КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2020 Т. 12 № 4 С. 773–794
DOI: 10.20537/2076-7633-2020-12-4-773-794

Ки&М

ЧИСЛЕННЫЕ МЕТОДЫ И ОСНОВЫ ИХ РЕАЛИЗАЦИИ

УДК: 519.6

Оценка масштабируемости программы расчета движения примесей в атмосфере средствами симулятора gem5

А. Н. Семакин

Московский государственный технический университет им. Н. Э. Баумана,
Россия, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1

E-mail: arte-semaki@yandex.ru

Получено 11.05.2019, после доработки — 02.04.2020.
Принято к публикации 13.04.2020.

В данной работе мы предлагаем новую эффективную программную реализацию алгоритма расчета трансконтинентального переноса примеси в атмосфере от естественного или антропогенного источника на адаптивной конечно-разностной сетке, концентрирующей свои узлы внутри переносимого облака примеси, где наблюдаются резкие изменения значений ее массовой доли, и максимально разрежающей узлы во всех остальных частях атмосферы, что позволяет минимизировать общее количество узлов. Особенностью реализации является представление адаптивной сетки в виде комбинации динамических (дерево, связный список) и статических (массив) структур данных. Такое представление сетки позволяет увеличить скорость выполнения расчетов в два раза по сравнению со стандартным подходом представления адаптивной сетки только через динамические структуры данных.

Программа создавалась на компьютере с шестиядерным процессором. С помощью симулятора gem5, позволяющего моделировать работу различных компьютерных систем, была произведена оценка масштабируемости программы при переходе на большее число ядер (вплоть до 32) на нескольких моделях компьютерной системы вида «вычислительные ядра – кэш-память – оперативная память» с разной степенью детализации ее элементов. Отмечено существенное влияние состава компьютерной системы на степень масштабируемости исполняемой на ней программы: максимальное ускорение на 32-х ядрах при переходе от двухуровневого кэша к трехуровневому увеличивается с 14.2 до 22.2. Время выполнения программы на модели компьютера в gem5 превосходит время ее выполнения на реальном компьютере в 10^4–10^5 раз в зависимости от состава модели и составляет 1.5 часа для наиболее детализированной и сложной модели.

Также в статье рассматриваются подробный порядок настройки симулятора gem5 и наиболее оптимальный с точки зрения временных затрат способ проведения симуляций, когда выполнение не представляющих интерес участков кода переносится на физический процессор компьютера, где работает gem5, а непосредственно внутри симулятора выполняется лишь исследуемый целевой кусок кода.

Ключевые слова: gem5, масштабируемость программ, трехмерный перенос примесей в атмосфере

COMPUTER RESEARCH AND MODELING
2020 VOL. 12 NO. 4 P. 773–794
DOI: 10.20537/2076-7633-2020-12-4-773-794

Ки&М

NUMERICAL METHODS AND THE BASIS FOR THEIR APPLICATION

UDC: 519.6

Evaluation of the scalability property of the program for the simulation of atmospheric chemical transport by means of the simulator gem5

A. N. Semakin

Bauman Moscow State Technical University,
5/1 Baumanskaya 2-a st., Moscow, 105005, Russia

E-mail: arte-semaki@yandex.ru

Received 11.05.2019, after completion — 02.04.2020.
Accepted for publication 13.04.2020.

In this work we have developed a new efficient program for the numerical simulation of 3D global chemical transport on an adaptive finite-difference grid which allows us to concentrate grid points in the regions where flow variables sharply change and coarsen the grid in the regions of their smooth behavior, which significantly minimizes the grid size. We represent the adaptive grid with a combination of several dynamic (tree, linked list) and static (array) data structures. The dynamic data structures are used for a grid reconstruction, and the calculations of the flow variables are based on the static data structures. The introduction of the static data structures allows us to speed up the program by a factor of 2 in comparison with the conventional approach to the grid representation with only dynamic data structures.

We wrote and tested our program on a computer with 6 CPU cores. Using the computer microarchitecture simulator gem5, we estimated the scalability property of the program on a significantly greater number of cores (up to 32), using several models of a computer system with the design "computational cores – cache – main memory". It has been shown that the microarchitecture of a computer system has a significant impact on the scalability property, i.e. the same program demonstrates different efficiency on different computer microarchitectures. For example, we have a speedup of 14.2 on a processor with 32 cores and 2 cache levels, but we have a speedup of 22.2 on a processor with 32 cores and 3 cache levels. The execution time of a program on a computer model in gem5 is 10^4–10^5 times greater than the execution time of the same program on a real computer and equals 1.5 hours for the most complex model.

Also in this work we describe how to configure gem5 and how to perform simulations with gem5 in the most optimal way.

Keywords: gem5, scalability property of programs, 3D atmospheric chemical transport

Citation: *Computer Research and Modeling*, 2020, vol. 12, no. 4, pp. 773–794 (Russian).

Введение

Во второй половине 20 века рост производительности процессоров в значительной степени обеспечивался за счет быстрого увеличения их тактовой частоты. Однако уже к середине первого десятилетия 21 века возможности по наращиванию тактовой частоты были исчерпаны из-за физических ограничений, обусловленных среди прочего проявлением эффекта теплового шума [Fuller, Millett, 2011; Kish, 2002]. Начиная с этого момента производители пошли по пути стремительного увеличения количества параллельно работающих вычислительных ядер в рамках одного процессора. Например, последние модели процессоров семейства Xeon Phi x200 компании Intel уже содержат 72 ядра.

Разработка программ, показывающих линейный прирост производительности с увеличением количества используемых вычислительных ядер, требует ювелирной балансировки нагрузки и довольно продолжительного времени отладки (десятки и сотни рабочих часов). Однако современные рабочие станции, построенные на базе многоядерных процессоров, стоят очень дорого (стоимость одного только процессора может достигать $10000). Поэтому их использование на этапе написания и отладки программ нерационально. Программы можно писать на дешевых маломощных компьютерах, а дорогие рабочие станции использовать только на этапе массовых вычислений. Данная стратегия позволяет существенно повысить отдачу от использования вычислительной техники в рамках отдела или организации в целом, особенно когда идет работа одновременно с несколькими программными кодами.

Во время написания и отладки программы масштабируемость кода на различное число ядер можно оценить с помощью симуляторов микроархитектуры, представляющих собой программное обеспечение, имитирующее работу компьютера с заданными характеристиками. Симуляторы компьютерных систем делятся на два класса: функциональные (functionally-accurate) и тактовые (cycle-accurate). Первые (QEMU [Bellard, 2005], OVPsim [www.ovpworld.org], Simics [Aarno, Engblom, 2015]) воспроизводят функциональность элементов микроархитектуры компьютера путем имитации последовательного изменения их текущего состояния согласно заданному потоку команд программы пользователя, но не учитывают время, которое требуется на выполнение этих команд на реальном физическом воплощении данной микроархитектуры. Поэтому функциональные симуляторы можно использовать для проверки работоспособности программ на целевом оборудовании, но оценить эффективность программ с точки зрения затрат времени на их выполнение и полноты использования всех задействованных вычислительных мощностей невозможно. Для этого необходимо использовать тактовые симуляторы (PTLsim [Yourst, 2007], gem5 [Binkert et al., 2011], MARSS [Patel et al., 2011], SimFlex [Hardavellas et al., 2004], SimpleScalar [Austin et al., 2002]), которые симулируют выполнение очередной команды по тактам, что позволяет рассчитывать время, необходимое на ее выполнение на моделируемой микроархитектуре.

Тактовый симулятор gem5 был создан путем объединения наиболее разработанных частей симуляторов M5 и GEMS: M5 предоставил рабочие модели процессоров, а вклад GEMS состоял в хорошо детализированной системе памяти. Симулятор gem5 непрерывно развивается, постоянно оптимизируются уже существующие модели аппаратного обеспечения компьютерной системы, а также своевременно добавляются модели вновь выпускаемых модификаций компьютерных комплектующих [Hansson et al., 2014; Jagtap et al., 2017]. Например, на данный момент gem5 содержит 14 типов памяти, включая DDR3, DDR4 и GDDR5. Разработкой gem5 занимается коллектив, в который входят как представители академического сообщества (MIT, Princeton), так и специалисты коммерческого сектора (AMD, HP).

Моделирование микроархитектуры компьютера на тактовом симуляторе занимает на несколько порядков больше времени, чем непосредственная работа самого моделируемого компьютера. Симуляция 1 секунды реальной работы программы может занимать несколько часов,

что сильно ограничивает практическую применимость тактовых симуляторов при тестировании программ. Решением проблемы в gem5 стало введение возможности аппаратного ускорения симуляции с помощью программного обеспечения KVM, что существенно сокращает время симуляции [Sandberg et al., 2015]. Поскольку в этом режиме большая часть команд программы, запущенной внутри симулятора, выполняется напрямую на физическом процессоре, а сам симулятор использует реальное, а не моделируемое время, то при включении аппаратного ускорения gem5 работает как функциональный симулятор. Отсюда вытекает наиболее оптимальная на данный момент стратегия применения gem5: симуляция в целом идет в режиме аппаратного ускорения, но на время выполнения интересующего нас участка кода gem5 переходит в тактовый режим и определяет нужные временные характеристики.

Для наиболее распространенных архитектур x86 и ARM был выполнен ряд работ по оценке достоверности результатов симулятора gem5 в сравнении с физическим оборудованием [Butko et al., 2012; Gutierrez et al., 2014; Akram, Sawalha, 2016]. На реальном компьютере и его модели в gem5 запускались программы из нескольких специализированных тестовых наборов (SPLASH-2, SPEC CPU2006 и т. д.) и сравнивались полученные значения метрик. Величина ошибки абсолютных метрик (скорость передачи данных между памятью и процессором, кэш-промахи, кэш-попадания и т. д.) варьировалась в широких пределах (от 0 % до 100 % и выше) в зависимости от выполнявшейся тестовой программы. Это объяснялось тем, что представленные в gem5 модели процессора не учитывают все архитектурные нюансы каждого отдельно взятого физического процессора. Однако относительная метрика «масштабируемость», показывающая ускорение работы программы при переходе на несколько вычислительных ядер, дала приемлемые результаты. Ее ошибка не превосходила 5 % во всех тестах, в которых она измерялась. Следовательно, gem5 вполне можно применять для оценки масштабируемости разрабатываемых программ.

Задачу расчета переноса примесей в атмосфере коротко можно сформулировать следующим образом: в произвольный момент времени из некоторого точечного естественного или антропогенного источника в атмосферу попало значимое количество загрязняющих примесей, и необходимо определить, где территориально окажется получившееся облако примесей, какую структуру и состав оно будет иметь через заданный промежуток времени. Такие задачи возникают в результате различных техногенных катастроф. В качестве примера можно привести аварию на японской АЭС «Фукусима-1», когда в атмосферу попали радиоактивные вещества, следы которых спустя несколько дней были зафиксированы на территории США [Thakur et al., 2012].

Особенностью задач этого класса является необходимость при численном решении подробно разрешать с помощью разностной сетки лишь небольшую часть пространства, занятого облаком примеси, в пределах которого наблюдаются значительные и резкие пространственные изменения значений массовой доли примеси и всех зависящих от нее величин. С течением времени облако примеси движется и меняет свою структуру. Это означает, что в разные моменты времени подробная сетка требуется в разных частях пространства. Использование равномерной сетки, когда одинаково подробно разрешается как облако примеси, так и пространство за его пределами, требует привлечения значительного количества вычислительных ресурсов. Например, можно упомянуть расчеты на равномерной сетке с шагом в несколько километров на суперкомпьютере Mira в Аргоннской национальной лаборатории США с привлечением более 700 тысяч вычислительных ядер [Muller et al., 2015].

Адаптивная сетка строится с переменным шагом, что позволяет ей концентрировать свои узлы в той части пространства, которая требует подробного разрешения, и уменьшать количество узлов в остальном пространстве. К настоящему времени адаптивные сетки уже широко используются при решении задач вычислительной гидромеханики в целом [Berger, Colella, 1989; Quirk, 1996; Baeza et al., 2012; Pau et al., 2012], но в области численного моделирования физиче-

ских процессов в атмосфере все еще встречаются достаточно редко [Constantinescu et al., 2008; Kopera, Giraldo, 2014].

Различные виды адаптивных сеток отличаются друг от друга по критерию адаптации, который определяет переменную величину шага сетки. В своей работе в качестве критерия мы используем адаптацию по вейвлетам [Schneider, Vasilyev, 2010; Paolucci et al., 2014], которая базируется на хорошо разработанной математической теории вейвлетов. К преимуществам данного критерия адаптации относятся: 1) высокий порядок точности численных расчетов, что позволяет сочетать этот критерий с конечно-разностными схемами высокого порядка (3-й и выше); 2) быстрый, рациональный и одновременно простой алгоритм перестройки сетки, что позволяет проводить адаптацию на каждом шаге по времени; 3) возможность работать с геометрически сложными областями и криволинейными системами координат, что особенно важно при расчете движения воздушных потоков в нижних слоях тропосферы. Наши численные эксперименты показали, что благодаря указанным особенностям адаптация по вейвлетам позволяет строить сетку в среднем на пять порядков меньшего размера, чем соответствующая равномерная сетка при той же самой точности расчетов.

Целью данной работы является создание программного комплекса расчета движения примесей в атмосфере на адаптивной сетке с адаптацией по вейвлетам, который может использоваться с максимальной эффективностью на современных многоядерных рабочих станциях. Для достижения этой цели были решены две задачи: 1) создано представление адаптивной сетки в виде комбинации двух динамических (дерево и связный список) и одной статической (массив) структур данных, что позволило уменьшить время выполнения численного алгоритма в два раза и повысить эффективность распараллеливания на 10 % по сравнению с традиционным для адаптивных сеток представлением в виде исключительно динамических структур данных; 2) произведена оценка эффективности распараллеливания кода программы на несколько десятков вычислительных ядер с помощью симулятора gem5 в условиях отсутствия прямого доступа к соответствующим рабочим станциям.

Для оценки масштабируемости программы на произвольное число вычислительных ядер в gem5 проведена симуляция трех компьютерных систем различной степени детализации памяти и процессора. Показано, что способность к масштабированию программы зависит не только от качества кода, но и от микроархитектуры компьютера, на котором эта программа выполняется. В частности, переход от двухуровневого кэша к трехуровневому увеличивает эффективность распараллеливания в одном и том же коде с 44 % до 69 % на 32 вычислительных ядрах.

На компьютере, где запускается gem5, а также во всех симулируемых компьютерных системах использовалась операционная система CentOS 7. Соответственно, порядок подготовки симулятора gem5 к работе, приведенный в данной статье, справедлив именно для ОС CentOS 7 (или ее платного аналога Red Hat Enterprise Linux 7).

Структура статьи следующая. В § 2 приведена математическая модель переноса примеси в атмосфере. § 3 содержит описание алгоритма численного решения уравнения переноса на адаптивной сетке. Программная реализация численного алгоритма представлена в § 4. Порядок подготовки симулятора gem5 к работе и модели компьютерных систем рассмотрены в § 5. § 6 содержит оценку масштабируемости программы расчета переноса примесей в атмосфере на процессорных ядрах в диапазоне 1–32.

Математическая модель

Природные процессы, определяющие пространственно-временное распределение различных примесей в атмосфере, можно разделить на четыре группы: адвективный и турбулентный перенос, химические реакции, эмиссии, осаждения. Протекание данных процессов зависит

от значений метеорологических переменных, таких как скорость ветра, температура, влажность, интенсивность турбулентности и т. д.

Поля метеорологических переменных рассчитываются с помощью метеорологических моделей двух видов — диагностических и динамических [Seaman, 2000]. В основе диагностических моделей лежит объективный анализ доступных данных наблюдений, результатом которого является набор внутренне непротиворечивых метеорологических полей. Динамические модели строятся на базе дифференциальных уравнений законов сохранения массы, импульса и энергии, а метеорологические поля получаются путем их численного интегрирования тем или иным методом. Данные наблюдений используются как для задания начальных условий, так и для коррекции численных ошибок по ходу решения.

В зависимости от размера области покрытия среди динамических моделей выделяют модели общей циркуляции (General Circulation Models, GCMs) и региональные климатические модели (Regional Climate Models, RCMs). Модели общей циркуляции моделируют динамику планетарной атмосферы в целом, а региональные климатические модели используются для расчета метеорологических полей лишь в определенных регионах. Примером модели общей циркуляции служит Community Earth System Model версии 2 (CESM2) [Danabasoglu et al., 2020], которая позволяет одновременно моделировать физические процессы и явления в атмосфере, океане и на земле с учетом их взаимного влияния друг на друга. К региональным климатическим моделям относится Weather Research and Forecasting Model (WRF) [Powers et al., 2017], используемая среди прочего для расчета метеорологических условий над континентальной частью США и Антарктикой. Образцы метеорологических полей, собираемых в настоящее время в рамках проекта CMIP6 [Eyring et al., 2016] от различных динамических моделей, можно найти на веб-сайте https://esgf-node.llnl.gov/search/cmip6/.

Непосредственное распространение примесей в атмосфере описывается химическими транспортными моделями (Chemical Transport Models, CTMs). Эти модели состоят из дифференциальных уравнений в частных производных, определяющих значимые для рассматриваемых примесей физические и химические процессы [Hundsdorfer, Verwer, 2010; Sportisse, 2010]:

$$\frac{\partial c_i}{\partial t} + \mathbf{u} \cdot \nabla c_i = \frac{1}{\rho}\mathrm{div}(\rho D \nabla c_i) + \omega_i/\rho, \quad i = 1, \ldots, N, \tag{1}$$

где c_i — массовая доля i-й примеси в воздухе ($c_i = \rho_i/\rho$, ρ_i — плотность i-й примеси, ρ — плотность воздуха), N — общее количество рассматриваемых примесей, $\mathbf{u}$ — скорость ветра, D — тензор коэффициентов турбулентной диффузии второго ранга, ω_i — скорость изменения массовой доли примеси в результате химических реакций в атмосфере, естественных и антропогенных эмиссий и осаждения.

Поскольку коэффициенты молекулярной диффузии по величине на несколько порядков меньше коэффициентов турбулентной диффузии, действие молекулярной диффузии обычно исключается из рассмотрения. В свою очередь, турбулентная диффузия обусловлена переносом вещества хаотическим вихревым движением воздуха, т. е. является характеристикой воздушного потока, а не конкретной примеси. Поэтому тензор D изменяется во времени и пространстве, но одинаков для всех примесей.

Входными данными химических транспортных моделей выступают метеорологические поля (скорость ветра $\mathbf{u}$, плотность воздуха ρ, коэффициенты турбулентной диффузии D и переменные, определяющие величину ω_i в уравнении (1)), распределенные и точечные источники поступления примесей в атмосферу (например, распределенным источником может быть крупный лесной пожар, а точечным — какой-нибудь завод) и начальное распределение примесей $c_i(\mathbf{x}, t = 0)$, $i = 1, \ldots, N$ в атмосфере. На выходе получаются функции $c_i(\mathbf{x}, t)$, $i = 1, \ldots, N$, описывающие поведение примесей в рассматриваемой области на заданном отрезке времени $[0, T]$.

Примерами химических транспортных моделей выступают GEOS-Chem [Bey et al., 2001] и TM5 [Huijnen et al., 2010].

В настоящее время наиболее распространенный подход к изучению процесса переноса примесей в атмосфере заключается в следующем: метеорологическая модель производит набор метеорологических полей, на основании которых химическая транспортная модель делает серию симуляций, изменяя начальные распределения примесей, их состав, порядок взаимодействия и источники эмиссии в атмосферу. Необходимость разделения на две модели обусловлена существенной разницей в сложности самих моделей, а также разницей в требуемых вычислительных ресурсах, затрачиваемых на проведение расчетов. Поэтому существенно более затратное по времени и ресурсам изготовление метеорологических полей производится только один раз, а не во время каждой отдельной симуляции распространения примесей, что существенно сокращает время исследований. К тому же одна и та же химическая транспортная модель может использовать метеорологические поля, произведенные разными метеорологическими моделями. Это позволяет оперативно заменить имеющиеся метеорологические поля, если, например, качество работы текущей метеорологической модели стало неудовлетворительным или появилась новая более совершенная модель. Чтобы учесть влияние находящихся в атмосфере примесей на метеорологию, обе модели иногда объединяют в одну (например, WRF-Chem [Grell et al., 2005]).

Уравнение (1) записано в инвариантной относительно систем координат форме. Это значит, что по обе стороны от знака равенства находятся функции точек пространства, а не функции их координат в какой-либо фиксированной системе. Однако количественное описание физических процессов, определяемых уравнением (1), требует введения определенной системы координат, позволяющей ассоциировать каждую точку пространства с набором из трех действительных чисел. Если полярные области не представляют интереса, то чаще всего в качестве горизонтальных координат используются широта θ и долгота φ. Выбор вертикальной координаты в значительной степени зависит от конкретной задачи. Это может быть расстояние от центра Земли r или высота над земной поверхностью h, давление p ($p_t \le p \le p_s$) или производная от него $\sigma = (p - p_t)/(p_s - p_t)$ ($0 \le \sigma \le 1$), где p_s — давление на земной поверхности под данной точкой пространства, p_t — давление на верхней границе рассматриваемой части атмосферы [Kasahara, 1974]. Также существуют и другие варианты выбора вертикальной координаты.

В своей работе мы в основном рассматриваем перенос примесей с восточного побережья Азии на западное побережье Северной Америки, который осуществляется над северной частью Тихого океана. Поскольку поверхность океана с высокой степенью точности можно рассматривать как часть сферической поверхности, мы используем обычную сферическую систему координат.

В сферической системе координат уравнение (1) для движения облака одной инертной примеси, в отсутствие диффузии, эмиссии и осаждения, переходит в линейное уравнение переноса:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial r} + \frac{v}{r}\frac{\partial c}{\partial \theta} + \frac{w}{r\sin\theta}\frac{\partial c}{\partial \varphi} = 0, \tag{2}$$

где r — расстояние от центра Земли до данной точки, $0 \le \theta \le \pi$ — угловое расстояние вдоль меридиана от южного полюса, $0 \le \varphi \le 2\pi$ — угловое расстояние вдоль параллели от нулевого меридиана на восток, u — радиальная компонента скорости, v и w — проекции вектора скорости на касательные к меридиану и параллели в данной точке. Это уравнение требует задания начального распределения массовой доли примеси $c(r, \theta, \varphi, 0)$ и поля скорости $\mathbf{u} = (u, v, w)$ на всем рассматриваемом отрезке времени $[0, T]$. В расчетах мы используем те же поля скоростей, что и модель GEOS-Chem. Архивные файлы с данными полями доступны для свободного скачивания с официального сайта модели: http://acmg.seas.harvard.edu/geos/. Поскольку GEOS-Chem

использует гибридную $\sigma - p$ вертикальную координату, перед непосредственным использованием значения компонент скорости переводятся в сферическую систему координат.

Линейное уравнение переноса (2) сохраняет величину массовой доли c в каждой отдельно взятой материальной точке вдоль траектории ее движения $(r(t), \theta(t), \varphi(t))$:

$$c(r(t), \theta(t), \varphi(t), t) = c(r(0), \theta(0), \varphi(0), 0). \tag{3}$$

Это математическое свойство отражает результаты многочисленных наблюдений за распространением различных примесей в верхней тропосфере, где составленные из них облака перемещаются на тысячи километров без каких-либо существенных искажений своей пространственной структуры [Newell et al., 1999]. Воспроизведение данного свойства при численном решении требует привлечения разностной сетки с очень мелким шагом (~ 1 километра) в месте нахождения моделируемого облака, что негативно (в сторону сильного увеличения) сказывается на размере сетки.

Численный метод

Численный метод решения уравнения (2) состоит в циклическом выполнении двух последовательных шагов [Semakin, Rastigejev, 2016]:

1) построение адаптивной разностной сетки по известным на текущий момент времени значениям функции c;
2) формирование на построенной сетке разностной схемы и вычисление значений функции c в следующий момент времени.

Сначала рассмотрим шаг 1. Пусть нам известны значения функции c в момент времени t^n, и для простоты будем считать, что массовая доля моделируемой примеси изменяется только в вертикальном направлении — $c = c(r, t)$. Типичный вид такой функции c представлен на рис. 1 (сплошная линия). Подобное распределение образуется в результате выброса в атмосферу загрязняющих веществ на промышленных предприятиях через высокие трубы. Попав в атмосферу, загрязняющие вещества собираются в горизонтальном слое конечной толщины (пик функции c на рис. 1, *а*), и далее под воздействием восходящих потоков воздуха данный слой может подниматься в верхнюю часть тропосферы (см. рис. 1, *б*).

Обозначим: $c^n(r) = c(r, t^n)$. Функцию $c^n(r)$ раскладываем в ряд по вейвлетам [Daubechies, 1992]:

$$c^n(r) = \sum_i b_i \phi_i(r) + \sum_{l=1}^{\infty} \sum_i d_{l,i} \psi_{l,i}(r). \tag{4}$$

Здесь $\phi_i(r) = \phi(r - i)$, $i \in Z$, — копии одной и той же масштабирующей функции ϕ, смещенные друг относительно друга в пространстве. Аналогично: $\psi_{l,i} = \psi(2^{l-1} r - i)$, $l \geq 1$, $i \in Z$, — это копии вейвлета ψ, отвечающие l-му уровню разрешения (детализации) и расположенные в месте, определенном индексом i, причем сам вейвлет ψ связан с масштабирующей функцией по формуле $\psi(r) = \phi(2r - 1)$.

Существует множество различных семейств вейвлетов. В этой работе мы используем семейство Deslauriers – Dubuc. Данный выбор обусловлен простотой их построения с помощью лагранжевой интерполяции. Подробную инструкцию по построению данных вейвлетов, а также схему вычисления коэффициентов b_i и $d_{l,i}$ в ряде (4) можно найти в [Klees, Haagmans, 2000, p. 72–107].

На рис. 2 приведены графики нескольких функций $\phi_i(r)$ и $\psi_{l,i}(r)$, а также их взаимное расположение вдоль оси r. По построению функции локализованы в пространстве ($\operatorname{supp} \phi$ —

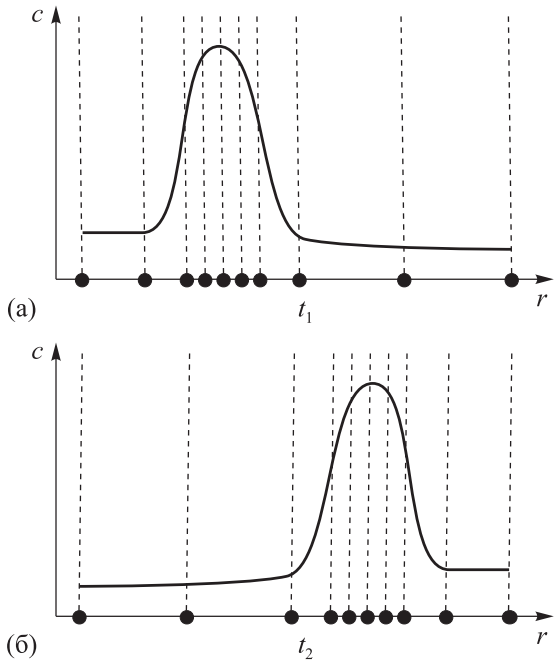

Рис. 1. Пример возможного распределения массовой доли примеси c в радиальном направлении от поверхности Земли над антропогенным источником загрязнения (например, заводом) и соответствующая адаптивная сетка в два последовательных момента времени t_1 и t_2, $t_1 < t_2$, получающиеся в результате действия восходящих потоков воздуха

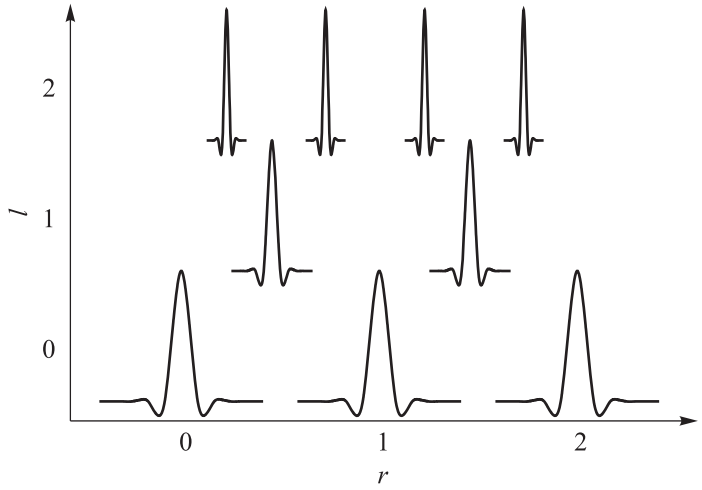

Рис. 2. Масштабирующие функции ϕ_i ($l = 0$) и вейвлеты $\psi_{l,i}$ ($l = 1, 2$). Положение каждой функции вдоль оси r определяется ее индексами l и i. На оси r отмечены значения индекса i только для функций ϕ_i

конечное число), причем при переходе на каждый последующий уровень разрешения степень локализации увеличивается. Данное свойство указывает на роль слагаемых в ряде (4). Масштабирующие функции используются для представления основных крупномасштабных особенностей поведения функции $c^n(r)$, а вейвлеты описывают поведение функции на мелких масштабах. Чем выше уровень разрешения l, тем более мелкие детали вейвлеты захватывают.

Поскольку вклад вейвлета $\psi_{l,i}(r)$ в ряд (4) определяется величиной его коэффициента $d_{l,i}$, то можно сжать представление функции путем исключения из (4) вейвлетов, чьи коэффициенты меньше некоторого порога ε [Donoho, 1992]. Значение порога ε выбирается достаточно малым,

чтобы удалить из исходного ряда только те вейвлеты, которые отражают лишь несущественную информацию о структуре функции. В результате получаем разреженный ряд вейвлетов:

$$c_\varepsilon^n(r) = \sum_i b_i \phi_i(r) + \sum_{l=1}^{J} \sum_{|d|>\varepsilon} d_{l,i} \psi_{l,i}(r), \tag{5}$$

где $c_\varepsilon^n(r)$ — приближение исходной функции $c^n(r)$, точность которого зависит от величины ε; J — конечный максимальный уровень разрешения. Обычно ряд (5) содержит лишь небольшое число вейвлетов, которые пространственно концентрируются в областях резкого изменения значений $c^n(r)$.

Локализация вейвлетов в пространстве позволяет ассоциировать с каждым вейвлетом некоторую точку. Самый очевидный и оптимальный выбор — это точка, через которую проходит ось симметрии вейвлета. Тогда ряд (5) задает некоторое множество точек на оси r. Учитывая пространственное положение вейвлетов в (5), мы получаем адаптивную сетку, которая концентрирует точки в областях резкого изменения значений исходной функции и поддерживает большой шаг в областях, где функция изменяется плавно. Типичный пример адаптивной сетки, построенной по ряду (5), изображен на рис. 1.

Разностная сетка, построенная по ряду (5), адаптирована к массовой доли примеси $c^n(r)$ на фиксированный момент времени t^n. К следующему моменту времени t^{n+1} из-за адвективного переноса значения массовой доли немного изменятся, что наглядно показано на рис. 1, и, соответственно, текущая адаптивная сетка уже не сможет полностью отобразить новое распределение массовой доли $c^{n+1}(r)$. Поэтому мы расширяем ряд (5) путем добавления к каждому входящему в него вейвлету несколько дополнительных вейвлетов, расположенных рядом как на том же уровне разрешения, так и на следующем, если эти вейвлеты отсутствуют в ряде (5). Например, если вейвлет $\psi_{l,i}(r)$ входит в (5), то мы также включаем в этот ряд вейвлеты $\psi_{p,m}(r)$, где $|m - i| \le s$, $p = \overline{l, l+1}$, а количество включаемых с каждой стороны соседних вейвлетов s зависит от величины изменений массовой доли при переходе от t^n к t^{n+1}, т. е. фактически от выбираемого шага по времени. В нашей работе мы полагаем $s = 2$. Адаптивная сетка, построенная на базе получившегося расширенного разреженного ряда вейвлетов, будет несколько избыточной для $c^n(r)$, но она будет захватывать возможные изменения в массовой доле при переходе к $c^{n+1}(r)$.

В более сложных случаях, когда перенос примеси осуществляется не только по высоте атмосферы, но и вдоль поверхности Земли, для представления массовой доли примеси необходимо привлекать многомерные масштабирующие функции и вейвлеты, определяемые через произведения соответствующих одномерных масштабирующих функций и вейвлетов [Vasilyev, 2003].

1. Двухмерный случай.

Пусть перенос примеси идет вдоль земной поверхности на фиксированной высоте. Тогда ее массовая доля есть функция широты θ и долготы φ — $c = c(\theta, \varphi, t)$, а двухмерные масштабирующие функции и вейвлеты задаются формулами

$$\begin{aligned} \phi_{i,j}(\theta,\varphi) &= \phi_i(\theta) \cdot \phi_j(\varphi), \\ \psi^1_{l,i,j}(\theta,\varphi) &= \psi_{l,i-1/2}(\theta) \cdot \psi_{l,j}(\varphi), \\ \psi^2_{l,i,j}(\theta,\varphi) &= \psi_{l,i}(\theta) \cdot \psi_{l,j-1/2}(\varphi), \\ \psi^3_{l,i,j}(\theta,\varphi) &= \psi_{l,i}(\theta) \cdot \psi_{l,j}(\varphi). \end{aligned} \tag{6}$$

Взаимное расположение масштабирующих функций $\phi_{i,j}$ и вейвлетов $\psi^m_{l,i,j}$, $m = 1, 2, 3$, первого уровня разрешения ($l = 1$) показано на рис. 3, где для каждой функции отмечена ее центральная точка. Носители двухмерных масштабирующих функций $\operatorname{supp} \phi_{i,j}$ и вейвлетов $\operatorname{supp} \psi^m_{l,i,j}$

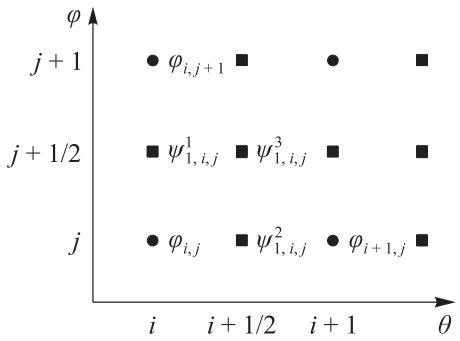

Рис. 3. Расположение двухмерных масштабирующих функций $\phi_{i,j}$ и вейвлетов $\psi^1_{1,i,j}$, $\psi^2_{1,i,j}$, $\psi^2_{1,i,j}$ первого уровня разрешения ($l = 1$) на плоскости $\theta{-}\phi$. Положение функций задается через указание их центральных точек, в которых они достигают своего максимального значения, равного единице. Центральные точки масштабирующих функций обозначаются кружками, а вейвлетов — квадратами. По мере удаления от центральных точек масштабирующие функции и вейвлеты быстро устремляются к нулю

представляют собой квадраты, длина стороны которых уменьшается в два раза при каждом переходе на следующий уровень разрешения. Центральная точка определяется как центр этого квадрата, в котором соответствующая функция достигает своего максимального значения, равного единице.

Разложение функции $c^n(\theta, \varphi) = c(\theta, \varphi, t^n)$ в момент времени t^n в ряд по двухмерным вейвлетам имеет вид

$$c^n(\theta,\varphi) = \sum_{i,j} b_{i,j}\phi_{i,j}(\theta,\varphi) + \sum_{l=1}^{\infty}\sum_{i,j}\sum_{m=1}^{3} d^m_{l,i,j}\psi^m_{l,i,j}(\theta,\varphi). \tag{7}$$

2. Трехмерный случай.

Трехмерные масштабирующие функции и вейвлеты задаются формулами

$$\begin{aligned}
\phi_{i,j,k}(r,\theta,\varphi) &= \phi_i(r)\cdot\phi_j(\theta)\cdot\phi_k(\varphi), & \psi^1_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i}(r)\cdot\psi_{l,j-1/2}(\theta)\cdot\psi_{l,k-1/2}(\varphi),\\
\psi^2_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i-1/2}(r)\cdot\psi_{l,j}(\theta)\cdot\psi_{l,k-1/2}(\varphi), & \psi^3_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i}(r)\cdot\psi_{l,j}(\theta)\cdot\psi_{l,k-1/2}(\varphi),\\
\psi^4_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i-1/2}(r)\cdot\psi_{l,j-1/2}(\theta)\cdot\psi_{l,k}(\varphi), & \psi^5_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i}(r)\cdot\psi_{l,j-1/2}(\theta)\cdot\psi_{l,k}(\varphi),\\
\psi^6_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i-1/2}(r)\cdot\psi_{l,j}(\theta)\cdot\psi_{l,k}(\varphi), & \psi^7_{l,i,j,k}(r,\theta,\varphi) &= \psi_{l,i}(r)\cdot\psi_{l,j}(\theta)\cdot\psi_{l,k}(\varphi),
\end{aligned} \tag{8}$$

а соответствующее разложение функции c имеет вид

$$c^n(r,\theta,\varphi) = \sum_{i,j,k} b_{i,j,k}\phi_{i,j,k}(r,\theta,\varphi) + \sum_{l=1}^{\infty}\sum_{i,j,k}\sum_{m=1}^{7} d^m_{l,i,j,k}\psi^m_{l,i,j,k}(r,\theta,\varphi). \tag{9}$$

Носителями трехмерных масштабирующих функций и вейвлетов служат кубы, центры которых определяются как центральные точки соответствующих масштабирующих функций и вейвлетов.

Коэффициенты двух- и трехмерных вейвлетов $d^m_{l,i,j}$, $d^m_{l,i,j,k}$ вычисляются аналогично одномерному случаю, с поправкой на большее число измерений, и их величина определяет размер вклада соответствующего вейвлета в ряды (7) и (9). Удалив из рядов (7) и (9) те вейвлеты, чьи коэффициенты меньше заданного порога ε, получим разреженные ряды, которые содержат лишь небольшое число членов, пространственно концентрирующихся в области резкого изменения функции c. Каждому вейвлету в разреженном ряде ставится в соответствие его центральная точка. Тогда сам разреженный ряд на плоскости или в пространстве будет задавать множество точек,

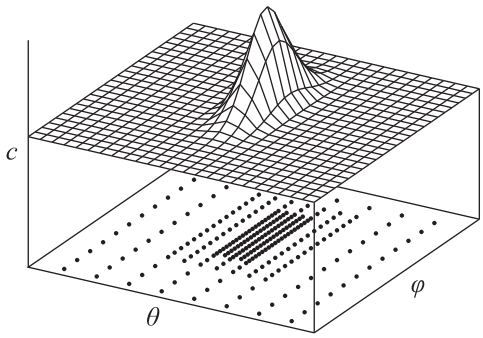

Рис. 4. Пример возможного двухмерного распределения массовой доли примеси $c(\theta, \varphi, t^n)$ в некотором горизонтальном слое атмосферы в момент времени t^n и соответствующая адаптивная сетка. Примесь занимает лишь небольшую область горизонтального слоя, в которой ее массовая доля $c(\theta, \varphi, t^n)$ быстро уменьшается от своего максимального значения в центре до нуля на границе. Соответственно, точки адаптивной сетки размещаются в области, занятой примесью, с малым шагом, а за ее пределами с крупным шагом, причем переход от крупного шага к малому идет постепенно. В частности, на рисунке между участками с крупным и малым шагами располагается участок с промежуточным шагом

формирующих сетку с малым шагом в области с большими по модулю градиентами функции c и крупным шагом там, где модуль градиента функции c достаточно мал. Чтобы учесть изменения в пространственной структуре массовой доли при переходе от t^n к t^{n+1}, на базе разреженного ряда вейвлетов строим расширенный разреженный ряд, из которого получаем итоговую адаптивную сетку, учитывающую как текущее поведение массовой доли, так и ее возможное изменение к следующему моменту времени. Пример двухмерной адаптивной сетки приведен на рис. 4.

Теперь перейдем к шагу 2. При интегрировании уравнения (2) по времени используется явно-неявная схема второго порядка точности:

$$\frac{c^{n+1} - c^n}{\Delta t} + \frac{1}{2}\Lambda(c^{n+1} + c^n) = 0, \tag{10}$$

где Λ — конечно-разностный аналог дифференциального оператора адвекции, получающийся путем замены пространственных производных в (2) несимметрическими конечными разностями [Durran, 2010]. В работе мы использовали разностные производные 3, 5 и 7 порядков аппроксимации. Например, несимметрическая разностная производная 3 порядка аппроксимации по переменной r в точке с индексом i имеет вид

$$\left(\frac{\partial c}{\partial r}\right)_i \approx \begin{cases} \dfrac{c_{i-2} - 6c_{i-1} + 3c_i + 2c_{i+1}}{6h_i}, & u_i \geq 0, \\ \dfrac{-2c_{i-1} - 3c_i + 6c_{i+1} - c_{i+2}}{6h_i}, & u_i < 0. \end{cases} \tag{11}$$

Шаг h_i определяется как расстояние от точки с индексом i до ближайшей к ней точки адаптивной сетки в направлении изменения переменной r. По этому шагу восстанавливается равномерный шаблон разностной производной (11) в зависимости от знака скорости u. Если какая-то точка шаблона отсутствует в адаптивной сетке, то значение функции c в этой точке восстанавливается с помощью лагранжевой интерполяции.

Конечно-разностную схему (10) можно переписать в виде системы линейных алгебраических уравнений

$$Ac^{n+1} = b, \tag{12}$$

решая ее, мы находим массовую долю примеси c^{n+1} в следующий момент времени t^{n+1}. Численные эксперименты показали, что для решения системы (12) достаточно простейшего метода Якоби.

Реализация алгоритма

В наших предыдущих работах [Семакин, 2017] программная реализация численного алгоритма базировалась исключительно на динамических структурах данных. Адаптивная сетка представлялась одновременно в виде двух структур — дерево и связный список, — составленных из одних и тех же элементов. Дерево использовалось для быстрого доступа и индивидуальной работы с отдельными узлами сетки, а связный список предназначался для выполнения единообразных вычислительных операций сразу на всех узлах сетки. Однако вычислительные эксперименты показали, что с точки зрения времени выполнения программы более эффективным подходом является использование трех структур данных — двух динамических (дерево и связный список) и одной статической (массив). Схема работы такой программы показана на рис. 5.

Динамические структуры хранят информацию о текущем строении адаптивной сетки и изменяются по мере ее перестройки. Соответственно, их время жизни совпадает с временем работы программы. Эти структуры используются на шаге 1 численного метода при адаптации сетки по мере изменения пространственного распределения массовой доли моделируемой примеси c с течением времени.

Массивы применяются на шаге 2 при решении системы (12) для хранения элементов матрицы A и векторов c^{n+1}, c^n, b. Время жизни массивов совпадает с временем выполнения этого шага. Предварительно (после шага 1, но перед шагом 2) они размещаются в памяти, причем размер массивов определяется текущим количеством узлов сетки. Далее производится перенос значений c^n из элементов динамических структур в ячейки соответствующего массива

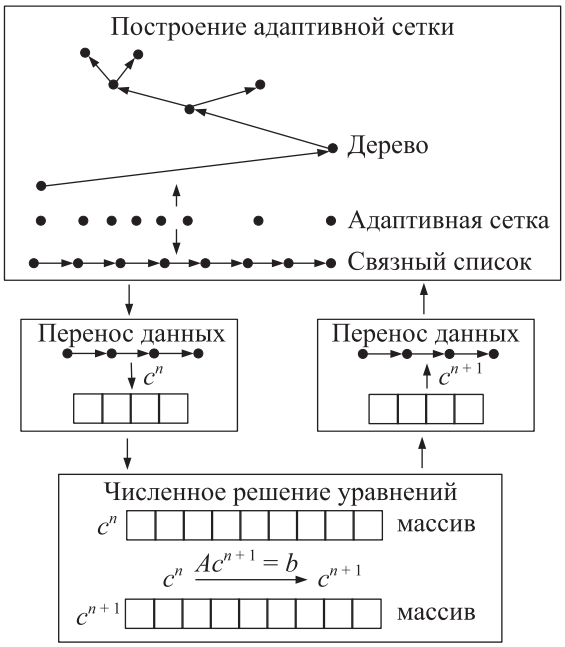

Рис. 5. Схема работы программы в пределах одной итерации численного метода. Итерация начинается с адаптации разностной сетки по найденному на предыдущей итерации пространственному распределению массовой доли примеси c^n. Во время адаптации сетки одновременно используются две динамические структуры данных — дерево и связный список. После перестроения сетки создается статическая структура данных — массив, в которую переносятся значения c^n. Фактически массив является текущей статической копией связного списка. Далее, массив используется во время численного решения уравнения переноса примеси, что сводится к решению системы линейных уравнений $Ac^{n+1} = b$. В результате в массиве значения c^n заменяются значениями c^{n+1}. Наконец, при завершении итерации данные из массива переносятся в связный список и, далее, в дерево, а сам массив удаляется из памяти

и рассчитываются значения элементов A и b. После завершения шага 2 происходит обратный перенос данных из ячеек массива c^{n+1} в соответствующие элементы динамических структур данных, массивы удаляются из памяти, и начинается новая итерация по времени.

Замена динамических структур на массивы на втором шаге численного метода приводит к уменьшению времени решения системы (12) при сохранении того же объема вычислений. Это достигается за счет более быстрого выполнения встроенных в Fortran матрично-векторных операций над целыми массивами по сравнению со скалярными операциями над элементами динамических структур.

Масштабируемость кода по количеству вычислительных ядер определяется по двум параметрам.

1. Ускорение $s(N) = t_1/t_N$ показывает, во сколько раз уменьшилось время t_N выполнения расчетов на N ядрах по сравнению с временем t_1 работы на одном ядре.

2. Эффективность распараллеливания $e(N) = s(N)/N \cdot 100\,\%$ показывает среднюю загруженность задействованных в расчетах ядер. Например, $e(N) = 70\,\%$ означает, что в среднем ядра производили полезные расчеты лишь в течение 70 % времени работы программы.

Время t_1 можно разложить на составляющие:

$$t_1 = t_s + t_p, \tag{13}$$

где t_s — время выполнения участков кода, которые невозможно распараллелить; t_p — время выполнения параллелизуемых частей кода. Тогда

$$t_N = t_s + t_p/N. \tag{14}$$

Последняя формула справедлива лишь в идеальном случае, когда все ядра могут работать с общей памятью параллельно без каких-либо задержек. Однако в реальных компьютерных системах шина памяти имеет ограниченную пропускную способность, что создает ситуацию конкуренции за доступ к памяти, когда возникает очередь запросов на чтение/запись от разных ядер, которые не могут быть удовлетворены одновременно. В результате в составе t_N появляется еще одно слагаемое t_o, обусловленное конечностью скорости передачи данных по шине памяти,

$$t_N = t_s + t_p/N + t_o, \tag{15}$$

которое показывает время простоя ядер при ожидании данных из памяти, причем с ростом N доля t_o в t_N увеличивается.

Симулятор gem5

Симулятор gem5 распространяется в виде исходного кода под лицензией BSD. Ссылку на скачивание и инструкцию по компиляции можно найти на официальном сайте симулятора www.gem5.org. В данной работе используется версия кода, актуальная на сентябрь 2018 года. Исходный код симулятора написан на языке C++, а для взаимодействия с пользователями используются скрипты языка Python.

Далее мы исходим из предположения, что на хосте (компьютере, где запускается gem5) и во всех симулируемых компьютерных системах используется ОС CentOS 7.

Для симуляции работы компьютерной системы gem5 требует наличие отдельного ядра Linux и образа диска со всем необходимым программным обеспечением. На сайте www.gem5.org для архитектуры x86 предлагаются ядро серии 2.6 и образ диска с Gentoo Linux, потерявшие свою актуальность более 10 лет назад. Поэтому ядро Linux собиралось самостоятельно.

Для сборки использовались исходники ядра последней на момент расчетов стабильной версии 4.18.14. Настройка ядра производилась с помощью подготовленного одним из разработчиков gem5 конфигурационного файла .config (www.lowepower.com/jason/files/config). Этот файл копировался в папку с исходниками ядра и далее выполнялись настройка и сборка:

```
make oldconfig
make vmlinux
```

Поскольку конфигурационный файл создавался под более ранную версию ядра (4.8.13), то на этапе настройки надо было задать значения ряда дополнительных параметров. Мы использовали значения, предлагаемые скриптом настройки по умолчанию. В итоге получается ядро vmlinux.

Пустой образ диска создается и монтируется в файловую систему хоста для последующей установки программного обеспечения входящим в состав gem5 скриптом gem5img.py с командами init (создание) и mount (монтирование). Далее, на смонтированный диск устанавливается ОС CentOS 7 посредством yumbootstrap. В установленной ОС необходимо задать пароль суперпользователя и режим загрузки в минимальной конфигурации. Настройка ОС осуществляется через chroot-окружение. Для этого командой chroot запускаем оболочку bash, корневым каталогом для которой задаем папку с монтированным диском (chroot-окружение оболочки bash). Соответственно, команды, выполняющиеся в этой оболочке, будут действовать исключительно в рамках ее chroot-окружения, как если бы это была отдельная файловая система. По умолчанию yumbootstrap не устанавливает пакет passwd, необходимый для задания паролей пользователей. Поэтому в bash сначала устанавливаем пакет passwd командой yum install и далее задаем пароль суперпользователя. Чтобы получить доступ к сети интернет из chroot-окружения, необходимо скопировать файл resolv.conf из папки /etc хоста в аналогичную папку смонтированного диска. Минимальный режим загрузки CentOS включаем командой:

```
systemctl set-default emergency.target
```

В итоге получаем рабочий образ диска.

В ходе симуляций были задействованы три типа процессоров gem5.

1. AtomicSimpleCPU — функциональная модель процессора с выполнением поступающих инструкций по порядку и мгновенным доступом к данным в памяти.

2. DerivO3CPU — тактовая модель процессора с внеочередным исполнением инструкций и конвейером из 5 стадий. Работа с памятью возможна в двух режимах: мгновенный доступ к данным (atomic access) и доступ с задержкой (timing access), учитывающий, например, одновременное обращение к памяти нескольких параллельно работающих процессов.

3. X86KvmCPU — модель процессора с аппаратным ускорением. Включение этого процессора в состав моделируемой компьютерной системы существенно увеличивает скорость выполнения симуляции. Однако, поскольку ускорение достигается за счет прямого выполнения большей части команд программ, работающих в рамках симуляции, на физическом процессоре хоста, замеряемые метрики симуляции будут определяться техническими характеристиками именно физического процессора.

Исходные коды gem5 с официального сайта содержат ошибки, делающие модель процессора X86KvmCPU неработоспособной. Чтобы их исправить, необходимо дополнительно установить специально написанный патч (gem5-review.googlesource.com/c/public/gem5/+/7362) и запускать gem5 от имени суперпользователя.

На рис. 6 показана схема выполнения симуляции в gem5. Симуляция проводится в два этапа. На первом этапе сразу после запуска gem5 проводится загрузка операционной системы (в нашем случае это CentOS 7) и в загруженной ОС запускается программа расчета движения примеси в атмосфере, масштабируемость которой мы исследуем. Программа инициализирует начальные данные (исходное положение облака примеси и соответствующую ему начальную адаптивную сетку) и выполняет четыре итерации численного алгоритма. Далее, в контрольной

Рис. 6. Схема симуляции в gem5. Симуляция идет в два этапа. На первом этапе выполняются необходимые подготовительные операции, которые не представляют интереса с точки зрения сбора статистической информации, но должны быть выполнены. На этом этапе используется модель процессора X86Kvm с аппаратным ускорением, что позволяет сократить время выполнения этого этапа до нескольких секунд. На втором этапе выполняется исследуемая часть кода программы и собирается статистика. Здесь используются процессорные модели AtomicSimpleCPU и DerivO3CPU, которые уже полностью имитируют работу процессора. Как следствие, этот этап может длиться несколько часов

точке gem5 останавливает симуляцию, записывает на диск ее текущие состояние и заканчивает работу. Первый этап является подготовительным, его единственная задача — как можно быстрее подвести симуляцию к выполнению интересующего нас участка кода программы. Поэтому на данном этапе симуляция запускается с процессором X86KvmCPU, благодаря чему время выполнения первого этапа не превышает нескольких десятков секунд.

На втором этапе симулируется выполнение целевого участка кода, по которому оценивается масштабируемость кода в целом. В нашем случае это пятая итерация численного алгоритма. Поскольку с алгоритмической точки зрения все итерации алгоритма идентичны друг другу, то результаты оценки одной итерации будут справедливы и для всех остальных, а значит, и для самой программы. На этом этапе используются процессоры AtomicSimpleCPU и DerivO3CPU. Симуляция начинается с сохраненного на предыдущем этапе состояния и завершается сразу после пятой итерации.

Создание контрольных точек и сбор статистики по симуляции обеспечивается входящим в состав исходников gem5 набором функций m5ops, которые необходимо включать непосредственно в код пользовательской программы и вызывать во время симуляции.

Симулятор gem5 содержит две модели системы памяти компьютера: Classic и Ruby. Модель Classic работает быстрее и сочетается со всеми типами процессоров gem5, что достигается за счет меньшей детализации реальных элементов систем памяти. Наличие кэш-памяти в данной модели ограничивает допустимое количество ядер процессора 16 штуками, исключение кэша увеличивает возможное число ядер до 64. Модель Ruby наиболее полно отражает принципы работы реальной физической памяти и позволяет подключать до 64 процессорных ядер на любой конфигурации. Поскольку в Ruby используется исключительно режим доступа с задержкой (timing access), эта модель не может использоваться с процессором AtomicSimpleCPU. В Ruby реализованы протокол когерентности кэша MESI и используемая фирмой AMD модификация MOESI. К сожалению, модификация MESIF, применяемая в процессорах фирмы Intel, на данный момент не включена в gem5.

В табл. 1 приведены параметры моделей трех компьютерных систем (модели 1, 2 и 3), работа которых симулировалась средствами gem5. Модель 1 — это простейшая функциональная модель без кэша. Модели 2 и 3 — тактовые модели, включающие двух- и трехуровневый кэши соответственно.

Для единообразия обозначений в табл. 1 в качестве модели 4 также приведен реальный компьютер, служивший хостом для gem5 и использовавшийся при разработке и оптимизации ко-

Таблица 1. Модели компьютерных систем. Модели 1, 2 и 3 симулировались в gem5, модель 4 — реальный компьютер

Модель	1	2	3	4
CPU, тип	AtomicSimple	DerivO3	DerivO3	Intel Core i5-8400
CPU, ядра	32	32	32	6
CPU, частота	3GHz	3GHz	3GHz	3GHz
Система памяти	Classic	Ruby	Ruby	Real
DRAM, тип	SimpleMemory	DDR4_2400	DDR4_2400	DDR4_2400
DRAM, размер	1 GB	1 GB	1 GB	8 GB
Когерентность кэша	—	MESI	MESI	MESIF
L1-I, размер, ассоц.	—	32kB, 8-way	32kB, 8-way	32kB, 8-way
L1-D, размер, ассоц.	—	32kB, 8-way	32kB, 8-way	32kB, 8-way
L2, размер, ассоц.	—	48MB, 12-way	256kB, 4-way	256kB, 4-way
L3, размер, ассоц.	—	—	48MB, 12-way	9MB, 12-way

да по расчету движения примесей в атмосфере. На время определения значений ускорения $s(N)$, достигаемых на модели 4, в процессоре этой модели отключалось динамическое изменение тактовой частоты, которая фиксировалась на значении 3 GHz для любого количества активных ядер.

Конфигурация кэша (протокол когерентности и число уровней) модели памяти Ruby определяется на этапе компиляции исходного кода gem5 через переменную PROTOCOL и в дальнейшем изменена быть не может. Поэтому для каждой из моделей 1–3 должен быть скомпилирован отдельный исполняемый файл. Для модели 1, где используется модель памяти Classic, компиляция идет со значением по умолчанию, а для моделей 2 и 3 при компиляции необходимо указать PROTOCOL=MESI_Two_Level и PROTOCOL=MESI_Three_Level соответственно.

Во время запуска исполняемого файла gem5 создает модель компьютерной системы путем выполнения скрипта fs.py с необходимыми опциями настройки (тип процессора, тип памяти, размер памяти и т. д.). Скрипт fs.py формирует модели наиболее распространенной стандартной структуры «процессор – кэш – оперативная память». Если требуется что-то специфическое, то необходимо уже писать отдельный скрипт под целевую модель.

Необходимо отметить, что в модели системы памяти Ruby нумерация уровней двухуровневого кэша стандартная (L1 и L2), а нумерация уровней трехуровневого кэша сдвинута на единицу вниз, т. е. уровни L1, L2 и L3 обозначаются как L0, L1 и L2, причем параметры кэша первого уровня (L0 в Ruby) можно задать только путем внесения правок в файл MESI_Three_Level.py.

Как было указано ранее, каждая симуляция выполняется в два этапа. Задачей первого этапа является подведение симуляции к моменту начала исполнения целевого кода за минимально возможное время. На этом этапе gem5 запускается с передачей скрипту fs.py всего трех опций: --cpu-type=X86KvmCPU, --mem-size=1GB, --num-cpus=32. Первая опция задает тип процессора, вторая определяет размер оперативной памяти, а третья задает число вычислительных ядер. Вторую и третью опции необходимо задавать на первом этапе, поскольку gem5 не позволяет изменять их значения во время возобновления симуляции из контрольных точек. На втором этапе gem5 начинает симуляцию из контрольной точки Checkpoint (см. рис. 6), и при запуске ему передаются уже все определяющие конкретную модель опции. Например, для модели 3 имеем

```
--cpu-type=DerivO3CPU --cpu-clock=3GHz --ruby
--mem-type=DDR4_2400_8x8 --mem-size=1GB --l1d_size=256kB
--l1d_assoc=4 --l2_size=48MB --l2_assoc=12 --num-cpus=32
--restore-with-cpu=X86KvmCPU --checkpoint-restore=1.
```

Таблица 2. Ускорение $s(N)$ программы расчета переноса примеси в атмосфере на моделях 1–4 при одновременном использовании динамических и статических структур данных. В колонке 4а дополнительно приведено ускорение программы на модели 4, когда в расчетах используются только динамические структуры данных как на этапе построения адаптивной сетки, так и на этапе численного решения уравнения переноса

Ядра/Модель	1	2	3	4	4а
2	1.99	1.95	1.95	1.95	1.91
3	2.95	2.87	2.92	2.92	2.70
4	3.94	3.71	3.84	3.81	3.51
5	4.84	4.57	4.77	4.74	4.28

Здесь --ruby включает модель памяти Ruby (по умолчанию используется модель Classic), --l1d_size, --l1d_assoc и --l2_size, --l2_assoc определяют размер и ассоциативность кэшей второго и третьего уровней (аналогичные параметры для кэша первого уровня должны быть заранее прописаны вручную в файле MESI_Three_Level.py), --restore-with-cpu указывает на тип процессора, который использовался на момент создания контрольной точки, --checkpoint-restore определяет номер контрольной точки, с которой начинается симуляция, значение остальных опций достаточно очевидно из их названия.

Результаты расчетов

Ускорение работы нашей программы $s(N)$ на каждой из рассмотренных моделей компьютерных систем при распределении вычислительной нагрузки на 2–5 ядер по сравнению с последовательным исполнением на одном ядре тех же моделей показано в табл. 2. Результаты для моделей 1, 2 и 3 получены в симуляторе gem5, а для модели 4 — путем запуска программы на реальном компьютере, который представляет эта модель. Также в последнем столбце с номером 4а приведено ускорение, которое мы получали на модели 4 для предыдущей реализации численного метода при использовании исключительно динамических структур данных как при построении сетки, так и при вычислении на ней новых значений массовой доли примеси (см. [Семакин, 2017]). Как видно из столбцов 4 и 4а, переход к массивам на втором этапе численного алгоритма позволил повысить эффективность распараллеливания на 10 %. В свою очередь, время работы новой программной реализации на одном ядре уменьшилось в два раза по сравнению со старой реализацией за счет использования векторных операций языка Fortran над массивами вместо скалярных (поэлементных) операций над элементами динамических структур на втором этапе численного алгоритма.

Сравнивая столбец 4 со столбцами 1, 2 и 3, можно заключить, что модель 3 дает наиболее близкие к реальности значения ускорения. Это объясняется тем, что данная модель имеет наиболее близкое к реальному компьютеру строение процессора и системы памяти.

На рис. 7 и 8 показаны ускорение $s(N)$ и эффективность распараллеливания $e(N)$ на моделях 1–3 при распараллеливании вычислений на 1–32 ядра. Наилучшие результаты показывает функциональная модель 1: $s(32) = 26.1$, $e(32) = 82\,\%$. В этой модели все ядра при параллельной работе одновременно получают данные из памяти или записывают их в память, т. е. в формуле (15) $t_o = 0$. Соответственно, функциональная модель дает верхнюю предельную оценку масштабируемости по вычислительным ядрам, которую теоретически можно достичь на идеально сбалансированном оборудовании.

Модель 2 с двухуровневым кэшем дает наихудшие результаты: $s(32) = 14.2$, $e(32) = 44\,\%$. Переход к трехуровневому кэшу (в модели 3) существенно улучшает эффективность распараллеливания: $s(32) = 22.2$, $e(32) = 69\,\%$. Такой резкий рост в эффективности можно объяснить

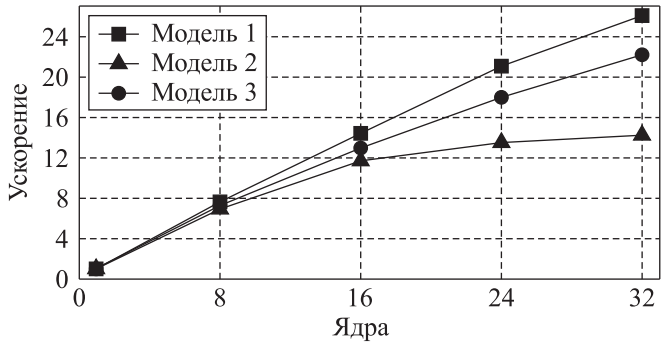

Рис. 7. Ускорение $s(N)$ программы расчета переноса примеси в атмосфере на ядрах 2–32 по сравнению с одним ядром. Результаты получены в симуляторе gem5

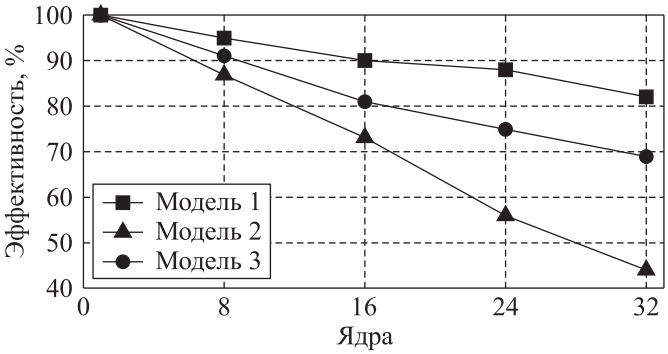

Рис. 8. Эффективность распараллеливания $e(N)$ во время выполнения программы расчета переноса примеси в атмосфере. Результаты получены в симуляторе gem5

одновременным действием трех факторов: 1) появление промежуточного уровня делает работу кэша более эффективной; 2) система памяти модели 3 наиболее близка к системе памяти компьютера (модель 4), на котором создавался и далее оптимизировался код программы; 3) различие в реализации моделей двух- и трехуровневого кэшей в самой gem5 также может сказываться на результате.

Симуляции в gem5 происходят на достаточно продолжительных промежутках времени, которые, однако, остаются приемлемыми для многократного тестирования программ. Так, выполнение целевого участка кода программы (этап 2 на рис. 6) на модели 1 при ее симуляции в gem5 занимает в 10^4 раз больше времени, чем аналогичное выполнение на реальном компьютере. В свою очередь, во время симуляции модели 2 целевой код выполняется в 10 раз медленнее, чем на модели 1, а модель 3 работает на 10 % медленнее модели 2. В абсолютных единицах целевой код на реальном компьютере выполняется за 20–40 мс, на модели 1 — 10 мин, на модели 2 — 1.5 часа и минут на 10–15 дольше на модели 3. Здесь указано приблизительное среднее время по результатам нескольких десятков запусков. Разумеется, абсолютные цифры зависят от конкретного компьютера, на котором производятся симуляции.

Заключение

В данной статье мы представили новую версию программной реализации численного алгоритма расчета переноса примесей в атмосфере на адаптивной сетке. Благодаря более эффективной схеме работы с памятью (одновременное использование динамических и статических структур данных для представления разностной сетки) время расчетов сократилось в 2 раза по сравнению с предыдущей версией программной реализации, основанной исключительно

на динамических структурах данных, и на 10 % выросла способность кода к масштабируемости по вычислительным ядрам. Программный код разрабатывался и оптимизировался на компьютере, оснащенном всего лишь шестью вычислительными ядрами, мощности которых, к сожалению, уже не достаточно для проведения высокоточных расчетов трехмерных движений примесей на межконтинентальные расстояния. Однако выполненные на симуляторе gem5 оценки показали, что созданный код поддерживает достаточно высокий уровень параллелизма на существенно большем количестве вычислительных ядер (вплоть до 32). Практически это означает, что данный код можно использовать на мощных многоядерных рабочих станциях без каких-либо доработок и дополнительных оптимизаций, при этом мы гарантированно получим существенный прирост производительности в расчетах.

Список литературы (References)

Семакин А. Н. Программное обеспечение для создания адаптивных сеток // Труды ИСП РАН. — 2017. — Т. 29, вып. 5. — С. 311–328.
Semakin A. N. Programmnoe obespechenie dlya sozdaniya adaptivnykh setok [Software for adaptive grid construction] // Trudy ISP RAN [Proceedings of ISP RAS]. — 2017. — Vol. 29, Iss. 5. — P. 311–328 (in Russian).

Aarno D., Engblom J. Software and system development using virtual platforms: full-system simulation with Wind River Simics. — Morgan Kaufmann, 2015. — 366 p.

Akram A., Sawalha L. A comparison of x86 computer architecture simulators, Tech. Rep. TR-CASRL-1-2016. — Western Michigan University. — Kalamazoo, MI, USA, 2016.

Austin T., Larson E., Ernst D. SimpleScalar: an infrastructure for computer system modeling // Computer. — 2002. — Vol. 35, No. 2. — P. 59–67.

Baeza A., Martinez-Gavara A., Mulet P. Adaptation based on interpolation errors for high order mesh refinement methods applied to conservation laws // Appl. Numer. Math. — 2012. — Vol. 62. — P. 278–296

Bellard F. QEMU, a fast and portable dynamic translator // The proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference. — 2005. — P. 41–46.

Berger M. J., Colella P. Local adaptive mesh refinement for shock hydrodynamics // J. Comput. Phys. — 1989. — Vol. 82. — P. 64–84.

Bey I., Jacob D. J., Yantosca R. M., Logan J. A., Field B. D., Flore A. M., Li Q., Liu H. Y., Mickley L. J., Schultz M. G. Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation // Journal of Geophysical Research: Atmospheres. — 2001. — Vol. 106, Iss. D19. — P. 23073–23095.

Binkert N., Beckmann B., Black G., Reinhardt S. K., Saidi A., Basu A., Hestness J., Hower D. R., Krishna T., Sardashti S., Sen R., Sewell K., Shoaib M., Vaish N., Hill M. D., Wood D. A. The gem5 simulator // ACM SIGARCH Computer Architecture News. — 2011. — Vol. 39, No. 2. — P. 1–7.

Butko A., Garibotti R., Ost L., Sassatelli G. Accuracy evaluation of gem5 simulator system // 2012 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip. — 2012. — P. 1–7.

Constantinescu E. M., Sandu A., Carmichael G. R. Modeling atmospheric chemistry and transport with dynamic adaptive resolution // Comput. Geosci. — 2008. — Vol. 12. — P. 133–151.

Danabasoglu G., Lamarque J.-F., Bacmeister J., Bailey D. A., DuVivier A. K., Edwards J., Emmons L. K., Fasullo J., Garcia R., Gettelman A., Hannay C., Holland M. M., Large W. G., Lauritzen P. H., Lawrence D. M., Lenaerts J. T. M., Lindsay K., Lipscomb W. H., Mills M. J., Neale R.,

Oleson K. W., Otto-Bliesner B., Phillips A. S., Sacks W., Tilmes S., van Kampenhout L., Vertenstein M., Bertini A., Dennis J., Deser C., Fischer C., Fox-Kemper B., Kay J. E., Kinnison D., Kushner P. J., Larson V. E., Long M. C., Mickelson S., Moore J. K., Nienhouse E., Polvani L., Rasch P. J., Strand W. G. The Community Earth System Model Version 2 (CESM2) // Journal of Advances in Modeling Earth Systems. — 2020. — Vol. 12, Iss. 2.

Daubechies I. Ten lectures on wavelets. — SIAM, 1992. — 377 p.

Donoho D. L. Interpolating wavelet transforms. — Tech. Rep. 408. — Department of Statistics, Stanford University, 1992. — 54 p.

Durran D. R. Numerical methods for fluid dynamics with applications to geophysics. — Springer, 2010. — 516 p.

Eyring V., Bony S., Meehl G. A., Senior C., Stevens B., Stouffer R. J., Taylor K. E. Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization // Geoscientific Model Development. — 2016. — Vol. 9, Iss. 5. — P. 1937–1958.

Fuller S. H., Millett L. I. The future of computing performance: game over or next level? — The National Academies Press, 2011. — 200 p.

Grell G. A., Peckham S. E., Schmitz R., McKeen S. A., Frost G., Skamarock W. C., Eder B. Fully coupled “online” chemistry within the WRF model // Atmospheric Environment. — 2005. — Vol. 39, Iss. 37. — P. 6957–6975.

Gutierrez A., Pusdesris J., Dreslinski R., Mudge T., Sudanthi C., Emmons C., Hayenga M., Paver N. Sources of error in full-system simulation // 2014 IEEE International Symposium on Performance Analysis of Systems and Software. — 2014. — P. 13–22.

Hansson A., Agarwal N., Kolli A., Wenisch T., Udipi A. N. Simulating DRAM controllers for future system architecture // 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). — 2014. — P. 201–210.

Hardavellas N., Somogyl S., Wenisch T. F., Wunderlich R. E., Chen S., Kim J., Falsafi B., Hoe J. C., Nowatzyk A. G. SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture // ACM SIGMETRICS Performance Evaluation Review. — 2004. — Vol. 31, Iss. 4. — P. 31–34.

Huijnen V., Williams J. E., van Weele M., van Noije T. P. C., Krol M. C., Dentener F., Segers A., Houweling S., Peters W., de Laat A. T. J., Boersma K. F., Bergamaschi P., van Velthoven P. F. J., Le Sager P., Eskes H. J., Alkemade F., Scheele M. P., Nedelec P., Patz H.-W. The global chemistry transport model TM5: description and evaluation of the tropospheric chemistry version 3.0 // Geoscientific Model Development. — 2010. — Vol. 3, Iss. 2. — P. 445–473.

Hundsdorfer W., Verwer J. G. Numerical solution of time-dependent advection-diffusion-reaction equations. — Springer, 2010. — 471 p.

Jagtap R., Jung M., Elsasser W., Weis C., Hansson A., Wehn N. Integrating DRAM power-down modes in gem5 and quantifying their impact // The proceedings of the International Symposium on Memory Systems. — 2017. — P. 86–95.

Kasahara A. Various vertical coordinate systems used for numerical weather prediction // Monthly Weather Review. — 1974. — Vol. 102. — P. 509–522.

Kish L. B. End of Moore’s law: Thermal (noise) death of integration in micro and nano electronics // Phys. Lett. A. — 2002. — Vol. 305. — P. 144–149.

Klees R., Haagmans R. Wavelets in the geosciences. — Springer-Verlag, 2000. — 256 p.

Kopera M. A., Giraldo F. X. Analysis of adaptive mesh refinement for IMEX discontinuous Galerkin solutions of the compressible Euler equations with application to atmospheric simulations // J. Comput. Phys. — 2014. — Vol. 275.— P. 92–117.

Muller A., Kopera M., Giraldo F. X. Strong scaling on more than 700,000 cores with the atmospheric model NUMA // ICMS Workshop on Galerkin Methods with Applications in Weather and Climate Forecasting. — Edinburgh, United Kingdom.

Newell R. E., Thouret V., Cho J. Y. N., Stoller P., Marenco A., Smith H. G. Ubiquity of quasi-horizontal layers in the troposphere // Nature. — 1999. — Vol. 398. — P. 316–319.

Paolucci S., Zikoski Z. J., Wirasaet D. WAMR: An adaptive wavelet method for the simulation of compressible reacting flow. Part I. Accuracy and efficiency of algorithm // J. Comput. Phys. — 2014. — Vol. 272. — P. 814–841.

Patel A., Afram F., Chen S., Ghose K. MARSS: a full system simulator for multicore x86 cpus // The proceedings of the 48th Design Automation Conference (DAC). — 2011. — P. 1050–1055.

Pau G. S. H., Bell J. B., Almgren A. S., Fagnan K. M., Lijewski M. J. An adaptive mesh refinement algorithm for compressible two-phase flow in porous media // Comput. Geosci. — 2012. — Vol. 16. — P. 577–592.

Powers J. G., Klemp J. B., Skamarock W. C., Davis C. A., Dudhia J., Gill D. O., Coen J. L., Gochis D. J., Ahmadov R., Peckham S. E., Grell G. A., Michalakes J., Trahan S., Benjamin S. G., Alexander C. R., Dimego G. J., Wang W., Schwartz C. S., Romine G. S., Liu Z., Snyder C., Chen F., Barlage M. J., Yu W., Duda M. G. The Weather Research and Forecasting model: Overview, system efforts, and future directions // Bulletin of the American Meteorological Society. — 2017. — Vol. 98, No. 8. — P. 1717–1737.

Quirk J. J. A parallel adaptive grid algorithm for computational shock hydrodynamics // Appl. Numer. Math. — 1996. — Vol. 20. — P. 427–453.

Sandberg A., Nikoleris N., Carison T. E., Hagersten E., Kaxiras S., Black-Schaffer D. Full speed ahead: detailed architectural simulation at near-native speed // 2015 IEEE International Symposium on Workload Characterization. — 2015. — P. 183–192.

Schneider K., Vasilyev O. V. Wavelet methods in computational fluid dynamics // Annu. Rev. Fluid Mech. — 2010. — Vol. 42.— P. 473–503.

Seaman N. L. Meteorological modeling for air-quality assessments // Atmospheric Environment. — 2000. — Vol. 34, Iss. 12–14. — P. 2231–2259.

Semakin A. N., Rastigejev Yu. Numerical simulation of global-scale atmospheric chemical transport with high-order wavelet-based adaptive mesh refinement algorithm // Monthly Weather Review. — 2016. — Vol. 144, No. 4. — P. 1469–1486.

Sportisse B. Fundamentals in air pollution: from processes to modelling. — Springer, 2010. — 299 p.

Thakur P., Ballard S., Nelson R. Radioactive fallout in the United States due to the Fukushima nuclear plant accident // Journal of Environmental Monitoring. — 2012. — Vol. 14, Iss. 5.— P. 1317–1324.

Vasilyev O. V. Solving multi-dimensional evolution problems with localized structures using second generation wavelets // International Journal of Computational Fluid Dynamics. — 2003. — Vol. 17, Iss. 2. — P. 151–168.

Yourst M. T. PTLsim: a cycle accurate full system x86-64 microarchitectural simulator // IEEE International Symposium on Performance Analysis of Systems and Software. — 2007. — P. 23–34.