AP-929

Streaming SIMD Extensions - Inverse of 6x6 Matrix

March 1999

Order Number: 245044-001

Table of Contents

Revision History

Revision	Revision History	Date
1.0	First external publication	3/99
0.99	Internal publication	1/99

References

The following documents are referenced in this application note, and provide background or supporting information for understanding the topics presented in this document.

1. *Increasing the Accuracy of the Results from the Reciprocal and Reciprocal Square Root Instructions using the Newton-Raphson Method*, Intel Application Note AP-803, Order No: 243637-001.

2. *Using the RDTSC Instruction for Performance Monitoring*, http://www.intel.com/drg/pentiumII/appnotes/RDTSCPM1.HTM

1 Introduction

This application note describes the inversion of a 6x6-matrix using Streaming SIMD Extensions.

The performance of the C code with Streaming SIMD Extensions, which implements the inverse of a 6x6-matrix under some general assumptions, is approximately 2.5 times better that C code implementation based on Gaussian elimination method. (See section 5.1.)

2 The Inverse of a Matrix

The matrix A^{-1} is said to be ***inverse*** of a matrix A if it satisfies the following equality:

$A^{-1} * A = I$,

where I is the unit matrix (with main diagonal elements equal to 1 and all others equal to 0).

The inverse matrix exists if and only if $\det(A) \neq 0$. The considered algorithm of inverse matrix evaluation consists of 3 stages.

1. By transforming the columns, reduce the matrix A to the form: $A_1 = \begin{bmatrix} E_{2x2} & O_{2x4} \\ B_{4x2} & C_{4x4} \end{bmatrix}$, where E_{2x2} is the unit matrix and O_{2x4} is the zero matrix. This transformation can be written in matrix form:

   $A_1 = A \cdot P_1$

2. Find an inverse matrix of C_{4x4} using Cramer's Rule. Multiply matrix A_1 by $P_2 = \begin{bmatrix} E_{2x2} & O_{2x4} \\ O_{4x2} & C^{-1}{}_{4x4} \end{bmatrix}$ as follows:

   $A_2 = (A \cdot P_1) \cdot P_2 = \begin{bmatrix} E_{2x2} & O_{2x4} \\ B_{4x2} & E_{4x4} \end{bmatrix}$

3. Then multiply matrix A_2 by $P_3 = \begin{bmatrix} E_{2x2} & O_{2x4} \\ -B_{4x2} & E_{4x4} \end{bmatrix}$:

   $A_3 = (A \cdot P_1 \cdot P_2) \cdot P_3 = \begin{bmatrix} E_{2x2} & O_{2x4} \\ B_{4x2} & E_{4x4} \end{bmatrix} \cdot \begin{bmatrix} E_{2x2} & O_{2x4} \\ -B_{4x2} & E_{4x4} \end{bmatrix} = E.$

4. Matrix $P_1 \cdot P_2 \cdot P_3$ is the inverse of matrix A.

2.1 Implementation Details

Right multiplication of matrix A by matrix P may be performed by manipulations of columns of matrix A, while left multiplication is achieved with some transformations of the rows. Using this observation, we can easily design an effective implementation of matrix inverse. By applying some column transformations (multiplication of a column by a number, addition of other columns to linear combination of columns, or rearrangement of columns), we can reduce a given matrix A

to the form A_1. Then we may evaluate the matrix $P = P_2 \cdot P_3$ and, applying the same transformations to the column s of the matrix P, which were used to compute A_1, evaluate $A^{-1} = P_1 \cdot P_2 \cdot P_3$.

3 Performance

The performance of a 6x6-matrix inversion can be increased if we use Streaming SIMD Extensions instructions. One SIMD instruction may be used to operate on 4 data elements. This allows you to process 4 elements of the row or column of the matrix in one instruction.
Additional performance gain is achieved by substituting the `divps` instruction, characterized by rather high latency, with the low-latency `rcpps` instruction, followed, if high accuracy is required, with Newton-Raphson approximation. For more information, refer to the Intel Application Note AP-803, *Increasing the Accuracy of the Results from the Reciprocal and Reciprocal Square Root Instructions using the Newton-Raphson Method*, Order No: 243637.
.
Table 1 compares performance of the following functions:

- Implemented using C code for the case when there are no numbers close to zero on the main diagonal
- Implemented using C code with Streaming SIMD Extensions for a general case
- Implemented using C code with Streaming SIMD Extensions for a special case, when $a_{00} \neq 0$ and $(a_{01} \cdot a_{10} - a_{10} \cdot a_{01}) \neq 0$.

Processor cycles were measured by using the `rdtsc` instruction (see http://www.intel.com/drg/pentiumII/appnotes/RDTSCPM1.HTM).

Table 1: Performance Gains Using Streaming SIMD Extensions [1]

Version	Cycles
C code in special case	1610
C code in general case and Streaming SIMD Extensions	852
C code in special case and Streaming SIMD Extensions	625

4 Conclusion

Streaming SIMD Extensions yield an increase in the performance of 6x6 matrix inversion. The key reasons for this performance boost in applications using Streaming SIMD Extensions are:

- Use of single-instruction-multiple-data commands of the Pentium® III processor;

[1] These measurements are based on tests run on a 450MHz, 64MB SDRAM, 100MHz bus Pentium® III processor. This is the first Pentium III processor release. Performance on future releases of Pentium III processor may vary.

- The `rcpps` instruction, combined with a Newton-Raphson algorithm, can substitute for the `divps` command, characterized by high latency in cases when complete accuracy is not required.

5 Source Code

Below three different codes are provided. The first example is matrix inversion in cases when there are no numbers close to zero on the main diagona;, the second example is an implementation of a general case; while the third code corresponds to a case when the reference matrix can be reduced to the form A_1 without column rearrangements (i.e. $a_{00} \neq 0$ and $a_{00} \cdot a_{11} - a_{10} \cdot a_{01} \neq 0$).
These examples require the Intel® C/C++ Compiler (http://support.intel.com/support/performancetools/c/).

5.1 C Code in Special Case

```
bool PII_Invert_6x6(float *src)
{
    float d, di;

    di = src[0];
    src[0] = d = 1.0f / di;
    src[6] *= -d;
    src[12] *= -d;
    src[18] *= -d;
    src[24] *= -d;
    src[30] *= -d;
    src[1] *= d;
    src[2] *= d;
    src[3] *= d;
    src[4] *= d;
    src[5] *= d;
    src[7] += src[6] * src[1] * di;
    src[8] += src[6] * src[2] * di;
    src[9] += src[6] * src[3] * di;
    src[10] += src[6] * src[4] * di;
    src[11] += src[6] * src[5] * di;
    src[13] += src[12] * src[1] * di;
    src[14] += src[12] * src[2] * di;
    src[15] += src[12] * src[3] * di;
    src[16] += src[12] * src[4] * di;
    src[17] += src[12] * src[5] * di;
    src[19] += src[18] * src[1] * di;
    src[20] += src[18] * src[2] * di;
    src[21] += src[18] * src[3] * di;
    src[22] += src[18] * src[4] * di;
    src[23] += src[18] * src[5] * di;
    src[25] += src[24] * src[1] * di;
    src[26] += src[24] * src[2] * di;
    src[27] += src[24] * src[3] * di;
    src[28] += src[24] * src[4] * di;
    src[29] += src[24] * src[5] * di;
    src[31] += src[30] * src[1] * di;
    src[32] += src[30] * src[2] * di;
    src[33] += src[30] * src[3] * di;
    src[34] += src[30] * src[4] * di;
    src[35] += src[30] * src[5] * di;
    di = src[7];
    src[7] = d = 1.0f / di;
    src[1] *= -d;
    src[13] *= -d;
    src[19] *= -d;
    src[25] *= -d;
    src[31] *= -d;
    src[6] *= d;
    src[8] *= d;
    src[9] *= d;
    src[10] *= d;
    src[11] *= d;
    src[0] += src[1] * src[6] * di;
    src[2] += src[1] * src[8] * di;
```

```
src[3] += src[1] * src[9] * di;
src[4] += src[1] * src[10] * di;
src[5] += src[1] * src[11] * di;
src[12] += src[13] * src[6] * di;
src[14] += src[13] * src[8] * di;
src[15] += src[13] * src[9] * di;
src[16] += src[13] * src[10] * di;
src[17] += src[13] * src[11] * di;
src[18] += src[19] * src[6] * di;
src[20] += src[19] * src[8] * di;
src[21] += src[19] * src[9] * di;
src[22] += src[19] * src[10] * di;
src[23] += src[19] * src[11] * di;
src[24] += src[25] * src[6] * di;
src[26] += src[25] * src[8] * di;
src[27] += src[25] * src[9] * di;
src[28] += src[25] * src[10] * di;
src[29] += src[25] * src[11] * di;
src[30] += src[31] * src[6] * di;
src[32] += src[31] * src[8] * di;
src[33] += src[31] * src[9] * di;
src[34] += src[31] * src[10] * di;
src[35] += src[31] * src[11] * di;
di = src[14];
src[14] = d = 1.0f / di;
src[2] *= -d;
src[8] *= -d;
src[20] *= -d;
src[26] *= -d;
src[32] *= -d;
src[12] *= d;
src[13] *= d;
src[15] *= d;
src[16] *= d;
src[17] *= d;
src[0] += src[2] * src[12] * di;
src[1] += src[2] * src[13] * di;
src[3] += src[2] * src[15] * di;
src[4] += src[2] * src[16] * di;
src[5] += src[2] * src[17] * di;
src[6] += src[8] * src[12] * di;
src[7] += src[8] * src[13] * di;
src[9] += src[8] * src[15] * di;
src[10] += src[8] * src[16] * di;
src[11] += src[8] * src[17] * di;
src[18] += src[20] * src[12] * di;
src[19] += src[20] * src[13] * di;
src[21] += src[20] * src[15] * di;
src[22] += src[20] * src[16] * di;
src[23] += src[20] * src[17] * di;
src[24] += src[26] * src[12] * di;
src[25] += src[26] * src[13] * di;
src[27] += src[26] * src[15] * di;
src[28] += src[26] * src[16] * di;
src[29] += src[26] * src[17] * di;
src[30] += src[32] * src[12] * di;
src[31] += src[32] * src[13] * di;
src[33] += src[32] * src[15] * di;
src[34] += src[32] * src[16] * di;
src[35] += src[32] * src[17] * di;
di = src[21];
src[21] = d = 1.0f / di;
src[3] *= -d;
src[9] *= -d;
src[15] *= -d;
src[27] *= -d;
src[33] *= -d;
src[18] *= d;
src[19] *= d;
src[20] *= d;
src[22] *= d;
src[23] *= d;
src[0] += src[3] * src[18] * di;
src[1] += src[3] * src[19] * di;
src[2] += src[3] * src[20] * di;
src[4] += src[3] * src[22] * di;
src[5] += src[3] * src[23] * di;
src[6] += src[9] * src[18] * di;
src[7] += src[9] * src[19] * di;
src[8] += src[9] * src[20] * di;
src[10] += src[9] * src[22] * di;
```

```
src[11] += src[9] * src[23] * di;
src[12] += src[15] * src[18] * di;
src[13] += src[15] * src[19] * di;
src[14] += src[15] * src[20] * di;
src[16] += src[15] * src[22] * di;
src[17] += src[15] * src[23] * di;
src[24] += src[27] * src[18] * di;
src[25] += src[27] * src[19] * di;
src[26] += src[27] * src[20] * di;
src[28] += src[27] * src[22] * di;
src[29] += src[27] * src[23] * di;
src[30] += src[33] * src[18] * di;
src[31] += src[33] * src[19] * di;
src[32] += src[33] * src[20] * di;
src[34] += src[33] * src[22] * di;
src[35] += src[33] * src[23] * di;
di = src[28];
src[28] = d = 1.0f / di;
src[4] *= -d;
src[10] *= -d;
src[16] *= -d;
src[22] *= -d;
src[34] *= -d;
src[24] *= d;
src[25] *= d;
src[26] *= d;
src[27] *= d;
src[29] *= d;
src[0] += src[4] * src[24] * di;
src[1] += src[4] * src[25] * di;
src[2] += src[4] * src[26] * di;
src[3] += src[4] * src[27] * di;
src[5] += src[4] * src[29] * di;
src[6] += src[10] * src[24] * di;
src[7] += src[10] * src[25] * di;
src[8] += src[10] * src[26] * di;
src[9] += src[10] * src[27] * di;
src[11] += src[10] * src[29] * di;
src[12] += src[16] * src[24] * di;
src[13] += src[16] * src[25] * di;
src[14] += src[16] * src[26] * di;
src[15] += src[16] * src[27] * di;
src[17] += src[16] * src[29] * di;
src[18] += src[22] * src[24] * di;
src[19] += src[22] * src[25] * di;
src[20] += src[22] * src[26] * di;
src[21] += src[22] * src[27] * di;
src[23] += src[22] * src[29] * di;
src[30] += src[34] * src[24] * di;
src[31] += src[34] * src[25] * di;
src[32] += src[34] * src[26] * di;
src[33] += src[34] * src[27] * di;
src[35] += src[34] * src[29] * di;
di = src[35];
src[35] = d = 1.0f / di;
src[5] *= -d;
src[11] *= -d;
src[17] *= -d;
src[23] *= -d;
src[29] *= -d;
src[30] *= d;
src[31] *= d;
src[32] *= d;
src[33] *= d;
src[34] *= d;
src[0] += src[5] * src[30] * di;
src[1] += src[5] * src[31] * di;
src[2] += src[5] * src[32] * di;
src[3] += src[5] * src[33] * di;
src[4] += src[5] * src[34] * di;
src[6] += src[11] * src[30] * di;
src[7] += src[11] * src[31] * di;
src[8] += src[11] * src[32] * di;
src[9] += src[11] * src[33] * di;
src[10] += src[11] * src[34] * di;
src[12] += src[17] * src[30] * di;
src[13] += src[17] * src[31] * di;
src[14] += src[17] * src[32] * di;
src[15] += src[17] * src[33] * di;
src[16] += src[17] * src[34] * di;
src[18] += src[23] * src[30] * di;
```

```
    src[19] += src[23] * src[31] * di;
    src[20] += src[23] * src[32] * di;
    src[21] += src[23] * src[33] * di;
    src[22] += src[23] * src[34] * di;
    src[24] += src[29] * src[30] * di;
    src[25] += src[29] * src[31] * di;
    src[26] += src[29] * src[32] * di;
    src[27] += src[29] * src[33] * di;
    src[28] += src[29] * src[34] * di;

    return true; // always, even when there is no A^-1.

}
```

5.2 C Code in General Case with Streaming SIMD Extensions

```
void PIII_Invert_6x6(float *src)
{
#define   EPSILON          1e-8
#define   REAL_ZERO(x)     (fabs(x) < EPSILON ? 1:0)

        __m128  minor0, minor1, minor2, minor3;
        __m128  det, tmp1, tmp2, tmp3, mask, index;
        __m128  b[6];
        __m128  row[6];

        static const unsigned long        minus_hex        = 0x80000000;
        static const __m128               minus    = _mm_set_ps1(*(float*)&minus_hex);
        static const __m128               e        = _mm_set_ps(1.0f, 0.0f, 0.0f, 1.0f);
        static const __m128               epsilon  = _mm_set_ss(EPSILON);

        float   max, f;

        int i, j, n1, n2, k, mask1, mask2, mask3;

        // Loading matrixes: 4x2 to row[0], row[1] and 4x4 to row[2]...row[5].

        tmp1    = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[12])), (__m64*)(&src[18]));
        tmp2    = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[24])), (__m64*)(&src[30]));

        row[0]  = _mm_shuffle_ps(tmp1, tmp2, 0x88);
        row[1]  = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

        tmp1    = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[14])), (__m64*)(&src[20]));
        tmp2    = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[26])), (__m64*)(&src[32]));

        row[2]  = _mm_shuffle_ps(tmp1, tmp2, 0x88);
        row[3]  = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

        tmp1    = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[16])), (__m64*)(&src[22]));
        tmp2    = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[28])), (__m64*)(&src[34]));

        row[4]  = _mm_shuffle_ps(tmp1, tmp2, 0x88);
        row[5]  = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

        // Finding the max(|src[0]|, |src[1]|, ..., |src[5]|).

        tmp1    = _mm_loadh_pi(_mm_load_ss(&src[2]), (__m64*)&src[0]);
        tmp2    = _mm_loadh_pi(_mm_load_ss(&src[3]), (__m64*)&src[4]);

        tmp1    = _mm_andnot_ps(minus, tmp1);
        tmp2    = _mm_andnot_ps(minus, tmp2);

        tmp3    = _mm_max_ps(tmp1, tmp2);
        tmp3    = _mm_max_ps(tmp3, _mm_shuffle_ps(tmp3, tmp3, _MM_SHUFFLE(3, 2, 3, 2)));
        tmp3    = _mm_max_ss(tmp3, _mm_shuffle_ps(tmp3, tmp3, _MM_SHUFFLE(1, 1, 1, 1)));
        tmp3    = _mm_shuffle_ps(tmp3, tmp3, _MM_SHUFFLE(0, 0, 0, 0));

        mask1   = _mm_movemask_ps(_mm_cmpeq_ps(tmp1, tmp3));
        mask1   |= _mm_movemask_ps(_mm_cmpeq_ps(tmp2, tmp3))<<4;

        mask2   = mask1 & 0x98;
        mask2   = mask2 - (mask2 << 1);
        n1      = ((unsigned int)mask2) >> 31;

        n1      |= ((mask1 & 0x11) != 0) << 1;
```

```
mask2    = mask1 & 0xC0;
mask2    = mask2 - (mask2 << 1);
n1       |= (((unsigned int)mask2) >> 29) & 4;

if(REAL_ZERO(src[n1]))
        return;

// The first Gauss iteration.

tmp1     = row[n1];
row[n1]  = row[0];
row[0]   = tmp1;

tmp2     = _mm_load_ss(&src[n1]);

src[n1]  = src[0];

f        = src[n1+6];
src[n1+6]        = src[6];
src[6]   = f;

tmp1     = _mm_rcp_ss(tmp2);
tmp2     = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(tmp2, _mm_mul_ss(tmp1, tmp1)));

_mm_store_ss(&src[0], tmp2);

tmp2     = _mm_shuffle_ps(tmp2, tmp2, 0x00);
row[0]   = _mm_mul_ps(row[0], tmp2);

tmp1     = _mm_load_ss(&src[1]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[1]   = _mm_sub_ps(row[1], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[2]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[2]   = _mm_sub_ps(row[2], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[3]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[3]   = _mm_sub_ps(row[3], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[4]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[4]   = _mm_sub_ps(row[4], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[5]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[5]   = _mm_sub_ps(row[5], _mm_mul_ps(row[0], tmp1));

tmp3     = _mm_load_ss(&src[6]);
tmp3     = _mm_mul_ss(tmp3, tmp2);
_mm_store_ss(&src[6], tmp3);

tmp1     = _mm_load_ss(&src[1]);
tmp2     = _mm_load_ss(&src[7]);
tmp2     = _mm_sub_ss(tmp2, _mm_mul_ss(tmp1, tmp3));
_mm_store_ss(&src[7], tmp2);

tmp3     = _mm_shuffle_ps(tmp3, tmp3, 0x00);
tmp1     = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)&src[2]), (__m64*)&src[ 4]);
tmp2     = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)&src[8]), (__m64*)&src[10]);

tmp2     = _mm_sub_ps(tmp2, _mm_mul_ps(tmp1, tmp3));

_mm_storel_pi((__m64*)&src[ 8], tmp2);
_mm_storeh_pi((__m64*)&src[10], tmp2);

// Finding the max(src[7], src[8], ..., src[11]).

tmp1     = _mm_loadh_pi(_mm_load_ss(&src[7]), (__m64*)&src[10]);
tmp2     = _mm_loadl_pi(tmp2, (__m64*)&src[8]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, _MM_SHUFFLE(0,3,2,2));

tmp1     = _mm_andnot_ps(minus, tmp1);
tmp2     = _mm_andnot_ps(minus, tmp2);

tmp3     = _mm_max_ps(tmp1, tmp2);
tmp3     = _mm_max_ps(tmp3, _mm_shuffle_ps(tmp1, tmp1, _MM_SHUFFLE(0,0,3,2)));
tmp3     = _mm_max_ss(tmp3, _mm_shuffle_ps(tmp3, tmp3, _MM_SHUFFLE(1,1,1,1)));
tmp3     = _mm_shuffle_ps(tmp3, tmp3, _MM_SHUFFLE(0,0,0,0));
```

```
        mask1    = _mm_movemask_ps(_mm_cmpeq_ps(tmp2, tmp3));
        mask2    = _mm_movemask_ps(_mm_cmpeq_ps(tmp1, tmp3));

        n2       = ((mask1 & 3) | (mask2 & 7)) + 7;

        if(REAL_ZERO(src[n2]))
                return;

        // The second Gauss iteration.

        tmp2     = _mm_load_ss(&src[n2]);
        src[n2]  = src[7];

        n2       -= 6;

        tmp1     = row[n2];
        row[n2]  = row[1];
        row[1]   = tmp1;

        f        = src[n2];
        src[n2]  = src[1];
        src[1]   = f;

        //if(n2==n1) n2 = 0;
        n2       *= (n1!=n2);

        tmp1     = _mm_rcp_ss(tmp2);
        tmp2     = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(tmp2, _mm_mul_ss(tmp1, tmp1)));

        _mm_store_ss(&src[7], tmp2);

        tmp2     = _mm_shuffle_ps(tmp2, tmp2, 0x00);
        row[1]   = _mm_mul_ps(row[1], tmp2);

        tmp1     = _mm_load_ss(&src[6]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[0]   = _mm_sub_ps(row[0], _mm_mul_ps(row[1], tmp1));

        tmp1     = _mm_load_ss(&src[8]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[2]   = _mm_sub_ps(row[2], _mm_mul_ps(row[1], tmp1));

        tmp1     = _mm_load_ss(&src[9]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[3]   = _mm_sub_ps(row[3], _mm_mul_ps(row[1], tmp1));

        tmp1     = _mm_load_ss(&src[10]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[4]   = _mm_sub_ps(row[4], _mm_mul_ps(row[1], tmp1));

        tmp1     = _mm_load_ss(&src[11]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[5]   = _mm_sub_ps(row[5], _mm_mul_ps(row[1], tmp1));

        row[0]   = _mm_xor_ps(row[0], minus);
        row[1]   = _mm_xor_ps(row[1], minus);

        // Inverting the matrix 4x4 by the Kramers method.

        row[3]   = _mm_shuffle_ps(row[3], row[3], 0x4E);
        row[5]   = _mm_shuffle_ps(row[5], row[5], 0x4E);

        tmp2     = _mm_mul_ps(row[4], row[5]);
        tmp1     = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor0   = _mm_mul_ps(row[3], tmp1);
        minor1   = _mm_mul_ps(row[2], tmp1);

        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0   = _mm_sub_ps(_mm_mul_ps(row[3], tmp1), minor0);
        minor1   = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor1);
        minor1   = _mm_shuffle_ps(minor1, minor1, 0x4E);
//      -----------------------------------------------
        tmp2     = _mm_mul_ps(row[3], row[4]);
        tmp1     = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor0   = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor0);
        minor3   = _mm_mul_ps(row[2], tmp1);
```

```
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0  = _mm_sub_ps(minor0, _mm_mul_ps(row[5], tmp1));
        minor3  = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor3);
        minor3  = _mm_shuffle_ps(minor3, minor3, 0x4E);
//      ----------------------------------------------
        tmp2    = _mm_mul_ps(_mm_shuffle_ps(row[3], row[3], 0x4E), row[5]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);
        row[4]  = _mm_shuffle_ps(row[4], row[4], 0x4E);

        minor0  = _mm_add_ps(_mm_mul_ps(row[4], tmp1), minor0);
        minor2  = _mm_mul_ps(row[2], tmp1);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0  = _mm_sub_ps(minor0, _mm_mul_ps(row[4], tmp1));
        minor2  = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor2);
        minor2  = _mm_shuffle_ps(minor2, minor2, 0x4E);
//      ----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[3]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor2  = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor2);
        minor3  = _mm_sub_ps(_mm_mul_ps(row[4], tmp1), minor3);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor2  = _mm_sub_ps(_mm_mul_ps(row[5], tmp1), minor2);
        minor3  = _mm_sub_ps(minor3, _mm_mul_ps(row[4], tmp1));
//      ----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[5]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor1  = _mm_sub_ps(minor1, _mm_mul_ps(row[4], tmp1));
        minor2  = _mm_add_ps(_mm_mul_ps(row[3], tmp1), minor2);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor1  = _mm_add_ps(_mm_mul_ps(row[4], tmp1), minor1);
        minor2  = _mm_sub_ps(minor2, _mm_mul_ps(row[3], tmp1));
//      ----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[4]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor1  = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor1);
        minor3  = _mm_sub_ps(minor3, _mm_mul_ps(row[3], tmp1));

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor1  = _mm_sub_ps(minor1, _mm_mul_ps(row[5], tmp1));
        minor3  = _mm_add_ps(_mm_mul_ps(row[3], tmp1), minor3);
//      ----------------------------------------------
        det     = _mm_mul_ps(row[2], minor0);
        det     = _mm_add_ps(_mm_shuffle_ps(det, det, 0x4E), det);
        det     = _mm_add_ss(_mm_shuffle_ps(det, det, 0xB1), det);

        if(_mm_movemask_ps(_mm_cmplt_ss(_mm_andnot_ps(minus, det), epsilon)) & 1)
                return;

        tmp1    = _mm_rcp_ss(det);
        det     = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(det, _mm_mul_ss(tmp1, tmp1)));
        det     = _mm_shuffle_ps(det, det, 0x00);

        row[2]  = _mm_mul_ps(det, minor0);
        row[3]  = _mm_mul_ps(det, minor1);

        ////////////////////////////////

        b[0]    = _mm_unpacklo_ps(row[0], row[1]);
        b[2]    = _mm_unpackhi_ps(row[0], row[1]);
        row[4]  = _mm_mul_ps(det, minor2);
        b[1]    = _mm_shuffle_ps(b[0], b[2], 0x4E);
        row[5]  = _mm_mul_ps(det, minor3);
        b[3]    = _mm_shuffle_ps(b[2], b[0], 0x4E);

        tmp1    = _mm_shuffle_ps(row[2], row[3], 0x50);
        tmp2    = _mm_mul_ps(b[0], tmp1);

        tmp1    = _mm_shuffle_ps(row[2], row[3], 0xA5);
        tmp2    = _mm_add_ps(tmp2, _mm_mul_ps(b[1], tmp1));

        tmp1    = _mm_shuffle_ps(row[2], row[3], 0xFA);
```

```
tmp2    = _mm_add_ps(tmp2, _mm_mul_ps(b[2], tmp1));

tmp1    = _mm_shuffle_ps(row[2], row[3], 0x0F);
row[0]  = _mm_add_ps(tmp2, _mm_mul_ps(b[3], tmp1));

tmp1    = _mm_shuffle_ps(row[4], row[5], 0x50);
tmp2    = _mm_mul_ps(b[0], tmp1);

tmp1    = _mm_shuffle_ps(row[4], row[5], 0xA5);
tmp2    = _mm_add_ps(tmp2, _mm_mul_ps(b[1], tmp1));

tmp1    = _mm_shuffle_ps(row[4], row[5], 0xFA);
tmp2    = _mm_add_ps(tmp2, _mm_mul_ps(b[2], tmp1));

tmp1    = _mm_shuffle_ps(row[4], row[5], 0x0F);
row[1]  = _mm_add_ps(tmp2, _mm_mul_ps(b[3], tmp1));

b[2]    = _mm_shuffle_ps(row[0], row[0], 0x44);
b[3]    = _mm_shuffle_ps(row[0], row[0], 0xEE);
b[4]    = _mm_shuffle_ps(row[1], row[1], 0x44);
b[5]    = _mm_shuffle_ps(row[1], row[1], 0xEE);
```

// Calculating row number n2

```
tmp1    = _mm_load_ss(&src[8]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [1]  = _mm_sub_ps(_mm_shuffle_ps(e, e, 0x4E), _mm_mul_ps(b[2], tmp1));
row[1]  = _mm_xor_ps(_mm_mul_ps(row[2], tmp1), minus);

tmp1    = _mm_load_ss(&src[9]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [1]  = _mm_sub_ps(b  [1], _mm_mul_ps(b  [3], tmp1));
row[1]  = _mm_sub_ps(row[1], _mm_mul_ps(row[3], tmp1));

tmp1    = _mm_load_ss(&src[10]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [1]  = _mm_sub_ps(b  [1], _mm_mul_ps(b  [4], tmp1));
row[1]  = _mm_sub_ps(row[1], _mm_mul_ps(row[4], tmp1));

tmp1    = _mm_load_ss(&src[11]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [1]  = _mm_sub_ps(b  [1], _mm_mul_ps(b  [5], tmp1));
row[1]  = _mm_sub_ps(row[1], _mm_mul_ps(row[5], tmp1));

tmp1    = _mm_load_ss(&src[6]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [1]  = _mm_sub_ps(b[1], _mm_mul_ps(e, tmp1));

tmp2    = _mm_load_ss(&src[7]);
tmp2    = _mm_shuffle_ps(tmp2, tmp2, 0x00);

b  [1]  = _mm_mul_ps(b  [1], tmp2);
row[1]  = _mm_mul_ps(row[1], tmp2);
```

// Calculating row number n1

```
tmp1    = _mm_load_ss(&src[1]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [0]  = _mm_sub_ps(e, _mm_mul_ps(b[1], tmp1));
row[0]  = _mm_xor_ps(_mm_mul_ps(row[1], tmp1), minus);

tmp1    = _mm_load_ss(&src[2]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [0]  = _mm_sub_ps(b  [0], _mm_mul_ps(b  [2], tmp1));
row[0]  = _mm_sub_ps(row[0], _mm_mul_ps(row[2], tmp1));

tmp1    = _mm_load_ss(&src[3]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

b  [0]  = _mm_sub_ps(b  [0], _mm_mul_ps(b  [3], tmp1));
row[0]  = _mm_sub_ps(row[0], _mm_mul_ps(row[3], tmp1));

tmp1    = _mm_load_ss(&src[4]);
tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);
```

```
        b  [0]  = _mm_sub_ps(b  [0], _mm_mul_ps(b  [4], tmp1));
        row[0]  = _mm_sub_ps(row[0], _mm_mul_ps(row[4], tmp1));

        tmp1    = _mm_load_ss(&src[5]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b  [0]  = _mm_sub_ps(b  [0], _mm_mul_ps(b  [5], tmp1));
        row[0]  = _mm_sub_ps(row[0], _mm_mul_ps(row[5], tmp1));

        tmp2    = _mm_load_ss(&src[0]);
        tmp2    = _mm_shuffle_ps(tmp2, tmp2, 0x00);

        b  [0]  = _mm_mul_ps(b  [0], tmp2);
        row[0]  = _mm_mul_ps(row[0], tmp2);

        n2      = (n2==0)*(n1-n2)+n2;

        tmp1    = row[ 1];      row[ 1] = row[n2];      row[n2] = tmp1;
        tmp2    = b  [ 1];      b  [ 1] = b  [n2];      b  [n2] = tmp2;

        tmp1    = row[ 0];      row[ 0] = row[n1];      row[n1] = tmp1;
        tmp2    = b  [ 0];      b  [ 0] = b  [n1];      b  [n1] = tmp2;

        _mm_storel_pi((__m64*)&src[ 0], b  [0]);
        _mm_storel_pi((__m64*)&src[ 2], row[0]);
        _mm_storeh_pi((__m64*)&src[ 4], row[0]);

        _mm_storel_pi((__m64*)&src[ 6], b  [1]);
        _mm_storel_pi((__m64*)&src[ 8], row[1]);
        _mm_storeh_pi((__m64*)&src[10], row[1]);

        _mm_storel_pi((__m64*)&src[12], b  [2]);
        _mm_storel_pi((__m64*)&src[14], row[2]);
        _mm_storeh_pi((__m64*)&src[16], row[2]);

        _mm_storel_pi((__m64*)&src[18], b  [3]);
        _mm_storel_pi((__m64*)&src[20], row[3]);
        _mm_storeh_pi((__m64*)&src[22], row[3]);

        _mm_storel_pi((__m64*)&src[24], b  [4]);
        _mm_storel_pi((__m64*)&src[26], row[4]);
        _mm_storeh_pi((__m64*)&src[28], row[4]);

        _mm_storel_pi((__m64*)&src[30], b  [5]);
        _mm_storel_pi((__m64*)&src[32], row[5]);
        _mm_storeh_pi((__m64*)&src[34], row[5]);
}
```

5.3 C Code in Special Case with Streaming SIMD Extensions

```
void  PIII_Invert_6x6_fast(float *src)
{
#define EPSILON           1e-8
#define   REAL_ZERO(x)    (((x)>-EPSILON && (x)<EPSILON) ? 1:0)

        __m128  minor0, minor1, minor2, minor3;
        __m128  det, tmp1, tmp2;
        __m128  b0, b1, b2, b3;
        __m128  row[6];

        static const unsigned long minus_hex      = 0x80000000;
        static const __m128       minus           = _mm_set_ps1(*(float*)&minus_hex);
        static const __m128       zero            = _mm_setzero_ps();
        static const __m128       e               = _mm_set_ps(1.0f, 0.0f, 0.0f, 1.0f);
        static const __m128       epsilon         = _mm_set_ss(EPSILON);
        static const __m128       epsilon1        = _mm_set_ss(-EPSILON);

        // Loading matrixes: 4x2 to row[0], row[1] and 4x4 to row[2]...row[5].

        tmp1    = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[12])), (__m64*)(&src[18]));
        tmp2    = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[24])), (__m64*)(&src[30]));

        row[0]  = _mm_shuffle_ps(tmp1, tmp2, 0x88);
        row[1]  = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

        tmp1    = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[14])), (__m64*)(&src[20]));
        tmp2    = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[26])), (__m64*)(&src[32]));
```

```
row[2]   = _mm_shuffle_ps(tmp1, tmp2, 0x88);
row[3]   = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

tmp1     = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)(&src[16])), (__m64*)(&src[22]));
tmp2     = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)(&src[28])), (__m64*)(&src[34]));

row[4]   = _mm_shuffle_ps(tmp1, tmp2, 0x88);
row[5]   = _mm_shuffle_ps(tmp1, tmp2, 0xDD);

//--------------

tmp2     = _mm_load_ss(&src[0]);
tmp1     = _mm_rcp_ss(tmp2);
tmp2     = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(tmp2, _mm_mul_ss(tmp1, tmp1)));

_mm_store_ss(&src[0], tmp2);

tmp2     = _mm_shuffle_ps(tmp2, tmp2, 0x00);
row[0]   = _mm_mul_ps(row[0], tmp2);

tmp1     = _mm_load_ss(&src[1]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[1]   = _mm_sub_ps(row[1], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[2]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[2]   = _mm_sub_ps(row[2], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[3]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[3]   = _mm_sub_ps(row[3], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[4]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[4]   = _mm_sub_ps(row[4], _mm_mul_ps(row[0], tmp1));

tmp1     = _mm_load_ss(&src[5]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[5]   = _mm_sub_ps(row[5], _mm_mul_ps(row[0], tmp1));

b0       = _mm_load_ss(&src[6]);
b0       = _mm_mul_ss(b0, tmp2);
_mm_store_ss(&src[6], b0);

tmp1     = _mm_load_ss(&src[1]);
tmp2     = _mm_load_ss(&src[7]);
tmp2     = _mm_sub_ss(tmp2, _mm_mul_ss(tmp1, b0));
_mm_store_ss(&src[7], tmp2);

b0       = _mm_shuffle_ps(b0, b0, 0x00);
tmp1     = _mm_loadh_pi(_mm_loadl_pi(tmp1, (__m64*)&src[2]), (__m64*)&src[ 4]);
tmp2     = _mm_loadh_pi(_mm_loadl_pi(tmp2, (__m64*)&src[8]), (__m64*)&src[10]);

tmp2     = _mm_sub_ps(tmp2, _mm_mul_ps(tmp1, b0));

_mm_storel_pi((__m64*)&src[ 8], tmp2);
_mm_storeh_pi((__m64*)&src[10], tmp2);

//--------------

tmp2     = _mm_load_ss(&src[7]);
tmp1     = _mm_rcp_ss(tmp2);
tmp2     = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(tmp2, _mm_mul_ss(tmp1, tmp1)));

_mm_store_ss(&src[7], tmp2);

tmp2     = _mm_shuffle_ps(tmp2, tmp2, 0x00);
row[1]   = _mm_mul_ps(row[1], tmp2);

row[0]   = _mm_sub_ps(row[0], _mm_mul_ps(row[1], b0));

tmp1     = _mm_load_ss(&src[8]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[2]   = _mm_sub_ps(row[2], _mm_mul_ps(row[1], tmp1));

tmp1     = _mm_load_ss(&src[9]);
tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);
row[3]   = _mm_sub_ps(row[3], _mm_mul_ps(row[1], tmp1));

tmp1     = _mm_load_ss(&src[10]);
```

```
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[4]  = _mm_sub_ps(row[4], _mm_mul_ps(row[1], tmp1));

        tmp1    = _mm_load_ss(&src[11]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);
        row[5]  = _mm_sub_ps(row[5], _mm_mul_ps(row[1], tmp1));

        row[0]  = _mm_xor_ps(row[0], minus);
        row[1]  = _mm_xor_ps(row[1], minus);

        row[3]  = _mm_shuffle_ps(row[3], row[3], 0x4E);
        row[5]  = _mm_shuffle_ps(row[5], row[5], 0x4E);

        // Inverting the matrix 4x4 by the Kramers method.

        tmp2    = _mm_mul_ps(row[4], row[5]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor0  = _mm_mul_ps(row[3], tmp1);
        minor1  = _mm_mul_ps(row[2], tmp1);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0  = _mm_sub_ps(_mm_mul_ps(row[3], tmp1), minor0);
        minor1  = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor1);
        minor1  = _mm_shuffle_ps(minor1, minor1, 0x4E);
//      -----------------------------------------------
        tmp2    = _mm_mul_ps(row[3], row[4]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor0  = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor0);
        minor3  = _mm_mul_ps(row[2], tmp1);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0  = _mm_sub_ps(minor0, _mm_mul_ps(row[5], tmp1));
        minor3  = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor3);
        minor3  = _mm_shuffle_ps(minor3, minor3, 0x4E);
//      -----------------------------------------------
        tmp2    = _mm_mul_ps(_mm_shuffle_ps(row[3], row[3], 0x4E), row[5]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);
        row[4]  = _mm_shuffle_ps(row[4], row[4], 0x4E);

        minor0  = _mm_add_ps(_mm_mul_ps(row[4], tmp1), minor0);
        minor2  = _mm_mul_ps(row[2], tmp1);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor0  = _mm_sub_ps(minor0, _mm_mul_ps(row[4], tmp1));
        minor2  = _mm_sub_ps(_mm_mul_ps(row[2], tmp1), minor2);
        minor2  = _mm_shuffle_ps(minor2, minor2, 0x4E);
//      -----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[3]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor2  = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor2);
        minor3  = _mm_sub_ps(_mm_mul_ps(row[4], tmp1), minor3);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor2  = _mm_sub_ps(_mm_mul_ps(row[5], tmp1), minor2);
        minor3  = _mm_sub_ps(minor3, _mm_mul_ps(row[4], tmp1));
//      -----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[5]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor1  = _mm_sub_ps(minor1, _mm_mul_ps(row[4], tmp1));
        minor2  = _mm_add_ps(_mm_mul_ps(row[3], tmp1), minor2);

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);

        minor1  = _mm_add_ps(_mm_mul_ps(row[4], tmp1), minor1);
        minor2  = _mm_sub_ps(minor2, _mm_mul_ps(row[3], tmp1));
//      -----------------------------------------------
        tmp2    = _mm_mul_ps(row[2], row[4]);
        tmp1    = _mm_shuffle_ps(tmp2, tmp2, 0xB1);

        minor1  = _mm_add_ps(_mm_mul_ps(row[5], tmp1), minor1);
        minor3  = _mm_sub_ps(minor3, _mm_mul_ps(row[3], tmp1));

        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x4E);
```

```
        minor1   = _mm_sub_ps(minor1, _mm_mul_ps(row[5], tmp1));
        minor3   = _mm_add_ps(_mm_mul_ps(row[3], tmp1), minor3);
//      ----------------------------------------------
        det      = _mm_mul_ps(row[2], minor0);
        det      = _mm_add_ps(_mm_shuffle_ps(det, det, 0x4E), det);
        det      = _mm_add_ss(_mm_shuffle_ps(det, det, 0xB1), det);

        if(_mm_movemask_ps(_mm_and_ps(_mm_cmplt_ss(det, epsilon), _mm_cmpgt_ss(det, epsilon1))) & 1)
                return;

        tmp1     = _mm_rcp_ss(det);
        det      = _mm_sub_ss(_mm_add_ss(tmp1, tmp1), _mm_mul_ss(det, _mm_mul_ss(tmp1, tmp1)));
        det      = _mm_shuffle_ps(det, det, 0x00);

        row[2]   = _mm_mul_ps(det, minor0);
        row[3]   = _mm_mul_ps(det, minor1);
        row[4]   = _mm_mul_ps(det, minor2);
        row[5]   = _mm_mul_ps(det, minor3);

        b0       = _mm_unpacklo_ps(row[0], row[1]);
        b2       = _mm_unpackhi_ps(row[0], row[1]);
        b1       = _mm_shuffle_ps(b0, b2, 0x4E);
        b3       = _mm_shuffle_ps(b2, b0, 0x4E);

        tmp1     = _mm_shuffle_ps(row[2], row[3], 0x50);
        tmp2     = _mm_mul_ps(b0, tmp1);

        tmp1     = _mm_shuffle_ps(row[2], row[3], 0xA5);
        tmp2     = _mm_add_ps(tmp2, _mm_mul_ps(b1, tmp1));

        tmp1     = _mm_shuffle_ps(row[2], row[3], 0xFA);
        tmp2     = _mm_add_ps(tmp2, _mm_mul_ps(b2, tmp1));

        tmp1     = _mm_shuffle_ps(row[2], row[3], 0x0F);
        row[0]   = _mm_add_ps(tmp2, _mm_mul_ps(b3, tmp1));

        tmp1     = _mm_shuffle_ps(row[4], row[5], 0x50);
        tmp2     = _mm_mul_ps(b0, tmp1);

        tmp1     = _mm_shuffle_ps(row[4], row[5], 0xA5);
        tmp2     = _mm_add_ps(tmp2, _mm_mul_ps(b1, tmp1));

        tmp1     = _mm_shuffle_ps(row[4], row[5], 0xFA);
        tmp2     = _mm_add_ps(tmp2, _mm_mul_ps(b2, tmp1));

        tmp1     = _mm_shuffle_ps(row[4], row[5], 0x0F);
        row[1]   = _mm_add_ps(tmp2, _mm_mul_ps(b3, tmp1));

        // Calculating row number 1

        b0       = e;
        tmp1     = _mm_load_ss(&src[8]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0       = _mm_sub_ps(b0, _mm_mul_ps(_mm_shuffle_ps(row[0], row[0], 0x4E), tmp1));
        b1       = _mm_xor_ps(_mm_mul_ps(row[2], tmp1), minus);

        tmp1     = _mm_load_ss(&src[9]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0       = _mm_sub_ps(b0, _mm_mul_ps(row[0], tmp1));
        b1       = _mm_sub_ps(b1, _mm_mul_ps(row[3], tmp1));

        tmp1     = _mm_load_ss(&src[10]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0       = _mm_sub_ps(b0, _mm_mul_ps(_mm_shuffle_ps(row[1], row[1], 0x4E), tmp1));
        b1       = _mm_sub_ps(b1, _mm_mul_ps(row[4], tmp1));

        tmp1     = _mm_load_ss(&src[11]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0       = _mm_sub_ps(b0, _mm_mul_ps(row[1], tmp1));
        b1       = _mm_sub_ps(b1, _mm_mul_ps(row[5], tmp1));

        tmp1     = _mm_load_ss(&src[6]);
        tmp1     = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0       = _mm_sub_ps(b0, _mm_mul_ps(_mm_shuffle_ps(e, e, 0x4E), tmp1));

        tmp2     = _mm_load_ss(&src[7]);
```

```
        tmp2    = _mm_shuffle_ps(tmp2, tmp2, 0x00);

        b0      = _mm_mul_ps(b0, tmp2);
        b1      = _mm_mul_ps(b1, tmp2);

        _mm_storeh_pi((__m64*)&src[ 6], b0);
        _mm_storel_pi((__m64*)&src[ 8], b1);
        _mm_storeh_pi((__m64*)&src[10], b1);

        // Calculating row number 0

        tmp1    = _mm_load_ss(&src[1]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0      = _mm_sub_ps(e, _mm_mul_ps(_mm_shuffle_ps(b0, b0, 0x4E), tmp1));
        b1      = _mm_xor_ps(_mm_mul_ps(b1, tmp1), minus);

        tmp1    = _mm_load_ss(&src[2]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0      = _mm_sub_ps(b0, _mm_mul_ps(row[0], tmp1));
        b1      = _mm_sub_ps(b1, _mm_mul_ps(row[2], tmp1));

        tmp1    = _mm_load_ss(&src[3]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0      = _mm_sub_ps(b0, _mm_mul_ps(_mm_shuffle_ps(row[0], row[0], 0x4E), tmp1));
        b1      = _mm_sub_ps(b1, _mm_mul_ps(row[3], tmp1));

        tmp1    = _mm_load_ss(&src[4]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0      = _mm_sub_ps(b0, _mm_mul_ps(row[1], tmp1));
        b1      = _mm_sub_ps(b1, _mm_mul_ps(row[4], tmp1));

        tmp1    = _mm_load_ss(&src[5]);
        tmp1    = _mm_shuffle_ps(tmp1, tmp1, 0x00);

        b0      = _mm_sub_ps(b0, _mm_mul_ps(_mm_shuffle_ps(row[1], row[1], 0x4E), tmp1));
        b1      = _mm_sub_ps(b1, _mm_mul_ps(row[5], tmp1));

        tmp2    = _mm_load_ss(&src[0]);
        tmp2    = _mm_shuffle_ps(tmp2, tmp2, 0x00);

        b0      = _mm_mul_ps(b0, tmp2);
        b1      = _mm_mul_ps(b1, tmp2);

        _mm_storel_pi((__m64*)&src[0], b0);
        _mm_storel_pi((__m64*)&src[2], b1);
        _mm_storeh_pi((__m64*)&src[4], b1);

        _mm_storel_pi((__m64*)&src[12], row[0]);
        _mm_storel_pi((__m64*)&src[14], row[2]);
        _mm_storeh_pi((__m64*)&src[16], row[2]);

        _mm_storeh_pi((__m64*)&src[18], row[0]);
        _mm_storel_pi((__m64*)&src[20], row[3]);
        _mm_storeh_pi((__m64*)&src[22], row[3]);

        _mm_storel_pi((__m64*)&src[24], row[1]);
        _mm_storel_pi((__m64*)&src[26], row[4]);
        _mm_storeh_pi((__m64*)&src[28], row[4]);

        _mm_storeh_pi((__m64*)&src[30], row[1]);
        _mm_storel_pi((__m64*)&src[32], row[5]);
        _mm_storeh_pi((__m64*)&src[34], row[5]);

}
```