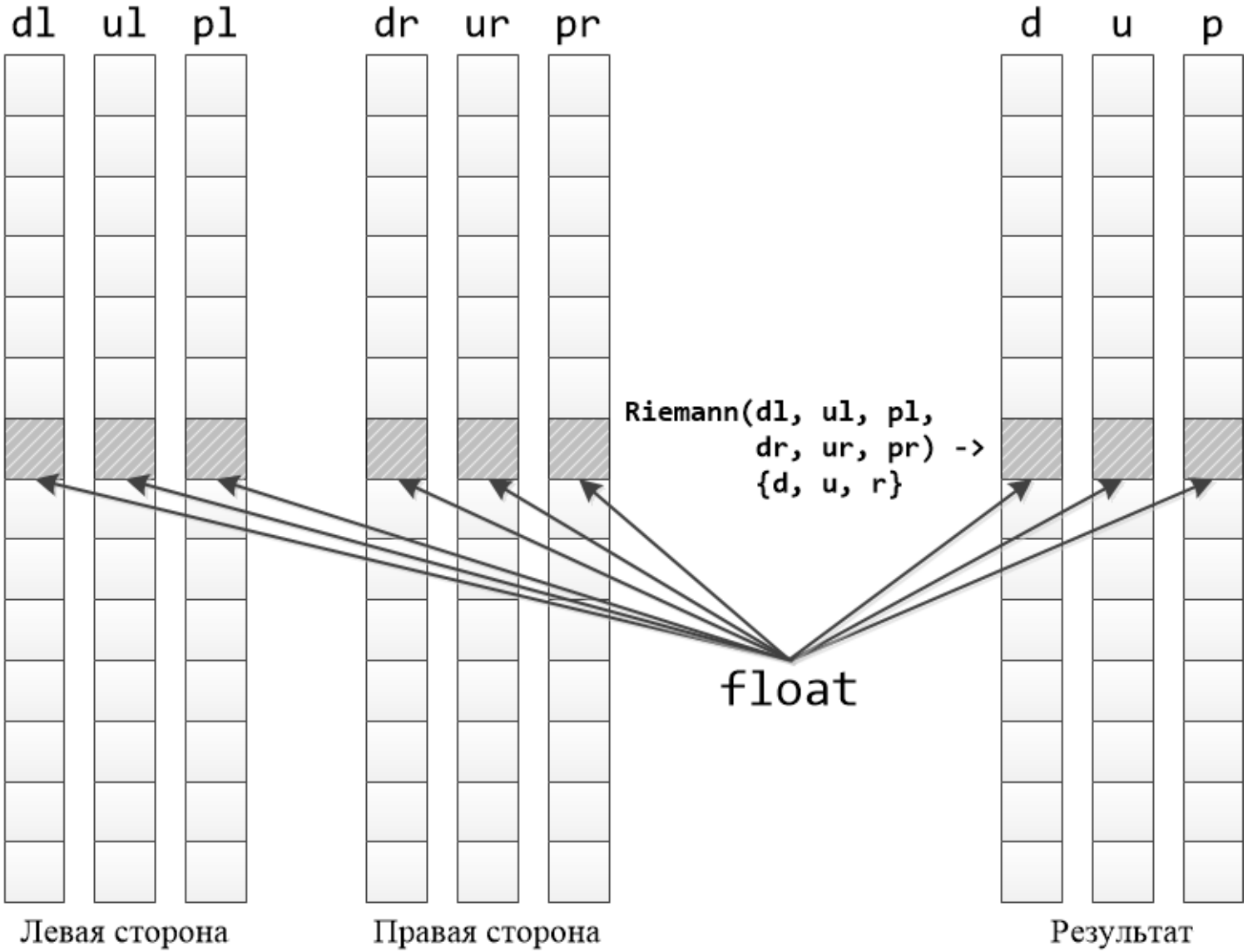
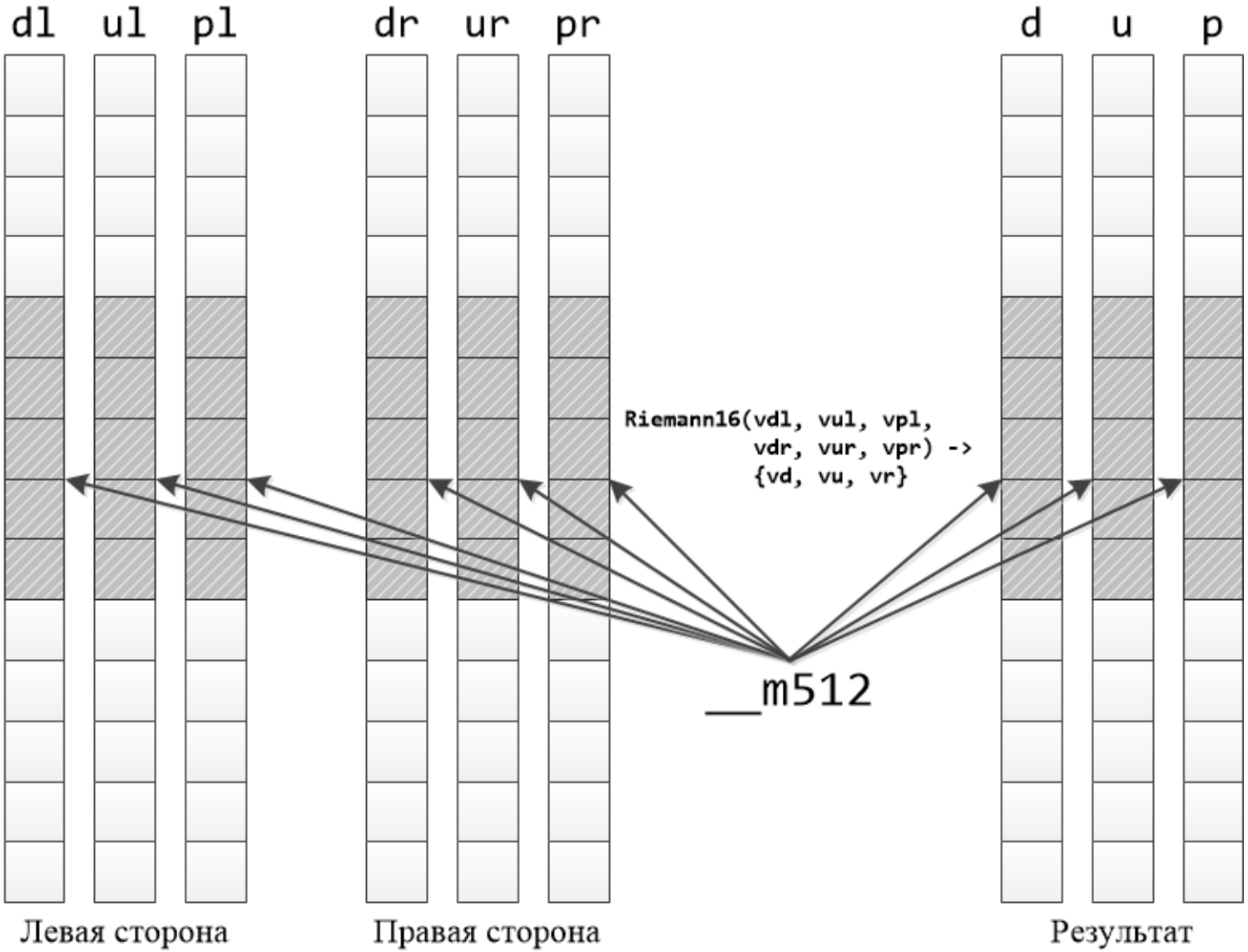
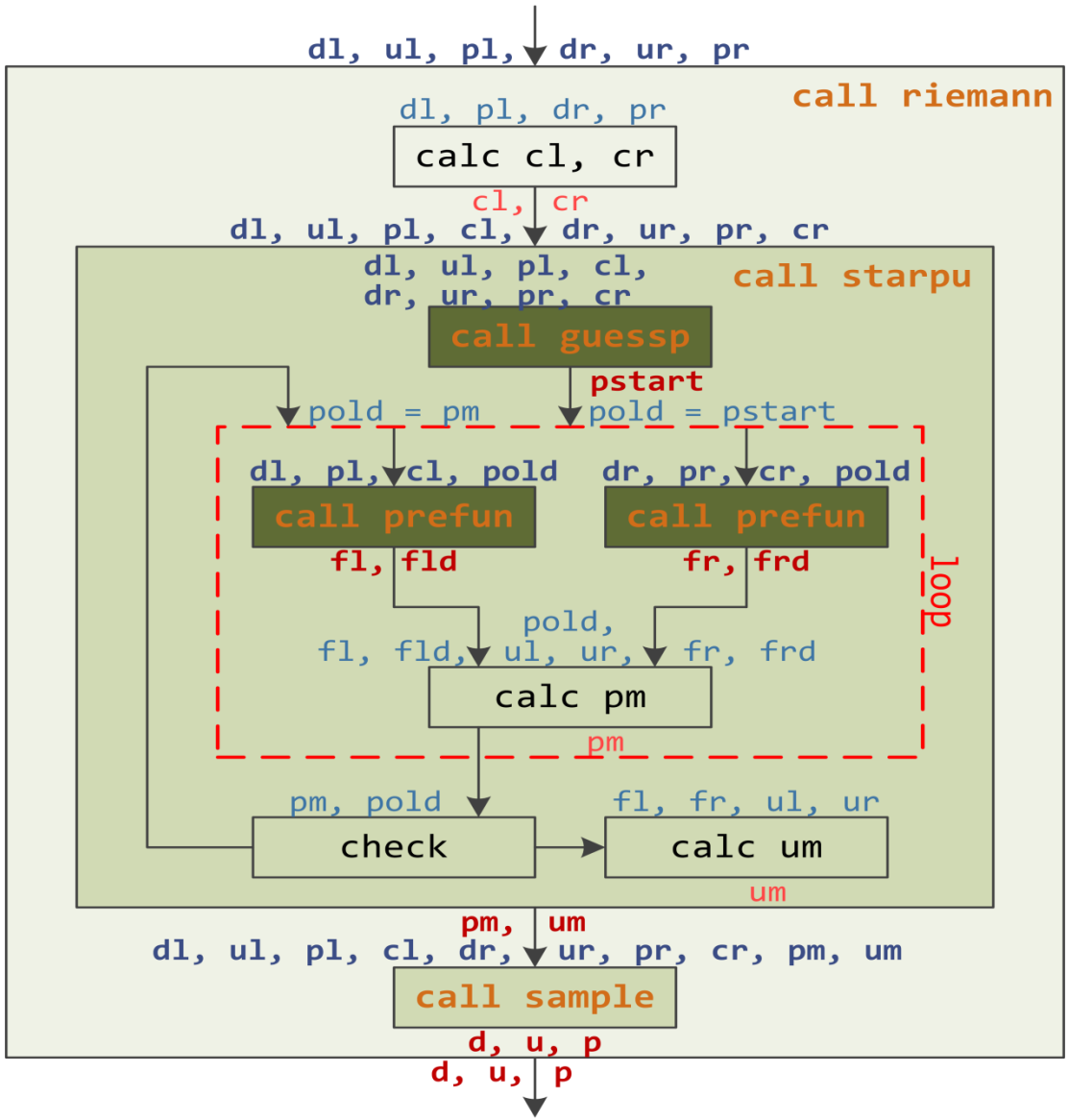


Схема обработки массивов данных Векторизованным римановским решателем



Общая схема римановского решателя



- `prefun` – функция, содержащая одно условие,
- `guessp` – функция с несколькими условиями,
- `sample` – функция с множественными вложенными условиями,
- `starpu` – функция, содержащая цикл с неизвестным числом итераций и вызовами других функций.

Подходы к векторизации плоского цикла с одним if-else условием с маловероятной else веткой

a)

```
for (i = 0; i < w; i++)
{
    if (<cond_i>)
    {
        // prob. p, len. t1
        <block_1>;
    }
    else
    {
        // prob. 1-p, len. t2
        <block_2>;
    }
}
```

b)

```
<vblock_1> ? <vcond>;

if (~<vcond>)
{
    for (i = 0; i < w; i++)
    {
        if (!<cond_i>)
        {
            <block_2>;
        }
    }
}
```

vectorization

vectorized condition

c)

```
<vblock_1> ? <vcond>;
<vblock_2> ? ~<vcond>;
```

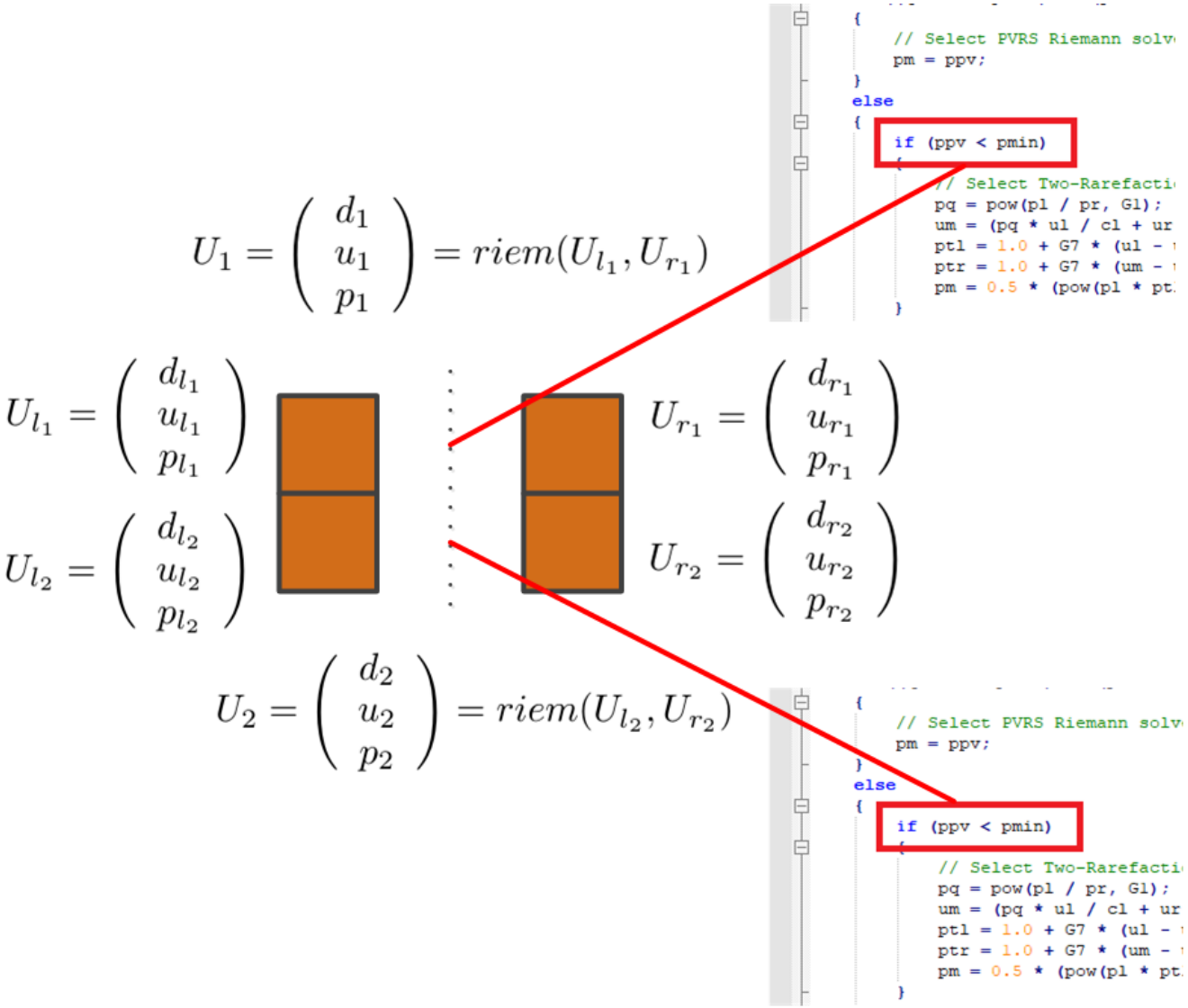
d)

```
<vblock_1> ? <vcond>;

if (~<vcond>)
{
    <vblock_2> ? ~<vcond>;
}
```

if-else statement	operations count
a)	$(pt_1 + (1-p)t_2)\,w$
b)	$t_1 + (1-p)t_2 w$
c)	$t_1 + t_2$
d)	$t_1 + (1-p^w)t_2$

Непрерывность потоков данных в физических задачах обработка близких пар ячеек имеет схожую структуру

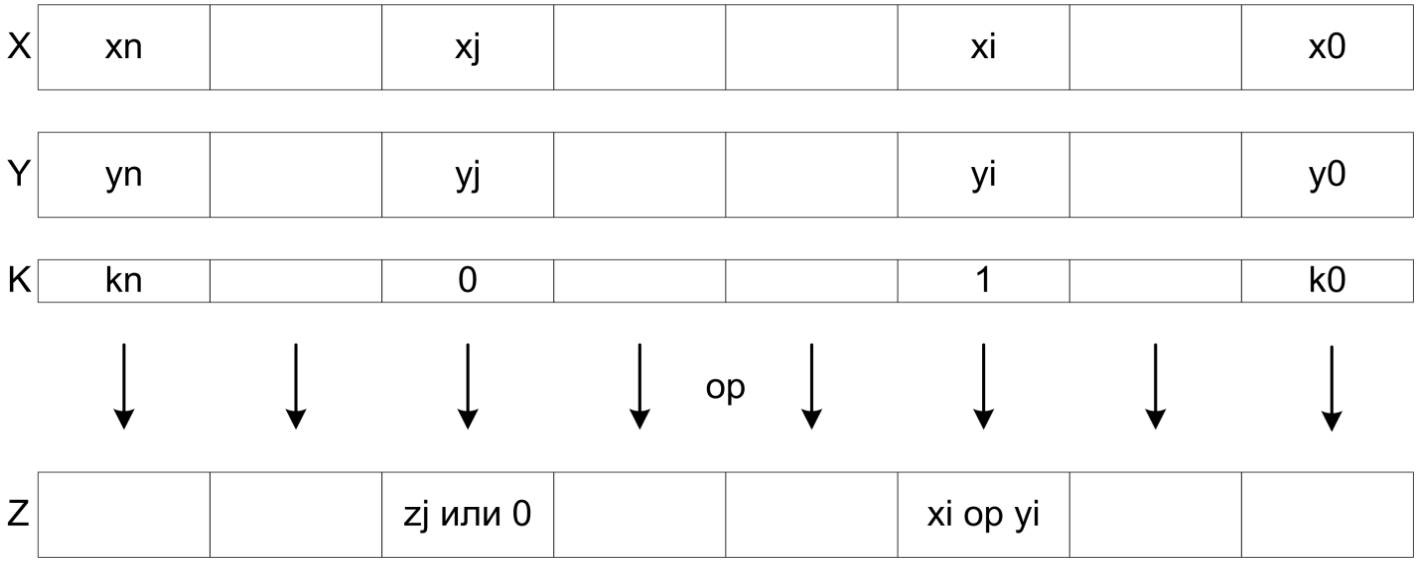


Оригинальная версия функции prefun

```
01  void prefun(float &f, float &fd, float &p,
02              float &dk, float &pk, float &ck)
03  {
04      float ak, bk, pratio, qrt;
05
06      if (p <= pk)
07      {
08          pratio = p / pk;
09          f = G4 * ck * (pow(pratio, G1) - 1.0);
10          fd = (1.0 / (dk * ck)) * pow(pratio, -G2);
11      }
12      else
13      {
14          ak = G5 / dk;
15          bk = G6 * pk;
16          qrt = sqrt(ak / (bk + p));
17          f = (p - pk) * qrt;
18          fd = (1.0 - 0.5 * (p - pk) / (bk + p)) * qrt;
19      }
20  }
```

Простые арифметические операции, использующиеся в векторизации римановского решателя

`_mm512_add_ps`	`_mm512_mask_add_ps`
`_mm512_sub_ps`	`_mm512_mask_sub_ps`
`_mm512_mul_ps`	`_mm512_mask_mul_ps`
`_mm512_div_ps`	`_mm512_mask_div_ps`
`_mm512_pow_ps`	`_mm512_mask_pow_ps`
`_mm512_sqrt_ps`	`_mm512_mask_sqrt_ps`



Векторизованная версия функции prefun

```
01  void prefun_16(__m512 *f, __m512 *fd, __m512 p,
02                 __m512 dk, __m512 pk, __m512 ck,
03                 __mmask16 m)
04  {
05      __m512 pratio, ak, bkp, ppk, qrt;
06      __mmask16 cond, ncond;
07
08      // Conditions.
09      cond = _mm512_mask_cmp_ps_mask(m, p, pk, _MM_CMPINT_LE);
10      ncond = m & ~cond;
11
12      // The first branch.
13      if (cond != 0x0)
14      {
15          pratio = _mm512_mask_div_ps(z, cond, p, pk);
16          *f = _mm512_mask_mul_ps(*f, cond, MUL(g4, ck),
17                   SUB(_mm512_mask_pow_ps(z, cond, pratio, g1), one));
18          *fd = _mm512_mask_div_ps(*fd, cond,
19                    _mm512_mask_pow_ps(z, cond, pratio, SUB(z, g2)),
20                    MUL(dk, ck));
21      }
22
23      // The second branch.
24      if (ncond != 0x0)
25      {
26          ak = _mm512_mask_div_ps(z, ncond, g5, dk);
27          bkp = FMADD(g6, pk, p);
28          ppk = SUB(p, pk);
29          qrt = _mm512_mask_sqrt_ps(z, ncond,
30                                    _mm512_mask_div_ps(z, ncond, ak, bkp));
31          *f = _mm512_mask_mul_ps(*f, ncond, ppk, qrt);
32          *fd = _mm512_mask_mul_ps(*fd, ncond, qrt,
33                    FNMADD(_mm512_mask_div_ps(z, ncond, ppk, bkp),
34                           SET1(0.5), one));
35      }
36  }
```

Оригинальная версия функции guessp

```
01 void guessp(float dl, float ul, float pl, float cl,
02              float dr, float ur, float pr, float cr,
03              float &pm)
04 {
05     float cup, gel, ger, pmax, pmin, ppv, pq, ptl, ptr, qmax, quser, um;
06
07     quser = 2.0;
08
09     cup = 0.25 * (dl + dr) * (cl + cr);
10     ppv = 0.5 * (pl + pr) + 0.5 * (ul - ur) * cup;
11     ppv = (ppv > 0.0) ? ppv : 0.0;
12     pmin = (pl < pr) ? pl : pr;
13     pmax = (pl > pr) ? pl : pr;
14     qmax = pmax / pmin;
15
16     if ((qmax <= quser) && (pmin <= ppv) && (ppv <= pmax))
17     {
18         pm = ppv;
19     }
20     else
21     {
22         if (ppv < pmin)
23         {
24             pq = pow(pl / pr, G1);
25             um = (pq * ul / cl + ur / cr + G4 * (pq - 1.0)) / (pq / cl + 1.0 / cr);
26             ptl = 1.0 + G7 * (ul - um) / cl;
27             ptr = 1.0 + G7 * (um - ur) / cr;
28             pm = 0.5 * (pow(pl * ptl, G3) + pow(pr * ptr, G3));
29         }
30         else
31         {
32             gel = sqrt((G5 / dl) / (G6 * pl + ppv));
33             ger = sqrt((G5 / dr) / (G6 * pr + ppv));
34             pm = (gel * pl + ger * pr - (ur - ul)) / (gel + ger);
35         }
36     }
37 }
```

Тайминги выполнения отдельных векторных операций для микропроцессора Intel Xeon Phi KNL

Arithmetic					
ADDSS SUBSS	x,x	1	FP0/1	6	0.5
ADDSD SUBSD	x,x	1	FP0/1	6	0.5
ADDPS SUBPS	x,x	1	FP0/1	6	0.5
VADDPS VSUBPS	v,v,v	1	FP0/1	6	0.5
ADDPD SUBPD	x,x	1	FP0/1	6	0.5
VADDPD VSUBPD	v,v,v	1	FP0/1	6	0.5
ADDSUBPS/D	x,x	1	FP0/1	6	0.5
VADDSUBPS/D	v,v,v	1	FP0/1	6	0.5
HADDPS/D HSUBPS/D	x,x	3		15	8
VHADDPS/D VHSUBPS/D	yy,y,	3		15	8
MULSS/D	x,x	1	FP0/1	6	0.5
MULPS/D	x,x	1	FP0/1	6	0.5
VMULPS/D	v,v,v	1	FP0/1	6	0.5
DIVSS	x,x	3	FP0	27	17
DIVSD	x,x	3	FP0	42	42
DIVPS	x,x	18	FP0	32	20
VDIVPS	v,v,v	18	FP0	32	32
DIVPD	x,x	18	FP0	32	20
VDIVPD	v,v,v	18	FP0	32	32

Оптимизация программного кода путем удаления избыточных операций деления

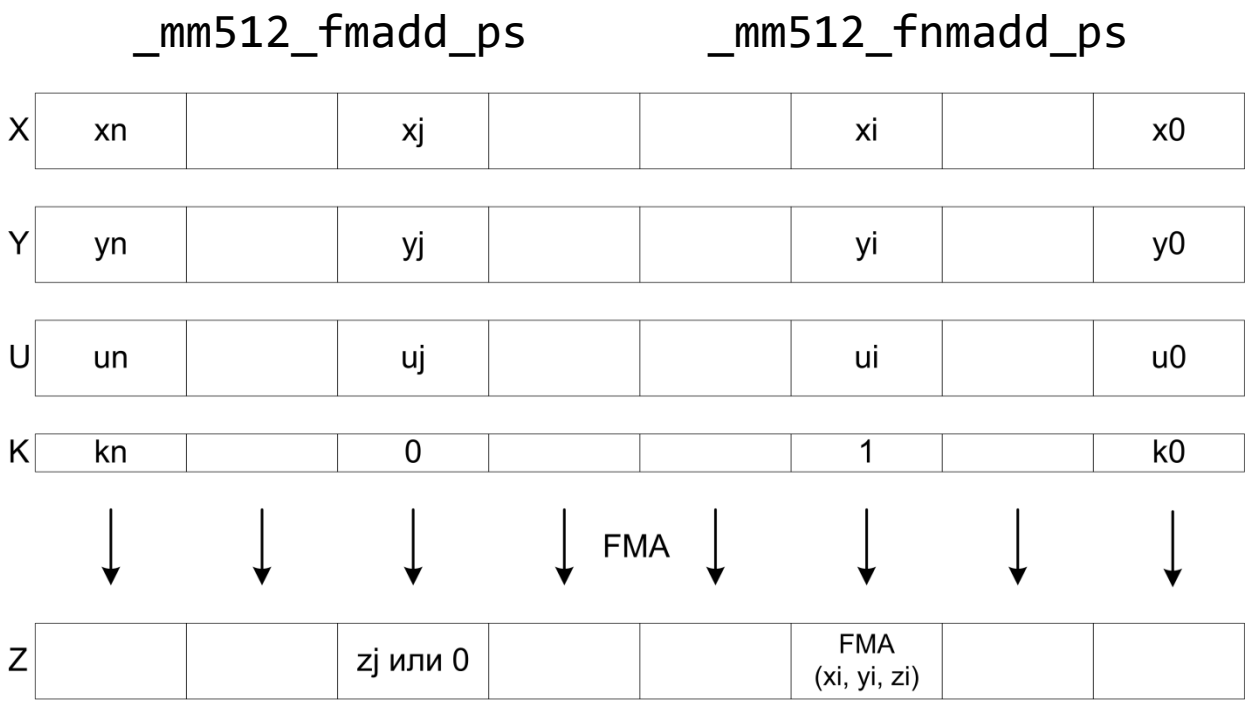
$$\frac{a}{\frac{b}{c}} = \frac{ac}{b}$$

$$\frac{\frac{a}{b}}{c} = \frac{a}{bc}$$

$$\frac{a}{b} \pm \frac{c}{d} = \frac{ad \pm bc}{bd}$$

$$\frac{\frac{a}{c} \pm x}{\frac{b}{c} \pm y} = \frac{a \pm cx}{b \pm cy}$$

Комбинированные арифметические операции для вычисления выражений вида ±a·b±c



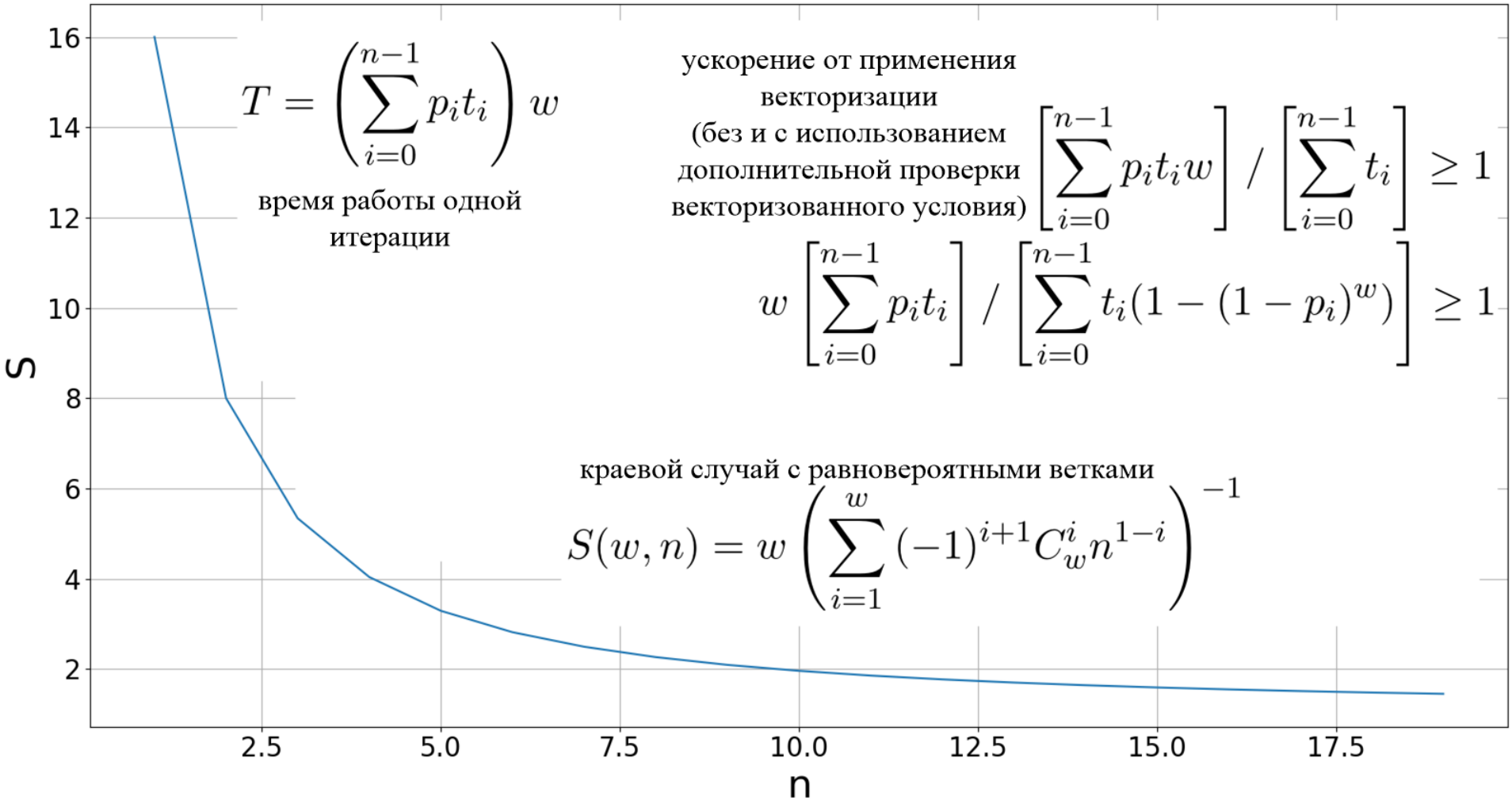
Другие используемые операции

```
_mm512_set1_ps
_mm512_cmp_ps_mask
_mm512_mask_cmp_ps_mask
_mm512_mask_mov_ps
```

Векторизованная версия функции guessp

```
01  void guessp_16(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
02                 __m512 dr, __m512 ur, __m512 pr, __m512 cr,
03                 __m512 *pm)
04  {
05      __m512 two, half, cup, ppv, pmin, pmax, qmax, pq, um, ptl, ptr, gel, ger, pqcr;
06      __mmask16 cond_pvrs, cond_ppv, ncond_ppv;
07
08      two = SET1(2.0);
09      half = SET1(0.5);
10      cup = MUL(SET1(0.25), MUL(ADD(dl, dr), ADD(cl, cr)));
11      ppv = MUL(half, FMADD(SUB(ul, ur), cup, ADD(pl, pr)));
12      ppv = MAX(ppv, z);
13      pmin = MIN(pl, pr);
14      pmax = MAX(pl, pr);
15      qmax = DIV(pmax, pmin);
16
17      // Conditions.
18      cond_pvrs = CMP(qmax, two, _MM_CMPINT_LE)
19                  && CMP(pmin, ppv, _MM_CMPINT_LE)
20                  && CMP(ppv, pmax, _MM_CMPINT_LE);
21      cond_ppv = _mm512_mask_cmp_ps_mask(~cond_pvrs, ppv, pmin, _MM_CMPINT_LT);
22      ncond_ppv = ~cond_pvrs & ~cond_ppv;
23
24      // The first branch.
25      *pm = _mm512_mask_mov_ps(*pm, cond_pvrs, ppv);
26
27      // The second branch.
28      if (cond_ppv != 0x0)
29      {
30          pq = _mm512_mask_pow_ps(z, cond_ppv,
31                                  _mm512_mask_div_ps(z, cond_ppv, pl, pr), g1);
32          pqcr = MUL(pq, cr);
33          um = _mm512_mask_div_ps(z, cond_ppv,
34                                  FMADD(FMADD(SUB(pqcr, cr), g4, ur), cl, MUL(pqcr, ul)),
35                                  ADD(pqcr, cl));
36          ptl = FMADD(_mm512_mask_div_ps(z, cond_ppv, SUB(ul, um), cl), g7, one);
37          ptr = FMADD(_mm512_mask_div_ps(z, cond_ppv, SUB(um, ur), cr), g7, one);
38          *pm = _mm512_mask_mul_ps(*pm, cond_ppv, half,
39                                   ADD(_mm512_mask_pow_ps(z, cond_ppv, MUL(pl, ptl), g3),
40                                       _mm512_mask_pow_ps(z, cond_ppv, MUL(pr, ptr), g3)));
41      }
42
43      // The third branch.
44      if (ncond_ppv != 0x0)
45      {
46          gel = SQRT(_mm512_mask_div_ps(z, ncond_ppv, g5, MUL(FMADD(g6, pl, ppv), dl)));
47          ger = SQRT(_mm512_mask_div_ps(z, ncond_ppv, g5, MUL(FMADD(g6, pr, ppv), dr)));
48          *pm = _mm512_mask_div_ps(*pm, ncond_ppv,
49                                   FMADD(gel, pl, FMADD(ger, pr, SUB(ul, ur))),
50                                   ADD(gel, ger));
51      }
52  }
```

Теоретическое ускорение при векторизации цикла со сложным управлением



$$T = \left(\sum_{i=0}^{n-1} p_i t_i \right) w$$

время работы одной итерации

ускорение от применения векторизации (без и с использованием дополнительной проверки векторизованного условия)

$$\left[\sum_{i=0}^{n-1} p_i t_i w \right] / \left[\sum_{i=0}^{n-1} t_i \right] \geq 1$$

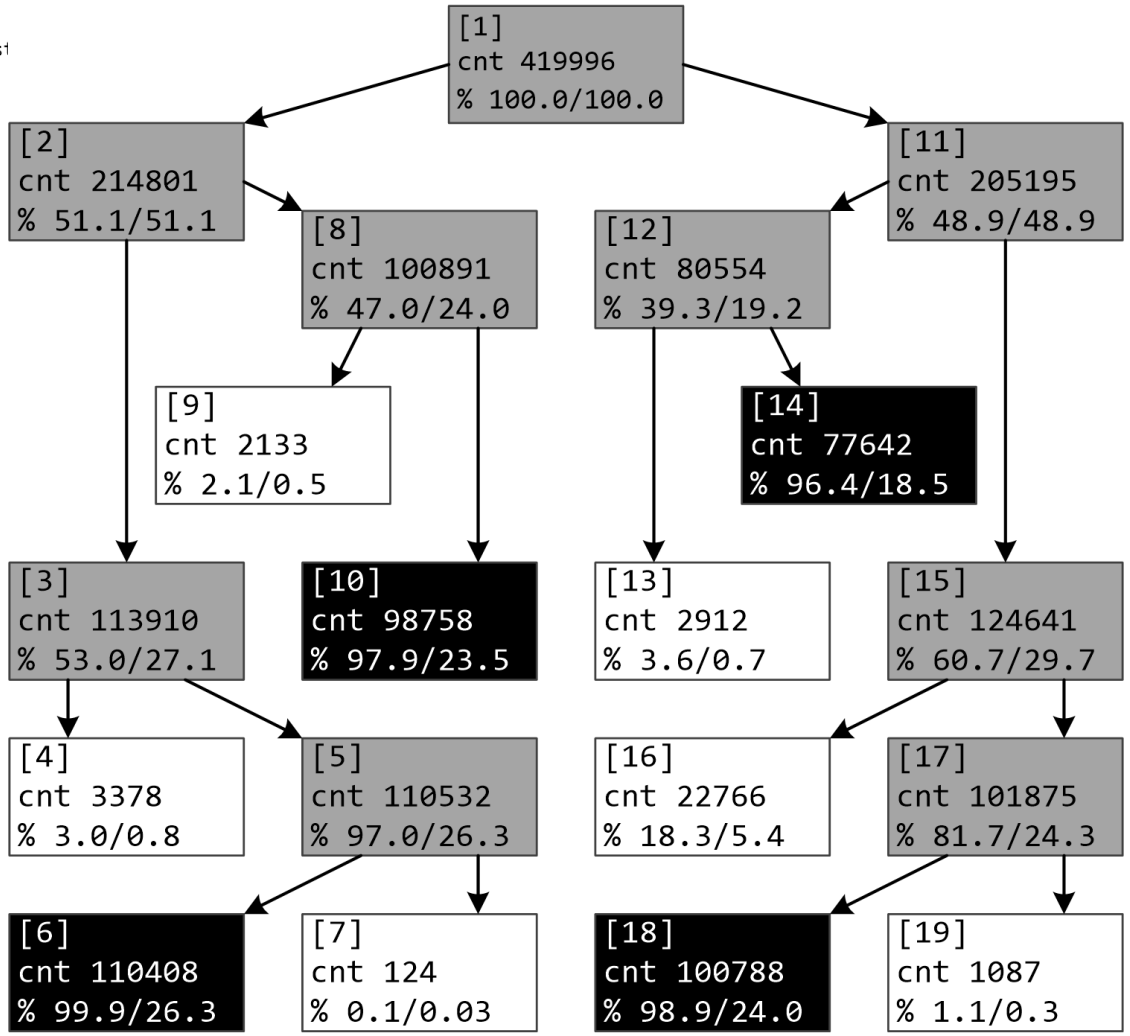
$$w \left[\sum_{i=0}^{n-1} p_i t_i \right] / \left[\sum_{i=0}^{n-1} t_i (1 - (1 - p_i)^w) \right] \geq 1$$

краевой случай с равновероятными ветками

$$S(w, n) = w \left(\sum_{i=1}^{w} (-1)^{i+1} C_w^i n^{1-i} \right)^{-1}$$

Оригинальная версия функции sample

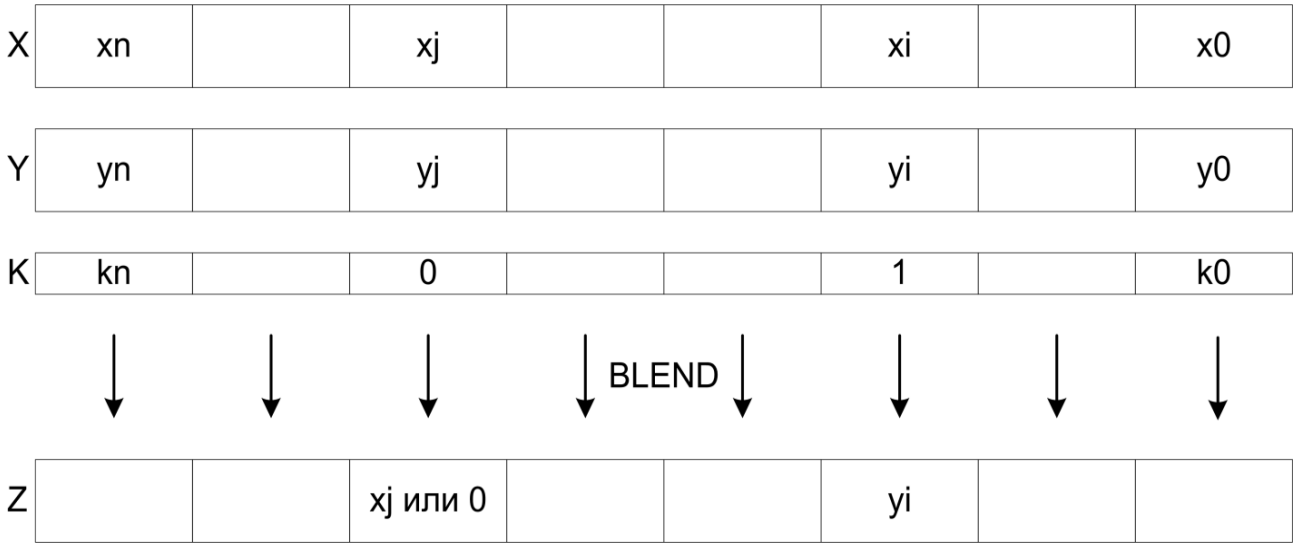
```
001 void sample(float dl, float ul, float pl, float cl,
002              float dr, float ur, float pr, float cr,
003              const float pm, const float um,
004              float &d, float &u, float &p)
005 {
006     float c, cml, cmr, pml, pmr, shl, shr, sl, sr, stl, st
007
008     if (0.0 <= um)
009     {
010         if (pm <= pl)
011         {
012             shl = ul - cl;
013
014             if (0.0 <= shl)
015             {
                    < d, u, p = dl, ul, pl >
019             }
020             else
021             {
022                 cml = cl * pow(pm / pl, G1);
023                 stl = um - cml;
024
025                 if (0.0 > stl)
026                 {
027                     d = dl * pow(pm / pl, 1.0 / GAMA);
028                     u = um;
029                     p = pm;
030                 }
031                 else
032                 {
                        < high-density code, low prob >
037                 }
038             }
039         }
040         else
041         {
042             pml = pm / pl;
043             sl = ul - cl * sqrt(G2 * pml + G1);
044
045             if (0.0 <= sl)
046             {
                    < d, u, p = dl, ul, pl >
050             }
051             else
052             {
053                 d = dl * (pml + G6) / (pml * G6 + 1.0);
054                 u = um;
055                 p = pm;
056             }
057         }
058     }
059     else
060     {
            < symmetrical branch >
109     }
110 }
```



Операции слияния двух векторных регистров с использованием маски

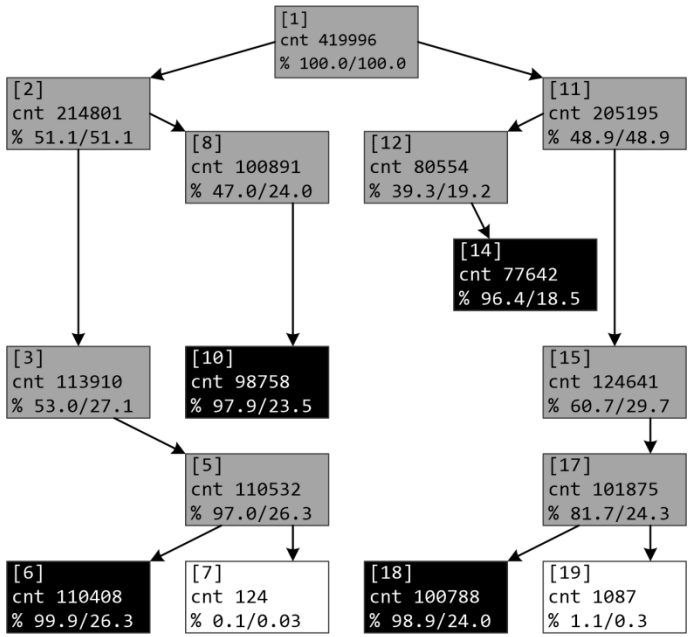
```
VBLENDMPS (EVEX encoded versions)
(KL, VL) = (4, 128), (8, 256), (16, 512)
FOR j ← 0 TO KL-1
    i ← j * 32
    IF k1[j] OR *no controlmask*
        THEN
            IF (EVEX.b = 1) AND (SRC2 *is memory*)
                THEN
                    DEST[i+31:i] ← SRC2[31:0]
                ELSE
                    DEST[i+31:i] ← SRC2[i+31:i]
            FI;
        ELSE
            IF *merging-masking*                ; merging-masking
                THEN DEST[i+31:i] ← SRC1[i+31:i]
                ELSE                            ; zeroing-masking
                    DEST[i+31:i] ← 0
            FI;
    FI;
ENDFOR
DEST[MAXVL-1:VL] ← 0
```

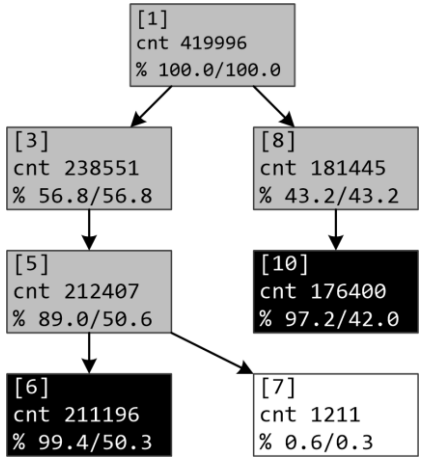
```
_mm512_mask_blend_ps
```



Векторизованная версия функции sample (упрощение графа потока управления функции)

```
01 void sample_16(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
02                __m512 dr, __m512 ur, __m512 pr, __m512 cr,
03                __m512 pm, __m512 um,
04                __m512 *d, __m512 *u, __m512 *p)
05 {
06     __m512 c, ums, pms, sh, st, s, uc;
07     __mmask16 cond_um, cond_pm, cond_sh, cond_st, cond_s, cond_sh_st;
08
09     // d/u/p/c/ums
10     cond_um = _mm512_cmp_ps_mask(um, z, _MM_CMPINT_LT);
11     *d = _mm512_mask_blend_ps(cond_um, dl, dr);
12     *u = _mm512_mask_blend_ps(cond_um, ul, ur);
13     *p = _mm512_mask_blend_ps(cond_um, pl, pr);
14     c = _mm512_mask_blend_ps(cond_um, cl, cr);
15     ums = um;
16     *u = _mm512_mask_sub_ps(*u, cond_um, z, *u);
17     ums = _mm512_mask_sub_ps(ums, cond_um, z, ums);
18
19     // Calculate main values.
20     pms = DIV(pm, *p);
21     sh = SUB(*u, c);
22     st = FNMADD(POW(pms, g1), c, ums);
23     s = FNMADD(c, SQRT(FMADD(g2, pms, g1)), *u);
24
25     // Conditions.
26     cond_pm = _mm512_cmp_ps_mask(pm, *p, _MM_CMPINT_LE);
27     cond_sh = _mm512_mask_cmp_ps_mask(cond_pm, sh, z, _MM_CMPINT_LT);
28     cond_st = _mm512_mask_cmp_ps_mask(cond_sh, st, z, _MM_CMPINT_LT);
29     cond_s = _mm512_mask_cmp_ps_mask(~cond_pm, s, z, _MM_CMPINT_LT);
30
31     // Store.
32     *d = _mm512_mask_mov_ps(*d, cond_st,
33                             MUL(*d, POW(pms, SET1(1.0 / GAMA))));
34     *d = _mm512_mask_mov_ps(*d, cond_s MUL(*d, DIV(ADD(pms, g6),
35                                                    FMADD(pms, g6, one))));
36     *u = _mm512_mask_mov_ps(*u, cond_st | cond_s, ums);
37     *p = _mm512_mask_mov_ps(*p, cond_st | cond_s, pm);
38
39     // Low prob - ignore it.
40     cond_sh_st = cond_sh & ~cond_st;
41     if (cond_sh_st != 0x0)
42     {
43         *u = _mm512_mask_mov_ps(*u, cond_sh_st, MUL(g5, FMADD(g7, *u, c)));
44         uc = DIV(*u, c);
45         *d = _mm512_mask_mov_ps(*d, cond_sh_st, MUL(*d, POW(uc, g4)));
46         *p = _mm512_mask_mov_ps(*p, cond_sh_st, MUL(*p, POW(uc, g3)));
47     }
48
49     // Final store.
50     *u = _mm512_mask_sub_ps(*u, cond_um, z, *u);
51 }
```





Оригинальная версия функции starpu

```
01  void starpu(float dl, float ul, float pl, float cl,
02              float dr, float ur, float pr, float cr,
03              float &p, float &u)
04  {
05      const int nriter = 20;
06      const float tolpre = 1.0e-6;
07      float change, fl, fld, fr, frd, pold, pstart, udiff;
08
09      guessp(dl, ul, pl, cl, dr, ur, pr, cr, pstart);
10      pold = pstart;
11      udiff = ur - ul;
12
13      int i = 1;
14
15      for ( ; i <= nriter; i++)
16      {
17          prefun(fl, fld, pold, dl, pl, cl);
18          prefun(fr, frd, pold, dr, pr, cr);
19          p = pold - (fl + fr + udiff) / (fld + frd);
20          change = 2.0 * abs((p - pold) / (p + pold));
21
22          if (change <= tolpre)
23          {
24              break;
25          }
26
27          if (p < 0.0)
28          {
29              p = tolpre;
30          }
31
32          pold = p;
33      }
34
35      if (i > nriter)
36      {
37          cout << "divergence in Newton-Raphson iteration" << endl;
38          exit(1);
39      }
40
41      u = 0.5 * (ul + ur + fr - fl);
42  }
```

Профиль исполнения цикла из функции `starpu`

```
131  41999600:  126:    int i = 1;
132        -:  127:
133 112947700:  128:    for ( ; i <= nriter; i++)
134        -:  129:    {
135 112947700:  130:        prefun(fl, fld, pold, dl, pl, cl);
136 112947700:  131:        prefun(fr, frd, pold, dr, pr, cr);
137 112947700:  132:        p = pold - (fl + fr + udiff) / (fld + frd);
138 112947700:  133:        change = 2.0 * abs((p - pold) / (p + pold));
139        -:  134:
140 112947700:  135:        if (change <= tolpre)
141        -:  136:        {
142  41999600:  137:            break;
143        -:  138:        }
144        -:  139:
145  70948100:  140:        if (p < 0.0)
146        -:  141:        {
147   4751000:  142:            p = tolpre;
148        -:  143:        }
149        -:  144:
150  70948100:  145:        pold = p;
151        -:  146:    }
152        -:  147:
153  41999600:  148:    if (i > nriter)
154        -:  149:    {
155     #####:  150:        cout << "divergence in Newton-Raphson iteration" << endl;
156        -:  151:
157     #####:  152:        exit(1);
158        -:  153:    }
```

× 2.7

× 1.7

Векторизованная версия функции starpu

```
01 void starpu_16(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
02                __m512 dr, __m512 ur, __m512 pr, __m512 cr,
03                __m512 *p, __m512 *u)
04 {
05     __m512 two, tolpre, tolpre2, udiff, pold, fl, fld, fr, frd, change;
06     __mmask16 cond_break, cond_neg, m;
07     const int nriter = 20;
08     int iter = 1;
09
10     two = SET1(2.0);
11     tolpre = SET1(1.0e-6);
12     tolpre2 = SET1(5.0e-7);
13     udiff = SUB(ur, ul);
14
15     guessp_16(dl, ul, pl, cl, dr, ur, pr, cr, &pold);
16
17     // Start with full mask.
18     m = 0xFFFF;
19
20     for (; (iter <= nriter) && (m != 0x0); iter++)
21     {
22         prefun_16(&fl, &fld, pold, dl, pl, cl, m);
23         prefun_16(&fr, &frd, pold, dr, pr, cr, m);
24         *p = _mm512_mask_sub_ps(*p, m, pold,
25                                 _mm512_mask_div_ps(z, m,
26                                                    ADD(ADD(fl, fr), udiff),
27                                                    ADD(fld, frd)));
28         change = ABS(_mm512_mask_div_ps(z, m, SUB(*p, pold),
29                                         ADD(*p, pold)));
30         cond_break = _mm512_mask_cmp_ps_mask(m, change,
31                                              tolpre2, _MM_CMPINT_LE);
32         m &= ~cond_break;
33         cond_neg = _mm512_mask_cmp_ps_mask(m, *p, z, _MM_CMPINT_LT);
34         *p = _mm512_mask_mov_ps(*p, cond_neg, tolpre);
35         pold = _mm512_mask_mov_ps(pold, m, *p);
36     }
37
38     // Check for divergence.
39     if (iter > nriter)
40     {
41         cout << "divergence in Newton-Raphson iteration" << endl;
42         exit(1);
43     }
44
45     *u = MUL(SET1(0.5), ADD(ADD(ul, ur), SUB(fr, fl)));
46 }
```

Выводы

Достигнутые результаты:

• Ускорение римановского решателя в 7 раз по отношению к невекторизованной версии.

Направления дальнейших исследований:

• Распространение векторизации на случай трехмерных блочно-структурированных расчетных сеток, состоящих из большого количества блоков.

• Повышение локальности в формируемых массивов данных для векторизованного решения задачи Римана. Исследование влияния локальности на эффективность векторизации.

• Исследование влияния характера функции распределения длины итерации векторизуемого цикла на эффективность векторизации.

Спасибо за внимание!

Рыбаков Алексей Анатольевич
rybakov@jscc.ru

Шумилин Сергей Сергеевич
shumilin@jscc.ru