СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6

1 СПЕКТРАЛЬНО-РАЗНОСТНОЕ МОДЕЛИРОВАНИЕ РАСПРОСТРАНЕНИЯ СЕЙСМИЧЕСКИХ ВОЛН В УПРУГИХ СРЕДАХ 10

1.1 Постановка задачи 10

1.2 Метод решения и вывод основных формул 12

1.2.1 Разностная схема для преобразованной задачи 16

1.2.2 Фильтрация (+аппроксимация) 18

2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 19

2.1 Задание основных параметров модели 19

2.2 Последовательный алгоритм 21

2.3 Параллельный алгоритм 23

2.3.1 Параллельная реализация с использованием OpenMP . . 24

2.3.2 Параллельная реализация с использованием системы LuNA 25

2.4 Адаптация и оптимизация под архитектуры 27

2.4.1 Оптимизации с использованием векторных расширений 27

2.4.2 Сравнение автовекторизации различных компиляторов . 29

2.4.3 Выравнивание памяти 29

3 СРАВНЕНИЕ С КОНЕЧНО-РАЗНОСТНЫМ МЕТОДОМ 31

3.1 Точность решения 31

3.2 Время работы ПО 31

ЗАКЛЮЧЕНИЕ 32

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 33

ПРИЛОЖЕНИЕ А 35

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 распростроняющихся 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: статья Глинского Б.М, Ковалевского В.В и др. по изучению грязевого вулкана «Гора Карабетова». (todo reference)

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 найти параметры распространения волны, например, продольные и поперечные скорости волн. Обратная же задача, ставит своей целью найти параметры среды, в которых распространяется волна порожденная вибро-излучателем, с заранее известными характеристиками, и связи с тем, что реальная область исследования имеет довольно сложный рельеф,

который может не позволять поставить площадную систему наблюдения для корректной постановки обратной задачи, обычно решается с помощью набора прямых задач, на основе проведения серии вычислительных экспериментов. (todo reference на работы).

Для решения геофизических задач необходим определенный математический аппарат. В настоящее время в ИВМиМГ СО РАН накоплен опыт в создании алгоритмов и программ для решения таких задач. 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.

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 при применении конечно-разностного метода второго порядка точности. Таким образом, можно получить экономию по памяти вычислительной системы, в сочетании с высокой точностью вычислений.

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 спектрально-разностного параллельного алгоритма и программного пакета на его основе для моделирования распространения упругих волн в 2D неоднородных слоистых средах с разрывными параметрами и исследование качества и времени решения по сравнению с конечно-разностным решением аналогичной задачи.

Для достижения этой цели были поставлены и выполнены следующие задачи:

- разработать и оптимизировать параллельное программное обеспечение, реализующее спектрально- разностный метод и эффективно использующее современную вычислительную архитектуру многоядерного центрального процессора;
- исследовать время работы и масштабируемость разработанного ПО в сравнении с уже имеющимися программами, реализующими конечно-разностную схему Верье;
- подобрать фильтр и интерполяционный полином для сглаживания разрывных параметров слоистой среды таким образом, чтобы численно получить волновое поле, качественно сравнимое с результатом конечно-разностного решения;

Работа предполагает оригинальное развитие известного спектрально-разностного подхода на основе конечного преобразования Фурье к моделирова-

нию 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.

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. (todo reference)

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. (todo picture)

Задача будет записана 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 соответственно. 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:

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

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

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

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.

Представленная система уравнений, в совокупности с начальными и граничными условиями, описывает распространения упругих волн в неоднородной среде с точечным истоником.

1.2 Метод решения и вывод основных формул

В отличии от работы (todo reference) 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.

Рассмотрим следующие прямые преобразования фурье для функций u, w, σ_{xx}, σ_{xz}, σ_{zz}. (todo написать про четность)

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

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

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 (todo reference). Ниже приведем только основные выкладки.

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

$$\frac{1}{\rho(x,z,t)} = \frac{\overline{\rho}_0}{2} + \sum_{l=1}^{\infty} \overline{\rho}_l \cdot cos(lx) = \frac{\overline{\rho}_0(0,z)}{2} + \sum_{l=1}^{\infty} \overline{\rho}_l(l,z) \cdot cos(lx)$$

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

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

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

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

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

$$+\frac{1}{2}\sum_{l=1}^{\infty}\overline{\rho}_l\cdot sin\left((k-l)x\right)\Bigg]dx= \tag{1.2.2}$$

$$=-\int_0^b\sigma_{xx}\left[k\frac{\overline{\rho}_0}{2}cos(kx)+\frac{1}{2}\left(\sum_{l=1}^{\infty}(k+l)\overline{\rho}_l\cdot cos\left((k+l)x\right)+\right.\right.$$

$$\left.\left.+\sum_{l=1}^{\infty}(k-l)\overline{\rho}_l\cdot cos\left((k-l)x\right)\right)\right]dx=$$

$$=-k\overline{\sigma}_{xx}^k\frac{\overline{\rho}_0}{2}-\frac{1}{2}\left(\sum_{l=1}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{\sigma}_{xx}^{(k+l)}+\sum_{l=1}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{\sigma}_{xx}^{(k-l)}\right)=$$

$$=-\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{\sigma}_{xx}^{(k+l)}+\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{\sigma}_{xx}^{(k-l)}\right)=$$

$$=-\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{p}^{(k+l)}+\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{p}^{(k-l)}\right)$$

Аналогичным образом получаем оставшиеся соотношения.

$$\int_0^b\frac{1}{\rho}\frac{\partial\sigma_{xz}}{\partial z}sin(kx)dx=$$

$$=\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial\overline{\sigma}_{xz}^{(k+l)}}{\partial z}++\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial\overline{\sigma}_{xz}^{(k-l)}}{\partial z}\right)=$$

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

$$\int_0^b\frac{1}{\rho}\frac{\partial\sigma_{xz}}{\partial x}cos(kx)dx=$$

$$=\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{\sigma}_{xz}^{(k+l)}+\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{\sigma}_{xz}^{(k-l)}\right)=$$

$$=\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\overline{q}^{(k+l)}+\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\overline{q}^{(k-l)}\right);$$

$$\int_0^b\frac{1}{\rho}\frac{\partial\sigma_{zz}}{\partial z}cos(kx)dx= \tag{1.2.3}$$

$$=\frac{1}{2}\cdot\left(\sum_{l=0}^{\infty}(k+l)\overline{\rho}_l\cdot\frac{\partial\overline{\sigma}_{zz}^{(k+l)}}{\partial z}+\sum_{l=0}^{\infty}(k-l)\overline{\rho}_l\cdot\frac{\partial\overline{\sigma}_{zz}^{(k-l)}}{\partial z}\right)=$$

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

$$\int_0^b (\lambda + 2\mu)\frac{\partial u}{\partial x} cos(kx)dx =$$

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

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

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

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

$$\int_0^b (\lambda + 2\mu)\frac{\partial w}{\partial z} cos(kx)dx =$$

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

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

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.

$$\frac{\partial \overline{u}_k}{\partial t} = \frac{1}{2} \cdot \sum_{l=0}^{\infty} \Big(-(k+l)\overline{\rho}_l \overline{p}^{(k+l)} - (k-l)\overline{\rho}_l \overline{p}^{(k-l)} +$$

$$+(k+l)\overline{\rho}_l \frac{\partial \overline{q}^{(k+l)}}{\partial z} + (k-l)\overline{\rho}_l \frac{\partial \overline{q}^{(k-l)}}{\partial z} \Big) + \overline{f}_x^k =$$

$$= \frac{1}{2} \cdot \left(-conv(\overline{\rho}, \overline{p})_k - corr(\overline{\rho}, \overline{p})_k + conv(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k + corr(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k \right) + \overline{f}_x^k =$$

$$= \frac{1}{2} \cdot \left(-sum(\overline{\rho}, \overline{p})_k + sum(\overline{\rho}, \frac{\partial \overline{q}}{\partial z})_k \right) + \overline{f}_x^k;$$

$$\frac{\partial \overline{w}_k}{\partial t} = \frac{1}{2} \cdot \left(sum(\overline{\rho}, \overline{q})_k + sum(\overline{\rho}, \frac{\partial \overline{s}}{\partial z})_k \right) + \overline{f}_z^k; \tag{1.2.4}$$
$$\frac{\partial \overline{p}_k}{\partial t} = \frac{1}{2} \cdot \left(sum((\overline{\lambda}_l + 2\overline{\mu}_l), \overline{u})_k + sum(\overline{\lambda}, \frac{\partial \overline{w}}{\partial z})_k \right);$$
$$\frac{\partial \overline{q}_k}{\partial t} = \frac{1}{2} \cdot \left(sum(\overline{\mu}, \frac{\partial \overline{u}}{\partial z})_k - sum(\overline{\mu}, \overline{w})_k \right);$$
$$\frac{\partial \overline{s}_k}{\partial t} = \frac{1}{2} \cdot \left(sum((\overline{\lambda}_l + 2\overline{\mu}_l), \frac{\partial \overline{w}}{\partial z})_k + sum(\overline{\lambda}, \overline{u})_k \right);$$

Начальные и граничные же условия примут вид (todo)

$$123todo \tag{1.2.5}$$

1.2.1 Разностная схема для преобразованной задачи

Новую, 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.

todo описание сетки

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

$$\frac{\overline{u}^k_{n+1,i} - \overline{u}^k_{n,i}}{\tau} = \frac{1}{2} \left[sum(\frac{\overline{q}^k_{n,i+\frac{1}{2}} - \overline{q}^k_{n,i-\frac{1}{2}}}{h}, \overline{\rho}_i)_k - sum(\overline{p}_{n,i}, \overline{\rho}_i)_k \right]$$
$$\frac{\overline{w}^k_{n+1,i+\frac{1}{2}} - \overline{w}^k_{n+1,i+\frac{1}{2}}}{\tau} = \frac{1}{2} \left[sum(\overline{q}^k_{n,i+\frac{1}{2}}, \overline{\rho}_{i+\frac{1}{2}})_k - sum(\frac{\overline{s}^k_{n,i+1} - \overline{s}^k_{n,i}}{h}, \overline{\rho}_{i+\frac{1}{2}})_k \right]$$

$$\frac{\overline{p}^k_{n+1,i} - \overline{p}^k_{n,i}}{\tau} = \frac{1}{2}\left[sum(\frac{\overline{w}^k_{n,i+\frac{1}{2}} - \overline{w}^k_{n,i-\frac{1}{2}}}{h}, \overline{\lambda}_i)_k - sum(\overline{u}_{n,i}, \overline{\lambda}_i + 2\overline{\mu}_i)_k\right] + f^k_{x,n}$$

(1.2.6)

$$\frac{\overline{q}^k_{n+1,i+\frac{1}{2}} - \overline{q}^k_{n+1,i+\frac{1}{2}}}{\tau} = \frac{1}{2}\left[sum(\frac{\overline{u}^k_{n,i+1} - \overline{u}^k_{n,i}}{h}, \overline{\mu}_{i+\frac{1}{2}})_k - sum(\overline{w}^k_{n,i+\frac{1}{2}}, \overline{\mu}_{i+\frac{1}{2}})_k\right]$$

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

Где функция источника представляется в виде:

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

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:

$$\tau \leq \frac{h}{\max\limits_z (V_p)} \tag{1.2.7}$$

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

todo описание формул обращения

1.2.2 Фильтрация (+аппроксимация)

todo

2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В данной главе представленно описание основых компонентов разработанного параллельного программного обеспечения, с помощью которого произведено исследование спектрально-разностного алгоритма в однородных 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: сгенированной 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. Шаг 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, 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 соответсвует индексу по пространственной сетке z, а вторая соответствует индексу гармоники k Аналогичным образом задаются расчитываемые величины $\overline{u}$, $\overline{w}$, $\overline{p}$, $\overline{q}$, $\overline{s}$.

2.2 Последовательный алгоритм

Алгоритм расчета явной разностной схемы (1.2.6), 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.

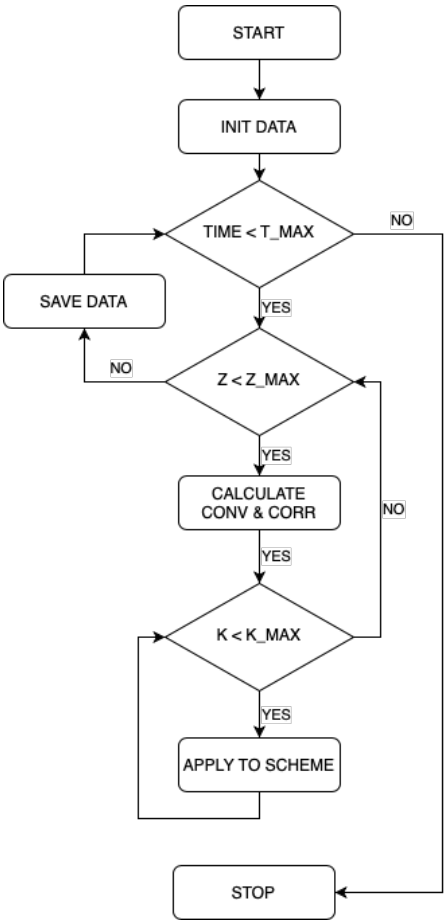

Рис. 2.1 Схема последовательного исполнения

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 рассчетов, так же вычисление сумм типа сверток требует больше машинного времени, чем вычисление самих коэфициентов 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 приемлимое 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 2,4 ГГц и 32 Гб DDR4 памяти, так и на отдельных серверах с общей памятью ЦКП ССКЦ ИВМиМГ СО РАН с конфигурацией (todo).

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.

todo графики сравнения параллельной и последовательной реализации

todo графики количества потоков параллельной реализации

Заметим, что такой вариант распараллеливания не является наиболее эффективным из-за того, что не дает программисту полного контроля над исполнением паралленьной программы, в частности над операциями синхронизации, которые при данном "наивном" варианте распараллеливания занимают значительную часть времени расчета. Однако такой подход позволяет получить зна-

чительное ускорение исполнения программы, при минимальной модификации исходного пользовательского кода.

Наиболее эффективный результат дает комбинация Intel MKL и OpenMP.

todo графики

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. В связи с этим, актуальным становится автоматизация конструирования параллельной программы. Одним из возможных решений может быть система фрагментированного программирования LuNA, разработанная лабораторией Синтеза параллельных программ под руководством В.Э.Малышкина. (todo reference)

Описание исходной программы для системы LuNA происходит в несколько этапов:

- Описание наборов фрагментов пользовательского исходного кода на любом из доступных языков программирования(C, C++, Fortran);
- Описание схемы параллельной программы на функциональном языке LuNA;
- Исполнение скомпилированного кода на системе LuNA RTS;

Преимуществом такого подхода к описанию параллельной программы является его простота. Программисту требуется только описать фрагменты и

задать порядок их исполнения, межпоточная синхронизация, обмен информацией между узлами (например, средствами MPI), балансировку нагрузки и т.п. возьмет на себя система кодогенерации.

Средства C++ позволяют реализовать описание пользовательских фрагментов в виде динамической библиотеки, для дальнейшего динамического связывания(dynamic linking) с параллельным программным кодом, сгенерированным системой LuNA, согласно соглашению о вызовах языка C.

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 следующим образом.

(todo схема исполнения параллельной программы)

Схема параллельной программы на языке LuNA представляет собой высокоуровненовое описание для реализованных на предыдущем шаге фрагментов. Например, схема для ранее поставленной задачи может быть описана в виде функциональной программы (todo ref).

Все дальнейшее действия по распределению нагрузки и порядке исполнения независимых фрагментов программы реализует программный модуль LuNA RTS.

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 1).

Стоит также отметить, что реализация аналогичной программы для систем с распределенной памятью, например, средствами MPI представляет значительные трудности, по сравнению с реализацией на системе LuNA, где от пользователя потребуется только задать количество исполняемых фрагментов.

(todo сравнение)

2.4 Адаптация и оптимизация под архитектуры

Не смотря на то, что параллельная реализация с использованием средств OpenMP и Intel MKL дает существенный выйгрыш, 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 последющего 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.

(todo оформленные результаты бенчмарков)

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.

(todo intel vs gcc vs clang)

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 значение 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.

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.

(todo aligned vs default)

3 СРАВНЕНИЕ С КОНЕЧНО-РАЗНОСТНЫМ МЕТОДОМ

3.1 Точность решения

todo

3.2 Время работы ПО

todo

ЗАКЛЮЧЕНИЕ

todo

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Каталог программных продуктов семейства Intel [Электронный ресурс] // Intel Россия URL: https://intel.com (дата обращения: 25.04.2020)

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];
  }
}
```