Программа трёхмерного моделирования расчёта теплового сопротивления планарных структур на основе быстрого дискретного преобразования Фурье (FourierHEAT3D).

И.М. Аболдуев, К.А. Иванов, Ал. В. Редька, А. А. Василевский, А.А. Золотарёв

АО «НПП «Пульсар», 105187, г. Москва, Окружной проезд, д. 27

В статье описана разработка алгоритма для моделирования топологии источников тепла и теплоотводов электронного оборудования. Разработанная программа, на основе данного алгоритма, позволяет проводить трёхмерное стационарное моделирование для определения распределения температуры и расчёта теплового сопротивления планарных структур. Теплоотвод состоит из нескольких слоев различных материалов. Пользователь программы может задавать от одного до девяти слоёв материалов, ортотропную теплопроводность в различных слоях по оси Oz, зависящую от температуры данного слоя. Программа позволяет варьировать размеры источников тепла и их количество, свойства материалов. В основе расчёта теплового сопротивления планарной структуры лежит метод расчета с помощью быстрого дискретного преобразования Фурье, эффективно распараллеленного на множество вычислительных ядер.

Ключевые слова: уравнение Пуассона, тепловое сопротивление, быстрое дискретное преобразование Фурье (fft), C/C++, OpenMP, cuda C, полевой транзистор с барьером Шоттки (ПТБШ).

Введение

Тепловое сопротивление полупроводникового прибора определяет предельный тепловой режим, гарантирующий его работоспособность: чем выше значение теплового сопротивления полупроводникового прибора, тем больше может быть его перегрев. Важнейшее значение приобретает контроль теплового сопротивления на стадии проектирования прибора. Для расчёта теплового сопротивления прибора на стадии проектирования в данной статье предлагается алгоритм решения уравнения теплопроводности с помощью быстрого дискретного преобразования Фурье[1]. Отличительными преимуществами предлагаемого алгоритма являются малое потребление оперативной памяти и высокая скорость расчёта. Главный недостаток алгоритма решения уравнения Пуассона на основе fft-алгоритма усматривается в том, что теплопроводности подложки в каждом её слое

могут принимать лишь постоянные значения, что может приводить к недостаточной точности расчёта.

Постановка математической задачи 2D

Прежде чем переходить к наиболее интересному для практики трёхмерному случаю, рассмотрим сначала двумерный вариант алгоритма. Для этого анализируем, не быстрое преобразование Фурье, а просто дискретное преобразование Фурье. В программе «COMGA» разрабатываемой М.К. Ермаковым [2] на протяжении ряда лет в «Институте проблем механики им. А.Ю. Ишлинского Российской академии наук» для быстрого нахождения функции тока ψ согласно уравнению Пуассона

$$\Delta\psi = \omega, \qquad (1)$$

где ω завихренность, $\Delta = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}$ оператор Лапласа, в прямоугольной на плоскости области используется дискретное преобразование Фурье [3].

Развитие вычислительной математики в последние годы привело к усовершенствованию ряда классических методов, казавшихся ранее малопригодными для численной реализации (например, метода Фурье). Рассмотрим вариант метода Фурье (метода разделения переменных), приспособленный для расчётов на ЭВМ. Использование этого метода связано с представлением искомого решения в виде конечного ряда Фурье. Запишем выражения для функции тока ψ и вихря ω в некотором узле сетки (сетка разбивает прямоугольную расчётную область на клеточки прямоугольной формы с равномерным шагом по оси Ox см. рисунок 1)

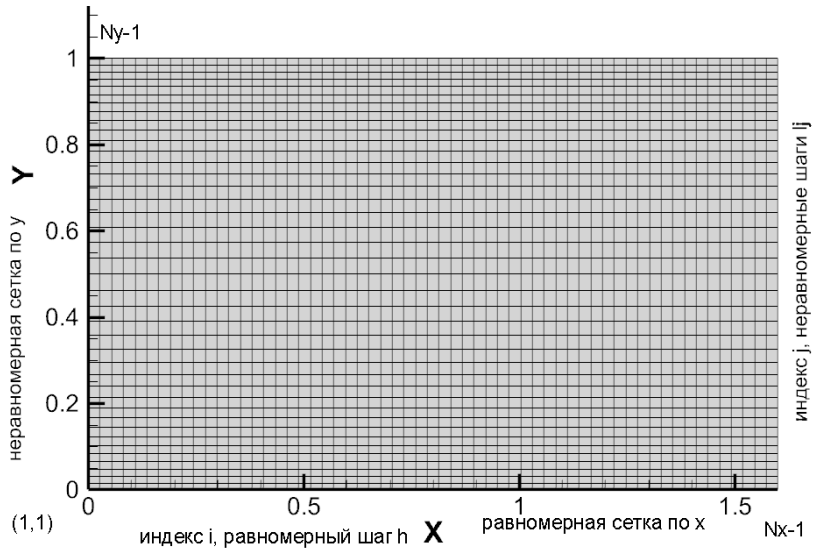

Рис. 1. Расчётная сетка.

в виде

$$\psi_{i,j} = \sum_{k=1}^{N_x-1} a_{k,j} sin\frac{\pi ki}{N_x},\ \omega_{i,j} = \sum_{k=1}^{N_x-1} b_{k,j} sin\frac{\pi ki}{N_x} \quad (2)$$

где

$$a_{k,j} = \frac{2}{N_x}\sum_{i=1}^{N_x-1} \psi_{i,j} sin\frac{\pi ki}{N_x},\ b_{k,j} = \frac{2}{N_x}\sum_{i=1}^{N_x-1} \omega_{i,j} sin\frac{\pi ki}{N_x} \quad (3)$$

для $1 \leq j \leq N_y - 1$.

На рисунке 2 представлено нахождение коэффициентов разложения в дискретный конечный ряд Фурье для правой части уравнения Лапласа.

```
#pragma omp parallel for
        for (int k = 1; k <= Nx - 1; ++k) {
                for (int j = 1; j <= Ny - 1; ++j) {
                        // j
                        b[k][j] = 0.0;
                        for (int i = 1; i <= Nx - 1; ++i) {
                                b[k][j] +=  (2.0/Nx)* rthdsd[i][j] * sin(M_PI * k * i / Nx);
                        }

                }
        }
```

Рис. 2. Нахождение коэффициентов разложения в дискретный конечный ряд Фурье за N^2 операций.

Здесь использовано разложение сеточной функции в ряд только в одном направлении. Будем предполагать разностную схему в этом направлении равномерной.

Разностная аппроксимация уравнения Пуассона будет иметь вид

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2} + \frac{2}{l_j + l_{j-1}}\left(\frac{\psi_{i,j+1} - \psi_{i,j}}{l_j} - \frac{\psi_{i,j} - \psi_{i,j-1}}{l_{j-1}}\right) = \omega_{ij}. \quad (4)$$

Для дальнейшего схему (4) удобнее переписать следующим образом:

$$\psi_{i+1,j} + \psi_{i-1,j} + \alpha_j\psi_{i,j} + \beta_j\psi_{i,j+1} + \gamma_j\psi_{i,j-1} = p\omega_{ij}, \quad (5)$$

где

$$\alpha_j = -2 - h^2\left(R_j + R_{j-1}\right), \beta_j = h^2R_j, \gamma_j = h^2R_{j-1},$$

$R_j = 2l_j^{-1}\left(l_j + l_{j-1}\right)^{-1},\ R_{j-1} = 2l_{j-1}^{-1}\left(l_j + l_{j-1}\right)^{-1}$, p=$h^2$.

Подставив (2),(3) в (5), получим

$$\sum_{k=1}^{N_x-1}\left[a_{k,j}\left(sin\frac{\pi k(i+1)}{N_x} + sin\frac{\pi k(i-1)}{N_x}\right) + \alpha_j a_{k,j} sin\frac{\pi ki}{N_x} + \right.$$
$$\left. + \beta_j a_{k,j+1} sin\frac{\pi ki}{N_x} + \gamma_j a_{k,j-1} sin\frac{\pi ki}{N_x}\right] = p\sum_{k=1}^{N_x-1} b_{k,j}\, sin\frac{\pi ki}{N_x}, \quad (6)$$

Далее, приравнивая коэффициенты при одинаковых гармониках в уравнении (6) и используя тригонометрическое соотношение

$$\sin(\alpha \pm \beta) = sin\alpha \cdot cos\beta \pm cos\alpha \cdot sin\beta, \quad (7)$$

придём к соотношению с трёхдиагональной матрицей коэффициентов

$$\beta_j a_{k,j+1} + \lambda_{k,j} a_{k,j} + \gamma_j a_{k,j-1} = p b_{k,j}. \quad (8)$$

где $\lambda_{k,j} = \alpha_j + 2cos\frac{\pi k}{N_x}$, для $1 \leq j \leq N_y - 1$. Система (8) для определения величин $a_{k,j}$ при каждом k решается методом прогонки (см. рисунок 3).

```
// постоянные шаги сетки h1 по оси x и h2 по оси y.
Real h1 = (Real)(lengthx / Nx);
Real h2 = (Real)(lengthy / Ny);

//  (0,Ny)
//  _____________________
// |                     |
// |      fluid          |
// |_____________________|
//(0,0)              (Nx,0)



#pragma omp parallel for
        for (int k = 1; k <= Nx - 1; k++) {

                double* P = new double[Ny + 1];
                double* Q = new double[Ny + 1];

                double beta = (h1 / h2), beta2 = beta * beta;

                // Для фиксированного k решение системы  с трёх диагональной матрицей:
                double b1 = beta2;
                double a1 = 2.0 + 2.0 * beta2 - 2.0 * cos(M_PI * k / Nx);
                P[1] = b1 / a1;
                double d1 = -h1 * h1 * b[k][1];
                Q[1] = d1 / a1;

                for (int i = 2; i <= Ny/*-1*/; i++) {
                        double bi = beta2;
                        double ci = beta2;
                        double ai = 2.0 + 2.0 * beta2 - 2.0 * cos(M_PI * k / Nx);
                        double di = -h1 * h1 * b[k][i];

                        P[i] = bi / (ai - ci * P[i - 1]);
                        Q[i] = (di + ci * Q[i - 1]) / (ai - ci * P[i - 1]);
                }
                a[k][Ny] = Q[Ny];

                for (int i = Ny - 1; i >= 1; i--) {
                        a[k][i] = P[i] * a[k][i + 1] + Q[i];
                }

                delete[] P;
                delete[] Q;
}
```

Рис. 3. Метод прогонки.

Затем функция тока $\psi_{i,j}$ отыскивается с помощью обратного преобразования (2) (см. рисунок 4).

```
#pragma omp parallel for
        for (int i = 1; i <= Nx - 1; ++i) {
                for (int j = 1; j <= Ny - 1; ++j) {
                        u[i][j] = 0.0;
                        // j
                        for (int k = 1; k <= Nx - 1; ++k) {
                                u[i][j] -= a[k][j] * sin(M_PI * k * i / Nx);
                        }
                }
        }
```

Рис. 4. Обратное преобразование за N^2 операций.

Использование такой методики ранее ограничивалось большим числом операций (N^2), необходимых для определения коэффициентов конечного дискретного преобразования Фурье. Развитие техники быстрого преобразования Фурье позволило сократить количество арифметических операций до величины порядка $N \cdot \log_2 N$, что делает этот метод весьма перспективным. При использовании метода дискретного преобразования Фурье невязка в уравнении Пуассона (1) опускается до машинного эпсилон.

Применение этого метода ограничивается простой формой области в форме прямоугольника на плоскости, равномерной сеткой по оси Ox в случае быстрого дискретного преобразования Фурье, характером граничных условий.

Постановка математической задачи 3D

Рассмотрим трёхмерный случай.

Для быстрого нахождения температуры T в твёрдом теле согласно уравнению Пуассона

$$\nabla(\lambda \nabla T) = P_{diss}, \qquad (9)$$

где P_{diss} тепловыделение в единице объёма, λ теплопроводность Вт/(м·К),

$\Delta = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}$ оператор Лапласа, в прямоугольном параллелепипеде в трёхмерном пространстве (см. рисунок 5) воспользуемся быстрым дискретным преобразованием Фурье. Теплопроводность λ постоянна при каждом фиксированном z в плоскости XOY, но может изменятся по оси Oz. Расчётная сетка равномерна в плоскости XOY и неравномерна по оси Oz.

Внешний вид расчётной области в трёхмерном случае представлен на рис. 5. На нижней грани расчётной области находится теплоотвод, на верхней

грани изображены источники тепла (ненулевая правая часть уравнения), на боковых гранях выставлено условие теплоизоляции.

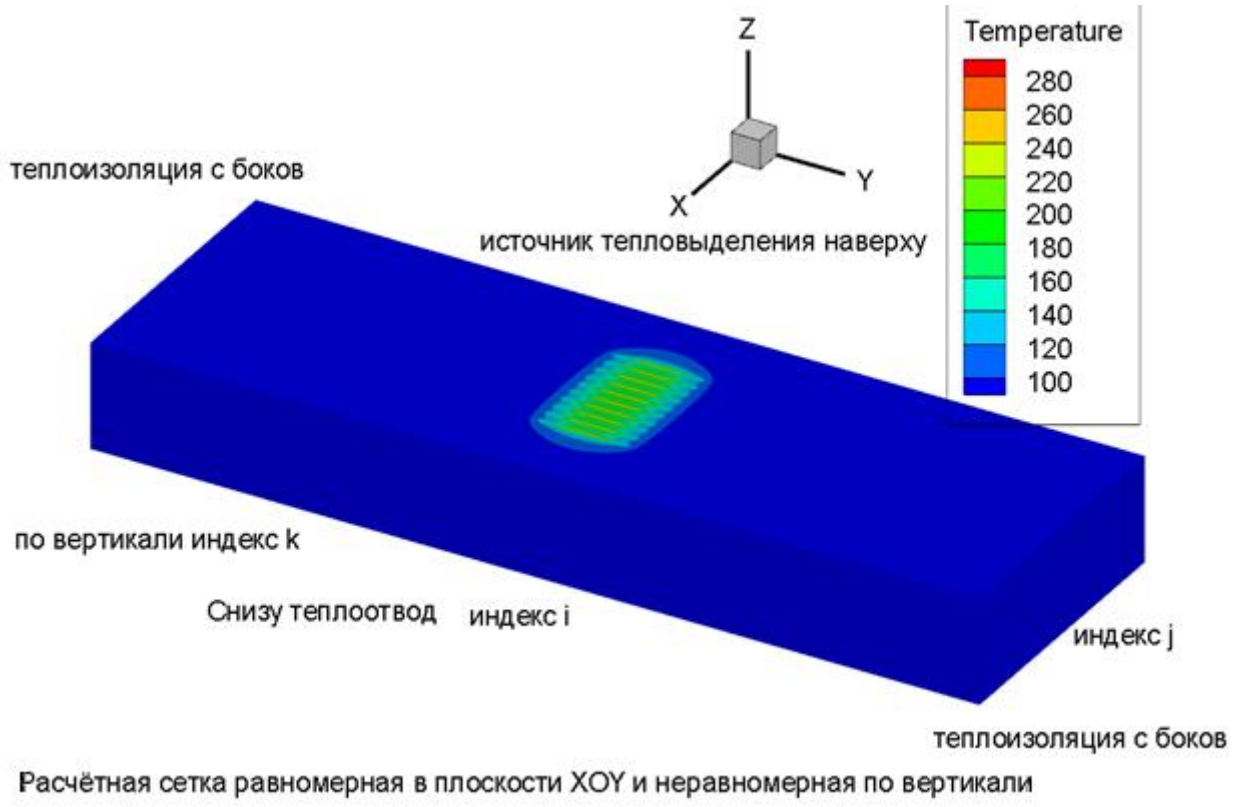

Рис. 5. Трёхмерная расчётная область.

Запишем выражения для температуры T и мощности тепловыделения в единице объёма P_{diss} в некотором узле сетки в виде:

$$T_{i,j,k} = \sum_{\hat{\imath}=1}^{N_x-1} \sum_{\hat{\jmath}=1}^{N_y-1} a_{\hat{\imath},\hat{\jmath},k} sin\frac{\pi \hat{\imath} i}{N_x} sin\frac{\pi \hat{\jmath} j}{N_y},$$
$$Pdiss_{i,j,k} = \sum_{\hat{\imath}=1}^{N_x-1} \sum_{\hat{\jmath}=1}^{N_y-1} b_{\hat{\imath},\hat{\jmath},k} sin\frac{\pi \hat{\imath} i}{N_x} sin\frac{\pi \hat{\jmath} j}{N_y}, \quad (10)$$

где следует обратить внимание на отличие роли индексов без крышечки и с крышечкой i и $\hat{\imath}$, а также j и $\hat{\jmath}$,

$$a_{\hat{\imath},\hat{\jmath},k} = \frac{2}{N_x \cdot N_y} \sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} T_{i,j,k}\, sin\frac{\pi \hat{\imath} i}{N_x} sin\frac{\pi \hat{\jmath} j}{N_y},$$
$$b_{\hat{\imath},\hat{\jmath},k} = \frac{2}{N_x \cdot N_y} \sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} Pdiss_{i,j,k}\, sin\frac{\pi \hat{\imath} i}{N_x} sin\frac{\pi \hat{\jmath} j}{N_y}, \quad (11)$$

для $1 \leq k \leq N_z - 1$.

В соотношениях (10), (11) использовано разложение сеточной функции в ряд только в двух из трёх координатных направлений. Будем предполагать

разностную схему в этих двух направлениях (в плоскости XOY) равномерной.

Разностная аппроксимация уравнения Пуассона будет иметь вид

$$\frac{\lambda_к(T_{i+1,j,к} - 2T_{i,j,к} + T_{i-1,j,к})}{h_x^2} + \frac{\lambda_к(T_{i,j+1,к} - 2T_{i,j,к} + T_{i,j-1,к})}{h_y^2} + \frac{2}{l_j + l_{j-1}}\left(\left(2\lambda_k\lambda_{k+1}/(\lambda_k + \lambda_{k+1})\right)\frac{T_{i,j,k+1} - T_{i,j,k}}{l_j} - \left(2\lambda_k\lambda_{k-1}/(\lambda_k + \lambda_{k-1})\right)\frac{T_{i,j,k} - T_{i,j,k-1}}{l_{j-1}}\right) = Pdiss_{i,j,к}. \quad (12)$$

Или после группировки слагаемых:

$$\frac{\lambda_к}{h_x^2}(T_{i-1,j,k} + T_{i+1,j,k}) + \frac{\lambda_к}{h_y^2}(T_{i,j-1,k} + T_{i,j+1,k}) + R_{k-1} \cdot T_{i,j,k-1} + R_k \cdot T_{i,j,k+1} - \left(\frac{2\lambda_к}{h_x^2} + \frac{2\lambda_к}{h_y^2} + R_{k-1} + R_k\right) \cdot T_{i,j,k} = Pdiss_{i,j,к}. \quad (13)$$

Для дальнейшего схему (13), после подстановок (10), применения тригонометрического соотношения (7) и приведения подобных слагаемых удобнее переписать в трёх-диагональном виде:

$$\alpha_{ij}a_{i,j,k} + \beta_{ij}a_{i,j,k+1} + \gamma_{ij}a_{i,j,k-1} = pb_{ijk}, \quad (12)$$

где

$$\alpha_{ij} = \left(-\frac{2}{h_x^2} - \frac{2}{h_y^2}\right) \cdot \lambda_k - (R_k + R_{k-1}) + 2 \cdot \frac{\lambda_k}{h_x^2} \cdot cos\frac{2\pi i}{N_x} + 2 \cdot \frac{\lambda_k}{h_y^2} \cdot cos\frac{2\pi j}{N_y},$$

$$\beta_{ij} = R_k, \gamma_{ij} = R_{k-1},$$

$$R_{k-1} = \left(2\lambda_k\lambda_{k-1}/(\lambda_k + \lambda_{k-1})\right)2l_{k-1}^{-1}(l_k + l_{k-1})^{-1},$$

$$R_k = \left(2\lambda_k\lambda_{k+1}/(\lambda_k + \lambda_{k+1})\right)2l_k^{-1}(l_k + l_{k-1})^{-1}, \text{ p=1.}$$

Система (12) для определения величин $a_{i,j,k}$ при каждом фиксированном наборе (i,j) на плоскости XOY решается методом прогонки [4], [5] (см. рис. 6).

```
// Ядро солвера.
// Реализует метод прогонки с трёхдиагональной матрицей для решения
// вдоль вертикального направления перпендикулярно слоям подложки.
__global__  void progonkaKernel(int Nx, int Ny, int Nz,
            const float* lambda,
            const float* lambda_multiplyer_normal, const float* lambda_multiplyer_plane,
            const float* hz, float h1, float h2,
            float* b, float* a, float* P, float* Q)
{
```

```
	int m = Nx - 1;
	int n = Ny - 1;
	int l = Nz - 1;

	const float M_PI = 3.14159265f;

	int im1 = Nz + 1;

	int i = threadIdx.x + blockIdx.x * blockDim.x;

	// Ядро солвера.
//#pragma omp parallel for
	//for (int i = 1; i < Nx - 1; i++)
	while (i < Nx - 1)
	{
		if ((i > 0) && (i < Nx - 1)) {
			for (int j = 1; j < Ny - 1; j++) {



float Rj1 = lambda_multiplyer_normal[1] *
				(2.0 * lambda[0] * lambda[1] /
				(lambda[0] + lambda[1])) *
				(2.0 / (hz[1] * (hz[1] + hz[0])));
float Rjm1 = lambda_multiplyer_normal[1] *
				(2.0 * lambda[0] * lambda[1] /
				(lambda[0] + lambda[1])) *
			(2.0 / (hz[0] * (hz[1] + hz[0])));

// Для фиксированного k решение системы
// с трёх диагональной матрицей:
float b1 = Rj1;
float a1 = (2.0 / (h1 * h1) + 2.0 / (h2 * h2)) *
			lambda[1] * lambda_multiplyer_plane[1] +
			(Rj1 + Rjm1) - 2.0 * cos(2.0 * M_PI * i / Nx) *
			lambda[1] * lambda_multiplyer_plane[1] / (h1 * h1) -
			2.0 * cos(2.0 * M_PI * j / Ny) *
			lambda[1] * lambda_multiplyer_plane[1] / (h2 * h2);
P[(i-1)*im1+1] = b1 / a1;
float d1 = -b[((i)+(j) * (m + 2) + (1) *
				(m + 2) * (n + 2))];// -b[i][j][1];
Q[(i - 1) * im1 + 1] = d1 / a1;

for (int k = 2; k <= Nz; k++) {

			float lmax = lambda[k];
			if (k != Nz) {
				lmax = lambda[k + 1];
			}
			float Rj = lambda_multiplyer_normal[k] *
					(2.0 * lambda[k] * lmax /
					(lambda[k] + lmax)) *
				(2.0 / (hz[k] * (hz[k] + hz[k - 1])));
			float Rjm1 = lambda_multiplyer_normal[k] *
				(2.0 * lambda[k - 1] * lambda[k] /
				(lambda[k - 1] + lambda[k])) *
				(2.0 / (hz[k - 1] * (hz[k] + hz[k - 1])));

			// bk -> k+1
			// ck -> k-1
			float bk = Rj;
			float ck = Rjm1;
			float ak = (2.0 / (h1 * h1) + 2.0 / (h2 * h2)) *
					lambda[k] * lambda_multiplyer_plane[k] +
					(Rj + Rjm1) -
```

```
                                        2.0 * cos(2.0 * M_PI * i / Nx) *
                                        lambda[k] * lambda_multiplyer_plane[k] /
                                        (h1 * h1) -
                                        2.0 * cos(2.0 * M_PI * j / Ny) *
                                         lambda[k] * lambda_multiplyer_plane[k] /
                                        (h2 * h2);
                                float dk = -b[((i)+(j) * (m + 2) +
                                                (k) * (m + 2) * (n + 2))];// b[i][j][k];

                                P[(i - 1) * im1 + k] = bk / (ak - ck * P[(i - 1) *
                                 im1 + k - 1]);
                                Q[(i - 1) * im1 + k] = (dk + ck * Q[(i - 1) *
                                        im1 + k - 1]) /
                                         (ak - ck * P[(i - 1) * im1 + k - 1]);
                        }
                        //a[i][j][Nz]
                        a[((i)+(j) * (m + 2) + (Nz) * (m + 2) * (n + 2))] =
                                        Q[(i - 1) * im1 + Nz];

                        for (int k = Nz - 1; k >= 1; k--) {
                                //a[i][j][k] = P[(i-1)*im1+k] *
                                //a[i][j][k + 1] + Q[(i-1)*im1+k];
                                a[((i)+(j) * (m + 2) + (k) * (m + 2) * (n + 2))] =
                                P[(i - 1) * im1 + k] *
                                a[((i)+(j) * (m + 2) + (k+1) * (m + 2) * (n + 2))] +
                                 Q[(i - 1) * im1 + k];
                        }


                }
        }

        i += blockDim.x * gridDim.x;
    }
} // progonkaKernel
```

Рис. 6. Метод прогонки на языке cuda C.

На рис. 6 переменные lambda_multiplyer_normal и lambda_multiplyer_plane множители задающие ортотропное (разное в различных координатных осях) значения теплопроводности для данного слоя по координате Oz.

Затем температура $T_{i,j,k}$ отыскивается с помощью обратного преобразования (9). При такой методике, если не применять быстрого дискретного преобразования Фурье, а ограничится просто дискретным преобразованием Фурье, наиболее медленным становится операция разложения в конечный дискретный ряд Фурье требующая в трёхмерном случае квадрат операций. Быстрое дискретное преобразование Фурье (fft), рассматриваемое далее, позволяет получить для операции нахождения коэффициентов конечного дискретного преобразования Фурье более оптимальную логарифмическую оценку числа операций типа умножение.

Алгоритм решения задачи

В трёхмерном случае ключ к успеху (быстродействию программы) обеспечивает применение быстрого дискретного преобразования Фурье.

На рисунке 7 представлен алгоритм быстрого дискретного преобразования Фурье:

```
#include <complex>
#include <cmath>
typedef std::complex<double> base;

void fft(base* &a, int n, bool invert) {

	if (n == 1) return;

	base* a0 = new base[n / 2];
	base* a1 = new base[n / 2];

	for (int i = 0, j = 0; i < n; i += 2, ++j) {
		a0[j] = a[i];
		a1[j] = a[i + 1];
	}
	fft(a0, n/2, invert);
	fft(a1, n/2, invert);

	base w  (1,0);
	double ang = 2.0*M_PI / n * (invert ? -1 : 1);
	base wn(cos(ang), sin(ang));
	for (int i = 0; i < n / 2; ++i) {
		a[i] = a0[i] + w * a1[i];
		a[i + n / 2] = a0[i] - w * a1[i];

		if (invert) {
			a[i] /= 2;
			a[i + n / 2] /= 2;
		}

		w *= wn;
	}

	delete[] a0;
	delete[] a1;
}
```

Рис. 7. Алгоритм быстрого дискретного преобразования Фурье.

Алгоритм, представленный на рисунке 7, подходит как для прямого, так и для обратного преобразования Фурье. См параметр invert. Алгоритм получает на вход вектор комплексных чисел a и на месте (в том же векторе a) находит результат разложения Фурье-коэффициенты. Используется арифметика комплексных чисел, реализованная в стандартной библиотеке.

Используя алгоритм с рисунка 7, алгоритм нахождения искомой функции u по заданным коэффициентам её разложения в конечный дискретный ряд Фурье для 3D случая запишется в виде (см. рис. 8):

```
void FFTx(Real***& u, Real***& a, int m, int n, int l) {

	int Nx = m + 1; // < M+2
```

```
	int Ny = n + 1; // < N+2
	int Nz = l + 1;

	// Находим искомую функцию.

#pragma omp parallel for
	for (int k = 1; k < Nz - 1; k++) {

		base* a1 = new base[Nx];

		for (int j = 0; j <= Ny - 1; j++) {

			a1[0]= (0.0, 0.0);
			for (int i = 1; i < Nx - 1; i++) {
				a1[i-1] = (0.0,a[i][j][k]);
			}
			fft(a1, Nx-2,false);
			for (int i = 1; i < Nx - 1; i++) {
				u[i][j][k] = -a1[i-1]._Val[0];//  Вещественная часть.
			}
		}

		delete[] a1;
	}

}

void FFTy(Real***& u, Real***& a, int m, int n, int l) {

	int Nx = m + 1; // < M+2
	int Ny = n + 1; // < N+2
	int Nz = l + 1;

	// Находим искомую функцию.
#pragma omp parallel for
	for (int k = 1; k < Nz - 1; k++) {

		base* a1 = new base[Ny];

		for (int i = 0; i <= Nx - 1; i++) {

			for (int j = 1; j < Ny - 1; j++) {
				a1[j-1] = (0.0,a[i][j][k]);
			}
			fft(a1, Ny-2, false);
			for (int j = 1; j < Ny - 1; j++) {
				u[i][j][k] = -a1[j-1]._Val[0]; //  Вещественная часть.
			}
		}

		delete[] a1;
	}
}
```

Рис. 8. Алгоритм нахождения искомой функции u по её коэффициентам разложения в конечный дискретный ряд Фурье. Формула (9).

Таким образом сначала применяется функция разложения по x, потом по y, последовательно, что устраняет один вложенный цикл. Обратное преобразование представлено на рисунке 9.

```
void IFFTx(Real***& u, Real***& f, int m, int n, int l) {

	int Nx = m + 1; // < M+2
```

```
	int Ny = n + 1; // < N+2
	int Nz = l + 1;

	// Преобразование правой части.
#pragma omp parallel for
	for (int k = 1; k < Nz - 1; k++) {


		base* a1 = new base[Nx];

		for (int j = 0; j <= Ny - 1; j++) {

			for (int i = 1; i < Nx - 1; i++) {

				a1[i-1] = (0.0,f[i][j][k]);
			}

			fft(a1, Nx-2,true);
			for (int i = 1; i < Nx - 1; i++) {

				u[i][j][k] = a1[i-1]._Val[0]; //  Вещественная часть.
			}
		}

		delete[] a1;
	}
}

void IFFTy(Real***& u, Real***& f, int m, int n, int l) {

	int Nx = m + 1; // < M+2
	int Ny = n + 1; // < N+2
	int Nz = l + 1;

	// Преобразование правой части.
#pragma omp parallel for
	for (int k = 1; k < Nz - 1; k++) {


		base* a1 = new base[Ny];

		for (int i = 0; i <= Nx - 1; i++) {

			for (int j = 1; j < Ny - 1; j++) {
				a1[j-1] = (0.0,f[i][j][k]);
			}
			fft(a1, Ny-2, true);
			for (int j = 1; j < Ny - 1; j++) {
				u[i][j][k] = a1[j-1]._Val[0]; //  Вещественная часть.
			}
		}

		delete[] a1;
	}
}
```

Рис. 9. Алгоритм нахождения коэффициентов разложения в конечный дискретный ряд Фурье по искомой функции. Формула (10).

Результаты моделирования

Возможности разработанного алгоритма демонстрируются на задаче нахождения теплового сопротивления серии полевых транзисторов с барьером Шоттки (ПТБШ) с ширинами затвора 1.25, 2.5, 5, 10 и 20 мм, а также двух (трёх) транзисторов на одном кристалле с заданным расстоянием между топологиями ПТБШ (зазор между кристаллами 300мкм). Некоторые результаты моделирования представлены на рисунках ниже.

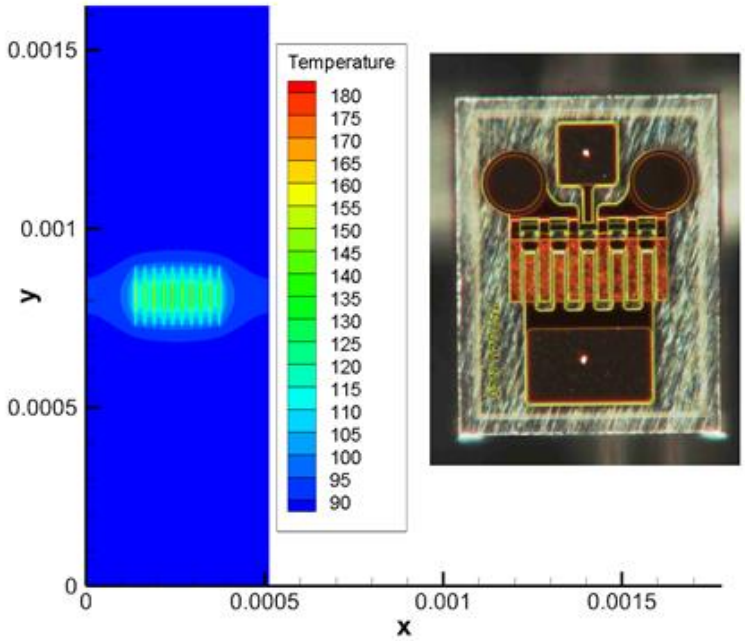

Рис. 10. Температурное поле на поверхности ПТБШ с суммарной шириной затвора Шз=1,25 мм при тепловой мощности 6.875 Вт.

На рисунке 10 изображён ПТБШ с суммарной шириной затвора Шз=1.25мм. Максимальная температура в канале по результатам fft расчёта равна 182.9°С. Тепловое сопротивление составило величину 14.7К/Вт. Алгоритм на основе fft потребляет на задаче с рисунка 10 3055Мб ОЗУ. Время расчёта теплового сопротивления, алгоритмом fft, составило 5с.

В расчётах, представленных на рисунках 10, 12-15, 17, температура основания корпуса прибора поддерживается равной T_{amb}=85°С. Размер источника тепла задаётся равным 0.5мкм·125мкм. Размер 0.5мкм следует из расчётов семейства ВАХ AlGaN/GaN ПТБШ на основе квазигидродинамической модели активной области транзистора (рисунок 11, файл rminneb80-30) [6]:

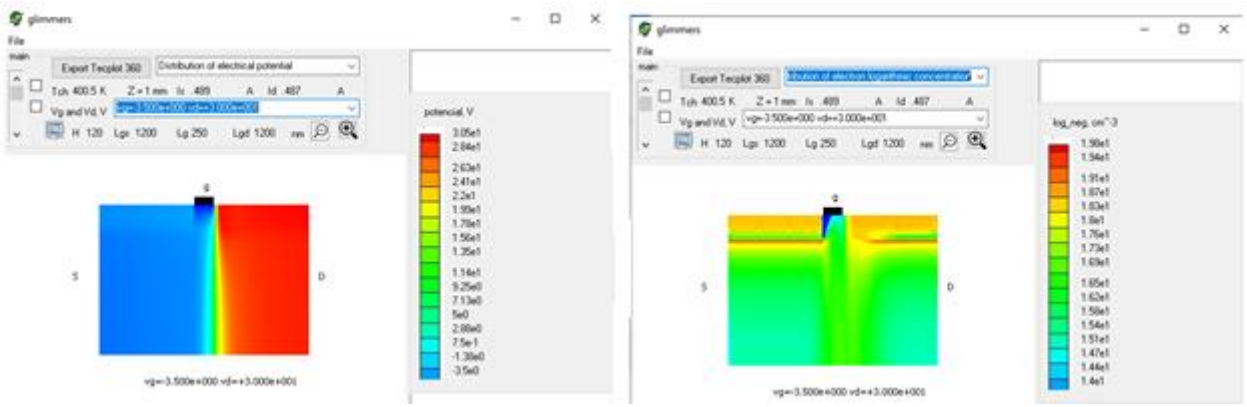

Рис.11. К оценке размеров источника тепла.

Тепло выделяется в зонах большой напряжённости электрического поля между затвором и стоком.

Во всех вариантах теплового расчёта шаг гребенчатой структуры равен 26мкм.

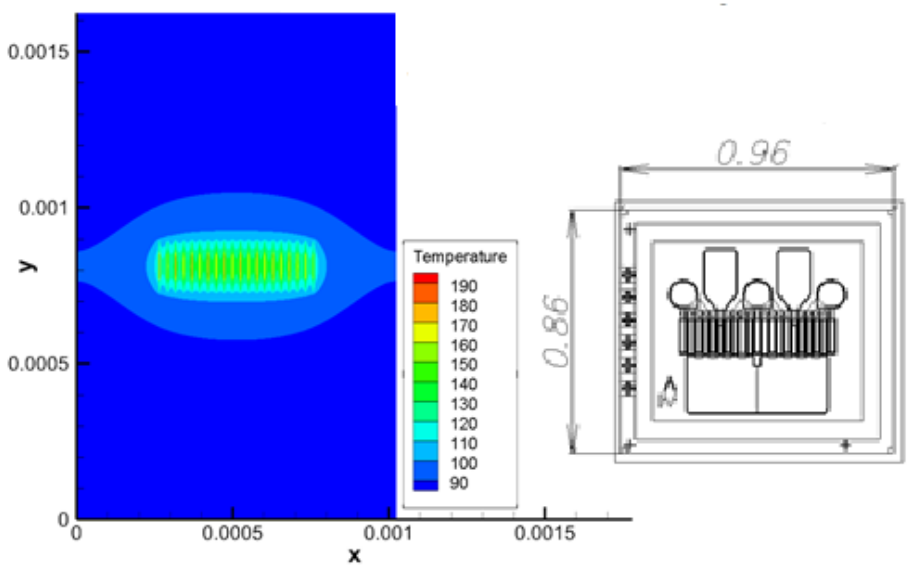

Рис. 12. Температурное поле на поверхности ПТБШ с суммарной шириной затвора Шз=2,5 мм при тепловой мощности 13,5 Вт.

На рисунке 12 изображён ПТБШ с суммарной шириной затвора Шз=2.5мм. Максимальная температура в канале по результатам fft расчёта равна 198.4°С. Тепловое сопротивление составило величину 8.62К/Вт. Алгоритм на основе fft потребляет на задаче с рисунка 12 3055Мб ОЗУ. Время расчёта теплового сопротивления алгоритмом fft составило 9с.

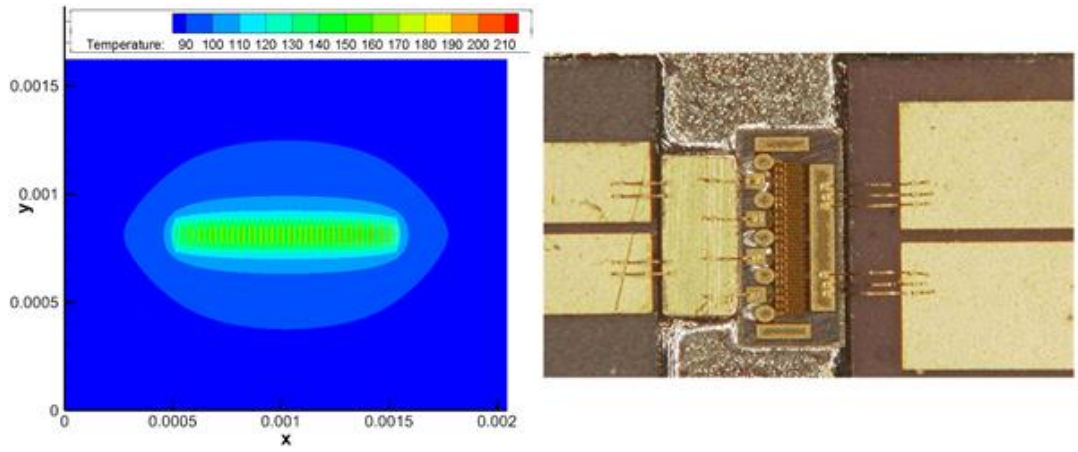

Рис. 13. Температурное поле на поверхности ПТБШ с суммарной шириной затвора Шз=5 мм при тепловой мощности 27 Вт.

На рисунке 13 изображён ПТБШ с суммарной шириной затвора Шз=5мм. Максимальная температура в канале по результатам fft расчёта равна 213°С. Тепловое сопротивление составило величину 4.86К/Вт. Алгоритм на основе fft потребляет на задаче с рисунка 13 3055Мб ОЗУ. Время расчёта теплового сопротивления алгоритмом fft составило 19с на компьютере с двумя процессорами xeon: 2*intel xeon 2630 v4. (40потоков 2.2ГГц).

Размеры расчётной области увеличиваются в реализованной программе автоматически.

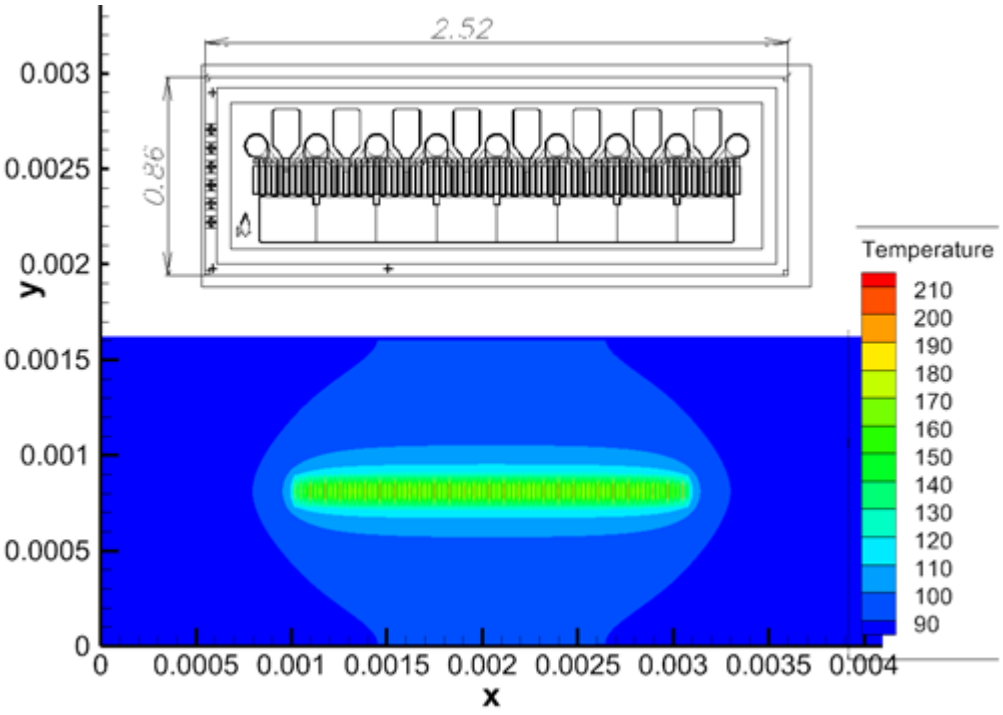

Рис. 14. Температурное поле на поверхности ПТБШ с суммарной шириной затвора Шз=10 мм при тепловой мощности 54 Вт.

На рисунке 14 изображён ПТБШ с суммарной шириной затвора Шз=10мм. Максимальная температура в канале по результатам fft расчёта равна 221°С. Тепловое сопротивление составило величину 2.58К/Вт. Алгоритм на основе fft потребляет на задаче с рисунка 14 3055Мб ОЗУ. Время расчёта теплового сопротивления алгоритмом fft составило 42с.

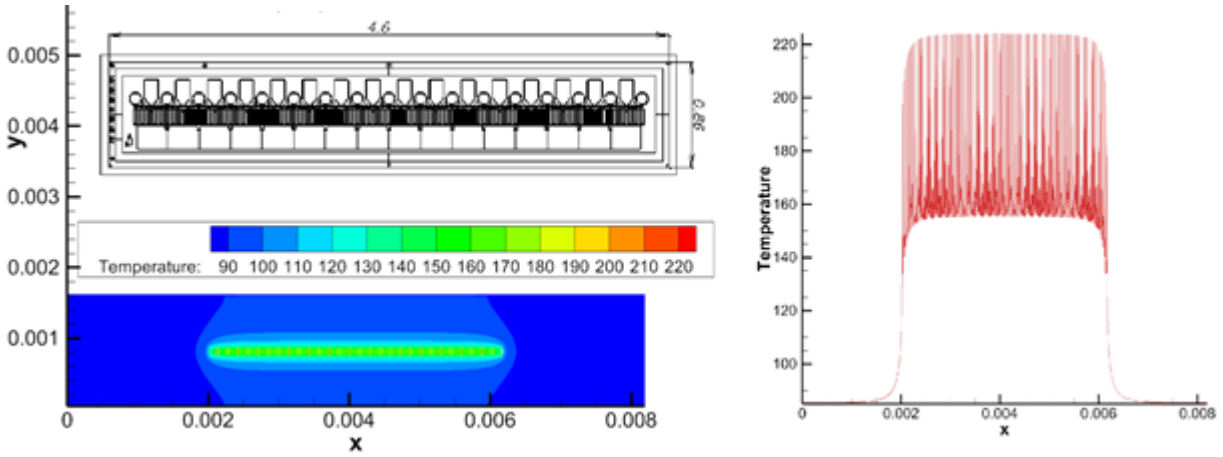

Рис. 15. Температурное поле на поверхности ПТБШ и профиль температуры вдоль кристалла с для транзистора с суммарной шириной затвора Шз=20 мм при тепловой мощности 108 Вт.

На рисунке 15 изображён ПТБШ с суммарной шириной затвора Шз=20мм. Максимальная температура в канале по результатам fft расчёта равна 224°С. Тепловое сопротивление составило величину 1,32К/Вт. Алгоритм на основе fft потребляет на задаче с рисунка 15 4514Мб ОЗУ. Время расчёта теплового сопротивления алгоритмом fft составило 1мин 25с.

Графический интерфейс разработанной программы FourierHEAT3D[1] представлен на рисунке 16.

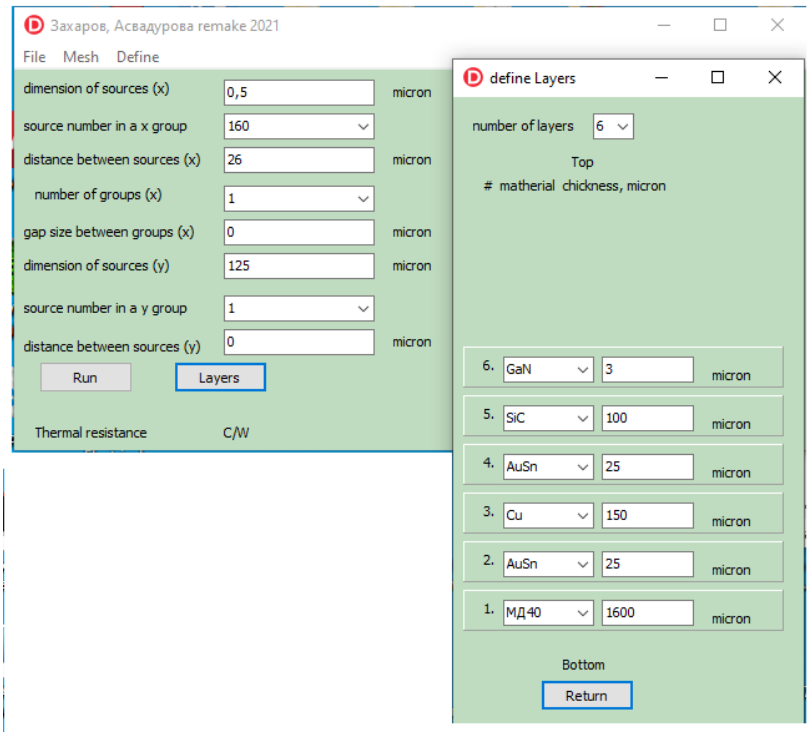

Рис. 16. Графический интерфейс программы FourierHEAT3D[1] для расчёта ПТБШ с суммарной шириной затвора Шз=20 мм.

На рис. 17 изображены рассчитанные профили температуры вдоль двух (трёх) кристаллов.

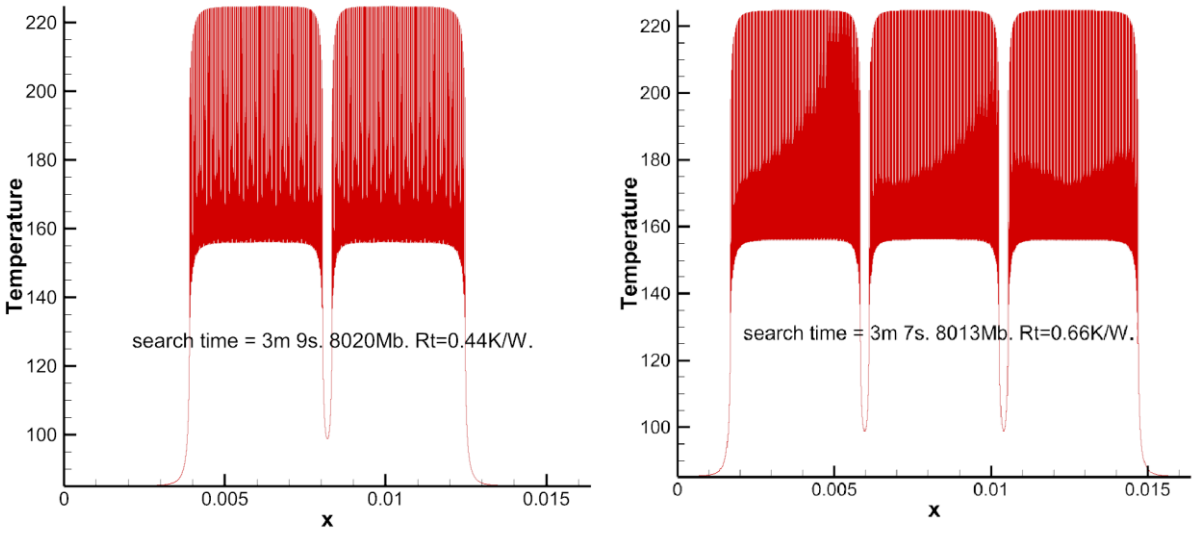

Рис. 17. Профиль температуры вдоль двух (и трёх) кристаллов на поверхности кристаллов для ПТБШ с суммарной шириной затвора 20 мм каждый (Тепловая мощность 108 Вт на каждый кристалл с Шз=20 мм).

Результаты моделирования с помощью fft алгоритма в арифметике одинарной точности (float) представлены в таблице № 1:

Таблица №1 – результаты моделирования с помощью fft алгоритма на 2*intel xeon 2630 v4 в многопоточном режиме на OpenMP[7].

Данные для расчёта теплового сопротивления	Время расчёта (2*intel xeon 2630 v4)	Оперативная память	Тепловое сопротивление, °С/Вт
3*ПТБШ Шз=20мм	3м 9с	8020Мб	0,44
2*ПТБШ Шз=20мм	3м 7с	8013,0 Мб	0,66
ПТБШ Шз=20мм	1м 25с	4514,5 Мб	1,32
ПТБШ Шз=10мм	42с	3054,9Мб	2,58
ПТБШ Шз=5мм	19с	3055Мб	4.86
ПТБШ Шз=2,5мм	9с	3055Мб	8,62
ПТБШ Шз=1,25мм	5с	3055Мб	14,67

Результаты моделирования с помощью amg алгоритма [8] представлены для сравнения с fft-алгоритмом в таблице № 2. Решающая связка алгоритмов в библиотеке AMGCL выбрана следующей: внешняя итерационная процедура BiCGStab; алгоритм построения вложенных сеток для amg предобуславливателя Д. Руге, К. Штубен; сглаживающая ошибку итерационная процедура spai0. [8].

Таблица №2 – результаты моделирования с помощью amg алгоритма[8] на 2*intel xeon 2630 v4 в многопоточном режиме на OpenMP[7].

Данные для расчёта теплового сопротивления	Время расчёта (2*intel xeon 2630 v4)	Оперативная память	Тепловое сопротивление, °С/Вт
2*ПТБШ Шз=20мм	1м 20с	4431Мб	1,19
ПТБШ Шз=20мм	17с	1088Мб	2.37
ПТБШ Шз=10мм	21с	1176Мб	4.08
ПТБШ Шз=5мм	10с	686Мб	7,39
ПТБШ Шз=1,25мм	12с	746Мб	14,0

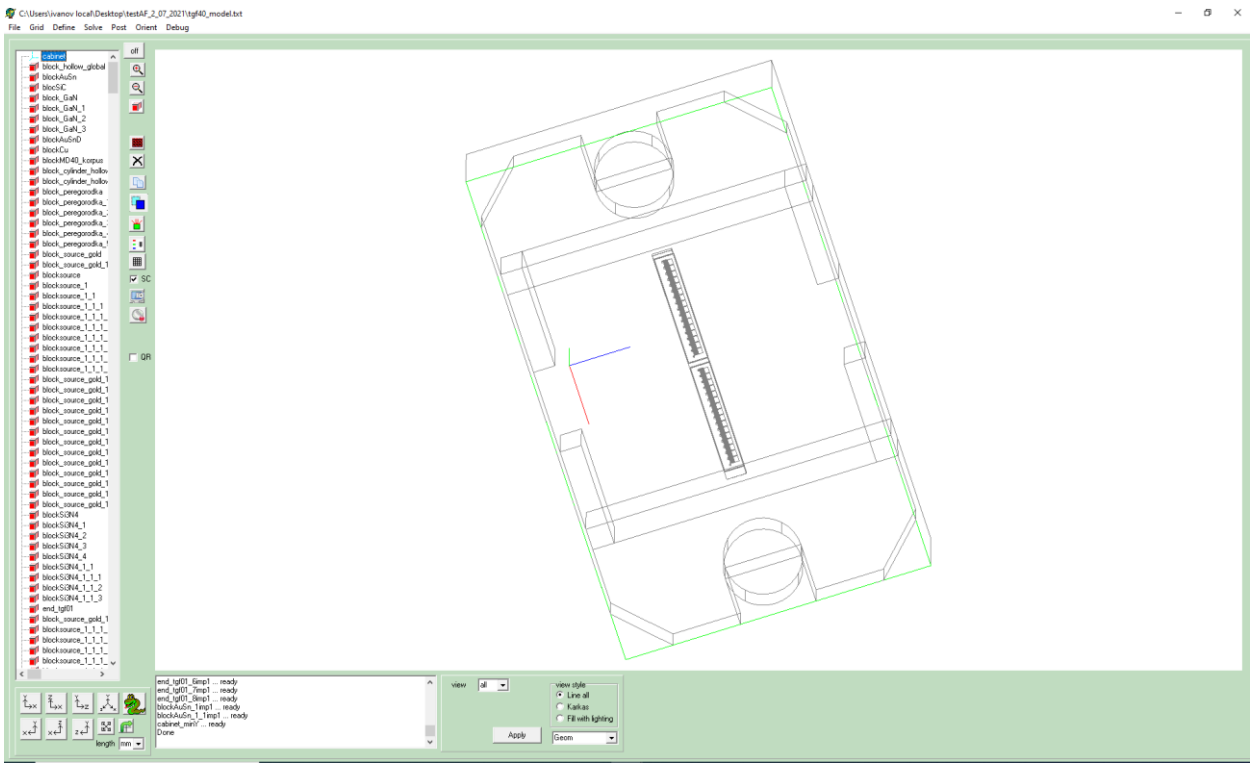

Рис. 18. Внешний вид тепловой модели двух-кристального прибора с суммарной шириной затвора 40 мм.

На задаче с рисунка 18 сравнивается быстродействие различных алгоритмов для расчёта теплового сопротивления ПТБШ [1],[8],[9], представленное на рисунке 19.

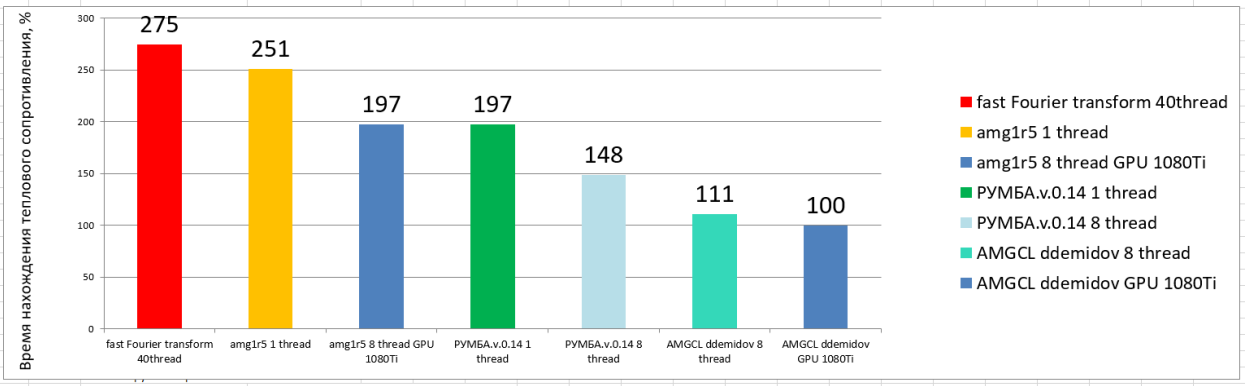

Рис. 19. Сравнение быстродействия различных алгоритмов, на примере решения задачи по расчёту теплового сопротивления двух-кристального ПТБШ с суммарной шириной затвора 40 мм, изображенного на рисунке 18.

Условия при которых были получены результаты моделирования

Таблицы 1 и 2 рассчитывались на разных расчётных сетках. Сетка для метода fft содержала существенно большее число ячеек, чем для метода amg, т.к. метод fft требует равномерной сетки в плоскости XOY, а в методе amg сетка подстраивается под геометрические особенности тепловой модели.

Время метода amg, представленное в таблице №2, содержало несколько решений СЛАУ из-за теплопроводностей материалов, зависящих от температуры, а также содержало время записи результата в файл. При расчёте с использованием amg метода участвовала реальная геометрия 13мм корпуса ПТБШ.

Время метода fft, представленное в таблице №1, также содержало несколько решений уравнения Пуассона из-за теплопроводностей материалов, зависящих от температуры.

Результаты, представленные в статье, были получены в одинаковых условиях на оборудовании: материнская плата supermicro серии X10 имеющая 2 сокета 2011-v3, два процессора intel xeon 2630 v4, 448ГГб оперативной памяти, графический ускоритель GeForce GTX 1080Ti, ssd диск Samsung, размером 1Тб подключённый по sata 3 интерфейсу. Для компиляции программ в обоих случаях использовался компилятор MS Visual Studio 2019 Community с одинаковыми настройками. Расчёт производился под управлением ОС Windows 10.

Выводы

Возможности разработанной программы FourierHEAT3D[1] демонстрируются на примере расчёта теплового сопротивления различных ПТБШ с суммарными ширинами затвора от 1.25 мм до 20 мм и двух (трех) ШТБШ с суммарной шириной затвора 40мм (60мм). По результатам исследования можно сделать следующие выводы. Для достижения точности fft моделирования теплового сопротивления необходимо для рассматриваемой серии ПТБШ выбрать размер источника тепла равный 0.5мкм. Метод расчёта теплового сопротивления с помощью fft алгоритма характеризуется равномерной расчётной сеткой в плоскости залегания рассчитываемой планарной структуры и при размере источника тепла 0.5мкм число ячеек расчётной сетки метода fft значительно превосходит число ячеек расчётной сетки используемое в расчёте теплового сопротивления с помощью алгоритма amg. Из-за этого метод fft уступает в скорости расчёта теплового сопротивления ПТБШ методу на основе алгоритма amg (рисунок 19) в два с лишним раза.

Литература

1. К.А. Иванов, А.А. Золотарев, А.А. Василевский. Программа трёхмерного моделирования расчёта теплового сопротивления планарных структур на основе быстрого дискретного преобразования

Фурье (FourierHEAT3D). Свидетельство о государственной регистрации программы для ЭВМ № 2021662363 от 27 июля 2021г.
2. М.К. Ермаков. Использование многосеточного метода в задачах физической механики.// Физико-химическая кинетика в газовой динамике. 2014. Т.15, вып. 2.
3. В.М. Пасконов, В.И. Полежаев, Л.А. Чудов. Численное моделирование процессов тепло- и массообмена. М. Наука, 1984.
4. С. Патанкар. Численные методы решения задач теплообмена и динамики жидкости. М. Энергоатомиздат, 1984.
5. Д. Сандерс, Э. Кэндрот. Технология CUDA в примерах. Введение в программирование графических процессоров. ДМК-Пресс, 2018г.
6. Г.З. Гарбер. Численное моделирование электронно-дырочной плазмы в гетероструктурных полевых транзисторах. Радиотехника и электроника, 2005, том 50, №10. с. 1308-1312.
7. Ш. Эхтер, Д. Робертс. Многоядерное программирование. М. Питер, 2010.
8. Demidov, Denis. AMGCL: An efficient, flexible, and extensible algebraic multigrid implementation. Lobachevskii Journal of Mathematics, 40(5):535–546, May 2019.
9. К.А. Иванов, А.А. Золотарев, А.А. Василевский. Параллельный алгебраический многосеточный метод (РУМБА_v.0.14). Свидетельство о государственной регистрации программы для ЭВМ № 2021662270 от 26 июля 2021г.