Анализ поведения параллельных MPI-программ на основе фаз межпроцессного взаимодействия*

М.В. Шубин, Н.Н. Попова

Московский государственный университет имени М.В. Ломоносова

Анализ поведения программ на основе выделения фаз в процессе их выполнения активно используется для решения многих проблем, связанных с оптимизацией программ, предсказания их поведения при изменении вычислительных платформ, входных параметров. Особую актуальность фазовый анализ получает для анализа параллельных программ.

В работе предлагается метод определения фаз параллельной программы на основе коммуникационных взаимодействий процессов, в рамках которых выполняется эта программа. Применение предложенного метода демонстрируется на примере ряда параллельных программ.

Основная идея предлагаемого метода заключается в сжатии информации о коммуникационных взаимодействиях процессов с использованием вейвлет-преобразований и разбиении временной шкалы параллельного приложения на отрезки, во время которых сохраняются некоторые свойства коммуникационных взаимодействий.

Предполагается дальнейшее развитие предложенного подхода и применение его для решения актуальной задачи динамического мэппинга параллельных программ.

Ключевые слова: Фазовый анализ, Коммуникационное поведение программ, Вейвлет-преобразования

1. Введение

С ростом вычислительных ресурсов растёт спрос на крупномасштабные вычисления. В больших суперкомпьютерах всё большее влияние на производительность оказывают накладные расходы на коммуникации. При скорости работы коммуникационной сети меньшей, чем требует параллельное приложение, эффективность параллельного приложения может существенно снижаться. Однако, более быстрые коммуникационные сети стоят дороже. Для пользователей и производителей важны как производительность, так и цена. Для достижения компромисса между ними требуется понимание коммуникационного поведения параллельных приложений.

Одним из путей снижения коммуникационных расходов в параллельных программах является оптимизация размещения процессов параллельных программ по процессорам (и ядрам) параллельной вычислительной системы. Решение этой проблемы, известной как *задача мэппинга*, требует анализа обмена сообщениями между процессами, которые проходили во время выполнения всего приложения. Задача мэппинга активно исследуется в настоящее время. Предлагаемые решения базируются на знании суммарных коммуникационных расходов параллельного приложения. Чаще всего информация о таких расходах представляется в виде коммуникационных матриц. Для приложений, характер взаимодействия параллельных процессов которых может меняться со временем, становится актуальным постановка задачи мэппинга процессов в динамике выполнения параллельного приложения. Назовем эту задачу *задачей динамического мэппинга*.

В данной статье предлагается и исследуется подход к анализу коммуникационных взаимодействий процессов параллельных программ на основе выделения фаз. *Фазой* назовем такой отрезок во временной шкале выполнения параллельной программы, при котором ха-

*Работа выполнена при финансовой поддержке РФФИ (грант №17-07-01562).

рактеристики коммуникационных взаимодействий меняются незначительно.

Фазовый анализ параллельной программы может быть применен для решения задачи динамического мэппинга. Мэппинг для отдельных фаз может повысить эффективность параллельного приложения. Фазовый анализ может быть полезен при переносе параллельной программы на другую вычислительную систему, при проведении сравнительного анализа поведения различных параллельных приложений, оптимизации параллельного приложения и для других целей.

Почти все существующие алгоритмы поиска наилучшего мэппинга базируются на итоговых коммуникационных матрицах, собираемых по завершению выполнения параллельной программы. Однако, характеристика коммуникационных взаимодействий параллельных процессов может существенно менятся. Предлагаемый подход позволит определить в параллельных MPI-программах моменты времени, в которые будет целесообразным изменять назначение процессов на узлы. Механизмы такого переназначения в данной статье не рассматриваются. Такое переназначение будет особо актуальным для параллельных программ, требующих длительного времени расчёта.

Выделение фаз может быть полезным и для лучшего понимания пользователем характера поведения параллельной программы на конкретных входных данных, возможного объяснения причин возникающих моментов неэффективного поведения программ. Эта проблема особенно актуальна для случаев использования открытых пакетов, например, Amber, Gromacs.

Характеристики коммуникационных взаимодействий, соответствующие фазам, могут быть использованы для настройки управляемых параметров конкретных реализаций MPI-библиотек перед запуском параллельных приложений.

Предлагаемый метод фазового анализа рассматривается в применении к параллельным MPI-программам. Метод основан на применении вейвлет-преобразований к последовательности коммуникационных матриц, построенных на основе трасс параллельного приложения, кластеризации преобразованных матриц и определения фаз - максимально длинных отрезков постоянства класса в построенной последовательности классов.

2. Обзор существующих подходов

Задача анализа поведения параллельных программ в том или ином виде рассматривается во многих работах.

Chao Ma, Yong Meng Teo, Verdi March и другие в работе [2] приводят метод сравнения работы двух параллельных программ. Они извлекают из коммуникационного поведения следующие характеристики: временной, объёмный и пространственный векторы, коммуникационный граф. Над векторами проводится ранговое преобразование, которое избавляет характеристики от влияния размера входных данных (размера задачи, на котором запускалась параллельная программа). Затем считается три коэффициента корреляции между векторами двух приложений (для времменного, пространственного и объёмного векторов), и усредняются. Получается метрика θ. Вычисляется также метрика ϵ схожести двух коммуникационных графов. На основе полученных двух метрик принимается решение о сходстве двух параллельных программ.

Ghislain Landry Tsafack Chetsa, Laurent Lefévre, J. M. Pierson и другие предлагают простой способ для выделения фаз в работе системы [3]. Однако этот метод можно применять и для анализа параллельных приложений. Метод заключается в измерении аппаратных счётчиков (промахи в кэше, промахи при предсказании вевтления и др.) в равные промежутки времени. Для каждого промежутка формируется вектор-столбец из этих счётчиков. Вектор каждого промежутка сравнивается с вектором последующего промежутка. Между ними считается *расстояние городских кварталов (Manhattan distance)*. Если это расстояние превышает некий порог, то считается, что в данный момент времени начилась новая фаза.

В работе также предлагается метод сравнения двух фаз: для каждой фазы вычисляется *представительский вектор*. Эти вектора сравниваются, и на основе этого сравнения фазы могут быть объявленны одинаковыми. Таким образом, метод позволяет обнаружить повторения фаз. Метод является удобным, т. к. не требует от пользователя дополнительных знаний, не требует трассировки программы и т. п. (для применения нужен только доступ к аппаратным счётчикам). Измерение проводятся на одном узле вычислительной системы, и может быть применён как для последовательных, так и для параллельных программ.

В работе [4] вводится понятие коммуникационного шаблона, приводится метод построения шаблонов, и предлагается метод для их классификации. При этом используются элементы теории графов, теории сетей. Предлагается два алгоритма сопоставления двух коммуникационных шаблонов: изоморфизм подграфов и проверка гипотез (критерий Колмогорова-Смирнова). Эти два метода сравниваются на больших наборах программ.

Метод, предлагаемый в данной работе, развивает идеи, рассматриваемые в ряде работ, посвященных фазовому анализу программ. Так, например, в работе [1] вейвлет-преобразования применяются для анализа поведения последовательных приложений для персональных компьютеров, не имеющих отношения к высокопроизводительным вычислениям. В работе анализируются обращения к памяти в динамике выполнения программы. На основе трасс обращений к памяти и применения к этим трассам вейвлет-преобразований и кластеризации преобразованных матриц выделяются фазы. Разработанный метод был применён к ряду прикладных программ (Mozilla, OpenOffice, Gimp и др.). В работе показано, что предложенный метод распознаёт поведение программ в памяти более точно, чем другие методы.

Большинство работ, связанных с анализом коммуникационных взаимодействий процессов, посвящены определению повторяющихся участков в межпроцессных взаимодействиях [3, 8]. В данной же работе предлагается метод обнаружения участков с различным характером коммуникационного поведения.

3. Предлагаемый метод решения

Основная идея предлагаемого в работе метода базируется на выделении характеристик коммуникационных матриц, собираемых в процессе выполнения параллельного приложения, с использованием вейвлет-преобразований. Вейвлет-преобразования широко используется в таких областях, как обработка изображений, сигналов. В работе [5], ставшей отправной точкой для многих работ, предложен метод поиска в базе данных изображений с использованием характеристик изображений, извлеченных применением 2-мерных вейвлет-преобразований Хаара. В работе найдено оптимальное число коэффициентов в разложении и параметры, на которые эти коэффициенты должны умножаться для оптимизации поиска изображений. Предложенные параметры были использованы в предлагаемом нами методе.

Алгоритм поиска фаз в коммуникационных взаимодействиях процессов параллельной программы, предлагаемый в данной работе, состоит из следующих основных шагов:

- Сбор трассы параллельного приложения
- Построение последовательности коммуникационных матриц по собранным трассам
- Масштабирование матриц к размеру 16×16.
- Применение к полученным матрицам 2D вейвлет-преобразования Хаара
- Умножение матриц на специальные поправочные веса
- Кластеризация матриц методом k-средних
- Определение фаз — максимально длинных отрезков постоянства класса в построенной последовательности классов

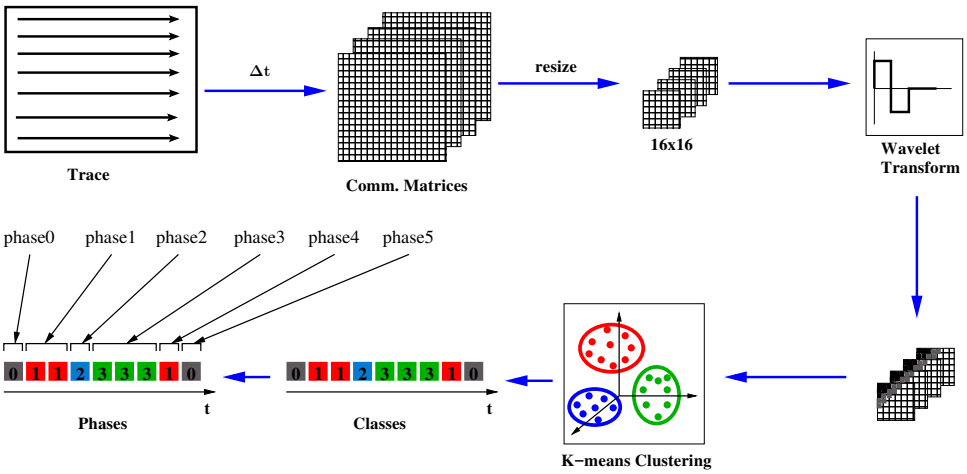

Рис. 1. Общая схема метода

Схема данного алгоритма приведена на рис. 1.

3.1. Сбор трассы

Рассмотрим параллельную MPI-программу, запущенную на n процессах. *Трассой параллельной программы* будем считать совокупность трасс процессов, в рамках которых выполняется эта программа. Трасса процесса состоит из *событий*. Событие — это вызов MPI-функции. В данной работе рассматриваются следующие вызовы: `MPI_Send`, `MPI_Isend`, `MPI_Sendrecv`. Каждое событие характеризуется *временем наступления*, *объёмом сообщения* в байтах, и *номером процесса-получателя*.

Для сбора трассы используется MPI Profiling interface. Его использование заключается в перехвате нужных MPI-функций путём их переопределения и использования их аналогов с префиксом PMPI.

Сбор трассы может проводиться как с использованием разработанных для этих целей реализаций соответствующих MPI-функций, так и с использованием системы сбора трасс Score-P [6]. В этом случае обеспечивается возможность работы с трассами, представляемыми в формате OTF2.

3.2. Коммуникационная матрица

По трассе параллельной программы можно построить т. н. *коммуникационную матрицу*. Существует несколько разных определений коммуникационной матрицы. Приведём то, которое используется в данной работе.

Коммуникационная матрица $\mathbf{Com}[\mathbf{n}, \mathbf{n}]$ — это квадратная матрица, размер которой равен числу процессов, элемент $\mathbf{Com}(\mathbf{i}, \mathbf{j})$ равен суммарному количеству байт, переданных от процесса с номером i к процессу с номером j.

Можно также рассматривать коммуникационную матрицу на временно́м промежутке. *Коммуникационная матрица на промежутке* $[t_1, t_2]$ — это квадратная матрица, размер которой равен числу процессов, элемент в позиции (i, j) равен суммарному количеству байт, отсылаемых из процесса i процессу j, передача которых была инициирована в момент времени $t : t_1 \leqslant t \leqslant t_2$.

3.3. Уменьшение размера матриц

Перед вейвлет-преобразованием проводится уменьшение размера коммуникационных матриц до размера 16×16 (предполагается, что число процессов анализируемой программы, больше, чем 16). Это делается по двум причинам: во-первых, двумерное вейвлет-преобразование Хаара требует, чтобы размер матрицы являлся степенью двойки; во-вторых, весь дальнейший анализ (вейвлет-преобразование, кластеризация) требует времени, и будет работать медленно, если размер матрицы большой. Как и в работе по анализу обращений в память [1], здесь также было решено, что размер 16 является компромиссом между производительностью и точностью метода.

Алгоритм уменьшения размера матрицы можно грубо описать следующим образом: вся матрица разбивается равномерно на $16 \times 16 = 256$ клеток, в каждую клетку кладётся усреднённое значение всех элементов, попавших в данную клетку.

3.4. Вейвлет-преобразование Хаара

Одномерное преобразование Хаара заключается в выполнении операций усреднения и разности $\log_2 N$ раз над массивом длины N. Сначала вычисляются средние по каждой паре чисел и записываются в первую половину выходного массива, затем вычисляются разности между первым числом в паре и средним и записываются во вторую часть массива (эти разности ещё называют *коэффициентами детализации*). Затем та же самая процедура применяется к первой половине массива. И так далее, до длины массива, равной единице.

Рассмотрим пример: пусть у нас есть массив [5 9 4 2]. Среднее чисел 5 и 9 равно 7, а чисел 4 и 2 равно 3, поэтому первая часть массива будет 7 3. Затем, вычисляем разности $5 - 7 = -2$, $4 - 3 = 1$, поэтому коэффициенты детализации будут равны -2 1. Получим массив [7 3 -2 1]. Затем, применяем это же действие к подмассиву [7 3]. $(7 + 3)/2 = 5$, $7 - 5 = 2$. В итоге, получим массив [5 2 -2 1].

Двумерное преобразование Хаара применяется к матрицам, и заключается в том, что сначала применяется одномерное преобразование Хаара к каждой строке, затем к каждому столбцу результирующей матрицы.

3.5. Умножение на поправочные веса

Как и в работе [1], после вейвлет-преобразования, элементы матрицы умножаются на поправочные веса. Это нужно, потому что «важность» элемента матрицы зависит от его позиции в матрице (при подсчёте евклидового расстояния между матрицами). Выбор этих весов подробно описан в статье [5], в которой излагается метод поиска изображений в базе по нарисованному или отсканированному изображению.

Поправочные веса имеют следующий вид:
$w_y[0] = 4.04$
$w_y[1] = 0.78$
$w_y[2] = 0.46$
$w_y[3] = 0.42$
$w_y[4] = 0.41$
$w_y[5] = 0.32$
Элемент матрицы с координатами (row, col) умножается на вес $w_y[bin(row, col)]$, где:
$bin(i, j) = min(max(level(i), level(j)), 5)$
$level(i) = \lfloor(\log_2(i + 1))\rfloor$

3.6. Кластеризация методом k-средних

Метод, называемый *методом k-средних* применяется для того, чтобы сгруппировать элементы (многомерные векторы) на классы с похожими свойствами. Изначально, каждому

элементу присваивается случайный класс. Для каждого класса вычисляется центр. Затем, каждому элементу присваивается тот класс, к центру которого он ближе всего расположен. После этого, снова вычисляются центры и так далее. Этот процесс повторяется до тех пор, пока классы не стабилизируются. Между элементами должно быть определено расстояние.

В нашем случае, кластеризуются матрицы, при этом они рассматриваются, как векторы размерности 256. В качестве расстояния между ними используется евклидово расстояние.

3.7. Определение фаз

После кластеризации мы имеем последовательность классов, к которым были отнесены матрицы временных промежутков анализируемой программы. В этой последовательности выделяются максимально длинные отрезки постоянства класса. Назовём эти отрезки *фазами.*

Например, для следующей последовательности классов (классы нумеруются с нуля):

1 1 0 3 4 4 4 4 4 4 0 1

выделятся следующие фазы:

$$\underbrace{1\,1}_{\text{фаза1}}\ \underbrace{0}_{\text{фаза2}}\ \underbrace{3}_{\text{фаза3}}\ \underbrace{4\,4\,4\,4\,4\,4}_{\text{фаза4}}\ \underbrace{0}_{\text{фаза5}}\ \underbrace{1}_{\text{фаза6}}$$

3.8. Выбор параметров

Параметрами предлагаемого метода являются Δt — временной шаг для сбора коммуникационных матриц и k — число классов для метода кластеризации. Исследование, представленное в статье, выполнялось при фиксированных значениях этих параметров. Выбор значений проводился эмпирически. Таким образом, значение k было выбрано равным 5, а значение Δt полагалось примерно равным одной десятой длины трассы. Критерии и механизмы выбора значений этих параметров подлежат дальнейшему исследованию.

4. Результаты анализа некоторых приложений

Наш метод был применён к нескольким параллельным программам. Все программы запускались на системе IBM BlueGene/P, установленной на факультете ВМК Московского государственного университета имени М. В. Ломоносова. Запуск почти всех программ, анализ которых приведен в этом разделе, проводился на 32 узлах. Для одной из программ также анализировался запуск на 64 узлах. Это числа были выбраны для удобства рассмотрения результатов, чтобы размер коммуникационных матриц был не очень большим.

Для каждой из программ приводится результат анализа в виде разбиения времени выполнения программы на фазы. Для фаз приводятся их коммуникационные матрицы. Коммуникационной матрице можно поставить соответствие *коммуникационный граф*. В этом коммуникационном графе можно вычислить среднюю, минимальную и максимальную *степени вершин*, эти степени приводятся в качестве характеристики фаз. Также, для каждой фазы приводятся средний, минимальный и максимальный суммарный объём отправленных сообщений среди всех процессов.

4.1. Тестовая программа

В качестве первой программы рассмотрим искусственную тестовую программу. Эта программа без вычислений, она содержит коммуникации и вызовы `sleep()` между ними. Процессы в ней выстраиваются в двумерный тор. Затем выполняются четыре пересылки с помощью вызова `MPI_Sendrecv`, между которыми выполняется вызов `sleep(1)`. Пересылки заключаются в том, что каждый процесс посылает данные соседу сверху (снизу, справа

```
MPI_Sendrecv(...) /* пересылка "вверх" */
sleep(1);
MPI_Sendrecv(...) /* пересылка "вправо" */
sleep(1);
MPI_Sendrecv(...) /* пересылка "влево" */
sleep(1);
MPI_Sendrecv(...) /* пересылка "вниз" */
```

Рис. 2. Фрагмент искусственной тестовой программы

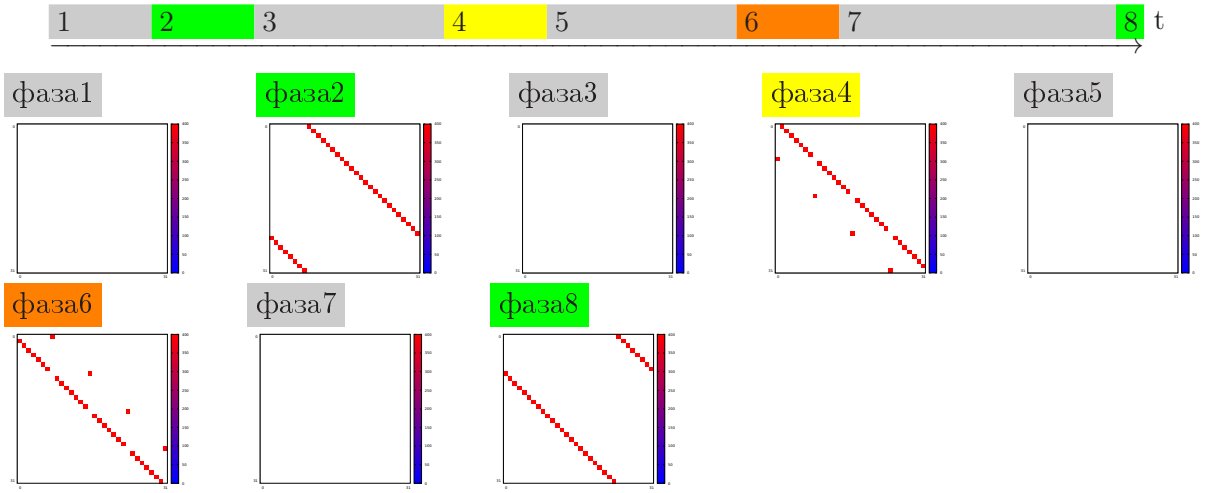

Рис. 3. Выделенные фазы и их коммуникационные матрицы, тестовая программа, 32 узла. Цвет соответствует номеру класса, к которому были отнесены матрицы фазы, однако если в фазе не было коммуникаций, она отбражается серым цветом

или слева) от себя данные и принимает данные «с противоположной» стороны. На рис. 2 приведён вид этой программы.

Проводился запуск этой программы на 32 узлах. Время выполнения программы составило 3.34 с. Параметры анализа были выбранными следующими: $\Delta t = 0.3$ с, $k = 5$. На рис. 3 представлен результат работы метода: показаны выделенные фазы и их коммуникационные матрицы. Из рисунка видно, что были выделенны 8 фаз, из которых 4 имеют коммуникации, остальные без коммуникаций. Фазы, содержащие коммуникацию, заметно отличаются друг от друга коммуникационными матрицами. «Фаза2» как раз соответствует передачи вверх по решётке. В этом легко убедиться, если учесть, что 32 процесса логически выстраиваются в решётку 4×8, «нижняя строка» из 8-ми процессов (номера 0–7), посылает сообщение строке, лежащей выше неё (номера 8–15), та, в свою очередь, следующей строке, и так далее. Как раз это и отражено в соответствующей коммуникационной матрице. Аналогично, фазы 4, 6 и 8 отражают передачи вправо, влево и вниз соответственно.

На рис. 4 приведены две характеристики фаз: средняя степень вершины в коммуникационном графе (т. е. число процессов, которым отправлял сообщение данный процесс, усреднённое по всем процессам), и средний объём сообщений, отправленных процессами (т. е. для каждого процесса считатется число байтов, которые он отправил, и затем берётся среднее по всем процессам). Заметим, что в данном случае эти характеристики в каждой фазе у всех процессов одинаковы, поэтому максимум и минимум совпадает со средним значением. Из рисунка видно, что все «ненулевые» с точки зрения коммуникаций фазы, одинаковы по этим характеристикам.

Можно считать, что в данном случае метод распознал все виды обменов между про-

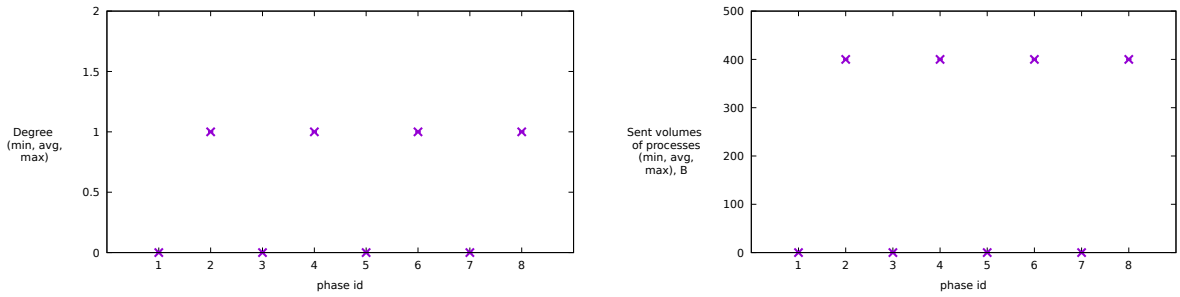

Рис. 4. Характеристики фаз: средняя степень вершины в коммуникационном графе (слева) и средний объём, отправленный процессами за каждую фазу (справа), тестовая программа, 32 узла

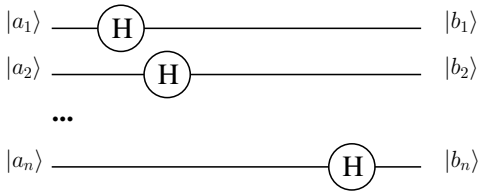

Рис. 5. Схема процесса, который моделирует программа «квантовое преобразование N-Адамар»

цессами в программе, и отнёс их к соответствующим фазам. Также, благодаря анализу мы увидели промежутки, в которых не было коммуникаций.

4.2. Квантовое преобразование N-Адамар

Рассмотрим теперь программу «Квантовое преобразование N-Адамар». Она осуществляет серию однокубитных преобразований Адамара, а также зашумлённого преобразования Адамара над n-кубитными состояниями. Есть два вектора, представляющих два n-кубитных состояния. К одному из них применяется преобразование Адамара, ко второму применяется зашумлённое преобразование Адамара. Преобразования применяются последовательно сначала к первому кубиту, затем ко второму и так далее до n-го. Схема преобразования приведена на рис. 5

Вектор, описывающий n-кубитное состояние, представляет собой набор из 2^n комплексных чисел, и хранится распределённо. Вектор распределяется по процессам поровну, первая его часть хранится у нулевого процесса, вторая у первого и так далее. Эта программа требует, чтобы число процессов, на котором она будет запущена, являлось степенью двойки.

Анализировался запуск этой программы также на 32 узлах на 25 кубитах. Время выполнения программы: 2.4 с. Параметры анализа: $\Delta t = 0.3$ с, $k = 5$. На рис. 6 показаны найденные фазы и их коммуникационные матрицы. Из рисунка видно, что выделилось пять различных фаз: все имеют различный портрет, пятая фаза представляет собой период без коммуникаций, который длится примерно всю вторую половину времени работы программы. Видно, что характер взаимодействия при движении от первой фазы к четвёртой меняется следующим образом: Вначале идёт обмен по схеме: первая половина процессов взаимодействует со второй (т. е. 0-процесс с 16-ым, 1-ый с 17-ый и т. д.), затем идёт обмен по схеме: первая четверть процессов со взаимодействует со второй, третья с четвёртой, затем также по восьмым частям от всего количества процессов и т. д. Такого рода обмены как раз соответствуют применению однокубитного преобразования к распределённому вектору-состоянию сначала для первого кубита, затем для второго и т. д. Если 2^m — это число процессов, k — номер кубита, к которому применяется однокубитное преобразование,

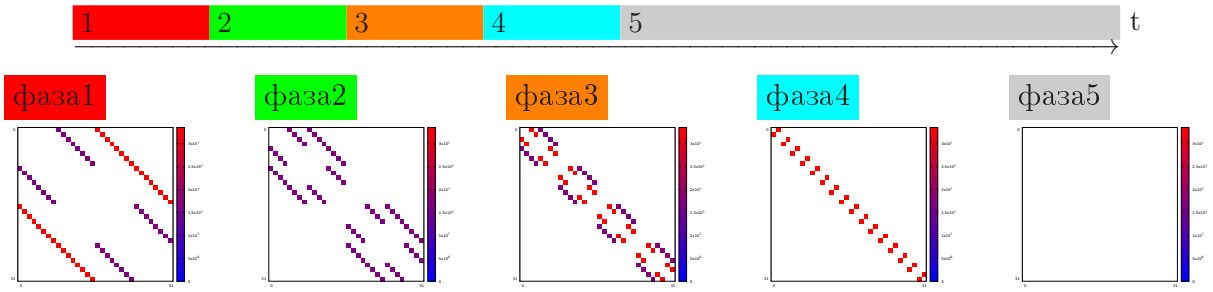

Рис. 6. Выделенные фазы и их коммуникационные матрицы, программа «Квантовое преобразование n-Адамар», 32 узла Цвет соответствует номеру класса, к которому были отнесены матрицы фазы, однако если в фазе не было коммуникаций, она отбражается серым цветом

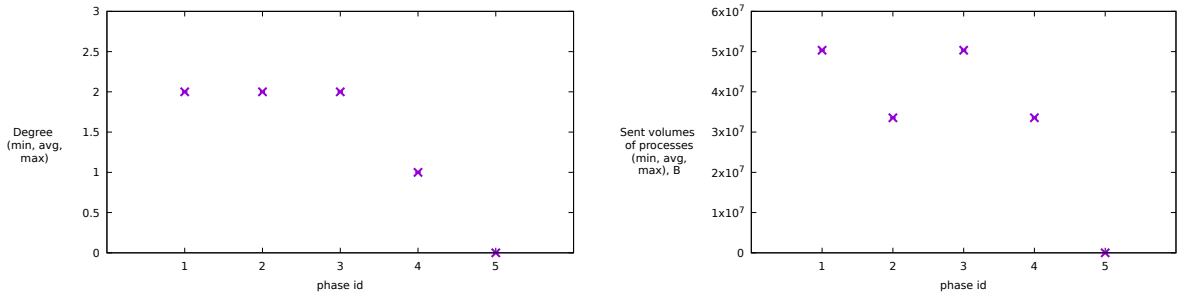

Рис. 7. Характеристики фаз: средняя степень вершины в коммуникационном графе (слева) и средний объём, отправленный процессами за каждую фазу (справа), «Квантовое преобразование n-Адамар», 32 узла

то при $k > m$ обмены не требуются вообще, что и объясняет отсутствие коммуникаций во второй половине работы программы.

На рис. 7 приведены характеристики выделенных фаз (число процессов, с которыми взаимодействовал данный процесс, усреднённое по всем процессам и средний объём отправленных байтов). Опять же, заметим, что и в этом случае эти характеристики в каждой фазе у всех процессов одинаковы, поэтому максимум и минимум совпадает со средним значением. Из рисунка видно, что у четвёртой фазы средняя степень коммуникационного графа меньше, чем у предыдущих фаз, средний объём и второй и четвёртой фазы меньше, чем у первой и третей. Эти отличия объясняются тем, что в разные фазы попало разное число однокубитных преобразований.

Также был проведён анализ запуска программы на 64 узлах при том же размере входных данных: 25 кубитов. Время выполнения программы: 1.39 с. Параметры анализа: $\Delta t = 0.15$ с, $k = 5$. Результаты соответствующего анализа представлены на рис. 8. Из рисунка видно, что как и для 32-х узлов были определенны несколько фаз, с различными коммуникационными матрицами. Также видно, что каждая следующая фаза содержит коммуникации от преобразования кубита, следующего по номеру. На рис. 9 приведены характеристики этих фаз (максимальная, минимальная и среднее степени вершин в коммуникационном графе и максимальный, минимальный и средний объёмы в байтах, отправленные процессами за каждую фазу). Видно, что средняя степень и объёмы в целом остаются постоянными со второй по пятую фазу. Однако, в третьей и четвёртой фазах видна «аномалия» в виде того, что есть отклонения от среднего. Это объясняется тем, что несколько процессов начали проводить коммуникации, относящиеся к однокубитному преобразованию раньше (в предыдущей фазе) относительно остальных.

Для данной программы, как и для тестовой программы, метод отнёс различные виды

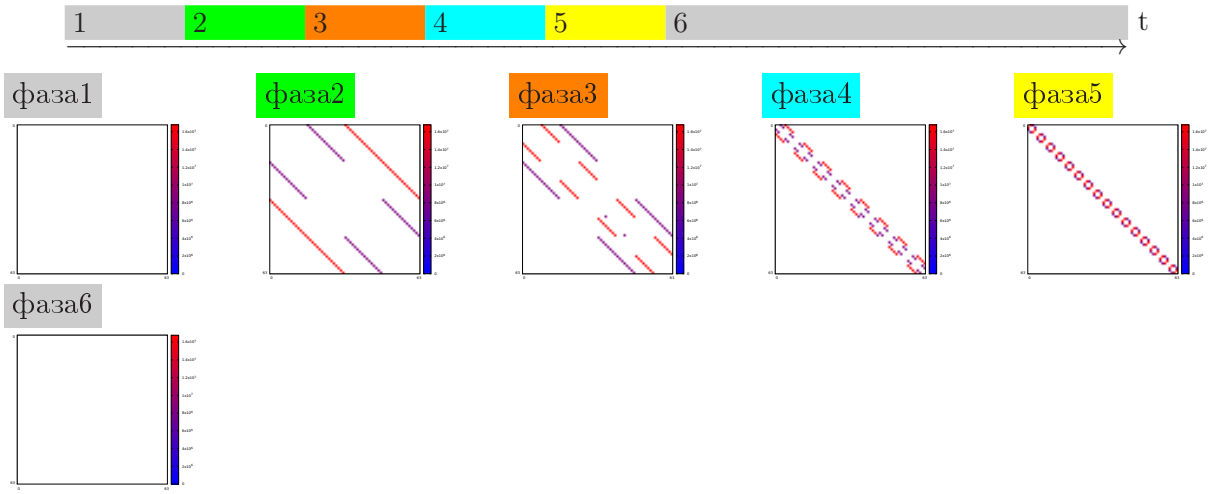

Рис. 8. Выделенные фазы и их коммуникационные матрицы, программа «Квантовое преобразование n-Адамар», 64 узла Цвет соответствует номеру класса, к которому были отнесены матрицы фазы, однако если в фазе не было коммуникаций, она отбражается серым цветом

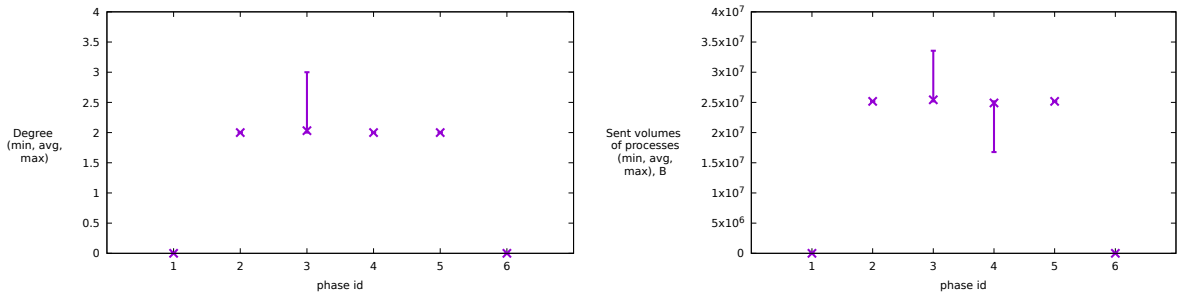

Рис. 9. Характеристики фаз: средняя, максимальная и минимальная степени вершин в коммуникационномграфе (слева) и средний, максимальный и минимальный объёмы, отправленные процессами за каждую фазу (справа), «Квантовое преобразование n-Адамар», 64 узла

коммуникаций к разным фазам, однако в одну фазу всё-таки попало два вида коммуникаций (от преобразования первого кубита и от преобразования второго), это может быть исправлено уменьшением параметра Δt. Здесь, мы проанализировали два запуска этой программы на разном числе узлов при одинаковом размере входных данных. Видно, что разбиение времени на фазы в обоих случаях похоже: вторая половина программы и там и там без коммуникаций, первая половина содержит несколько разльичных фаз как для 32-х, так и для 64-х процессов. Можно предположить, что эта программа будет демонстрировать стабильное фазовое поведение и дальше при сильной масштабируемости.

4.3. HPCG

Теперь рассмотрим программу HPCG. HPCG является одним из тестов, используемых для измерения производительности суперкомпьютеров [7].

Использовалась версия HPCG 3.0. Параметры запуска следующие: $dx = dy = dz = 32$ (размер области в каждом процессе), $dt = 10$ (требуемое время счёта в секундах). Число узлов: 32.

Время выполнения составило: 25.976537 с. Параметры анализа: $\Delta t = 3$ с, $k = 5$. На рис. 10 представлен результат работы метода: выделенные фазы и их матрицы. Из рисунка

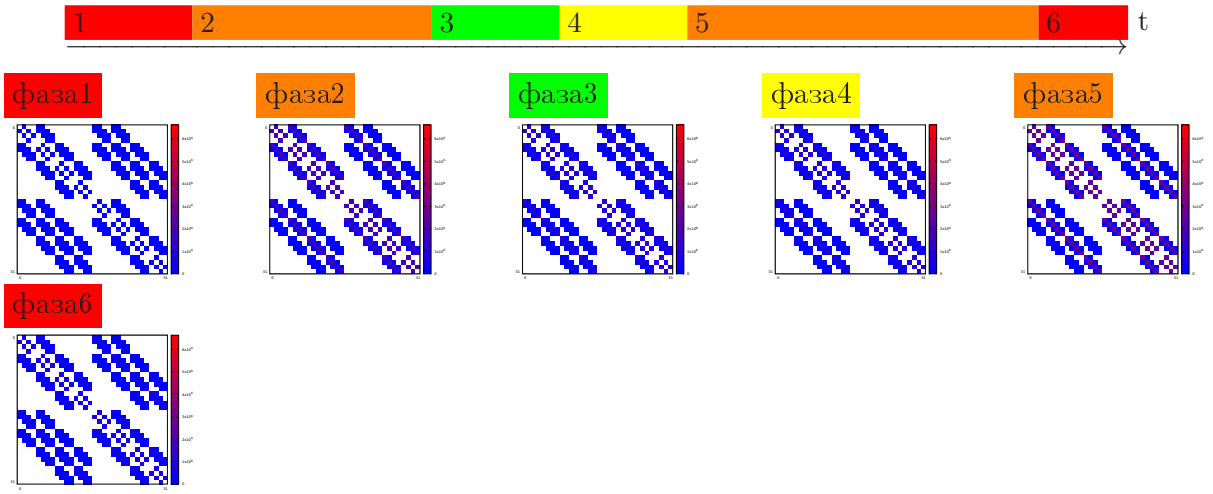

Рис. 10. Выделенные фазы, HPCG, 32 процесса

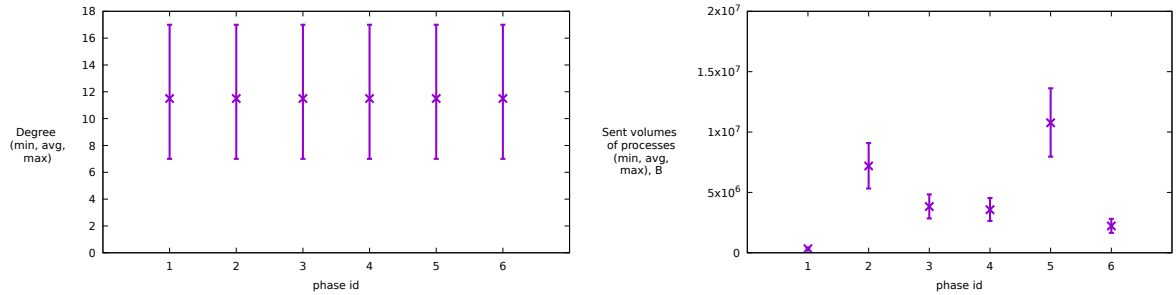

Рис. 11. Характеристики фаз: средняя, максимальная и минимальная степени вершин в коммуникационномграфе (слева) и средний, максимальный и минимальный объёмы, отправленные процессами за каждую фазу (справа), HPCG, 32 узла

видно, что анализ дал несколько фаз различной длины. Первая и последняя (шестая) фазы были отнесены к одному классу. Вторая и пятая также были отнесены к одному классу. Во время работы программы не было «пустых» промежутков без коммуникаций. Портреты матриц всех фаз одинаковы, откуда можно сделать вывод, что в целом характер межпроцессного взаимодействия не меняется во время выполнения программы. Это объясняется тем, что данный параллельный тест представляет собой итерационный алгоритм.

На рис. 11 приведены характеристики соответствующих фаз (степени вершин коммуникационного графа: средняя, минимальная, максимальная и суммарные объёмы в байтах, отправленные процессами: средний, минимальный, максимальный для каждой фазы). Видно, что степени вершин графа (т. е. количества «соседей», с которыми взаимодействовали процессы) разные у разных вершин (процессов). Однако в каждой фазе эти степени одинаковы. Это объясняется одинаковым видом коммуникационных матриц у каждой фазы. Что касается объёмов, переданных, процессами, то здесь они разные у каждой фазы: наиболее интенсивный обмен был у пятой фазы, у первой фазы — наименее интенсивный.

Метод показал, что характер взаимодействия не меняется со временем (это было видно из схожести «портретов» матриц). На этапе разбиения трассы в каждый промежуток времени длины Δt попало несколько итераций. Поскольку все итерации одинаковы между собой по характеру межпроцессного взаимодействия, получились одинаковые на вид коммуникационные матрицы.

5. Заключение

В работе представлен метод фазового анализа коммуникационных взаимодействий процессов параллельных MPI-программ. Применение метода к тестовой параллельной программе и ряду реальных параллельных приложений позволило выделить классы программ с изменчивым характером взаимодействий.

Метод позволил выделить в приложениях несколько различных фаз, относя разные типы взаимодействий к разным фазам. Для каждой фазы определены характеристики соответствующих им коммуникаций: степени вершин коммуникационного графа, размеры отправленных процессом сообщений. Эти характеристики могут быть полезными при изучении свойств приложений.

Для программы «Квантовое преобразование N-Адамар» было обнаружено, что при сильной масштабируемости, фазовое поведение остается относительно стабильным. Выделенное свойство может быть использовано для предсказания поведения программы на большем числе вычислительных узлов.

В дальнейшем предполагается развитие предложенного метода. Планируется применить другие методы на этапе кластеризации и сравнить прменимость этих методов с методом k-средних. Планируется расширить разработанный метод учётом коллективных операций. Предполагается применение метода для практических задач, таких как задача динамического мэппинга.

Литература

1. Ted Huffmire and Tim Sherwood, Wavelet-Based Phase Classification, Department of Computer Science, University of California, Santa Barbara, 2006.

2. Chao Ma, Yong Meng Teo, Verdi March, Naixue Xiong, Ioana Romelia Pop, Yan Xiang He, Simon See, An Approach for Matching Communication Patterns in Parallel Applications, 2009.

3. Ghislain Landry Tsafack Chetsa, Laurent Lefévre, J. M. Pierson, Patricia Stolf, Georges Da Costa. A User Friendly Phase Detection Methodology for HPC Systems' Analysis. The 2013 IEEE International Conference on Green Computing and Communications (GreenCom 2013), Aug 2013, Beijing, China. pp.118 – 125, 2013.

4. Sean Whalen, Sean Peisert, Matt Bishop, Network-Theoretic Classification of Parallel Computation Patterns, Berkeley Lab and University of California, Davis, 2011.

5. Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. In SIGGRAPH 1995, Los Angeles, CA, August 1995.

6. Knüpfer A. et al. (2012) Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Brunst H., Müller M., Nagel W., Resch M. (eds) Tools for High Performance Computing 2011. Springer, Berlin, Heidelberg
URL: http://link.springer.com/chapter/10.1007%2F978-3-642-31476-6_7

7. HPCG — The High Performance Conjugate Gradients Benchmark project:
URL: http://www.hpcg-benchmark.org.

8. François Trahay, Élisabeth Brunet, Mohamed Mosli Bouksiaa, Jianwei Liao, Selecting points of interest in traces using patterns of events, 2015