ИССЛЕДОВАНИЕ АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ ДЛЯ АНАЛИЗА РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ КОММУНИКАЦИОННОЙ СРЕДЫ МНОГОПРОЦЕССОРНЫХ СИСТЕМ

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

Современные тенденции развития многопроцессорной техники, предназначенной для решения задач математического моделирования, направлены на увеличение числа процессорных элементов, что, в свою очередь, опосредованно, влечёт увеличение сложности организации коммуникаций между этими процессорными элементами. На сегодняшний день в СНГ (согласно списку top50[13] на состояние 30.05.2011) зарегистрировано 13 многопроцессорных систем с общим числом процессорных элементов больше 1000. В мире, согласно списку top500[14] от ноября 2011 года насчитывается 291 система с числом процессоров колеблющемся в диапазоне от 4 до 8 тысяч, а также 147 систем, где число процессоров больше чем 8 тысяч. Сложности возникают ещё при анализе коммуникационной среды для тысячи процессоров. С увеличением числа процессоров сложности растут многократно.

Наиболее популярной технологией программирования для таких систем является Message Passing Interface (MPI)[1]. Однако согласно статье[15] наблюдаются серьёзные проблемы при использовании MPI на машинах с более чем 5000 процессоров, которые связаны с неэффективностью алгоритмов заложенных в реализации MPI доступные на состояние 2009 года. Чем большее число процессоров в системе, тем большее время обычно требуется на передачу данных от одного MPI-процесса к другому, и тем большая неоднородность в величинах задержек при передаче сообщений будет наблюдаться. 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 в понимания 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[18], Vampir[19], Intel Trace Analiser, Tau[20]. Второй подход основан на изменении задержек с помощью некоторых специальных тестов заранее, до начала выполнения параллельной программы, а затем, при исполнении программы как-то использовать полученные результаты тестирования. К таким системам относятся Intel MPI Benchmark[10], MPI-bench-suit[11], SkaMPI[16], MPIBlib[17], а также разрабатываемое на факультете ВМК МГУ программное средство network_tests2, которое является частью проекта PARUS[5]. Результаты тестирования могут быть использованы не только 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 объём данных разнообразного характера, который не является познаваемым человеком как есть и требует разработке методов сжатия, визуализации, автоматического анализа для выявления аномалий и закономерностей. Всё это делается с помощью алгоритмов кластеризации[3], алгоритмов выделения наиболее значимых параметров, и т.п.

Данная работа посвящена попытке автоматического анализа данных полученных как результат тестирования коммуникаций многопроцессорной системы с помощью программного средства network_tests2.

Наиболее интересными для анализа представляются MPI тесты, предоставляющие информацию по задержкам для конкретных пар процессов. Результаты таких тестов представляются в виде набора матриц задержек для каждой пары процессов по разным длинам сообщений. В случае подхода сбора трасс, такие матрицы называются коммуникационными матрицами. К сожалению для достаточно большого числа процессоров данные результатов теста будут обладать большим объёмом, так как размер матриц пропорционален квадрату числа процессов, участвовавших в тесте.

Основной проблемой при анализе результатов тестирования является большой объем данных. Так для всего 128 процессов матрица для конкретной длины сообщения будет состоять из 16384 элементов с плавающей точкой. Если различных длин сообщений 100, то имеем достаточно большое число. Для размеров тестов с числом процессов, сопоставимым с суммарным числом ядер многопроцессорного комплекса, имеем объемы, которыми может быть достаточно сложно оперировать. Например, BlueGene/P на факультете ВМК имеет 8192 процессорных ядер [9], 67 108 864 чисел с плавающей точкой в матрице, то есть 64Мб занимает только одна матрица.

Естественным, с точки зрения сокращения объёма хранимой информации, является нахождение групп схожих значений в результатах тестирования. Для решения данной задачи применяется кластерный анализ. Одним из популярных методов кластерного анализа является иерархическая кластеризация, которая позволяет иерархически объединять элементы в группы. В силу архитектурных особенностей, задержки при обмене сообщениями между процессами в многопроцессорных комплексах часто тяготеют к иерархической структуре. Так процессы, находящиеся на разных ядрах одного процессора будут давать результаты тестирования очень схожие, образуя общий кластер; процессы распределенные по разным процессорам, но в пределах одной платы будут также походить друг на друга: тем самым образуется логическая иерархия — процессы одного процессора входят в группу процессоров одной платы. Архитектура коммуникационной среды может сильно повлиять на различие во времени передачи сообщений между процессорами. В ходе визуального анализа результатов тестирования было замечено, что величины задержек на показываемом изображении группируются в некоторые, обычно прямоугольные области. Причем эти области остаются устойчивыми со сменой длины сообщения в тесте.

Для получения данных о скорости передачи сообщений в коммуникационной среде системы в данной работе использовалась программа тестирования network_test, которая является частью системы Parus[7]. Результатом работы программы является несколько файлов в формате NetCDF[2], которые хранят: минимальные значения, медианы, средние и стандартные отклонения задержек. Каждый файл содержит множество матриц, где каждая матрица отвечает своей длине сообщения. В позиции i,j матрицы содержится задержка при передаче сообщения от процесса с номером i к процессу с номером j.

Ранее, для тех же данных Дмитрием Андреевым и Алексеем Сальниковым был предложен алгоритм кластеризации[4]. Этот алгоритм преследовал две цели: уменьшить размер файла результатов тестирования, при этом сохранив быструю скорость позиционного доступа к данным, и уменьшить число хранимых одновременно в памяти матриц, уменьшая расходы оперативной памяти. Для каждой отдельной матрицы можно выделить группы элементов с близкими значениями. Выделенные группы могут различаться для матриц, соответствующих разным длинам сообщений, из-за возможных различий в алгоритмах маршрутизации для конкретной длины и ограниченности размеров кэшей, используемых для передачи данных.

Данный алгоритм кластеризации основывается на предположении, что большей экономии памяти можно добиться, выделяя группы пар (i,j), которым соответствуют близкие по значению элементы нескольких матриц результатов теста, тогда для всех исходных матриц потребуется только одна матрица с индикаторами групп, к которым принадлежат элементы (i,j). Данный алгоритм не требует больших ресурсов оперативной памяти, и при правильно подобранных параметрах способен сохранять структуру матриц, при этом сжимая размер данных в несколько раз (рис. 3).

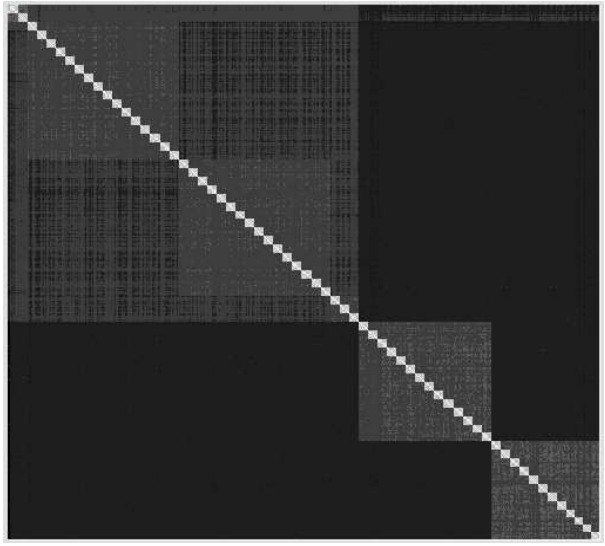

Рис.1 Ломоносов. 1000 процессов. Длина сообщения 1000.

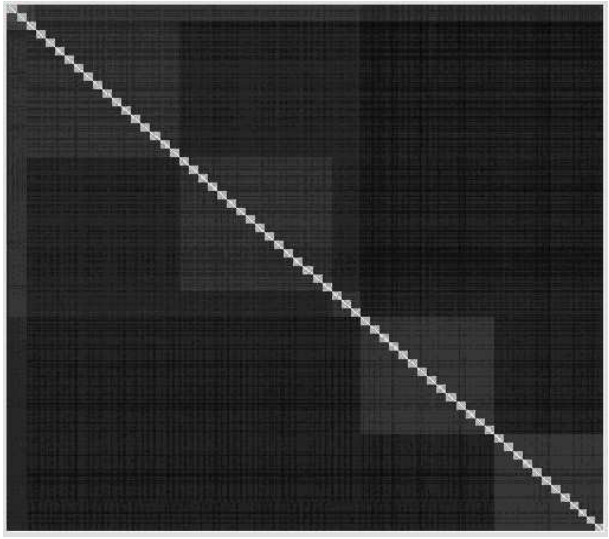

Рис. 2 Ломоносов. 1000 процессов. Длина сообщения 8000.

По отношению к работе[4] в алгоритм были внесены некоторые изменения. Был изменён параметр precision, отвечающий за разбиение на кластеры каждой отдельной матрицы. В текущей реализации он задаёт относительную долю в разности значений кластеров нормированный на максимальное значение в матрице и первый не нулевой элемент. Это связано с тем, что диагональ матрицы соответствует посылке сообщения самому себе и как следствие довольно часто имеет нулевые значения (например для режима one_to_one в network_test это всегда так). При том, что значения других задержек могут быть сконцентрированы в небольшом промежутке, где абсолютное значение может сильно отличаться от 0.

К сожалению, несмотря на хорошую скорость работы и нетребовательность к памяти, алгоритм обладает рядом серьёзных недостатков, которые актуальны именно с точки зрения автоматического определения аномалий. Данный алгоритм старается выделять трехмерные кластеры, но делает это с помощью попыток применить разбиение полученное для первой матрицы к остальным в окне 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.

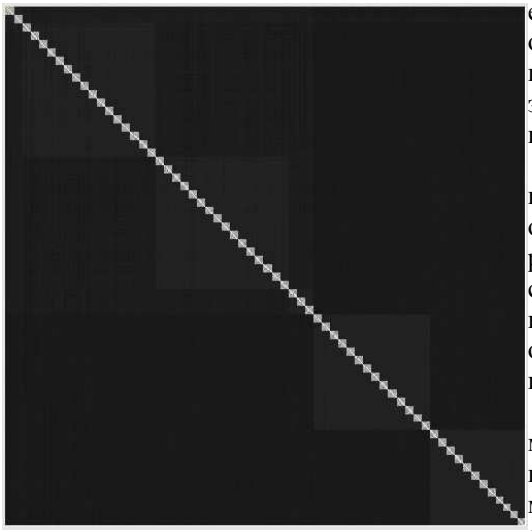

Рис. 3: Результат кластеризации. Ломоносов. 1000 процессов. длина сообщений 8000

Основной проблемой алгоритма является то, что выделение кластеров происходит строго последовательно по одному из измерений (по размеру передаваемых сообщений). В ряде случаев, кластер образуемый на шаге N может включить в себя матрицы, который могли бы создать больший кластер на шаге N+1. Данный алгоритм способен находить лишь локально оптимальное решение. Проблемы алгоритма видны из сравнения исходных и обработанных данных (рис. 4-5).

При определённом 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 Ломоносова для 1000 процессов. Улучшенная версия алгоритма выложена на сайте проекта Parus[7].

В следствие описанных проблем был предпринят поиск алгоритма кластеризации, способного предоставить больше информации из результатов тестирования. Как уже было описано выше, коммуникационная среда многопроцессорных комплексов тяготеет к иерархической структуре, потому поиск интересующего алгоритма был осуществлен среди алгоритмов иерархической кластеризации.

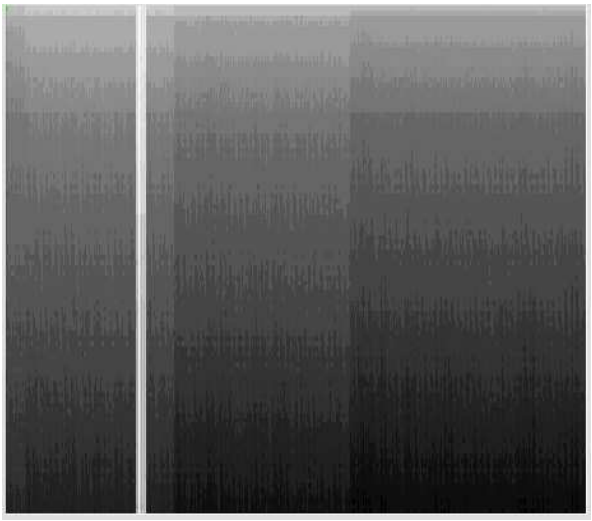

Рис. 4 Ломоносов. 1000 процессов. Срез по длинам сообщений. Необработанные данные

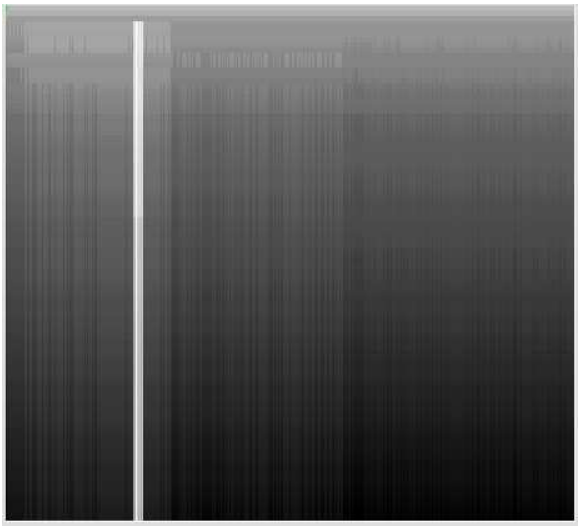

Рис.5 Ломоносов. 1000 процессов. Срез по длинам сообщений после кластеризации

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

Neighbor Joining — популярный в биоинформатике метод иерархической кластеризации, используемый для построения филогенетических деревьев (phylogenetic tree) [6]. Данные деревья отображают близость организмов по всем признакам одновременно, показывая как далеко ушли друг от друга в процессе эволюции организмы. Характерной особенностью алгоритма является способность сохранять расстояния от элементов до

общего предка. Алгоритм работает с матрицей расстояний, потому возникает необходимость преобразовать данные. Для этого должна существовать метрика, позволяющая рассчитать расстояние между двумя элементами. 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 исследуемых многопроцессорных комплексов достаточно симметричны, т.е. отличие между (i,j) и (j,i) элементами совсем невелико. Данное предположение, вообще говоря, неверно, например, для GRID-систем, однако мы решили в нашей работе не обращать на него внимание, считая, что основной исследуемый объект для нас вычислительные кластеры.

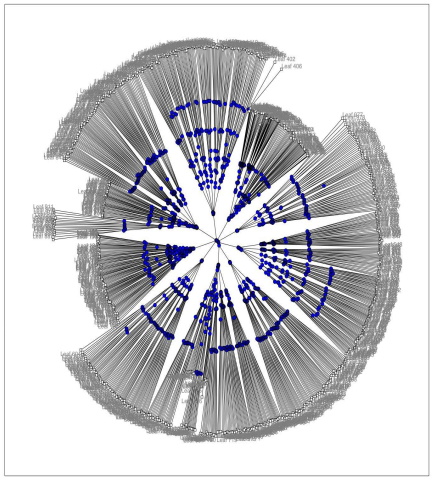

Рис. 6: СКИФ-МГУ. Результат применения Neighbor-Joining для матрицы, соответствующей длине сообщения 0

Алгоритм Neighbor-Joining является представителем класса алгоритмов агломеративной кластеризации, наиболее популярном подходе иерархической кластеризации данных. Здесь каждый элемент изначального множества является кластером. На каждой итерации алгоритма пара самых близких кластеров U и V образует новый кластер W. 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 способы определить расстояние между кластерами [12].

Работа алгоритма была опробована на данных результатов тестирования one_to_one многопроцессорных комплексов СКИФ-МГУ, MVS50k, BlueGene/P, Regatta. В качестве матрицы расстояний в основном брались матрицы для минимальных длин сообщений. Из результатов применения Neighbor-Joining с матрицей задержек в качестве матрицы расстояний, хорошо определяется структура многопроцессорных комплексов с иерархической архитектурой, таких как СКИФ-МГУ[8] (рис. 6), MVS50k, MVS100k, а для таких систем как BlueGene/P[9] с топологией трехмерного тора матрицу задержек в качестве матрицы расстояний для иерархической кластеризации использовать нельзя (рис. 7-8).

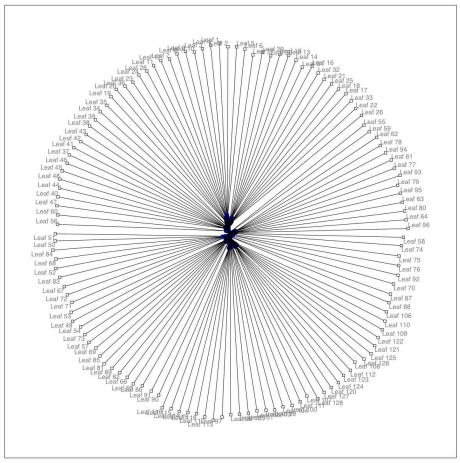

Рис. 7 BlueGene/P. 128 процессов, длина 80.

Результат Neighbor-Joining

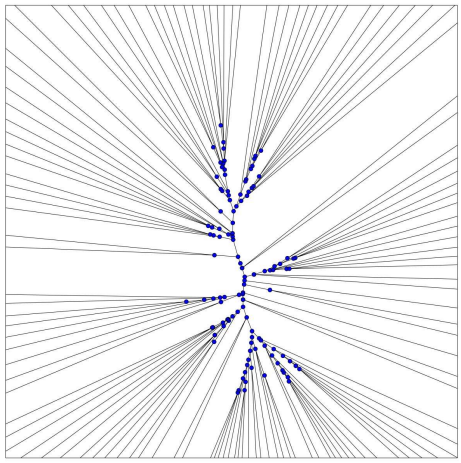

Рис. 8 BlueGene/P. 128 процессов, длина 80.

Увеличенный результат Neighbor-Joining

Исходя из вышеописанного замечания по поводу применения иерархической кластеризации к матрицам задержек как к матрицам расстояний, было предложено считать пары процессов (i,j) отдельными элементами, таким образом перейти от матриц к набору векторов. Т.е. от матриц $A^l = a_{ij}^l$, l — длина сообщения, переходим к векторам $B^{ij} = b_l^{ij}$. Теперь будем искать среди множества связей (i,j), такие, что задержка при передаче растет похожим образом. Тогда связи с похожей зависимостью от длины сообщений будут объединятся в группы, а так как связи между любыми двумя процессами на одном процессоре одной модели должны иметь схожую

зависимость от длины сообщений, то можно сделать предположение, что в результате кластеризации будет восстановлена актуальная топология связей между процессорами.

Для применения алгоритмов кластеризации на новых данных необходимо ввести метрику над пространством векторов B^{ij}. С ростом длины сообщений увеличивается и разброс в задержках, таким образом в вводимой метрике необходимо учитывать дисперсию. Для удовлетворения вышеописанных требований была предложена следующая метрика*:

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

$average_{ij}(l)$ - среднее значение задержек между процессами i и j для длины l.

$deviation_{ij}(l)$ - квадрат дисперсии

Алгоритмам агломеративной кластеризации необходимо знание расстояний между всеми объектами. При числе процессов n, участвующих в тестировании, имеем n^2 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 (либо матрица 16 384 x 16 384). Что 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. Это свойство отлично подходит для больших объемов данных: нет необходимости расчитывать расстояние между элементами разных кластеров, разбиение внутри одного кластера не зависит от разбиения в другом, что предоставляет возможность параллельной реализации алгоритма. Также дивизивная кластеризация позволяет остановится на любом числе разбиений, когда агломеративной необходимо объединить все элементы, вне зависимости от необходимого числа кластеров.

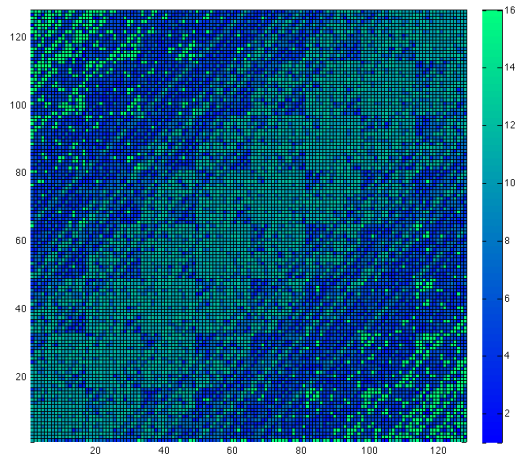

Рис. 9 BlueGene/P 128 процессов.
Агломеративная кластеризация. 16 кластеров

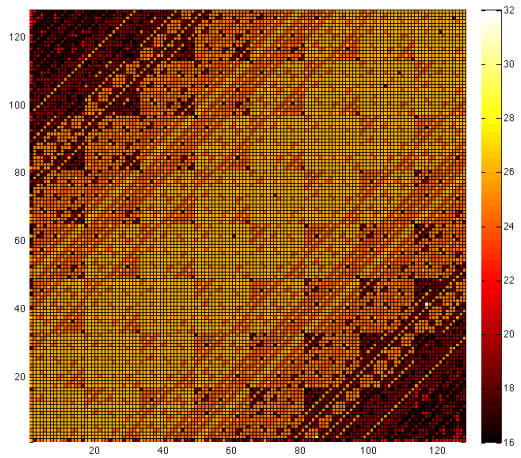

Рис. 10 BlueGene/P 128 процессов.
Дивизивная кластеризация. 16 кластеров

К сожалению, за счёт не полных сравнений, дивизивная кластеризация проигрывает в точности агломеративной. Например дивизиная кластеризация может разбить множество на два кластера, основываясь на близости элементов к центрам кластеров. Так два пограничных элемента близких между собой могут оказаться в совершенно разных кластерах.

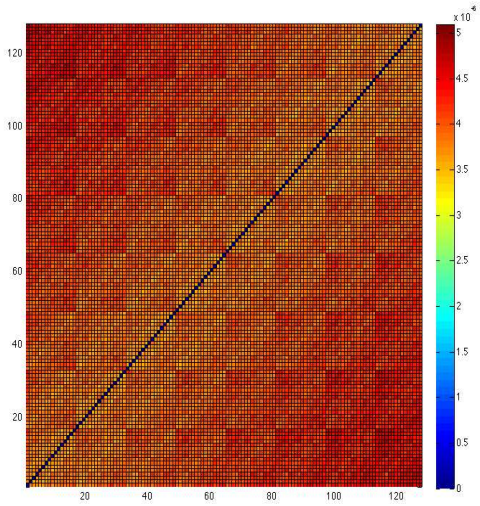

Рис. 11: BlueGene/P 128 процессов. Длина сообщений 0.

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

1. 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. Запоминаем расстояние между ними. Считаем r и s центрами кластеров.
3. Распределяем все элементы по двум кластерам.
4. Повторяем алгоритм независимо для двух кластеров. Причем, в ходе распределения уже имеем самый удаленный элемент от центра кластера, т.е. шаг 1 уже выполнен.

Алгоритмы агломеративной и дивизивной кластеризации были опробованы на данных результатов тестирования one_to_one 128 процессов BlueGene/P. На рис. 9-11 представлены матрицы задержек для 128 процессов BlueGene/P. На рис 9-10 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.

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 алгоритмов дивизивной кластеризации. Данный класс алгоритмов, при небольших потерях в точности по сравнению с 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.

ЛИТЕРАТУРА:

1. MPI: A Message-Passing Interface Standard [HTML] (http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/mpi-report.htm).
2. NetCDF User's Guide [PDF] (http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.pdf)
3. Jain A., Murty M., Flynn P. Data clustering: A review // ACM Computing Surveys. — 1999. — Vol. 31, no. 3. — Pp. 264–323.
4. А.Н. Сальников, Д.Ю. Андреев «Разработка инструментальных средств контроля за коммуникационнойсредой вычислительных кластеров с большим числом процессорных элементов» // Труды Пятой Международной конференции "Параллельные вычисления и задачи управления" PACO'2010. М.: Учреждение Российской академии наук Институт проблем управления им. В.А. Трапезникова РАН, 2010г. Стр. 1187-1208
5. А.Н. Сальников, Д.Ю. Андреев «Исследование методов тестирования коммуникационной среды многопроцессорных систем» // Научный сервис в сети Интернет: решение больших задач: Труды Всероссийской научной конференции, издательство МГУ имени М.В. Ломоносова, 2008г. С. 237-240.
6. Saitou, N., and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4(4), p. 406–425.
7. Проект Parus http://parus.sourceforge.net/

8. Описание вычислительного комплекса СКИФ-МГУ [HTML] (http://www.t-platforms.ru/ru/skif_msu.php)
9. IBM System Blue Gene Solution: Blue Gene/P Application Development. An IBM Redbooks publication. [PDF] (http://www.redbooks.ibm.com/abstracts/sg247287.html)
10. Intel® MPI Benchmarks: Users Guide and Methodology Description [HTML] (http://www3.intel.com/cd/software/products/asmo-na/eng/219848.html).
11. MPI-bench-suit: mpitest [HTML] (http://parallel.ru/testmpi/mpitest.html).
12. Lance G. N., Willams W. T. A general theory of classification sorting strategies. 1. hierarchical systems // Comp. J. — 1967. — no. 9. — Pp. 373–380.
13. Список 50 наиболее высокопроизводительных установок по тесту linpack в России и странах СНГ. [HTML] (http://top50.supercomputers.ru).
14. Список 500 наиболее высокопроизводительных установок по тесту linpack в Мире [HTML] (http://www.top500.org).
15. Balaji P., Buntinas D., Goodell D., et al.: MPI on a Million Processors// Recent Advances in Parallel Virtual Machine and Message Passing Interface. Vol. 5759/2009. p .20—30.
16. Reussner R., Hunzelman G. Achieving Performance Portability with SKaMPI for High Performance MPI Programs // LNCS. 2001. N 2074. pp. 841—850.
17. Lastovetsky A., Rychkov V., O'Flynn M. MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneous and Heterogeneous Clusters // LNCS. 2008. N 5205. pp. 227—238.
18. Markus Geimer, Felix Wolf, Brian J. N. Wylie, Daniel Becker, David Böhme, Wolfgang Frings, Marc-André Hermanns, Bernd Mohr, Zoltán Szebenyi: Recent Developments in the Scalasca Toolset. In Tools for High Performance Computing 2009, Proc. of the 3rd Parallel Tools Workshop, Dresden, Germany, chapter 4, pages 39-51, Springer, 2010.
19. Brunst, H., Nagel, W.E., Malony, A.D.: A distributed performance analysisarchitecture for clusters. In: IEEE International Conference on Cluster Computing, Cluster 2003, Hong Kong, China, pp. 73–81.
20. S. Shende and A. D. Malony, "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287-331, 2006