Вестник УГАТУ
Уфа : УГАТУ, 2012 Т. 16, № 6 (51). С. 149–157

УДК 519.7

А. Н. Сальников, А. И. Майсурадзе, Д. Ю. Андреев, Г. А. Костин

КЛАСТЕРИЗАЦИЯ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ КОММУНИКАЦИОННОЙ СРЕДЫ МНОГОПРОЦЕССОРНЫХ СИСТЕМ: ЕДИНИЦЫ АНАЛИЗА, ИССЛЕДОВАНИЕ МЕТОДОВ, ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ

Число процессорных элементов в суперкомпьютерах постоянно растет, что приводит к росту сложности организации коммуникационной среды. С целью минимизации потерь производительности от задержек в коммуникационной среде и облегчения понимания особенностей поведения коммуникационной среды применяются различные подходы к сбору статистической информации о передаче сообщений и последующему анализу собранной информации. Рассматривается подход, основанный на измерении задержек при передаче сообщений с помощью специальных тестов. В ходе исполнения тесты создают огромный объем данных, который невозможно проанализировать вручную. Рассматриваются задачи сжатия, визуализации, автоматического анализа таких данных, которые предлагается решать с помощью методов кластеризации. Результаты анализа предлагается использовать при создании и оптимизации параллельных программ, для обнаружения аномалий коммуникационной среды и выявления проблемных узлов вычислительного кластера. В работе рассмотрено применение методов дивизизной, агломеративной кластеризации и локальной последовательной кластеризации к данным, полученным с суперкомпьютеров: «Ломоносов-МГУ», «Скиф-МГУ», «Bluegene-МГУ». *Интеллектуальный анализ данных*; *кластерный анализ*; *большой объем данных*; *синтетические MPI-тесты*; *коммуникационная среда многопроцессорных систем*

ВВЕДЕНИЕ

Современные тенденции развития многопроцессорной техники, предназначенной для решения задач математического моделирования, направлены на увеличение числа процессорных элементов, что, в свою очередь, опосредованно влечет увеличение сложности организации коммуникаций между этими процессорными элементами. На сегодняшний день в СНГ (согласно [1]) зарегистрировано 13 многопроцессорных систем с общим числом процессорных элементов больше 1000. В мире (согласно [2]) насчитывается 291 система с числом процессоров от 4 до 8 тысяч, а также 147 систем, где число процессоров превышает 8 тысяч. Сложности возникают уже при анализе коммуникационной среды для тысячи процессоров. С увеличением числа процессоров сложности обработки и анализа растут многократно.

Наиболее популярной технологией программирования для указанных систем является Message Passing Interface (MPI) [3]. Однако наблюдаются серьезные проблемы при использовании реализаций MPI на машинах с более чем 5000 процессоров (например, см. [4]), часть из них связана с неэффективностью алгоритмов, заложенных в реализации MPI, на большом числе процессоров. Архитектура и масштаб коммуникационной среды могут сильно повлиять на различие во времени передачи сообщений между процессами. Чем больше процессоров в системе, тем большее время обычно требуется на передачу данных от одного MPI-процесса к другому, и тем большая неоднородность наблюдается в величинах задержек при передаче сообщений. Из-за большого числа процессоров в большинстве случаев коммуникационная среда многопроцессорной системы неоднородна, разные пары процессоров могут взаимодействовать с разной скоростью. Это приводит к сложности понимания поведения коммуникационной среды, к сложности анализа поведения параллельной программы, к потере производительности и масштабируемости параллельного кода.

С целью минимизации потерь и облегчения понимания сути происходящих явлений применяются некоторые ухищрения. Первый подход основан на измерении задержек, информация о которых получается после сбора статистики по трассам, получаемым после выполнения параллельной программы. Изучение трасс позволит выдать рекомендации о том, что нужно изменить в коде, чтобы сократить время исполнения программы. К программным средствам, которые придерживаются такого подхода, можно отнести, например: Scalasca [5], Vampir [6], Intel Trace Analiser, Tau [7]. Второй подход основан на измерении задержек с помощью некоторых

Контактная информация: 8(495)939-17-90

Работа частично поддержана грантами РФФИ: 12-07-31088, 12-07-31229, 11-07-00756, 11-07-00614

специальных тестов заранее, до начала выполнения параллельной программы и последующем использовании полученных результатов тестирования при исполнении программы. К таким системам относятся средства синтетического тестирования MPI: Intel MPI Benchmark [8], MPI-bench-suit [9], SkaMPI [10], MPIBlib [11], а также разрабатываемое на Факультете ВМК МГУ имени М. В. Ломоносова программное средство network_test2 [12], которое является частью проекта PARUS [13, 14]. Результаты тестирования могут быть использованы не только разработчиками при создании и оптимизации параллельных программ, но и системными администраторами для обнаружения аномалий и выявления проблемных узлов вычислительного кластера.

В настоящий момент оба рассматриваемых подхода в процессе исполнения своих программных частей создают огромный объем данных разнообразного характера, которые невозможно проанализировать вручную. Требуется разработка методов сжатия, визуализации, автоматического анализа таких данных для выявления аномалий и закономерностей. Указанные задачи предлагается решать с помощью алгоритмов кластеризации [15].

В данной работе методы кластеризации применяются для различных объектов. Сначала в качестве объектов рассматриваются отдельные процессы, причем игнорируется различие между процессами-источниками и процессами-приемниками. В следующем разделе объектами станут пары процессов (источник, приемник). Наконец, в качестве объектов будут рассмотрены тройки (источник, приемник, длина сообщения).

ИСХОДНЫЕ ДАННЫЕ

Данная работа посвящена автоматическому анализу данных, собранных в рамках второго подхода как результат тестирования коммуникаций многопроцессорной системы с помощью программного средства network_test2.

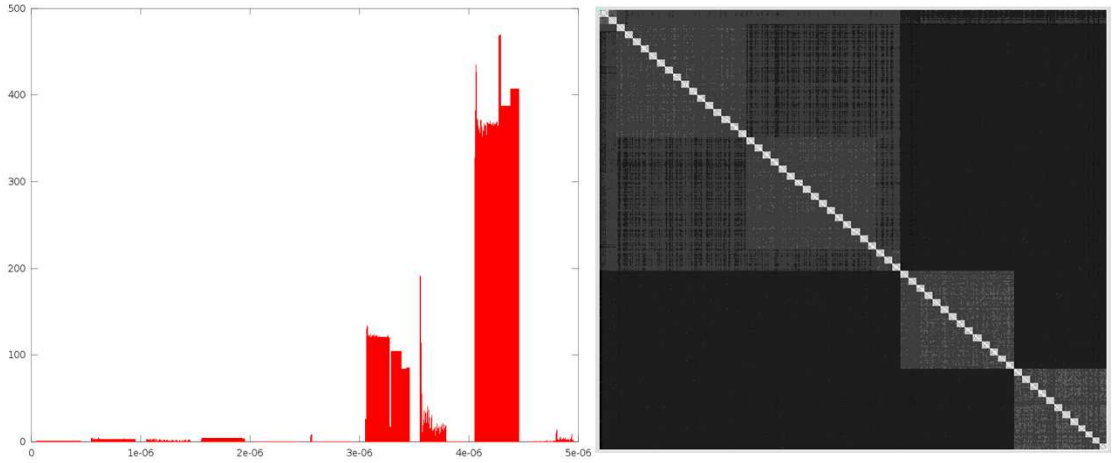
Средство network_test2позволяет получить данные по задержкам для конкретных пар процессов при разных длинах сообщений. Результаты таких тестов сохраняются в виде набора матриц задержек для каждой упорядоченной пары процессов. Первый процесс пары – источник, второй – приемник. Для каждой исследованной длины сообщения сохраняется своя матрица. Элемент каждой такой матрицы содержит статистически отобранную информацию о распределении задержки между выбранными процессами. В простейшем случае элемент содержит среднюю величину задержки. В случае подхода на основе сбора трасс такие матрицы называются коммуникационными. Другими словами, анализируемые данные представляют собой трехмерное аналитическое пространство (трехмерный куб данных), определяемое двумя измерениями процессов (источников и приемников) и измерением длин сообщений.

Результатом работы программы network_ test2 являются несколько файлов в формате NetCDF [16], которые для каждой упорядоченной пары процессов и каждой длины сообщения хранят следующие числовые характеристики эмпирического распределения задержки: минимальные значения, медианы, средние значения и стандартные отклонения задержек. Каждый файл содержит множество матриц, где каждая матрица соответствует своей длине сообщения. В позиции i, j матрицы содержится числовая характеристика задержки при передаче сообщения от процесса с номером i к процессу с номером j.

К сожалению, для достаточно большого числа процессов данные результатов теста будут иметь большой объем, так как размер матриц пропорционален квадрату числа процессов, участвовавших в тесте. Для размеров тестов с числом процессов, сопоставимым с суммарным числом ядер многопроцессорного комплекса, имеем объемы, которыми достаточно сложно оперировать. Например, система BlueGene/P на факультете ВМК имеет 8 192 процессорных ядер [17], что дает 67 108 864 чисел с плавающей точкой, то есть 512Мб занимает только одна коммуникационная матрица (8 байт на элемент).

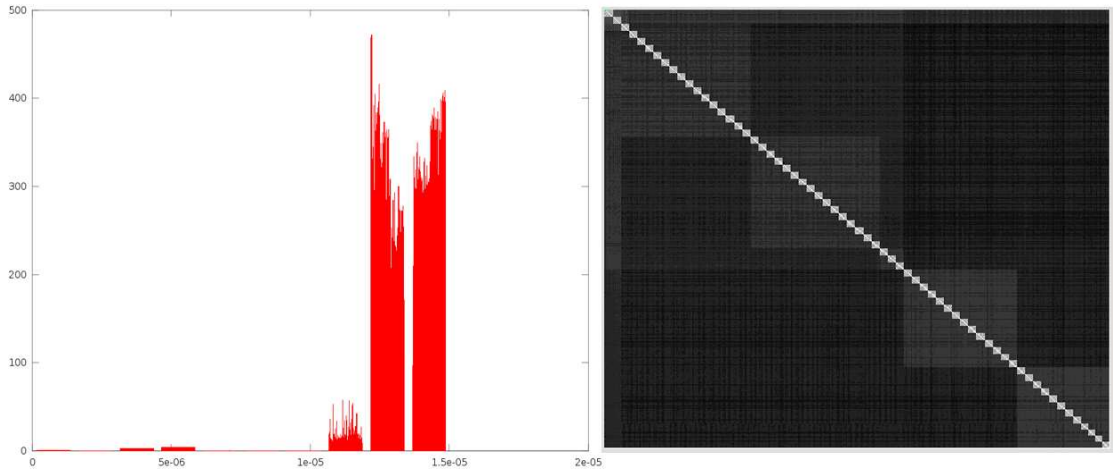
Представляется естественным попытаться сократить размер указанных трехмерных описаний. Таким образом, ставится задача агрегирования и последующей аппроксимации исходных данных. Существует множество подходов к решению указанной задачи, однако в данной работе из-за большого объема данных авторы вынуждены отдать предпочтение быстрым алгоритмам.

Одним из путей сокращения описания является нахождение групп схожих значений в результатах тестирования. В работе для решения данной задачи применяется кластерный анализ.



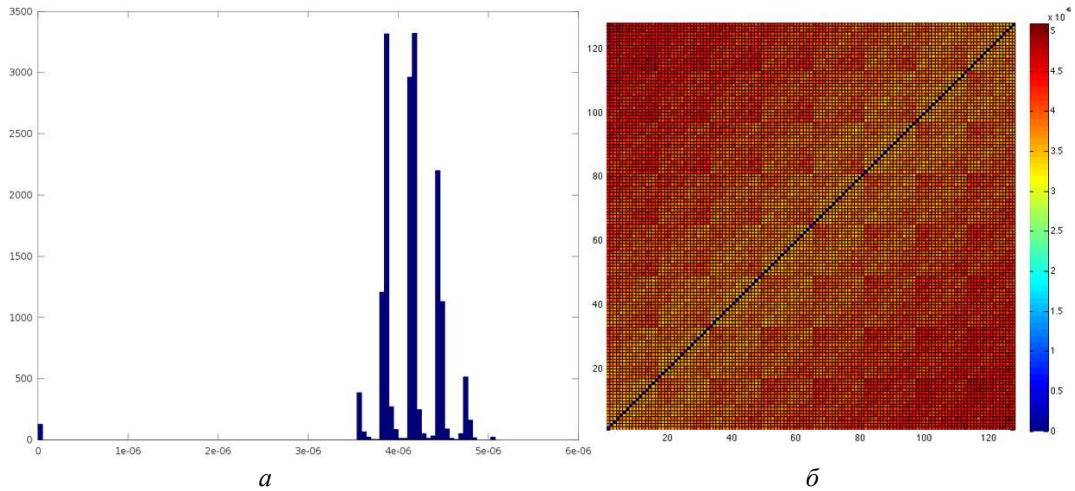
а *б*

Рис. 1. Данные задержек для «Ломоносова», 500 процессов, длина сообщения 0 Байт: *а* – гистограмма распределения значений задержек; *б* – картограмма матрицы задержек



а *б*

Рис. 2. Данные задержек для «Ломоносова», 500 процессов, длина сообщения 8000 Байт: *а* – гистограмма распределения значений задержек; *б* – картограмма матрицы задержек



а *б*

Рис. 3. Данные задержек для Bluegene/P, 128 процессов, длина сообщения 0 Байт: *а* – гистограмма распределения значений задержек; *б* – картограмма матрицы задержек

ВИЗУАЛЬНЫЙ АНАЛИЗ КОММУНИКАЦИОННЫХ МАТРИЦ

Одним из простейших способов визуализации одной коммуникационной матрицы является ее картограмма, в которой каждый элемент матрицы окрашивается в оттенок, соответствующий его значению. На рис. 1, *а*, *б* представлены картограммы матриц задержек для 500 процессов в одной и той же системе «Ломоносов» при разных длинах сообщения. На рис. 3, *б* представлена картограмма матрицы задержек для 128 процессов в системе BlueGene/P при длине сообщения 0 в режиме one_to_one.

В ходе визуального анализа картограмм результатов тестирования было замечено следующее. Величины задержек на показываемом изображении группируются в области. Причем эти области остаются устойчивыми со сменой длины сообщения в тесте. Форма областей на картограмме зависит от особенностей архитектуры коммуникационной среды. Это видно при сравнении рис. 1, *а* и *б* с рис. 3, *б*.

Из анализа картограмм можно сделать предположение, что элементы каждой отдельной матрицы хорошо группируются по близости значений, что иллюстрируется гистограммой значений задержек (рис. 1, *а*, 2, *а*, 3, *а*).

Видно, что структура гистограммы устойчива при изменении длины сообщения – см. рис. 1, *а* и 2, *а*. Более того, выделенные группы в матрицах для идущих подряд длин сообщения будут попадать на одни и те же пары MPI-процессов. Существенное изменение областей при увеличении длины сообщения может свидетельствовать о возможной смене алгоритма маршрутизации для конкретной длины сообщения, а также об исчерпании размеров кэшей, используемых для передачи данных.

АГЛОМЕРАТИВНАЯ КЛАСТЕРИЗАЦИЯ ОТДЕЛЬНЫХ ПРОЦЕССОВ

В силу архитектурных особенностей, процессы с точки зрения задержек при обмене сообщениями между процессорами в многопроцессорных вычислительных системах часто тяготеют к иерархической структуре. Так, процессы, находящиеся на разных ядрах одного процессора, будут давать очень схожие результаты тестирования, образуя общий кластер; процессы, распределенные по разным процессорам, но в пределах одной платы, будут также походить друг на друга. Тем самым образуется логическая иерархия процессов: процессы одного процессора входят в группу процессоров одной платы.

В силу тех же архитектурных особенностей, задержки для фиксированной длины сообщений тяготеют к группам, которые можно линейно упорядочить. Задержки между ядрами одного процессора для всех процессоров будут похожи. Задержки между ядрами разных процессоров одной платы будут похожи и превосходить задержки между ядрами одного процессора.

Поскольку коммуникационная среда многопроцессорных комплексов тяготеет к иерархической структуре, выбор алгоритма группировки процессов был осуществлен среди алгоритмов иерархической кластеризации, которая позволяет объединять элементы не просто в группы, а в иерархию (элементы становятся листьями дерева) [15].

Иерархическая кластеризация делится на два подхода: агломеративная кластеризация и дивизивная кластеризация. Агломеративный подход, кластеризация снизу-вверх, объединяет отдельные элементы в группы, начиная с множества отдельных элементов. Дивизивный подход, кластеризация сверху-вниз, наоборот, разбивает большие кластеры на кластеры поменьше, пока не дойдет до отдельных элементов.

Алгоритмы иерархической кластеризации работают с матрицей попарных расстояний. Следовательно, нужна метрика (симметричная, неравенство треугольника выполнено) или хотя бы функция расстояния (симметричная, но с нарушением неравенства треугольника) на парах элементов.

В некоторых протоколах маршрутизации в сети в качестве расстояния используется задержка при передаче пакета. Таким образом, возникло предположение, что матрицу задержек для определенной длины можно использовать в качестве матрицы расстояний. Матрицы задержек для исследуемых в работе многопроцессорных комплексов достаточно симметричны, т. е. отличие между (i, j) и (j, i) элементами совсем невелико. Например, для системы «Ломоносов» при длине сообщения более 1000 несимметричность составляет менее 5 %. Хотя свойство симметричности, вообще говоря, неверно, например, для GRID-систем, в данной работе, считая, что основной исследуемый объект – вычислительные кластеры, авторы сочли возможным симметризовать матрицы. Таким образом, различие между процессами-источниками и процессами-приемниками в данном разделе игнорируется.

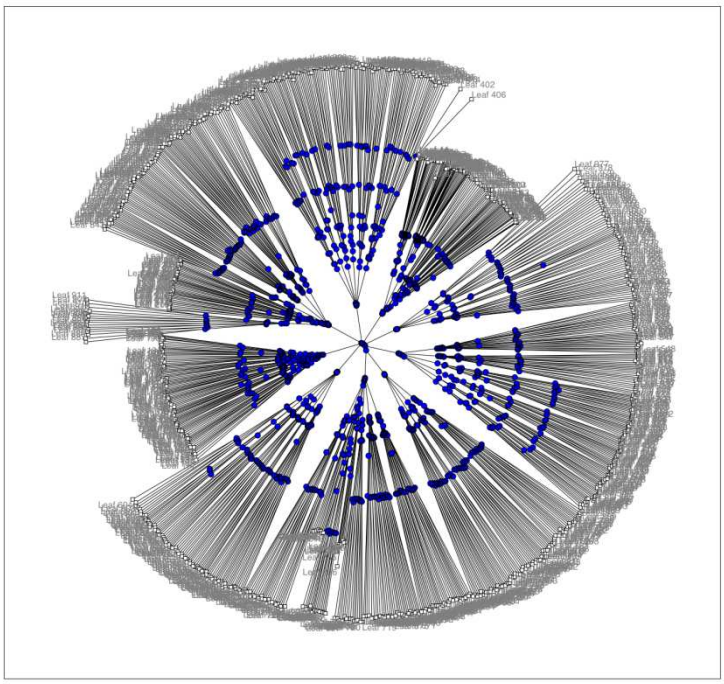
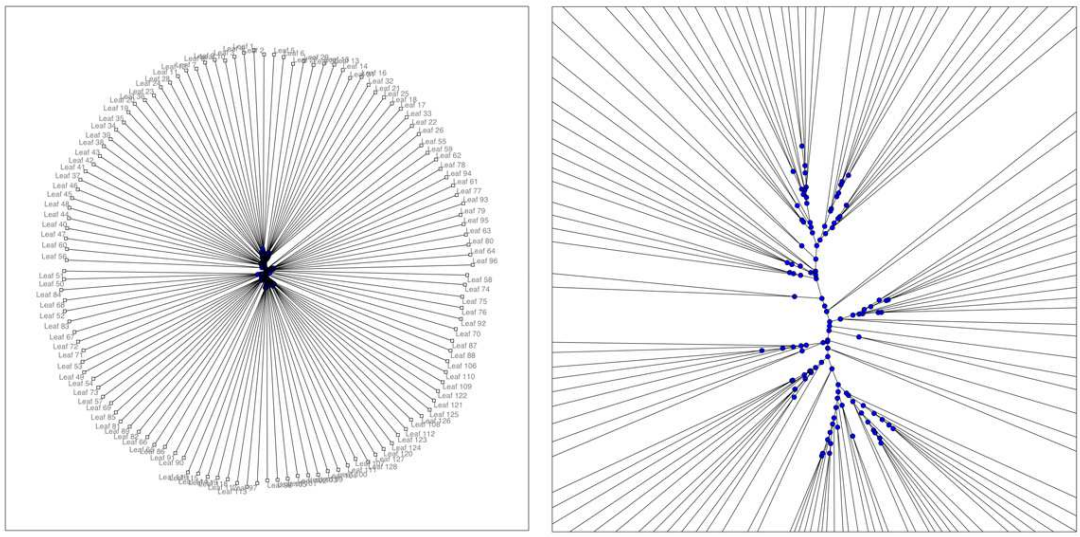


Рис. 4. СКИФ-МГУ «Чебышев» 1000 процессов, результат Neighbor-Joining



а *б*

Рис. 5. Bluegene/P 128 процессов: *а* – результат Neighbor-Joining, *б* – увеличенный вариант

Известно, что методы агломеративной кластеризации обычно дают лучшие результаты, чем методы дивизивной кластеризации. Однако последние требуют меньше трудозатрат. Поскольку число отдельных процессов не очень велико (тысячи), то в данном разделе используем агломеративную кластеризацию.

Напомним общую схему агломеративной кластеризации. Сначала каждый элемент является кластером. На каждой итерации алгоритма пара самых близких кластеров U и V образует новый кластер W. После этого приходится пересчитать расстояния от всех остальных текущих кластеров до вновь образованного. Отличие различных алгоритмов агломеративной кластеризации состоит в том, как вводится расстояние между новым кластером W и остальными кластерами. Лансом и Уильямсом была предложена

универсальная формула, способная описать практически все популярные способы определить расстояние между кластерами [18].

Если допустить дополнительные трудозатраты, то можно использовать Neighbor-Joining – популярный в биоинформатике метод иерархической кластеризации, используемый для построения филогенетических деревьев (phylogenetic tree) [19]. Данные деревья отображают близость организмов по всем признакам одновременно, показывая, насколько далеко ушли друг от друга в процессе эволюции организмы. Характерной особенностью алгоритма является способность сохранять расстояния от элементов до общего предка. Алгоритм Neighbor–Joining является представителем класса алгоритмов агломеративной кластеризации.

Работа алгоритма была опробована на данных результатов тестирования one_to_one многопроцессорных комплексов СКИФ-МГУ, MVS50k, BlueGene/P, Regatta. В качестве матрицы расстояний брались медианы средних величин задержек. Из результатов применения метода Neighbor-Joiningвидно, что хорошо определяется структура многопроцессорных комплексов с иерархической архитектурой, таких как СКИФ-МГУ [20] (рис. 4), MVS50k, MVS100k. Для таких систем, как BlueGene/P [17] с топологией трехмерного тора, архитектуру не удается полноценно восстановить с помощью иерархической кластеризации процессов (рис. 5), так как деревья ее не описывают.

ДИВИЗИВНАЯ КЛАСТЕРИЗАЦИЯ СВЯЗЕЙ МЕЖДУ ПРОЦЕССАМИ

В предыдущем разделе кластеризуемыми элементами были отдельные процессы. Такая кластеризация помогает понять архитектуру некоторых многопроцессорных комплексов. Однако основной задачей в работе является анализ и аппроксимация задержек между процессами. Поэтому в данном разделе авторы считают кластеризуемыми элементами упорядоченные пары процессов (i, j).

Будем искать среди множества связей (i, j) такие, что задержка при передаче с ростом длины сообщения растет похожим образом. В группы будут объединяться связи с похожей зависимостью задержки от длины сообщений. Так как связи между любыми двумя процессами на одном процессоре или на одной плате должны иметь схожую зависимость задержки от длины сообщений, то можно сделать предположение, что в результате кластеризации будут восстановлены реальные типы связей между процессорами.

С точки зрения представления исходной информации, исходный трехмерный куб мы рассматриваем теперь не как вектор матриц с компонентами $A^l = \{a_{ij}^l\}$, где l – длина сообщения, а как матрицу векторов с компонентами $B^{ij} = \{b_l^{ij}\}$. Каждая связь (i, j) описывается своим вектором B^{ij}.

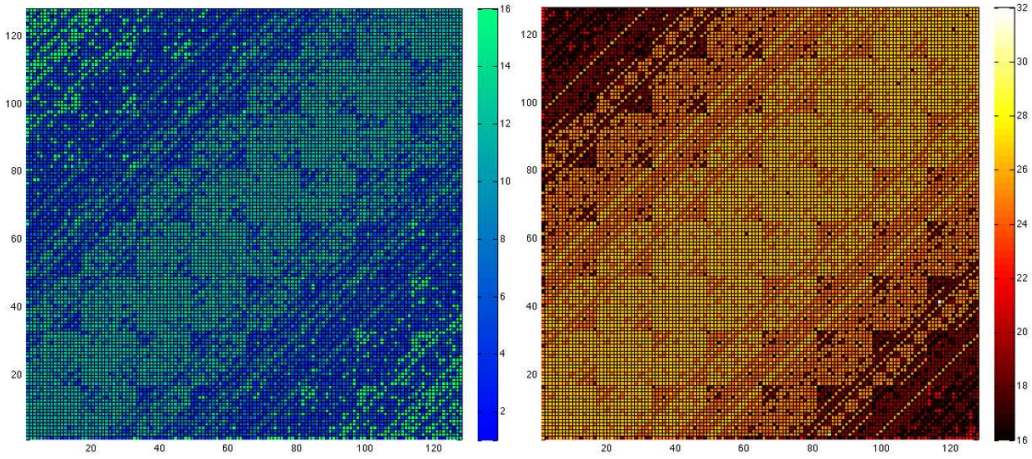
Для применения алгоритмов кластеризации необходимо ввести метрику над векторами B^{ij}. С ростом длины сообщений увеличивается и разброс в задержках, таким образом, во вводимом расстоянии необходимо учитывать дисперсию. Для удовлетворения вышеописанных требований была предложена следующая функция расстояния:

$$\rho^2((i,j),(p,q)) = \sum_{l=1}^{L}(a_{ij}(l) - a_{pq}(l))^2\left(\frac{1}{d_{ij}(l)} + \frac{1}{d_{pq}(l)}\right),$$

где $a_{ij}(l)$ есть среднее значение задержек между процессами i и j для длины l, $d_{ij}(l)$ дисперсия. Предложенная функция расстояния на связях позволяет использовать все длины пакетов одновременно и однородно, а не последовательно.

Алгоритмам агломеративной кластеризации необходимо знать расстояния между всеми объектами кластеризации. При n участвующих в тестировании процессах имеем n^2 разных векторов. Между этими векторами существует $\frac{n^2(n^2-1)}{2}$ расстояний. Так, для всего 128 процессов BlueGene/P имеем 16 384 вектора, 134 209 536 расстояний (верхний треугольник матрицы16 384 × 16 384). Это делает агломеративную кластеризацию практически неприменимой к системам с большим числом процессоров.

Преимущество дивизивной кластеризации заключается в том, что на каждом шаге задача разбивается на несколько независимых подзадач. Не требуется вычислять какие бы то ни было новые расстояния. Более того, при некоторых эвристических модификациях (изложены ниже), доказавших на практике свою применимость, большинство исходных расстояний вообще никогда не будут запрошены. Это свойство отлично подходит для больших объемов данных: нет необходимости рассчитывать расстояние между элементами разных кластеров, разбиение внутри одного кластера не зависит от разбиения в другом, что предоставляет возможность параллельной реализации алгоритма.



а *б*

Рис. 6. Bluegene/P 128 процессов, результаты кластеризации (одним цветом обозначены элементы, вошедшие в один кластер): *a* – агломеративная с расстоянием дальнего соседа; *b* – дивизивная

Если ищутся крупные кластеры, дивизивная кластеризация позволяет остановиться на этих крупных кластерах, не рассматривая мелких, тогда как агломеративной кластеризации необходимо объединить все элементы, вне зависимости от необходимого числа кластеров.

К сожалению, дивизивная кластеризация иногда проигрывает в качестве агломеративной. Дивизивная кластеризация разбивает кластер на два кластера, основываясь на близости элементов к диаметрально противоположным элементам кластера. Тогда два близких между собой элемента из «центра» кластера могут оказаться в разных кластерах на высоком уровне иерархии. И нет действия, чтобы снова их объединить. С другой стороны, при распределении элементов по двум кластерам вычисляется информация, которая используется при поиске следующих диаметральных элементов.

На основании вышеперечисленных фактов и с учетом описанной проблемы расстояний был предложен следующий алгоритм кластеризации.

0. Все элементы объединяем в одни кластер. Называем его текущим.

1. Приближенно выбираем диаметрально противоположные элементы текущего кластера. Берется произвольный элемент. Считаем расстояния от него до всех остальных элементов и ищем самый удаленный элемент r.

2. Для найденного элемента r ищем самый удаленный элемент s. Вычисленные расстояния от r до остальных элементов еще раз понадобятся на шаге 3.

3. Считаем r и s центрами новых кластеров. Распределяем элементы текущего кластера по двум новым кластерам. При этом элементы r и s становятся в своих кластерах первыми элементами для шага 1 и сразу находятся новые r' и r''.

4. Теперь с новыми кластерами можем работать независимо. Выполняем шаг 2 – находим s' и s''. Запоминаем оценку диаметра $\rho(r', s')$ и $\rho(r'', s'')$ для соответствующих кластеров.

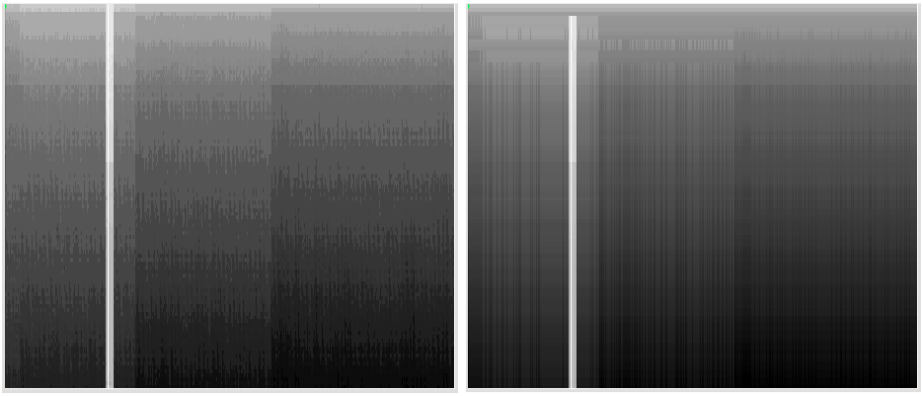
5. Повторяем с шага 3 независимо для каждого нового кластера. В первую очередь на обработку стараемся направить кластеры с большим диаметром. Однако нет необходимости строго соблюдать порядок.

6. Процесс можно остановить, когда разница в диаметрах кластеров станет существенно меньше предыдущего типичного значения разницы. Это критерий каменистой осыпи (Cattell, 1966) [21].

Отметим, что расстояния вычисляются только при необходимости (lazycomputing).

В экспериментах использовались алгоритмы агломеративной и дивизивной кластеризации на данных результатов тестирования one_to_one 128 процессов BlueGene/P. Расстояние между кластерами вычислялось по методу дальнего соседа.

На рис. 6 отображены результаты кластеризации: одинаковым цветом помечены элементы, вошедшие в один кластер. Как видно из иллюстраций, результаты работы предложенного алгоритма дивизивной кластеризации схожи с результатами агломеративной кластеризации.



а *б*

Рис. 7. «Ломоносов» 500 процессов, срез по длинам сообщений:
а – исходные данные; *б* – после кластеризации

ЛОКАЛЬНАЯ ПОСЛЕДОВАТЕЛЬНАЯ КЛАСТЕРИЗАЦИЯ ТРЕХМЕРНЫХ ЭЛЕМЕНТОВ

В данном разделе объектами кластеризации являются тройки (процесс-источник, процесс-приемник, длина сообщения). Таких объектов очень много (до сотен миллиардов на BlueGene/P), следовательно, приходится рассматривать сверхбыстрые алгоритмы.

Ранее для указанных данных Д. Ю. Андреевым и А. Н. Сальниковым был предложен алгоритм кластеризации [12]. Этот алгоритм преследовал две цели: уменьшить размер файла результатов тестирования, при этом сохранив хорошую скорость позиционного доступа к данным, и уменьшить расходы оперативной памяти за счет сокращения числа хранимых одновременно в памяти матриц. Алгоритм кластеризации из [12] основывается на предположении, что экономии памяти можно добиться, выделяя группы пар (*i*, *j*), которым соответствуют близкие по значению элементы нескольких коммуникационных матриц. Таким образом, алгоритм выделяет в кубе кластеры в форме параллелепипедов. Тогда для приближенного представления всех исходных матриц потребуется матрица с номерами групп, к которым принадлежат элементы (*i*, *j*), и описание каждой из групп. Данный алгоритм не требует больших размеров оперативной памяти, так как рассматривает матрицы строго последовательно. При правильно подобранных параметрах алгоритм способен сохранять структуру областей в матрицах, сжимая данные в несколько раз (сравните рис. 1, *а* и рис. 2, *б*).

В данной работе по отношению к работе [12] в алгоритм были внесены некоторые изменения. К сожалению, несмотря на хорошую скорость работы и нетребовательность к памяти, даже модифицированный алгоритм обладает рядом серьезных недостатков, например, теряет аномалии коммуникационной среды. Данный алгоритм старается выделять трехмерные кластеры, причем делает это с помощью попыток применить разбиение, полученное для предыдущей матрицы, к следующим по длине сообщения. Из-за этого, если число кластеров для следующей матрицы уменьшилось, но они имеют примерно такую же структуру, то алгоритм применит первое разбиение к этой матрице. Таким образом, алгоритм замечает изменения в сторону увеличения числа кластеров или изменения структуры, но не замечает уменьшения разбиения. Потеря особенностей в данных заметна при сравнении исходных и обработанных данных (рис. 7, *а*, *б*) при разных длинах сообщений.

При определенном наборе параметров данный алгоритм может достаточно долго работать. Здесь его было бы целесообразно распараллелить и выполнять на той же многопроцессорной системе, однако, в силу итеративности, алгоритм сложно разбить на независимые подзадачи для параллельного исполнения.

Работа алгоритма при разных параметрах исследовалась на результатах тестирования системы Ломоносов для 1000 процессов. Описанная версия алгоритма выложена на сайте проекта Parus [14].

Вследствие описанных проблем продолжаются работы по поиску алгоритма кластеризации, обеспечивающего лучшую аппроксимацию результатов тестирования.

ЗАКЛЮЧЕНИЕ

По результатам исследования можно сделать следующие выводы. Для анализа результатов MPI-тестирования коммуникационной среды многопроцессорных систем с большим числом процессоров наиболее перспективным алгоритмом кластеризации, по мнению авторов, на текущий момент видится применение приближенных алгоритмов дивизивной кластеризации. Данный класс алгоритмов при небольших потерях в точности по сравнению с точными агломеративными методами имеет большую скорость вычисления на экспериментальных данных, решает проблему хранения / вычисления большого объема расстояний. Также данный алгоритм предоставляет возможность параллельной реализации, что может быть существенно при больших размерах тестов.

Предложенная функция расстояния на связях позволила использовать все длины пакетов одновременно и однородно, а не последовательно.

СПИСОК ЛИТЕРАТУРЫ

1. Список 50 наиболее высокопроизводительных установок по тесту linpack в России и странах СНГ. [HTML] (http://top50.supercomputers.ru), дата обращения 30.05.2011.
2. Список 500 наиболее высокопроизводительных установок по тесту linpack в Мире [HTML] (http://www.top500.org), дата обращения 17.11.2011.
3. MPI: A Message-Passing Interface Standard [HTML] (http://www.mcs.anl.gov/research/projects/ mpi/mpi-standard/mpi-report-1.1/mpi-report.htm), дата обращения 17.11.2011.
4. MPI on a Million Processors / P. Balaji [et al.] // Recent Advances in Parallel Virtual Machine and Message Passing Interface. 2009. Vol. 5759. P. 20–30.
5. Recent Developments in the Scalasca Toolset / M. Geimer [et al.] // Tools for High Performance Computing 2009, Proc. of the 3rd Parallel Tools Workshop, Dresden, Germany, chapter 4, pages 39–51, Springer, 2010.
6. **Brunst H., Nagel W. E., Malony A. D.** A distributed performance analysisarchitecture for clusters // IEEE International Conference on Cluster Computing, Cluster 2003, Hong Kong, China. P. 73–81.
7. **Shende S., Malony A. D.** The TAU Parallel Performance System // International Journal of High Performance Computing Applications, SAGE Publications. 2006. № 20(2). P. 287–331.
8. Intel® MPI Benchmarks: Users Guide and Methodology Description [HTML] (http://www3.intel.com/ cd/software/products/asmo-na/eng/219848.html).
9. MPI-bench-suit: mpitest [HTML] (http:// parallel.ru/testmpi/mpitest.html).
10. **Reussner R., Hunzelman G**. Achieving Performance Portability with SKaMPI for High Performance MPI Programs // LNCS. 2001. № 2074. P. 841–850.
11. **Lastovetsky A., Rychkov V., O'Flynn M**. MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneous and Heterogeneous Clusters // LNCS. 2008. № 5205. P. 227–238.
12. **Сальников А. Н., Андреев Д. Ю.** Разработка инструментальных средств контроля за коммуникационной средой вычислительных кластеров с большим числом процессорных элементов // Параллельные вычисления и задачи управления: Тр. 5-й Международн. конф. PACO'2010. М.: Учреждение Российск. акад. наук Ин-т проблем управления им. В. А. Трапезникова, РАН, 2010. С. 1187–1208.
13. **Сальников А. Н., Андреев Д. Ю.** Исследование методов тестирования коммуникационной среды многопроцессорных систем // Науч. сервис в сети Интернет: решение больших задач: тр. Всероссийск. науч. конф. М.: МГУ им. М. В. Ломоносова, 2008. С. 237-240.
14. Проект Parus http://parus.sourceforge.net/
15. **Jain A., Murty M., Flynn P.** Data clustering: A review // ACM Computing Surveys. 1999. Vol. 31, № 3. P. 264–323.
16. NetCDF User's Guide [PDF] (http://www.unidata. ucar.edu/software/netcdf/docs/netcdf.pdf)
17. IBM System Blue Gene Solution: Blue Gene/P Application Development. An IBM Redbooks publication. [PDF] (http://www.redbooks.ibm.com/abstracts/ sg247287.html)
18. **Lance G. N., Willams W. T. A** general theory of classification sorting strategies. 1. hierarchical systems // Comp. J. 1967. № 9. P. 373–380.
19. **Saitou N., Nei, M.** The neighbor-joining method: A new method for reconstructing phylogenetic trees. Molecular Biology and Evolution. 1987. № 4(4). P. 406–425.
20. Описание вычислительного комплекса СКИФ-МГУ [HTML] (http://www.t-platforms.ru/ ru/skif_msu.php)
21. **Cattell. R. B**. Handbook of Multivariate Experimental Psychology. Chicago: Rand McNally, 1966.

ОБ АВТОРАХ

Сальников Алексей Николаевич, ст. науч. сотр. лаборатории вычислительн. электродинамики Московск. гос. ун-та им. М. В. Ломоносова. Дипл. математик, системный программист по прикладн. математике и информатике (ВМК МГУ, 2001). Канд .физ.-мат. наук по математическ. обеспечению вычислительн. машин, комплексов и комп. сетей (ВМК МГУ, 2007). Иссл. в обл. параллельных вычислений, биоинформатики, комуникационных сред вычислительн. кластеров.

Майсурадзе Арчил Иериевич, доц. каф. математическ. методов прогнозирования того же ун-та, доц. каф. интеллектуальн. систем Московск. физ.-техн. ин-та. Дипл. инженер-математик по прикладн. математике и физике, теории управления и исследованию операций (МФТИ, 1999). Канд. физ.-мат. наук по теор. основам информатики (ВЦ РАН, 2005) Иссл. в обл. машинного обучения, интеллектуального анализа данных, сложных сетей.

Андреев Дмитрий Юрьевич, асп. Вычислительн. центра Российск. Акад. наук им. А. А. Дородницына. Дипл. математик, системный программист по прикладн. математике и информатике (МГУ ВМК, 2010). Иссл. в обл. параллельных вычислений, коммуникационных сетей массивно-параллельных вычислительных систем, параллельного ввода-вывода.

Костин Григорий Александрович, студ. Московск. гос. ун-та им. М. В. Ломоносова. Иссл. в обл. машинного обучения.