МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Механико-математический факультет

Кафедра вычислительных систем

Направление подготовки «Математика и компьютерные науки», бакалавриат

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Макарова Ильи Олеговича

Разработка и исследование спектрально-разностного алгоритма для моделирования распространения упругих волн в неоднородных средах

«К защите допущен»

Заведующий кафедрой,

д.т.н., проф.

Глинский Б. М. / _____

(Фамилия,И.О.) (подпись,МП)

«____» _____ 20____ г.

«Научный руководитель»

м.н.с., ИВМиМГ СО РАН

Сапетина А.Ф. / _____

(Фамилия,И.О.) (подпись,МП)

«____» _____ 20____ г.

Дата защиты: «____» _____ 20____ г.

Новосибирск, 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

К числу достаточно новых геофизических технологий относится активный вибросейсмический мониторинг, он включает в себя методы по наблюдению и исследованию состояния земной коры по изменению различных характеристик вибро-сейсмических волн, порожденных некоторым вибрационным источником, и распространяющихся в некоторой среде.

Само исследование процесса распространения упругих волн в неоднородных средах широко используется при вибросейсмическом мониторинге различных геологических объектов. Стоит отметить, что научные группы из Института вычислительной математики и математической геофизики Сибирского отделения Российской Академии наук (ИВМиМГ СО РАН) уже достаточно давно занимаются задачами вибро-сейсмического мониторинга и имеют уникальный опыт в исследованиях.

В этом легко убедится на примерах работ Глинского Б.М, Ковалевского В.В и др. по изучению грязевого вулкана «Гора Карабетова» [1], вулкана «Шуго» и т.д.

Саму задачу вибро-сейсмического мониторинга следует разделить на 2 больших класса: прямую и обратную. При решении прямой задачи, при заданных параметрах среды, в которой распространяется волна, исследователи ставят своей целью рассчитать волновое поле. Обратная же задача, ставит своей целью найти параметры среды, в которых распространяется волна порожденная вибро-излучателем с заранее известными характеристиками. Часто реальные области исследования имеют довольно сложный рельеф, который не позволяет расположить площадную систему наблюдения для корректной постановки об-

ратной задачи. В связи с этим ее решение обычно осуществляется с помощью решения набора прямых задач на основе проведения серии вычислительных экспериментов с подбором значений параметров среды так, чтобы численное решение совпадало с результатами натурных экспериментов .

Для решения этих задач используются различные сеточные методы, которые в зависимости от особенностей моделируемой среды позволяют получить решение с необходимой точностью. В настоящее время в ИВМиМГ СО РАН накоплен опыт в создании алгоритмов и программ для решения таких задач [2, 3, 4, 5]. Однако в связи с большими масштабами реальных задач и необходимостью, в первую очередь, решать обратную задачу геофизики, через решение набора прямых задач, постоянно возникает необходимость в разработке экономичных с точки зрения используемой памяти и времени вычислений параллельных алгоритмов и программ, позволяющих с приемлемой точностью моделировать распространение упругих волн в неоднородных средах на современных многоядерных вычислительных системах различной архитектуры.

С начала 70-х годов в ИВМиМГ СО РАН начал развиваться комбинированный подход, основанный на использовании аналитического метода разделения переменных в сочетании с конечно-разностным методом решения редуцированных одномерных задач [6]. Спектральные методы являются альтернативными по отношению к стандартным конечно- разностным схемам для расчета сейсмических полей. Важным достоинством спектральных методов является высокая скорость сходимости, если решение обладает высокой степенью гладкости. Это позволяет получить хорошую точность взяв всего две-три пространственные гармоники на минимальную длину волны, что значительно меньше, чем достаточное количество узлов на длину волны при применении конечно-разностного метода второго порядка точности. Таким образом, можно получить

экономию по памяти вычислительной системы, в сочетании с высокой точностью вычислений.

С учетом современного развития архитектуры ПЛИС и разработки эффективных реализаций алгоритма быстрого преобразоваия Фурье (БПФ), дополнительную актуальность получает разработка с спектральных методов на основе разложения Фурье.

В данной работе рассматривается 2D спектрально-разностный метод, основанный на объединении конечно-разностного метода по одной координате и конечного преобразования Фурье по другой. Возникающие при этом суммы типа свертки вычисляются с помощью быстрого преобразования Фурье (БПФ).

При использовании такого подхода для сред с разрывными параметрами возникает явление Гиббса, которое можно устранить, предварительно фильтруя и сглаживая разрывные функции, чтобы получить решение сравнимое с конечно-разностным.

Таким образом, целью работы является разработка и исследование спектрально-разностного параллельного алгоритма и программного пакета на его основе для моделирования распространения упругих волн в 2D неоднородных средах.

Для достижения этой цели были поставлены и выполнены следующие задачи:

- разработать параллельный алгоритм и программное обеспечение, реализующее спектрально- разностный метод и эффективно использующее современную вычислительную архитектуру многоядерного центрального процессора (CPU);
- на примере выбранного алгоритма исследовать особенности оптимизации и адаптации программ под архитектуру многоядерных CPU;

– исследовать время работы и масштабируемость разработанного ПО в сравнении с уже имеющимися программами, реализующими конечно-разностную схему Верье.

Работа предполагает оригинальное развитие известного спектрально-разностного подхода на основе конечного преобразования Фурье к моделированию упругих волн в неоднородных средах, который может стать альтернативой стандартным конечно-разностным схемам.

Отдельно отметим практическую значимость разработки таких подходов в связи с развитием программируемых логических схем (ПЛИС), выполняющих БПФ за минимальное время.

1 СПЕКТРАЛЬНО-РАЗНОСТНОЕ МОДЕЛИРОВАНИЕ РАСПРОСТРАНЕНИЯ СЕЙСМИЧЕСКИХ ВОЛН В УПРУГИХ СРЕДАХ

В данной главе описана общая математическая модель для решения численной задачи о распространении упругих волн в 2D неоднородных средах, которые возникают в результате воздействия источника, расположенного непосредственно в самой исследуемой среде (вблизи ее поверхности).

1.1 Постановка задачи

Исследование распространения упругих сейсмических волн в неоднородных средах будет проводится на основе 2D модели теории упругости с соответсвующими начальными и граничными условиями [7, 8, 9]

В качестве среды рассматривается изотропная 2D неоднородная среда, представляющая собой прямоугольник, размера $a \times b$, $a, b > 0$, одна из границ (плоскость $z = 0$) которого является свободной поверхностью.

Прямоугольная декартова система координат введена таким образом, чтобы ось Oz была направлена вертикально вниз, а ось Ox лежала на свободной поверхности (рис. 1.1).

Рис. 1.1 Система координат

Задача записывается в терминах вектора скоростей смещений $\vec{u} = (u, w)^T$ и тензора напряжений $\vec{\sigma} = (\sigma_{xx}, \sigma_{xz}, \sigma_{zz})^T$ в следующем виде.

$$\begin{cases} \rho \dfrac{\partial \vec{u}}{\partial t} = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} \end{bmatrix} \vec{\sigma} + f(x,z,t), \\ \dfrac{\partial \vec{\sigma}}{\partial t} = \begin{bmatrix} (\lambda + 2\mu)\frac{\partial}{\partial x} & \lambda \frac{\partial}{\partial z} \\ \mu \frac{\partial}{\partial z} & \mu \frac{\partial}{\partial x} \\ \lambda \frac{\partial}{\partial x} & (\lambda + 2\mu)\frac{\partial}{\partial z} \end{bmatrix} \vec{u}, \end{cases} \tag{1.1.1}$$

где t - время, u, w - компоненты скоростей смещений по Ox и Oz соотвественно. Параметры $\lambda(x,z), \mu(x,z)$ - коэффициенты Ламе, удовлетворяющие соотношениям: $\lambda + 2\mu = \rho v_p^2$, $\mu = \rho v_s^2$, где v_p - скорость распространения продольных волн, v_s - скорость распространения поперечных волн, а $\rho(x,z) > 0, \forall x, z$ - плотность среды.

Граничные условия на свободной поверхности (плоскость $z = 0$) задаются в виде:

$$\sigma_{zz}|_{z=0} = \sigma_{xz}|_{z=0} = 0 \tag{1.1.2}$$

В качестве начальных условий, в момент времени $t = 0$ положим:

$$\begin{cases} u|_{t=0} = w|_{t=0} = 0, \\ \sigma_{xx}|_{t=0} = \sigma_{xz}|_{z=0} = \sigma_{zz}|_{z=0} = 0 \end{cases} \tag{1.1.3}$$

Также предполагается, что функция источника представима в виде $f(x,z,t) = f_x i + f_z k$, где i, k - единичные направляющие вектора соответствующих координатных осей.

В случае точечного источника типа "центр давления", расположенного в среде, функция $f(x,z,t)$ примет вид $f(x,z,t) = \delta(x-x_0)\delta(z-z_0)f(t)$, где δ - дельта функция Дирака, а x_0, z_0 - координаты источника (рис. 1.2).

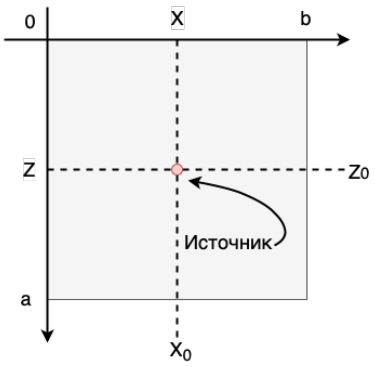

Рис. 1.2 Положение источника

Представленная система уравнений (1.1.1), в совокупности с начальными и граничными условиями (1.1.2-1.1.3), описывает распространения упругих волн в неоднородной среде с точечным истоником.

1.2 Метод решения: спектральное представление

В отличии от работы [10] метод расчета сейсмических полей для задачи (1.1.1-1.1.3) основан на альтернативном, не столь популярном, как его конечно-разностные аналоги, спектрально-разностном подходе. Этот метод обладает высокой скоростью сходимости, при достаточно гладких решениях исходной задачи, что позволяет получить решение сравнимое по точности со стандартным конечно-разностным методом.

Также в спектрально-разностном методе происходит автоматический учет характера неоднородности среды по горизонтальным переменным при переходе от одной точки разностной схемы к другой, и в зависимости от этого меняется трудоемкость вычислений. Такой подход позволяет эффективно вычислять

теоретические сейсмограммы и численные снимки для сложнопостроенных моделей реальных сред.

Рассмотрим следующие прямые преобразования Фурье для функций u, w, σ_{xx}, σ_{xz}, σ_{zz}.

$$\begin{cases} \overline{u}_k = \int_0^b u(x,z,t) \cdot sin(\overline{k}x)dx = \overline{u}_k(x,z,t), \\ \overline{w}_k = \int_0^b w(x,z,t) \cdot cos(\overline{k}x)dx = \overline{w}_k(x,z,t), \\ \overline{p}_k = \overline{\sigma}^k_{xx} = \int_0^b \sigma_{xx}(x,z,t) \cdot cos(\overline{k}x)dx = \overline{\sigma}^k_{xx}(x,z,t), \\ \overline{q}_k = \overline{\sigma}^k_{xz} = \int_0^b \sigma_{xz}(x,z,t) \cdot sin(\overline{k}x)dx = \overline{\sigma}^k_{xz}(x,z,t), \\ \overline{s}_k = \overline{\sigma}^k_{zz} = \int_0^b \sigma_{zz}(x,z,t) \cdot cos(\overline{k}x)dx = \overline{\sigma}^k_{zz}(x,z,t) \end{cases} \tag{1.2.1}$$

Где полагаем, что $\overline{k} = \frac{k\pi}{b}$.

2D спектрально-разностный метод для решения задачи (1.1.1-1.1.3), основанный на объединении конечно-разностного метода по z координате и конечного преобразования Фурье по x координате наиболее подробно описан в работе [11]. Ниже приведем только основные выкладки.

Умножим каждое уравнение системы (1.1.1) на соответсвующую базисную k-функцию и проинтегрируем по x, предполагая, что

$$\frac{1}{\rho(x,z,t)} = \frac{\overline{\rho}_0}{2} + \sum_{l=1}^{\infty} \overline{\rho}_l \cdot cos(\overline{l}x)$$

$$(\lambda + 2\mu)(x,z) = \frac{(\lambda + 2\mu)_0}{2} + \sum_{l=1}^{\infty} (\overline{\lambda} + 2\overline{\mu})_l \cdot cos(\overline{l}x)$$

$$\mu(x,z) = \frac{\overline{\mu}_0}{2} + \sum_{l=1}^{\infty} \overline{\mu}_l \cdot cos(\overline{l}x)$$

$$\lambda(x,z) = \frac{\overline{\lambda}_0}{2} + \sum_{l=1}^{\infty} \overline{\lambda}_l \cdot cos(\bar{l}x)$$

Используя формулу интегрирования по частям, формулу произведения тригонометрических функций, приводя подобные слагаемые, и замену (1.2.1) Получим следующее соотношение.

$$\begin{aligned}
&\int_0^b \frac{1}{\rho}\frac{\partial \sigma_{xx}}{\partial x} sin(\overline{k}x)dx = \int_0^b \frac{\partial \sigma_{xx}}{\partial x}\left(\frac{1}{\rho} sin(\overline{k}x)dx\right) = \\
&= \underbrace{\left[\sigma_{xx}\frac{1}{\rho} sin(\overline{k}x)\right]_0^b}_{=0} - \int_0^b \sigma_{xx}\frac{\partial}{\partial x}\left[\frac{1}{\rho} sin(\overline{k}x)\right] dx = \\
&= -\int_0^b \sigma_{xx}\frac{\partial}{\partial x}\left[\left(\frac{\overline{\rho}_0}{2} + \sum_{l=1}^{\infty}\overline{\rho}_l \cdot cos(\bar{l}x)\right) sin(\overline{k}x)\right] dx = \\
&= -\int_0^b \sigma_{xx}\frac{\partial}{\partial x}\left[\frac{\overline{\rho}_0}{2} sin(\overline{k}x) + \sum_{l=1}^{\infty}\overline{\rho}_l \cdot cos(\bar{l}x) sin(\overline{k}x)\right] dx = \\
&= -\int_0^b \sigma_{xx}\frac{\partial}{\partial x}\left[\frac{\overline{\rho}_0}{2} sin(\overline{k}x) + \frac{1}{2}\sum_{l=1}^{\infty}\overline{\rho}_l \cdot sin\left((\overline{k}+\bar{l})x\right) + \right. \\
&\left. + \frac{1}{2}\sum_{l=1}^{\infty}\overline{\rho}_l \cdot sin\left((\overline{k}-\bar{l})x\right)\right] dx = \qquad (1.2.2) \\
&= -\int_0^b \sigma_{xx}\left[k\frac{\overline{\rho}_0}{2} cos(\overline{k}x) + \frac{1}{2}\left(\sum_{l=1}^{\infty}(k+l)\overline{\rho}_l \cdot cos\left((\overline{k}+\bar{l})x\right) + \right.\right. \\
&\left.\left. + \sum_{l=1}^{\infty}(k-l)\overline{\rho}_l \cdot cos\left((\overline{k}-\bar{l})x\right)\right)\right] dx = \\
&= -k\overline{\sigma}_{xx}^{k}\frac{\overline{\rho}_0}{2} - \frac{1}{2}\left(\sum_{l=1}^{\infty}(k+l)\overline{\rho}_l \cdot \overline{\sigma}_{xx}^{(k+l)} + \sum_{l=1}^{\infty}(k-l)\overline{\rho}_l \cdot \overline{\sigma}_{xx}^{(k-l)}\right) = \\
&= -\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l \cdot \overline{\sigma}_{xx}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l \cdot \overline{\sigma}_{xx}^{(k-l)}\right) = \\
&= -\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l \cdot \overline{p}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l \cdot \overline{p}^{(k-l)}\right)
\end{aligned}$$

Аналогичным образом получаем оставшиеся соотношения.

$$\int_0^b \frac{1}{\rho}\frac{\partial \sigma_{xz}}{\partial z} sin(\overline{k}x)dx =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial \overline{\sigma}_{xz}^{(k+l)}}{\partial z} + +\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial \overline{\sigma}_{xz}^{(k-l)}}{\partial z}\right) =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial \overline{q}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial \overline{q}^{(k-l)}}{\partial z}\right);$$
$$\int_0^b \frac{1}{\rho}\frac{\partial \sigma_{xz}}{\partial x} cos(\overline{k}x)dx =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{\sigma}_{xz}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{\sigma}_{xz}^{(k-l)}\right) =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{q}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{q}^{(k-l)}\right);$$
$$\int_0^b \frac{1}{\rho}\frac{\partial \sigma_{zz}}{\partial z} cos(kx)dx = \qquad (1.2.3)$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial \overline{\sigma}_{zz}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial \overline{\sigma}_{zz}^{(k-l)}}{\partial z}\right) =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial \overline{s}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial \overline{s}^{(k-l)}}{\partial z}\right);$$
$$\int_0^b (\lambda+2\mu)\frac{\partial u}{\partial x} cos(\overline{k}x)dx =$$
$$= \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)(\overline{\lambda}_l+2\overline{\mu}_l)\cdot\overline{u}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)(\overline{\lambda}_l+2\overline{\mu}_l)\cdot\overline{u}^{(k-l)}\right);$$
$$\int_0^b \lambda\frac{\partial w}{\partial z} cos(\overline{k}x)dx = \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\lambda}_l\cdot\frac{\partial \overline{w}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty}(k-l)\overline{\lambda}_l\cdot\frac{\partial \overline{w}^{(k-l)}}{\partial z}\right);$$
$$\int_0^b \lambda\frac{\partial u}{\partial z} sin(\overline{k}x)dx = \frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\mu}_l\cdot\frac{\partial \overline{u}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty}(k-l)\overline{\mu}_l\cdot\frac{\partial \overline{u}^{(k-l)}}{\partial z}\right);$$
$$\int_0^b \mu\frac{\partial w}{\partial x} sin(\overline{k}x)dx = -\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\mu}_l\cdot\overline{w}^{(k+l)} + \sum_{l=0}^{\infty}(k-l)\overline{\mu}_l\cdot\overline{w}^{(k-l)}\right);$$
$$\int_0^b (\lambda+2\mu)\frac{\partial w}{\partial z} cos(\overline{k}x)dx =$$

$$= \frac{1}{2} \cdot \left(\sum_{l=0}^{\infty} (k+l)(\overline{\lambda}_l + 2\overline{\mu}_l) \cdot \frac{\overline{w}^{(k+l)}}{\partial z} + \sum_{l=0}^{\infty} (k-l)(\overline{\lambda}_l + 2\overline{\mu}_l) \cdot \frac{\overline{w}^{(k-l)}}{\partial z} \right);$$

$$\int_0^b \lambda \frac{\partial u}{\partial x} cos(\overline{k}x)dx = -\frac{1}{2} \cdot \left(\sum_{l=0}^{\infty} (k+l)\overline{\lambda}_l \cdot \overline{u}^{(k+l)} + \sum_{l=0}^{\infty} (k-l)\overline{\lambda}_l \cdot \overline{u}^{(k-l)} \right);$$

Таким образом, используя (1.2.2) и (1.2.3), и обозначая

$$conv(x,y)_k = \sum_{l=0}^{\infty} (k-l)x^{(k-l)}y^l$$

$$corr(x,y)_k = \sum_{l=0}^{\infty} (k+l)x^{(k+l)}y^l$$

$$sum(x,y)_k = conv(x,y)_k + corr(x,y)_k$$

можно привести исходную систему (1.1.1-1.1.3) к виду.

$$\begin{aligned}
&\frac{\partial \overline{u}_k}{\partial t} = \frac{1}{2} \cdot \sum_{l=0}^{\infty} \left(-(k+l)\overline{\rho}_l \overline{p}^{(k+l)} - (k-l)\overline{\rho}_l \overline{p}^{(k-l)} + \right. \\
&\left. +(k+l)\overline{\rho}_l \frac{\partial \overline{q}^{(k+l)}}{\partial z} + (k-l)\overline{\rho}_l \frac{\partial \overline{q}^{(k-l)}}{\partial z} \right) + \overline{f}_x^k = \\
&= \frac{1}{2} \cdot \left(-conv(\overline{\rho},\overline{p})_k - corr(\overline{\rho},\overline{p})_k + conv(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k + corr(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k \right) + \overline{f}_x^k = \\
&= \frac{1}{2} \cdot \left(-sum(\overline{\rho},\overline{p})_k + sum(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k \right) + \overline{f}_x^k; \\
&\frac{\partial \overline{w}_k}{\partial t} = \frac{1}{2} \cdot \left(sum(\overline{\rho},\overline{q})_k + sum(\overline{\rho}, \frac{\partial \overline{s}}{\partial z})_k \right) + \overline{f}_z^k; \qquad (1.2.4) \\
&\frac{\partial \overline{p}_k}{\partial t} = \frac{1}{2} \cdot \left(sum((\overline{\lambda} + 2\overline{\mu}), \overline{u})_k + sum(\overline{\lambda}, \frac{\partial \overline{w}}{\partial z})_k \right); \\
&\frac{\partial \overline{q}_k}{\partial t} = \frac{1}{2} \cdot \left(sum(\overline{\mu}, \frac{\partial \overline{u}}{\partial z})_k - sum(\overline{\mu}, \overline{w})_k \right); \\
&\frac{\partial \overline{s}_k}{\partial t} = \frac{1}{2} \cdot \left(sum((\overline{\lambda} + 2\overline{\mu}), \frac{\partial \overline{w}}{\partial z})_k + sum(\overline{\lambda}, \overline{u})_k \right);
\end{aligned}$$

Начальные и граничные условия примут вид:

$$\begin{cases} \overline{u}^k|_{t=0} = \overline{w}^k|_{t=0} = \overline{p}^k|_{t=0} = \overline{q}^k|_{t=0} = \overline{s}^k|_{t=0} = 0; \\ \overline{p}^k|_{z=0} = \overline{s}^k|_{z=0} = 0 \end{cases} \tag{1.2.5}$$

Формулы обращения для нахождения решения исходной системы (1.1.1-1.1.3) примут вид:

$$\begin{aligned} u(x,z,t) &= \frac{2}{b}\sum_{k=1}^{\infty}\overline{u}_k(k,z,t)sin(\overline{k}x) \\ w(x,z,t) &= \frac{1}{b}\overline{w}(0,z,t) + \frac{2}{b}\sum_{k=1}^{\infty}\overline{w}_k(k,z,t)cos(\overline{k}x) \\ \sigma_{xx}(x,z,t) &= \frac{1}{b}\overline{\tau}_{xx}(0,z,t) + \frac{2}{b}\sum_{k=1}^{\infty}\overline{\sigma}^k_{xx}(k,z,t)cos(\overline{k}x) \\ \sigma_{xz}(x,z,t) &= \frac{2}{b}\sum_{k=1}^{\infty}\overline{\sigma}^k_{xz}(k,z,t)sin(\overline{k}x) \\ \sigma_{zz}(x,z,t) &= \frac{1}{b}\overline{\sigma}_{zz}(0,z,t) + \frac{2}{b}\sum_{k=1}^{\infty}\overline{\sigma}^k_{zz}(k,z,t)cos(\overline{k}x) \end{aligned} \tag{1.2.6}$$

1.3 Метод решения: разностная схема для преобразованной задачи

Преобразованную, уже одномерную задачу (1.2.4-1.2.5), полученную из исходной постановки (1.1.1-1.1.3), можно решать различными конечно-разностными методами. В данной работе в качестве разностной схемы для решения (1.2.4-1.2.5) был взят одномерный аналог, хорошо себя зарекомендовавшей, конечно-разностной схемы Верье на сдвинутых сетках (рис. 1.3) [12].

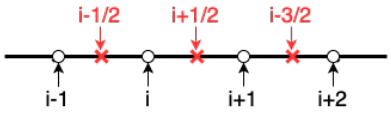

Рис. 1.3 Расположение узлов сетки

Полагая, что τ - шаг по времени, а h - по пространственной координате, получаем следующее конечно-разностное представление системы (1.2.4-1.2.5)

$$\begin{aligned}
&\frac{\overline{u}^k_{n+1,i}-\overline{u}^k_{n,i}}{\tau}=\frac{1}{2}\left[sum(\frac{\overline{q}^k_{n,i+\frac{1}{2}}-\overline{q}^k_{n,i-\frac{1}{2}}}{h},\overline{\rho}_i)_k-sum(\overline{p}_{n,i},\overline{\rho}_i)_k\right]\\
&\frac{\overline{w}^k_{n+1,i+\frac{1}{2}}-\overline{w}^k_{n+1,i+\frac{1}{2}}}{\tau}=\frac{1}{2}\left[sum(\overline{q}^k_{n,i+\frac{1}{2}},\overline{\rho}_{i+\frac{1}{2}})_k-sum(\frac{\overline{s}^k_{n,i+1}-\overline{s}^k_{n,i}}{h},\overline{\rho}_{i+\frac{1}{2}})_k\right]\\
&\frac{\overline{p}^k_{n+1,i}-\overline{p}^k_{n,i}}{\tau}=\frac{1}{2}\left[sum(\frac{\overline{w}^k_{n,i+\frac{1}{2}}-\overline{w}^k_{n,i-\frac{1}{2}}}{h},\overline{\lambda}_i)_k-sum(\overline{u}_{n,i},\overline{\lambda}_i+2\overline{\mu}_i)_k\right]+f^k_{x,n}
\end{aligned}\tag{1.3.1}$$

$$\begin{aligned}
&\frac{\overline{q}^k_{n+1,i+\frac{1}{2}}-\overline{q}^k_{n+1,i+\frac{1}{2}}}{\tau}=\frac{1}{2}\left[sum(\frac{\overline{u}^k_{n,i+1}-\overline{u}^k_{n,i}}{h},\overline{\mu}_{i+\frac{1}{2}})_k-sum(\overline{w}^k_{n,i+\frac{1}{2}},\overline{\mu}_{i+\frac{1}{2}})_k\right]\\
&\frac{\overline{s}^k_{n+1,i}-\overline{s}^k_{n,i}}{\tau}=\frac{1}{2}\left[sum(\frac{\overline{w}^k_{n,i+\frac{1}{2}}-\overline{w}^k_{n,i-\frac{1}{2}}}{h},\overline{\lambda}_i+2\overline{\mu}_i)_k-sum(\overline{u}_{n,i},\overline{\lambda}_i)_k\right]+f^k_{z,n}
\end{aligned}$$

Где функция источника аппроксимируется следующим образом:

$$f^k_{x,n}=f^k_{z,n}=\begin{cases}f(t)cos(\overline{k}x_0) & ,z=z_0\\ 0 & ,z\neq z_0\end{cases}$$

А граничные условия:

$$\frac{\overline{w}_{n+1,\frac{1}{2}}-\overline{w}_{n,\frac{1}{2}}}{\tau}=\frac{1}{2}sum(\frac{2\overline{s}^k_{n,1}}{h},\overline{\rho}_1)$$

Для того, чтобы обеспечить сходимость представленной выше явной разностной схемы, также необходимо выполнение условий Куранта:

$$\tau \leq \frac{h}{\max\limits_{z}(V_p)} \tag{1.3.2}$$

2 ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В данной главе представлено описание разработанного параллельного алгоритма и основных компонентов программного обеспечения, с помощью которого произведено исследование вычислительных особенностей спектрально-разностного алгоритма в неоднородных средах. Представлена эффективная последовательная и параллельная реализация алгоритма под системы с общей памятью, на языке программирования C++ с использованием стандарта C++17, в том числе под вычислительные системы, использующиеся в Сибирском Суперкомпьютерном центре Коллективного Пользования ИВМиМГ СО РАН (ЦКП ССКЦ ИВМиМГ СО РАН). В ходе работы проведена адаптация параллельной реализации алгоритма под архитектуры, поддерживающие векторные расширения (SIMD). Также рассмотренна одна из возможных параллельных реализаций для гетерогенных систем с распределенной памятью при помощи системы фрагментированного программирования LuNA, разрабатываемой лабораторией синтеза параллельных программ ИВМиМГ СО РАН. Проведено сравнение между программами: сгенерированной системой LuNA для вычислительных систем с общей памятью и реализованной средствами OpenMP.

2.1 Задание основных параметров модели

Для того, чтобы провести численный расчет приведенной задачи в первую очередь необходимо задать сеточную область. В настоящее время существует множество программных средств, позволяющих это реализовать, однако

большинство из них входит в состав больших коммерческих программных продуктов, использование которых избыточно для данной задачи.

В первую очередь задаются основные параметры модели: a, b - размеры исследуемоей области. Количество шагов по времени t и размеры сеточной области по вертикальной координате z, а также количество гармоник, используемых для преобразования Фурье. Для того, чтобы обеспечить сходимость явной разностной схемы, необходимо проверить выполнение условий Куранта.

Для того, чтобы не тратить вычислительные ресурсы во время исполнения основного кода и автоматически проверять корректность заданных условий перед постановкой задачи в очередь на кластере, удобно вычислять все параметры на этапе компиляции программы, условия Куранта и остальные ограничения на входные данные могут быть проверены аналогичным образом.

Средства C++17 позволяют реализовать эти проверки на этапе компиляции. Например, следующим образом.

```
using Precision = double;

constexpr auto g_z_limit_value = static_cast<Precision>(1.);
constexpr auto g_t_limit_value = static_cast<Precision>(1.);

constexpr auto g_t_grid_size = 2048u;
constexpr auto g_z_grid_size = 1024u;

constexpr auto g_k_limit = 32u;

constexpr auto g_z_grid_step = g_z_limit_value / static_cast<Precision>(
    g_z_grid_size);
constexpr auto g_t_grid_step = g_t_limit_value / static_cast<Precision>(
    g_t_grid_size);

static_assert(g_t_grid_step * g_t_grid_step <= g_z_grid_step * g_z_grid_step /
    static_cast<Precision>(2.));
```

```
static_assert(g_z_grid_step < static_cast<Precision>(1.) / (static_cast<
    Precision>(2.) * static_cast<Precision>(g_k_limit)));
```

Параметры среды $\overline{\rho}$, $\overline{\lambda}$, $\overline{\mu}$ задаются в виде 2D массивов, где первая координата соответствует индексу по пространственной сетке z, а вторая соответствует индексу гармоники k. Аналогичным образом задаются рассчитываемые величины $\overline{u}$, $\overline{w}$, $\overline{p}$, $\overline{q}$, $\overline{s}$.

После формирования массивов, заполненных коэффициентами среды, при помощи БПФ происходит переход от $\overline{\rho}$, $\overline{\lambda}$, $\overline{\mu}$ к их частотному представлению. На этом этапе также возможно проведение дополнительной предобработки этих данных (например, фильтрации) с целью улучшения дискретного представления спектра параметров среды. Все эти преобразования производятся один раз и поэтому оказывают минимальное влияние на время работы программы.

2.2 Последовательный алгоритм

Алгоритм расчета явной разностной схемы (1.3.1), представляет собой процедуру вычисления серии сумм типа сверток и корреляций, представленных в (1.2.3). Вычисление каждой из таких сумм, предварительно сопровождается умножением элементов исходных массивов на соотвествующие коэффициенты, после чего выполняется дискретная свертка или корреляция.

Так как полная реализация алгоритма средствами C++17 является достаточно громоздкой, ниже приведена только блоксхема (рис. 2.1) описываемоего алгоритма.

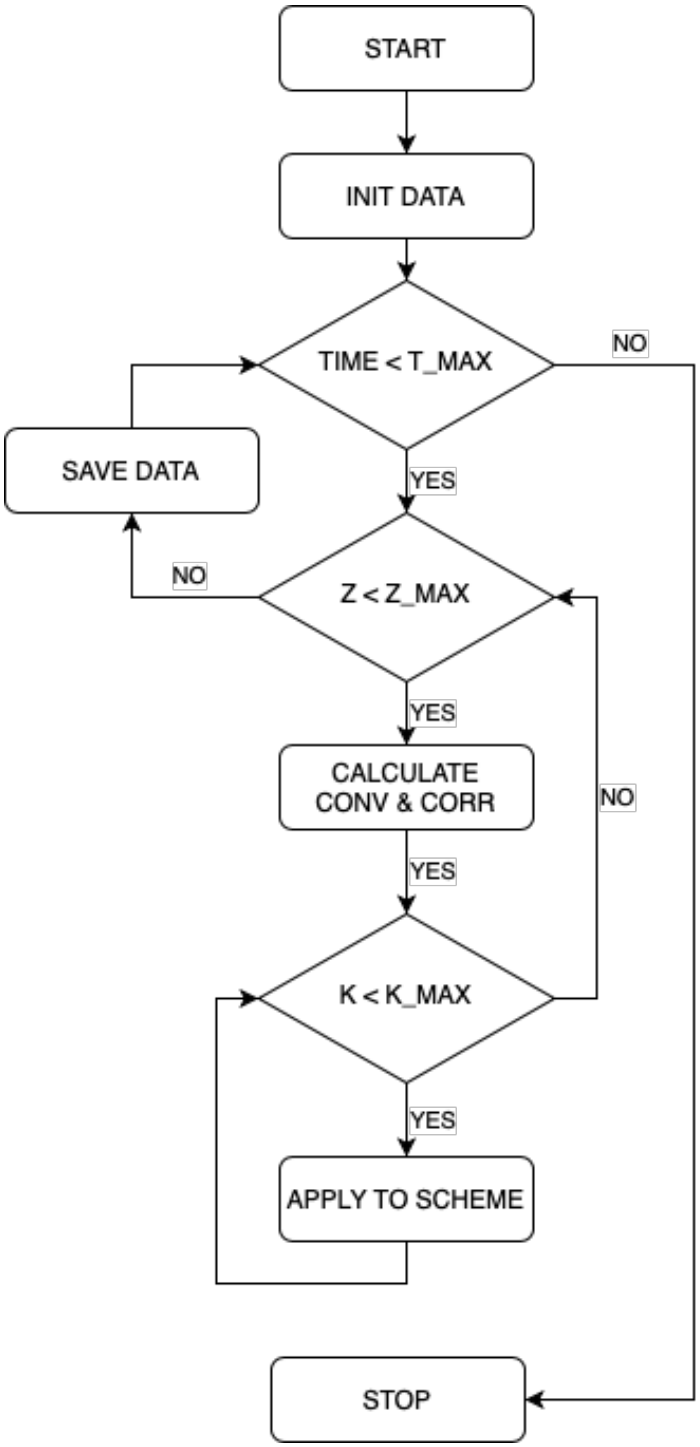

Рис. 2.1 Схема последовательного исполнения

Стоит отметить, что описываемый метод решения исходной задачи относится к так называемым memory-bounded алгоритмам. Поэтому для более эффективного доступа к памяти, и как следствие, более эффективному исполнению, становится важным правильно расположить циклы по времени, пространственной координате и гармоникам.

```
for (t=0; t < t_limit; ++t)
{
 for (z=0; z < z_limit; ++z)
 {
  //! Умножаем на соотвествующие k-коэффициенты
  calculate_coefs_corr_and_conv(values_1[z], t);
  //! Вычисляем свертки и корреляции
  calculate_all_corr_and_conv(values_1[z], t);
  for (k=0; k < k_limit; ++k)
  {
   //! Применяем схему для соотвествующих z, k и функции источника
   values_2[z][k] = apply_scheme(values_1[z][k], t, f);
  }
 }
 //! Сохраняем результат вычисления шага t+1
 save(values_2);
 //! Значения шага t+1 задаем как исходные для следующего шага
 swap(values_1, values_2);
}
```

Для реальных численных задач число шагов по z координате обычно значительно больше, чем число гармоник используемых для расчетов, так же вычисление сумм типа сверток требует больше машинного времени, чем вычисление самих коэффициентов разностной схемы. Таким образом "бутылочным горлышком" такого алгоритма является скорость вычисления сумм типа сверток и корреляций для каждого из узлов по z координате.

Реализация эффективных операций сверток и корреляций может потребовать от программиста глубоких знаний архитектуры целевой вычислительной платформы, поэтому для вычислений было принятно решение использовать, хорошо зарекомендовавшую себя библиотеку Intel MKL, которая предоставляет процедуры для эффективного вычисления сверток/корреляций.

2.3 Параллельный алгоритм

Для проведения численных экспериментов с полномасштабной моделью (с большим количеством узлов расчетной сетки и шагов по времени) требуется значительное число вычислительных ресурсов. Поэтому, чтобы получить приемлемое время расчетов для эксперимента, потребовалось провести распараллеливание исходной последовательной программы.

В настоящее время, существует множество различны многоядерных вычислительных комплексов, в числе которых, как комплексы с общей памятью, так и вычислительные кластера, в самом общем случае, с гетерогенными узлами.

В данной работе вычисления и изучение результатов распараллеливания проводились, как на персональном компьютере с процессором Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB и 32 Гб DDR4 памяти, так и на отдельных серверах с общей памятью ЦКП ССКЦ ИВМиМГ СО РАН.

Также стоит отметить, что одним из основных требований к исходной программе была ее переносимость между различными платформами, что позволило провести большое количество численных экпериметов на различных вычислительных системах без необходимости в существенной доработке исходной программы.

2.3.1 Параллельная реализация с использованием OpenMP

Для распараллеливания исходной схемы итерации в цикле по координате z распределяются между параллельными потоками средствами распараллеливания и блокирующей синхронизации OpenMP (рис. 2.2).

Заметим, что такой вариант распараллеливания не является наиболее эффективным из-за того, что не дает программисту полного контроля над исполнением парлленьной программы, в частности над операциями синхронизации, которые при данном "наивном" варианте распараллеливания занимают значительную часть времени расчета. Однако такой подход позволяет получить значительное ускорение исполнения программы, при минимальной модификации исходного пользовательского кода.

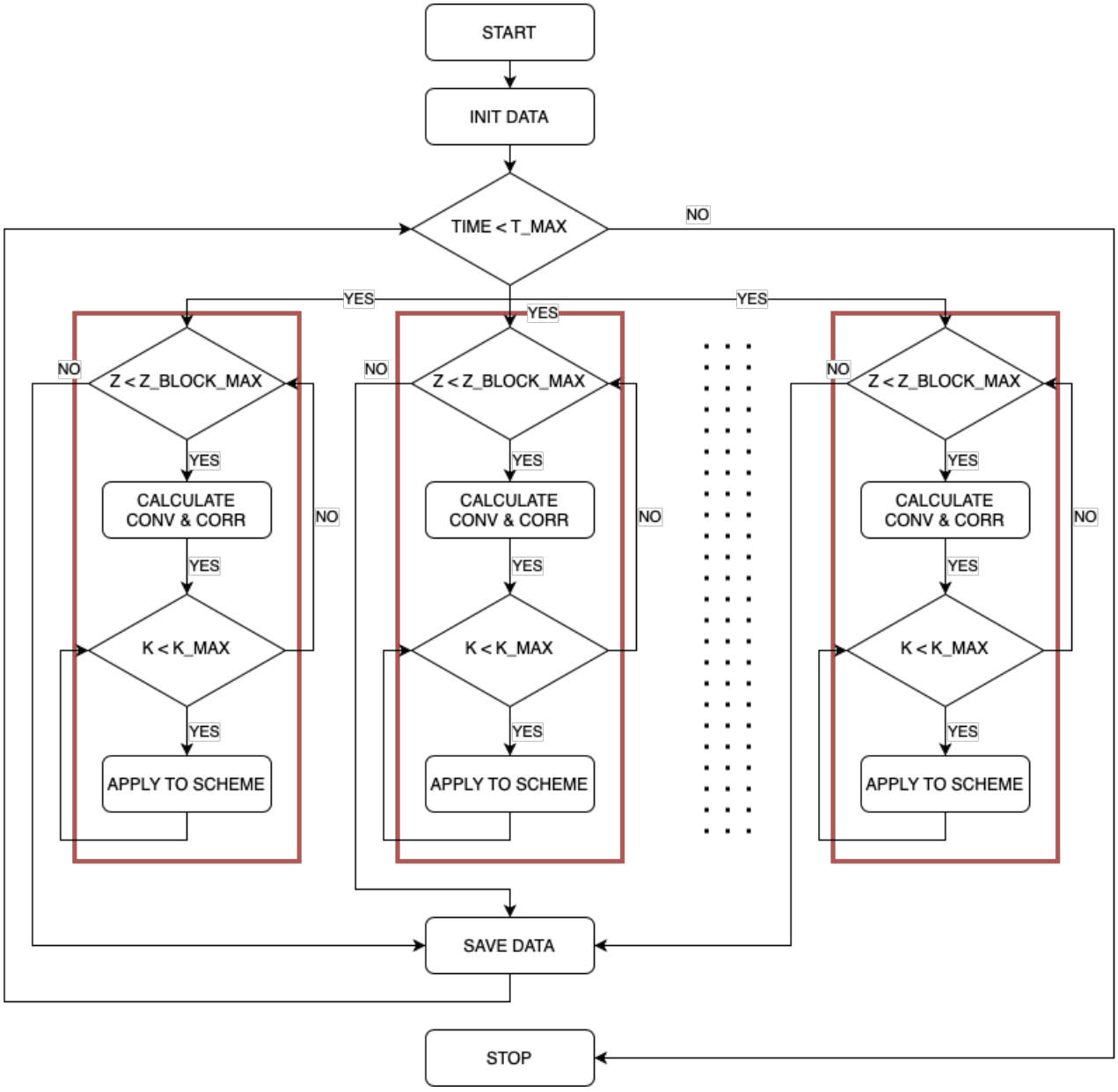

Рис. 2.2 Схема параллельного исполнения с использованием OpenMP

2.3.2 Параллельная реализация с использованием системы LuNA

Необходимым требованием к параллельной программе является эффективность ее работы. Эффективность понимается в смысле времени выполнения, расхода памяти, нагрузки на коммуникационную подсистему и т.п. Поэтому написание оптимальной программы для гетерогенных систем, реализующей представленный численный алгоритм, удовлетворяющей требованием описанным выше, может представлять значительные трудности. Так как исследователю приходится решать задачи из области системного программирования, которые не относятся к хорошо знакомой ему прикладной предметной области. В связи с этим, актуальным становится автоматизация конструирования параллельной программы. Одним из возможных решений, которое позволяет избежать описанных выше трудностей, может быть система фрагментированного программирования LuNA, разработанная лабораторией Синтеза параллельных программ под руководством В.Э.Малышкина.

Описание исходной программы для системы LuNA происходит в несколько этапов:

- описание наборов входных/выходных фрагментов данных(ФД) и вычислений(ФВ), связанных информационными зависимостями на любом из доступных языков программирования(C, C++, Fortran);
- описание фрагментированной программы на функциональном языке LuNA (приложение А);
- исполнение скомпилированного кода на системе LuNA RTS.

Алгоритм исполнения программы (рис. 2.3) состоит из следующих шагов, которые могут повторяться:

– фрагмент вычислений исполняется, при условии, что все его выходные фрагменты данных готовы;
– после завершения вычислений выходные фрагменты получают свои значения;
– после того, как были вычислены все фрагменты данных, программа завершается.

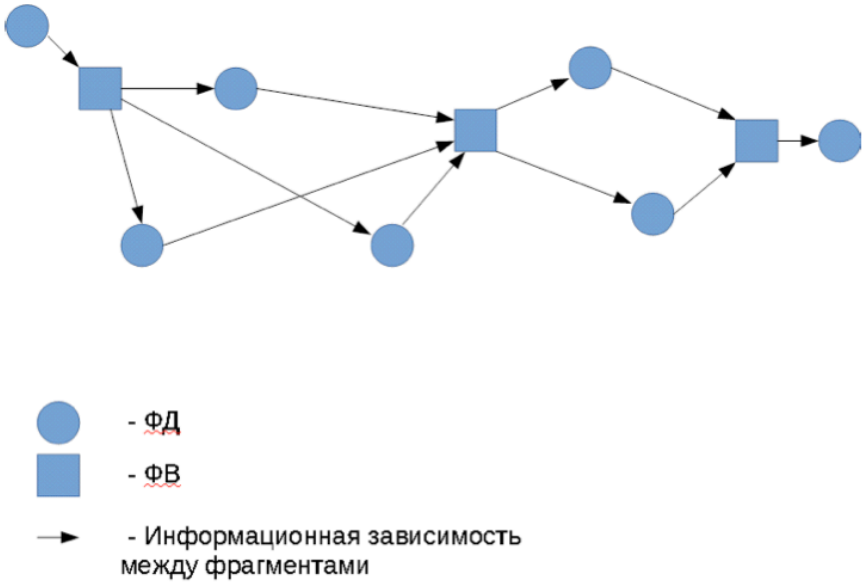

Рис. 2.3 Схема исполнения фрагментированной программы

Фрагменты вычислений являются аналогами функций процедурных языков программирования, а фрагменты данных – переменных единственного присваивания. Поток управления для программ на LuNA не задан, фрагмент вычислений исполняется только если все его входные фрагменты данных получили значения [13].

Средства C++ позволяют реализовать описание пользовательских фрагментов-вычислений в виде динамической библиотеки для дальнейшего динамического связывания(dynamic linking) с параллельным программным

кодом, сгенерированным компилятором системы LuNA, согласно соглашению о вызовах языка C.

Преимуществом такого подхода к описанию параллельной программы является его простота. Программисту не требуется решать задачи прикладного системного программирования, которые могут возникнуть, например, при использовании средств MPI. Задачи балансировки нагрузки, обмена границами и т.п. возьмет на себя система.

Полагая, что размер N сетки по z соотносится с размерами фрагментов M и их количеством K, следующим образом $N = M * K$, вычислительный процесс может быть реализован аналогичным с (рис. 2.2) образом. Главным отличием будет то, что в самом общем случае, каждый фрагмент может быть исполнен на любом из доступных узлов, в произвольном порядке, не нарушающем информационных зависимостей.

Отдельно отметим практическую значимость такого подхода. Повышение уровня программирования, что является целью разработки LuNA [13], достигается за счет следующих задач, решаемых системой исполнения:

- асинхронный запуск фрагментов-вычислений;
- выбор запускаемых фрагментов-вычислений из списка готовых фрагментов-вычислений;
- локальная балансировка нагрузки;
- порождение, распределение по распределенной памяти фрагментов данных, сборка мусора;
- обработка исключительных ситуаций, отказоустойчивость.

2.4 Адаптация и оптимизация под архитектуры

Не смотря на то, что параллельная реализация с использованием средств OpenMP и Intel MKL дает существенный выигрыш, по сравнению с последовательной реализацией, чтобы добиться пиковой производительности нужно учитывать особенности архитектуры конкретной вычислительной системы и применять точечные оптимазации отдельных программных процедур.

2.4.1 Оптимизации с использованием векторных расширений

Одними из популярных способов такого рода оптимизаций является использование специальных инструкций процессора, так называемых векторных расширений (SIMD), которые позволяют ускорить исполнение процедур, например, посредством разворачивания циклов или использования специальных векторных регистров и операций над ними.

Библиотека OpenMP позволяет векторизовать пользовательский код с помощью специальных директив препроцессора C++, а средства современных компиляторов позволяют получать отчеты о векторизации на этапе комплиляции. Например, флагами компиляции -Rpass="loop|vect" -Rpass-missed="loop|vect" -Rpass-analysis="loop|vect" для компилятора LLVM Clang.

Пример векторизованного средствами OpenMP кода на C++, для функции вычисления коэффициентов для последующего вычисления сумм типа сверток.

```
inline auto apply_corr_factor(
 const Grid1D& input,
 const Precision mult,
 Grid1D& result) noexcept
{
  #pragma omp simd
```

```
  for (auto k_idx = 0u; k_idx < input.size(); ++k_idx)
  {
    result[k_idx] = mult
     * static_cast<Precision>(input.size() - k_idx)
     * input[k_idx];
  }
}
```

При таком подходе набор используемых векторных инструкций, параметры разворачивания циклов и другие, будут подобраны библиотекой автоматически, что может быть не эффективно для некоторых программных процедур. Поэтому в данной работе исследовался вариант с ручной векторизацией, что позволило незначительно, но все же ускорить исполнение пользовательских функций.

Вариант с ручной векторизацией с использованием векторных расширений AVX2.

```
inline auto apply_corr_factor(
 const Grid1D& input,
 const Precision mult,
 Grid1D& result) noexcept
{
  static_assert(sizeof(Grid1D::value_type) == 8u);
  for (auto idx = 0u; idx < input.size() / 4u; ++idx)
  {
    inline const auto k_idx = idx * 4u;
    inline auto input_data = _mm256_load_pd(input.data() + k_idx);
    inline const auto factors_data = _mm256_set_pd(
        mult * static_cast<Precision>(input.size() - k_idx - 3u),
        mult * static_cast<Precision>(input.size() - k_idx - 2u),
        mult * static_cast<Precision>(input.size() - k_idx - 1u),
        mult * static_cast<Precision>(input.size() - k_idx));
    inline const auto factorized = _mm256_mul_pd(input_data, factors_data);
    _mm256_store_pd(result.data() + k_idx, factorized);
  }
  for (
```

```
   auto k_idx = input.size() - input.size() \% 4u;
   k_idx < input.size();
   ++k_idx)
  {
    result[k_idx] = mult
     * static_cast<Precision>(input.size() - k_idx)
     * input[k_idx];
  }
}
```

Для сравнения результатов различных векторизаций использовалась библиотека Google Benchmark. Сравнения выполнены на процессоре Intel Core-i9(16 X 2300 MHz CPUs) L3 Unified 16384 KiB, на последних доступных на момент написания работы версиях компиляторов.

На графиках для компиляторов GNU GCC и LLVM Clang (рис. 2.5, рис. 2.6) аномальных результатов не наблюдается, тогда как, для компилятора Intel C++ векторизация средствами OpenMP оказывается медленнее, чем векторизация выполненная этим компилятором автоматически (рис. 2.4).

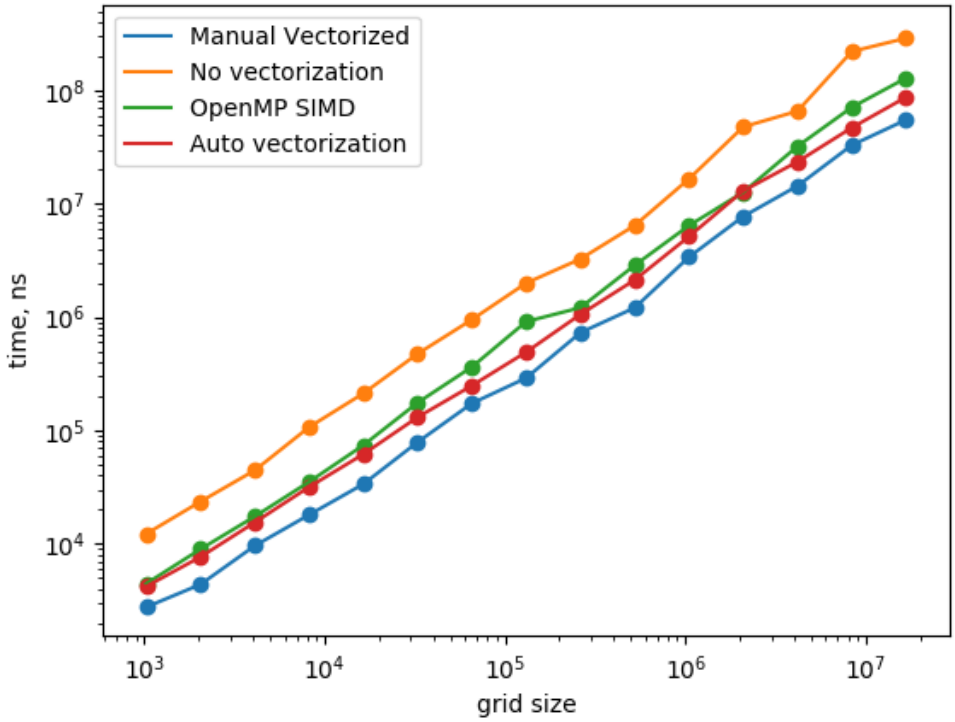

Рис. 2.4 Результаты векторизации для Intel C++. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

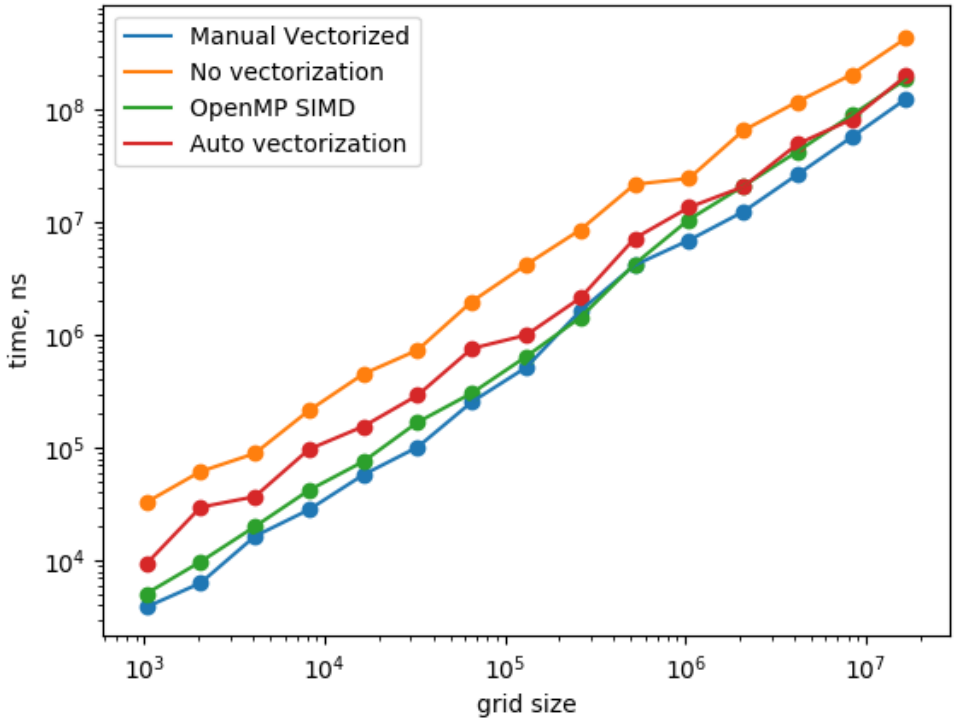

Рис. 2.5 Результаты векторизации для GNU GCC. Intel Core-i9 (16 X 2300 MHz CPUs L3 Unified 16384 KiB)

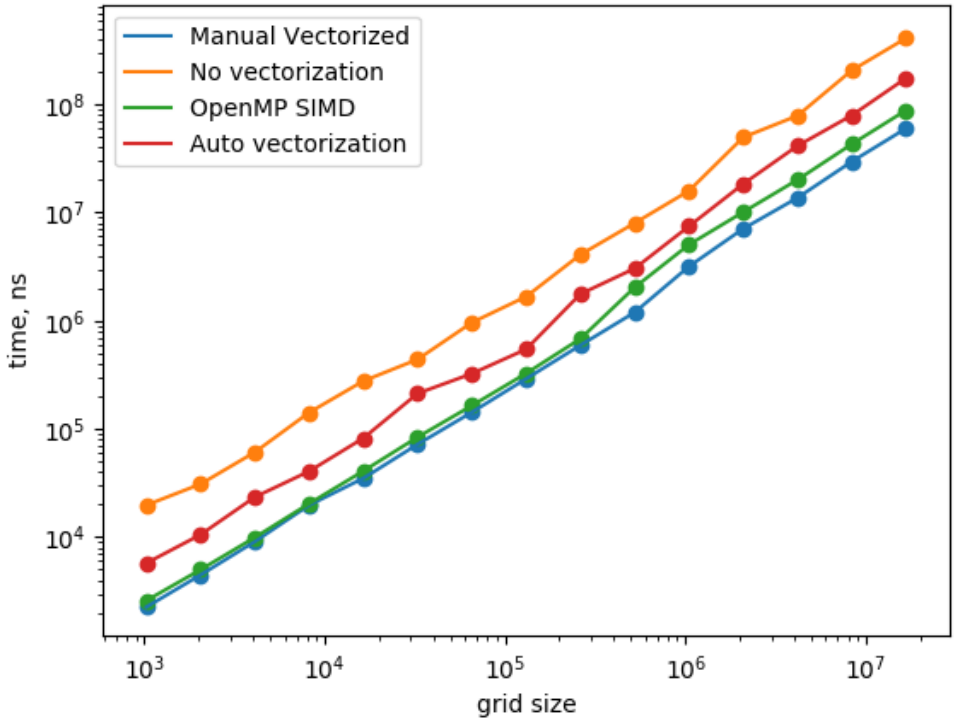

Рис. 2.6 Результаты векторизации для LLVM Clang. Intel Core-i9 (16 X 2300 MHz CPUs L3 Unified 16384 KiB)

2.4.2 Сравнение автовекторизации различных компиляторов

Современные компиляторы, при правильно заданных опциях компиляции, достаточно хорошо векторизуют код, при этом от программисту не требуется никаких дополнительных вмешательств от разработчика ПО. Поэтому еще одной темой представляющей интерес является сравнение производительности кода, скомпилированного различными компиляторами (рис. 2.7).

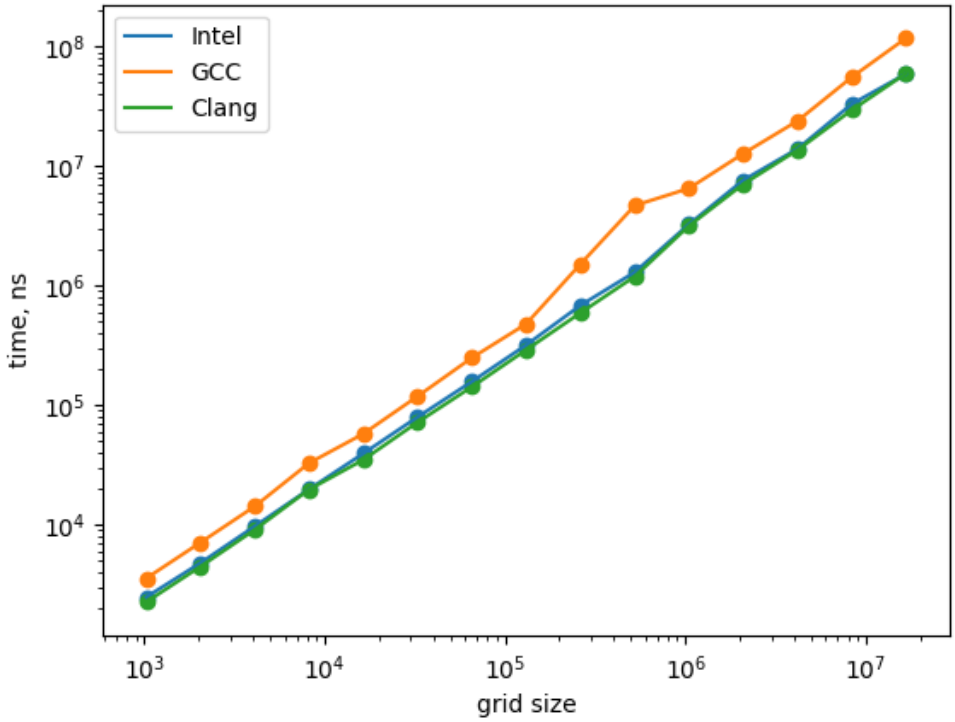

Рис. 2.7 Сравнение автовекторизации. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

Из приведенного графика видно, что наилучший результат автоматической векторизации показывают компиляторы Intel C++ и LLVM Clang, синяя и зеленая линии соответственно.

2.4.3 Выравнивание памяти

Еще одним широкоприменяемым способом ускорения исполнения векторизованного кода является выравнивание виртуальных адресов для динамически аллоцируемых объектов. Этот способ позволяет организовать более эффективный доступ к памяти вычислительной системы, по средствам более эффективной утилизации кэшей L3, L2, L1.

Стандартный аллокатор памяти C++(std::allocator) выравнивает объекты, аллоцируемые динамически, на размер их типа, огрубляя полученное значе-

ние до ближайшей степени двойки. Однако полученное таким образом значение выравнивания не может превосходить платформенно-зависимого значения sizeof(std::max_align_t), обычно равное 16 байтам.

Для большинства AVX/SSE инструкций отсутствуют требования к выравниванию операндов в памяти. Однако разработчиками этих расширений рекомендуется следить за выравниванием на размер операнда во избежание значительного снижения производительности [14].

Поэтому в рамках данной работы был реализован клаас aligned_allocator - аллокатор выравненный на 16 и 32 байт для SSE и AVX/AVX2 расширений соотвественно. Класс был реализован в соотвествии с требованиями на интерфейс аллокатора C++17, поэтому может быть использован совместно с любым контейнером из стандартной библиотеки, например, с контейнером std::vector.

3 ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В главе производится сравнение параллельной и последовательной программ реализующих, рассматриваемый в данной работе, спектрально-разностный метод. Приводятся результаты исследований параллельной программы сгенированной системой LuNA. Также представленно сравнение времени работы программ реализующих спектрально-разностный алгоритм и конечно-разностную схему Верье.

3.1 Сравнение времени работы последовательной и параллельной реализаций

Время работы спектрально-разностного алгоритма, при фиксированных физических параметрах среды и временном интервале, зависит от 2 параметров: от размера сетки по z кординате и от количества рассматриваемых гармоник.

Если положить, что количество узлов по z координате равно N, а количество рассматриваемых гармоник равно K, то ассимпточеская оценка времени работы последовательного алгоритма составит $O(N \cdot K \cdot log(K))$ для $K = 2^p$.

Если рассмотреть идеальную систему с T физическими вычислительными ядрами, то при достаточно большой значении N, ожидаемое ускорение согласно Закону Амдала не превзойдет величины $S = \frac{1}{\frac{T}{N} + \frac{1 - \frac{T}{N}}{T}}$

Для эксперимента была взята система с 8 физическими ядрами($T = 8$), значения сетки по z варьировались от 2^8 до 2^{16}. Также для того, чтобы показать, что ассимтотическая оценка сложности может быть показательной для оценки

сложности алгоритма, варьировалось число рассматриваемых гармоник (рис. 3.1).

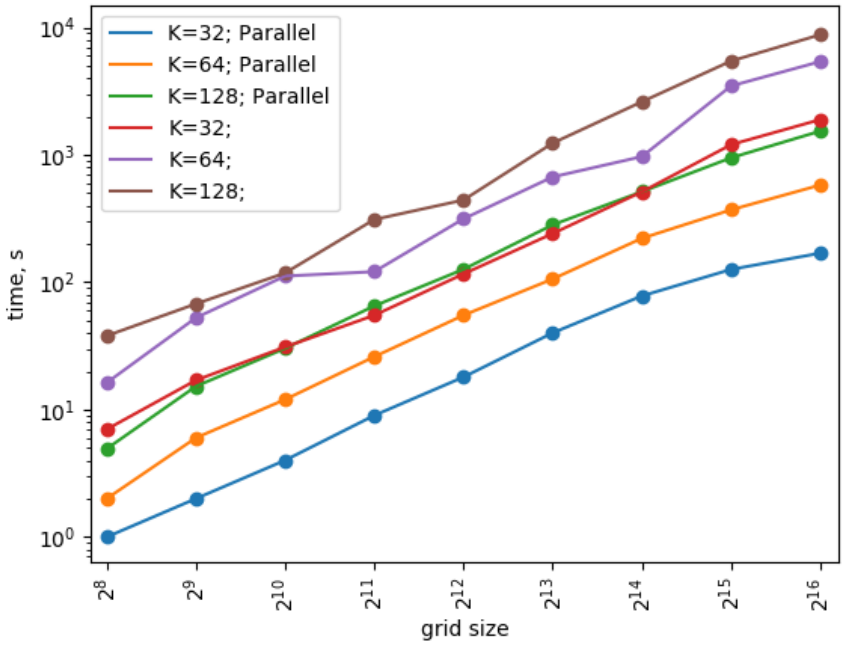

Рис. 3.1 Сравнение последовательной и параллельной реализации. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

В результате распараллеливания удается добиться прироста производительности примерно в 6-7 раз. Тогда как ожидаемое значение S по закону Амдала составляет 7,5-8 для рассматриваемых сеток. Также из графиков видно, что прирост производительности для различных значений K(числа рассматриваемых в эксперименте гармоник) не изменяется.

Другим интересным результатом является то, что при достаточно большом количестве рассматриваемых гармоник наиболее эффективный результат дает комбинация Intel MKL и OpenMP (рис. 3.2).

Из графика (рис. 3.2) видно, что при количестве гармоник большем чем 128, использование библиотеки Intel MKL дает выйгрыш производительности, тогда как при относительно маленьком числе гармоник(порядка 32), наблюдается существенное замедление.

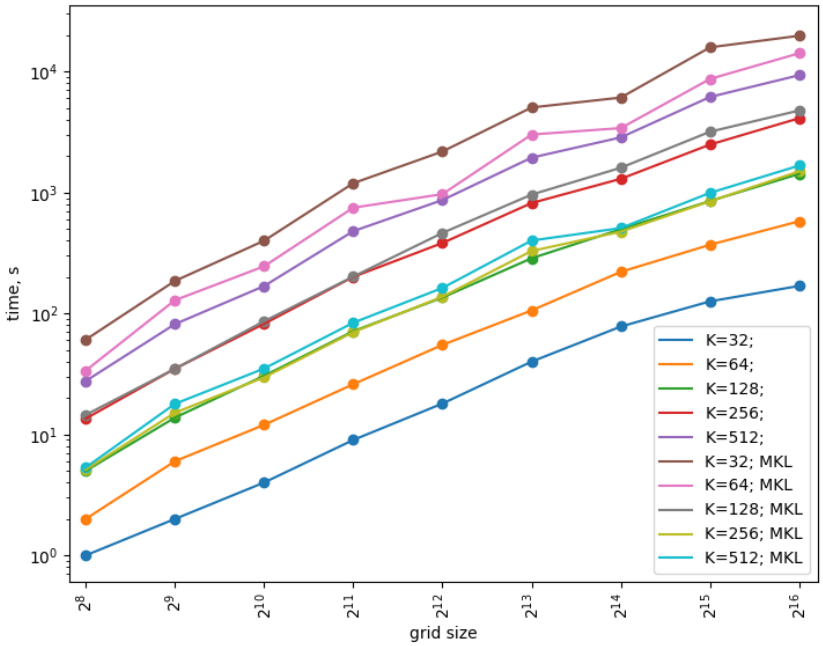

Рис. 3.2 Сравнение параллельных реализаций с/без использования Intel MKL. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

Отдельный интерес представляет тест масштабируемости параллельной реализации с использованием OpenMP (рис. 3.3). Он показывает, что величина $T(1)/T(N)$, где $T(N)$ - время работы программы для N потоков, а $T(1)$ - время ее последовательного исполнения, растет с увеличением числа потоков N практически пропорционально их количеству, пока N не превышает количества физических ядер процессора. Это говорит о хорошей сильной масштабируемости рассматриваемой параллельной реализации.

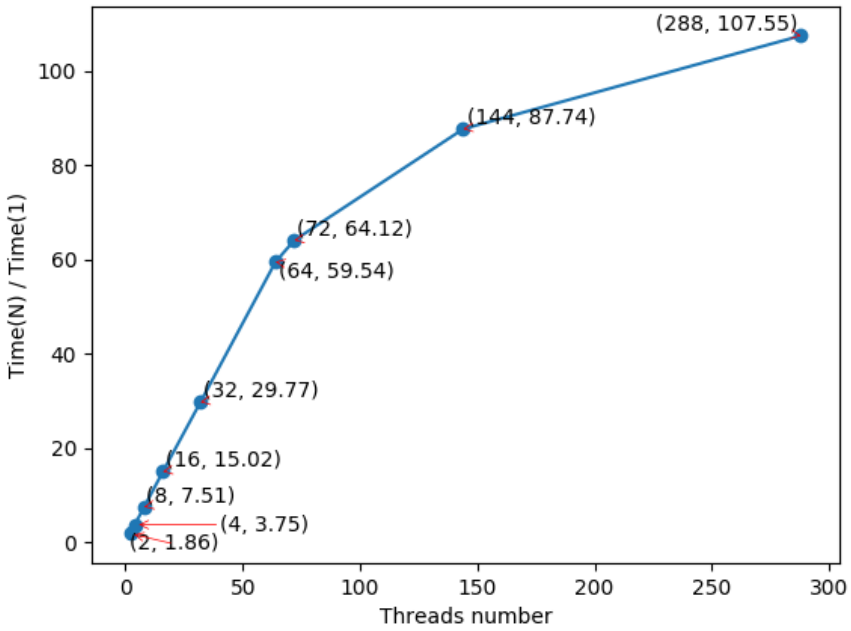

Рис. 3.3 Тест масштабируемости OpenMP Intel Xeon Phi 7290 KNL (72 X 1.5 GHz X 4 Hyper-Threading, 16GB MCDRAM)

Отметим, что технология Hyper-Threading (для $N > 72$) хоть и не значительно, но позволяет дополнительно ускорить исполнение параллельной программы.

3.2 Сравнение OpenMP и LuNA параллельных реализаций

В работе произведено сравнение производительности параллельной программы реализованной средствами OpenMP вручную и программы автоматически сконструированной системой LuNA для одного узла с общей памятью автоматически (количество фрагментов равно количеству физических ядер CPU этого узла).

Результаты сравнения показывают, что время вычислений для фрагментированной программы (LuNA time), при фиксированном числе шагов по времени (1024) и количестве гармоник (64), сопоставимо с временем вычислений про-

граммы реализованной средствами OpenMP (OMP time) (таблица 3.1). Однако количество потребляемой фрагментированной программой памяти (LuNA mem) значительно выше, чем у программы реализованной срествами OpenMP (OMP mem) (таблица 3.2).

Z	LuNA time, s	OMP time, s	Luna time / OMP time
8192	36.48	15.50	2.35
4096	18.29	8.00	2.28
2048	8.78	4.11	2.14
1024	4.76	1.70	2.8

Таблица 3.1 Сравнение времени работы LuNA и OpenMP реализаций при фиксированных количестве шагов по времени и количестве гармоник. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

Z	LuNA mem, GB	OMP mem, GB	LuNA mem / OMP mem
8192	5.87	0.05	117.4
4096	3.07	0.02	153.5
2048	1.37	0.01	137.0
1024	0.51	0.01	51.0

Таблица 3.2 Сравнение потребляемой памяти для LuNA и OpenMP реализаций при фиксированных количестве шагов по времени и количестве гармоник. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

Недостатком фрагментированной программы является резкий рост потребляемой памяти, при увеличении шагов по времени. Это связанно с неоптимальной реализацией асинхронной "сборки мусора" в системе LuNA, эта проблема может быть частично решена с помощью специальных языковых конструкций - "рекомендаций", которые позволяют "подсказать" компилятору, что данный фрагмент данных больше не используется, и память занимаемая им может быть освобождена.

Для тестов количество шагов по времени варьировалось с 2^5 до 2^{12}, а количество узлов по z и гармоник было зафиксировано (2048 и 64 соответственно) (рис. 3.4).

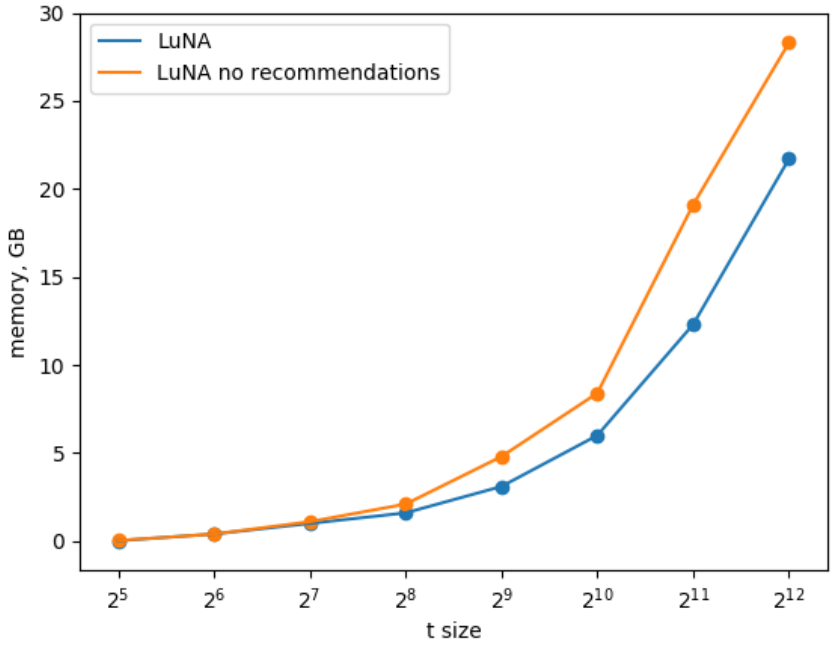

Рис. 3.4 Сравнение потребления памяти с рекомендациями и без для системы LuNA. Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB

Достоинством фрагментированной программы является то, что она может быть исполнена на вычислительных системах, в самом общем случае, с гетерогенными узлами. Тогда как у параллельной программы с ипользованием OpenMP такой возможности нет (необходимо реализовывать ее аналог, например, с использованием средств MPI).

3.3 Сравнение с конечно-разностным методом

Отдельным вопросом интересующим исследователя является вычислительная эффективность спектрально-разностного подхода (SDM).

Для конечно-разностного метода Верье (FDM) оптимальное количество узлов на длину волны составляет 40 (согласно результатам полученным в работе [12]). Для спектрально-разностного метода выбрано 5 точек на рассматриваемую длину волны [11].

Для тестов, в однородной среде с источником типа центр давления, расположенном в центре исследуемой среды, было зафиксировано число шагов по времени равное 2048, размер области по z 8192. Размер области по x варьировался от 2^9 до 2^{13} узлов. Значения K выбирались согласно рекомендациям описанным выше.

Численные эксперименты (таблица 3.3) показывают, что время работы спектрального алгоритма значительно больше, чем время работы конечно-разностной схемы Верье.

K	**SDM time, s**	**X**	**FDM time, s**	**SDM time / FDM time**
64	61	512	5.7	10.7
128	147	1024	10.9	14.7
256	407	2048	22.2	18.3
512	1777	4096	48.6	36.5
1024	5440	8192	125	43.5

Таблица 3.3 Сравнение конечно-разностного метода Верье и спектрально-разностного подхода. Intel Xeon Phi 7290 KNL (72 X 1.5 GHz X 4 Hyper-Threading, 16GB MCDRAM)

Достоинством спектрального-разностного алгоритма является то, что при количестве гармоник K и узлов по пространству N, использующихся в реальных численных задачах, он потребляет меньше памяти ($O(N \cdot K)$), чем его конечно-разностный аналог ($O(N^2)$). Однако для этих же параметров количество операций в спектрально-разностном алгоритме ($O(N \cdot K \cdot log(K))$), оказывается больше, чем количество операций в алгоритме для конечно-разностной

схемы Верье ($O(N^2)$). Этим фактом объясняется наблюдаемое замедление исполнения.

Другим интересным результатом является, что при увеличении размера области по z для спектрально-разностного метода, при фиксированном x для конечно-разностной схемы Верье, разница во времени исполнения между этими двумя алгоритмами уменьшается.

Для исследования данного эффекта, было зафиксировано число шагов по времени равное 2048, число гармоник 256, размер области по x равный 2048. Размер расчетной области по z координате варьировался от 2^{11} до 2^{14} (таблица 3.4).

Z	**SDM time, s**	**FDM time, s**	**SDM time / FDM time**
2048	111.0	5.1	21.76
4096	211.0	10.6	19.79
8192	407.0	22.2	18.33
16384	800.0	48.2	16.60

Таблица 3.4 Сравнение времени работы спектрально-разностного и конечно-разностного методов для различных значений z. Intel Xeon Phi 7290 KNL (72 X 1.5 GHz X 4 Hyper-Threading, 16GB MCDRAM)

Более удачной архитектурой для реализации спектрально-разностного алгоритма может стать программируемая логическая схема (ПЛИС). Время выполнения операций типа сверток (реализованных при помощи БПФ) на таких схемах может быть значительно меньше, чем время выполнения на стандартных CPU. Например, для 256 точечного алгоритма БПФ время обработки на CPU (Intel Core-i9 (16 X 2300 MHz CPUs) L3 Unified 16384 KiB) составляет величину порядка 2 мс., в то время как аналогичный алгоритм реализованный на

ПЛИС выполняется 0.1 мс [15]. Таким образом время исполнения спектрально-разностного алгоритма может быть существенно уменьшено.

ЗАКЛЮЧЕНИЕ

В результате выполнения квалификационной работы бакалавра был изучен спектрально-разностный подход к вычислению сейсмических полей в задачах геофизики.

Разработан параллельный алгоритм, последовательное и параллельное программное обеспечение (ПО), эффективно реализующие спектрально- разностный метод на современных многоядерных вычислительных системах.

Исследованы особенности оптимизации и адаптации под архитектуру многоядерных CPU при реализации спектрально-разностного метода. Проведено тестирование времени работы и масштабируемости разработанного программного обеспечения, как на персональных компьютерах, так и на специализированных вычислительных системах ЦКП ССКЦ ИВМиМГ СО РАН, показавшие эффективность распараллеливания.

Реализовано сравнение времени расчетов на основе исследуемого спектрального-разностного подхода с расчетами с использованием конечно-разностной схемы Верье.

В дальнейшие планы входит более подробное исследование поведения рассматриваемого алгоритма в гетерогенных средах, а также подбор низкочастотных фильтров для необходимой предобработки параметров среды с целью улучшения качества получаемого в результате расчетов решения. Также будет продолжена работа над эффективной реализацией параллельной программы средствами системы фрагментированного программирования LuNA.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Глинский Б. М., Караваев Д. А., Ковалевский В. В., Мартынов В. Н. Численное моделирование и экспериментальные исследования грязевого вулкана «Гора Карабетова» вибросейсмическими методами. // Вычислительные методы и программирование. Москва, 2010.

2. Коновалов А. Н. Численные методы в динамических задачах теории упругости. // Сиб. матем. журн., 1997, том 38, номер 3, С. 551–568

3. B.G. Mikhailenko. Spectral Laguerre method for the approximate solution of time dependent problems. // Appl. Math. Letters, 1999, № 12, P. 105-110.

4. Terekhov A. V. Spectral-difference parallel algorithm for the seismic forward modeling in the presence of complex topography. // Journal of Applied Geophysics, 2015, Vol. 115., P. 206–219.

5. Фатьянов А.Г. Полуаналитический метод решения прямых динамических задач в слоистых средах. // ДАН, т. 310, 1990, №2, С. 323-327

6. Alekseev A.S., Mikhailenko B.G. The solution of dynamic problems of elastic wave propagation in inhomogeneous media by a combination of partial separation of variables and finite difference methods. // International Journal of Geophysics, 1980, Vol. 48, P. 161-172

7. Новацкий В. Теория упругости. // Москва, Мир, 1975, 872 с.

8. Филоненко-Бородич М.М. Теория упругости. // Государственное издательство физико-математической литературы, Москва, 1959, 365 с.

9. Шемякин Е.И. Динамические задачи теории упругости и пластичности. // Курс лекций для студентов. НГУ, Новосибирск, 1968. 336 с.

10. Караваев Д. А. Чиссленное моделирование 3D волновых полей в задачах сейсмического зондирования вулканических структур. // Диссертация на соискание ученой степени кандидата физико-математических наук. Новосибирск, 2011. С. 19-21.

11. Мартынов В. Н. Волновые поля от сосредоточенных источников в трансверсально-изотропных средах. // Физика земли, 1986, № 11.

12. Virieux J. PSV wave propagation in heterogeneous media: Velocitystress finitedifference method // Geophysics, Volume 51, April 1986, Number 4, P. 889-901

13. С.В. Мачульскис Разработка и реализация алгоритмов статического анализа фрагментированных программ. // Квалификационная работа магистра, Новосибисрк, 2015, С. 16-17

14. Intel Россия Каталог программных продуктов семейства Intel [Электронный ресурс]. // URL: https://intel.com

15. Иванова Н. Н., Галанина Н. А., Моисеев Д. В. Особенности реализации алгоритма БПФ на ПЛИС типа FPGA // Вестник увашского университета, 2018, №3.

ПРИЛОЖЕНИЕ

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ НА ЯЗЫКЕ LUNA

```
import init_env(int #fgnum, int #fgsize, name) as init_env;
import check_env(value #env, int #fgnum, int #size) as check_env;
import init_values(int #fgnum, int #fgsize, name) as init_values;
import check_values(value #values, int #fgnum, int #size) as check_values;
import calculate_one_step(
 value #prev_values,
 value #env,
 int #t_idx,
 int #fgnum,
 int #fgsize,
 int #fgcount,
 name #values) as calculate_one_step;
import check_one_step(
 value #prev_values,
 value #values,
 value #env,
 int #t_idx,
 int #fgnum,
 int #fgsize,
 int #fgcount) as check_one_step;

##const FG_SIZE 512
##const FG_COUNT 4
##const T_IDX_LIMIT 1024

sub main() {
  df env, values, last_iter;
  init_env($FG_COUNT, $FG_SIZE, env);
  check_env(env, $FG_COUNT, $FG_SIZE);
  for fg_num = 0..$FG_COUNT-1 {
    init_values(fg_num, $FG_SIZE, values[0][fg_num]);
```

```
  }
  while (int(it < $T_IDX_LIMIT)), it = 0 .. out last_iter {
    for fg_num = 0..$FG_COUNT-1 {
      calculate_one_step(
        values[it][fg_num],
        env,
        it,
        fg_num,
        $FG_SIZE,
        $FG_COUNT,
        values[it + 1][fg_num]) --> (values[it][fg_num]);
    }
  }
  for fg_num = 0..$FG_COUNT-1 {
   save_fg(values[last_iter][fg_num]) --> values[last_iter][fg_num];
  }
}
```