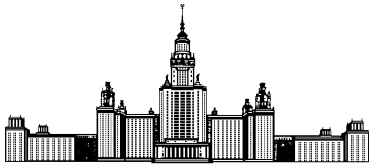
Московский государственный университет имени М.В.Ломоносова



Факультет Вычислительной Математики и Кибернетики
Кафедра Суперкомпьютеров и Квантовой Информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка метода прогнозирования слабой масштабируемости суперкомпьютерных приложений»

Выполнил:
студент 4 курса 423 группы
Мокров Кирилл Сергеевич

Научный руководитель:
к.ф-м.н., ведущий научный сотрудник НИВЦ МГУ
Антонов Александр Сергеевич

Москва, 2020

Содержание

1 Введение

Без параллельных технологий сейчас невозможно обойтись во многих прикладных областях науки: гидро- и аэро- динамике, квантовой химии, сейсмике, компьютерном моделировании лекарств, криптографии и многих других. Это связано с необходимостью обрабатывать большие объёмы данных и производить колоссальное количество вычислений. Что стимулирует создание больших суперкомпьютерных центров, развитие технологий конструирования аппаратных комплектующих, разработку новых методик и алгоритмов. Если используемые пакеты прикладных программ и научные приложения будут плохо написаны, выбранный алгоритм слабо параллелизуем или неоптимально реализован в коде, то они не смогут в полной мере использовать вычислительные ресурсы, предоставляемые высокопроизводительным центром. Если рассмотреть результаты запусков бенчмарков HPL и HPCG на суперкомпьютерах из рейтинга TOP500 [1], то окажется, что для HPL среднее отношение реальной и пиковой производительностей среди всех суперкомпьютеров рейтинга около 63.5%, для HPCG этот показатель сильно ниже - около 1.5% (результаты тестирований HPCG предоставлены только для 70 суперкомпьютеров). Данные бенчмарки - это сверх оптимизированные приложения, написанные большими коллективами математиков и программистов, лишь малая часть программ имеют схожую степень параллелизуемости. В статье [2] приводятся данные с производительностью реальных научных приложений (моделирование распространения сейсмических волн, высокотемпературной плазмы, термоядерного синтеза) на суперкомпьютере Blue Water из Университета Иллинойса. По всем приложениям производительность не превышает 20% от пиковой.

Существует множество характеристик работы параллельных программ, например, время её выполнения, ускорение, эффективность, производительность, количество обращений в память и кэш-промахов. Все эти характеристики имеют динамическую сущность, то есть могут изменяться от запуска к запуску, зависят от параметров запуска программы и машины, на которой она выполняется. Поэтому будем называть их динамическими характеристиками параллельной программы.

Чтобы определить свойства параллельных программ и причины найденных в них особенностей, нужно рассматривать все доступные динамические характеристики на всём пространстве параметров запуска. Эта задача напрямую связана с понятием «масштабируемость», свойством параллельной программы, характеризую-

щим зависимость изменения всей совокупности динамических характеристик работы этой программы от множества параметров её запуска [3].

Здесь и далее обозначим: p - количество процессов, на которых запущено приложение; N - размера задачи; $T_A(N)$ - сложность алгоритма A для значения N размера входа. $T_A(N) = \max_{||y||=N} C_A^T(y)$, где $C_A^T(y)$ сложность алгоритма A для входа y [4]. Сложность алгоритма следует понимать, как последовательную сложность, то есть, как число операций, которое нужно выполнить при последовательном выполнении алгоритма.

Во время исследования масштабируемости приложения необходимо указывать, на какой области изменения значений параметров проводятся запуски. По выбору параметров запуска, которые будут изменяться, масштабируемость согласно [5] можно разделить на три основных типа:

- Сильная масштабируемость (strong scaling) - зависимость динамических характеристик от количества процессов p при фиксированной вычислительной сложности задачи ($T_A(N) = const \Rightarrow N = const$).
- Слабая масштабируемость (weak scaling) - зависимость динамических характеристик от количества процессов p при фиксированной вычислительной сложности задачи в пересчете на один процесс ($T_A(N) \,/\, p = const$).
- Масштабируемость вширь (wide scaling) - зависимость динамических характеристик от размера задачи при фиксированном количестве процессов ($p = const$).

Масштабируемость - ключевая характеристика параллельных программ. Крайне важно учитывать её во время исследования свойств этих программ или в процессе их разработки. Однако не всегда возможно получить в распоряжение большое количество узлов, чтобы увидеть характер изменения различных динамических характеристик приложения с ростом числа используемых ресурсов системы, при увеличении размера задачи. Так же ожидание этого может занять непозволительно много времени, например, авторы статьи [6] утверждают, что в худшем случае время ожидания выделения необходимого количества узлов растёт экспоненциально с ростом количества запрашиваемых ресурсов. Но ведь именно возможность решать за разумное время задачи больших размеров, используя большое количество узлов, является главным преимуществом суперкомпьютерных систем, поэтому необходимо, чтобы приложение было хорошо масштабируемо. Обычно пользователь может выполнить задачу на небольшой конфигурации быстрее, чем запустить на большом

количестве узлов. Исходя из этого, актуальна задача прогнозирования масштабируемости приложения на большие конфигурации вычислительной системы, основываясь только на данных, полученных из множественных запусков на малых конфигурациях. В этой работе предлагается метод решения поставленной задачи в условиях слабой масштабируемости суперкомпьютерных приложений.

2 Постановка задачи

Целью данной работы является разработка метода прогнозирования слабой масштабируемости суперкомпьютерных приложений. Метод должен удовлетворять требованиям универсальности, то есть для своей работы он не должен использовать информацию о коде, алгоритме и системе, на которой производятся запуски.

Для достижения поставленной цели требуется решить следующие задачи:

- Разработать метод, предсказывающий слабую масштабируемость суперкомпьютерных приложений на основе экспериментальных данных.
- Проверить применимость метода на различных приложениях, собрав экспериментальную базу и оценив точность предсказаний.

3 Обзор существующих подходов к предсказанию масштабируемости

Когда говорят о предсказании масштабируемости, то рассматривают не всю совокупность динамических характеристик работы программы, а её часть. Большинство исследований направлены на предсказание времени исполнения программы в зависимости от параметров её запуска. Также существуют работы, рассматривающие предсказание производительности, ускорения, эффективности и энергопотребления. Однако этими характеристиками всё не ограничивается, так, например, в исследовании [7] авторы строят модель, предсказывающую несколько не совсем стандартных характеристик параллельной программы: первая отражает потенциальную потерю эффективности, вызванную различным временем вычислений разных процессов, вторая - неэффективность, вызванную зависимостями в коде, а третья - потерю производительности, вызванную передачей данных.

Большинство работ направлено на исследование и предсказание масштабируемости приложений на суперкомпьютерах, но встречаются и такие, которые используют для запуска параллельных приложений отдельные узлы, небольшие вычислительные системы, грид-системы, платформы для облачных вычислений.

Все исследования можно разделить на две группы: первая объединяется в себе те труды, которые ставят перед собой цель, используя запуски на малых конфигурациях, экстраполировать их результаты на большие; вторая же состоит из тех работ, которые, основываясь на результатах запусков, равномерно распределённых по пространству параметров, пытаются интерполировать их на всё пространство параметров.

Наиболее распространёнными для предсказания масштабируемости являются подходы, использующие аппарат линейной регрессии, методы машинного обучения, а именно нейронные сети, симуляцию исполнения программы и коллаборативную фильтрацию. Все они будут рассмотрены в следующей части этой главы.

3.1 Линейная регрессия

В статье [6] предложены 3 техники предсказания масштабируемости параллельных программ, в основе которых лежит подбор коэффициентов линейной регрессии. Эти техники используют набор запусков приложений на небольшом количестве процессов с разными входными парамерами, чтобы предсказать производительность

на большом количестве процессов. Первая техника является идейно самой простой: результаты тестовых запусков используются для подбора коэффициентов регрессии, а затем с помощью построенной модели результаты экстраполируются на конфигурации большого размера. Вторая и третья техники являются усовершенствованиями первой, они обе рассматривают время вычислений и время коммуникаций отдельно. Отличия в том, что вторая техника основывается на временах вычислений и коммуникаций, полученных от каждого из процессов, выбирая максимальный по времени вычислений процесс и используя его же время коммуникаций, а третья техника основывается на определении времени коммуникаций и вычислений на критическом пути, самой длинной последовательности выполнения программы без блокировок.

В статье рассматривается сильная масштабируемость, где это возможно, то есть, где задача помещается в память при выполнении на малом количестве процессов, или гибрид сильной и слабой масштабируемостей. Строятся предсказания времени выполнения программы на большом количестве процессов, используя несколько запусков той же программы на меньшем количестве. Для оценки качества предсказаний используется относительная ошибка. Из-за этого необходимо применять приближение в логарифмическом масштабе, а с учётом того, что авторами выбрана линейная регрессионная модель, формулу предиктора можно записать в виде:

$$\log_2(T) = \sum_{i=1}^{n} \beta_i \cdot \log_2(x_i) + g(p) + error$$

Где для выражения, объясняющего вклад количества используемых процессов выбирается либо линейное, либо квадратичное представление:

$$g(p) = \gamma_0 + \gamma_1 \cdot \log_2(p) + [\gamma_2 \cdot log_2^2(p)]$$

Несмотря на наличие логарифмов, статистически это всё ещё линейная модель, поскольку она линейна относительно неизвестных параметров.

Для различных приложений медианная ошибка предсказаний изменяется от 1% до 92% в случае с первой техникой и от 7% до 67% для второй и третьей техник.

Построенная модель обладает несколькими недостатками: не всегда возможно разделить время коммуникаций и вычислений, а чтобы это сделать, возможно, необходимо обладать знаниями об исходном коде программы или быть способным его модифицировать, поэтому использование второй и третьей техник не всегда представляется возможным. Также неясно, какой вид для функции $g(q)$ следует выбрать.

Дополнительно авторами метода прогнозирования предполагалось, что исследуемые задачи обладают хорошей вычислительной сбалансированностью, но далеко не все параллельные приложения отвечают этому предположению.

Такой же подход, использование линейной регрессии и логарифмической шкалы, применяется в работе [8]. В ней авторы ставят перед собой задачу разработать модель, позволяющую находить конфигурации запуска так, чтобы они находились на кривой постоянного времени работы приложения, если в качестве пространства параметров конфигураций рассматривать размер задачи и количество процессов, на которых будет запущено приложение. Для построения модели используется информация о том, есть ли связь между параметрами запуска приложения, специфицируется ли при запуске процессорная сетка, позволяющая контролировать распределение данных, и набор запусков приложения на небольшим количестве процессов. Экспериментальная проверка построенной модели на семи суперкомпьютерных приложениях позволила предсказать неизвестные значения параметров так, что медианная ошибка предсказаний получилась меньше 13%.

Подход, предложенный в [9], отличается от всех остальных прежде всего тем, что строятся не просто предсказания значений той или иной динамической характеристики параллельной программы, а предлагается функция, описывающая изменение этой характеристики и зависящая от параметров запуска. Такой подход находит своё применение, когда необходимо оценить асимптотику поведения динамических характеристик, но построение точной, аналитической модели является затруднительной задачей.

На первом этапе работы собирается такая информация, как время исполнения, значения различных аппаратных (количество операций с числами с плавающей запятой) и программных (количество байт, которые MPI функции отправили и приняли) счётчиков производительности. Чтобы обеспечить статистически значимый набор данных о производительности, измерения для одних их тех же конфигураций запуска проводятся несколько раз. Собранные результаты запусков разделяются на две группы: в первой находятся запуски на небольших конфигурациях, на них основывается работа по подбору коэффициентов, во второй - запуски на конфигурациях большего размера, именно на них поведение приложений представляет наибольший интерес, использующиеся для оценки качества предсказаний модели. С помощью профайлера Scalasca в коде программы выделяются фрагменты, так называемые ядра, к которым можно отнести вызовы MPI функций и наиболее вычислительно интенсивные участи кода. Далее, используется регрессия для получения грубой мо-

дели производительности для каждого ядра. В качестве моделей рассматриваются выражения вида:

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p),\ i_k \in I,\ j_k \in J$$

Где I и J заданные заранее множества. Эти модели позднее проходят итеративный процесс уточнения, во время которого методами регрессионного анализа подбираются наиболее оптимальные значения c_k для всевозможных троек $n,\ i_k,\ j_k$, пока качество модели не достигнет точки насыщения. Если степень точности предсказаний недостаточна для получения действенной рекомендации, то исследуемые ядра могут быть дополнительно уточнены с помощью более подробного инструментария. Разработанный подход тестировался на приложениях SWEEP3D, MILC и предсказал асимптотики масштабируемости, в точности согласующиеся с ранее созданными точными моделями. Для приложения HOMME, масштабируемость которого раннее не исследовалась, и поэтому такой точной модели не было, предложенный авторами подход позволил обнаружить плохую масштабируемость одного из ядер, что затем полностью подтвердилось результатами проведённых экспериментов.

Стоит отметить, что множества I и J задаются для каждого приложения по-своему, количество и значения элементов в них устанавливается исходя только из логических соображений, что не удовлетворяет требованиям универсальности, ведь результаты предсказаний напрямую зависят от человека, исследующего характеристики приложения и задающего эти множества.

3.2 Методы машинного обучения

Методы машинного обучения широко применяются для предсказания производительности параллельных приложений, так как построение точной аналитической модели приложения часто является очень затруднительным, а сами модели не способны уловить сложные аспекты взаимодействия между архитектурой суперкомпьютера и исследуемыми программами.

В статье [10] трёхслойная полносвязная нейронная сеть прямого распространения с сигмоидой в качестве функции активации используется для предсказания времени работы приложения SMG2000. В процессе исследований авторы работы сталкиваются с двумя проблемами, из-за которых средняя ошибка предсказаний даже при использовании 10 тысяч запусков, равномерно распределённых по пространству па-

раметров, в качестве обучающего множества, превосходит 15%. Первая из проблем - это значительные шумы в результатах проведённых запусков, проявляющиеся в значительных падениях производительности. Они, например, могут возникать из-за работы операционной системы, использующей ресурсы вычислительного комплекса совместно с потоками исполнения приложения. В качестве способа преодоления этой проблемы авторы выбрали резервирование, как минимум, одного процесса на узле на нужды ОС. Вторая проблема является более серьёзной и не так просто преодолимой. Она заключается в ориентированности алгоритма подбора весов в нейронной сети на минимизацию среднего квадрата ошибки, в то время как качество предсказания оценивается по вычислениям относительных ошибок. Чтобы исправить это, применяются техники стратификации выборки с применением весов и беггинг для обучения ансамблей моделей и дальнейшего усреднения прогнозов. После осуществления вышеописанных изменений качество предсказаний значительно возрастает. Теперь при использовании 500 запусков средняя ошибка предсказаний не превышает 12%, а при увеличении размера обучающей выборки в 5 раз, до 2500 запусков, авторы получают уменьшение средней ошибки до 6,5%, что уже является достаточно высокой точностью.

Применение нейронных сетей не ограниченно предсказаниями производительности на классических HPC системах, так технологии машинного обучения используют в [11] для предсказания времени работы научных приложений, состоящих из нескольких подзадач со сложными внутренними зависимостями, на Grid системах. Моделирование и предсказание времён выполнения таких приложений очень сложно из-за распределения и параллельного исполнения подзадач на столь разнородной системе и динамического поведения общих Grid ресурсов. Ошибка предсказаний рассматриваемой в работе трёхслойной нейронной сети с радиально-базисной функцией, в качестве функции активации, в среднем не превышает 12%. Но стоит отметить, что для обучения потребовалось 10000 запусков рассматриваемых в статье приложений.

Нейронную сеть такой же конфигурации, как в [10], авторы [12] применют для предсказания наиболее оптимальных параметров запусков приложений на SMP системах. В качестве параметров рассматриваются количество процессоров и количество запрашиваемых ядер на одном процессоре. Поиск оптимальной конфигурации необходим, так как использование оптимальных параметров позволяет не только сократить время вычислений и, как следствие, увеличить производительность, но и уменьшить энергопотребление. Проведённые авторами эксперименты показали, что разработанная нейросеть позволяет достичь медианной ошибки предсказаний време-

ни 7.5% (для оценки используются показатели времени на предложенной нейросетью конфигурации и оптимальной конфигурации, найденной полным перебором).

3.3 Симуляция исполнения программы

Подход с использованием симуляции исполнения приложений является наиболее сложным в реализации. Он часто использует в своей работе информацию о структуре программы и о подробных технических характеристиках используемой системы, но, несмотря на это, подобные подходы встречаются.

Так описанный в статье [13] фреймворк FASE (The Fast and Accurate Simulation Environment) позволяет проводит симуляцию исполнений приложений на HPC системах. FASE предоставляет инструментарий для оценки производительности виртуально-прототипированных систем для своевременного и экономически эффективного определения идеальной конфигурации системы для конкретного набора приложений. Работа фреймворка состоит из двух этапов: сначала с помощью встроенных инструментов собирается некоторая характеристика рассматриваемого приложения, далее вводятся параметры системы (тип коммуникационной сети, сетевой протокол, информация об используемых процессорах, оперативной памяти), то есть создаётся некоторая модель системы. Если рассматривается производительность приложения на уже существующей системе и известны некоторые её характеристики, например, пропускная способность или среднее время задержки сети, то добавление этой информации на втором этапе позволит сделать предсказания FASE более точными. Полученные относительные ошибки предсказания времени выполнения приложения в большинстве случаев не превосходят 10% для бенчмарка Sweep3D и 1% для перемножения матриц (корректность проверяется посредством моделирования исполнения приложений на существующих системах и сравнения с результатами реальных запусков на них).

Также существует техника детерминированного воспроизведения приложения, предложенная в [14]. Она лежит в основе работы среды PHANTOM, реализованной авторами статьи. PHANTOM позволяет предсказать производительность и время исполнения приложения на конфигурации с большим количеством процессов, используя множественные запуски того же приложения, но на конфигурациях меньшего размера. Для построения прогноза производительности используется симуляция на основе трассировки. Общую схему работы этой среды можно описать так: вычисления и коммуникации разделяются, собираются их трассы, время по-

следовательных вычислений измеряется для каждой отдельной группы процессов путём детерминированного воспроизведения процесса выполнения приложения. После этого разработанный авторами симулятор SIM-MPI по собранным трассам, по полученному на предыдущем этапе времени вычислений и по параметрам сети целевой системы предсказывает поведение различных операций коммуникации и время работы всего приложения.

Для тестирования PHANTOM использовались шесть традиционных HPC платформ с различными конфигурациями и одна платформа облачных вычислений. Рассматривался большой набор бенчмарков: 6 ядер из NPB, 8 приложений из SPEC MPI2007, ASCI Sweep3D и NWChem. Максимальная относительная ошибка для PHANTOM на классических системах не превосходит 10%, а для облачного сервиса - 7%.

3.4 Коллаборативная фильтрация

Авторы [15] используют подход, основанный на коллаборативной фильтрации и UV факторизации матрицы, для предсказания времени исполнения параллельных приложений на гетерогенных системах. Коллаборативная фильтрация хорошо себя зарекомендовал и широко используется в рекомендательных системах для построения прогнозов. Основная идея предложенного подхода состоит в том, чтобы использовать собранные профили выполненных запусков приложения в качестве набора обучающих данных, а затем построить специальную матрицу для прогнозирования производительности на новой конфигурации. Количество строк в этой матрице равно количеству проведённых тестирований исследуемой программы. Столбцами являются как параметры запуска программы (количество доступной памяти, количество процессов), так и значения динамических характеристик исполнения программы, полученные после её завершения. В качестве прогнозируемых характеристик в работе рассматриваются минимальное, среднее и максимальное времена исполнения приложения на нескольких запусках с идентичными конфигурациями. Чтобы предсказать значения этих характеристик в интересуемой конфигурации, необходимо дополнить матрицу ещё одной строкой, заполнить в ней первые ячейки, отвечающие за параметры запуска, и произвести UV факторизацию полученной матрицы. После перемножения U и V результирующая матрица будет «ближе» всего к изначальной, в качестве меры близости используется среднеквадратичная ошибка, а незаполненные ранее ячейки будут содержать предсказанные значения времён исполнения.

Для запусков приложений использовалась платформа для параллельных вычислений Apache Spark, время исполнения предсказывалось для трёх приложений: Shapelet Finding, Common Neighbor и WordCount. При использовании 70% от всех конфигураций в качестве тестовой выборки среднее значение ошибки по трём приложениям не превосходит 20%, при увеличении количества тестовых конфигураций до 90% значение средней ошибки снижается до 8-12%.

Чтобы предложенный метод был применим, необходимо, чтобы исходная таблица обладала большим количеством строк и столбцом, то есть было проведено много тестирований программы и были доступны большие наборы параметров запуска программы и динамических характеристик исполнения.

3.5 Базовый подход к предсказанию масштабируемости

В работах [16] и [17] рассматриваются все три вида масштабируемости: слабая, сильная и вширь. Предсказание динамических характеристик исполнения строятся для больших задач по результатам исполнения малых задач. В качестве динамической характеристики для всех видов масштабируемости выбрано время выполнения программы. Для построения прогноза в случае сильной масштабируемости и масштабируемости вширь используются экстраполирующие функции:

$T(p) = a \cdot p + b \cdot p^{-1} + c \cdot p^{-0.5}$, для слабой масштабируемости.

$T(s) = b \cdot (1 + a)^s + c$, для масштабируемости вширь.

Независимо от вида функции подбор регрессионных коэффициентов осуществляется с помощью алгоритма Левенберга-Марквардта, итеративного алгоритма оптимизации параметров нелинейных регрессионных моделей. Попытки построить предсказания с помощью подбора экстраполирующей функции в случае слабой масштабируемости не дают приемлемых результатов. Тем не менее, предсказания строятся при помощи объединения методов (поочерёдного применения) предсказаний сильной масштабируемости и масштабируемости вширь. Для пяти рассматриваемых приложений, HPL, NPB (BT, SP, LU) и Graph500, общая медианная относительная ошибка не превышает 23%.

Также в данных работах предложен альтернативный применению логарифмической шкалы из [6] и [8] способ - использование регрессионной модели с весами, где под весом некоторого тестового запуска подразумевается количество его

экземпляров, которые будут входить в итоговый сформированный набор запусков, по которым строится аппроксимация. Добавление весов направленно на устранение двух возможных причин увеличения ошибок предсказания: минимизацию абсолютных ошибок вместо относительных во время регрессии и одинаковую значимость ошибок для всех размеров конфигурации при действительной важности результатов предсказания только для конфигураций большего размера. Данный способ позволил уменьшить средние и медианные относительные ошибки предсказаний для всех рассматриваемых приложений.

Предложенный в следующей главе метод перенимает многие идеи и принципы из работ [16] и [17], но лежащая в его основе математическая модель, с помощью которой строятся предсказания, отличается. Помимо этого, конфигурации запуска для построения предсказаний слабой масштабируемости выбираются строго согласно определению этого вида масштабируемости, такой набор конфигураций позволяет определить динамику изменения рассматриваемых динамических характеристик и, как следствие, построить более точные предсказания.

4 Описание разработанного метода

В работе рассматривается слабая масштабируемость суперкомпьютерных приложений, которая характеризует способность параллельной программы сохранять эффективность распараллеливания при увеличении числа процессоров и одновременном сохранении объема работы, приходящегося на каждый процесс, то есть выполняется соотношение:

$$T_A(N) \,/\, p = const \tag{1}$$

4.1 Экстраполирующая функция и оценка ошибок предсказаний

В качестве метода для построения предсказания из всех рассмотренных ранее выбрана линейная регрессия. Во-первых, модель получается простой, что нельзя сказать о сложно реализуемой симуляции исполнения программы, однако точность предсказаний либо сопоставима, либо даже лучше, чем у остальных методов, во-вторых, чтобы осуществить поиск регрессионных параметров, не требуется большого количества запусков приложения, как, например, для обучения нейронной сети или построения прогноза с помощью коллаборативной фильтрации, в-третьих, для построения точного прогноза нет необходимости использовать исходный код программ и сведения о вычислительной системе, то есть выбранный метод является достаточно универсальным.

С помощью линейной регрессии строятся предсказания значений динамических характеристик работы программы на p_{target} процессах, используя эмпирические данные, полученные из нескольких запусков этой же программы на q процессах, где $q \in \mathbb{Q} = \{q_1, \ldots, q_n\},\ q_1 < q_2 < \ldots < q_n < p_{target}$. Предиктор значения динамической характеристики представляет собой функцию, зависящую от параметров запуска $X = (x_1, x_2, \ldots, x_n)$, определяющих конфигурацию, таких как количества используемых процессов p, размер задачи, конфигурация процессорной сетки, размеры задачи, обсчитываемой локально одним процессом, и других:

$$DF = \hat{DF} + error = F(x_1, x_2, \ldots, x_n, p) + error \tag{2}$$

Где DF и $\hat{DF}$ - полученное из эксперимента и предсказываемое значения динамической характеристики. Предполагается, что функция F зависит от неизвестных регрессионных параметров линейно.

Для нахождения оптимальных значений этих параметров используется метод наименьших квадратов, он минимизирует сумму квадратов ошибок:

$$SSE = |f(w, g(X)) - y|_2 = \sum_{i=1}^{N} (f(w, g_i(X)) - y_i)^2 \to min$$

Что эквивалентно минимизации абсолютной ошибки $error$ из формулы (2).

$$RE = \frac{|DF - \hat{DF}|}{DF} \tag{3}$$

$$RE_{norm} = \frac{|error|}{DF} \tag{4}$$

$$RE_{log} = 2^{|error|} - 1 \tag{5}$$

Однако для оценки качества предсказаний принято использовать относительную ошибку (3). Если использовать введённые ранее обозначения, то её можно переписать в виде (4). Получается, что относительная ошибка зависит не только от минимизируемой с помощью метода наименьших квадратов ошибки, но и от значения динамической характеристики, из-за этого на малом количестве процессов абсолютная ошибка может быть маленькой при большой относительной, а на большом количестве процессов большой при малой относительной ошибке, что никак не может устраивать. Поэтому исходное выражение было модифицировано. Для этого применён такой же, как в [6], подход, а именно приближение аппроксимируемой величины в логарифмическом масштабе. Тогда исходное выражение (2) преобразуется в (6), а формула (4) для вычисления относительной ошибки в (5). Теперь значение относительной ошибки зависит только от минимизируемой абсолютной ошибки, что и являлось целью преобразования.

$$\log(DF) = \log(\hat{DF}) + error \tag{6}$$

Ключевым шагом является параметризация функции $\hat{log(DF)} = log(F(x_1, x_2, \ldots, x_n, p))$. Наиболее точные предсказания удалось получить при

помощи функции вида:

$$\log(\hat{DF}) = \beta_1 \cdot \log(p) + \beta_2 \cdot \log(N) + \beta_3 \cdot \log(p) \cdot \log(N) \quad (7)$$

Здесь за N обозначен размер задачи, а за p количество процессов, на которых эта задача запускается. Несмотря на наличие логарифмов, статистически это всё ещё линейная модель, поскольку она линейна относительно неизвестных параметров, поэтому можно использовать обширную статистическую теорию линейных моделей, в том числе поиск коэффициентов с помощью метода наименьших квадратов.

4.2 Выбор конфигураций и проведение запусков

Для того чтобы исследовать слабую масштабируемость, должно быть заранее известно, какие параметры запуска отвечают за определение размера задачи и как именно сложность работы программы зависит от этих самых параметров. Зная на скольких процессах требуется запустить задачу, сложность программы и используя соотношение (1), можно установить, какие должны быть сложность задачи и определяющие её параметры запуска программы. Тогда процессы из множества $\mathbb{Q}$ вместе с соответствующими им параметрами запуска будем называть тестовыми конфигурациями, а конфигурации, на которых будет оцениваться качество предсказаний метода, - целевыми.

При сборе тестовых данных необходимо прежде всего определить тестовые конфигурации запусков. Для этого нужно узнать, сколько доступно процессов - p_{max}, составить множество $\mathbb{Q} = \{q_{i+1} - q_i = const,\ i = \overline{1, n-1};\ q_n = p_{max}\}$ (необязательно использовать равномерное распределение) и определить соответствующие параметры запуска.

Для нескольких запусков приложения с фиксированными параметрами наблюдается разброс значений динамических характеристик. Это можно объяснить различным размещением процессов на узлах вычислительной системы, различной степенью загруженности коммуникационной сети во время работы приложения. Поэтому необходимо проводить множественные (5-7) запуски приложения для каждой конкретной конфигурации. Чтобы затем, в зависимости от рассматриваемой динамической характеристики, выбрать достигнутый максимум или минимум значения динамической характеристики среди запусков с идентичными параметрами запуска.

В качестве динамических характеристик, для которых строятся предсказание масштабируемости, выбраны время выполнения программы и её производительность. Для двух из пяти выбранных для проверки применимости программ производительность измеряется в GFlops, ещё для одной в MTeps, а для оставшихся двух рассматривается только время выполнения, так как точно посчитать производительность не представляется возможным. Для набора запусков с идентичными конфигурациями выбирается минимум времени исполнения и максимум производительности среди всех запусков набора.

4.3 Общая схема работы метода

I Определение набора тестовых конфигураций.

II Проведение запусков с этими конфигурациями.

III Извлечение из результатов запусков необходимых для поиска неизвестных коэффициентов модели данных, для идентичных конфигураций выбирается минимум времени / максимум производительности исполнения.

IV Использование метода наименьших квадратов для подбора коэффициентов линейной регрессии.

V Построение предсказаний значений динамической характеристики для заданного множества целевых запусков с помощью построенной модели.

5 Экспериментальная проверка применимости метода на суперкомпьютере

Применимость метода оценивалась с помощью запусков приложений на суперкомпьютере «Ломоносов-2». Вычислительные узлы которого включают процессоры Intel Haswell-EP E5-2697v3 (2.6 GHz), оборудованы 64 GB памяти и связаны коммуникационной сетью InfiniBand FDR [18].

В качестве приложений для тестирования использовались реализации HPL, HPCG, алгоритмов матричного умножения (DNS и SUMMA), Graph500. Так как рассматривается слабая масштабируемость, то тестирования проводились с конфигурациями запуска, удовлетворяющими выражению (1). Для того чтобы более полно проверить применимость метода на реальных суперкомпьютерных приложениях и оценить их возможности к слабой масштабируемости, для некоторых из приложений тестирования проводились с несколькими наборами конфигураций запуска (HPL - 3 различных набора; Graph500, DNS - 2 различных набор; HPCG, SUMMA - 1 набор).

Используемые обозначения в таблицах и рисунках:

- PN - количество процессов, на которых производится запуск задачи;
- PS - размер задачи;
- SC - scalefactor;
- EF - edgefactor;
- RE_time - относительная ошибка предсказания времени;
- RE_perf - относительная ошибка предсказания производительности.

Размер задачи во всех таблицах и рисунках измеряется в тысячах единиц, кроме HPCG и Graph500.

5.1 HPL

HPL (High Performance Computing Linpack Benchmark) — тест производительности вычислительной системы, на основе результатов которого формируется современный список TOP500 [1] лучших суперкомпьютеров в мире. Суть теста заключается в решении плотных систем линейных алгебраических уравнений, используя LU факторизацию, на системах с распределённой памятью.

HPL имеет сложность алгоритма $\mathcal{O}(N^3)$, а количество операций чтения/записи пропорционально $\mathcal{O}(N^2)$. То есть, если размер задачи увеличить в два

	C_1		C_2		C_3	
№	PN	PS	PN	PS	PN	PS
1	6	18	4	20	4	25
2	8	20	6	23	9	33
3	12	23	8	25,3	12	36,4
4	16	25	10	27,35	16	40
5	20	27	12	28,9	25	46,4
6	27	30	15	31,1	35	52
7	40	34	20	34,35	42	55,2
8	42	35	25	36,82	49	58,1
9	50	37	30	39,1	56	60,7
10	60	39	36	41,6	64	63,5
11	64	40	49	46,1	81	68,7
12	80	43	64	50,4	110	76,1
13	90	45	70	51,95	121	78,5
14	98	46	80	54,3	144	83,2
15	110	48	81	54,5	169	87,8
16	125	50	121	62,3	182	90
17	140	52	144	66,05	196	92,2
18	156	53,85	169	69,6	210	94,4

Таблица 1: Тестовые конфигурации запусков HPL для трёх различных значений констант

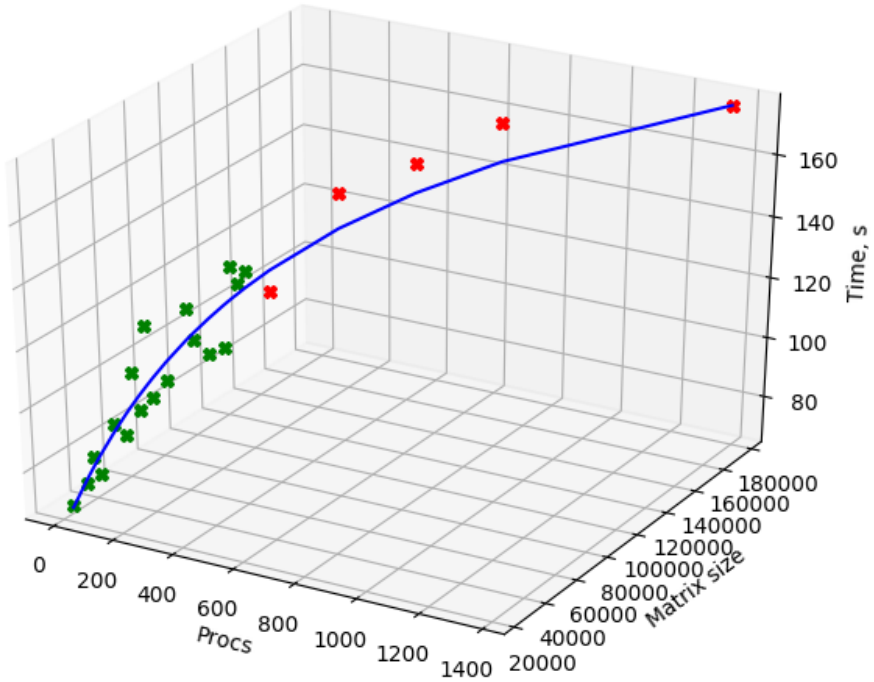


Рис. 1: Аппроксимирующая функция предсказания времени HPL, конфигурации соответствуют константе C_3

	C_1			
№	PN	PS	RE_time	RE_perf
1	225	60,8	7,58	6,82
2	400	73,7	1,79	1,90
3	576	83,2	0,37	0,45
4	784	92,2	0,12	0,09
5	1369	111	0,16	0,07
	C_2			
№	PN	PS	RE_time	RE_perf
1	225	76,65	3,82	0,82
2	400	92,85	7,53	16,35
3	576	104,8	0,62	9,25
4	784	116,1	1,42	10,59
5	1369	140	11,35	4,41
	C_3			
№	PN	PS	RE_time	RE_perf
1	225	100,8	5,79	5,69
2	400	117	7,73	7,73
3	576	132	6,06	5,72
4	784	146,3	7,42	6,95
5	1369	176,2	0,02	1,69

Таблица 2: Целевые конфигурации запусков HPL для трёх различных значений констант и значения относительных ошибок предсказания времени и производительности на этих конфигурациях

раза, то количество операций с плавающей запятой увеличится в восемь раз, в то время, как количество операций с памятью увеличится только в четыре раза. Подобные значения асимптотик свойственны приложениям с большим количеством вычислений над плотными структурами данных, такие приложения часто используют в своей работе графические ускорители.

Тестовые конфигурации запуска, используемые для вычислений параметров модели, и целевых конфигураций, необходимые для оценки погрешности предсказаний, приведены в таблицах (1) и (2) соответственно. Коэффициенты C_1, C_2, C_3, определяющие значение отношения количества работы приходящееся на один процесс и количества используемых процессов из выражения (1) связаны соотношением: $4 \cdot C_1 = 2 \cdot C_2 = C_3$, то есть с увеличением на единицу номера коэффициента количество работы на один процесс увеличивается в два раза.

Относительные ошибки для всех целевых конфигураций слабой масштабируемости HPL представлены в таблице (2). Значения относительных ошибок предсказа-

ния времени по всем целевым конфигурациям варьируются от 0,02% до 11,35%, среднее значение - 4,12%, медиана - 3,82%, а производительности - от 0,07% до 16,35%, среднее значение - 5,23%, медиана - 5,69%. Если рассматривать более детально, как меняются относительные ошибки с увеличением числа используемых процессов и размера задачи, усредняя значение ошибок по конфигурациям с одинаковым количеством используемых процессов, 225 - 5,09%, 400 - 7,17%, 576 - 3,74%, 784 - 4,43%, 1369 - 2,95%, то можно однозначно сказать, что увеличение конфигурации не приводит к росту значений относительных ошибок. Что говорит об отсутствии необходимости увеличивать количество тестовых конфигураций и, самое главное, способности предложенного метода предсказывать значения различных динамических характеристик на конфигурациях в 6-7 раз больших, чем самые большие тестовые. Получающуюся аппроксимирующую функцию предсказания времени для одной из констант можно видеть на рисунке (1).

5.2 HPCG

PN	PS	Parameter i
$14 \cdot i$	$PN \cdot 104^3$	Тестовые: $i \in \{1, \ldots, 15\}$
		Целевые: $i \in \{20, 40, 50, 70, 100\}$

Таблица 3: Тестовые и целевые конфигурации запусков HPCG

PN	PS	RE_time	RE_perf
280	$PN \cdot 104^3$	0,02	0,37
560		1,56	0,80
700		1,89	13,07
980		2,85	19,54
1400		7,05	8,99

Таблица 4: Относительные ошибки предсказания времени и производительности HPCG

HPCG (High Performance Conjugate Gradients Benchmark) был разработан, чтобы стать альтернативной HPL метрикой оценки производительности суперкомпьютеров. HPCG сильно выделяется на фоне остальных приложений, так как и сложность последовательного алгоритма, и количество операций чтения/записи бенчмарка пропорциональны $\mathcal{O}(N)$. В тесте преобладают нерегулярный доступ к памяти и мелкоструктурные рекурсивные вычисления, которые, как утверждают авторы бенчмарка, свойственны многих научным вычислительным приложениям [19]. Основное

вычислительное ядро HPCG занимается решением СЛАУ с разреженной положительно определённой симметричной матрицей с помощью метода сопряжённых градиентов.

Тестовые и целевые конфигурации представлены в таблице (3), а получившиеся относительные ошибки предсказаний в таблице (4). Несмотря на то что в двух случая значения относительных ошибок сильно отличаются от остальных и почти достигают 20%, среднее имеет меньшее значение, оно составляет 2,674% для предсказания времени и 8,554% для производительности.

5.3 Алгоритмы матричного умножения

5.3.1 SUMMA

№	PN	PS
1	1	4
2	4	6,4
3	9	8,4
4	16	10
5	25	11,5
6	36	13,2
7	49	14,7
8	64	16
9	81	17,1
10	100	18,5
11	121	19,8
12	144	21
13	169	22,1
14	196	23,1

Таблица 5: Тестовые конфигурации запусков матричного умножения по алгоритму SUMMA

Первый из двух рассматриваемых алгоритмов матричного умножения - SUMMA(Scalable Universal Matrix Multiply)[20]. Различные реализации этого алгоритма используются такими библиотеками, как ScaLAPACK и PBLAS. Он, так же как и HPL, имеет сложность $\mathcal{O}(N^3)$ и количество операций чтения/записи пропорционально $\mathcal{O}(N^2)$. Тестовые и целевые конфигурации запуска приведены в таблицах (5) и (6) соответственно. Для тестирования использовалась реализация, требующая квадратной процессорной сетки, однако в общем случае это не является обязательным условием для данного алгоритма. Коротко алгоритм можно записать так:

№	PN	PS	RE_time
1	225	25,6	1,95
2	400	30	3,94
3	576	33,6	3,01
4	784	37,8	7,59
5	1024	40	1,39

Таблица 6: Целевые конфигурации запусков матричного умножения по алгоритму SUMMA и значения относительных ошибок предсказания времени на этих конфигурациях

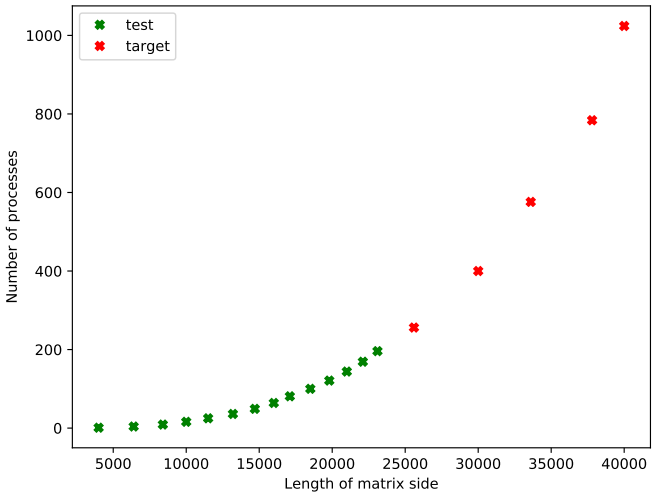


Рис. 2: Конфигурации запусков матричного умножения по алгоритму SUMMA

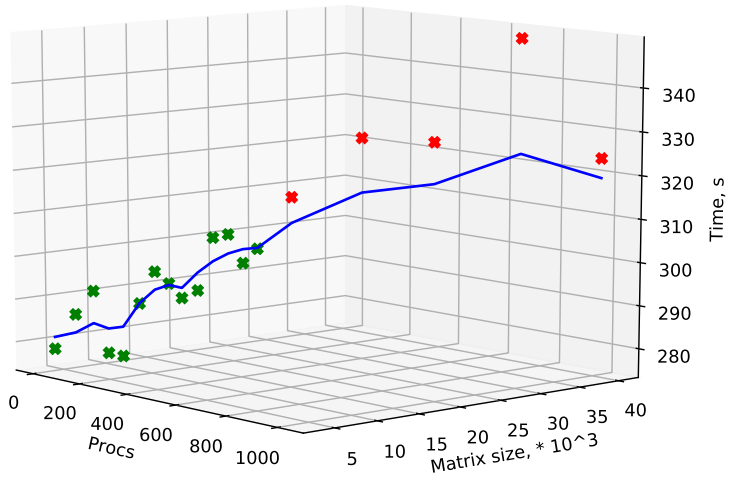


Рис. 3: Аппроксимирующая функция предсказания времени матричного умножения по алгоритму SUMMA

$C_{ij} = 0$
for $l = \overline{0, k-1}$ **do**
broadcast a_i^l within my row
broadcast b_l^j within my column
$C_{ij} = C_{ij} + a_i^l \cdot b_l^{jT}$
end for

Из-за того что строятся предсказания слабой масштабируемости приложений, то отношение сложности алгоритма к количеству процессов должно быть постоянным. Так как известно, что данный алгоритм имеет кубическую сложность, то можно в явном виде написать, как количество процессов зависит от размера задачи $T_A(N)$ / $p = const \Rightarrow p = N^3$ / $const$. Как видно по графику (2), конфигурации запусков выбираются строго согласно этому выражению.

Несмотря на сложную, скачкообразную зависимость времени работы программы (3) предложенному методу удаётся установить основной характер изменения времени выполнения и построить предсказание так, что среднее значение относительной ошибки равно 3,58%, а медиана - 3,01%, значения относительных ошибок для всех целевых конфигураций приведены в таблице (6).

5.3.2 DNS

Следующий из рассматриваемых алгоритмов матричного умножения - DNS. Для его работы необходимо, чтобы количество процессов было равно кубу некоторого натурального числа. Из этих процессов создаётся 3D решётка. Перемножаемые матрицы считываются нижним слоем решётки, делятся на блоки и рассылаются специальным образом остальным процессам. Далее происходят локальное перемножение полученных блоков матриц и пересылка получившихся значений на нижний слой, из которых там собирается результирующая матрица.

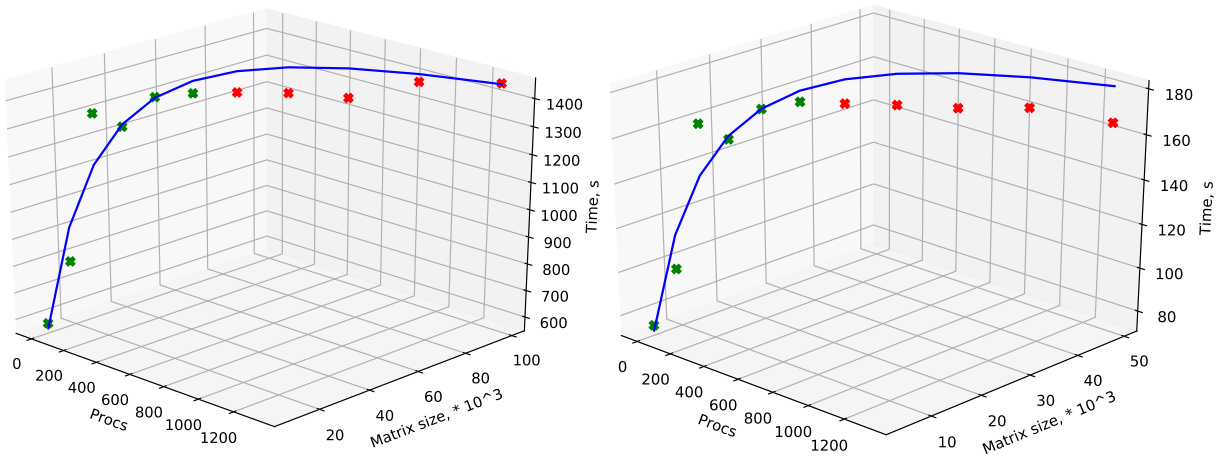
Из-за ограничения, накладываемого алгоритмом на количество используемых для запуска процессов, тестовая выборка получается достаточно маленькая, всего 6 различных конфигураций. Но, несмотря на это, среднее значение относительной ошибки составляет всего 6,43%. Получившиеся аппроксимирующие функции для двух рассматриваемых наборов конфигураций представлены на рисунках (4). С некоторого момента, при увеличением размера конфигурации, независимо от рассматриваемого набора, время работы алгоритма почти прекращает расти. Это говорит о его

	C_1		C_2	
№	PN	PS	PN	PS
1	1	4,5	1	9
2	8	9	8	18
3	27	13,5	27	27
4	64	18	64	36
5	125	22,5	125	45
6	216	27	216	54

Таблица 7: Тестовые конфигурации запусков матричного умножения по алгоритму DNS для двух различных значений констант

	C_1			C_2		
№	PN	PS	RE_time	PN	PS	RE_time
1	343	31,5	6,55	343	63	5,66
2	512	36	8,42	512	72	6,84
3	729	40,5	9,35	729	81	7,85
4	1000	45	7,94	1000	90	1,94
5	1331	49,5	9,63	1331	99	0,19

Таблица 8: Целевые конфигурации запусков DNS для двух различных значений констант и значения относительных ошибок предсказания времени и производительности на этих конфигурациях



(a) C_1 (b) C_2

Рис. 4: Аппроксимирующие функции предсказания времени для двух различных значений констант, DNS

хорошей слабой масштабируемости, а небольшое увеличение времени работы можно связать с возрастающими накладными расходами на коммуникации.

5.4 Graph500

	C_1			C_2		
№	PN	SC	EF	PN	SC	EF
1	14	25	16	14	26	16
2	28	26	16	28	27	16
3	42	26	25	42	27	25
4	56	27	16	56	28	16
5	70	27	20	70	28	20
6	84	27	25	84	28	25
7	98	27	29	98	28	29
8	112	28	16	112	29	16
9	126	28	18	126	29	18
10	140	28	20	140	29	20
11	154	28	22	154	29	22
12	168	28	25	168	29	25
13	182	28	27	182	29	27
14	196	29	14	196	30	14
15	210	29	15	210	30	15

Таблица 9: Тестовые конфигурации запусков бенчмарка Graph500 для двух различных значений констант

Последнее из рассматриваемых приложений - Graph500. Это ещё одна альтернатива HPL, помимо уже ранее рассмотренного бенчмарка HPCG, по результатам работы Graph500 также составляется рейтинг суперкомпьютеров. Однако этот бенчмарк, в отличии от всех ранее рассмотренных приложений, больше нагружает коммуникационную подсистему компьютера и не так сильно зависит от количества исполняемых операций над числами с плавающей запятой в секунду. Подобное поведение свойственно многим задачам из области графовой аналитики. Бенчмарк Graph500 включает в себя два графовых алгоритма: поиск в ширину (BFS) и поиск кратчайшего пути от одной вершины (SSSP). Из-за того, что в основе лежит работа с графами, то производительность измеряется не в GFlops, а в MTeps (количество пройденных дуг в секунду).

Сложность алгоритма, в случае работы с графами, выражается через число его вершин и рёбер. Так для Graph500 это $\mathcal{O}(V + E)$, где V- количество вершин, а E

	C_1						
	PN	SC	EF	RE_time(BFS)	RE_perf(BFS)	RE_time(SSSP)	RE_perf(SSSP)
1	280	30	20	3,23	4,20	6,40	7,06
2	560	31	20	5,09	2,03	17,62	17,67
3	700	31	26	12,94	8,82	5,24	9,99
4	980	31	36	20,93	19,92	0,61	5,76
5	1400	32	26	15,59	8,54	5,29	12,98
	C_2						
№	PN	SC	EF	RE_time(BFS)	RE_perf(BFS)	RE_time(SSSP)	RE_perf(SSSP)
1	280	29	20	12,42	12,03	16,87	15,05
2	560	30	20	5,08	7,86	25,16	21,94
3	700	30	26	5,74	0,50	27,14	24,63
4	980	30	36	25,80	27,76	8,41	11,22
5	1400	31	26	26,45	24,55	19,61	21,93

Таблица 10: Целевые конфигурации запусков бенчмарка Graph500 для двух различных значений констант и значения относительных ошибок предсказания времени и производительности на этих конфигурациях

- рёбер. Размер задачи задаётся двумя параметрами $scale - SC$ и $edgefactor - EF$: количество вершин графа равно 2^{SC}, а рёбер $EF \cdot 2^{SC}$. Используя эти два параметра для вычисления сложности работы алгоритма, можно получить:

$$V + E = 2^{SC} + EF \cdot 2^{SC} = 2^{SC} \cdot (1 + EF)$$

Во время тестирования этого приложения становится ясно видна основная проблема предсказывания слабой масштабируемости приложений: конфигурации запуска выбираются согласно отношению (1), но не всегда возможно обеспечить строгое равенство, поэтому приходится округлять некоторые параметры запуска. Из-за этого возможны провалы или всплески показателей рассматриваемых динамических характеристик, которые мешают построению точных предсказаний, так как линейная регрессия неустойчива к выбросам. сильно портят качество предсказаний. В случае Graph500 приходилось округлять значения SC и EF, так как они могут принимать только целочисленные значения. В связи с этим средние значения относительных ошибок предсказания времени и производительности для этого приложения составляют около 13%.

6 Заключение

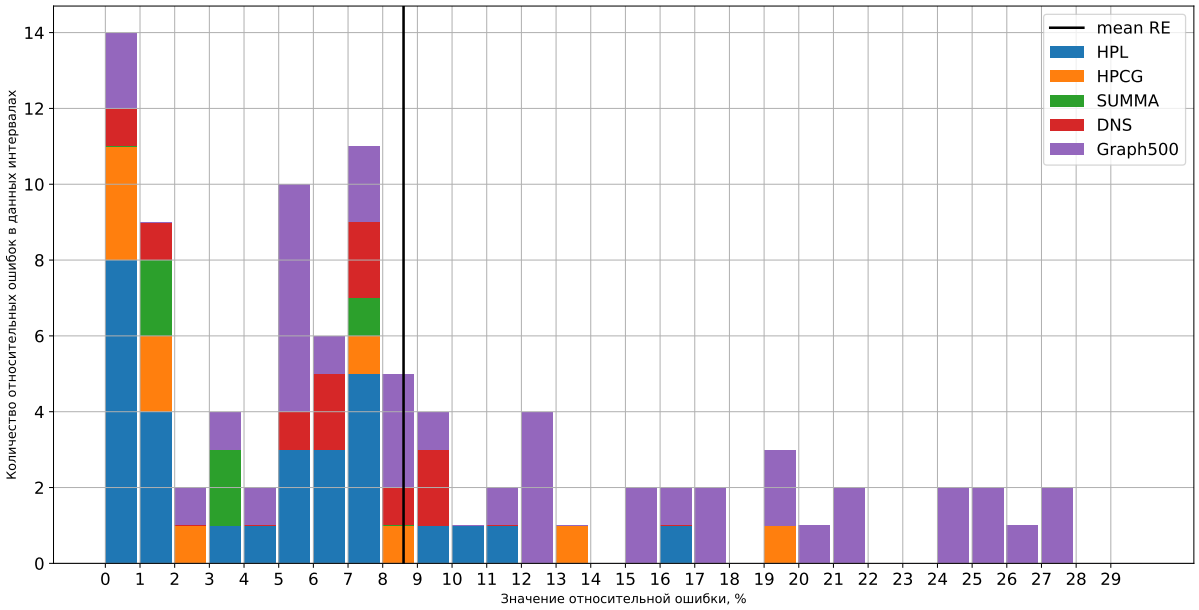


Рис. 5: Относительные ошибки предсказаний по всем рассматриваемым приложениям

В данной работе были получены следующие основные результаты:

- Разработан метод, предсказывающий слабую масштабируемость суперкомпьютерных приложений на основе экспериментальных данных со средней относительной ошибкой по всем смотренным приложениям равной 8,6%.
- Выполнена экспериментальная проверка метода с помощью запусков на суперкомпьютере «Ломоносов-2» на примере HPL, HPCG, алгоритмов матричного умножения (SUMMA и DNS), Graph500.

На основании предложенного метода удалось построить предсказания слабой масштабируемости для всех рассматриваемых приложений так, что максимальная относительная ошибка предсказания значения динамической характеристики не превышает 28%, однако подобные ошибки, как видно, достаточно редки. Средние значения относительных ошибок для различных приложений: HPL - 4,9%, HPCG - 5,6%, SUMMA - 3,6%, DNS - 6,4%, Graph500 - 13,2%. Средняя относительная ошибка по всем приложениям составляет 8,6%. Гистограмма со всеми ошибками предсказаний представлена на рисунке (5). Таким образом, предложенный метод строит предсказания так, что получаемые относительные ошибки, сравнимы с ошибками предсказания у существующих подходов при сопоставимых размерах конфигураций предсказыва-

емых запусков. Но разработанный метод отличается от них простотой построения, отсутствием необходимости собирать большой набор тестовых данных и способностью строить предсказания, не используя информацию о коде, алгоритме и системе, на которой производятся запуски, то есть он является достаточно универсальным.

7 Список литературы

[1] The 54nd edition of the TOP500 list [Электронный ресурс]. – Электрон. дан. – URL: https://www.top500.org/lists/2019/11/. (дата обращения 16.03.2020).

[2] Leland, Robert & Ang, Jim & Barnette, Daniel & Benner, Bob & Goudy, Sue & Malins, Bob & Rajan, Mahesh & Vaughan, Courtenay & Black, Amalia & Doerfler, Doug & Domino, Stefan & Franke, Brian & Ganti, Anand & Laub, Tom & Meyer, Hal & Scott, Ryan & Stevenson, Joel & Sturtevant, Judy & Taylor, Mark. (2016). Performance, Efficiency and Effectiveness of Supercomputers.

[3] Antonov, Alexander & Teplov, Alexey. (2016). Generalized Approach to Scalability Analysis of Parallel Applications. 291-304. 10.1007/978-3-319-49956-7_23.

[4] Абрамов С.А. Лекции о сложности алгоритмов - М.: МЦНМО, 2012. - 245 с.

[5] Антонов, А. С., & Теплов, А. М. (2013). Исследование масштабируемости программ с использованием инструментов анализа параллельных приложений на примере модели атмосферы Nh3d. Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2 (1), 5-16.

[6] Barnes, B. J., Reeves, J., Rountree, B., De Supinski, B., Lowenthal, D. K., & Schulz, M. (2008). A regression-based approach to scalability prediction. In ICS'08 - Proceedings of the 2008 ACM International Conference on Supercomputing (pp. 368-377). (Proceedings of the International Conference on Supercomputing). https://doi.org/10.1145/1375527.1375580

[7] Rosas, C., Giménez, J., & Labarta, J. (2014). Scalability prediction for fundamental performance factors. Supercomputing Frontiers And Innovations, 1(2), 4-19. doi:http://dx.doi.org/10.14529/jsfi140201

[8] Barnes, Brad & Garren, Jeonifer & Lowenthal, David & Reeves, Jaxk & Supinski, Bronis & Schulz, Martin & Rountree, Barry. (2010). Using focused regression for accurate time-constrained scaling of scientific applications. Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010. 1 - 12. 10.1109/IPDPS.2010.5470431.

[9] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf. 2013. Using automated performance modeling to find scalability bugs in complex codes. In

Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). Association for Computing Machinery, New York, NY, USA, Article 45, 1–12. DOI:https://doi.org/10.1145/2503210.2503277

[10] Ipek E., de Supinski B.R., Schulz M., McKee S.A. (2005) An Approach to Performance Prediction for Parallel Applications. In: Cunha J.C., Medeiros P.D. (eds) Euro-Par 2005 Parallel Processing. Euro-Par 2005. Lecture Notes in Computer Science, vol 3648. Springer, Berlin, Heidelberg

[11] Nadeem, Farrukh & Alghazzawi, Daniyal & Mashat, Abdulfattah & Fakieh, Khalid & Almalaise, Abduallah & Hagras, Hani. (2017). Modeling and predicting execution time of scientific workflows in the Grid using radial basis function neural network. Cluster Computing. 20. 2805–2819. 10.1007/s10586-017-1018-x.

[12] Singh, Karan & Curtis-Maury, Matthew & McKee, Sally & Blagojevic, Filip & Nikolopoulos, Dimitrios & Supinski, Bronis & Schulz, Martin. (2010). Comparing Scalability Prediction Strategies on an SMP of CMPs. 6271. 143-155. 10.1007/978-3-642-15277-1_14.

[13] Grobelny, Eric & Bueno, David & Troxel, Ian & George, Alan & Vetter, Jeffrey. (2007). FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications. Simulation. 83. 10.1177/0037549707084939.

[14] Zhai, Jidong & Chen, Wenguang & Zheng, Weiming & Li, Keqin. (2015). Performance Prediction for Large-Scale Parallel Applications Using Representative Replay. IEEE Transactions on Computers. 65. 1-1. 10.1109/TC.2015.2479630.

[15] Shao, Qingshi & Li, Pan & Liu, Shijun & Liu, Xinyan. (2017). A collaborative filtering based approach to performance prediction for parallel applications. 331-336. 10.1109/CSCWD.2017.8066716.

[16] К.П. Казьмина, А.С. Антонов. Разработка методов прогнозирования масштабируемости приложений на конфигурации суперкомпьютеров // Вестник компьютерных и информационных технологий. N 12, 2018. С. 45-56.(http://www.vkit.ru/index.php/current-issue-rus/770-045-056) DOI:10.14489/vkit.2018.12.pp.045-056

[17] Pavel Valkov, Kristina Kazmina, and Alexander Antonov. Using Empirical Data for Scalability Analysis of Parallel Applications // Communications in Computer and Information Science. Vol. 1063. 2019. Pp. 58-73. DOI:10.1007/978-3-030-28163-2_5

[18] Voevodin, V., Antonov, A., Nikitenko, D., Shvets, P., Sobolev, S., Sidorov, I., Stefanov, K., Voevodin, V., & Zhumatiy, S. (2019). Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers And Innovations, 6(2), 4-11. doi:http://dx.doi.org/10.14529/jsfi190201

[19] Report, S., Dongarra, J., & Heroux, M.A. (2013). Toward a New Metric for Ranking High Performance Computing Systems.

[20] Robert A. van de Geijn and Jerrell Watts. 1995. SUMMA: Scalable Universal Matrix Multiplication Algorithm. Technical Report. University of Texas at Austin, USA.