УДК 004.051 DOI: 10.14529/cmse200103

НЕЙРОСЕТЕВОЙ МЕТОД РЕШЕНИЯ ЗАДАЧИ МЭППИНГА ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ*

© 2020 Н.Н. Попова, М.В. Козлов, М.В. Шубин

Московский государственный университет имени М.В. Ломоносова
(119991 Москва, ул. Ленинские горы, д. 1)
E-mail: popova@cs.msu.su, rat.taurus@gmail.com, mihshub@gmail.com
Поступила в редакцию: 12.11.2019

Статья посвящена проблеме повышения эффективности параллельных приложений. В статье предлагается подход к решению проблемы, основанный на сокращении накладных расходов, связанных с передачей данных между процессами параллельной программы во время ее выполнения на высокопроизводительной вычислительной системе. С ростом числа процессорных узлов расходы на передачу сообщений между узлами оказывают все большее влияние на производительность параллельных приложений. В связи с этим становится особо актуальной задача размещения процессов параллельной программы по вычислительным узлам суперкомпьютера, известная, как задача мэппинга. В работе предлагается новый подход к решению задачи мэппинга. Ключевой особенностью подхода является выбор коммуникационного шаблона путем фазового анализа приложения и использование сверточной нейронной сети для быстрого выбора подходящего алгоритма мэппинга, исходя из построенного коммуникационного шаблона.

Для построения коммуникационных шаблонов проводится анализ поведения приложения с точки зрения передачи сообщений точка-точка между процессами параллельной программы. Временная шкала событий передачи сообщений разбивается на равные промежутки, для каждого из которых строится коммуникационных шаблон. К построенным шаблонам применяется двумерное вейвлет-преобразование Хаара для выделения признаков. Затем проводится кластеризация признаков и построение фаз во временной шкале работы приложения. Для каждой фазы строится коммуникационный шаблон, соответствующий этой фазе.

Выбор подходящего алгоритма мэппинга проводится с помощью сверточной нейронной сети. Использование нейронной сети предполагает знание о свойствах коммуникационного поведения различных типов приложений и подходящих для этих типов алгоритмов мэппинга. Эти знания должны быть представлены в виде набора классов коммуникационных шаблонов (матриц) с известным для каждого класса наилучшим алгоритмом мэппинга. Нейронная сеть обучается на данном наборе классов. Обученная сеть решает задачу классификации входного коммуникационного шаблона, выбирая наиболее подходящий алгоритм мэппинга для данного параллельного приложения.

В статье представлена реализация отдельных этапов метода, и продемонстрирована их работа на тестовых примерах.

Ключевые слова: суперкомпъютер, мэппинг, коммуникационный шаблон, сверточная нейронная сеть.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Попова Н.Н., Козлов М.В., Шубин М.В. Нейросетевой метод решения задачи мэппинга параллельных приложений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 1. С. 36–49. DOI: 10.14529/cmse200103.

Введение

Одним из путей повышения эффективности параллельных приложений является оптимизация размещения процессов параллельной программы по узлам и ядрам целевой вычислительной системы, на которой выполняется приложение. Задача об оптимальном размещении, получившая название мэппинг параллельных приложений, активно исследуется. Решение этой задачи приобретает особую ценность для высокопроизводительных систем

*Статья рекомендована к публикации программным комитетом Международной конференции «Суперкомпьютерные дни в России — 2019».

(HPC-систем) рекордной производительности [1]. В основе большинства предлагаемых подходов к решению задачи мэппинга лежит модель взаимодействия параллельных процессов, представленная в виде коммуникационных матриц, элементами которых являются численные характеристики сообщений, передаваемых между каждой из пар взаимодействующих процессов. В статье предлагается уточнение этой модели. Уточнение проводится в двух направлениях: альтернативного подхода к построению коммуникационной матрицы и нового подхода к определению мэппинга, основанного на использовании сверточных нейронных сетей. Оба направления предлагаемого подхода основаны на использовании алгоритмов машинного обучения. Хотя каждое из направлений является самодостаточным, они могут использоваться в различных задачах, связанных с анализом поведения и настройкой эффективности параллельных программ.

Задача мэппинга является NP-полной [2]. Для ее решения широко используются разнообразные эвристики. Однако, далеко не каждый алгоритм мэппинга уменьшает время выполнения программы, а некоторые даже его увеличивают. В качестве подтверждения этого можно привести примеры, представленные в статье [3]. В данной работе показано, что в зависимости от используемого алгоритма мэппинга, время выполнения приложения может как уменьшаться по сравнению с принятым по умолчанию стандартным мэппингом, так и увеличиваться. На подбор «правильного» алгоритма для данной задачи может уйти много времени и затрачены значительные вычислительные ресурсы. Вследствие этого определение подходящего алгоритма мэппинга для данной программы является еще одним способом оптимизации параллельных программ.

Основное содержание этапов предлагаемого подхода заключается в следующем. Коммуникационная матрица параллельного приложения рассматривается как изображение. На основе анализа динамики изменения коммуникационных взаимодействий выбирается коммуникационная матрица, отвечающая наиболее характерному паттерну взаимодействия процессов. Для построения мэппинга формируется сверточная нейронная сеть, которая относит коммуникационную матрицу, рассматриваемую как изображение, к одному из заданных классов. Для каждого из заданных классов заранее определяется алгоритм мэппинга, с использованием которого и находится лучший мэппинг. Для демонстрации разрабатываемого подхода в статье предлагается жадный алгоритм мэппинга, ориентированный на архитектуру с топологией 3-х-мерного тора. Подход находится в стадии разработки. Экспериментальная реализация этапов предлагаемого подхода демонстрируется на примере тестовых приложений для суперкомпьютера IBM Blue Gene/P.

Статья состоит из введения, 6 разделов и заключения. В разделе 1 дается обзор подходов к построению мэппинга. Общее описание предлагаемого подхода дано в разделе 2. В разделе 3 приводится краткое описание метода определения характерных коммуникационных матриц. В разделе 4 описывается модель сверточной нейронной сети для классификации коммуникационных матриц и выбора на этой основе алгоритма мэппинга. В разделе 5 подробно рассматривается предлагаемый метод построения мэппинга. В разделе 6 приводятся результаты проведенных вычислительных экспериментов с использованием разработанных алгоритмов. В заключении подводятся итоги исследования и предлагаются направления дальнейших работ.

1. Обзор алгоритмов мэппинга параллельных приложений

Алгоритмы, направленные на анализ коммуникационного поведения программ и решение задачи мэппинга, активно исследуются. Регулярно появляются новые публикации, посвященные различным аспектам решения этой актуальной задачи.

Алгоритмы, решающие задачу мэппинга, можно, условно, разделить на 3 типа: алгоритмы, работающие с графами, эволюционные алгоритмы и улучшающие алгоритмы. К первой группе можно отнести алгоритмы для нахождения изоморфизма графов разными способами, например, рекурсивное разбиение графа, поиск в ширину, «лучший — первый», RCM. При рекурсивном разбиении графов [4, 5] граф параллельного приложения и граф вычислительный системы рекурсивно разрезаются на два подграфа по какому-либо свойству графа. На выходе из рекурсии происходит отображение вершин графа приложения на граф вычислительной системы [2, 3]. При поиске в ширину во время первого прохода граф помечается. Далее на каждом шаге алгоритма на граф вычислительный системы отображается вершина графа приложения с наибольшим количеством еще не отображенных соседей [3]. Одним из простейших алгоритмов вида «лучший — первый» является жадный алгоритм [2, 5]. Случайным образом выбирается процесс, который помещается в коммуникационную сеть. Потом из списка выбирается сосед, с которым есть взаимодействие и ставится рядом в сети. На вход RCM алгоритму подается коммуникационная матрица, которая рассматривается как матрица смежности графа приложения. RCM алгоритм переставляет столбцы и строки матрицы таким образом, чтобы уменьшить ширину ленты матрицы [2, 3]. Эволюционными алгоритмами являются, например, различные генетические алгоритмы, разыскивающие мэппинг с помощью «направленного» перебора, отсекая заведомо плохие решения. Улучшающие алгоритмы предполагают наличие какого-либо начального решения и направлены на улучшение полученного решения. Например, итерационный алгоритм может менять расположение случайных процессов для получения лучшего локального решения [2].

2. Схема предлагаемого подхода

Основная идея предлагаемого подхода заключается в следующем.

Рассмотрим параллельную MPI-программу, запущенную на n процессах. Под трассой T параллельной программы будем понимать совокупность трасс процессов: $T = \{T_1, T_2, \ldots, T_n\}$. Трасса процесса — это последовательность событий: $T_i = \{s_1, s_2, \ldots, s_k\}$. Событие — это вызов MPI-функции. Учитывая особенности организации коммуникационных систем передачи данных, реализованных в BlueGene/P, будем рассматривать только функции передач типа точка-точка. Каждое событие характеризуется временем наступления, объемом сообщения, номером процесса-получателя. Под коммуникационной матрицей в данной работе понимается целочисленная квадратная матрица, размер которой равен числу процессов в MPI-программе. Элемент в позиции (i, j) равен суммарному размеру передаваемых сообщений (в байтах) от процесса с номером i процессу с номером j. Матрица может рассматриваться на временном промежутке $[t_1, t_2]$. В этом случае элемент в позиции (i, j) равен суммарному объему сообщений, переданных от процесса i к процессу j, передача которых была инициирована в момент времени $t \in [t_1, t_2]$.

Формально постановку задачи можно определить следующим образом. Пусть параллельная программа задана множеством параллельных процессов $p_i : P = \{p_1, p_2, \ldots, p_n\}$. Известна трасса программы T. Архитектура вычислительной системы представлена мно-

жеством узлов Q. Требуется найти отображение $F : P \to Q$ множества $P = \{p_1, p_2, \ldots, p_n\}$ процессов параллельной программы на множество $Q = \{q_1, q_2, \ldots, q_n\}$ узлов вычислительной системы. Такое отображение называется мэппингом. Большинство существующих алгоритмов мэппинга решают более частную задачу. В этой постановке рассматривается граф параллельного приложения $G = (V(G), E(G))$ и граф вычислительной системы $H = (V(H), E(H))$. Вершинами графа G являются процессы. Ребра соответствуют наличию коммуникаций между ними и имеют вес, соответствующий интенсивности этих коммуникаций. Вершины графа H соответствуют узлам вычислительной системы, ребра соответствуют наличию каналов для передачи сообщений.

Требуется найти отображение $T : V(G) \to V(H)$ вершин графа приложения на вершины графа вычислительной системы на котором достигается минимум функции:

$$\sum_{i,j=1}^{N} d(T(p_i), T(p_j)) \cdot w(p_i, p_j),$$

где $V(G) = \{p_1, p_2, \ldots, p_n\}$, $V(H) = \{q_1, q_2, \ldots, q_n\}$, $w(p_i, p_j)$ — вес ребра между вершинами p_i и p_j в графе G, $d(a, b)$ — кратчайшее расстояние между вершинами a и b графа H.

Таким образом, для использования алгоритмов мэппинга при решении общей задачи требуется по трассе приложения предъявить граф приложения. Для представления графа параллельного приложения используется коммуникационная матрица, которая является матрицей смежности графа.

Для описания коммуникационного поведения параллельной программы будем использовать коммуникационные матрицы, соответствующие некоторым заданным интервалам времени. Конкретный способ построения таких матриц может выбираться, исходя из имеющихся инструментов анализа поведения параллельных программ. В работе рассматривается метод сбора коммуникационных матриц для MPI-программ на основе их трасс. Сбор трасс реализован неинвазивным методом с помощью переопределения MPI-функций и их подключением к исполняемому коду путем динамической линковки. Учитываются вызовы `MPI_Send`, `MPI_Isend`, `MPI_Sendrecv`, как наиболее часто используемые вызовы для передач типа точка-точка. Коллективные операции, выполнение которых осуществляется в BlueGene/P выделенной коммуникационной сетью, в нашем случае рассматривать не будем. Данный метод не требует наличия исходного кода программы.

Альтернативным способом построения мэппинга может быть подача на вход алгоритму мэппинга коммуникационной матрицы, соответствующей какому-то интервалу времени, вместо суммарной коммуникационной матрицы программы, так как эта матрица может лучше представлять коммуникации в приложении, существенно влияющие на его производительность.

Как было показано выше, различные алгоритмы мэппинга могут по-разному влиять на время выполнения параллельного приложения по сравнению с используемым принятым по умолчанию стандартным алгоритмом мэппинга. Предположим, что у нас есть априорные знания об алгоритмах мэппинга и типах приложений, которые они ускоряют. Пусть эти знания представлены в виде набора классов коммуникационных матриц и подходящих алгоритмов мэппинга для каждого класса. Тогда данную коммуникационную матрицу мы можем отнести к одному из этих классов и на основе этого выбрать подходящий алгоритм мэппинга. Для классификации заданной коммуникационной матрицы предлагается использовать сверточную нейронную сеть, заранее обученную на различных выборках мат-

риц. Основным преимуществом нейронной сети является быстрота подбора подходящего алгоритма мэппинга.

Таким образом, предлагается следующая идея (рис. 1). Временная шкала выполнения параллельной программы разбивается на отрезки, называемые фазами. Метод выделения фаз описывается в разделе 3. Для каждой фазы вычисляется коммуникационная матрица, представляющая шаблон взаимодействия процессов в рамках этой фазы. Среди полученных коммуникационных матриц фаз на основе заданного критерия определяется одна матрица. Проводится классификация этой матрицы с помощью сверточной нейронной сети. Модель сверточной нейронной сети описана в разделе 4. В зависимости от класса, к которому была отнесена матрица, выбирается соответствующий данному классу алгоритм мэппинга. Пример одного из таких алгоритмов описан в разделе 5.

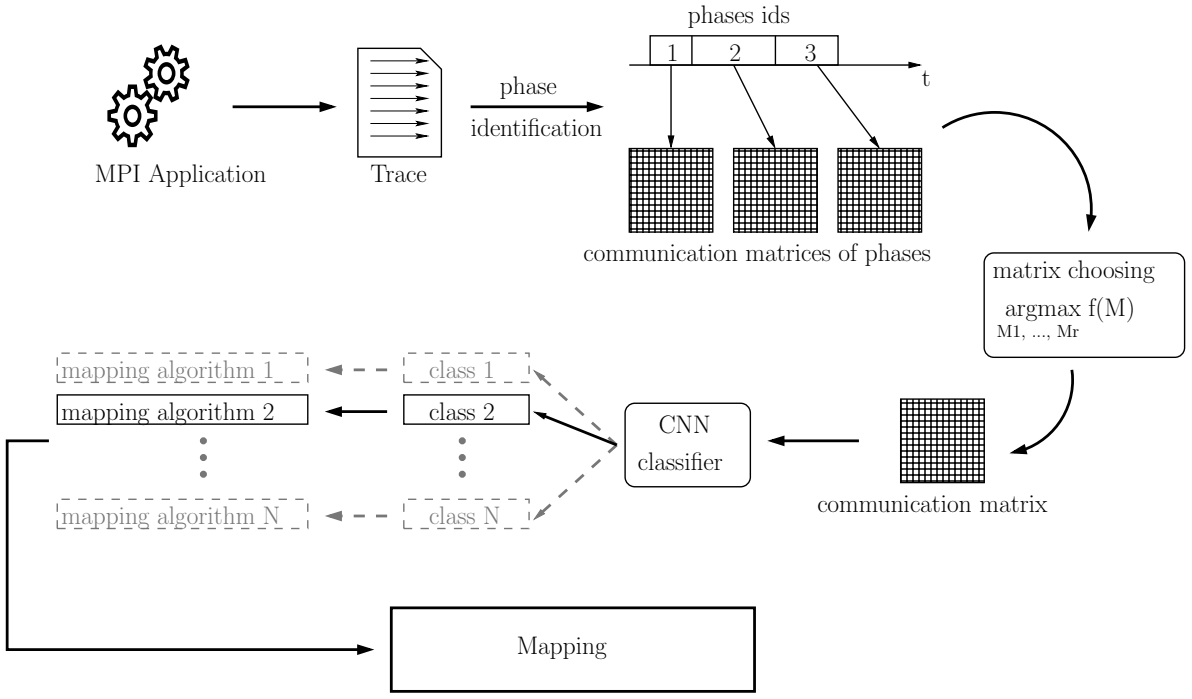


Рис. 1. Схема предлагаемого подхода к решению задачи мэппинга

3. Метод выбора коммуникационного шаблона для построения мэппинга

Опишем более детально этапы выделения фаз и выбора подходящей фазы в качестве коммуникационного шаблона для построения мэппинга. Описание метода приведем в соответствии с работой, выполненной авторами ранее [6]. Идея метода заключается в использовании вейвлет-преобразований для выделения характеристик коммуникационных матриц. Коммуникационные матрицы при этом будем представлять, как изображения. Применим к временному ряду построенных таким образом изображений метод поиска изображений, описанный в работе [7]. Метод использует двумерное вейвлет-преобразование Хаара. В указанной работе найдены оптимальные веса, на которые должны умножаться признаки после вейвлет-преобразования для оптимизации поиска изображения. Предложенные веса используются в предлагаемом нами методе. Метод состоит из следующих шагов: сбор трассы параллельного приложения; разбиение трассы равными временными промежутками длины Δt и построение коммуникационной матрицы для каждого промежутка (построение после-

довательности коммуникационных матриц); приведение матриц к стандартному размеру $m \times m$; применение к матрицам двумерного вейвлет-преобразования Хаара; умножение матриц на специальные поправочные веса; кластеризация полученных матриц; определение фаз в построенной последовательности классов.

Трасса параллельной программы разбивается на равные промежутки фиксированной длины Δt (Δt является параметром метода). Для каждого промежутка строится коммуникационная матрица. Имеем последовательность коммуникационных матриц.

Матрицы уменьшаются до размера $m \times m$ следующим образом: вся матрица разбивается равномерно на $m \times m$ клеток, в каждую клетку заносится усредненное значение всех элементов, попавших в данную клетку. Это делается по двум причинам: во-первых, используемое на следующем шаге вейвлет-преобразование требует, чтобы размер входной матрицы m являлся степенью двойки; во-вторых, при больших размерах матрицы вейвлет-преобразование и последующая кластеризация будут занимать много времени.

Далее, к полученным матрицам применяется двумерное вейвлет-преобразование Хаара. Оно заключается в применении одномерного вейвлет-преобразования Хаара сначала к каждой строке матрицы, а затем к каждому столбцу результирующей матрицы.

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 еще называют коэффициентами детализации). Затем та же самая процедура применяется к первой половине массива. И так далее до длины массива, равной единице.

После вейвлет-преобразования элементы матрицы умножаются на поправочные веса. Это умножение проводится с целью учета более важных коэффициентов после вейвлет-преобразования при подсчете евклидового расстояния между матрицами (во время кластеризации). Выбор этих весов $w[i], i = 0, 1, \ldots, 5$ подробно описан в статье [7].

Элемент матрицы с координатами (i, j) умножается на вес $w[bin(i, j)]$, где:

$$bin(i, j) = min(max(level(i), level(j), 5),$$

$$level(i) = [log_2(i + 1)].$$

Для отнесения похожих матриц к одному классу проводится их кластеризация. Матрицы рассматриваются как векторы размерности m^2. В качестве расстояния используется евклидово расстояние. Для кластеризации рассматривались методы: k-средних, DBSCAN.

После кластеризации имеем последовательность классов, к которым были отнесены матрицы временных промежутков анализируемой программы. В этой последовательности выделяются максимально длинные отрезки постоянства класса. Назовем эти отрезки фазами. Таким образом, каждая фаза состоит из стоящих рядом отрезков длины Δt, отнесенных к одному классу. Следовательно, в рамках фазы коммуникационное поведение не меняется или меняется незначительно. Если в программе коммуникационное поведение будет существенно меняться, каждый «шаблон» взаимодействия окажется в отдельной фазе.

Для каждой из полученных фаз строится коммуникационная матрица этой фазы. Среди них выбирается одна матрица в качестве коммуникационного шаблона приложения для построения мэппинга. Критерий выбора в общем виде имеет вид: $\underset{M \in \{M_1, M_2, \ldots, M_r\}}{argmax} f(M)$, где r — число фаз; $M_1, M_2, \ldots, M_r$ — матрицы фаз, $f : \mathbb{R}^{n \times n} \to \mathbb{R}$ — некоторая числовая

функция от матрицы. Т. е. требуется выбрать матрицу на которой достигается максимум функции f. В данной работе в качестве функции f рассматривается сумма всех элементов матрицы. Такой выбор обоснован тем, что фаза с наибольшим объемом переданных данных (сумма элементов ее матрицы будет больше сумм элементов матриц остальных фаз) предположительно представляет те коммуникации в программе, которые наиболее сильно влияют на производительность. Выбор функции f подлежит дальнейшему исследованию.

4. Сверточная нейронная сеть для классификации коммуникационных матриц

Классификацию коммуникационных матриц выполним на основе нейросетевого подхода. За основу нейросетевой модели была выбрана нейронная сеть VGG-16 [8], которая хорошо зарекомендовала себя в работе с изображениями, а коммуникационную матрицу можно представить, как изображение с одним каналом.

Построенная нейросетевая модель состоит из трех блоков свертка-свертка-макспулинг (conv-conv-maxpool), в каждом из которых свертка происходит с ядром размера 3×3, а пулинг с ядром размера 4×4 в первых двух блоках и 2×2 в последнем блоке. Количество фильтров в сверточных слоях равно 16, 32, 64 для соответствующих блоков. После трех блоков, описанных выше, следует персептрон с одним внутренним слоем.

Схема предложенной нейронной сети представлена на рис. 2.

Для построения и обучения модели использовался фреймворк Keras [9], который разработан специально для ускорения разработки и простоты использования нейронных сетей. Keras работает поверх одной из мощных библиотек Tensorflow, CNTK, Theano. В данной работе была использована библиотека Tensorflow.

Входными данными для обучения являются матрица и ее класс (целое число). Перед началом обучения к матрицам применяется ранговое преобразование по строкам: по каждой строке создается массив пар (номер элемента в строке, элемент), после чего полученный массив сортируется по элементу в порядке возрастания. Далее каждому элементу в исходной строке присваивается 0, если элемент был 0, и номер элемента в массиве пар, если элемент не 0. Нумерация начинается с единицы. Таким образом, исходная строка преобразуется с помощью рангового преобразования и имеет следующие свойства: все элементы будут в пределах от 0 до размерности матрицы, и если элемент i в исходной строке был больше элемента j, то тоже самое соотношение будет в полученной строке.

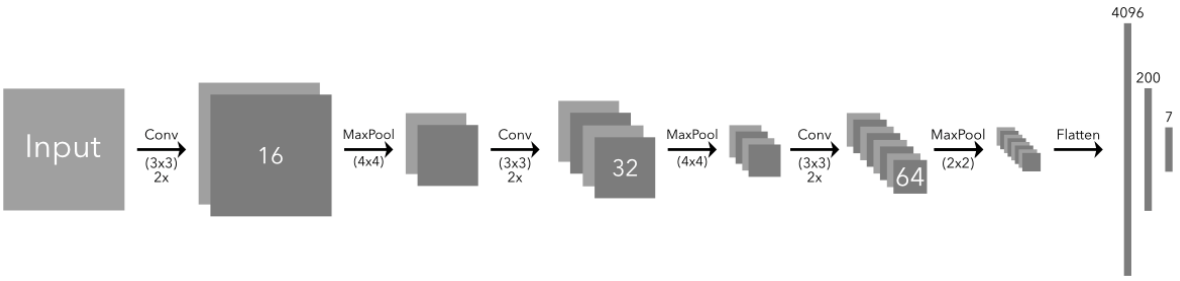


Рис. 2. Архитектура предложенной нейросетевой модели

Такая нейросетевая модель решает задачу классификации параллельных приложений по коммуникационной матрице, вследствие чего можно применить наилучший алгоритм мэппинга для полученного класса, а не подбирать алгоритм вручную.

5. Алгоритм мэппинга для HPC-систем с топологией 3D тор

Общая идея разработанного алгоритма мэппинга состоит в размещении наиболее часто взаимодействующих процессов на как можно более близких узлах параллельной вычислительной системы. Коммуникации между процессами определяются с помощью коммуникационной матрицы параллельного приложения, которая подается на вход алгоритму. Для каждого процесса алгоритм пытается найти 6 наиболее коммуницирующих соседей, если такие есть. Количество соседей выбрано не случайно: алгоритм создавался для коммуникационной сети типа 3-х-мерный тор, а в данной сети каждый узел имеет соединение с 6 узлами. После нахождения подходящих соседей для каждого процесса, алгоритм определяет новые координаты для процессов в трехмерном торе и строит мэппинг для данного параллельного приложения.

На рис. 3 представлена схема предложенного алгоритма.

Опишем предложенный алгоритм подробнее. На вход алгоритму подается коммуникационная матрица параллельной программы. Каждая строка матрицы сортируется, и создается внутренняя структура, которая называется мэппинг-вектор: матрица, размером $N \times 6$, где каждой строке под номером i выбирается 6 соседей, с которыми у данного процесса наибольший объем коммуникаций.

Далее генерируется внутреннее представление трехмерного тора, и процессы сортируются по множествам Process, Last, Done:

- множество Process содержит процессы, у которых есть хотя бы одна коммуникация;
- множество Last содержит процессы, которые не участвуют в коммуникациях точка-точка;
- множество Done содержит процессы, которые уже расставлены на трехмерный тор.

После распределения процессов по множествам, используя мэппинг-вектор и построенные множества, алгоритм расставляет процессы на трехмерный тор: выбирается самый нагруженный по объему сообщений процесс и ставится в 3D решетку вместе со своими соседями, с которыми у данного процесса наибольший объем коммуникаций. Все эти процессы заносятся в множество Done, так как они уже расставлены на коммуникационной решетке, а «инициирующий» процесс удаляется из множества Process. Далее выбирается один из соседей и те же шаги повторяются для него, если он еще содержится в множестве Process.

В конце расстановки по свободным местам в решетке распределяются процессы из множества Last, так как процессы без коммуникаций можно поставить в любое место решетки. Когда в множествах Process и Last не остается элементов, расстановка завершается. Для вывода мэппинга трехмерный тор переводится в одномерный массив пар (номер процесса, координаты в решетке) и сортируется по номеру процесса. После этого вторые значения в паре (координаты) выводятся в файл, который и будет мэппингом параллельной программы.

Сложность алгоритма порядка $O(N^2)$, где N — количество процессов.

6. Вычислительный эксперимент

6.1. Пример определения фаз и выбора коммуникационного шаблона

Для демонстрации применимости метода определения коммуникационного шаблона рассмотрим искусственное приложение, состоящее из последовательного вызова двух тестов из набора NAS Parallel Benchmarks: LU и CG [10]. Эксперимент проводился на си-

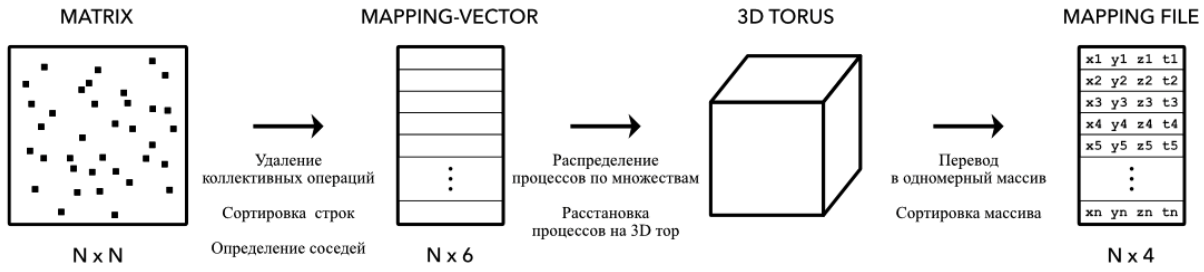
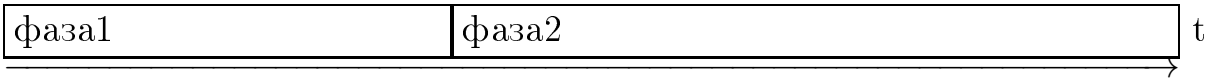
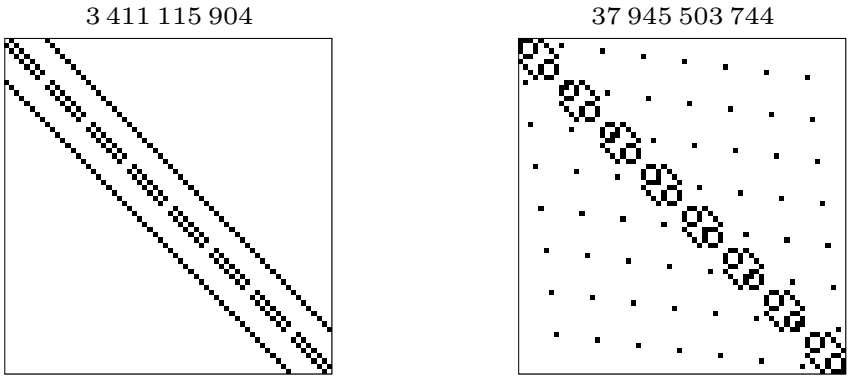


Рис. 3. Схема предложенного алгоритма мэппинга

стеме IBM Blue Gene/P на 64 узлах. Полученные фазы и их коммуникационные матрицы представлены на рис. 4. Числа над матрицами указывают сумму всех элементов матрицы. Видно, что вторая матрица имеет сумму значительно больше первой. Следовательно, она должна выбираться в качестве коммуникационного шаблона для построения мэппинга.



а) Фазы на временной шкале выполнения программ



б) Фаза 1 в) Фаза 2

Рис. 4. Пример определения фаз

6.2. Пример применения нейросетевой классификации тестовых коммуникационных матриц

Для тестирования нейросетевой модели и алгоритма мэппинга разработан генератор матриц 7 классов, модели которых представлены на рис. 5. Такие матрицы были выбраны после анализа нескольких статей (например, [11]) и немного упрощены для более легкого построения большого количества матриц. Среди представленных классов есть как похожие друг на друга матрицы, так и совершенно различные, что требуется для тестирования нейросетевой модели.

При генерации матриц случайно выбирались матрицы, которые случайным образом «зашумлялись»: добавлялись случайные ненулевые числа в случайные ячейки матрицы. Процент шума для каждой матрицы выбирался также случайным образом.

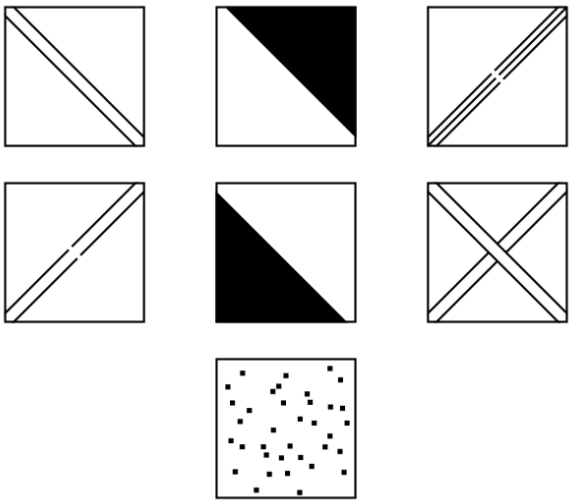


Рис. 5. Классы матриц, которые были использованы для обучения и тестирования

Для исследования эффективности разработанного алгоритма мэппинга использовались несколько классов матриц из числа описанных выше. С этой целью были разработаны тестовые программы, которые выполняют пересылки таким образом, что коммуникационная матрица таких программ имеет вид случайной матрицы, матрицы с двумя обратными диагоналями и зашумленной матрицы с двумя обратными диагоналями. На рис. 6 представлено сравнение времени выполнения параллельных приложений при различных алгоритмах мэппинга. Для каждой из трех тестовых программ использовалось два мэппинга: стандартный (XYZT) и полученный с помощью алгоритма greedy. Время работы параллельной программы со стандартным мэппингом принято за 100%. Из рисунка видно, что для данных классов матриц удалось получить ускорение на 36%, 26% и 56% соответственно для каждого из классов, относительно стандартного мэппинга.

Рис. 6. Сравнение времени выполнения параллельных приложений при различных алгоритмах мэппинга

Нейросетевая модель тестировались на матрицах размера 256×256, а эксперимент для алгоритма мэппинга проводился для программ, запущенных на 256 узлах соответственно.

Заключение

В статье описан разрабатываемый подход к повышению эффективности параллельных приложений. Разработанные в рамках подхода методы позволяют выделить наиболее характерные для заданного приложения паттерны коммуникационного поведения, провести классификацию выделенных паттернов, выбрать лучший алгоритм мэппинга из заданного набора и построить мэппинг. Подход к решению задачи мэппинга на основе использования нейросетевых моделей свертки не имеет аналогов. В настоящее время реализованы основные этапы предложенного подхода: реализован метод построения характерного паттерна для заданного приложения, реализован нейросетевой подход для классификации коммуникационных матриц, предложен новый алгоритм мэппинга для HPC-систем с архитектурой 3-х-мерного тора. Проведенные вычислительные эксперименты с использованием реализованных методов демонстрируют возможность их применения не только для решения задачи мэппинга в рассматриваемой постановке, но позволяют решать вопрос об их применимости и для других задач, связанных с исследованием поведения параллельных приложений на высокопроизводительных системах.

Горизонт применения и дальнейшего развития предложенного подхода широкий. Прежде всего, планируется расширение множества рассматриваемых классов коммуникационных матриц и алгоритмов мэппинга, используемых для предложенной нейросетевой модели. Будет проведено исследование конкретных параллельных приложений с использованием предложенного подхода. Будут уточняться уже реализованные методы.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-07-01053.

Литература

1. Список TOP500. URL: `https://www.top500.org/` (дата обращения: 10.04.2019).
2. Hoefler T., Snir M. Generic topology mapping strategies for large-scale parallel architectures // Proceedings of the international conference on Supercomputing (ICS '11) (Tucson, Arizona, USA, May, 31–June, 04, 2011). ACM, 2011. P. 75–84. DOI: 10.1145/1995896.1995909.
3. Sreepathi S., D'Azevedo E., Philip B., Worley P. Communication Characterization and Optimization of Applications Using Topology-Aware Task Mapping on Large Supercomputers // Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16) (Delft, The Netherlands, March, 12–16, 2016). ACM, 2016. P. 225–236. DOI: 10.1145/2851553.2851575.
4. Wu J., Xiong X., Lan Z. Hierarchical task mapping for parallel applications on supercomputers // The Journal of Supercomputing. 2015. Vol. 71, no. 5. P. 1776–1802. DOI: 10.1007/s11227-014-1324-5.
5. Hoefler T., Jeannot E., Mercier G. An overview of topology mapping algorithms and techniques in high-performance computing // High-Performance Computing on Complex Environment. 2014. P. 75–94. DOI: 10.1002/9781118711897.ch5.
6. Шубин М.В., Попова Н.Н. Анализ поведения параллельных MPI-программ на основе фаз межпроцессного взаимодействия // Суперкомпьютерные дни в России: Труды международной конференции (Москва, 24–25 сентября 2018 г.). Москва: Издательство МГУ, 2018. С. 662–672.

7. Jacobs C., Finkelstein A., Salesin D. Fast multiresolution image querying // Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (SIGGRAPH '95) (Los Angeles, CA, August, 1995). ACM, 1995. P. 277–286. DOI: 10.1145/218380.218454.

8. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. URL: https://arxiv.org/abs/1409.1556 (дата обращения: 10.04.2019).

9. Документация фреймворка Keras. URL: https://keras.io/ (дата обращения: 10.04.2019).

10. Описание приложений NPB. URL: https://www.nas.nasa.gov/publications/npb.html (дата обращения: 10.04.2019).

11. Asanovic K., Bodik R., Catanzaro B., Gebis J., Husbands P., Keutzer K., Patterson D., Plishker W., Shalf J., Williams S., Yelick K. A View of the Parallel Computing Landscape // Communications of the ACM. 2009. Vol. 52, no. 10. P. 56–67, DOI: 10.1145/1562764.1562783.

Попова Нина Николаевна, к.ф.-м.н., доцент, кафедра суперкомпьютеров и квантовой информатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Козлов Михаил Владимирович, студент, кафедра суперкомпьютеров и квантовой информатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Шубин Михаил Витальевич, студент, кафедра суперкомпьютеров и квантовой информатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

DOI: 10.14529/cmse200103

NEURAL NETWORK APPROACH FOR MAPPING OF PARALLEL APPLICATIONS

© 2020 N.N. Popova, M.V. Kozlov, M.V. Shubin

Lomonosov Moscow State Universisty

(GSP-1, Leninskie Gory 1, Moscow, 119991 Russia)

E-mail: popova@cs.msu.su, rat.taurus@gmail.com, mihshub@gmail.com

Received: 12.11.2019

The article is about the problem of improving parallel applications efficiency. It proposes an approach to solve this problem. The approach aims to reduce communication overheads of data exchange between parallel program processes during its execution on high-performance computing system. Growing number of computer nodes leads increasing impact of the communication overhead on application performance. As a sequence the problem of parallel processes allocation to hardware nodes (mapping problem) becomes very actual. The new approach for mapping problem is proposed in this work. The key characteristic of the approach is to extract a communication pattern by phase analysis of application using a convolutional neural network to fast choosing the most relevant mapping algorithm for extracted pattern.

Investigation of results of point-to-point communications between parallel program processes is used to build a communication pattern. The application timeline is broken into equal intervals. Communication patterns are created for each interval. Then Haar 2D wavelet transform is applied to these patterns for generating features. Features are clustered, after that, timeline is splitted into phases. Each phase has its own communication pattern.

The selection of the most relevant mapping algorithm is performed by using convolutional neural network. It supposes some knowledge about different types of parallel applications and most suitable mapping algorithms. This knowledge must be represented by a set of pattern classes (classes of matrices), each class has the mapping algorithm, which fits best for application with this type of pattern. This set can be a training sample for the neural network. Thus the neural network classifies an input application communication pattern and finds a mapping algorithm for the application.

The stages of proposed approach are implemented in this work. This implementation is demonstrated for some tests.

Keywords: high performance computing systems, topology-aware mapping, communication pattern, convolution neural network.

FOR CITATION

Popova N.N., Kozlov M.V., Shubin M.V. Neural Network Approach for Mapping of Parallel Applications. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2020. Vol. 9, no. 1. P. 36–49. (in Russian) DOI: 10.14529/cmse200103.

References

1. TOP500 List. Available at: `https://www.top500.org/` (accessed: 10.04.2019).
2. Hoefler T., Snir M. Generic topology mapping strategies for large-scale parallel architectures. Proceedings of the international conference on Supercomputing (ICS '11) (Tucson, Arizona, USA, May, 31–June, 04, 2011). ACM, 2011. P. 75–84. DOI: 10.1145/1995896.1995909.
3. Sreepathi S., D'Azevedo E., Philip B., Worley P. Communication Characterization and Optimization of Applications Using Topology-Aware Task Mapping on Large Supercomputers. Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16) (Delft, The Netherlands, March, 12–16, 2016). ACM, 2016. P. 225–236. DOI: 10.1145/2851553.2851575.
4. Wu J., Xiong X., Lan Z. Hierarchical task mapping for parallel applications on supercomputers. The Journal of Supercomputing. 2015. Vol. 71, no. 5. P. 1776–1802. DOI: 10.1007/s11227-014-1324-5.
5. Hoefler T., Jeannot E., Mercier G. An overview of topology mapping algorithms and techniques in high-performance computing. High-Performance Computing on Complex Environment. 2014. P. 75–94. DOI: 10.1002/9781118711897.ch5.
6. Shubin M.V., Popova N.N. Study of the behavior of parallel MPI-programs based on phases of interprocess communication. Russian Supercomputing Days: Proceedings of the international conference (Moscow, Russia, September, 24–25, 2018). Moscow, Moscow State University, 2018. P. 662–672. (in Russian)
7. Jacobs C., Finkelstein A., Salesin D. Fast multiresolution image querying. Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (SIGGRAPH '95) (Los Angeles, CA, August, 1995). ACM, 1995. P. 277–286. DOI: 10.1145/218380.218454.
8. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. Available at: `https://arxiv.org/abs/1409.1556` (accessed: 10.04.2019).

9. Keras Documentation. Available at: `https://keras.io/` (accessed: 10.04.2019).

10. NAS Parallel Benchmarks. Available at: `https://www.nas.nasa.gov/publications/npb.html` (accessed: 10.04.2019).

11. Asanovic K., Bodik R., Catanzaro B., Gebis J., Husbands P., Keutzer K., Patterson D., Plishker W., Shalf J., Williams S., Yelick K. A View of the Parallel Computing Landscape. Communications of the ACM. 2009. Vol. 52, no. 10. P. 56–67. DOI: 10.1145/1562764.1562783.