Сравнение процессоров Intel Core-i7, Intel Xeon, Intel Xeon Phi и IBM Power 8 на примере задачи восстановления начальных данных

А.Ю. Горчаков, В.У. Малкова

***Аннотация* — В данной работе проводится сравнительный анализ четырех типов процессоров на примере задачи восстановления начальных данных для уравнения переноса. Задача решается методом Левенберга-Марквардта, который декомпозирован на четыре подзадачи – вычисление вектор-функции, вычисление матрицы первых производных вектор-функции (матрицы Якоби), матричное умножение и решение системы линейных уравнений. Вычисление матрицы Якоби производится с помощью методологии быстрого автоматического дифференцирования, при этом применяется пакет прикладного программного обеспечения Adept версии 1.1. Для ускорения расчетов использовано директивное многопоточное программирование с динамическим распределением вычислений между потоками и технология SIMD (Single Instruction Multiple Data) – принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. Вышеупомянутые технологии позволили в полной мере задействовать особенности архитектуры современных процессоров Intel, такие как многоядерность/многопоточность, расширение системы команд микропроцессоров Intel/AMD – Advanced Vector Extensions (AVX/AVX2), обрабатывающее данные в формате с плавающей запятой в группах длиной 256 бит, и FMA (Fused Multiply-Add) – технологию, предназначенную для выполнения совмещенной операции умножения-сложения. В сравнении участвуют процессоры Intel Core i7 4770 (Haswell), Intel Xeon E5-2683V4 (Broadwell), Intel Xeon Phi coprocessor SE10/7120 и IBM Power 8. Даны рекомендации по выбору типа процессора в зависимости от решаемой задачи.**

***Ключевые слова* — OpenMP, SIMD, матрица Якоби, задача о наименьших квадратах, метод Левенберга-Марквардта.**

*Работа выполнена при поддержке РФФИ, проект 16-07-00458

А.Ю. Горчаков – ведущий математик Вычислительного центра им. А.А. Дородницина Федерального исследовательского центра «Информатика и управление» Российской академии наук. andrgor12@gmail.com

В.У. Малкова – инженер-исследователь Вычислительного центра им. А.А. Дородницина Федерального исследовательского центра «Информатика и управление» Российской академии наук. vmalkova@yandex.ru

I. Введение

Целью данной работы является сравнение производительности архитектуры IBM Power8 Minsky и традиционных архитектур x86 при решении одной из научных задач, стоящих перед авторами.

Сравнение проводится на примере модельной задачи восстановления начальных данных для уравнения переноса. Задача решается методом Левенберга-Марквардта. Вычисление матрицы первых производных вектор-функции (матрицы Якоби) производится с помощью методологии быстрого автоматического дифференцирования, при этом применяется пакет прикладного программного обеспечения Adept версии 1.1 [1]. Для ускорения расчетов использовано директивное многопоточное программирование с динамическим распределением вычислений между потоками и технология SIMD (Single Instruction Multiple Data) – принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. Вышеупомянутые технологии позволили в полной мере задействовать особенности архитектуры современных процессоров Intel, такие как многоядерность/многопоточность, расширение системы команд микропроцессоров Intel/AMD – Advanced Vector Extensions (AVX/AVX2), обрабатывающее данные в формате с плавающей запятой в группах длиной 256 бит, и FMA (Fused Multiply-Add) – технологию, предназначенную для выполнения совмещенной операции умножения-сложения. Для решения систем линейных уравнений использовался cLapack [2]. Для выполнения расчетов были использованы вычислительные ресурсы ФИЦ ИУ РАН [3].

II. Состав испытательного стенда

Испытательный стенд состоял из следующих средств вычислительной техники:

- Intel Core i7 4770 3.4 Гц, 8 Гб RAM – 4-х ядерный процессор архитектуры Haswell, с поддержкой технологий Hyper-Threading (два вычислительных

протока на ядро, всего 8 логических ядер), AVX2 – Advanced Vector Extensions (в контексте решаемой задачи – обработка данных в формате с плавающей запятой в группах длиной 256 бит – т.е. для чисел с двойной точностью одновременно обрабатывается вектор из 4-х чисел), и FMA – Fused Multiply-Add – выполнение совмещенной операции умножения-сложения вида $a = a+b*c$. Использовался компилятор gcc 6.4.0.

- 2xIntel Xeon E5-2683V4 2.1 Гц, 512 Гб RAM – двух процессорный сервер, каждый процессор содержит 16 ядер с поддержкой Hyper-Threading, AVX2 и FMA (всего 64 логических ядра). Использовался компилятор Intel Parallel Studio XE 17.0.5 – штатный для данного сервера.
- Intel Xeon Phi coprocessor SE10/7120 1.238 Гц, 16 Гб RAM – 61 ядро x86. Поддерживается 4 вычислительных протока на ядро (всего 244 логических ядер). Имеется возможность использовать технологию AVX512MIC (обработка вектора из 8-и чисел с двойной точностью), но из-за отличий ее поддержки компиляторами от технологий AVX2 и AVX512 она не использовалась. Использовался компилятор Intel Parallel Studio XE 17.0.5.
- 2xIBM Power8 3.5 Гц, 512 Гб RAM – 2-х процессорный сервер, каждый процессор содержит 8 ядер с поддержкой технологии SMT8 (8 вычислительных потоков на ядро) – всего 128 логических ядер. Использовался компилятор gcc 7.2.0.

III. Постановка модельной задачи

Рассмотрим задачу восстановления начальных данных для одномерного уравнения переноса пассивной примеси в жидкости движущейся с постоянным значением скорости:

$$\frac{\partial \varphi}{\partial t} + c\frac{\partial \varphi}{\partial x} = 0, c = 1, \quad (x,t) \in Q, \tag{1}$$

$$\varphi(x,0) = w_1(x), \qquad x \in [0,1], \tag{2}$$

$$\varphi(0,t) = w_2(t), \qquad t \in (0,1]. \tag{3}$$

Здесь x – пространственная координата; t – время; c – скорость движения жидкости; $\varphi(x,t)$ – концентрация примеси в точке x в момент времени t; прямоугольник $Q = (0,1)\times(0,1)$; $w_2(x)$ – заданная функция, определяющая концентрацию примеси в точке $x=0$; $w_1(x)$ – искомая функция, определяющая концентрацию примеси в начальный момент времени. Пусть на прямой $x+t=1$ задана функция $\bar{\varphi}(x,t)$, где $(x,t \in Q)$. Ее можно рассматривать как концентрацию примеси, полученную в результате некоторых экспериментальных исследований. Поставим следующую задачу: требуется определить оптимальное управление $u_*(x) = w_{1_*}(x)$ и соответствующее решение $\varphi_*(x,t)$ системы (1)-(3), при котором интегральный функционал

$$W(u) = \frac{1}{2}\int_0^1 [\varphi(x,1-x) - \bar{\varphi}(x,1-x)]^2\,dx \tag{4}$$

достигал бы минимально возможного значения.

Задачи, подобные этой, обычно решаются численно с помощью некоторого метода спуска, который требует знания градиента функционала (4).

Перейдем сразу же к дискретному варианту системы (1)-(4). Для этого покроем прямоугольник Q регулярной расчетной сеткой с шагами $h=1/J$ и $\tau=1/N$, где N – количество интервалов по времени, а J – количество интервалов по оси x (для удобства расчетов возьмем N кратным J, т.е. $N = k * J$). Используем хорошо известную схему «уголок», аппроксимирующую уравнение (1) на трехточечном шаблоне с первым порядком точности:

$$\frac{\varphi_j^{n+1} - \varphi_j^n}{\tau} + c\frac{\varphi_j^n - \varphi_{j-1}^n}{h} = 0\,, \tag{5}$$

$$\varphi_j^0 = w_{1_j} = u_j, \tag{6}$$

$$\varphi_0^n = {w_2}^n, \tag{7}$$

$$W(u) = \frac{1}{2}\sum_{j=1}^{J}(\varphi_j^{N-k*j} - \bar{\varphi}_j^{N-k*j})^2, \tag{8}$$

где $\varphi_j^n = \varphi(x_j,t_n), \bar{\varphi}_j^n = \bar{\varphi}(x_j,t_n), x_j = jh, t_n = n\tau,$
$\varphi_j^0 = w_{1_j} = u_j = u(x_j), \varphi_0^n = {w_2}^n = w_2(t_n),$
$0 \le j \le J, 0 \le n \le N.$

Требуется найти вектор u, при котором сложная дискретизированная функция $W(u)$ принимает минимальное значение.

IV. Метод Левенберга-Марквардта

Среди задач безусловной оптимизации часто выделяют задачи минимизации функций $F(x)$ вида:

$$F(x) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x) = \frac{1}{2}\|f(x)\|_2^{\,2} \to \min, \tag{9}$$

Где $f(x)$ – нелинейная вектор функция с m компонентами $f_i(x)$, а $x \in R^n$, $\|f(x)\|_2$- евклидова норма.

Задачи подобного типа возникают при математическом моделировании физических процессов, например, задача об оптимальном нагреве стержня [4], задача параметрической идентификации потенциалов межатомного взаимодействия [5], многие задачи машинного обучения.

Хотя для решения задач (1) можно использовать универсальные методы, более эффективным будет применение специальных алгоритмов, разработанных именно для задач о наименьших квадратах. Примерами таких алгоритмов являются метод Гаусса-Ньютона и Левенберга-Марквардта [6]. Оба метода используют особую структуру градиента функции и её матрицы Гессе.

Пусть $J(x)$ матрица Якоби для $f(x)$ и $H_i(x)$ матрица Гессе для $f_i(x)$. Тогда градиент g(x) и матрицу Гессе H(x) можно записать как:

$$g(x) = J(x)^T f(x) \tag{10}$$

$$H(x) = J(x)^T J(x) + Q(x) \tag{11}$$

где $Q(x) = \sum_{i=1}^{m} f_i(x)H_i(x)$. Сделав предположения о том, что слагаемое $J(x)^T J(x)$ доминирует над $Q(x)$, получаем следующие итерационные схемы методов - метод Гаусса-Ньютона:

$$x_{k+1} = x_k + \alpha_k p_k, \tag{12}$$

где $0 < \alpha_k \leq 1$ шаг метода, p_k- направление поиска определяемое из решения системы уравнений

$$J_k^T J_k p_k = -J_k^T f_k \quad (13)$$

метод Левенберга-Марквардта:

$$x_{k+1} = x_k + p_k, \quad (14)$$

где I - единичная матрица, λ_k – некоторая неотрицательная константа, своя для каждого шага, p_k- направление поиска определяемое из решения системы уравнений

$$(J_k^T J_k + \lambda_k I) p_k = -J_k^T f_k . \quad (15)$$

Оба метода, при определенных условиях, могут достигать квадратичной скорости сходимости, хотя при расчете используются только первые производные (матрица Якоби).

V. Методология быстрого автоматического дифференцирования

Пусть $z \in R^n$ и $u \in R^r$ – векторы. Дифференцируемые функции *W(z, u)* и *Φ(z, u)* определяют отображения $W: R^n \times R^r \to R^1$, $\Phi: R^n \times R^r \to R^r$. Вектора z и u удовлетворяют следующей системе из *n* нелинейных скалярных уравнений *Φ(z, u)* = 0. Если матрица $\Phi_z^T(z,u)$ невырождена, то сложная функция $\Omega(u) = W(z(u),u)$ дифференцируема и ее градиент относительно переменных u вычисляется по формуле:

$$\frac{d\Omega}{\mathrm{du}} = W_u(\mathrm{z(u)},\mathrm{u}) + \Phi_u^T(z(u),u) \cdot p, \quad (16)$$

где вектор $p \in R^n$ находится из решения линейной алгебраической системы:

$$W_z(\mathrm{z},\mathrm{u}) + \Phi_z^T(z,u) \cdot p = 0. \quad (17)$$

Здесь и далее индекс T обозначает транспонирование, нижние индексы *z*, *u* обозначают частные производные функций по векторам *z* и *u*: $W_u = \frac{\partial W}{\partial u}, W_z = \frac{\partial W}{\partial z}, \Phi_u^T = \frac{\partial \Phi^T}{\partial u}, \Phi_z^T = \frac{\partial \Phi^T}{\partial z}$, так же будем обозначать *i*-е, *j*-е компоненты векторов z и u как z_i, z_j, u_i, u_j.

Предположим, что каждый компонент вектора z_i последовательно выражается только через компоненты вектора u и предыдущие z_j, т.е. *1≤j<i.*, тогда формулы (16)-(17) можно записать в следующем виде:

$$p_i = \sum_{j=i+1}^{n} \Phi_{z_j}^T(z,u) p_j + W_{z_i}(z,u) \quad (18)$$

$$\frac{d\Omega}{\mathrm{du}_i} = W_{u_i}(z,u) + \sum_{j=1}^{n} \Phi_{u_j}^T(z,u) \cdot p_j \quad (19)$$

В работе [7] дана оценка временной сложности вычисления градиента $T_g/T_0 \leq 3$, где T_0 полное время, требуемое для вычисления значения функции, T_g дополнительное время, требуемое для вычисления всех частных производных функции; оценка дана без учета времени, необходимого для работы с памятью компьютера. Напомним, что в случае численного дифференцирования для расчета градиента (и матрицы Якоби) потребуется времени не менее $(1+n)T_0$, где n число необходимых нам частных производных. Теоретическая оценка временной сложности вычисления $T_J/T_0 \leq 3\mathrm{m}$, где T_J время вычисления матрицы Якоби. При использовании пакета Adept, матрица Якоби рассчитывается с помощью функции:

Stack::jacobian_reverse_openmp(Real* jacobian_out)

функция устроена следующим образом - матрица Якоби разбивается по столбцам на блоки размера ADEPT_MULTIPASS_SIZE, при этом размер последнего блока может быть меньше этого параметра. Директивой #pragma omp parallel объявляется параллельный регион программы. Цикл по блокам разбивается на потоки с помощью директивы OpenMP - #pragma omp for. Для данной задачи статическое разбиение цикла не является эффективным, поэтому произведена замена директивы на #pragma omp for schedule(dynamic):

```
#pragma omp for schedule(dynamic)
    for (iblock = 0; iblock < n_block; iblock++) {
……..
```

. Циклы внутри блока распараллеливаются с помощью директивы #pragma omp simd:

```
#pragma omp simd
  for (int i = 0; i < ADEPT_MULTIPASS_SIZE; i++)
…….
```

VI. Численный эксперимент

Приведем описание использованного в численных экспериментах метода:

1. Положить k=0, $\lambda_0 = 0.01, \varepsilon = 10^{-20}$
2. Вычислить $F_k; f_k$
3. Вычислить J_k
4. Вычислить $J_k^T J_k$ и $-J_k^T f_k$
5. Сложить $\lambda_k I$ с $J_k^T J_k$
6. Найти p_k, решив систему уравнений (15)
7. Положить $u_{k+1} = u_k + p_k$
8. Вычислить $F_{k+1}; f_{k+1}$
9. Если $F_{k+1} < \epsilon$, то завершить работу.
10. Если $F_{k+1} < F_k$, то положить $\lambda_{k+1} = 0.1\lambda_k, k = k+1$, и перейти к шагу 3.
11. Если $\lambda_k > 10^{20}$, то завершить работу.
12. Если $F_{k+1} \geq F_k$, то положить $\lambda_{k+1} = 10\lambda_k$, и перейти к шагу 5.

Проведем расчеты в однопоточном режиме на Intel Core i7 для расчетной сетки с 12000 интервалов по времени и 4000 интервалов по оси *x*.

Таблица 1 – Сходимость метода Левенберга-Марквардта

Шаг	Значение найденного минимума
Начальная точка	1.430922e-01
1	5.963379e-05
2	2.654437e-10
3	1.241797e-17
4	3.839771e-25

Из таблицы 1 видно, что скорость сходимости метода Левенберга-Марквардта достаточна для того, чтобы решить задачу за 4 итерации. При этом функция вычисляется 5 раз, матрица Якоби 4 раза и 4 раза решается система уравнений (15).

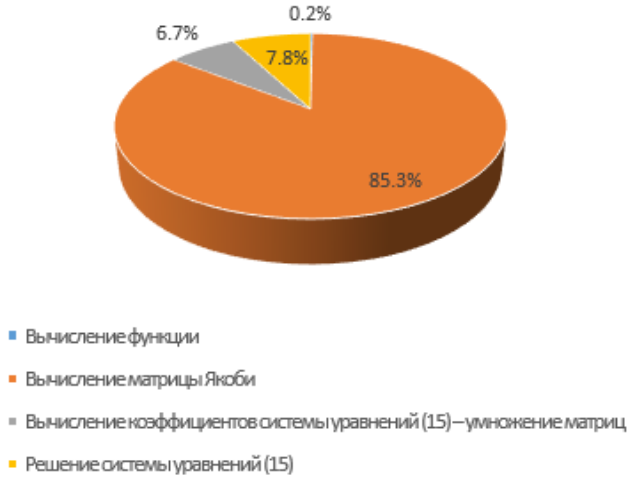
Таблица 2 – Время вычислений в однопоточном режиме для Intel core i7

Шаг	Время, сек
Вычисление функции	5.0

Вычисление матрицы Якоби	2271.0
Вычисление коэффициентов системы уравнений (15) – умножение матриц	179.3
Решение системы уравнений (15)	206.4
Всего	2661.7

Как видно из таблицы 2 более 85% времени уходит на вычисление матрицы первых производных – матрицы Якоби. Для наглядности представим данные в виде диаграммы.

Таблица 3 – Относительная доля времени вычислений в однопоточном режиме для Intel core i7



В процессе работы над методом решения задачи основные усилия были направлены на ускорение расчета матрицы Якоби. Применение директивного многопоточного программирования с динамическим распределением вычислений между потоками и использования расширений команд микропроцессоров Intel AVX2 позволило ускорить вычисление матрицы Якоби более чем в 12 раз [8] (при расчетах на Intel core i7).

Теперь запустим расчет данной задачи в многопоточном режиме на имеющихся вычислительных устройствах.

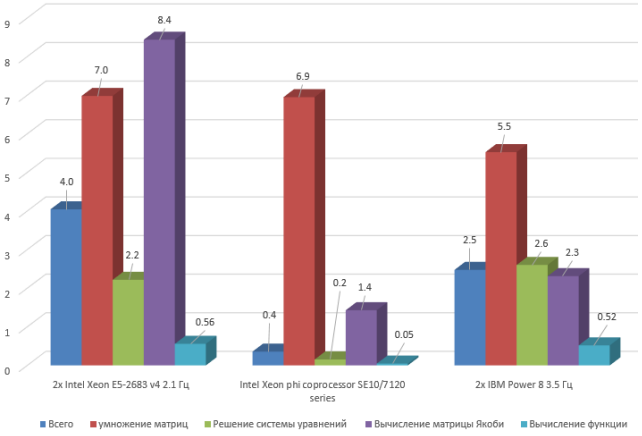
Таблица 4 – Время вычислений в многопоточном режиме

Шаг/Время сек	Intel Core i7	Intel Xeon	Intel Xeon Phi	IBM Power 8
Вычисление функции	1.6	2.8	33.5	3.0
Вычисление матрицы Якоби	155.3	18.4	108.6	67.1
Вычисление коэффициентов системы уравнений (15) – умножение матриц	21.6	3.1	3.1	3.9
Решение системы уравнений (15)	97.3	43.8	625.6	37.4
Всего	275.8	68.2	770.8	111.5

Как видно из таблицы для вычисления функции более подходит Intel Core i7 – процессор с небольшим количеством высокочастотных ядер, поддерживающих технологию AVX2. Intel Xeon и IBM Power 8 показывают вдвое меньшую производительность. Intel Xeon Phi, имеющий самые низкочастотные ядра, к тому же используемые на 1/8 мощности, оказывается более чем в 20 раз медленнее. Вычисление матрицы Якоби быстрее всего производится на Intel Xeon – 18.4 секунды, в 8.4 раза быстрее чем на Intel Core i7. IBM Power 8, из-за отсутствия поддержки AVX2, показывает средний результат 67.1 секунд. На задаче умножения матриц многоядерные системы превосходят Intel Core -7 в 5.5-7 раз. И при решении системы линейных уравнений быстродействие Intel Xeon и IBM Power 8 примерно одинаковое. Intel Xeon Phi снова оказывается аутсайдером.

Если посмотреть на задачу в целом, то ускорение по сравнению с Intel core i7 (Intel core i7 в однопоточном режиме) будет – у Intel Xeon в 4 раза (39 раз), у IBM Power 8 в 2.5 раза (24 раза). Переход на Intel Xeon Phi приводит к замедлению расчетов. Ускорение по каждой подзадаче и задаче в целом, при переходе от Intel core i7 к другим процессорам, можно посмотреть в графическом виде в таблице 5.

Таблица 5 – Ускорение расчетов по отношению к Intel Core i7



Далее приводятся таблицы 6, 7, 8, из которых видно, что при переходе к более многоядерным процессорам узким местом в задаче становится решение системы линейных уравнений. Данная проблема может быть решена за счет перехода на параллельные версии LaPack или же за счет применения параллельных версий метода сопряженных градиентов.

Таблица 6 – Относительная доля времени вычислений в многопоточном режиме для Intel core i7

Таблица 7 – Относительная доля времени вычислений в многопоточном режиме для Intel Xeon

Таблица 8 – Относительная доля времени вычислений в многопоточном режиме для Intel Xeon Phi

VII. Заключение

Исследование показало, что, используя технологии директивного многопоточного программирования и возможности компилятора, можно ускорить расчет матрицы Якоби более чем в 12 раз. Переход к более производительным процессорам Intel Xeon или IBM Power 8 дает еще прирост производительности в 2.5-4 раза, а по сравнению с однопоточным исполнением в 20-40 раз. Причем Intel Xeon за счет использования технологии AVX2 оказывается более производительным, чем IBM Power 8 (следует отметить, что метод решения был специально оптимизирован под технологию AVX2). Использование же Intel Xeon Phi, по крайней мере имеющейся в распоряжении авторов версии, затруднено в связи с отсутствием удобной поддержки компиляторами технологии AVX2/AVX512. Основываясь на опыте работы, авторы рекомендуют для задач, допускающих использование технологии AVX2/AVX512, использовать преимущественно процессоры Intel, в иных случаях, в особенности, при активной работе с оперативной памятью, использовать процессоры IBM Power 8.

Библиография

[1] Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++ //ACM Transactions on Mathematical Software (TOMS). – 2014. – Т. 40. – №. 4. – С. 26.

[2] Anderson E. et al. LAPACK Users' guide. – Society for Industrial and Applied Mathematics, 1999

[3] Федеральный исследовательский центр Информатика и управление РАН [Электронный ресурс]: сайт. – Москва: ФИЦ ИУ РАН. – URL: *http://frccsc.ru*

[4] Евтушенко Ю. Г., Зубов В. И. Об обобщенной методологии быстрого автоматического дифференцирования. Журнал вычислительной математики и математической физики, 2016, vol. 56 (11), pp. 1847-1862.

[5] Абгарян К. К., Посыпкин М. А. Программный комплекс для решения задач параметрической идентификации потенциалов межатомного взаимодействия //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 10.

[6] Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. – 1985.

[7] Евтушенко Ю. Г. Оптимизация и быстрое автоматическое дифференцирование //М.: Научное издание ВЦ РАН. – 2013.

[8] Горчаков А. Ю. Использование возможностей OpenMP для ускорения расчетов в пакете прикладного программного обеспечения Adept // CEUR Workshop Proceedings (CEUR-WS.org): Selected Papers of the II International Scientific Conference Convergent Cognitive Information Technologies (Convergent 2017), Moscow, Russia, November 24-26, 2017. — 2017. — P. 217-223. DOI:10.25559/SITITO.2017.6.630

Comparison of Intel Core-i7, Intel Xeon, Intel Xeon Phi and IBM Power 8 processors on the example of the task of restoring initial data

A.Yu Gorchakov, V.U. Malkova

Abstract - **In this paper, a comparative analysis of four types of processors is performed using the example of the problem of restoring the initial data for the transport equation. The problem is solved by the Levenberg-Marquardt method, which is decomposed into four subtasks - calculation of the vector function, calculation of the matrix of the first derivatives of the vector-function (the Jacobi matrix), matrix multiplication and the solution of the system of linear equations. The calculation of the Jacobi matrix is performed using the methodology of fast automatic differentiation, using the application software package Adept version 1.1. To speed up the calculations, we used directive multithreaded programming with the dynamic distribution of computations between threads and SIMD (Single Instruction Multiple Data) technology, the principle of computer computation, which allows to provide parallelism at the data level. The aforementioned technologies allowed to fully exploit the features of the architecture of modern Intel processors, such as multi-core / multithreading, the expansion of the command system of microprocessors Intel / AMD - Advanced Vector Extensions (AVX / AVX2), processing data in floating point format in groups of 256 bits, and FMA (Fused Multiply-Add) - a technology designed to perform a combined multiply-add operation. In comparison, the processors Intel Core i7 4770 (Haswell), Intel Xeon E5-2683V4 (Broadwell), Intel Xeon Phi coprocessor SE10 / 7120 and IBM Power 8 are given. Recommendations are given for choosing the type of processor depending on the task being solved.**

Keywords — **OpenMP, SIMD, the Jacobi matrix, the least-squares problem, the Levenberg-Marquardt method.**