ISSN 2079-3316 **ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ** **т. 9**, №4(39), с. 361–381

ББК 32.973.5
ГРНТИ 50.41.23
УДК 519.682.3, 519.245

В. А. Роганов, В. И. Осипов, Г. А. Матвеев

Интеграция пакета Mantevo MiniAMR и OpenTS Сравнение OpenTS DMPI и Open MPI

Аннотация. Mantevo MiniAMR – инструментальное средство для распараллеливания широкого класса 3D приложений с использованием адаптивной расчетной сетки. Оно может быть применено для параллельной реализации разностных схем; систем, которые моделируются с помощью уравнений с частными производными. Проведена интеграция пакета MiniAMR и системы для параллельного программирования OpenTS. Произведено сравнение OpenTS DMPI и Open MPI.

Ключевые слова и фразы: адаптивный алгоритм, параллельный алгоритм, язык программирования T++, OpenTS, T-Система, adaptive mesh refinement.

Введение

T-система является оригинальной российской разработкой, объединяющей в себе наиболее удачные черты функционального программирования, dataflow-систем и традиционных языков и методов программирования. T-система базируется на функциональной парадигме и предполагает определенные ограничения на стиль программирования в случае ее использования. Взамен она предоставляет бесконфликтную модель динамического распараллеливания, в которой невозможны взаимные блокировки и некорректный доступ к разделяемым переменным.

T-система с открытой архитектурой (OpenTS) [**1**–**9**] была разработана в ИПС им. А. К. Айламазяна РАН в рамках суперкомпьютерной программы «СКИФ» Союзного государства, представляет собой современную реализацию идей T-системы и обеспечивает лучшую интеграцию базовых возможностей функционального подхода с возможностями языка программирования Си++, чем предыдущие варианты. Эта версия

© В. А. Роганов, В. И. Осипов, Г. А. Матвеев, 2018
© Институт программных систем имени А. К. Айламазяна РАН, 2018
© Программные системы: теория и приложения (дизайн), 2018
10.25209/2079-3316-2018-9-4-361-381

Т-системы обладает открытой и масштабируемой архитектурой, легко адаптируемой к стремительно меняющимся аппаратным платформам современных суперкомпьютеров. Поддерживаемый системой OpenTS входной язык программирования Т++ является синтаксически и семантически гладким расширением языка программирования Си++, а среда исполнения Т-приложений представляет собой ортогональную надстройку (Т-суперструктуру) над стандартной последовательной средой программирования.

Подход к автоматическому динамическому распараллеливанию программ, предложенный в Т-системе, позволяет получить хорошие результаты по утилизации вычислительной мощности современных кластерных установок, что связано с природой используемой модели вычислений. Система OpenTS ассимилирует многие другие технологии параллельного программирования: специальную модель общей памяти, модель распределенных потоков и объектов, распределенную сборку мусора, и, наконец, высокоуровневую языковую надстройку, являющуюся уникальной по своим характеристикам технологией для поддержки максимальной совместимости с традиционными языками по синтаксису и семантике, но при этом эффективно распараллеливаемой моделью вычислений.

MiniAMR [24, 25] — небольшое приложение для работы с адаптивными расчетными сетками (AMR — Adaptive Mesh Refinement). Оно воспроизводит часто встречающиеся задачи и модели взаимодействия в приложениях AMR.

Далее описывается интеграция пакета Mantevo MiniAMR с системой для параллельного программирования OpenTS, а также приводятся результаты сравнительных испытаний интегрированного пакета с реализацией пакета MiniAMR под управлением Open MPI.

Статья помогает понять принципы работы адаптивных сеток в целом, и может быть полезна разработчикам, которые хотят перейти в своих приложениях от равномерных сеток к адаптивным для повышения производительности.

1. Описание пакета MiniAMR

MiniAMR — компактная библиотека для работы с адаптивными расчетными сетками, основанными на октодеревьях — рекурсивных структур данных, каждый узел которых содержит восемь потомков [10, 13, 17, 18, 20–22]. Высокая эффективность MiniAMR достигается путем оптимальной обработки структур данных.

Адаптивные расчетные сетки последнее время все чаще используются для решения уравнений с частными производными. Мелкая сетка используется в областях, требующих более высокого разрешения (например, из-за динамически возникающей турбулентности), а для обработки областей без особенностей используется более редкая сетка. Это снижает расход вычислительных ресурсов (как в части времени вычисления, так и в части необходимой памяти) по сравнению с равномерной сеткой. Поскольку области, требующие мелкой/грубой расчетной сетки, могут меняться в процессе вычислений, соответствующие изменения отражаются в структуре октодерева. Это влияет на объем вычислений, выполняемых каждым процессором; поэтому время от времени становится необходимо перераспределять вычисления на сетке между процессорами. Хорошее распределение минимизирует обмен между процессами и выравнивает вычислительную нагрузку, уменьшая тем самым время выполнения. Перераспределение расчета узлов сетки между процессорами не должно происходить слишком часто; это должно быть сбалансировано с затратами на перераспределение расчета узлов сетки между процессорами, перенос данных и восстановление распределенных структур данных, используемых приложением. Многие приложения AMR используют заполняющие пространство кривые (SFC — Space filling curves) для распределения вычислений на сетке, поскольку они помогают эффективно решить эту задачу.

В библиотеке MiniAMR для представления октодеревьев используется улучшенная линеаризованная структура данных. Для сгущения/разрежения сетки используется низкозатратный алгоритм адаптации, а также используется схема распределения вычислений, которая уменьшает общее время выполнения, уменьшая частоту перераспределения ресурсов.

В пакете реализован низкозатратный взвешенный алгоритм распараллеливания для SFC, который может использоваться для балансировки нагрузки приложений AMR, использующих субциклинг [**23**].

MiniAMR использует 3D сеть с октодеревьями следующим образом: смежные ячейки сети группируются в кубические блоки, степень детализации которых может быть определена использующим ее приложением. Каждый блок имеет область обмена, определенную вокруг своих ячеек сетки, которая используется для обмена граничных значений с соседними блоками. Если блок следует измельчить, он делится на 8 кубических вложенных блоков; если блоки укрупняются,

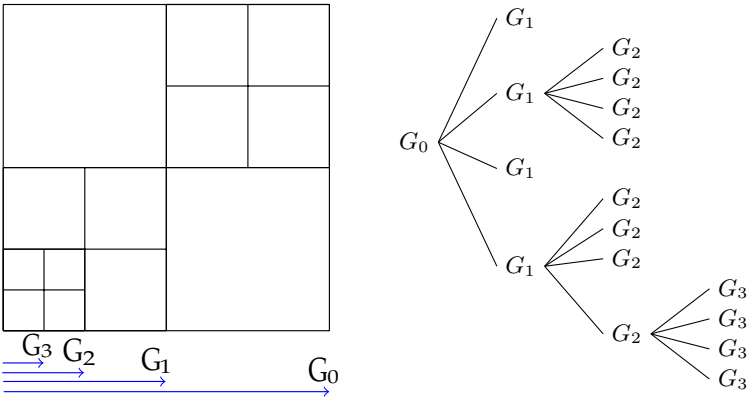

Рисунок 1. Схема представления блочно-структурированной расчетной сетки и соответствующее ей октодерево

то 8 соседних вложенных блоков заменяются исходным родительским блоком. При адаптации сохраняется условие сбалансированности 2:1: уровни измельчения соседних блоков отличаются не более чем на 1. Приложение использует так называемый семиточечный шаблон, так что любые два блока являются соседними, если они имеют общую грань. В вычислениях участвуют только листья октодерева. Приложение может моделировать AMR, использующие субциклинг.

На абстрактном уровне, структура данных сетки представляет собой динамический лес октодеревьев с корнями леса, являющимися начальными грубыми блоками сетки. Узлы октодерева эквивалентны блокам сетки, определенным ранее (далее будем использовать эти термины взаимозаменяемо). Эта структура данных поддерживает следующие операции:

- Измельчение: 8 узлов создаются как дочерние элементы предыдущего листового узла.
- Укрупнение: 8 узлов-братьев удаляются, а их родительский узел становится листовым узлом.

На рисунке 1 приведен пример адаптивной сетки, которая удовлетворяет критерию баланса 2:1 и соответствует 2D 4-дереву.

Для параллельного вычисления структура данных октодерева распределяется по вычислительным узлам. В процессе распределения необходимо балансировать количество вычислений, выполняемых каждым узлом, при одновременном снижении обменов. Основными

ресурсоемкими, управляющими процессом вычислений операциями в AMR на основе октодерева являются следующие процедуры:

Балансировка: операции измельчения и укрупнения должны сохранять условие баланса 2:1. Измельчение узла может привести к дальнейшему измельчению грубых соседей. Аналогично, укрупнение группы узлов может привести к укрупнению соседней группы. Обе операции требуют связи между узлом и его соседями.

Обновление блоков обмена: Блоки обмениваются граничными данными со своими соседями в каждый момент времени моделирования. Накладные расходы на коммуникацию зависят от типа используемой схемы конечной разности. В простейшем случае симуляция происходит только на листовых узлах октодерева. В этом случае только листовые блоки выполняют граничный обмен. Если ядро блока тоже измельчается, то скорость обмена между соседними блоками зависит от их уровня измельчения. Накладные расходы различны для каждого типа ядра. Но все варианты в итоге выигрывают от декомпозиции сетки, которая сбалансирована по нагрузке и локально устроена так, чтобы соседние блоки в основном хранились на одном вычислительном узле (хорошая привязка к вычислительным узлам).

Как правило, параллельная сетевая платформа октодерева начинает с распределения узлов октодерева процессам и адаптивно модифицирует ее путем измельчения и укрупнения узлов дерева в ходе моделирования. Это может привести к перебоям с дисбалансом нагрузки и ухудшению связи при балансировке и обмене граничными значениями. Поэтому октодерево необходимо периодически перераспределять по вычислительным узлам. В процессе вычисления нового распределения происходит перенос данных и восстановление информации, необходимой для обмена граничными значениями. Для распределения адаптивной сетки используются кривые заполнения пространства. SFC используется для создания линейного упорядоченного списка блоков сетки (листьев деревьев), которые затем распределяются между участвующими процессами с балансировкой нагрузки. При этом внутренний блок хранится на том же узле, что и один из его дочерних элементов.

SFC можно использовать для распределения линейных сеток: каждая ячейка сетки представлена точкой, обычно расположенной в ее центре тяжести. Кривая генерируется с использованием пространственных координат точек. Для преобразования пространственных координат в уникальные ключи используется монотонная функция —

значения ключей, генерируемых функцией, должны сохранять порядок прохождения кривой. Затем точки разбиваются на p подмножеств, разбивая набор ключей на p сегментов, каждый из которых содержит примерно одинаковое количество точек, обеспечивая баланс нагрузки. Разделы имеют хорошую привязку к вычислительным узлам, поэтому, если для вычисления требуется связь между соседними ячейками сетки, затраты на коммуникацию будут минимальны.

Дискретная k-мерная кривая заполнения пространства (SFC) представляет собой биективное отображение $C : \{1, \ldots, N^k\} \to \{1, \ldots, N\}^k$ такое, что $d(C(i), C(i+1)) = 1$ для всех $0 < i < N^k$, где d — евклидово расстояние [14]. Определим *хорошую привязку к вычислительным узлам* как требование, что любые k последовательных точек $C(i), \ldots, C(i+k-1)$ на SFC содержались в гиперкубе размерности $O(k^{1/m})$, где m — любое ненулевое целое число. Некоторые из популярных SFC – кривые Гильберта, Пеано и Серпинского [19]. Хотя порядок (кривая) Мортона [11] не соответствует указанному выше определению, он широко используется из-за простоты вычислений. Функция отображения для порядка Мортона представляет собой простое чередование в пространстве двоичных цифр координат точек.

В MiniAMR используется порядок Мортона. Для адаптивной сетки октодерева избегается повторное вычисление SFC с нуля всякий раз, когда октодерево будет измельчаться или огрубляться. Для этого, когда блок с ключом ℓ измельчается, он заменяется на SFC восемью детьми с ключами «$\ell\ xyz$», помеченными в порядке прохождения кривой (в этом случае порядок Мортона). Когда 8 братских блоков огрубляются, грубый блок (родительский) заменяет один из дочерних блоков на SFC, а остальные дети удаляются. Для порядка Мортона, поскольку 8 братьев были смежными на SFC, у них были двоичные ключи $\ell\ xyz$, а грубый блок повторно помечен ℓ. Лексикографический порядок ключей сохраняется. Кроме того, каждый процесс может выполнять эти обновления независимо друг от друга, для тех блоков, которыми он владеет.

Чтобы перераспределить сеть, нам нужно вычислить (взвешенный) индекс всех точек (листовых узлов) на SFC. Когда узел измельчается, его дети хранятся в том же процессе, что и их родитель; Когда узлы огрубляются, родительский узел уже хранится в том же процессе, что и один из дочерних элементов. Поэтому операции измельчения и укрупнения сохраняют инвариант: все узлы, хранящиеся в процессе P_i, имеют ключи, которые в лексикографическом порядке строго меньше,

чем узлы, хранящиеся в процессе P_{i+1}. В результате достаточно сортировать ключи локально в каждом процессе, чтобы получить глобальную сортировку.

Каждый процесс локально задает свои ключи в отсортированном порядке. Параллельная предварительная операция используется для вычисления глобального ранга последнего ключа. После этого шага процессы могут самостоятельно вычислять глобальные ранги своих локальных узлов и повторно распределять их с балансировкой нагрузки.

Во время адаптации блоки имеют весовое значение, пропорциональное уровню их измельчения (поскольку мелкие блоки обновляются чаще). Кривая нарезается на равные весовые сегменты.

Опишем модель производительности и показатели, которые используются для оценки производительности мини-приложения. Ядро AMR состоит из нижеописанных неперекрывающихся фаз, которые выполняются с разными частотами. Ядро выполняет итерацию до сходимости, и мы рассматриваем каждую итерацию как единый временной шаг.

Измельчение / Огрубление: ядро проверяет критерии проверки на каждом шаге времени ref_freq и, если указано, выполняет проверку и укрупнение блоков. Добавление и / или удаление блоков зависит от критерия баланса 2:1, рассмотренного ранее.

Балансировка нагрузки: перераспределение узлов может выполняться после измельчения / укрупнения. Это включает в себя вычисление нового распределения, повторное распределение данных и вычисление новой модели взаимодействия (обмена), которая будет использоваться в последующие временные интервалы.

Вычисление: каждый шаг времени каждый листовой узел обновляет свои данные на основе соседних значений, используя 7-точечный шаблон. Для этого шага требуется обмен данными через общие грани листовых узлов.

Мы измеряем на каждом шаге i и каждом процессе j время, затраченное на вычисление $Tcomp(i, j)$, время $Tcomm(i, j)$, потраченное на обмен, время $Tref(i, j)$, потраченное на измельчение и укрупнение сетки, и время $Tlb(i, j)$ балансировка нагрузки. Пусть n — общее количество временных шагов, $ref_freq = r$, и пусть $t1, \ldots, tk$ — временные шаги, в которые происходит балансировка нагрузки. Тогда

общее время выполнения T_{exec} может быть определено как

$$T_{exec} = \sum_{t=1}^{n} T_{iter}(t) + \sum_{t=1}^{n/r} T_{ref}(rt) + \sum_{i=1}^{k} T_{lb}(t_i),$$

где

$$T_{iter}(i) = \max_j (T_{comp}(i,j) + T_{comm}(i,j)),$$
$$T_{ref}(i) = \max_j T_{ref}(i,j),$$
$$T_{lb}(i) = \max_j T_{lb}(i,j).$$

Представленные здесь уравнения выполняются для общего случая без какого-либо перекрытия вычислений в ядре. Однако реализация использует вычисление и обмен для шаблона разностной схемы. В случае полного перекрытия уравнение для T_{iter} становится

$$T_{iter}(i) = \max_j (\max(T_{comp}(i,j), T_{comm}(i,j))).$$

MiniAMR было разработано, чтобы лучше понимать коммуникационные накладные расходы в AMR на основе октодерева и использовать полученную модель для разработки алгоритмов с более низкими коммуникационными расходами. Ядро вычислений представляет собой простой трехмерный семиточечный шаблон. Задается начальное распределение блоков и область адаптации. Можно задать также параметры адаптации и частоту балансировки. Блоки, которые перекрываются с областью адаптации, отмечаются для изменения. Форму и область адаптации можно изменять, чтобы генерировать различные тестовые примеры. Область адаптации можно расширять или перемещать с постоянной скорость. Более эффективное распределение должно иметь более значительное влияние на время коммуникации, если используется более сложный шаблон разностной схемы. С другой стороны, использование более интенсивного вычислительного ядра уменьшит относительный вес коммуникаций (но не общий вес).

Распределенные структуры данных на основе указателей, такие как деревья, обычно реализуются путем явного хранения адресов их узлов, то есть идентификатора удаленного процесса и индекса глобального узла. Это требует дополнительного учета в ходе обновления структур данных; измельчение и укрупнение в этом случае: после каждой фазы адаптации и балансировки информация о соседстве обновляется путем отправки и получения сообщений, содержащих глобальные

идентификаторы новых блоков. Кроме того, после балансировки нагрузки требуется дополнительные коммуникации для обновления новых местоположений соседей.

Использование ключей SFC вместо указателей уменьшает накладные расходы этих операций, поскольку ключи обеспечивают уровень абстракции, который не зависит от декомпозиции базовых данных. Единственной информацией, необходимой для каждого процесса, является функция/таблица сопоставления, которая может выводить местоположение блока с учетом его ключа SFC. Эта идея была исследована в [**16**], где авторы сохранили весь блокмассив данных в хэш-таблице, которая отражает локальную память и производительность. В нашей реализации каждый процесс поддерживает локальную двухуровневую структуру данных. Первый уровень — это словарь его локальных ключей SFC, а второй уровень — массив смежных блоков в памяти. Поиск выполняется с использованием словаря, который преобразует ключ SFC в место в памяти. Словарь реализован как отсортированный массив и для всех наших тестовых примеров, он достаточно мал, чтобы поместиться в кеш, так что поиск ключа очень быстр. Альтернативной реализацией может быть использование хеш-таблицы. Эта структура данных уменьшает работу, необходимую для вычисления нового шаблона связи после миграции данных во время фазы балансировки нагрузки. Единственной информацией, необходимой для каждого процесса, является диапазон ключей (минимальный и максимальный ключи) других разделов. Эта информация передается, с помощью одной коммуникации «All-to-All» после переноса данных. Для всех рассмотренных примеров нелистовые узлы октодерева не используются в вычислениях, и поэтому они не хранятся.

Алгоритм измельчения-огрубления имеет две фазы: фазу согласования и фазу измельчения-укрупнения. Этот алгоритм выполняется на каждой фазе адаптации ядра.

Этап согласования: эта фаза алгоритма повторяется до тех пор, пока не выполнится условие прекращения. Блоки первоначально отмечаются для измельчения на основе того, лежат ли их центры в области, подлежащей измельчению. Блоки, которые не помечены для измельчения или вынуждены оставаться на их текущем уровне для удовлетворения критерия баланса, считаются подлежащими огрублению, если уровни их измельчения выше желаемого уровня. Следовательно, блок может находиться в одном из трех состояний — измельчаться, оставаться прежним или огрубляться. Состоянием по умолчанию

блока является огрубление. Любое изменение состояния блока должно быть сообщено его соседям и братьям. Каждый блок сохраняет текущие состояния своих соседей и обновляет эту информацию по мере выполнения алгоритма. Это производится с помощью локальных коммуникаций и может выполняться одновременно для различных областей сетки. Другими словами, область влияния блока измельчения или укрупнения ограничена. Здесь мы делаем два предположения, что сетка сбалансирована перед входом в фазу измельчения-укрупнения и что уровень измельчения блока может изменяться не более чем на единицу на протяжении фазы измельчения-укрупнения. Алгоритм аналогичен алгоритму распространения пульсаций в [**22**], в данном случае имеют место более низкие коммуникационные издержки из-за следующего: Соседи информируются только об изменении состояния.

Нет синхронизации между уровнями.

Кроме того, алгоритм в [**22**] не выполняет огрубление листовых блоков.

Каждый процесс имеет локальную очередь блоков, отмеченных для изменения. Единая итерация алгоритма согласования выполняет следующие шаги: Обработать записи в локальной очереди до тех пор, пока она не станет пустой.

Обновить состояния соседей и братьев и сестер, которые являются локальными (в том же процессе) Агрегировать сообщения для нелокальных соседей и братьев и сестер в очередях сообщений. Очередь сообщения для каждого соседа содержится в коммуникационном графе.

Сообщения, агрегированные процессом во время итерации, передаются с использованием коллективных операций [**15**] в MPI. Этот обмен состояниями может вызвать измельчение или огрубление большего количества блоков для каждого процесса. Новые блоки, отмеченные для измельчения/огрубления, добавляются в локальную очередь для обработки на следующей итерации. Алгоритм согласования заканчивается, когда локальные очереди и очереди сообщений каждого процесса пусты (состояние покоя, то есть все процессы стоят). Состояние покоя проверяется в конце каждой итерации алгоритма согласования. Это единственная точка синхронизации в этом алгоритме. Все процессы обмениваются целым числом, указывающим, занят он или нет (его локальная очередь и очереди сообщений пустые или нет). Это делается с помощью одного вызова функции MPI_Allreduce. Ниже приведен псевдокод для алгоритма.

Parallel Consensus Algorithm

```
procedure ParallelConsensus
     while ¬Terminate do
         while ¬q.empty() do
             n = q.pop()
             nbrs = blocks[n].nbrlist()
             sibs = blocks[n].slist()
             for all nbr nbrs do
                if nbr.is_local() then
                    local_update(n, nbr)
                else
                    aggregate_msg(nbr)
                end if
             end for
          for all sib ∈ sibs do
             if sib.is_local() then
                 local_update(n, sib)
             else
                aggregate_msg(sib)
             end if
          end for
      end while
      MPI_Neighbor alltoallw()
      MPI_alltoall(terminate)
   end while
end procedure
```

Существуют неитеративные версии алгоритма параллельного согласования [12, 20]. Однако вышеописанный алгоритм сходится быстро, поскольку измельчение и укрупнение являются локализованными операциями. Кроме того, поскольку процессы общаются только с их ближайшими соседями (по шаблону коммуникации обмена), количество сообщений и размер сообщений для каждого процесса очень малы. Единственной информацией, которую блок посылает своему соседу, является его состояние (одно целое число). С другой стороны, итеративная версия имеет меньшее количество итераций, но количество сообщений и размер сообщений для каждого процесса могут быть довольно высокими.

Этап измельчения-огрубления: это фаза, в которой происходит добавление и удаление блоков, а также обновление информации о соседстве. Новые блоки добавляются локально, а блоки, отмеченные для укрупнения, удаляют самих себя. Поскольку каждый блок знает о конечном решении своих соседних блоков, ключи SFC соседей

могут обновляться локально, манипулируя их текущими ключами (добавлением или удалением битов). На этом этапе коммуникаций нет.

Фаза вычисления по шаблону разностной схемы реализована, при параллельном выполнении операций вычислений и коммуникации для сокращения времени выполнения. Сообщения для граничных обменов агрегируются (упаковываются) и отправляются соседям. В ожидании граничной информации от удаленных соседей процессы выполняют локальные граничные обновления и вычисление шаблона разностной схемы внутренней части блока. Удаленные граничные области обновляются при поступлении сообщений от других вычислительных узлов. Это выгодно для AMR, когда блоки имеют хорошее отношение объема к поверхности, так что коммуникации выполняются совмещенно с локальными вычислениями.

2. Интеграция MiniAMR и OpenTS

Оригинальная функция main переписана на OpenTS в виде T-функции.

Реализованые некоторые MPI функции с учетом синтаксиса и семантики языка T++. Все базовые функции MPI были подключены из библиотеки Open MPI с помощью модуля OpenTS DMPI.

В T-функции main динамически осуществляется запуск нескольких задач (рабочих T-функций). Каждой рабочей T-функции передаются параметры командной строки.

Проведенная интеграция OpenTS с изначально ориентированным на MPI приложением дает возможность комбинировать уже существующие библиотеки с новыми адаптивными алгоритмами, используя при этом ядро T-системы в качестве балансировщика нагрузки верхнего уровня.

Для распараллеливания OpenTS использует MPI функции, которые берет из библиотеки MPI приложения (в нашем случае Open MPI), установленном на компьютере, где запускается приложение. Если Open MPI использует статический метод распараллеливания, то OpenTS запускает параллельно задачи, используя динамический метод распараллеливания, то есть сначала на главном узле вычислительной системы запускается основная задача, которая потом в соответствии с текстом программы запускает задачи на других вычислительных узлах.

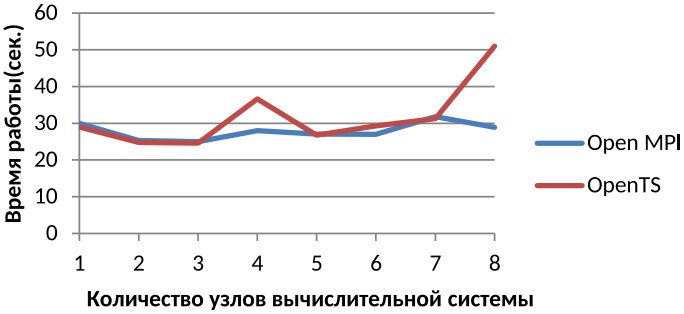

Рисунок 2. Тест «расширяющаяся сфера». Графики реального времени работы MiniAMR в среде Open MPI и MiniAMR, интегрированной с OpenTS

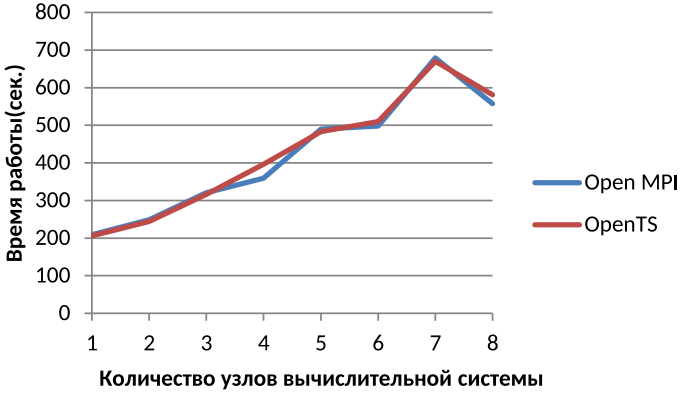

Рисунок 3. Тест «одна сфера, движущаяся по диагонали». Графики реального времени работы MiniAMR в среде Open MPI и MiniAMR, интегрированной с OpenTS

3. Сравнительные испытания OpenTS DMPI и Open MPI

Тесты входят в состав MiniAMR. Каждый тест запускается вызовом mpirun приложения MiniAMR с набором параметров. Подробное описание, как запускать каждый из тестов, находится в описании приложения.

Из графиков на рисунках 2, 3 и 4 видно, что OpenTS проигрывает по времени в случае, когда количество вычислительных узлов является

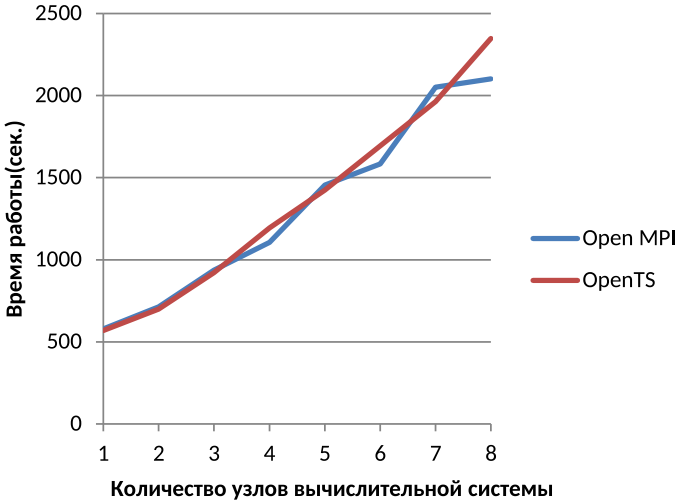

Рисунок 4. Тест «две движущиеся сферы». Графики реального времени, работы MiniAMR в среде Open MPI и MiniAMR, интегрированной с OpenTS

составным числом и не проигрывает, когда количество вычислительных узлов является простым числом. Это связано с тем, что в текущей версии OpenTS DMPI не реализована функция MPI_Comm_Split. В случае, когда количество узлов является простым числом, T- версия программы (OpenTS) показывает лучший результат по сравнению с MPI-версией программы. Это объясняется тем, что в этом случае вызов функции MPI_Comm_Split() не производит расщепление процессов на подгруппы.

Если в среде Open MPI «отключить» функцию MPI_Comm_Split(), то есть не создавать новый коммуникатор, а всегда использовать коммуникатор MPI_COMM_WORLD (рис. 5), то можно видеть, что MiniAMR ведет себя одинаково как в среде OpenTS, так и в среде Open MPI.

Из проведенных экспериментов можно сделать вывод о корректной работе системы для параллельного программирования OpenTS на рассмотренных тестах.

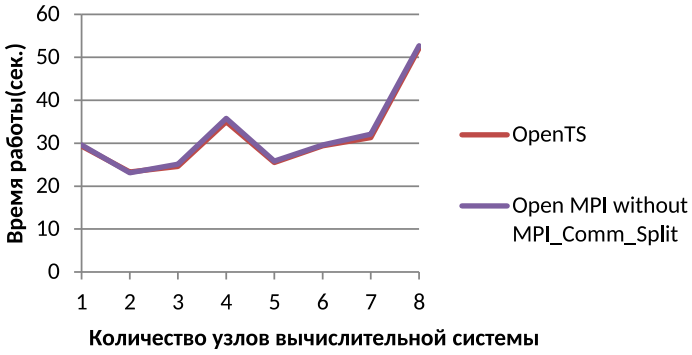

Рисунок 5. Тест «расширяющаяся сфера». Графики реального времени работы MiniAMR в среде Open MPI «с отключенной функцией MPI_Comm_Split()» и MiniAMR, интегрированной с OpenTS

Список литературы

[1] С. М. Абрамов, В. А. Васенин, Е. Е. Мамчиц, В. А. Роганов, А. Ф. Слепухин. «Динамическое распараллеливание программ на базе параллельной редукции графов. Архитектура программного обеспечения новой версии Т-системы», *Научная сессия МИФИ-2001*, Сборник научных трудов. Т. 2 (Москва, 22–26 января 2001 г.), с. 234. URL ↑361

[2] С. М. Абрамов, А. А. Кузнецов, В. А. Роганов. «Кроссплатформенная версия Т-системы с открытой архитектурой», *Труды Международной научной конференции "Параллельные вычислительные технологии (ПаВТ'2007)"*. Т. 1 (Челябинск, 29 января–2 февраля 2007 г.), Изд. ЮУрГУ, Челябинск, 2007, с. 115–121. ↑361

[3] С. М. Абрамов, А. А. Кузнецов, В. А. Роганов. «Кроссплатформенная версия Т-системы с открытой архитектурой», *Вычислительные методы и программирование*, **8**:1(2) (2007), с. 175–180. URL ↑361

[4] А. А. Кузнецов, В. А. Роганов. «Экспериментальная реализация отказоустойчивой версии системы OpenTS для платформы Windows CCS», *Труды Второй Международной научной конференции "Суперкомпьютерные системы и их применение (SSA'2008)"* (27–29 октября 2008 г., Минск), ОИПИ НАН Беларуси, Минск, 2008, с. 65–70. URL ↑361

[5] Е. А. Степанов. «Планирование в OpenTS — системе автоматического динамического распараллеливания», Информационные технологии и программирование, т. **2(14)**, МГИУ, М., 2005, с. 31–42. ↑361

[6] С. М. Абрамов, Г. И. Есин, И. М. Загоровский, Г. А. Матвеев, В. А. Роганов. «Принципы организации отказоустойчивых параллельных вычислений

для решения вычислительных задач и задач управления в Т-Системе с открытой архитектурой (OpenTS)», *Международная конференция "Программные системы: теория и приложения (PSTA-2006)"*, Сборник трудов конференции (23–28 октября 2006 г., г. Переславль-Залесский, Институт Программных Систем РАН), с. 257–264. URL ↑361

[7] V. Roganov, A. Slepuhin. “Distributed extension of the parallel graph reduction. GRACE: Compact and efficient dynamic parallelization technology for the *heterogeneous computing systems*”, International Conference on Parallel and Distributed Processing Techniques and Applications (June 25–28, 2001, Las Vegas, Nevada, USA). ↑361

[8] A. Moskovsky, V. Roganov, S. Abramov. “Parallelism granules aggregation with the T-system”, Parallel Computing Technologies: 9th International Conference, PaCT 2007 (Pereslavl-Zalessky, Russia, September 2007), Lecture Notes in Computer Science, vol. **4671**, ed. V. Malyshkin, Springer, Berlin etc., 2007, pp. 293–302. doi ↑361

[9] A. Moskovsky, V. Roganov, S. Abramov, A. Kuznetsov. “Variable reassignment in the T++ parallel programming panguage”, Parallel Computing Technologies: 9th International Conference, PaCT 2007 (Pereslavl-Zalessky, Russia, September 2007), Lecture Notes in Computer Science, vol. **4671**, ed. V. Malyshkin, Springer, Berlin etc., 2007, pp. 579–588. doi ↑361

[10] W. Bangerth, R. Hartmann, G. Kanschat. “deal.II — A general-purpose object-oriented finite element library”, *ACM Trans. Math. Softw.*, **33**:4 (2007), 24. doi ↑362

[11] I. Beichl, F. Sullivan. “Interleave in peace, or interleave in pieces”, *IEEE Comput. Sci. Eng.*, **5**:2 (1998), pp. 92–96. doi ↑366

[12] C. Burstedde, L. C. Wilcox, O. Ghattas. “p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees”, *SIAM Journal on Scientific Computing*, **33**:3 (2011), pp. 1103–1133. doi ↑371

[13] J. J. Camata, A. L. G. A. Coutinho. “Parallel linear octree meshing with immersed surfaces”, 22st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2010 (Petropolis, Brazil, October 27–30, 2010), 2010, pp. 151–158. doi ↑362

[14] C. Gotsman, M. Lindenbaum. “On the metric properties of discrete space-filling curves”, *IEEE Transactions on Image Processing*, **5**:5 (1996), pp. 794–797. doi ↑366

[15] T. Hoefler, J. L. Traff. “Sparse collective operations for MPI”, *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, IPDPS ’09 (Washington, DC, USA, 2009), IEEE Computer Society, pp. 1–8. doi ↑370

[16] M. Parashar, J. C. Browne. “Systems engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement”, *Structured Adaptive Mesh*

Refinement (SAMR) Grid Methods, IMA Volumes in Mathematics and its Applications, vol. **117**, Springer-Verlag, 1997, pp. 1–18. doi ↑369

[17] S. Popinet. “Gerris: A Tree-based adaptive solver for the incompressible Euler equations in complex geometries”, *J. Comput. Phys.*, **190**:2 (2003), pp. 572–600. doi ↑362

[18] D. Rosenberg, A. Fournier, P. Fischer, A. Pouquet. “Geophysical-astrophysical spectral-element adaptive refinement (GASpAR): Object-oriented h-adaptive fluid dynamics simulation”, *J. Comput. Phys.*, **215**:1 (2006), pp. 59–80. doi ↑362

[19] H. Sagan, *Space-filling curves*, Universitext, Springer-Verlag, New York, 1994. doi ↑366

[20] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, G. Biros. “Dendro: Parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees”, *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC ’08 (Piscataway, NJ, USA, 2008), IEEE Press, 2008, pp. 18:1–18:12. doi ↑362,371

[21] J. R. Stewart, H. C. Edwards. “A framework approach for developing parallel adaptive multiphysics applications”, *Finite Elem. Anal. Des.*, **40**:12 (2004), pp. 1599–1617. doi ↑362

[22] O. D. R. Tu Tiankai, G. Omar. “Scalable parallel octree meshing for terascale applications”, *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC’05 (Washington, DC, USA, 2005), IEEE Computer Society, 2005, pp. 4. URL doi ↑362,370

[23] M. O. Neal, T. Belytschko. “Explicit-explicit subcycling with non-integer time step ratios for structural dynamic systems”, *Computers & Structures*, **31**:6 (1989), pp. 871–880. doi ↑363

[24] A. Sasidharan, M. Snir. *MiniAMR — A miniapp for Adaptive Mesh Refinement*, 2016. URL ↑362

[25] *MiniAMR Adaptive Mesh Refinement (AMR) Mini-App.* URL ↑362

Поступила в редакцию 3.07.2018
Переработана 03.12.2018
Опубликована 17.12.2018

Рекомендовал к публикации *д.ф.-м.н. С. М. Абрамов*

Пример ссылки на эту публикацию:

В. А. Роганов, В. И. Осипов, Г. А. Матвеев. «Интеграция пакета Mantevo MiniAMR и OpenTS. Сравнение OpenTS DMPI и Open MPI». *Программные системы: теория и приложения*, 2018, **9**:4(39), с. 361–381.

10.25209/2079-3316-2018-9-4-361-381

http://psta.psiras.ru/read/psta2018_4_361-381.pdf

Об авторах:

Владимир Александрович Роганов

Научный сотрудник ИПС им. А.К. Айламазяна РАН. Разработчик современных версий Т-системы, ведущий разработчик системы OpenTS. Принимал активное участие в суперкомпьютерных проектах Союзного государства России и Беларуси, в том числе в проектах «СКИФ» и «СКИФ-ГРИД»

0000-0002-5500-0804

e-mail: var@pereslavl.ru

Валерий Иванович Осипов

К.ф.-м.н., старший научный сотрудник ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси.

0000-0002-8983-3791

e-mail: val@pereslavl.ru

Герман Анатольевич Матвеев

Научный сотрудник ИЦМС ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси.

0000-0002-2988-1791

e-mail: gera@prime.botik.ru

CSCSTI 50.41.23
UDC 519.682.3, 519.245

Vladimir Roganov, Valerii Osipov, German Matveyev. *Integration of Mantevo MiniAMR and OpenTS. Comparison of OpenTS DMPI and Open MPI.*

Abstract. Mantevo MiniAMR is a miniapp for adaptive mesh refinement (AMR) that mimics commonly found workloads and communication patterns in AMR applications, compact proxy for octree-based AMR. Adaptive Mesh Refinement is a frequently used technique for efficiently solving partial differential equations (PDEs): A finer mesh is used in regions that require a higher resolution (e.g., because of turbulence), and a coarse mesh is used to cover less sensitive regions. The authors describe integration of MiniAMR and OpenTS parallel programming system. There is performed comparison between OpenTS DMPI and Open MPI. (*In Russian*).

Key words and phrases: dynamic parallelization, T-system with an open architecture, OpenTS, T++ programming language, adaptive algorithm, adaptive mesh refinement.

2010 *Mathematics Subject Classification*: 65Y05; 68W10, 68W15

References

[1] S. M. Abramov, V. A. Vasenin, Ye. Ye. Mamchits, V. A. Roganov, A. F. Slepukhin. “Dynamic parallelization of programs based on parallel reduction of graphs. Software architecture of the new version of the T-system”, *Nauchnaya sessiya MIFI-2001*, Sbornik nauchnykh trudov. V. 2 (Moskva, 22–26 yanvarya 2001 g.), pp. 234 (in Russian). URL ↑361

[2] S. M. Abramov, A. A. Kuznetsov, V. A. Roganov. “Cross-platform version of the T-system with an open architecture”, *Trudy Mezhdunarodnoy nauchnoy konferentsii "Parallel’nyye vychislitel’nyye tekhnologii (PaVT’2007)"*. V. 1 (Chelyabinsk, 29 yanvarya–2 fevralya 2007 g.), Izd. YuUrGU, Chelyabinsk, 2007, pp. 115–121 (in Russian). ↑361

[3] S. M. Abramov, A. A. Kuznetsov, V. A. Roganov. “Cross-platform version of the T-system with an open architecture”, *Vychislitel’nyye metody i programmirovaniye*, **8**:1(2) (2007), pp. 175–180 (in Russian). URL ↑361

[4] A. A. Kuznetsov, V. A. Roganov. “An experimental implementation of OpenTS system fault-tolerant version for Windows CCS”, *Trudy Vtoroy Mezhdunarodnoy nauchnoy konferentsii "Superkomp’yuternyye sistemy i ikh primeneniye (SSA’2008)"* (27–29 oktyabrya 2008 g., Minsk), OIPI NAN Belarusi, Minsk, 2008, pp. 65–70 (in Russian). URL ↑361

© V. A. Roganov, V. I. Osipov, G. A. Matveyev, 2018
© Ailamazyan Program Systems Institute of RAS, 2018
© Program Systems: Theory and Applications (design), 2018
10.25209/2079-3316-2018-9-4-361-381

[5] Ye. A. Stepanov. “Scheduling in OpenTS — system with automatic dynamic parallelization”, Informatsionnyye tekhnologii i programmirovaniye, vol. **2(14)**, MGIU, M., 2005, pp. 31–42 (in Russian). ↑361

[6] S. M. Abramov, G. I. Yesin, I. M. Zagorovskiy, G. A. Matveyev, V. A. Roganov. “Principles of organization of the failure tolerance parallel computations for the computational and control tasks in the T-system with the open architecture (OpenTS)”, *Mezhdunarodnaya konferentsiya "Programmnyye sistemy: teoriya i prilozheniya (PSTA-2006)"*, Sbornik trudov konferentsii (23–28 oktyabrya 2006 g., g. Pereslavl’-Zalesskiy, Institut Programmnykh Sistem RAN), pp. 257–264 (in Russian). URL ↑361

[7] V. Roganov, A. Slepuhin. “Distributed extension of the parallel graph reduction. GRACE: Compact and efficient dynamic parallelization technology for the *heterogeneous computing systems*”, International Conference on Parallel and Distributed Processing Techniques and Applications (June 25–28, 2001, Las Vegas, Nevada, USA). ↑361

[8] A. Moskovsky, V. Roganov, S. Abramov. “Parallelism granules aggregation with the T-system”, Parallel Computing Technologies: 9th International Conference, PaCT 2007 (Pereslavl-Zalessky, Russia, September 2007), Lecture Notes in Computer Science, vol. **4671**, ed. V. Malyshkin, Springer, Berlin etc., 2007, pp. 293–302. doi ↑361

[9] A. Moskovsky, V. Roganov, S. Abramov, A. Kuznetsov. “Variable reassignment in the T++ parallel programming panguage”, Parallel Computing Technologies: 9th International Conference, PaCT 2007 (Pereslavl-Zalessky, Russia, September 2007), Lecture Notes in Computer Science, vol. **4671**, ed. V. Malyshkin, Springer, Berlin etc., 2007, pp. 579–588. doi ↑361

[10] W. Bangerth, R. Hartmann, G. Kanschat. “deal.II — A general-purpose object-oriented finite element library”, *ACM Trans. Math. Softw.*, **33**:4 (2007), 24. doi ↑362

[11] I. Beichl, F. Sullivan. “Interleave in peace, or interleave in pieces”, *IEEE Comput. Sci. Eng.*, **5**:2 (1998), pp. 92–96. doi ↑366

[12] C. Burstedde, L. C. Wilcox, O. Ghattas. “p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees”, *SIAM Journal on Scientific Computing*, **33**:3 (2011), pp. 1103–1133. doi ↑371

[13] J. J. Camata, A. L. G. A. Coutinho. “Parallel linear octree meshing with immersed surfaces”, 22st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2010 (Petropolis, Brazil, October 27–30, 2010), 2010, pp. 151–158. doi ↑362

[14] C. Gotsman, M. Lindenbaum. “On the metric properties of discrete space-filling curves”, *IEEE Transactions on Image Processing*, **5**:5 (1996), pp. 794–797. doi ↑366

[15] T. Hoefler, J. L. Traff. “Sparse collective operations for MPI”, *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, IPDPS ’09 (Washington, DC, USA, 2009), IEEE Computer Society, pp. 1–8. doi ↑370

[16] M. Parashar, J. C. Browne. “Systems engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement”, *Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, IMA Volumes in Mathematics and its Applications, vol. **117**, Springer-Verlag, 1997, pp. 1–18. doi ↑369

[17] S. Popinet. "Gerris: A Tree-based adaptive solver for the incompressible Euler equations in complex geometries", *J. Comput. Phys.*, **190**:2 (2003), pp. 572–600. ↑362

[18] D. Rosenberg, A. Fournier, P. Fischer, A. Pouquet. "Geophysical-astrophysical spectral-element adaptive refinement (GASpAR): Object-oriented *h*-adaptive fluid dynamics simulation", *J. Comput. Phys.*, **215**:1 (2006), pp. 59–80. ↑362

[19] H. Sagan, *Space-filling curves*, Universitext, Springer-Verlag, New York, 1994. ↑366

[20] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, G. Biros. "Dendro: Parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees", *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08 (Piscataway, NJ, USA, 2008), IEEE Press, 2008, pp. 18:1–18:12. ↑362,371

[21] J. R. Stewart, H. C. Edwards. "A framework approach for developing parallel adaptive multiphysics applications", *Finite Elem. Anal. Des.*, **40**:12 (2004), pp. 1599–1617. ↑362

[22] O. D. R. Tu Tiankai, G. Omar. "Scalable parallel octree meshing for terascale applications", *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC'05 (Washington, DC, USA, 2005), IEEE Computer Society, 2005, pp. 4. ↑362,370

[23] M. O. Neal, T. Belytschko. "Explicit-explicit subcycling with non-integer time step ratios for structural dynamic systems", *Computers & Structures*, **31**:6 (1989), pp. 871–880. ↑363

[24] A. Sasidharan, M. Snir. *MiniAMR — A miniapp for Adaptive Mesh Refinement*, 2016. ↑362

[25] *MiniAMR Adaptive Mesh Refinement (AMR) Mini-App.* ↑362

Sample citation of this publication:

Vladimir Roganov, Valerii Osipov, German Matveyev. "Integration of Mantevo MiniAMR and OpenTS. Comparison of OpenTS DMPI and Open MPI". *Program Systems: Theory and Applications*, 2018, **9**:4(39), pp. 361–381. (*In Russian*).

10.25209/2079-3316-2018-9-4-361-381

`http://psta.psiras.ru/read/psta2018_4_361-381.pdf`