Технология CUDA для высокопроизводительных вычислений на кластерах с графическими процессорами

Колганов Александр

alexander.k.s@mail.ru

Инкрементальное распараллеливание

- Поэтапное распараллеливание программы;

- Применяется только для общей памяти, таким образом идеально подходит как для ГПУ (CUDA, OpenACC, OpenCL), так и для ЦПУ (OpenMP);

- Не меняется структура программы.

- Единственность кода - нет необходимости поддерживать последовательный и параллельный вариант программы в случае использования директив (OpenMP, OpenACC).

Этапы инкрементального распараллеливания

- Запуск последовательной программы, получение ее профилировочной информации с помощью различных инструментов (intel, gcc, google);

- Определение времяемких участков кода и их дальнейшее изучение;

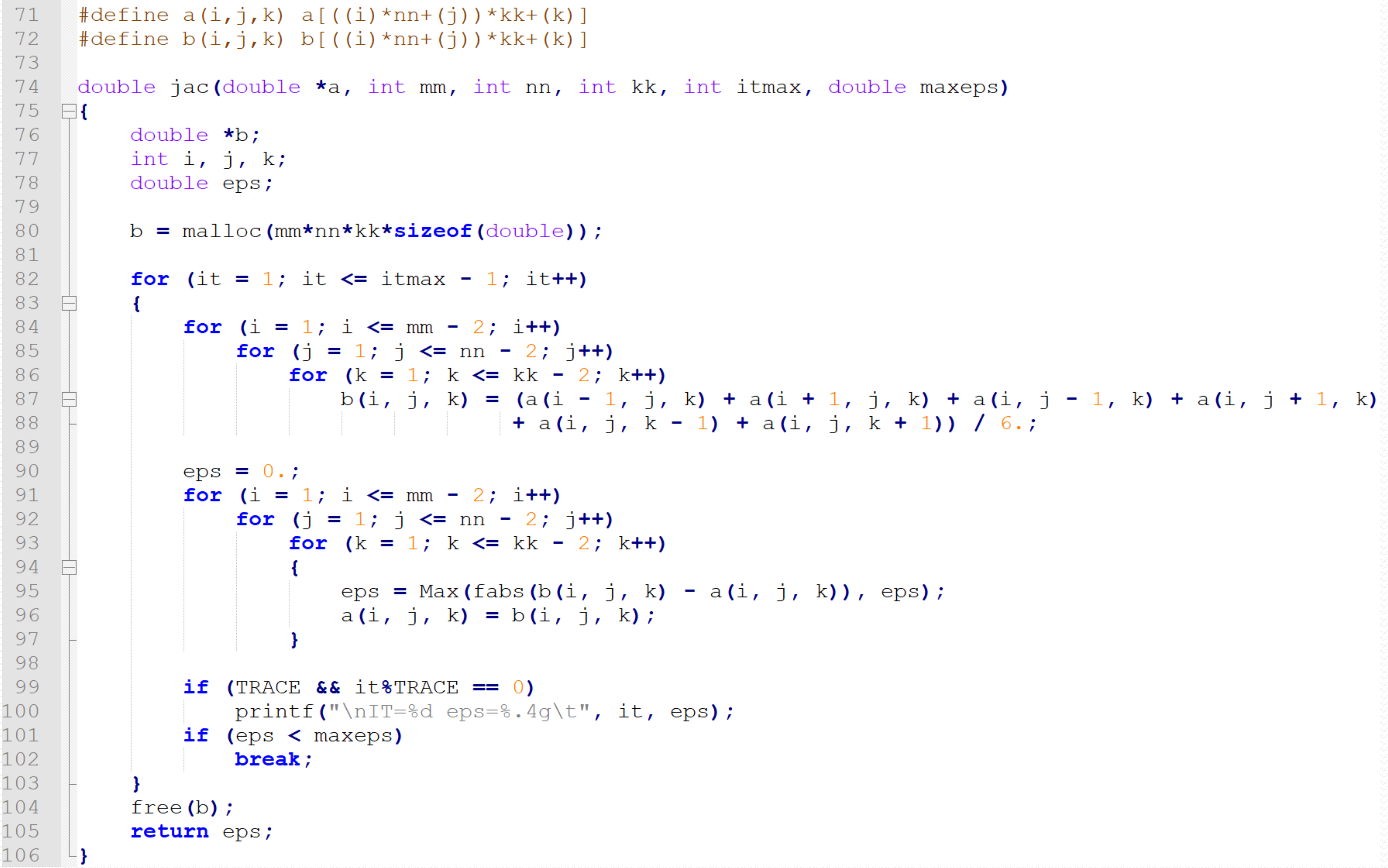
- Распараллеливание времяемких участвок кода с помощью целевых библиотек (CUDA, OpenMP …);

- Распараллеливание всей программы.

Использование компилятора Nvidia CUDA (nvcc)

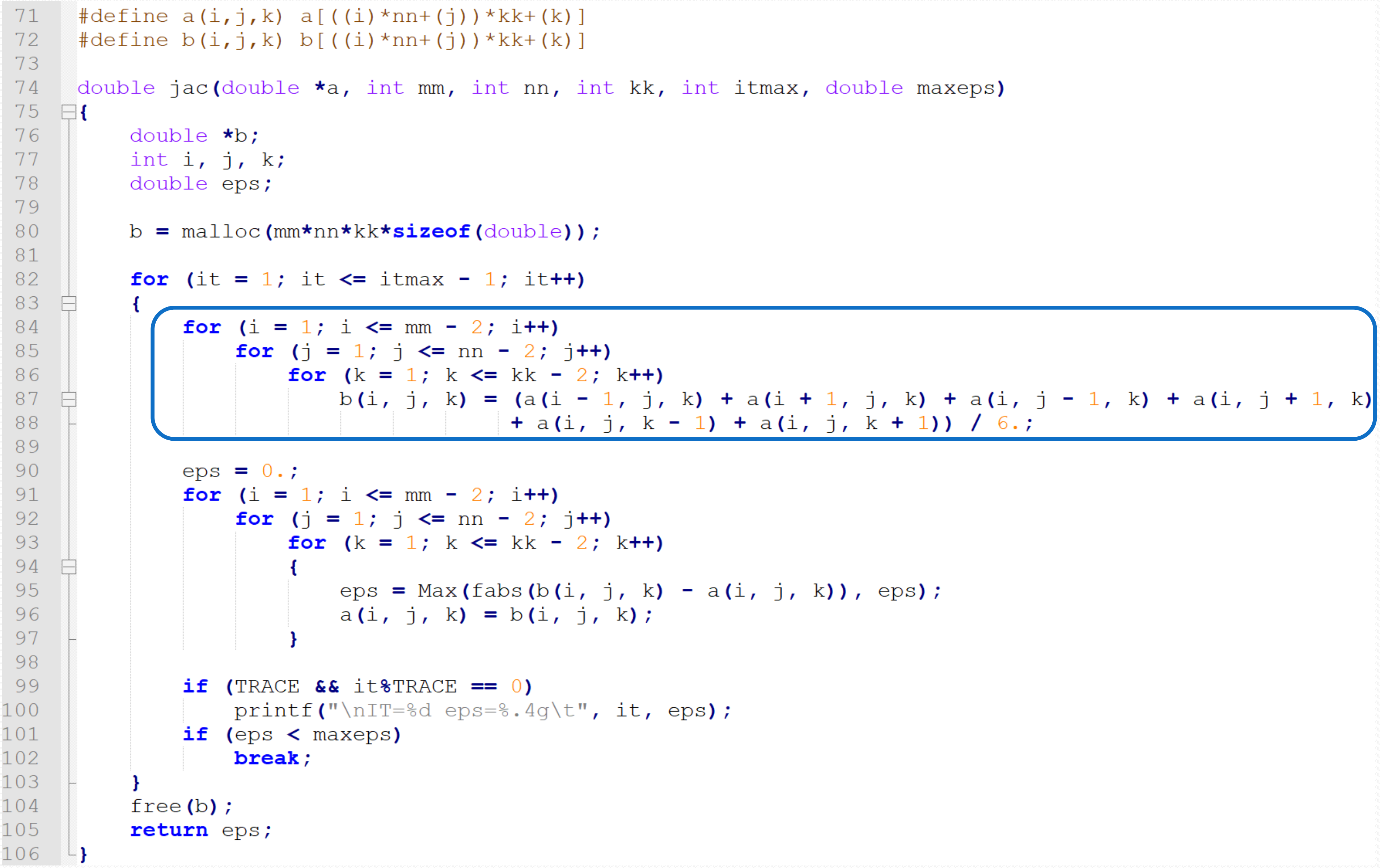
- Пробуем в Makefile использовать **NVCC** вместо **GCC:**
  - CC = **gcc** -> CC = **nvcc.** После данной замены не нужно прописывать пути к inclide и lib для CUDA. Но проект по прежнему компилируется **GCC**.
- Для того, чтобы добавить **__global__** функции, необходимы файлы с расширением .cu. Для этого достаточно установить опцию компилятору:
  - CFLAGS = ~~-Wall~~ -O3 -I${INC} **-x cu**
- Для использования архиректуры Kepler устанавлвиаем еще одну опцию:
  - CFLAGS = ~~-Wall~~ -O3 -I${INC} -x cu **-arch=sm_35**
  - CLINKFLAGS= -O3 **-arch=sm_35**
- Для использования опций для хоста:
  - CFLAGS = **-Xcompiler -Wall** -O3 -I${INC} -x cu -arch=sm_35

Распараллеливание Якоби



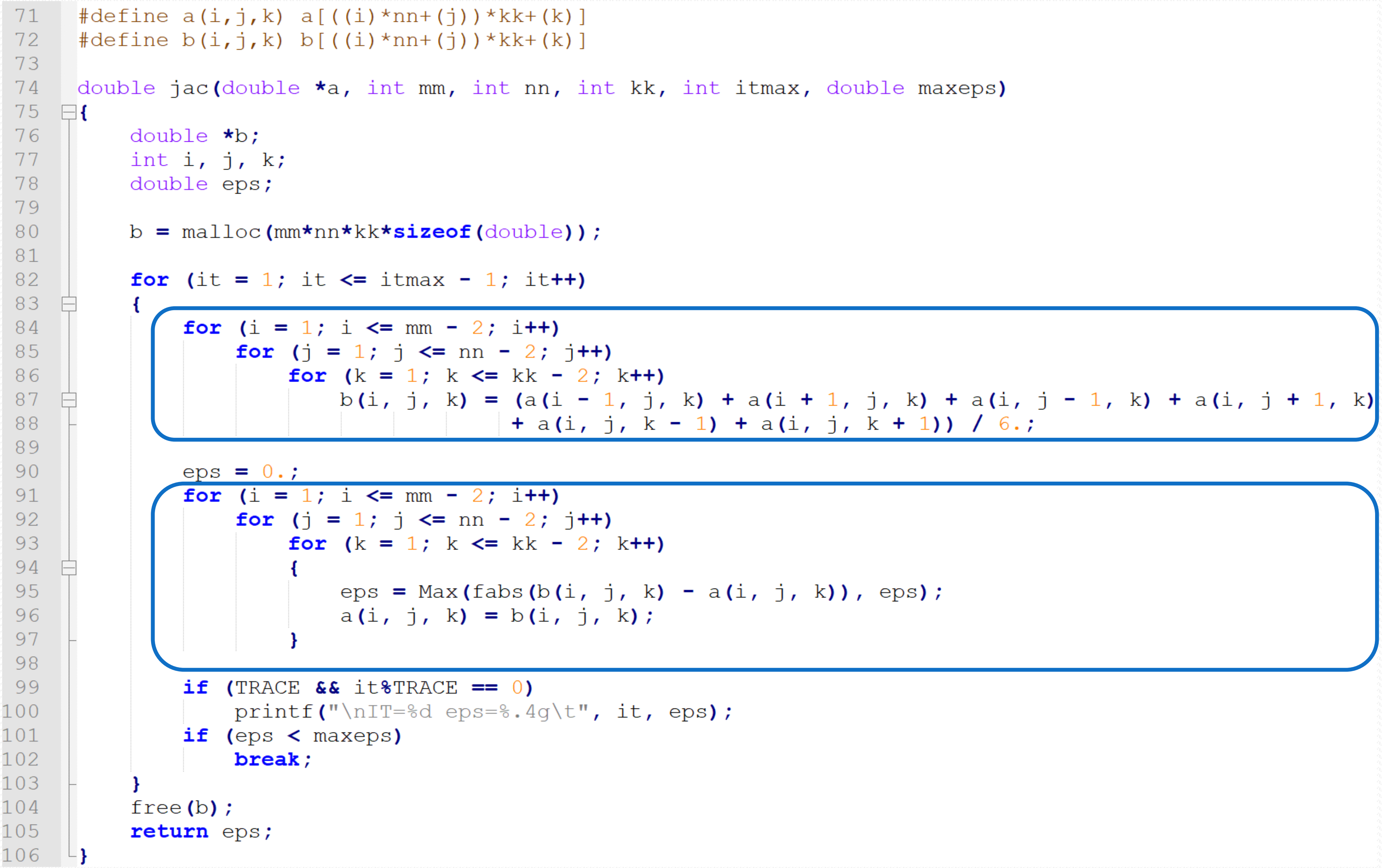
```c
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                    b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) + a(i, j + 1, k)
                                 + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



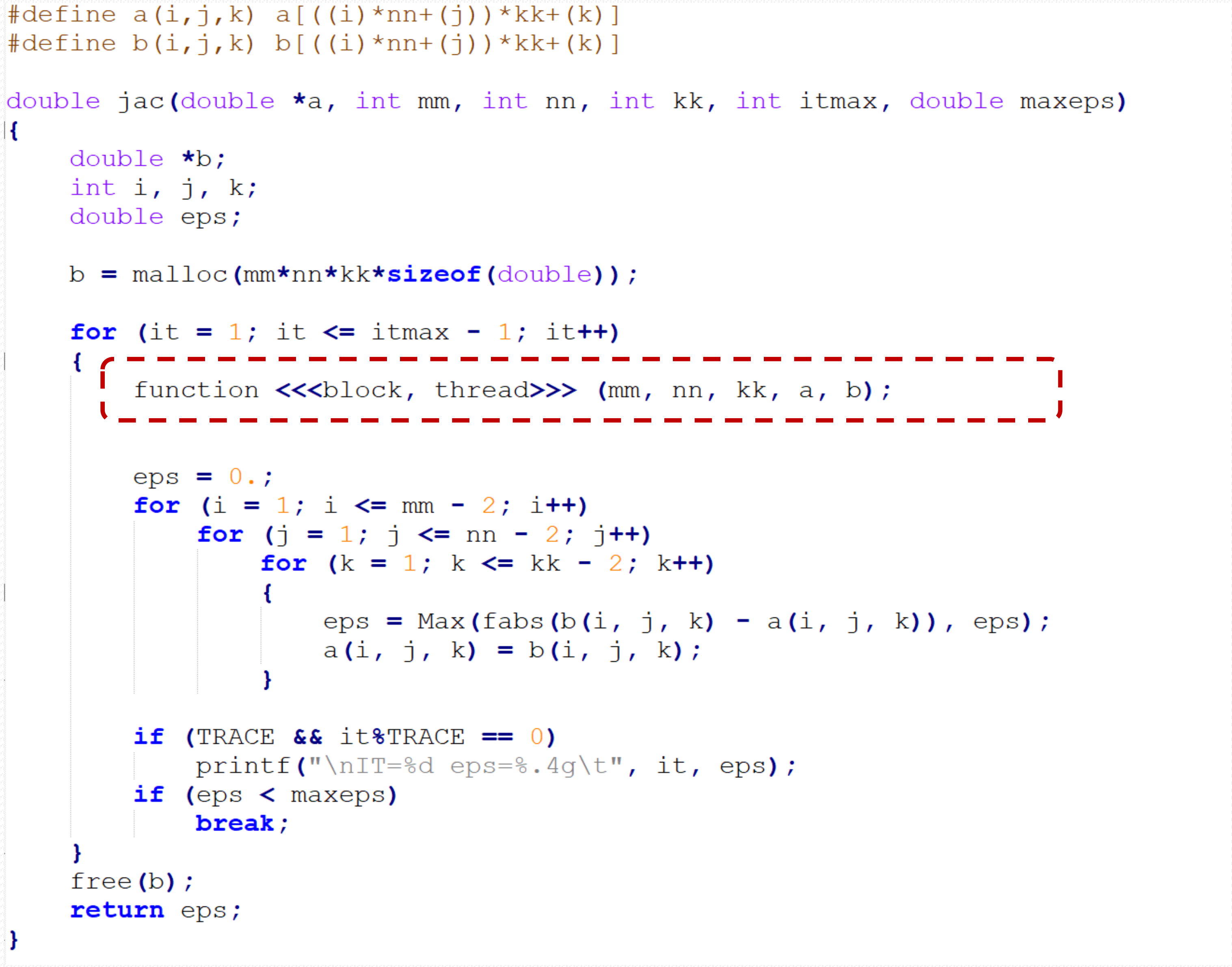
```c
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                    b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) + a(i, j + 1, k)
                                 + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



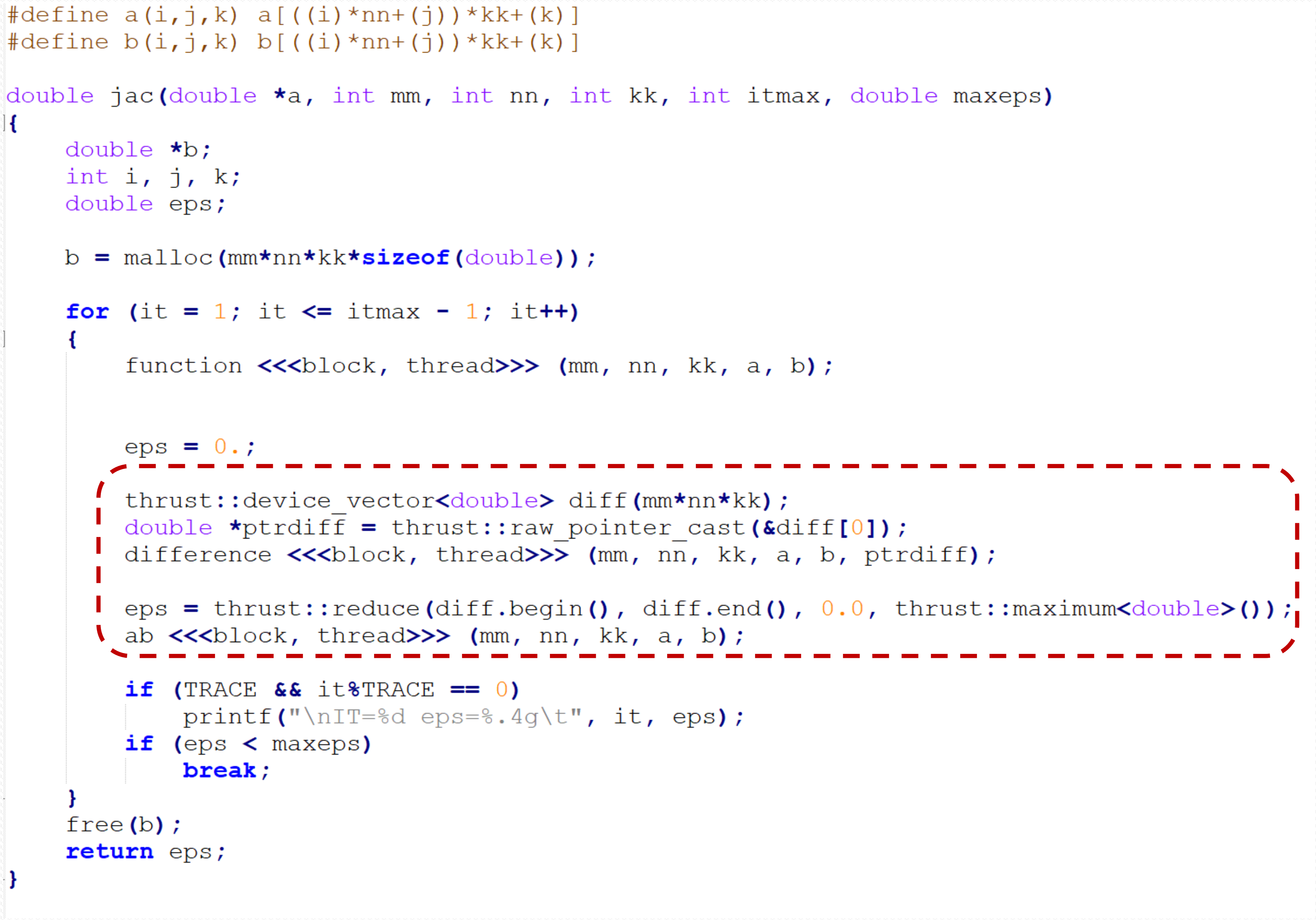
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                    b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) + a(i, j + 1, k)
                                 + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



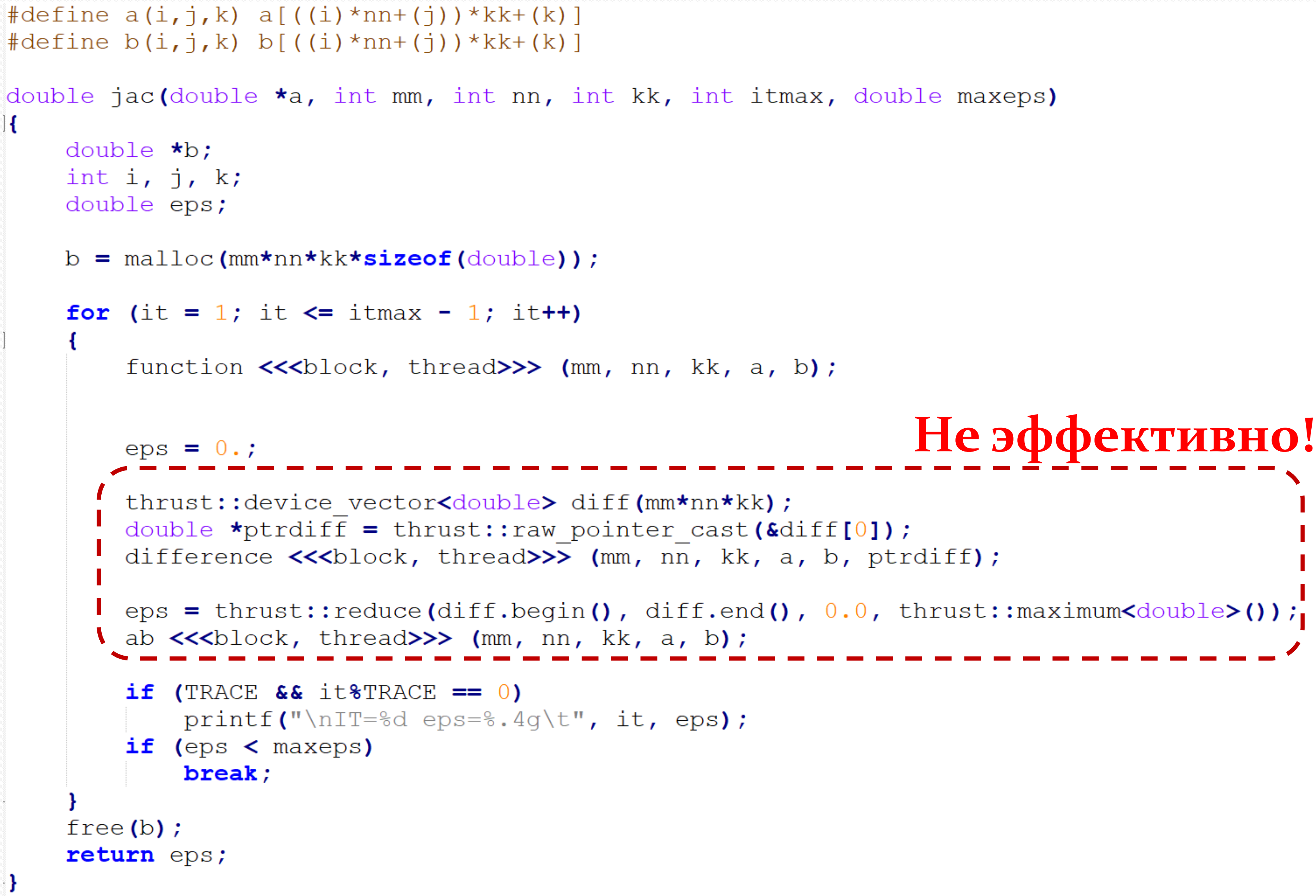
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

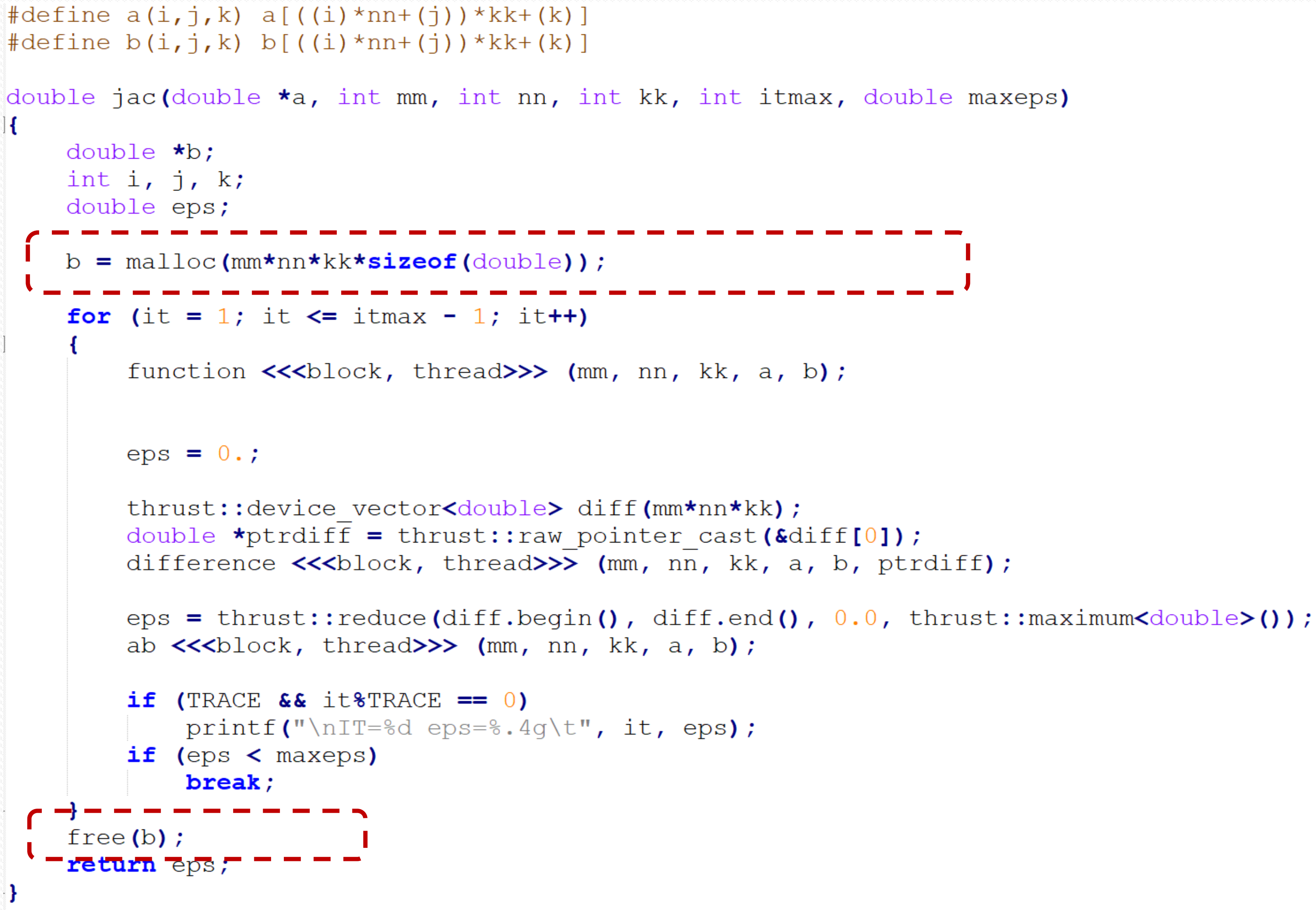
Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

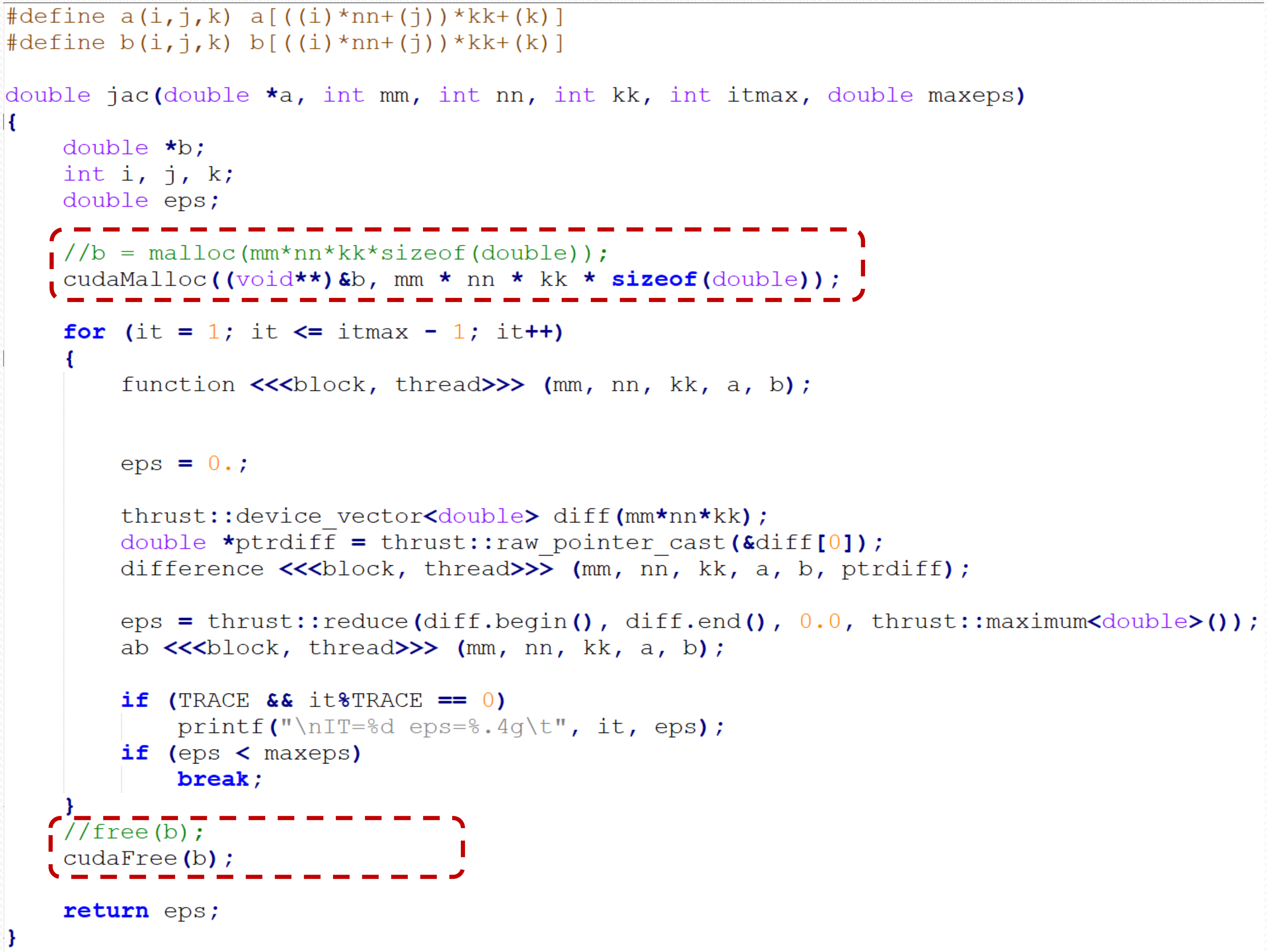
Не эффективно!

Распараллеливание Якоби



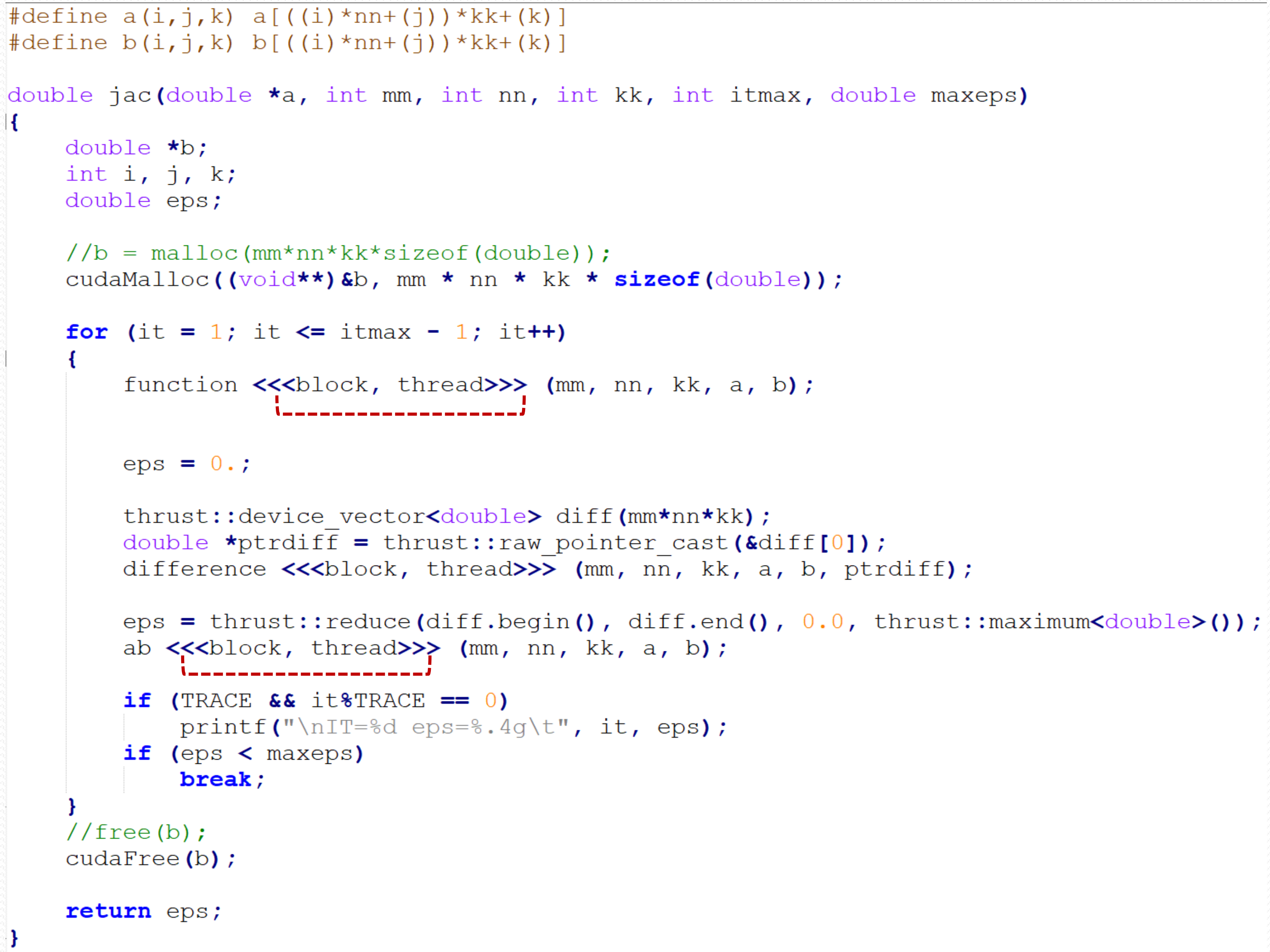
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



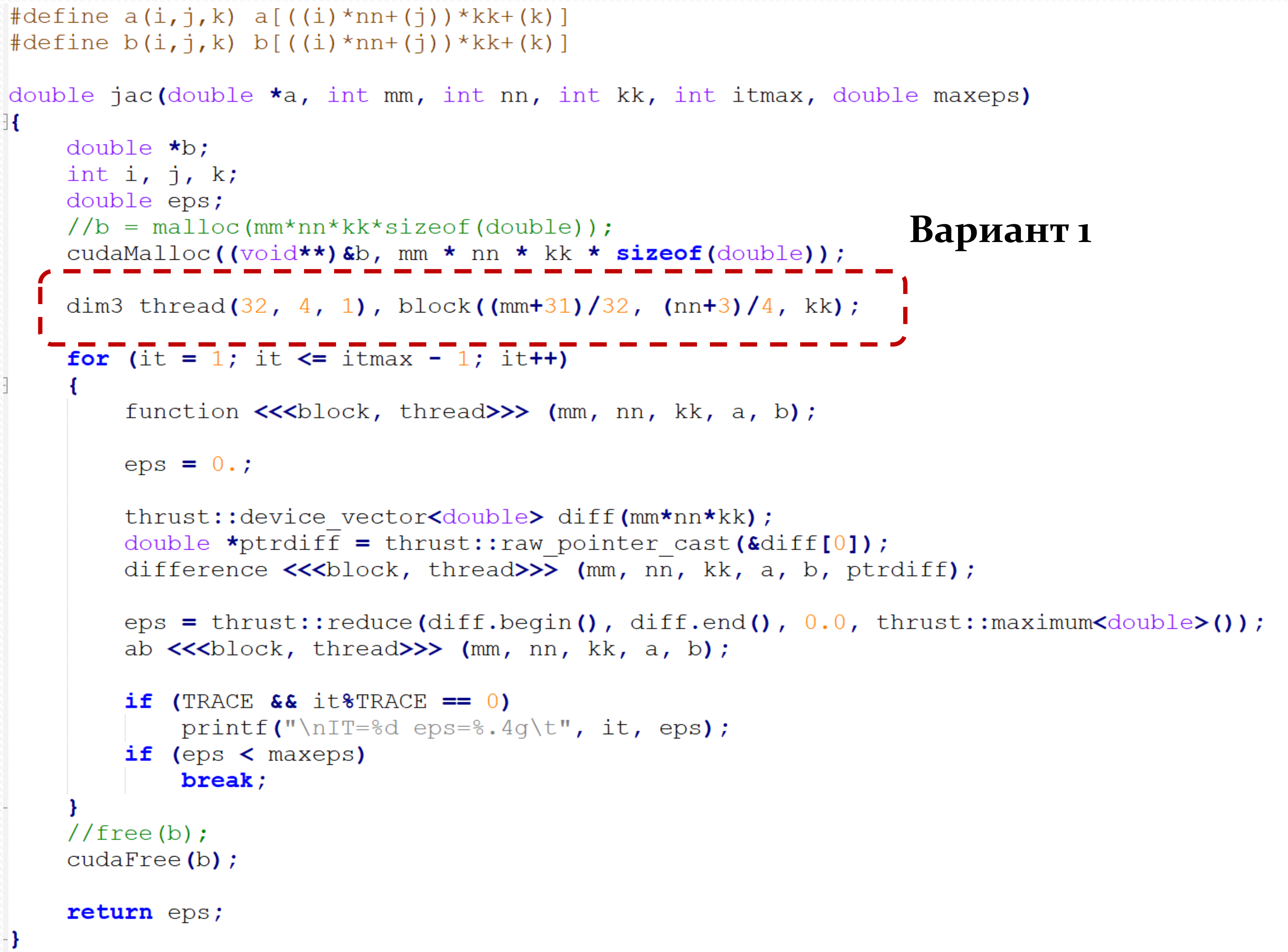
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

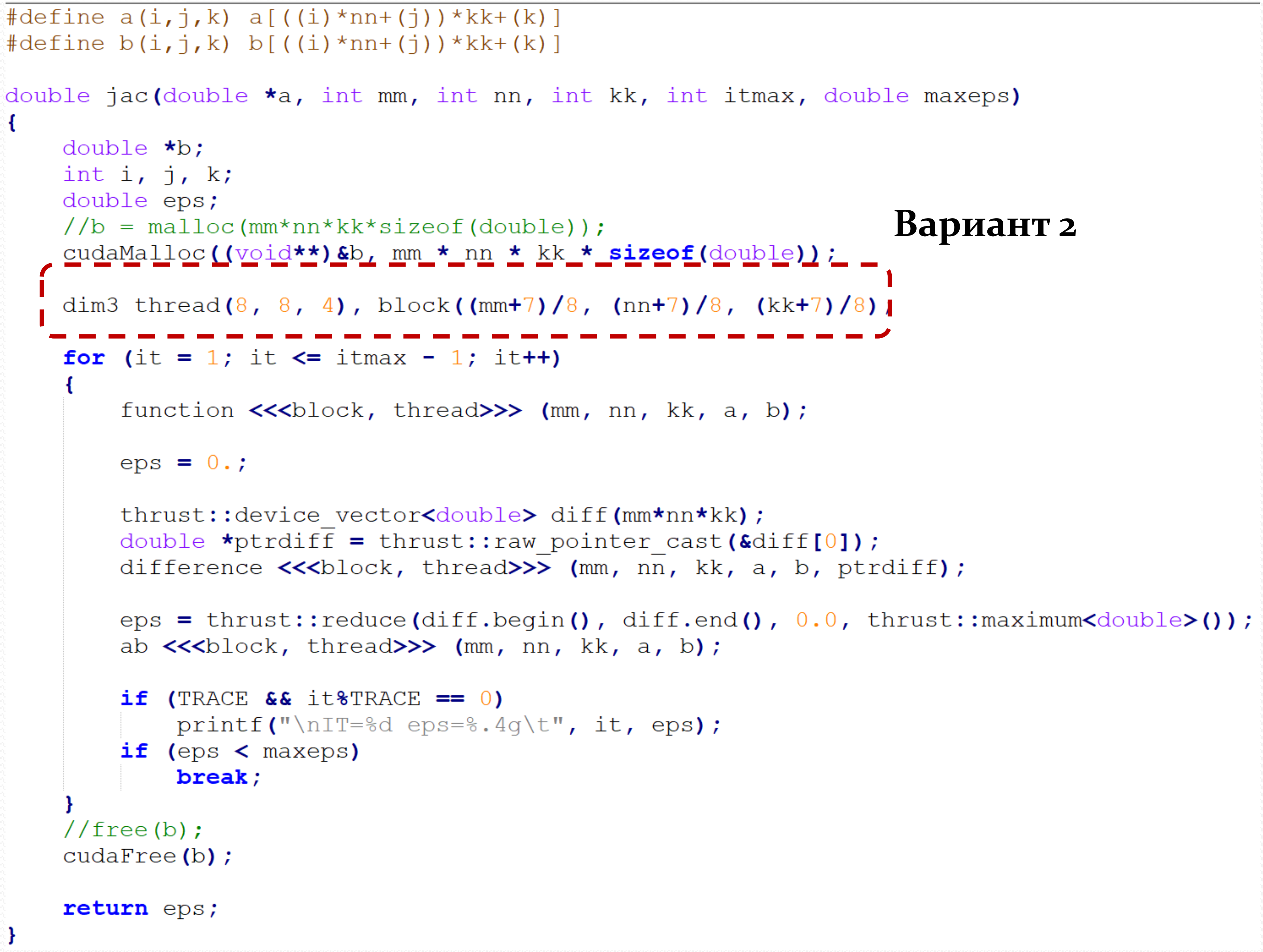
Распараллеливание Якоби



Вариант 1

```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;
    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    dim3 thread(32, 4, 1), block((mm+31)/32, (nn+3)/4, kk);

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);

        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

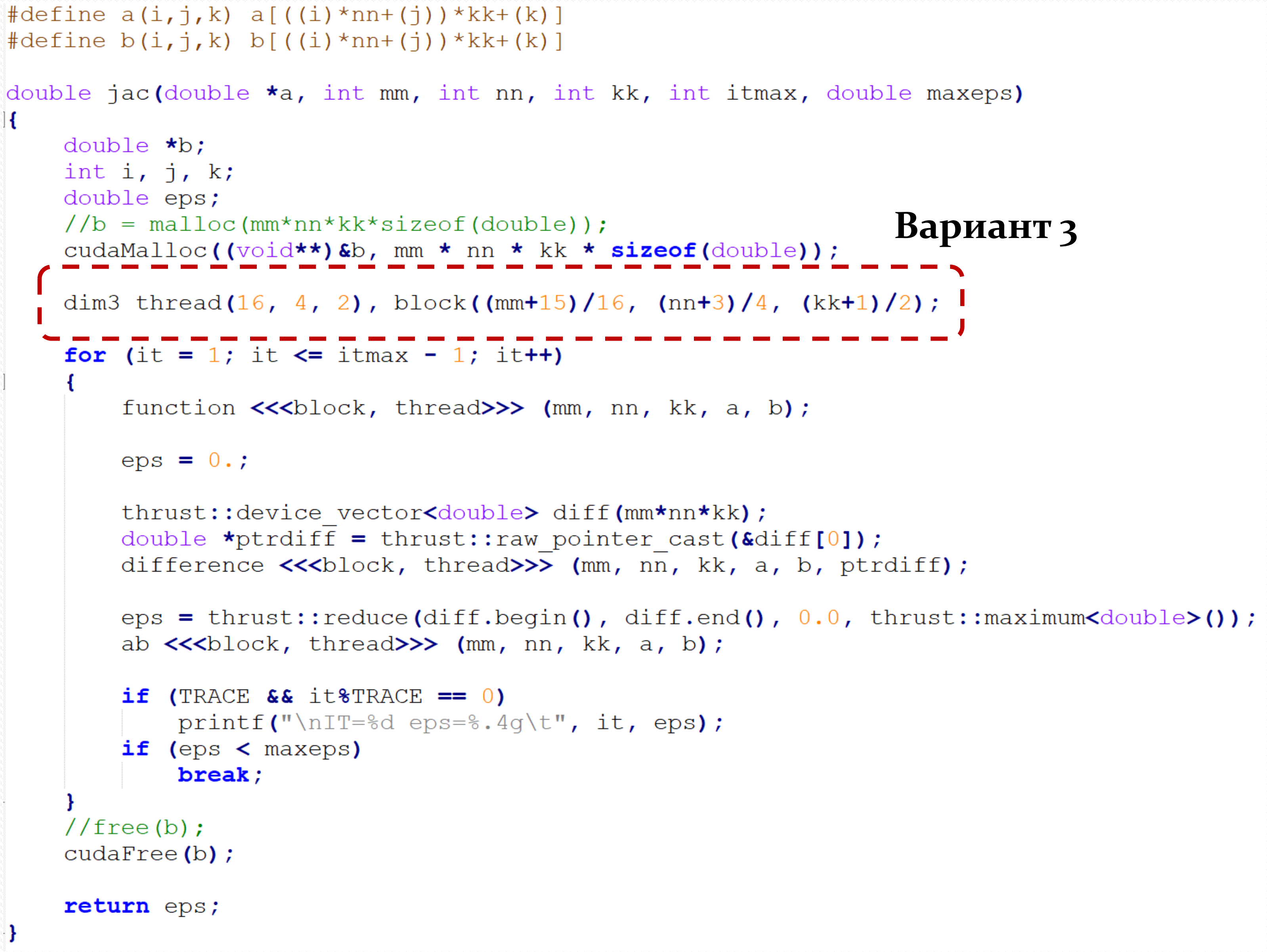
Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;
    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    dim3 thread(8, 8, 4), block((mm+7)/8, (nn+7)/8, (kk+7)/8);

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);

        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

Вариант 2

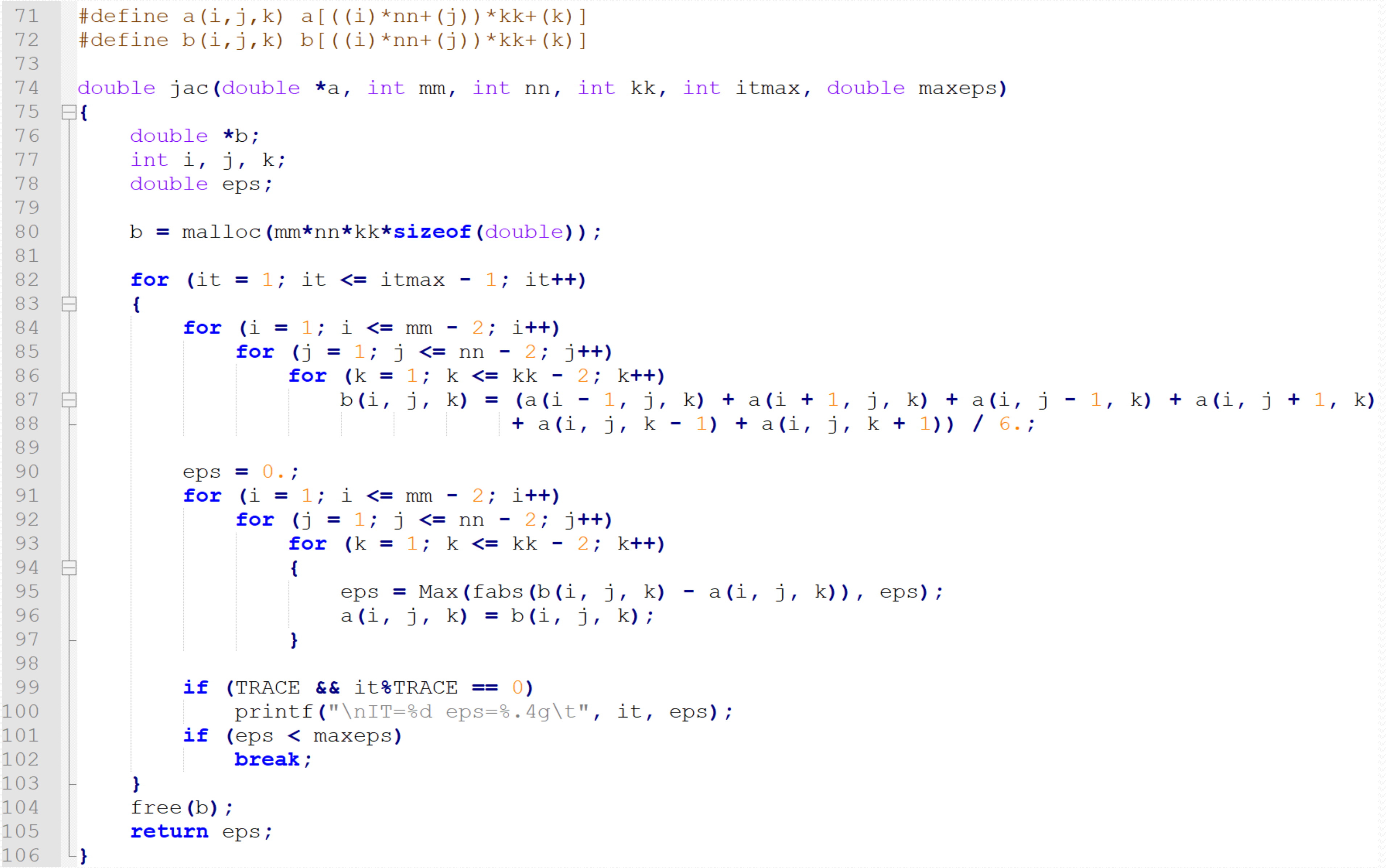
Распараллеливание Якоби



Вариант 3

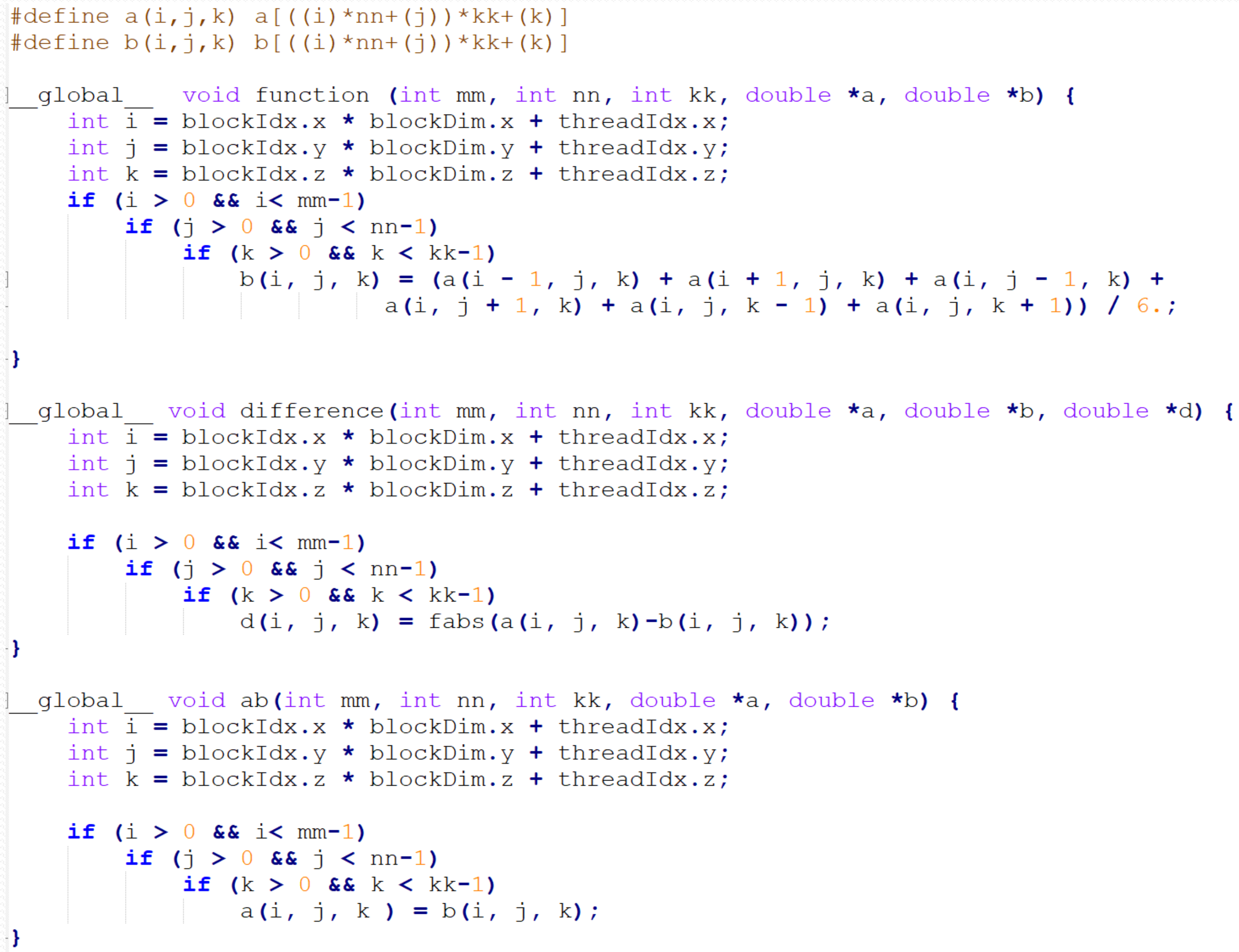
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;
    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    dim3 thread(16, 4, 2), block((mm+15)/16, (nn+3)/4, (kk+1)/2);

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);

        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

Распараллеливание Якоби



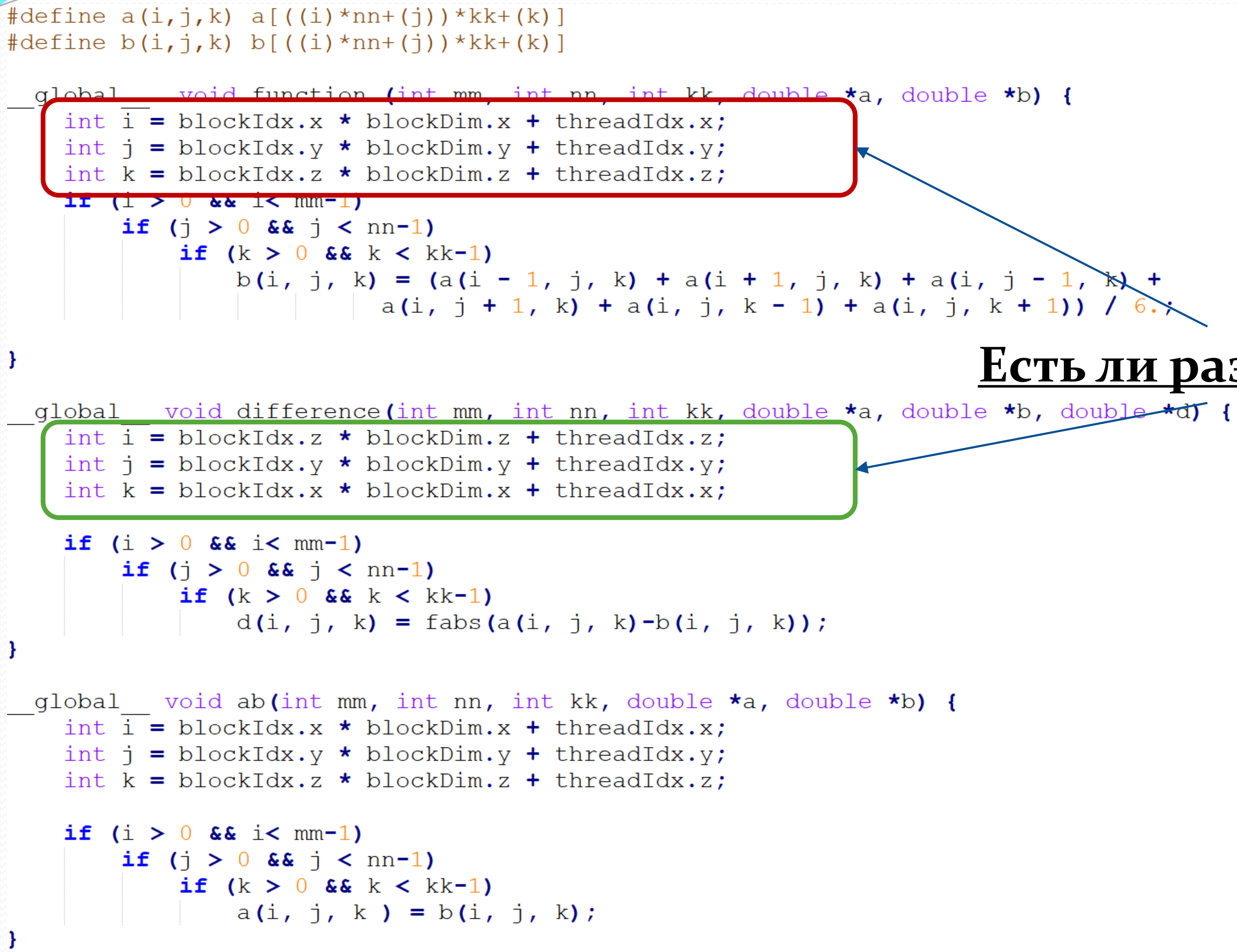
```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                    b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) + a(i, j + 1, k)
                                   + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

__global__ void function (int mm, int nn, int kk, double *a, double *b) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.z * blockDim.z + threadIdx.z;
    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) +
                          a(i, j + 1, k) + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

}

__global__ void difference(int mm, int nn, int kk, double *a, double *b, double *d) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.z * blockDim.z + threadIdx.z;

    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                d(i, j, k) = fabs(a(i, j, k)-b(i, j, k));
}

__global__ void ab(int mm, int nn, int kk, double *a, double *b) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.z * blockDim.z + threadIdx.z;

    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                a(i, j, k ) = b(i, j, k);
}
```

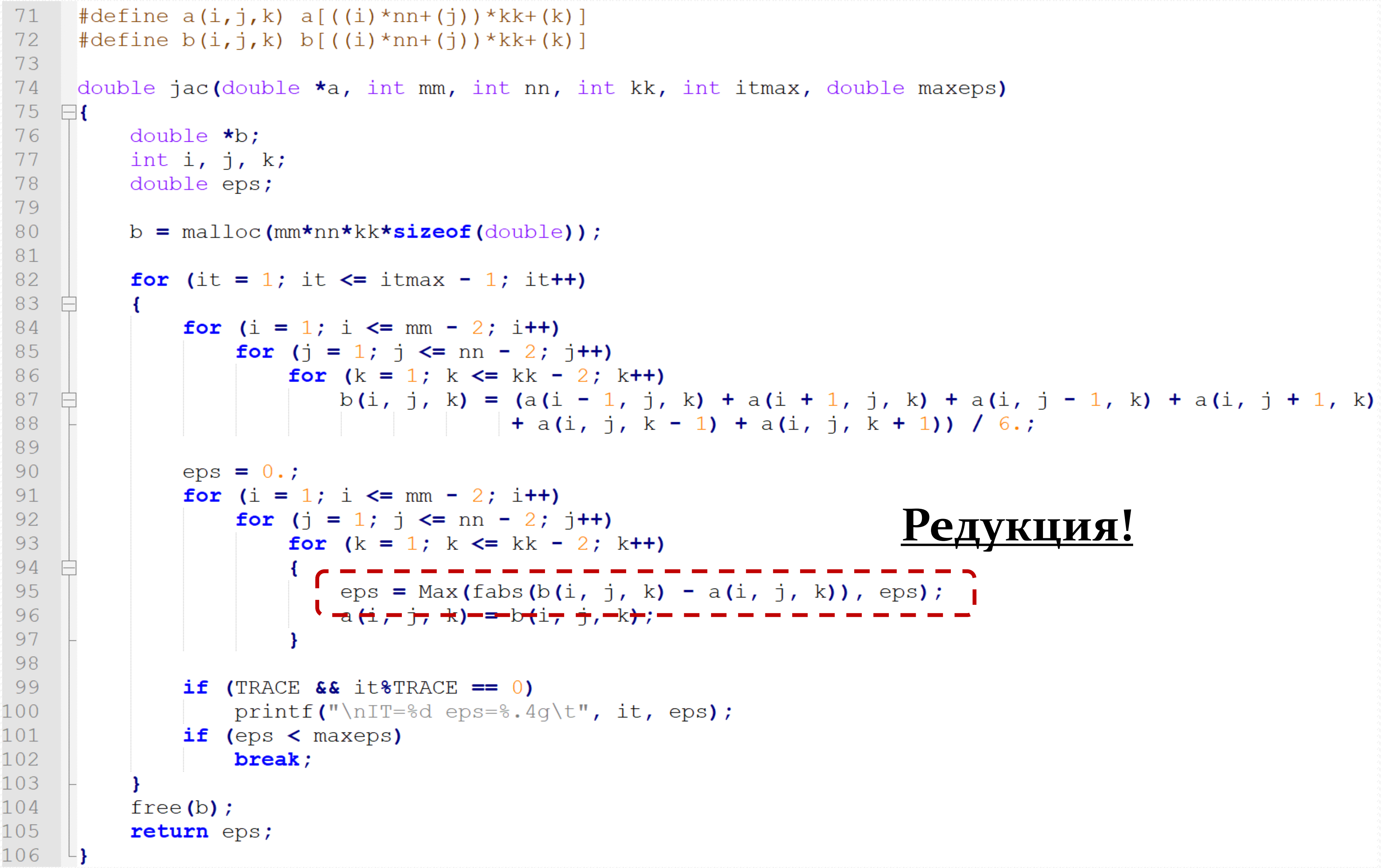
Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

__global__ void function (int mm, int nn, int kk, double *a, double *b) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.z * blockDim.z + threadIdx.z;
    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) +
                          a(i, j + 1, k) + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

}

__global__ void difference(int mm, int nn, int kk, double *a, double *b, double *d) {
    int i = blockIdx.z * blockDim.z + threadIdx.z;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.x * blockDim.x + threadIdx.x;

    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                d(i, j, k) = fabs(a(i, j, k)-b(i, j, k));
}

__global__ void ab(int mm, int nn, int kk, double *a, double *b) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int k = blockIdx.z * blockDim.z + threadIdx.z;

    if (i > 0 && i< mm-1)
        if (j > 0 && j < nn-1)
            if (k > 0 && k < kk-1)
                a(i, j, k ) = b(i, j, k);
}
```

Есть ли разница?

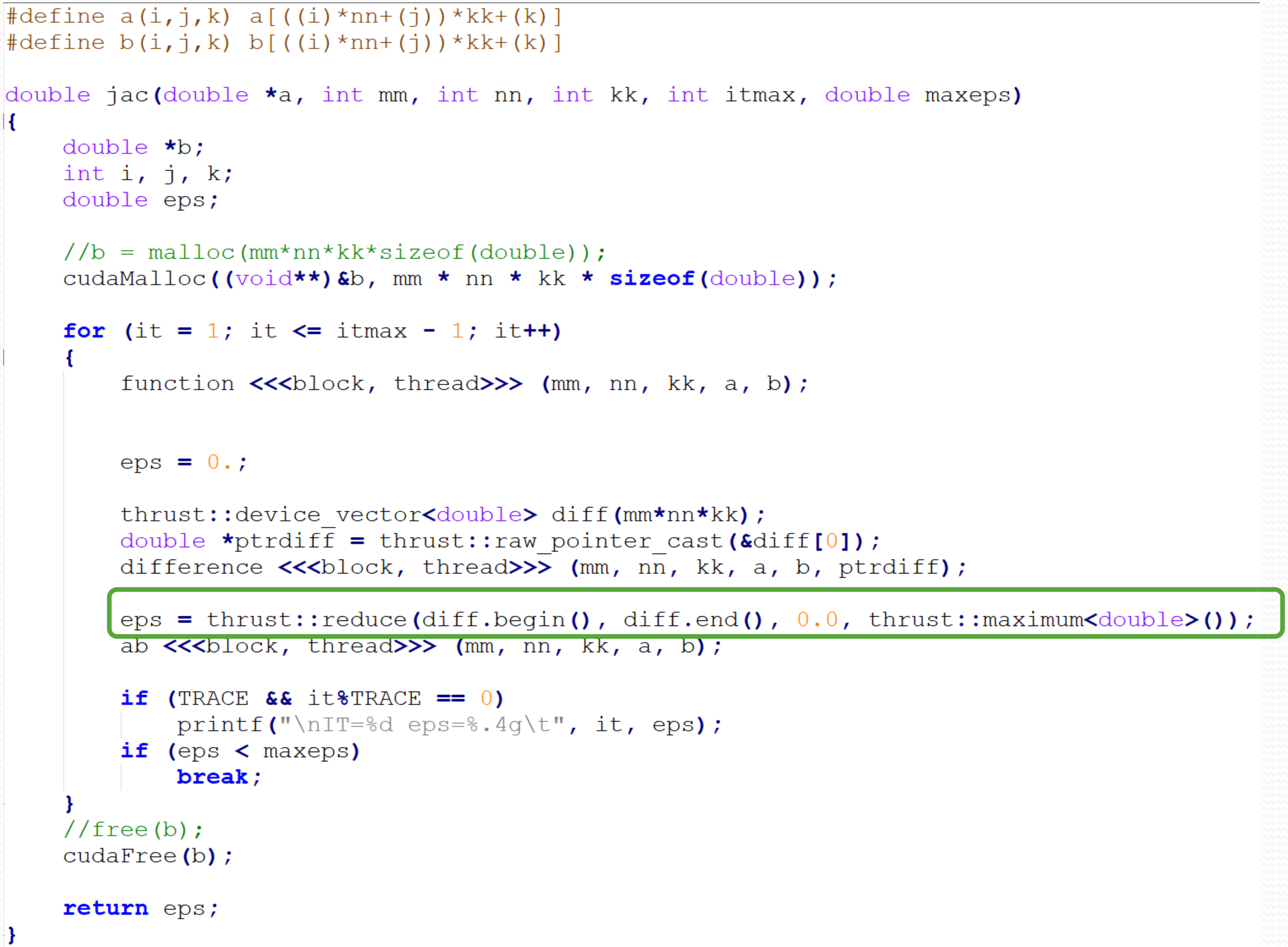
Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    b = malloc(mm*nn*kk*sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                    b(i, j, k) = (a(i - 1, j, k) + a(i + 1, j, k) + a(i, j - 1, k) + a(i, j + 1, k)
                                 + a(i, j, k - 1) + a(i, j, k + 1)) / 6.;

        eps = 0.;
        for (i = 1; i <= mm - 2; i++)
            for (j = 1; j <= nn - 2; j++)
                for (k = 1; k <= kk - 2; k++)
                {
                    eps = Max(fabs(b(i, j, k) - a(i, j, k)), eps);
                    a(i, j, k) = b(i, j, k);
                }

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    free(b);
    return eps;
}
```

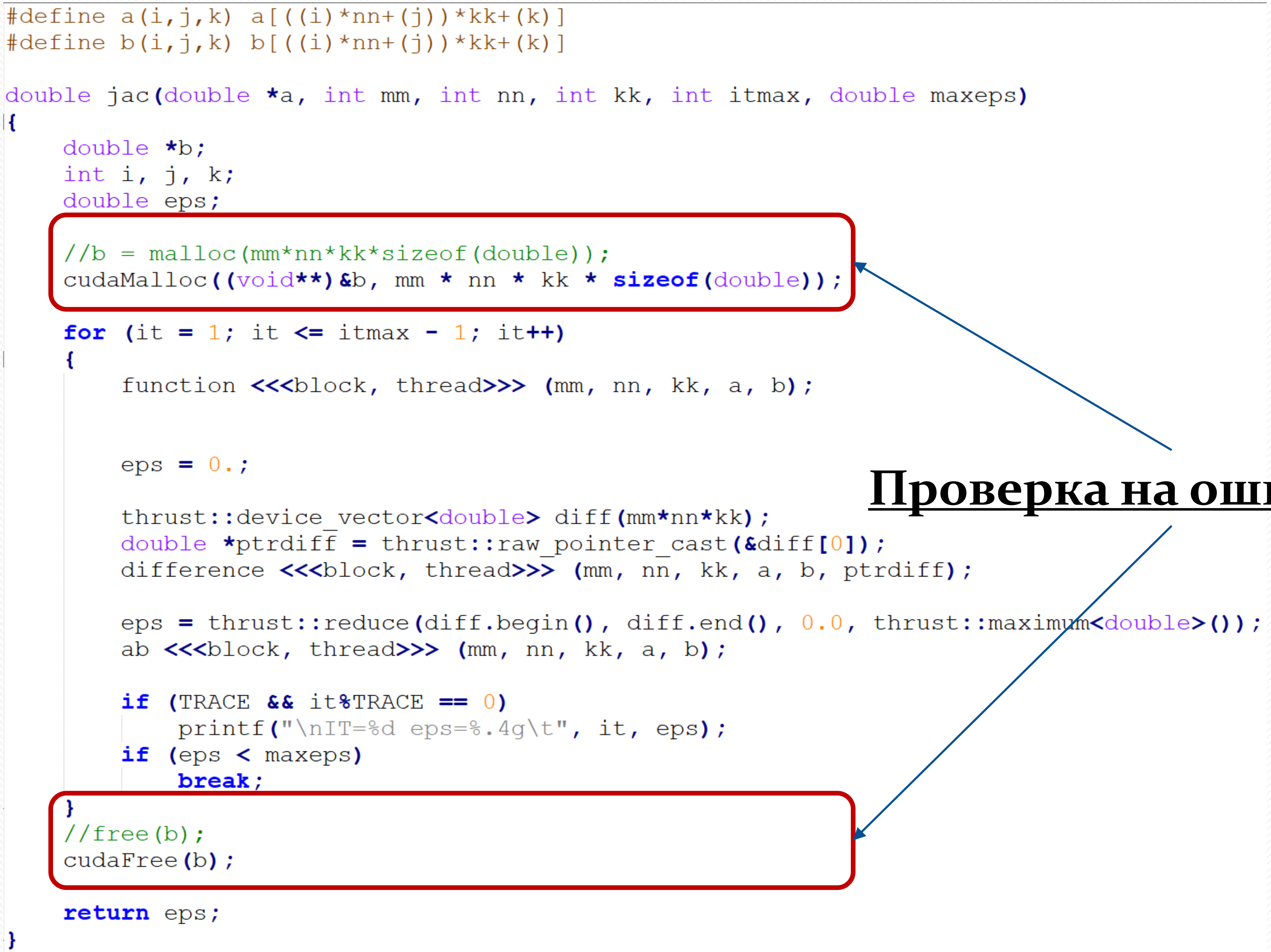
Редукция!

Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

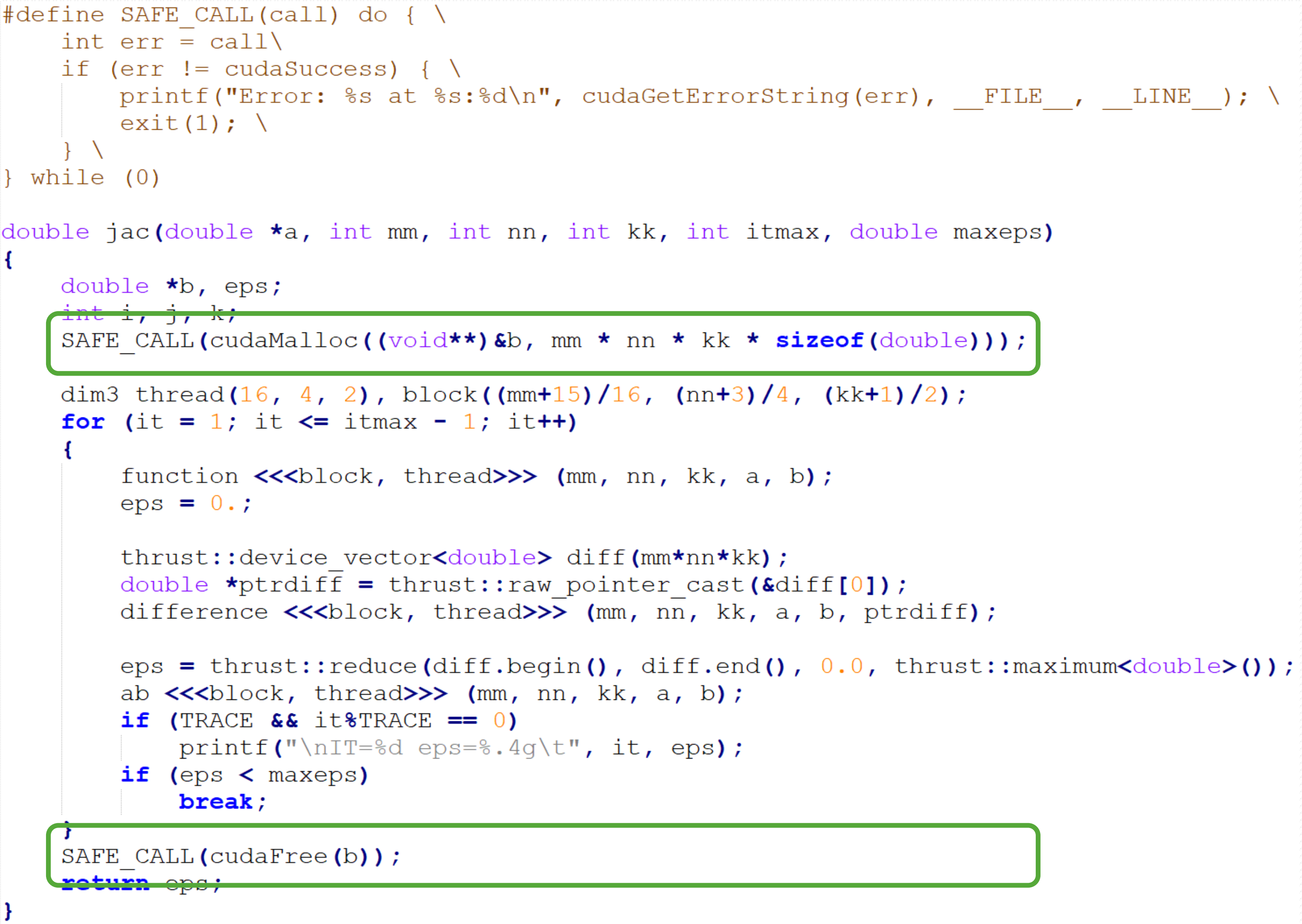
Распараллеливание Якоби



```
#define a(i,j,k) a[((i)*nn+(j))*kk+(k)]
#define b(i,j,k) b[((i)*nn+(j))*kk+(k)]

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b;
    int i, j, k;
    double eps;

    //b = malloc(mm*nn*kk*sizeof(double));
    cudaMalloc((void**)&b, mm * nn * kk * sizeof(double));

    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);


        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);

        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    //free(b);
    cudaFree(b);

    return eps;
}
```

Проверка на ошибки!

Распараллеливание Якоби



```
#define SAFE_CALL(call) do { \
    int err = call\
    if (err != cudaSuccess) { \
        printf("Error: %s at %s:%d\n", cudaGetErrorString(err), __FILE__, __LINE__); \
        exit(1); \
    } \
} while (0)

double jac(double *a, int mm, int nn, int kk, int itmax, double maxeps)
{
    double *b, eps;
    int i, j, k;
    SAFE_CALL(cudaMalloc((void**)&b, mm * nn * kk * sizeof(double)));

    dim3 thread(16, 4, 2), block((mm+15)/16, (nn+3)/4, (kk+1)/2);
    for (it = 1; it <= itmax - 1; it++)
    {
        function <<<block, thread>>> (mm, nn, kk, a, b);
        eps = 0.;

        thrust::device_vector<double> diff(mm*nn*kk);
        double *ptrdiff = thrust::raw_pointer_cast(&diff[0]);
        difference <<<block, thread>>> (mm, nn, kk, a, b, ptrdiff);

        eps = thrust::reduce(diff.begin(), diff.end(), 0.0, thrust::maximum<double>());
        ab <<<block, thread>>> (mm, nn, kk, a, b);
        if (TRACE && it%TRACE == 0)
            printf("\nIT=%d eps=%.4g\t", it, eps);
        if (eps < maxeps)
            break;
    }
    SAFE_CALL(cudaFree(b));
    return eps;
}
```