Содержание

Введение

Описание проблемы

В настоящее время существуют такие системы параллельного программирования, как Chapel [1], Charm++ [2], LuNA, MPI. Эти системы предназначены для упрощения реализаций прикладных задач на мультикомпьютерах. Качество реализации (свойство соблюдения нефункциональных требований) таких систем напрямую влияет на *эффективность* исполнения прикладных задач, т.е. на общее время исполнения и потребление ресурсов вычислителя. Поскольку вычислительное время мультикомпьютера стоит дорого, стоят две проблемы: прогнозирование времени исполнения прикладной задачи и оптимизация (уменьшение) этого времени. В частности, такие проблемы стоят в системе исполнения фрагментированных программ LuNA.

Одним из основных способов определения эффективности ПО является использование систем тестирования. Имея средства тестирования, позволяющие определять эффективность систем параллельного программирования, как правило, можно прогнозировать время исполнения заданного класса прикладных задач, а также найти «узкие места» самой системы и конкретных прикладных программ, требующие доработки. Таким образом, целесообразна разработка такой системы тестирования.

Система исполнения LuNA предназначена для исполнения на мультикомпьютере, из чего следует невозможность определения состояния всей системы целиком, имеется возможност определять только состояния отдельных вычислительных узлов. В системе используется определённая модель вычислений, диктуемая фрагментированной парадигмой, учёт особенностей которой позволяет проводить специфичные для этой системы исполнения тесты, более информативные, чем при стандартных методах тестирования.

Обзор

Рассмотрим существующие системы тестирования производительности систем параллельного программирования.

Система параллельного программирования Charm++, разработанная в Иллинойсском университете, упрощает разработку параллельных прикладных приложений для систем MIMD [2]. Система исполнения Charm++ предоставляет возможность журналирования информации о событиях во время исполнения прикладной программы. Примерами таких событий являются начало и конец исполнения задач или отправка сооюбщений. У каждого процессора имеется свой буфер журнала сообщений, который записывается в файл к окончанию исполнения прикладной программы. Эти файлы предназначены для анализа производительности после исполнения прикладной программы, используя средство визуализации Projections for Charm++ [3]. В текущем виде средство визуализации ориентируется на ручной анализ человеком. Средство поддерживает множество различных видов предоставления информации, таких как графики использования ресурсов, гистограммы, графики времени.

HPC Challenge Benchmark [4] — набор тестов производительности, предназначенный для оценки производительности суперкомпьютеров. Он состоит из следующих тестов:

1. HPL (High-Performance Linpack Benchmark) — тест производительности работы с числами с плавающей точкой. В процессе исполнения замеряется время решения систем линейных уравнений.

2. DGEMM — тест производительности работы с числами с плавающей точкой. В процессе исполнения замеряется время умножения матриц.

3. STREAM — тест, оценивающий скорость работы с памятью.

4. PTRANS (parallel matrix transpose) — тест коммуникационной подсистемы, в котором пары процессоров взаимодействуют между собой одновременно.

5. RandomAccess — тест произвольного доступа к оперативной памяти, в котором измеряются так называемые GUPS, Giga Updates Per Second.

6. FFT — тест быстрого преобразования Фурье.

7. b_{eff} (effective bandwidth benchmark) — тест производительности сетевой подсистемы. Алгоритм тестирования учитывает то, что в реальные прикладные приложения могут отсылать как короткие, так и длинные сообщения. Варьируются размеры сообщения и

коммуникационные паттерны (топологии): кольца разного размера и случайный многоугольник. Результатом данного теста является одно числовое значение.

Особенностью тестов является возможность замены исходного кода тестов, например, для дополнительной оптимизации под конкретный суперкомпьютер или для определения качества ПО, использующегося в замене. В частности, существует реализация этого набора тестов для системы Chapel [5].

Также существуют также профилировщики общего назначения, такие как GNU gprof [6] и Intel VTune [7]. Существенным недостатком профилировщиков общего назначения является отсутствие учёта специфики тестируемого приложения и неприспособленность к тестированию распределённых систем, однако их можно использовать как вспомогательные инструменты во время тестирования системы.

Отдельным подклассом профилировщиков общего назначения является профилировщики с ручным инструментированием (manual instrumental profiling). Примерами таких профилировщиков являются CxxProf [8] и GPTL [9]. Такие профилировщики можно использовать при реализации системы тестирования.

Из приведённого обзора можно заключить, что существующих решений не достаточно для решения описанной проблемы в полной мере.

Цель работы

Целью работы является разработка и реализация системы комплексного тестирования, учитывающей специфику LuNA.

В состав работ входит:

1. Анализ проблемы и определение ключевых характеристик системы LuNA, влияющих на производительность.
2. Создание методики тестирования для определения значений этих характеристик. Методика должна описывать набор конкретных приёмов для реализации тестов и проведения тестирования.
3. Реализация тестов и проведение тестирования.
4. Анализ результатов тестирования и получение выводов.

Новизна определяется ограничениями, накладываемыми спецификой системы исполнения LuNA.

Глава 1

Необходимые определения

1.1. Технология фрагментированного программирования

Технология фрагментированного программирования — это набор методов и инструментальных средств, предназначенных для эффективной реализации больших численных моделей на суперкомпьютерах, в первую очередь — вычислительных кластерах [10]. Целью технологии является исключение параллельного программирования из процесса параллельной реализации численных моделей, которое заменяется на последовательное программирование отдельных частей программы (фрагментов) и описание схемы соединения этих частей друг с другом. Из этого описания автоматически генерируется параллельная программа для суперкомпьютера. Эффективность сгенерированной программы обеспечивается наложением прямого управления (control-flow) на исходно непроцедурное (data-flow) описание вычислений, а также отображением данных на ограниченные ресурсы суперкомпьютера. Для многих численных методов, в числе которых итерационные методы на регулярных сетках, матрично-векторные операции и метод частиц-в-ячейках, прямое управление и отображение данных на ресурсы могут быть сконструированы автоматически. Существенной особенностью подхода технологии фрагментированного программирования является использование исполнительной системы, которая обеспечивает динамические свойства исполнению программы, такие как динамическая балансировка нагрузки на вычислительные узлы.

1.2. Система LuNA

LuNA (Language for Numerical Algorithms) — язык программирования и исполнительная система для параллельной реализации больших численных моделей на суперкомпьютерах.

Текущая реализация LuNA состоит из компилятора `lc` (LuNA compiler) и исполнительной системы `rts` (runtime system). Компилятор преобразует исходный код на языке LuNA во внутреннее представление в формате JSON, который исполняет исполнительная система.

В системе LuNA прикладной алгоритм представляется в виде фрагментированной программы. Исполнением прикладной программы на мультикомпьютере с распределённой памятью занимается система исполнения LuNA. Фрагментированная программа представляет собой описание множества фрагментов данных (ФД) и множества фрагментов вычислений (ФВ). Фрагмент данных – это переменная алгоритма. Он представлен в виде блока памяти. Фрагмент вычислений – исполняемая единица, имеющая в качестве входных и выходных значений заданные фрагменты данных. В системе LuNA существует два вида фрагментов вычислений: атормарные и структурированные. Атомарные фрагменты представляют собой подпрограмму на языке C++, написанную прикладным программистом. Структурированные фрагменты вычислений соответствуют управляющим конструкциям языка LuNA (циклы, условные операторы, вызовы подпрограмм) [11]. В частности, цикл for из N итераций, при его исполнении, раскрывается в N новых фрагментов вычислений, а вызов фрагментированной подпрограммы раскрывается в фрагменты вычислений, соответствующие инструкциям в исходном коде подпрограммы.

Исполнительная система реализована на языке C++ и состоит из следующих основных модулей:

- **Менеджер фрагментов вычислений.** Осуществляет управление параллельным исполнением подзадач, распределением подзадач по узлам и определением порядка вычислений. Благодаря менеджеру фрагментов вычислений, система LuNA позволяет освободить прикладного программиста от необходимости реализации механизма балансировки нагрузки.

- **Менеджер фрагментов данных.** Занимается распределением фрагментов данных по узлам вычислителя, их поиском и доставкой с других узлов по запросу, слежением за их готовностью и удалением более не нужных фрагментов данных (т.н. сборкой мусора).

- **Сетевая (коммуникационная) подсистема.** Эта подсистема реализует механизм асинхронной передачи сообщений между другими подсистемами, расположенными на разных узлах. Любая подсистема может отправить сообщение, указав номер узла и принимающую подсистему. По принятию сообщения вызывается определённая принимающей подсистемой процедура. Примерами сообщений являются отправка фрагментов

вычислений во время динамической балансировки нагрузки на узел, запросы на отправку и сама отправка фрагментов данных. Сетевая подсистема LuNA основана на MPI.

Глава 2

Алгоритмы системы тестирования

В больших параллельных численных программах, одной из самых медленных подсистем является сетевая подсистема. Также, менеджер фрагментов вычислений имеет высокие накладные расходы из-за необходимости наложения прямого управления на исходно непроцедурное. Поэтому две наиболее критичных с точки зрения производительности подсистемы — коммуникационная подсистема и менеджер фрагментов вычислений.

Для определения характеристик различных подсистем традиционно используются как синтетические тесты, так и профилирующие тесты, определяющими характеристики во время исполнения какой-либо прикладной задачи.

2.1. Тестирование сетевой подсистемы

Интерфейс сетевой подсистемы LuNA предоставляет процедуру неблокирующей отправки сообщения заданной подсистеме на заданном узле и возможность задать процедуру (коллбэк), которая которая будет вызвана по приходу сообщения. От сетевой подсистемы LuNA требуется производительность, достаточная для высокопроизводительных вычислений.

Сетевая подсистема LuNA является стандартной в своём роде, поэтому для её тестирования следует применить уже существующие методы.

2.1.1. Обзор

Стандартом де-факто в области тестирования сетевых устройств являются документы RFC 2544 «Benchmarking Methodology for Network Interconnect Devices» [12] и RFC 1242 «Benchmarking Terminology for Network Interconnection Devices» [13], описывающие необходимые определения и методологию тестирования производительности межсетевых соеди-

нений. Документы вводят определение следующих метрик и способ их замера:

- Пропускная способность — максимальное количество кадров в секунду, которое может передать устройство без ошибок.
- Латентность — временной интервал между поступлением последнего бита входного кадра во входной порт и выходом первого бита выходного кадра из выходного порта.
- Частота потери кадров — процент кадров, полученных для пересылки, которые устройство при постоянной нагрузке не смогло переслать по причине нехватки ресурсов.
- Максимальное количество кадров, отправленных минимальным межкадровым интервалом.
- Время восстановления после перегрузки.
- Время востановления после перезагрузки — программного или аппаратного сброса.

Эти документы оперируют достаточно низкоуровневыми понятиями, такими как кадр, и скорее подходят для тестирования устройств чем для тестирования программного обеспечения.

Также существует система Intel® MPI Benchmarks [14], предназначенная для тестирования производительности различных реализаций MPI. Эта система содержит набор тестов, каждый из которых тестирует производительность определённой операции MPI. Тесты классифицируются по версии MPI: тесты для MPI-1, для MPI-2 и MPI-3. Тесты для MPI-1 разбиваются на три категории:

- Тесты, в которых участвуют два узла. В частности, тест PingPong, в котором тестируется латентность и PingPing, являющийся одновременным симметричным запуском двух тестов PingPong.
- Тесты, в которых участвуют более двух узлов. Примером является тест Exchange, в котором каждый узел обменивается с двумя своими соседями по сообщению.
- Тесты коллективных операций MPI.

2.1.2. Предлагаемые тесты

Обзор показал, что эффективность коммуникационных подсистем определяется такими характеристиками, как латентность и пропускная способность, поэтому было предложено использовать тесты, определяющие именно эти характеристики.

Широко распространённым тестом для определения латентности между двумя узлами является ping-pong. Во время каждой итерации этого теста, первый узел отправляет сообщение второму и ждёт ответа. Второй узел ждёт сообщение от первого, и сразу после этого отправляет ответ обратно. Замеры времени, затраченного на исполнение большого количества последовательных итераций позволяют определить среднюю латентность. Следует использовать сообщение минимального размера для того, чтобы минимизировать влияние ширины канала на замеряемое время. Латентность L определяется как

$$L = \frac{T}{n} = \frac{T}{2c},$$

где T – время исполнения теста, n – количество пересылок, c – количество итераций ping-pong.

Для определения пропускной способности и количества сообщений в секунду в зависимости от размера сообщения предлагается использовать следующий алгоритм: первый узел отправляет на второй k сообщений размером в s байт, и ожидает ответа. После получения всех k сообщений, второй узел отправляет сообщение минимальной длины первому. Если k достаточно большое, то влиянием латентности и накладных расходов на время исполнения одной итерации теста можно пренебречь. Ширина канала B и количество сообщений в секунду M определяются как

$$B = \frac{T}{k \cdot s}; \;\; M = \frac{T}{k}.$$

2.1.3. Особенности тестирования сетевой подсистемы LuNA

Поскольку MPI широко используется в качестве сетевой подсистемы во многих прикладных вычислительных задачах, следует сравнить вышеперечисленные характеристики для сетевой подсистемы LuNA и MPI.

Также для определения накладных вычислительных расходов сетевой подсистемы необходимо провести описанные тесты не только на двух узлах, но и на одном, поскольку это отнимет от общего времени исполнения теста время, связанное непосредственно с отправкой данных по сети.

2.1.4. Профилирование количества и размера отправленных сообщений

Очевидно, на время работы сетевой подсистемы влияет количество и размер отправляемых во время исполнения прикладной программы сообщений. Зная распределение количе-

ства и размера сообщений, можно прогнозировать время работы сетевой подсистемы. Для того, чтобы определить количество и размер сообщений, отправленный во время исполнения прикладной задачи, необходим профилировщик, журналирующий количество отправленных сообщений в зависимости от размера сообщеня.

По результатам работы профилировщика можно определить:

- Долю количества отправленных сообщений определённого размера от общего числа всех отправленных сообщений.
- Долю объёма отправленных сообщений определённого размера в зависимости от общего объёма всех отправленных сообщений.
- Долю времени, потраченного на отправку сообщений определённого размера в зависимости от общего времени, потраченного на отправку сообщений. В отличие предыдущих двух характеристик, для определения этой, недостаточно одних только результатов профилирования, а требуется ещё и знание частоты передачи сообщений в зависимости от их размера, полученное в результате тестов, описанных в разделе 2.1.2.

Для упрощения понимания результатов журналирования, предлагается сгруппировать сообщения таким образом, чтобы в каждой группе содержались сообщения примерно одного размера.

2.2. Тестирование менеджера фрагментов вычислений

2.2.1. Устройство менеджера фрагментов вычислений LuNA

Менеджер фрагментов вычислений представляет собой конвейер (рис. 2.1), по которому проходят фрагменты вычислений, прежде чем раскрыться или исполниться. Конвейер состоит из следующих стадий:

- Стадия распределения, на которой фрагменты вычислений распределяются по вычислительным узлам. После поступления на эту стадию, фрагмент вычислений либо отправляется на следующую стадию, либо мигрирует на стадию распределения конвейера другого вычислительного узла. В зависимости от используемых алгоритмов распределения и балансировки нагрузки по вычислительным узлам, количество перенаправлений одного фрагмента вычислений может варьироваться.

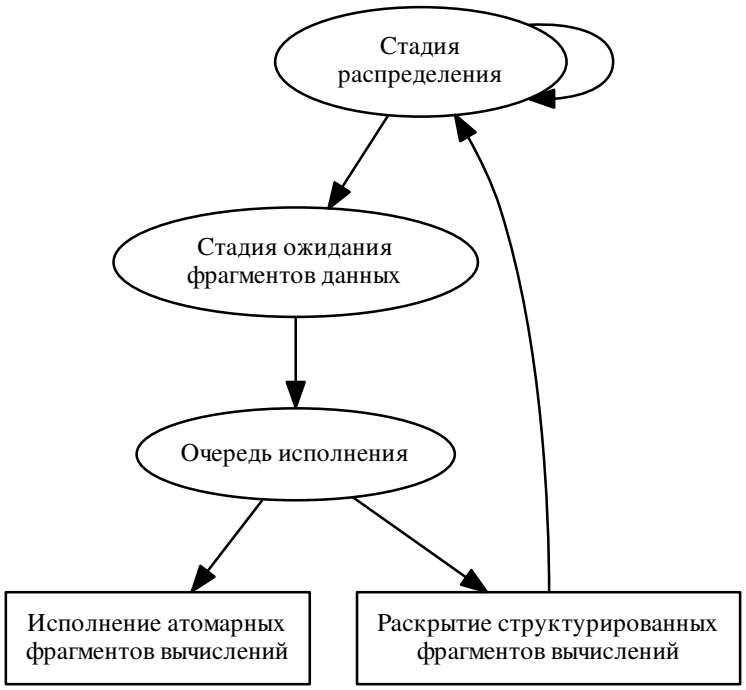


Рис. 2.1: Устройство конвейера

- Стадия ожидания фрагментов данных, на которой фрагменты вычислений ожидают, пока на вычислительный узел не поступят необходимые входные фрагменты данных.
- Очередь исполнения атомарных и раскрытия структурированных фрагментов вычислений. После исполнения или раскрытия, фрагменты вычислений уничтожается. Фрагменты вычислений, образованные в результате раскрытия структурированных фрагментов вычислений, попадают на стадию распределения.

Исполнение фрагментированной программы начинается с поступления на стадию распределения фрагмента вычислений, соответствующего подпрограмме `main`, определённой в коде фрагментированной программы.

2.2.2. Анализ менеджера фрагментов вычислений

Из приведённой структуры менеджера фрагментов вычислений видно, что эффективность работы менеджера фрагментов вычислений определяется объёмом накладных вычислительных расходов и качеством распределения фрагментов вычислений по узлам и определения порядка исполения. Поскольку фрагменты вычислений могут зависеть друг от друга, от правильного выбора порядка зависит время простоев.

Исходя из структуры менеджера фрагментов вычислений, можно выделить следующие метрики для определения его эффективности:

1. Количество миграций фрагментов вычислений на стадии распределения, поскольку это говорит о качестве распределения фрагментов вычислений по узлам.

2. Время прохождения фрагментами вычислений различных стадий. В частности, время ожидания на стадии ожидания говорит о качестве определения порядка вычислений.

3. Количество фрагментов вычислений, находящихся на различных стадиях в определённый момент времени, поскольку этого говорит о объёме занимаемой памяти.

4. Отношение времени простоя к времени накладных расходов и времени полезной нагрузки. Под временем полезной нагрузкой понимается процессорное время, затраченное на исполнение атомарных фрагментов вычислений, т.е. по сути подпрограмм на языке C++, написанных пользователем системы, а под временем накладных расходов понимается процессорное время, затраченное исполнением непосредственно самой системой исполнения.

2.2.3. Описание тестов

Получать значения метрик 1-3, можно при помощи системы профилирования, журналирующей события поступления фрагментов вычисления на следующую стадию конвейера, и анализатора полученных журналов профилирования. Поскольку фрагмент вычислений на стадии распределения мигрирует по нескольким узлам, из-за возможной рассинхронизации часов на разных узлах, точное время нахождения фрагментов вычисления, а также количество фрагментов вычисления на этой стадии определить невозможно. Однако получение количества пересылок на этой стадии остаётся возможным. После прохождения стадии распределения и до уничтожения, фрагмент вычислений не покидает вычислительный узел, на котором он находится, поэтому время нахождения фрагмента на стадии возможно.

Получение значения метрики 4 в ручном режиме возможно при использовании профилировщика общего назначения, либо в автоматическом при помощи использования профилировщика с ручным инструментированием.

2.2.4. Представление результатов тестирования

Для наглядности, результаты предложенных тестов предлагается представить следующим образом:

- Последовательность $[A_0, A_1, A_2, ...]$, где A_n — число ФВ, мигрировавших с узла на узел на стадии распределения n раз.

- Количество фрагментов вычислений, задерживающихся на стадии ожидания. Фрагмент вычислений пропускает ожидание на этой стадии в случае, если на момент его поступления на эту стадию, все соответствующие фрагменты данных уже имеются.
- График времени прохождения стадий фрагментом вычислений в зависимости от времени поступления на стадию и график количества фрагментов вычислений на стадии в зависимости от времени от начала исполнения программы. Это позволяет оценить динамику исполнения фрагментированной программы.

Глава 3

Практическая часть

3.1. Система тестов

В рамках данной работы была реализована система тестов, состоящая из:

- Тестов для сетевой подсистемы, определяющих латентность, пропускную способность и количество сообщений в секунду. Реализация тестов отделена от реализации интерфейсов к тестируемому модулю таким образом, чтобы не зависеть от конкретного тестируемого модуля. В качестве тестируемого модуля может выступать как сетевая подсистема LuNA, так и MPI.

- Профилировщика отправляемых сообщений во время исполнения прикладной задачи. По окончанию профилирования пользователь получает статистику в виде текстовой строки, содержащей набор пар $count_n : size_n$, где $size_n$ — размер в байтах, а $count_n$ — количество отправленных сообщений размера $size_n$. Также реализован визуализатор статистики, группирующий сообщения по размерам в зависимости от определённых пользователем групп и предоставляющий результат в виде круговой диаграммы или таблицы.

- Реализация системы тестов менеджера фрагментов вычислений состоит из профилировщика и анализатора. Во время исполнения прикладной программы на языке LuNA профилировщик фиксирует в журнале события перехода фрагментов вычисления на определённую стадию. После исполнения, анализатор анализирует журналы профилирования и выдаёт следующую информацию в виде текста и набора графиков:

Реализация системы тестов менеджера фрагментов вычисления состоит из профилировщика и анализатора. Во время исполнения прикладной программы на языке LuNA профили-

ровщик фиксирует в журнале события перехода фрагментов вычисления на определённую стадию. После исполнения, анализатор анализирует журналы профилирования и выдаёт следующую информацию в виде текста и набора графиков:

- Общее число фрагментов вычислений.
- Количество фрагментов вычислений, задерживающихся на стадии ожидания.
- Количество фрагментов вычислений в зависимости количества пересылок этого фрагмента на стадии distrib.
- Графики времени ожидания фрагментов вычислений на стадии ожидания и в очереди исполнения в зависимости от времени поступления соответствующего фрагмента на стадию.
- Графики количества фрагментов вычисления на стадиях ожидания и в очереди исполнения в зависимости от времени.

Для упрощения проведения комплексного тестирования был реализован тестер различных ревизий, принимающий в качестве входа произвольный тест и набор идентефикаторов ревизий.

В процесс сборки системы LuNA была внедрена инкрементальная система сборки из исходных кодов, что сократило время сборки при незначительных изменениях системы исполнения LuNA. В частности, это ускорило работу с тестером ревизий.

Также в язык LuNA введена поддержка параметров для основного фрагмента вычислений (точки входа).

3.1.1. Особенности и ограничения реализации

Анализаторы журналов написаны на языке программирования Python с использованием gnuplot и GNU R для построения графиков. Профилировщики отправляемых сообщений и менеджера фрагментов вычислений встроены в исходный код LuNA и являются опциональными.

Ограничениями реализации является невозможность определения времени простоя к времени накладных расходов и времени полезной нагрузки, а также возможное замедление времени работы прикладной программы из-за операций сохранения журнала в файл.

3.2. Тестирование при помощи профилировщика общего назначения и оптимизация системы исполнения LuNA

Во время профилирования системы исполнения LuNA при помощи профилировщика общего назначения было установлено, что довольно высокую долю времени исполнения занимает работа со строками (`std::string`). Основная часть работы со строками приходилась на операцию журналирования и работу со структурой `DFId`, предназначенной для хранения идентификатора фрагмента данных. Был разработан синтетический тест в виде задачи, в ходе исполнения которой система исполнения активно оперировала структурой `DFId`. Оптимизация операции журналирования значительно (на 21%) сократила время исполнения тестовой задачи.

Были исследованы операции, проводимые над этой структурой, а именно:

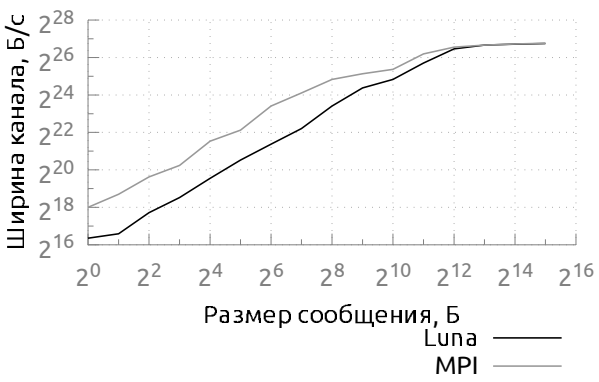
- Сравнение двух `DFId`, использующееся для задания отношения порядка при хранении `DFId` в дереве.
- Хеширование.
- Взятие индекса, т.е. получение нового `DFId` путём конкатенации со строкой `"[%d]"`, где `%d`— произвольный целочисленный индекс.

По результатам исследования была предложена и реализована альтернативная реализация `DFID`, которая показала лучшие результаты производительности на операции взятия индекса за счёт уменьшения расходов на выделение памяти, однако в целом потеряла в производительности из-за более медленной реализации операции сравнения.

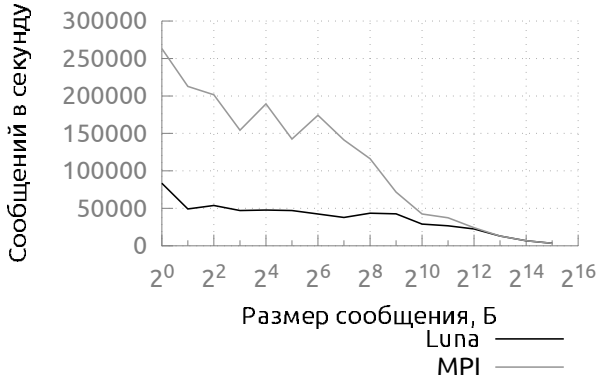
3.3. Результаты тестирования

Предлоденная система тестов была опробирована путём испытания системы исполнения LuNA. Тестирование было проведено на очереди G6 кластере ССКЦ СО РАН (НКС-30Т). В качестве реализации MPI использовалась библиотека MPICH. Характеристики очереди:

- 64 двойных блейд-сервера HP BL2x220 G6.
- 128 вычислительных модуля, ОП модуля – 16 Гбайт.
- 256 (1024 ядра) процессоров Intel Xeon E5540 2.53 Ггц (Nehalem).

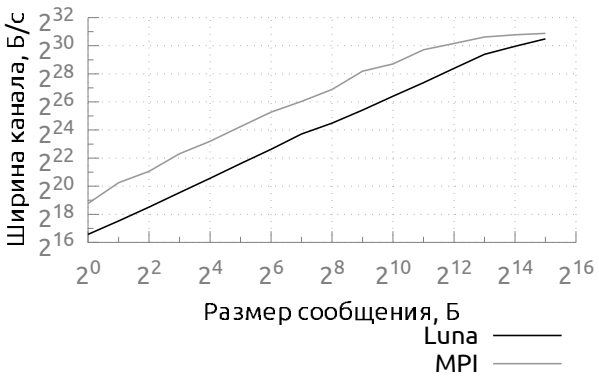


(а) Ширина канала

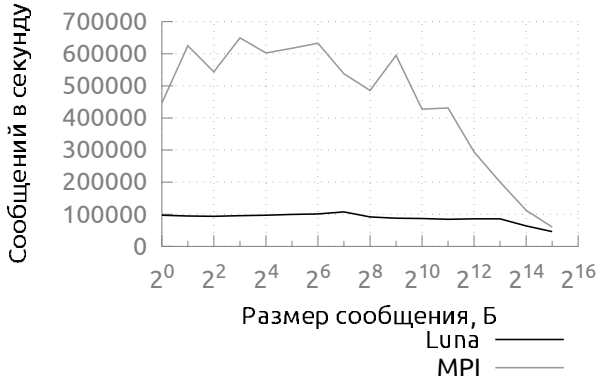


(б) Количество сообщений в секунду

Рис. 3.1: Результаты тестирования сетевой подсистемы на двух узлах.



(а) Ширина канала



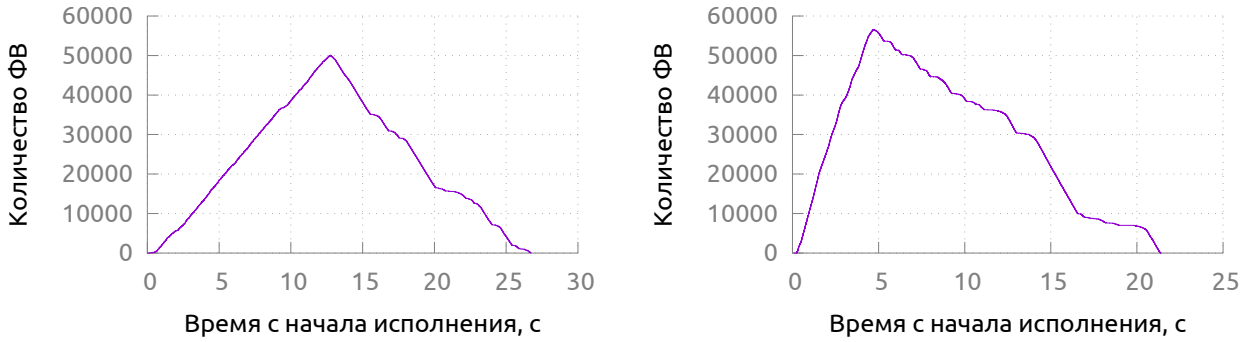
(б) Количество сообщений в секунду

Рис. 3.2: Результаты тестирования сетевой подсистемы на одном узле.

- Производительность – 10,36 Тфлопс.
- Коммуникационная сеть – QDR Infiniband.

При тестировании на двух узлах кластера, латентность MPI составила 52.3 мкс, латентносить LuNA — 80.7 мкс. При тестировании на одном узле, латентность MPI составила 12.5 мкс, латентносить LuNA — 34.7 мкс. Видно, что при работе на двух узлах (рис. 3.1) производительность сетевой подсистемы LuNA заметно ниже производительности MPI при сообщениях размером до 2^{12} байт, а при работе на одном узле (рис. 3.2) производительность заметно ниже при любом размере сообщения. Это говорит о высоком количестве накладных расходов в реализации сетевой подсистемы LuNA.

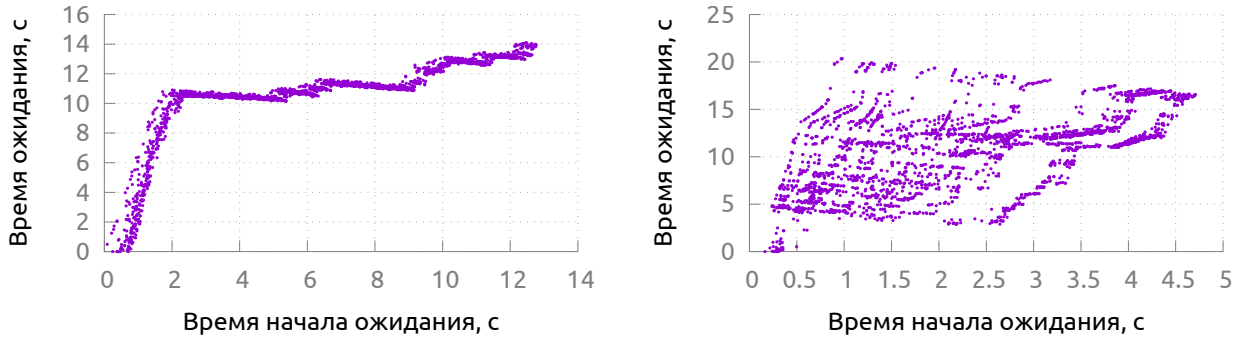
Также были протестированы две различных реализации менеджера фрагментов вычислений, использующихся в разных ревизиях системы исполнения. Для тестирования использовалась задача численного решения уравнения Пуассона. В тесте использовались 8 узлов кластера, на каждом из которых было запущено по 2 MPI-процесса. Время работы програм-



(а) Первая реализация

(б) Вторая реализация

Рис. 3.3: Количество фрагментов вычислений на стадии ожидания фрагментов данных при использовании различных реализаций менеджера фрагментов вычислений.



(а) Первая реализация

(б) Вторая реализация

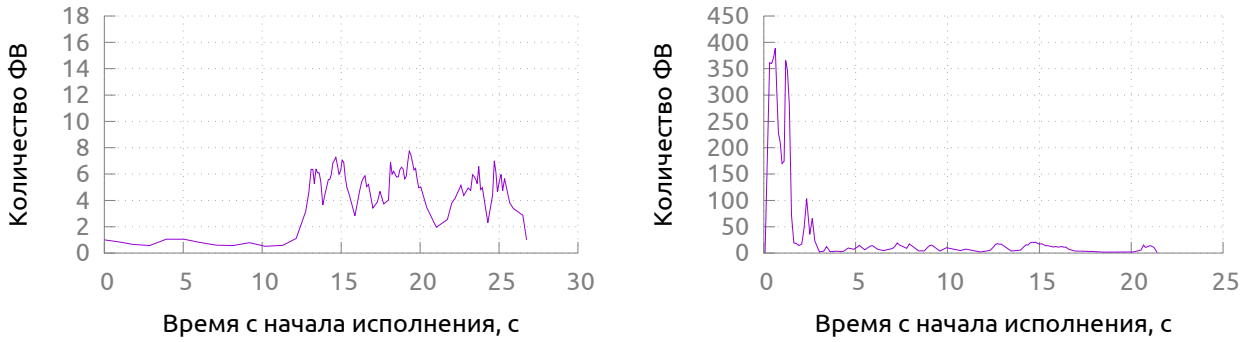
Рис. 3.4: Время ожидания фрагментов данных фрагментами вычислений при использовании различных реализаций менеджера фрагментов вычислений.

мы составило $28,38$ и $21,48$ секунд соответственно.

На графиках на рис. 3.3(б) и рис. 3.4(б) видно, что большинство фрагментовы вычисления поступило на стадию ожидания фрагментов данных в течение первых пяти секунд с начала исполнения, в то время, как в первой реализации (рис. 3.3(а) и рис. 3.4(а)), фрагменты данных поступали в течении первых денадцати секунд. Это говорит о том, что во второй реализации раскрытие фрагментов вычислений происходит быстрее, чем в старой. Более раннее раскрытие фрагментов вычисления приводит к повышению уровня параллелизма, что благоприятно влияет на время исполнения, однако также и приводит к большему потреблению памяти. График количества ФВ в очереди исполнения предоставлен на рис. 3.5.

Размер сообщения, Б	% от количества	% от объёма
32 – 40	44.7%	1.5%
69 – 80	37.2%	2.3%
96 – 652	15.6%	6.3%
41685 – 43288	2.5%	90.0%

Таблица 3.1: Результаты профилирования количества и размеров отправленных сообщений.



(а) Первая реализация

(б) Вторая реализация

Рис. 3.5: Количество ФВ в очереди исполнения при использовании различных реализаций менеджера фрагментов вычислений.

Профилирование показало (табл. 3.1), что суммарный объём сообщений, содержащих крупные фрагменты данных, составляет 90% от общего объёма отправленных сообщений, в то время как количество системных сообщений маленького размера (80 байт и менее) составило 81.2% от общего количества отправленных сообщений. Таким образом, тестирование показало, что сетевая подсистема LuNA требует усовершенствования при передаче сообщений малого объёма.

Заключение

В результате работы были определены ключевые характеристики LuNA и прикладных программ, влияющие на производительность, разработана и реализована система тестов для определения значений этих характеристик и предложена методика интерпретации результатов тестирования. По результатам тестирования были найдены и исправлены особенности реализации LuNA, негативно влияющие на производительность.

На защиту выносятся следующие положения:

- Разработана и реализована система тестов производительности для системы LuNA.
- Проведены работы по улучшению производительности системы LuNA.

Работу планируется продолжить в следующих направлениях:

- Усовершенствование методики тестирования.
- Увеличение количества тестируемых подсистем.
- Автоматизация интерпретации результатов.

Результаты работы были представлены на следующих конференциях:

- Параллельные вычислительные технологии (ПаВТ) 2015
- Конференция молодых учёных ИВМиМГ СО РАН 2015

Литература

1. The Chapel Parallel Programming Language // URL: http://chapel.cray.com/
2. L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, Parallel Programming using C++, pages 175–213. MIT Press, 1996.
3. Parallel Performance Analysis, Visualization and Optimization // URL: http://charm.cs.uiuc.edu/research/parallel_perf
4. HPC Challenge Benchmark // URL: http://icl.cs.utk.edu/hpcc/index.html
5. Chamberlain, Bradford L and Choi, Sung-Eun and Deitz, Steven J and Iten, David HPC Challenge benchmarks in Chapel // HPC Challenge Awards Competition at SC09 (November 2009), 2009
6. GNU gprof // URL: https://sourceware.org/binutils/docs/gprof/
7. Intel® VTune™ Amplifier // URL: https://software.intel.com/en-us/intel-vtune-amplifier-xe
8. CxxProf – CxxProf is a manual instrumented Profiling library for C++ // URL: https://github.com/monsdar/CxxProf
9. GPTL – General Purpose Timing Library // URL: https://jmrosinski.github.io/GPTL/
10. Малышкин В.Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2012. №46 (305). С.45-55.
11. Перепелкин В. А. Компилятор с языка представления фрагментированных программ // Труды конференции молодых ученых. Новосибирск: ИВМиМГ СО РАН, 2009 С.136-141.

12. RFC2544, Benchmarking Methodology for Network Interconnect Devices // URL: https://tools.ietf.org/html/rfc2544

13. RFC1242, Benchmarking Terminology for Network Interconnect Devices // URL: https://tools.ietf.org/html/rfc1242

14. Intel® MPI Benchmarks, User Guide and Methodology Description // URL: https://www.lrz.de/services/compute/courses/x_lecturenotes/MIC_GPU_Workshop/labs/IMB_Users_Guide.pdf

Приложение А. Документация к тестам

CFM

Что это

Построитель графиков времени ожидания фрагментов данных фрагментами вычислений и времени ожидания фрагментов вычислений на стадии WTP.

Как пользоваться

1. Собрать rts с флагом компилятора `-DPROFILER`.
2. Запустить программу на LuNA.
3. В результате в текущей директории появятся лог-файлы вида `prof_?????.json`, где `?????` — номер MPI-процесса.
4. Исполнить run.py, передав ему в качестве параметров имена лог-файлов. Например, команда может выглядеть так: `../../scripts/reports/cfm/run.py prof_*.json`
5. Пример текстового вывода скрипта:

```
Total cf:  60334[1]
Distribs:  [32431, 27903][2]
Wait df :  55801[3]
```

6. Графический вывод скрипта в форматах png, svg лежит в каталоге info.
   `info/wait_df_*.png` — время ожидания фрагментов данных фрагментами вычис-

[1] Всего фрагментов вычислений
[2] $[A_0,\ A_1,\ A_2,\ ...]$, где A_n — число ФВ, которые отправлялись с узла на узел n раз
[3] Скольким ФВ пришлось ожидать входных ФД, а не запускаться сразу

лений.

`info/wait_wtp_*.png` — время ожидания на стадии WTP.

cstest

1. Как провести тестирование производительности сетевой подсистемы

1. Убедиться, что для сетевой подсистемы реализован интерфейс ISenderReceiver (см. раздел 4).
2. Убедиться, что сетевая подсистема находится в списке тестируемых (см. раздел 4).
3. Запустить rts на двух MPI-процессах с переменной окружения `LUNA_TEST=cstest3`. Результат будет выведен в stdout.
4. Чтобы построить графики, нужно запустить
   `perfomance/tests.sh [вывод rts] [каталог с графиками]`.
   Графики строятся только для MPS (messages per second). Если тестируется больше двух подсистем, то графики строятся только для двух (Если потребуется, можно будет исправить).

2. Пример

```
cd $LUNA_HOME/scripts/reports/cstest
mpirun -np 2 env LUNA_TEST=cstest3 $LUNA_HOME/build/rts > out.txt
./tests.sh out.txt out
# В каталоге `out` будут лежать графики
```

3. Запуск на НКС-30Т

Также в `run_nks.sh` содержится скрипт для запуска теста на двух узлах кластера НКС-30Т. Использовать его можно так:

```
cd \$LUNA_HOME/scripts/reports/cstest
./run_nks.sh # результат работы скрипта записывается в файл `out_2.txt`
```

4. Реализация интерфейса ISenderReceiver

Интерфейс и его реализации описываются в файле `src/testing/commservtest3.cpp`. В данный момент этот интерфейс реализован для:

1. MPI (класс `MpiSenderReceiver`)
2. luna::comm::CommReceiver (класс `LunaSenderReceiver`)

Список тестируемых определяется в том же файле после комментария `// Push back new subsistems into `isrs`.`