Минобрнауки России

Федеральное государственное автономное образовательное учреждение высшего образования   
«Национальный исследовательский университет   
«Московский институт электронной техники»

Институт системной и программной инженерии и информационных технологий (СПИНТех)

Рулева Виктория Олеговна

Магистерская диссертация   
по направлению 09.04.04 «Программная инженерия»

Исследование и разработка методик и алгоритма анализа характеристик сетевой вычислительной инфраструктуры гетерогенных кластеров

Студент Рулева В. О.

Руководитель,   
к.т.н., доцент Федоров А. Р.

Москва 2023

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc136928838)

[ВВЕДЕНИЕ 4](#_Toc136928839)

[ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ АНАЛИЗА ОСНОВНЫХ ХАРАКТЕРИСТИК СЕТЕВЫХ ИНФРАСТРУКТУР 6](#_Toc136928840)

[1.1. Анализ функциональных особенностей и архитектуры суперкомпьютеров 6](#_Toc136928841)

[1.2. Исследование особенностей построения гетерогенных систем на кристалле 12](#_Toc136928842)

[1.3. Исследование составляющих информационной коммуникационной среды 19](#_Toc136928843)

[1.4. Аналитический обзор существующих методов и средств анализа основных характеристик сетевых инфраструктур 26](#_Toc136928844)

[1.5. Постановка целей и задач диссертационных исследований 31](#_Toc136928845)

[Выводы по главе 1 33](#_Toc136928846)

[ГЛАВА 2. ФОРМАЛИЗАЦИЯ ПРОЦЕССА ПЕРЕДАЧИ СООБЩЕНИЙ МЕЖДУ ДВУМЯ ПРОЦЕССАМИ В КЛАСТЕРАХ ГЕТЕРОГЕННОГО ТИПА 34](#_Toc136928847)

[2.1. Анализ операций при передаче сообщений между двумя процессами 34](#_Toc136928851)

[2.2. Разработка математической модели передачи сообщений на одном узле кластера 42](#_Toc136928852)

[2.3. Разработка математической модели передачи сообщений на разных узлах кластера 45](#_Toc136928853)

[2.4. Формализация процесса передачи сообщений между однородными и неоднородными узлами кластера 51](#_Toc136928854)

[2.5. Разработка методики расчета и анализа основных характеристик коммуникационной инфраструктуры кластера 52](#_Toc136928855)

[2.6. Разработка методики ранжирования стандартных интерфейсов межузловых коммуникаций 59](#_Toc136928856)

[Выводы по главе 2 69](#_Toc136928857)

[ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДИК И АЛГОРИТМА АНАЛИЗА ХАРАКТЕРИСТИК СЕТЕВОЙ ВЫЧИСЛИТЕЛЬНОЙ ИНФРАСТРУКТУРЫ ГЕТЕРОГЕННЫХ КЛАСТЕРОВ 70](#_Toc136928858)

[3.1. Модель виртуального разделения основного процесса на субпроцессы для запуска MPI-программ без изменений 70](#_Toc136928861)

[3.2. Разработка методики ускорения выполнения MPI-приложений на кластере гетерогенного типа 72](#_Toc136928862)

[3.3. Структура системы сборки дистрибутива ОС Linux Buildroot 83](#_Toc136928863)

[3.4. Пакетное добавление в сборку пользовательских программ и библиотек 87](#_Toc136928864)

[3.5. Разработка алгоритма измерения основных сетевых характеристик кластерной системы 93](#_Toc136928865)

[Выводы по главе 3 97](#_Toc136928866)

[ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНОЕ ПОДТВЕРЖДЕНИЕ РЕЗУЛЬТАТОВ ИССЛЕДОВАНИЯ 98](#_Toc136928867)

[4.1. Описание стенда для проведения экспериментальных исследований 98](#_Toc136928870)

[4.2. Порядок проведения экспериментальных исследований 103](#_Toc136928871)

[4.3. Экспериментальные исследования, подтверждающие достоверность полученных результатов 112](#_Toc136928872)

[4.4. Практическое применение методики ускорения выполнения MPI-приложений на кластере гетерогенного типа 123](#_Toc136928873)

[4.5. Перспективы практического приложения результатов диссертационного исследования 130](#_Toc136928874)

[Выводы по главе 4 133](#_Toc136928875)

[ЗАКЛЮЧЕНИЕ 134](#_Toc136928876)

[СПИСОК ЛИТЕРАТУРЫ 135](#_Toc136928877)

# ВВЕДЕНИЕ

Одной из главных проблем при оценке коммуникационных характеристик межпроцессного обмена в гетерогенных кластерах, является то, что в связи с отсутствием общепринятых методик и алгоритмов, используются методы и средства анализа однородных систем, что в отсутствие систематического подхода может привести к некорректным результатам. Основные причины сложившийся ситуации:

• широкое распространение кластерных систем, которые в начальной конфигурации проектируются с использованием однородных ВУ, но при наращивании вычислительной мощности становятся неоднородными;

• высокая сложность использования существующих средств анализа вычислительных систем в контексте гетерогенных систем на кристалле;

• необходимость учёта как внешней коммуникационной подсети кластера, так и матрицы коммутации между кластерами процессорных ядер и уровнями памяти.

Объектом исследования является сетевая вычислительная инфраструктура гетерогенных кластеров.

Предметом исследования является методики и алгоритм анализа основных коммуникационных характеристик.

Цель диссертационного исследования заключается в повышении быстродействия процесса передачи сообщений между узлами гетерогенной кластерной системы.

В соответствии с целью исследования в диссертации необходимо решить следующие задачи:

* аналитический обзор существующих методов и средств анализа коммуникационных характеристик сетевых инфраструктур;
* формализация процесса передачи сообщений между однородными и неоднородными узлами кластера;
* разработка методики расчета и анализа основных характеристик коммуникационной инфраструктуры кластера;
* разработка методики ранжирования стандартных интерфейсов межузловых коммуникаций;
* разработка методики ускорения выполнения MPI-приложений на кластере гетерогенного типа;
* разработка алгоритма измерения основных сетевых характеристик кластерной системы;
* проведение экспериментальных исследований, подтверждающих достоверность полученных результатов.

Положения, выносимые на защиту:

1. Формализованное представление процесса передачи сообщений между однородными и неоднородными узлами кластера.
2. Методика расчета и анализа основных характеристик коммуникационной инфраструктуры кластера.
3. Методика ранжирования стандартных интерфейсов межузловых коммуникаций.
4. Методика ускорения выполнения MPI-приложений на кластере гетерогенного типа.
5. Алгоритм измерения основных сетевых характеристик кластерной системы.
6. Оценка эффективности предлагаемых методик.

Структура диссертационной работы. Диссертационная работа содержит введение, четыре главы, заключение, список литературы, таблицы, рисунки. Объем работы составляет 161 страницу.

# ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ АНАЛИЗА ОСНОВНЫХ ХАРАКТЕРИСТИК СЕТЕВЫХ ИНФРАСТРУКТУР

Анализ функциональных особенностей и архитектуры суперкомпьютеров

В связи с тем, что обычные ЭВМ перестали за разумное время справляться с задачами, требующими осуществления интенсивных математических расчетов, распознавания и синтеза, моделирования сложных систем и т.д., встал вопрос о создании более производительных вычислительных машин. Подобным сверхмощным компьютерам, значительно превосходящим обычные компьютеры по характеристикам и скорости вычислений, а также поддерживающим в той или иной степени параллелизм на аппаратном уровне дали название суперЭВМ или суперкомпьютеры. Основное их отличие от традиционных компьютеров заключается в том, что они используют более одного ЦПУ и способны выполнять свыше 10 млрд. теоретических операций в сек. В настоящее время сфера применения суперкомпьютеров расширяется ещё больше: помимо областей, эффективное решение проблем в которых было бы невозможно без использования высокопроизводительных систем (астрономия, гидродинамика, генетика, термоядерный синтез и многих других), также добавились задачи, связанные с обработкой больших данных, системами поддержки решений, обработкой транзакций в режиме реального времени.

Если брать за основу критерий параллелизма вычислений, то все компьютерные системы в зависимости от числа потоков команд и данных можно классифицировать по таксономии Флинна (рис. 1.1) (1). Классификация базируется на всевозможных сочетаниях булевых переменных p1 (mk - множество потоков команд) и p2 (md - множество потоков данных), которые могут принимать одно из двух истинных значений, где 0 — одиночный поток, а 1 — множественный поток, а через переменные mk и md обозначают минимальное количество потоков. Соответственно, возможные комбинации вышеперечисленных значений порождают четыре типа вычислительных систем:

1. {p1p2} = 00, mk = md = 1: SISD (Single Instructions, Single Data). Класс представляет собой традиционные однопроцессорные компьютеры с одним операционным устройством. Как правило, инструкции выполняются последовательно, но их выполнение можно существенно ускорить при помощи механизма конвейерной обработки.

2. {p1p2} = 01, mk = 1, md > 1: SIMD (Single Instructions, Multi Data). К данному классу относятся однопроцессорные компьютеры, но в отличие от предыдущего класса способного только на скалярную обработку, SIMD-системы реализовывают векторную или матричную обработку ввиду наличия нескольких операционных устройств.

3. {p1p2} = 10, mk > 1, md = 1: MISD (Multi Instructions, Single Data). MISD-системы признаны нецелесообразными и на практике не реализуются.

4. {p1p2} = 11, mk > 1, md > 1: MIMD (Multi Instructions, Multi Data). Класс, включающий в себя все возможные многопроцессорные системы, в том числе все суперкомпьютеры. Системы с общей памятью данных и команд считаются сильно связанными и называется мультипроцессорами, а системы с распределенной организацией памяти, наоборот, слабо связаны и называются мультикомпьютерами. Далее рассматривается только этот класс.

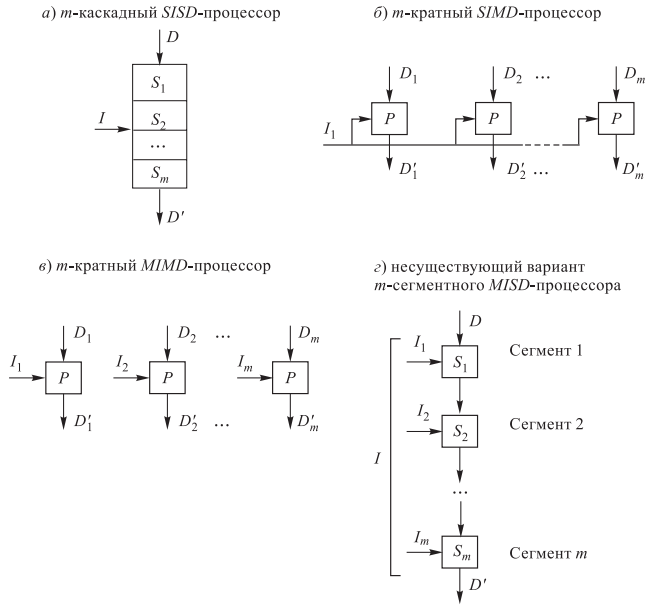


Рисунок 1.1 — Модели вычислительных процессов для процессоров потоковой классификации Флинна

Таким образом, по таксономии Флинна некоторая часть суперкомпьютеров относится к SIMD-системам, а именно векторные однопроцессорные системы, подавляющее же большинство относится к классу MIMD-систем, т.е. обладает множественным потоком команд и множественным потоком данных. В классе MIMD-систем в зависимости от характера вычислителей существует два параллельно развивающихся направления развития суперкомпьютеров: магистральное и кластерное (2). Магистральное направление подразумевает создание специализированных вычислителей на основе промышленных микросхем. К этому направлению относятся следующие классические архитектурные решения:

— SMP (Symmetric Multiprocessing) (3). Ключевым элементом SMP-системы является высокоскоростная системная шина, через которую два и более одинаковых процессора совместно используют ресурсы (модель программирования OpenMP), что сильно упрощает общение узлов, но при этом отрицательно сказывается на масштабируемости. В один момент времени шина способна обрабатывать только одну транзакцию, поэтому на практике число включаемых процессоров резко сокращается. Зачастую подобные процессоры абсолютно идентичны и могут выполнять одни и те же функции, а когерентность кэшей поддерживается аппаратно. Доступ к системам ввода/вывода и симметрично адресуемой оперативной памяти осуществляется с равными правами и за одинаковое время, тем самым относя SMP-системы к более общей архитектуре UMA (Uniform Memory Access). В общем случае рабочая нагрузка распределяется единой операционной системой автоматически между процессорами без специального учета в прикладной программе.

— ASMP (Asymmetric Multiprocessing) (3). Если в SMP-системе все процессоры должны были иметь одинаковую архитектуру, то ASMP система позволяет использовать процессоры разной архитектуры. Также выделяется главный ЦП, который выполняет задачи ОС и распределяет их между остальными процессорами согласно приоритету (подход Master-Slave). Таким образом, процессорам не обязательно напрямую взаимодействовать друг с другом через общую память, если они управляются главным. Данная архитектура считается более простой и дешевой в реализации, т. к. нет необходимости в синхронизации процессоров.

— MPP (Massive Parallel Processing) (4). MPP-система позволяет скоординировано обрабатывать одну задачу несколькими абсолютно самодостаточными вычислительными узлами, включающими: несколько ЦП, локальную память, собственную UNIX-подобную ОС (полноценную или урезанную) и другие устройства. Ввиду отсутствия общей физической памяти, скорость межпроцессного обмена заметно снижена, а коммуникация узлов реализована с помощью какого-либо интерфейса обмена сообщениями. В отличие от SMP-архитектуры пользователь не может оперировать сразу всем объемом памяти (модель программирования MPI), т. к. каждый процессор имеет доступ только к своей ограниченной по объему памяти. Несмотря на это, решение отличается гибкой масштабируемостью — на практике количество процессоров может достигать нескольких тысяч.

— NUMA (Non-Uniform Memory Access) (4). Данная архитектура считается гибридной и представляет собой комбинацию двух предыдущих — за основу берется MPP-архитектура, а вычислительные элементы имеют конфигурацию SMP узлов, что позволяет нивелировать недостатки этих архитектур. Базовый модуль состоит из нескольких ЦП и собственного блока памяти, посредством коммутатора данные модули объединяются в одну систему. Память физически разделена между всеми NUMA узлами, но логически является общей, поэтому пользователь видит единое адресное пространство. По отношению к конкретному модулю собственная память называется локальной и имеет самую низкую задержку доступа, память других узлов считается удаленной и имеет более высокую задержку. Также существуют расширения данной архитектуры — cc-NUMA, в которой обеспечивается когерентность кэшей, и nc-NUMA без кэширования.

— PVP (Parallel Vector Process) (4). Главное отличие параллельно-векторных систем в том, что вместо обычных процессоров, используются более дорогие векторно-конвейерные. Сама система может иметь любую из вышеперечисленных архитектур, в том числе однопроцессорную SIMD. Передача данных в векторном формате происходит гораздо быстрее, чем в скалярном, что значительно повышает пропускную способность и уменьшает задержку доступа.

Ко второму направлению относятся кластерные вычислительные системы COW (Clusters Of Workstations) (2). В классическом представлении кластер — это совокупность вычислительных устройств, связанных в 2D или 3D локальную вычислительную сеть и функционирующих как одна система. Как правило, ни пользователь, ни прикладная задача, для решения которой используется кластер, не различают отдельные компоненты кластера, а работают с ним как с единым аппаратным ресурсом (5). Подобная архитектура во многом схожа с MPP-системой, отличие заключается в том, что используются не дорогие вычислители, а обычные персональные ЭВМ. На рисунке 1.2 приведен пример высокопроизводительного вычислительного кластера на 18 узлов от НИВЦ МГУ.



Рисунок 1.2 — Схема вычислительного кластера НИВЦ МГУ

Узел кластерной системы характеризуется тем, что на нем работает собственная копия ОС, поэтому в случае его извлечения из сетевой инфраструктуры, любой другой способен самостоятельно функционировать. Тем не менее подобные самостоятельные узлы могут обладать как полной, так и частичной функциональность. Вычислители, снабженные устройствами ввода/вывода, графическими картами и другим периферийным оборудованием, обладают полной функциональностью. Но после включения в сетевую инфраструктуру кластера, нуждается в подобном оборудовании только управляющий узел, для других же оно избыточно. Поэтому, как правило, большинство узлов обладает лишь частичной функциональностью, что позволяет существенно повысить производительность кластера.

Совокупность всех вычислительных узлов может быть как однородна, так и неоднородна. Если все процессоры имеют одинаковую архитектуру и, как правило, производительность, то такие кластеры считаются гомогенными. По умолчанию предполагается, что начальная конфигурация узлов кластерной системы – однородна и, соответственно, гомогенна. Но, когда производится повышение вычислительной мощности путем включения в сетевую инфраструктуру процессоров, в большинстве случаев архитектура таких процессоров отличается от имеющихся. Подобные кластерные системы называются гетерогенными и имеют ряд проблем. Во-первых, из-за того, что производительность узлов неоднородного кластера различается, может потребоваться более тщательное планирование распределения задач по каждому узлу. Во-вторых, возникают дополнительные сложности, связанные с коммуникацией между узлами. Узлы в кластере с распределенной памятью могут взаимодействовать только посредством сообщений, но из-за различий в архитектуре узлов появляется необходимость приводить различающиеся данные к единому формату.

Кластерное направление долгое время отставало от магистрального, но с развитием высокоскоростных коммутационных систем (шина PCI Express, семейство сетей Ethernet и другие), сравнялось по производительности. Если раньше организация кластерных систем была дешевым способом наращивания производительности путем объединения пользовательских ПК в одну систему, то на сегодняшний момент существует тенденция к объединению двух направлений - кластерные системы строятся на основе специализированных вычислителей, например, многопроцессорных систем на кристалле. Таким образом, использование кластерной архитектуры на базе многопроцессорных вычислительных систем позволяет, как повысить производительность кластера, так и нивелировать недостатки многопроцессорных архитектур, например, плохую масштабируемость SMP-систем.

Исследование особенностей построения гетерогенных систем на кристалле

ЭВМ, использующие вычислители разных типов, называются гетерогенными (гибридными) (6) и, по сравнению с гомогенными (однородными) системами, они производят вычисления эффективнее за счет добавления специализированных узлов, например, массивно-параллельных ускорителей, оптимизированных для распараллеливания, что позволяет распределять данные и работать между узлами системы, тем самым решая задачи гораздо быстрее, нежели их решала бы однородная система, состоящая из универсальных узлов. Даже самые простые приложения могут много раз переходить между периодами последовательных операций и параллельных вычислений, но высокопроизводительная вычислительная машина должна обеспечивать оптимальную производительность и малую задержку перехода в обеих ситуациях, что и достигается путем использования различных типов вычислителей, что значительно превосходит любые решения в рамках однородной многопроцессорной архитектуры. Таким образом, в соответствии с моделью гетерогенных вычислений, сегменты рабочей нагрузки выполняются на наиболее подходящем процессоре — CPU с оптимизированной задержкой или высокой пропускной способностью, а оптимизированный GPU — обеспечивает более высокую общую производительность и меньшее энергопотребление и затраты на единицу вычислений.

Использование гетерогенных архитектур при построении вычислительных систем является одной из причин увеличения средней производительности суперкомпьютеров примерно в 100 тысяч раз за последние 20 лет (7). К настоящему времени при построении высокопроизводительных вычислительных систем используют преимущественно гетерогенные технологии (Summit (8) и большинство суперЭВМ на первых позициях рейтинга Топ500 (11)), т. к. классические гомогенные архитектуры больше не позволяют существенно повысить производительность без наращивания количества вычислителей или увеличения энергопотребления. Ещё одним из эффективных способов повышения общей производительности является расширение набора инструкций векторными (Supercomputer Fugaku (9)) или использование векторных процессоров в качестве ускорителей (NEC SX-Aurora TSUBASA (10)).

В отличие от самых высокопроизводительных суперкомпьютеров, недоступных широкой публике, линейки микропроцессоров от компаний Intel, Nvidia и т.д. производятся массово для смартфонов и планшетов, персональных компьютеров и рабочих станций, а также используются в автомобильной отрасли и для специализированных игровых устройств. В категорию систем на кристалле массового производства входит 50-ядерная микропроцессорная система на кристалле 1892ВМ248 под кодовым названием «RoboDeus» из серии 32 и 64-битных процессоров «Мультикор» (12). Обозначение «Мульти» в названии серии указывает на то, что процессоры имеют многоядерную гетерогенную архитектуру. Данная серия СнК разрабатывается и производится компанией АО НПЦ «ЭЛВИС» на базе RISC-ядер, которые используются в качестве управляющего процессора и оригинальных DSP-ядер («ELCore-xx») для цифровой обработки сигналов. Технология изготовления процессоров серии «Мультикор» варьируется от 250 нм до 16 нм. Области применения также достаточно широки: радиационно-стойкие процессоры для применения в космической и авиационной отрасли, встраиваемые процессоры для систем связи и навигации, мобильных телефонов, планшетов и одноплатных компьютеров, процессоры для систем компьютерного зрения и искусственного интеллекта.

СнК «RoboDeus» спроектирована с использованием IP-блоков проприетарной платформы «МУЛЬТИКОР» разработки АО НПЦ «ЭЛВИС» и покупных IP-блоков, в том числе от IMG (13). Управляющий CPU 0 – это основной сервисный процессор микросхемы, который представляет собой одноядерный MIPS I6400 (Samurai) совместимый с архитектурой MIPS R6 с двумя потоками. CPU 0, как и все блоки вычислительного кластера обладает полным доступом ко всей карте памяти микросхемы. Сам вычислительный кластер состоит из трех видов ядер (рис. 1.3):

1. Арифметический сопроцессор из двух CPU (CPU 1 и CPU 2) формирует основной прикладной процессор системы (14). Каждый CPU – это четырёхъядерный MIPS I6500 Daimyo, совместимый с архитектурой MIPS R6. Этот кластер отвечает за работу высокоуровневой операционной системы. Кэш данных L1 поддерживает протокол когерентности MESI (modified, exclusive, shared, invalid). Поддерживается работа как в режиме AMP (Asymmetric Multiprocessing Mode), так и в SMP (Symmetric Multiprocessing Mode). Каждый блок CPU сконфигурирован из 4х когерентных ядер MIPS64-R6 CPU с двумя потоками на каждое ядро. Общесистемный блок управления обеспечивает когерентность кэша L2 и прерываний по всем ядрам CPU (когерентный интерфейс памяти). Объемы кэша данных L1 и кэша инструкций L1 на каждое ядро – 64 Кб, кэш второго уровня L2 объемом 2 Мб. Загрузка CPU 1 производится ведущим CPU 0, также ведущий CPU 0 отвечает за установку вектора сброса и управление последовательностью сброса CPU 1 и CPU 2. CPU 2 может быть также запущен через CPU 1.

2. Многоядерный DSP кластер VELCore03 является вычислительным кластером, который содержит массив отечественных ядер DSP и RISC. Кластер разбит на четыре эквивалентных подкластера QUELCore. Каждый подкластер QUELCore включает в себя 4 ядра DSP Elcore50, управляющее ядро VCU (RISC архитектура), внутренний коммутатор и логику управления, и обладает доступом ко всем регистрам и памяти в системе. IP-блок VELCore содержит дополнительный вспомогательный блок VDEBUG для отладки VELCore через JTAG. Входной интерфейс JTAG блока подсоединяется к специализированному порту микросхемы JTAG (V\_JTAG), а выходные интерфейсы JTAG подсоединены к портам отладки QUELCore. Процесс загрузки VELCore аналогичен алгоритму загрузки на CPU 0, когда CPU 0 является ведущим.

3. Блок графического процессора (GPU) относится к серии 8XT Power VR в 4-х кластерной конфигурации. Графические процессоры PowerVR Series 8XT построены на основе многопоточных объединенных шейдерных кластеров (USC), которые представляют собой устройства с архитектурой АЛУ с эффективным SIMD, и поддерживают отсроченный тайловый рендеринг с одновременной обработкой множества тайлов. Блок графического процессора (GPU) может выполнять задачи работы с трехмерной графикой, которая включает обработку данных по вертексам и пикселям для рендеринга 3D-сцен, задачи работы с двумерной графикой, которая используется для быстрого асинхронного 2D-рендеринга, вычислительные задачи (GP-GPU), которые включают общую обработку данных, наложение видеосигналов, распознавание и поиск лиц, прочие алгоритмы видеоаналитики.

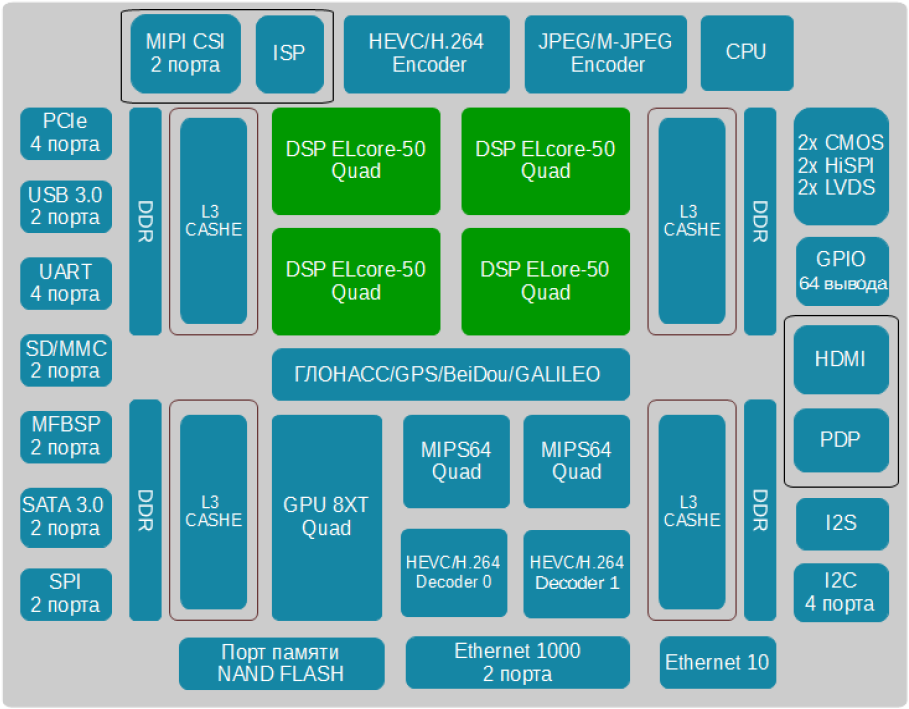


Рисунок 1.3 — Компоненты интегральной микросхемы 1892ВМ248 (СнК «RoboDeus»)

Общая структурная схема программного обеспечения СнК «RoboDeus» представлена на рисунке 1.4 (15). В состав поставляемого системного ПО входит тестовый образ дистрибутива Buildroot с ОС Alt Linux на основе ядра Linux 5.xx с необходимыми драйверами для:

* вычислительных устройств – Elcore50, GPU, ISP, VPU;
* видео кодеров и декодеров;
* высокоскоростных интерфейсов – PCIe 4.0, 1G, USB, SATA;
* низкоскоростных интерфейсов – UART, I2C, I2S;
* PVT и HDMI 2.0.

Инструментальное и отладочное ПО включает:

* компилятор языка С/C++ для процессорного блока CPU;
* компилятор С/С++/C11 для блока сигнальной обработки DSP;
* средства компиляции OpenCL для графического ускорителя GPU;
* пакет бинарных утилит на основе binutils;
* ассемблер, дизассемблер, линкер, библиотекарь.

В состав инфраструктурного ПО входят:

* программная платформа ELVEES Performance Platform, включающая DL compiler, DLI framework и DLT framework. Упрощает создание оптимизированного программного решения для умных устройств, решающих задачи в области Computer Vision, Speach Recognition, NLP и т. п. путем предоставления популярных библиотек и фреймворков, оптимизированных под ядра Elcore50 и другие особенности процессоров Elvees;
* библиотеки параллельного и распределенного программирования – проприетарный Multicore MPI (MMPI), OpenCL и другие API для GPU. Поддерживается режим программируемого вычислителя (pGPU) с API OpenCL, производительность под OpenCL не хуже FP16 – 307 GFLOP/s, FP32 – 153 GFLOP/s. Помимо OpenCL, поддерживаются стандарты OpenGL и OpenVG, причем производительность под OpenGL ES не хуже 150-400 Мполигонов/с, 1.2-3.2 Гпикселей/с;
* прикладные библиотеки для DSP-ядер Elcore50 - библиотека тензорных вычислений DnnLibrary, математическая библиотека скалярных, векторных и матричных операций, библиотека базовых подпрограмм линейной алгебры FastBLAS.

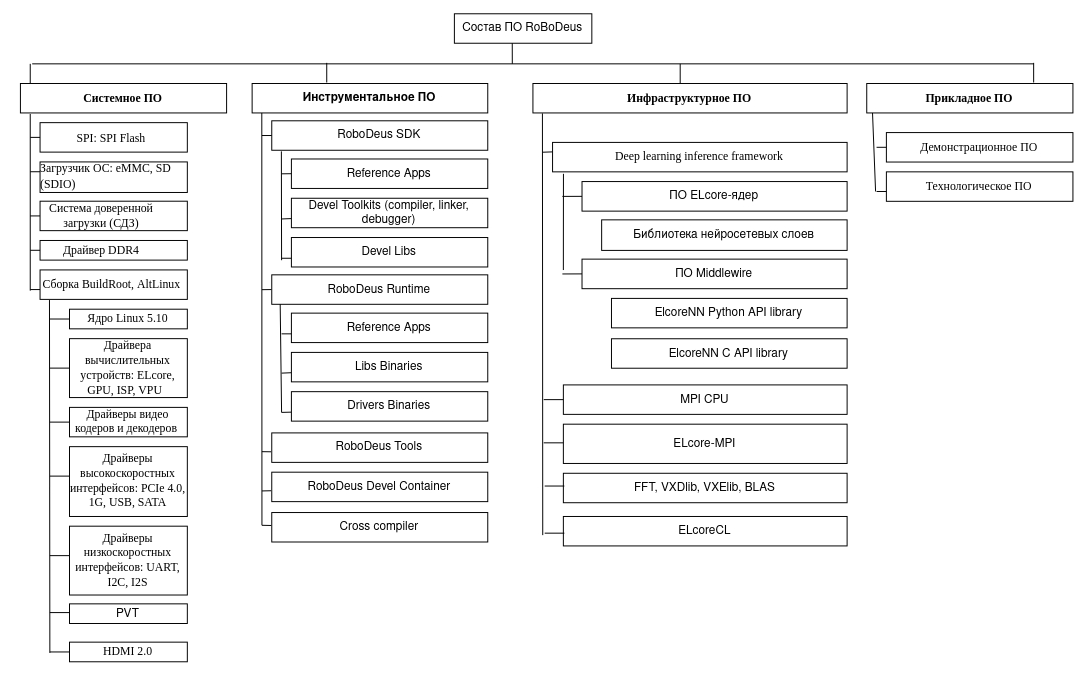


Рисунок 1.4 — Структурная схема ПО СнК «RoboDeus»

Среди общих характеристик СнК «RoboDeus» необходимо отметить, что энергопотребление микропроцессора не более 30 Вт (пиковое – до 70 Вт). Замеренная производительность – свыше 1,2 Tflop/s для формата с двойной точностью (FLP64), не менее 4 Tflop/s для формата с одинарной точностью (FLP32) и 16 Tflop/s с половинной точностью (FP16). Проектные нормы микропроцессора RoboDeus передовые для РФ 16 нм (процесс TSMC) и, согласно вышеописанным архитектуре, используемым кластерам ядер и стеку ПО, он может применяться для решения следующих задач:

* считывание информации от различных типов датчиков, благодаря многочисленным портам ввода-вывода, встроенным драйверам и библиотекам для взаимодействия с ними (рис. 1.5);
* последующая когнитивная обработка данных с целью распознавания видео, аудио и текстовой информации, а также использование в семантических серверах и центрах обработки больших данных;
* реализация и применение большинства типов алгоритмов, в том числе алгоритмов машинного обучения, прогнозирования и одновременное вычисление различных типов нейронных сетей;
* применение в автономных робототехнических комплексах и приложениях в связи с низким энергопотреблением, а также в качестве бортовых вычислительных устройств для беспилотных аппаратов.

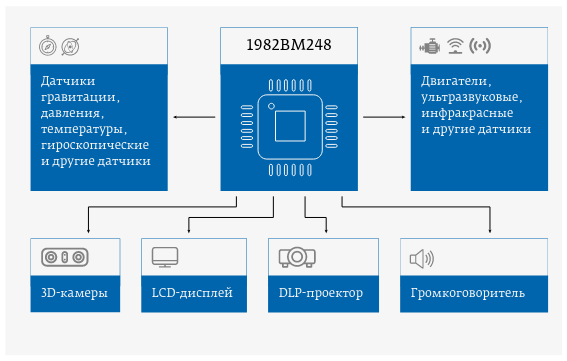


Рисунок 1.5 — Схема подключения датчиков и устройств к процессорному модулю 1892ВМ248 (СнК «RoboDeus»)

В таблице 1.1 приведены основные характеристики систем на кристалле, предназначенных для высокопроизводительных вычислений, от компаний Nvidia, Intel и «ЭЛВИС».

Таблица 1.1 — Сравнение передовых систем на кристалле

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Nvidia Tegra X1 (январь 2015)** | **Nvidia Tegra X1+ (лето 2019)** | **Nvidia Jetson AGX Xavier Series (октябрь 2018)** | **Nvidia Jetson AGX Orin Series (март 2022)** | **Dual Intel Xeon Gold 6240R (2020)** | **RoboDeus (весна 2021)** |
| **Техпроцесс** | TSMC 20нм | TSMC 16нм | TSMC 12нм | TSMC 8нм | TSMC 14нм | TSMC 16нм |
| **Производительность FLP64** | — | — | 0,705 Tflop/s | 1,7 Tflop/s | 1.845 Tflop/s | 1,2 Tflop/s |
| **Производительность FLP32** | 0,512 Tflop/s | 0,649 Tflop/s | 1,4 Tflop/s | 3,3 Tflop/s | 3.69 Tflop/s | 4 Tflop/s |
| **Производительность FLP16** | 1,024 Tflop/s | 1,298 Tflop/s | 2,8 Tflop/s | 6,6 Tflop/s | 7.38 Tflop/s | 16 Tflop/s |
| **Энергопотребление** | 15 Вт | 10 Вт | 10 - 40 Вт | 16 - 40 Вт | 165 Вт | 30 – 70 Вт |
| **Цена (апрель 2022)** | 239$ | ~500$ | 2278$ | 1999$ | 1933$ | 150 т. руб. |

Подводя итоги сравнительного обзора СнК, можно сделать вывод, что наибольшей пиковой производительностью обладает гетерогенный СнК «RoboDeus». Также по сравнению с процессорами серии Intel Xeon Gold его можно считать более энергоэффективным в связи с возможностью настройки пониженного энергопотребления. В дальнейшем планируется построение гетерогенного вычислительного кластера с вычислительными узлами на основе СнК 1892ВМ248 для более эффективного применения микропроцессора в системах когнитивной обработки информации (рис. 1.6).

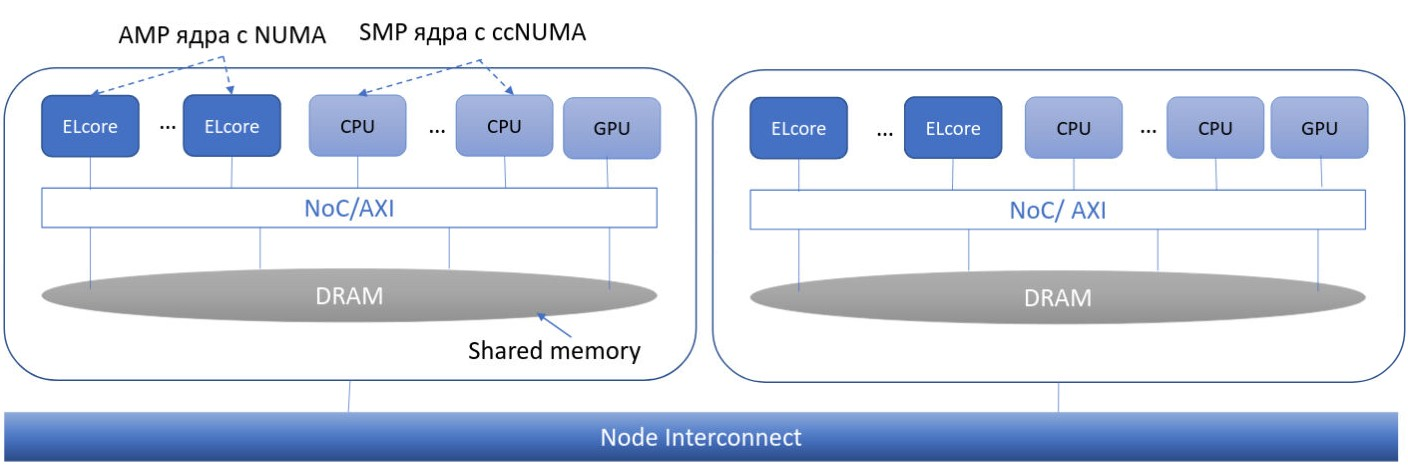


Рисунок 1.6 — Схема соединения кластера с вычислительными узлами на основе СнК «Robodeus»

Исследование составляющих информационной коммуникационной среды

Ключевой составляющей любой кластерной системы является информационная коммуникационная среда, которая позволяет процессам, выполняющимся в вычислительной сети, обмениваться данными, распределять задачи и синхронизировать работу. Коммуникационная среда состоит из аппаратно-программных средств, обеспечивающих данный обмен. Аппаратные средства включают в себя сетевое оборудование, такое как маршрутизаторы, коммутаторы, кабели и другие устройства, обеспечивающие физическое соединение между узлами кластера. Программные средства включают в себя операционную систему, драйверы устройств, протоколы связи и другие программы, которые обеспечивают логическое соединение между узлами кластера.

Таким образом, «сердце» современного суперкомпьютера коммутационная сеть или интерконнект проявляется на 3 уровнях (16):

1. Оборудование и топология сети, т.е. принцип физического объединения узлов каналами обмена данных.
2. Системное программное обеспечение, реализующее стандартные процедуры обмена данными (один-одному, один-всем, все-всем и т.п.).
3. Алгоритмы параллельного решения математической задачи, основанные на указанном системном программном обеспечении.

В системах с общей памятью информационный обмен между процессорами осуществляется через общую память, а в системах с распределенной памятью - путем копирования информации из оперативной памяти одного процессора в оперативную память другого с использованием специальной аппаратуры для передачи данных (рис. 1.7). Данное устройство может быть по-разному организовано (17):

1. В виде простого канала связи (симплексного или дуплексного) для последовательной передачи информации от одного блока ОЗУ к другому. Такая система реализована в распределенных системах, использующих высокопроизводительный интерфейс SCI. Он обеспечивает высокую скорость передачи данных (до 1 Гбит/с), низкую задержку и достаточную надежность. Этот интерфейс также поддерживает кэширование данных, что позволяет уменьшить количество обращений к общей памяти.
2. С использованием коммутаторов на основе матрицы коммутации. Это означает, что линии связи от отдельных процессорных узлов подсоединяются к специальным портам, каждый из которых может передавать данные независимо от других портов. Коммутаторы с матрицей коммутации обычно имеют более высокую стоимость, но обеспечивают более высокую производительность и надежность. В зависимости от задачи, которую необходимо решить, выбираются различные типы коммутаторов. Например, для создания вычислительного кластера используются коммутаторы InfiniBand, а для создания локальной сети в офисе - коммутаторы Fast Ethernet (из-за низкой стоимости оборудования). Кроме того, существуют коммутаторы Gigabit Ethernet, которые обеспечивают скорость передачи данных до 1 Гбит/с, и даже коммутаторы 100 Gigabit Ethernet, которые обеспечивают скорость передачи данных до 100 Гбит/с.
3. С использованием коммутаторов с общей шиной, к которой подсоединены несколько процессорных узлов с локальными ОЗУ. Такая шина представляет собой единую физическую среду, поэтому если несколько устройств пытаются передать данные одновременно, то возникают конфликты, и данные могут быть потеряны или повреждены. Коммутаторы с общей шиной обычно имеют меньшую стоимость, но не обеспечивают высокую производительность (например, классический Ethernet).
4. В виде комбинации вышеперечисленных технологий.

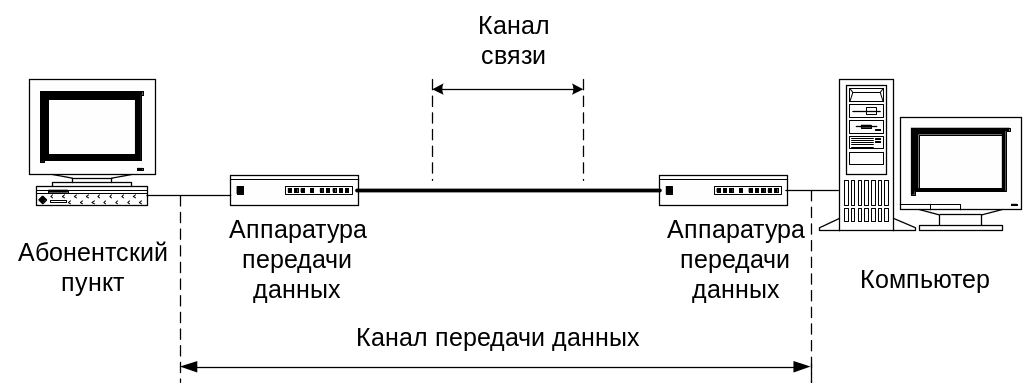


Рисунок 1.7 — Структура канала передачи данных

Из-за сетевой структуры кластеров, подобные системы обладают распределенной памятью по аналогии с архитектурой MPP. Эффективность распараллеливания во многих случаях сильно зависит от деталей подобной архитектуры, например, топологии соединения процессорных узлов. В зависимости от того, остается ли конфигурация взаимосвязей неизменной, по крайней мере пока выполняется определенное задание, различают сети со статической и динамической топологиями:

1. Статическая топология сети — это физическая конфигурация, которая не меняется без внешнего вмешательства, т.е. устройства подключены к определенным портам и не могут изменять свое расположение без переподключения.
2. Динамическая топология сети — это физическая конфигурация, которая может меняться автоматически в зависимости от изменений в сети. Например, в случае отказа одного из устройств, другие устройства могут автоматически перенастроить свое подключение для обеспечения непрерывной работы сети.

Выбор между статической и динамической топологией зависит от требований к гибкости и автоматизации управления сетью. Динамическая топология может быть более гибкой и удобной в управлении, но может требовать больших затрат на оборудование и настройку. Статическая топология может быть проще и дешевле в установке, но менее гибкой в управлении.

Очевидно, что самой эффективной была бы топология, в которой любой узел мог бы напрямую связаться с любым другим узлом. Однако в MPP-системах это технически трудно реализуемо. Обычно процессорные узлы в современных MPP-компьютерах образуют или двумерную решетку или гиперкуб. Ниже представлены самые распространенные топологии для организации узлов кластерной системы (18).

**Общая шина**

Пусть имеется N узлов, каждый из которых имеет по одному приёмнику и передатчику, тогда самым простым вариантом организации сети является соединить их все вместе. Сообщение с любого узла все остальные смогут принять сразу, но одновременная передача сообщения двумя или более узлами приведет к приёму мусора. Поэтому передатчики могут работать только поочерёдно, и в среднем на каждый узел приходится только 1/N пропускной способности шины (далее, пренебрегая различными физическими показателями, будем считать, что приёмник, передатчик и линия связи между ними способны пропустить условный единичный трафик). Можно заметить, что 1/N уменьшается при увеличении N, т. е. каждый узел сможет передавать всё меньше и меньше сообщений, передатчик узла будет всё больше простаивать, а приёмник принимать всё меньше адресованных ему сообщений.

**Однонаправленное кольцо**

Следующее, что можно построить, имея N приёмников и N передатчиков, это однонаправленное кольцо. Каждый передатчик передаёт сообщение приёмнику соседнего узла, конфликтов передачи здесь уже нет, но зато через каждый узел приходится пересылать и чужие сообщения. В среднем в сети одновременно присутствует по одному сообщению от каждого узла, но вопрос в том, за какое время они доберутся до получателя. Если отправить сообщение следующему узлу кольца, оно дойдёт за 1 шаг (от передатчика до приёмника), если отправить предыдущему, ему придётся сделать (N - 1) шагов, в сумме получается N. Если отправить сообщения следующей паре узлов, то 2 и (N - 2) тоже в сумме дают N, т.е. 2 сообщения от некоторого узла в сумме делают N шагов, прежде чем добираются до получателя. За счёт этого на 1 узел уже приходится 2/N пропускной способности связи, это в 2 раза лучше общей шины, но тоже падает с ростом N.

**Сегментированная сеть**

Если представить количество узлов следующим образом: N = S \* (M - 1), то сеть можно разделить на S сегментов по (M - 1) узлов, а каждый из сегментов подключить к коммутатору, имеющему S приёмников и S передатчиков, который способен пересылать сообщения из одного сегмента в другой. Если взять некоторое случайное сообщение, то с вероятностью 1/S отправитель и получатель будут в одном сегменте, а с вероятностью (1 - 1/S) нужно пройти через коммутатор. Если отправитель и получатель в одном сегменте, то сообщение пройдёт средний путь M/2, а если в разных, то M, т. е. в 2 раза больше. В среднем путь сообщения будет 1/S \* M/2 + (1 - 1/S) \* M = M - M/2S. Таким образом, если сеть разделена только на 1 сегмент, то путь получится как в кольце M/2 = (N+1)/2 (коммутатор можно считать обычным узлом), если 2, то 3/4\*M, если 3, то 5/6\*M и т. д.

**Звезда**

Если продолжить увеличивать число приёмников и передатчиков, то при S = M, средний путь будет M - M/2M = M - 1/2 = N + 1/2, т. е. порядка sqrt(N). Фактически, добавив S передатчиков, эффективность сети улучшится в S раз по сравнению с шиной. В предельном случае при S = N, т. е. имея в каждом сегменте всего по 1 узлу, будет получена классическая звезда. С такой организацией любое сообщение передаётся за 2 шага, от отправителя к коммутатору и далее к получателю, ситуация почти идеальная - все связи, передатчики и приёмники работают непрерывно. С точки зрения узла имеем единичную пропускную способность на передачу и приём, что в N раз лучше, чем в общей шине.

**Полносвязный граф**

В предыдущем случае уже была достигнута полная загрузка канала связи, дальнейшее повышение эффективности возможно только за счёт увеличения числа каналов в каждом узле. Чтобы передавать сообщения не за 2 шага, а за 1, нужно отказаться от коммутатора и перейти к прямым связям между узлами. В предельном случае получится полносвязный граф, который позволяет непрерывно передавать единичный трафик с задержкой в 1 шаг между любой парой узлов. На практике для больших N построить такую систему нереально, но подобная идея может использоваться внутри элементарных коммутаторов, из которых строится большая система.

**Двунаправленное кольцо**

Из двух путей однонаправленного кольца всегда можно выбрать более короткий, за счёт чего средний путь уменьшается в два раза, т. е. до N/4, а пропускная способность увеличивается в 2 раза за счёт двух передатчиков. То есть эта схема в 4 раза эффективнее однонаправленного кольца и в 8 раз эффективнее общей шины. При N = 2 или N = 3 получается полносвязный граф, в котором любой путь длиной в 1 шаг и каждый узел может непрерывно использовать оба передатчика. При N = 4 максимальная длина пути равна 2, в среднем узел может непрерывно использовать 1 передатчик, ситуация аналогична звезде с 4 узлами, но в ней фиксированная задержка передачи, поэтому при больших N звезда все-таки более эффективна. Рассматриваемая схема будет эффективнее однонаправленной сегментированной сети, если в ней менее 8 сегментов, поскольку у узлов нет второго передатчика.

**Двумерная сетка (двойной тор)**

Если N = R \* R, то все узлы можно соединить R горизонтальными и R вертикальными однонаправленными кольцами, средний путь до другого узла в таком случае - R, пропускная способность 2/R. Схема хуже звезды с центральным коммутатором, поэтому получается, что двумерные или трёхмерные торы просты в построении, но не слишком эффективны.

**Гиперкуб 2^P**

В этой схеме каждый узел имеет P передатчиков и приёмников, все связи между узлами двунаправленные, максимальный путь P шагов (при переходе в каждом измерении), средний путь P/2. За счёт этого в среднем узел может непрерывно использовать 2 из P своих передатчиков, а в остальное время через них проходят чужие сообщения. Если каждому узлу добавить еще по одному передатчику и приёмнику и соединить с максимально удалённым от него узлом, можно сократить максимальный путь до P/2. Средний при этом сократится не сильно, поскольку большинство путей и так имеет длину близкую к P/2 и движение в другую сторону не может их значительно укоротить.

Для обеспечения взаимодействия между программными процессами, помимо аппаратной составляющей, важной частью информационной коммуникационной среды является программное обеспечение. В системах класса MIMD обмен между процессами может быть организован следующим образом:

1. Если процессы имеют доступ к общей памяти (это может быть как однопроцессорный, так и многопроцессорный узел), то обмен осуществляется через неё. Такой подход соответствует модели параллельного программирования OpenMP. Для того чтобы избежать проблем с общей памятью, OpenMP предоставляет механизмы синхронизации, такие как блокировки и атомарные операции, а также позволяет использовать локальную память для хранения временных данных, которые не нужны другим потокам.
2. Если процессы работают в системе с распределённой памятью, то для организации взаимодействия используются специальные интерфейсы передачи сообщений между вычислительным процессом и каналом связи, что соответствует модели параллельного программирования MPI. Каждый процесс имеет свою собственную память, и данные могут быть переданы между процессами только с помощью явного вызова функций для отправки и получения сообщений.

Ввиду отсутствия совместно используемой памяти, единая коммуникационная среда обеспечивает взаимодействие узлов кластера между собой только посредством передачи сообщений. Поэтому для организации взаимодействия между процессорами одного узла и между разными узлами выбран стандарт интерфейса обмена данными MPI, широко используемый для параллельного программирования. Помимо того, что библиотека MPI поддерживает межпроцессорный обмен, MPI версии 3.0 и выше позволяет осуществлять межъядерный обмен с использованием разделяемой памяти. Это является еще одним определяющим фактором, поскольку, в отличие от других гетерогенных вычислительных систем, CPU, DSP и GPU ядра в процессоре «RoboDeus» располагаются на одной микросхеме, что позволяет организовать и использовать общую память.

Аналитический обзор существующих методов и средств анализа основных характеристик сетевых инфраструктур

Передача данных по сети связана с множеством характеристик, среди которых можно выделить: трафик, скорость передачи данных, пропускную способность, задержку, джиттер, потерю пакетов, шум, протоколы, безопасность, масштабируемость, надежность канала связи, достоверность передачи информации и т.п. Эти характеристики влияют на качество передачи данных и определяют эффективность работы сети. Рассмотрим некоторые из них (19):

1. Трафик — объём информации, передаваемой через компьютерную сеть за определённый период времени. Он может быть различного типа, например, передача файлов, отправка электронной почты, просмотр веб-страниц и т.д. Трафик измеряется в байтах.
2. Скорость передачи данных — это фактическая скорость потока данных, прошедшего через сеть. Определяется как отношение объема переданных данных за промежуток времени на продолжительность этого промежутка. Скорость передачи данных измеряется в битах в секунду (bits per second, bps), но может быть выражена в более крупных единицах (Mbps, Kbps, Gbps, Tbps и т.д.).
3. Пропускная способность (bandwidth) — это максимальное количество данных, которое может быть передано через сеть или канал связи за единицу времени. В случае однонаправленных пересылок равна максимальной скорости передачи сообщения от одного процесса другому, в случае двунаправленных пересылок учитывается максимальная скорость, с которой два процесса могут одновременно обмениваться данными. Измеряется также как и скорость в битах в секунду.
4. Задержка или латентность (latency) — это время начальной задержки при передаче данных между процессами. Определяется как интервал времени, начинающийся, когда последний бит входного кадра достигает входного порта, и заканчивающийся, когда первый бит выходного кадра появляется на выходном порту.

Согласно теории информации пропускная способность зависит от полосы пропускания канала связи и отношения сигнал/шум и может быть рассчитана по формуле Клода Шеннона вне зависимости от принятого способа физического кодирования (20):

|  |  |
| --- | --- |
|  | (1) |

где – максимальная пропускная способность канала (бит/с), – ширина полосы пропускания канала (Гц), – мощность полезного сигнала, – мощность шума/помех. Причем полоса пропускания – это диапазон частот, в пределах которого амплитудно-частотная характеристика (АЧХ) линии связи достаточно равномерна для того, чтобы обеспечить передачу сигнала без существенного искажения его формы.

Реальная скорость передачи данных по каналу связи обычно меньше его пропускной способности и зависит от:

* параметров каналообразующей аппаратуры;
* способов организации передачи данных;
* количества узлов, подключенных к каналу связи.

Близким по сути к формуле Клода Шеннона является соотношение, полученное Найквистом, которое также определяет максимально возможную пропускную способность линии связи, но без учета шума на линии:

|  |  |
| --- | --- |
|  | (2) |

где – количество различимых состояний информационного параметра.

Для измерения коммуникационных характеристик кластера на уровне передачи сообщений необходимо организовать пересылку сообщений длины между двумя процессами (21). Эти действия повторяются раз с целью минимизировать погрешность за счет усреднения, а головной процесс измеряет время , затраченное на все эти обмены. Итоговое значение пропускной способности определяется по следующей формуле:

|  |  |
| --- | --- |
|  | (3) |

Масштабируем для случая с несколькими узлами (табл.1.2). Пусть — число узлов, а — число каналов связи. Если — длина передаваемого сообщения в байтах, то суммарный объем проходящей по сети информации . Отношение количества всей переданной информации к времени , затраченному на передачу информации, считается суммарной пропускной способностью . Рассуждая аналогичным образом, i-тый узел передает и принимает следующее количество информации — . Отношение переданного на узле количества информации к времени , считается локальной пропускной способностью . Дополнительно можно определить среднюю пропускную способность , как отношение суммарной к числу задействованных каналов .

Таблица 1.2 — Расчет пропускной способности в различных логических топологиях

|  |  |  |  |
| --- | --- | --- | --- |
|  | Звезда | Кольцо | Полный граф |
| N(P) | P - 1 | P | P(P - 1) |
| R(local) | 2L(P - 1)/T | 4L/T | 2L(P - 1)/T |
| R(total) | 2L(P - 1)/T | 2LP/T | LP(P - 1)/T |
| R(avg) | 2L/T | | |

Методика измерения латентности с использованием библиотек на уровне передачи сообщений, а именно посредством вызовов MPI функций заключается в том, чтобы замерить время, необходимое на передачу сигнала или сообщения нулевой длины. Размер служебных данных такого пакета разнится в зависимости от сетевой технологии и получается, что в различных сетевых технологиях — различная длина служебных данных, что влияет на величину задержки. При этом, для снижения влияния погрешности и низкого разрешения системного таймера, важно повторить операцию посылки сигнала и получения ответа большое число раз. Таким образом, если время на итераций пересылки сообщений нулевой длины туда и обратно составило , то латентность измеряется следующим образом:

|  |  |
| --- | --- |
|  | (4) |

Исходя из формул для расчета пропускной способности и латентности на уровне передачи сообщений, можно выразить время , необходимое на передачу сообщения длиной :

|  |  |
| --- | --- |
|  | (5) |

Еще одним методом анализа коммуникационных характеристик сетевых инфраструктур является использование теории графов транспортной сети (22). Для этого исследуемая сеть представляется в виде ориентированного графа с конечным числом вершин без петель и кратных ребер. В таком случае, для вершины множество всех входящих в неё ребер обозначается через , а множество выходящих - .

Транспортной сетью (сетью) называется орграф, в котором:

1. Каждому ребру приписано положительное число , называемое пропускной способностью ребра. Функция такая, что для любого значение функции равно нулю.
2. Выделены две вершины и , называемые соответственно источником и стоком, при этом = , т.е. из источника ребра только выходят, а в сток только входят.

Пусть задана сеть с множеством вершин и множеством ребер . Пусть – функция с вещественными значениями, определенная на множестве . Для вершины обозначим:

|  |  |
| --- | --- |
| 𝑓+(𝑥) = ∑𝑓(𝑒); 𝑒 ∈  𝑓−(𝑥) = ∑𝑓(𝑒); 𝑒 ∈ | (6)  (7) |

Функция называется потоком в сети , если она удовлетворяет условиям:

* ограниченности: 0 ≤ 𝑓(𝑒) ≤ 𝑐(𝑒);
* сохранения: 𝑓+(𝑥) = 𝑓−(𝑥).

Разрезом сети называется множество дуг, удаление которых из сети приводит к тому, что исток и сток оказываются несвязанными. Пропускной способностью разреза называется число, равное сумме пропускных способностей дуг этого разреза. Информация, двигаясь из точки в точку, обязательно пройдет хотя бы один раз по какой-либо дуге разреза, поэтому справедливо то, что поток в сети всегда равен этому разрезу. Если на сети задан поток и разрез, то величина , представляющая сумму потоков по всем ребрам разреза, называется потоком через разрез.

На рисунке 1.8 изображена некоторая сеть с двумя разрезами: I и II. При разрезе I образовалось два подмножества вершин сети: подмножество A = {1, 2} и B = {3, 4, 5}, а ребрами, образующим разрез, стали (1, 3), (1, 4), (2, 4). При разрезе II образовались подмножества A = {1, 2, 3, 4} и B = {5} с образующими разрез ребрами (3, 5) и (4, 5).

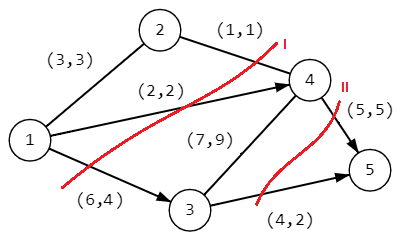


Рисунок 1.8 — Пример транспортной сети

Таким образом, для разреза I:

R(I) = r(1,3) + r(1,4) + r(2,4) = 6 + 2 + 1 = 9.

X(I) = x(1,3) + x(1,4) + x(2,4) = 4 + 2 + 1 = 7.

Для разреза II: R(II) = 9, X(II) = 7.

Такой способ позволяет оценить максимально возможную пропускную способность в сети любой топологии, но требует заранее известных значений пропускной способности каналов связи и становится малоэффективным в сетях, где отсутствует сложная топология узлов.

Таким образом, проанализировать основные коммуникационные характеристики гетерогенной кластерной системы можно, представив её сетевую инфраструктуру в виде графа транспортной сети или, например, в виде системы массового обслуживания (СМО). Но у такого подхода, несмотря на достаточную универсальность, есть существенные недостатки:

* крайнее упрощение узлов кластерной системы;
* сведение неоднородной инфраструктуры к однородной;
* ограниченное количество подходящих для анализа сетей;
* отсутствие возможности учета программных и аппаратных задержек;
* сложность определения узких мест кластерной системы.

С другой стороны, измерение и расчет основных коммуникационных характеристик на уровне передачи сообщений считается более корректным, но требует предварительной установки и настройки специального ПО (например, интерфейса для передачи сообщений MPI), что в рамках гетерогенных систем может оказаться достаточно трудоемкой задачей.

Постановка целей и задач диссертационных исследований

Цель диссертационного исследования заключается в повышении быстродействия процесса передачи сообщений между узлами гетерогенной кластерной системы.

В соответствии с целью исследования в диссертации необходимо решить следующие задачи:

* аналитический обзор существующих методов и средств анализа коммуникационных характеристик сетевых инфраструктур;
* формализация процесса передачи сообщений между однородными и неоднородными узлами кластера;
* разработка методики расчета и анализа основных характеристик коммуникационной инфраструктуры кластера;
* разработка методики ранжирования стандартных интерфейсов межузловых коммуникаций;
* разработка методики ускорения выполнения MPI-приложений на кластере гетерогенного типа;
* разработка алгоритма измерения основных сетевых характеристик кластерной системы;
* проведение экспериментальных исследований, подтверждающих достоверность полученных результатов.

## Выводы по главе 1

В ходе исследования произведены следующие работы:

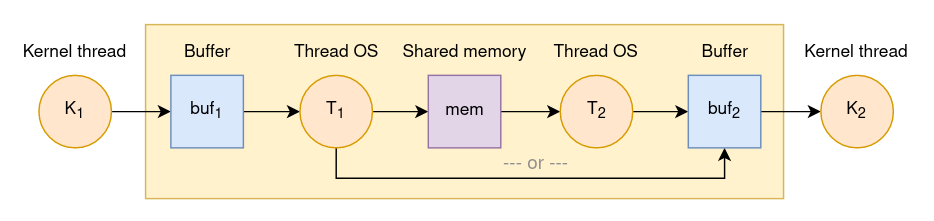
* проанализированы основные направления разработки суперкомпьютеров;
* исследованы особенности построения гетерогенных систем на кристалле, в частности, СнК «RoboDeus»;
* выделены основные коммуникационные характеристики;
* проведен аналитический обзор существующих методов и средств анализа коммуникационных характеристик сетевых инфраструктур;
* выделены достоинства и недостатки распространенных MPI бенчмарков для измерения пропускной способности сети;
* поставлена цель и задачи исследования.

# ГЛАВА 2. ФОРМАЛИЗАЦИЯ ПРОЦЕССА ПЕРЕДАЧИ СООБЩЕНИЙ МЕЖДУ ДВУМЯ ПРОЦЕССАМИ В КЛАСТЕРАХ ГЕТЕРОГЕННОГО ТИПА



Анализ операций при передаче сообщений между двумя процессами

Обмен данных между двумя параллельными процессами заключается в передаче данных от потока-отправителя к потоку-получателю с условием, что эти потоки выполняются в контекстах разных процессов. Если в случае, когда потоки выполняются в рамках одного процесса, для обмена можно просто использовать глобальные структуры данных, то при выполнении в разных процессах приходится организовывать специальный канал передачи данных (рис. 2.1).

Рисунок 2.1 — Обобщенная структура канала передачи данных

При пересылке данных пользовательский поток-отправитель K1 записывает данные в буфер buf1. Поток ядра ОС T1 считывает эти данные и в зависимости от способа передачи записывает их в общую память mem (передача группы байтов, т. е. сообщения) или сразу в буфер buf2 (передача непрерывной последовательности байтов, т. е. потока). При передаче данных сообщениями поток T2 считывает данные из общей памяти mem и записывает в буфер buf2. На этапе, когда поток-получатель K2 считывает данные из буфера buf2, полудуплексная передача в одном направлении завершена. Далее будет рассматриваться именно более сложная передача данных сообщениями, причем в обоих направлениях, т. е. дуплексная.

Для разработки модели передачи данных между процессами, находящимися как на одном вычислительном узле, так и на разных, также необходимо упрощенно описать работу с общей памятью, т.е. основные запросы, когда центральный процессор пытается обратиться на чтение или запись.

Современные компьютеры, а тем более кластеры, имеют многоуровневую архитектуру памяти, что позволяет совмещать высокую скорость работы с памятью и большой объем доступной памяти. В основном она состоит из регистровой памяти CPU, нескольких кэшей (оптимально 2х или 3х) и оперативной памяти. Все следующие уровни памяти уже считаются вторичными. Как правило, каждое ядро CPU имеет собственный кэш данных L1:D и инструкций L1:I, кэш L2 (может быть как общим для всех ядер, так и раздельным), а также имеет доступ к общему кэшу L3 и основной памяти RAM. Расположение запрашиваемых данных в памяти очень важно, поскольку время выполнения задачи обратно пропорционально зависит от объема данных, внесенных в кэш. Возможные случаи расположения данных отражены в таблице 2.1 (27).

Таблица 2.1 — Возможные случаи расположения данных в памяти

|  |  |  |
| --- | --- | --- |
| № | Расположение данных | Иллюстрация |
| 1 | В кэш-памяти первого уровня (L1) ядра, делающего запрос |  |
| 2 | В кэш-памяти второго уровня (L2) ядра, делающего запрос |  |
| 3 | В общей кэш-памяти третьего уровня (L3) |  |
| 4 | В общей основной памяти с произвольным доступом (RAM) |  |

Предположим, ядро CPU пытается обратится на запись новых данных в строку, отсутствующую в его локальной кэш-памяти. Осуществляется поочередный поиск на всех уровнях памяти, результатом которого может быть либо cachemiss (промах), либо cachehit (попадание). Поскольку данные новые, то при запросе на запись в L1 получаем кэш-промах. Аналогично получаем кэш-промах при обращении на следующие уровни иерархии, т. е. к L2 и L3. Если бы копия данных была найдена в кэш-памяти третьего уровня, но при этом оказалась не актуальна, то производились бы дополнительные обращения к L1 и L2 того ядра, которое хранит актуальную копию. Но поскольку в L3 ничего нет, то итоговое обращение будет к основной памяти RAM, в которой запрашиваемые данные также отсутствуют. Последующее поведение кэша при выполнении записи зависит от политики кэша при промахе:

1. В случае сквозной записи (writethrough), любая запись в кэш (с попаданием или без) приводит к записи в память. То есть если данные по запрашиваемому адресу отсутствуют, новая информация, обязательно записывается в основную память. В случае записи с размещением (writeallocate) данные синхронно записываются на каждый уровень кэш-памяти, замещая данные, к котором наиболее редко обращались. Такой подход обеспечивает полную согласованность данных на всех уровнях.
2. При отложенной записи (writeback), запись производится только в кэш-память. Измененная строка кэша записывается обратно в хранилище непосредственно перед её заменой, т. е. при вытеснении строки другой строкой. Но в случае промаха на всех уровнях, запись выполняется только в основную память. Такой порядок необходим, чтобы в случае сбоя записи в хранилище, кэш не был записан и не получилось несогласованности между основной и кэш-памятью.

Как правило, в многопроцессорных системах с общей оперативной памятью, но раздельными уровнями кэш-памяти при кэш-промахе используется сквозная запись в основную память с размещением в кэш памяти. При построении временной диаграммы (рис 2.2) использовалось предположение о том, что поиск в кэшах различных уровней идет последовательно согласно порядку поиска, т.к. большинство кэш-подсистем предусматривает соединение каждого последующего уровня с предыдущим, а не с централизованным коммутатором (28).

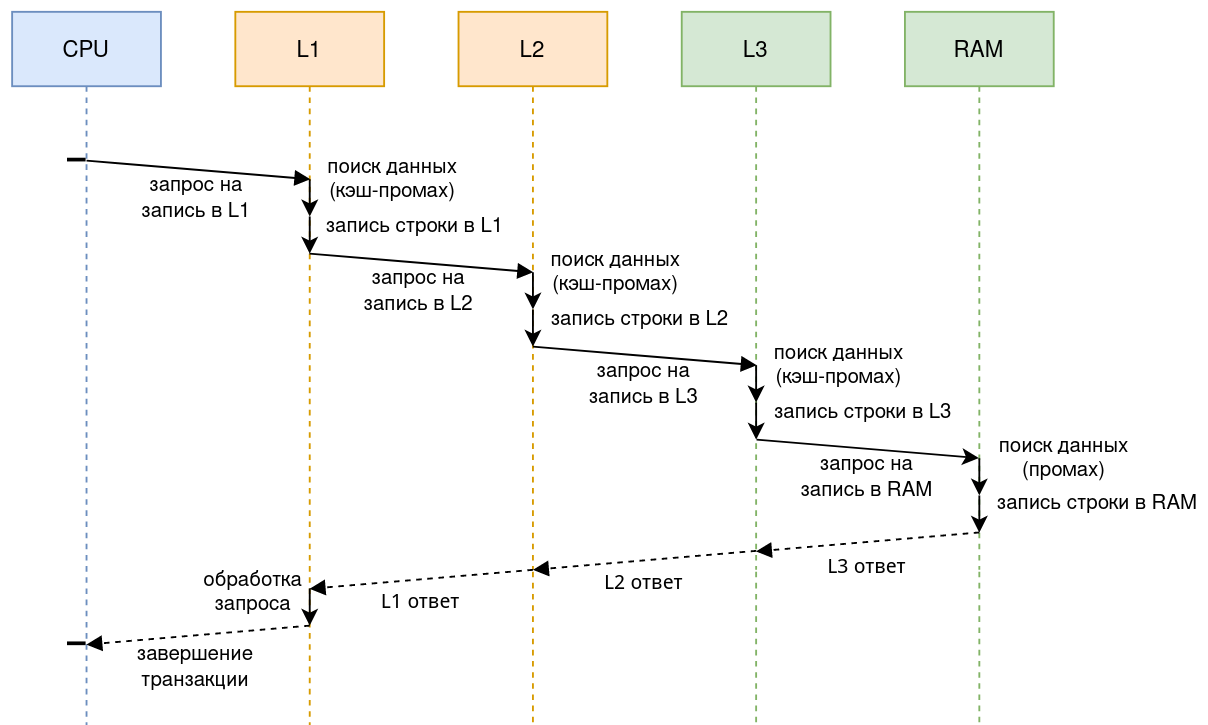


Рисунок 2.2 — Временная диаграмма, отражающая взаимодействие процессорного ядра с компонентами подсистемы памяти при выполнении операций записи

Далее другому ядру CPU необходимо считать новые данные. Аналогично запросу на запись, при попытке считать необходимые данные из собственных кэшей L1 и L2 получаем кэш-промах. Часть запросов к общему кэшу L3 также завершится промахом, но часть данных найдется, т.е. произойдет попадание. Оставшиеся данные будут найдены в общей памяти RAM. Далее в зависимости от политики кэша, либо данные будут сначала считаны в кэши более низкого уровня, и только потом возвращены (readallocate), либо данные будут сразу возвращены, минуя запись в кэш (readnoallocate) (29). Рассмотрим именно первый случай, т. к. чтение без размещения считается тривиальным и почти никогда не используется (рис. 2.3).

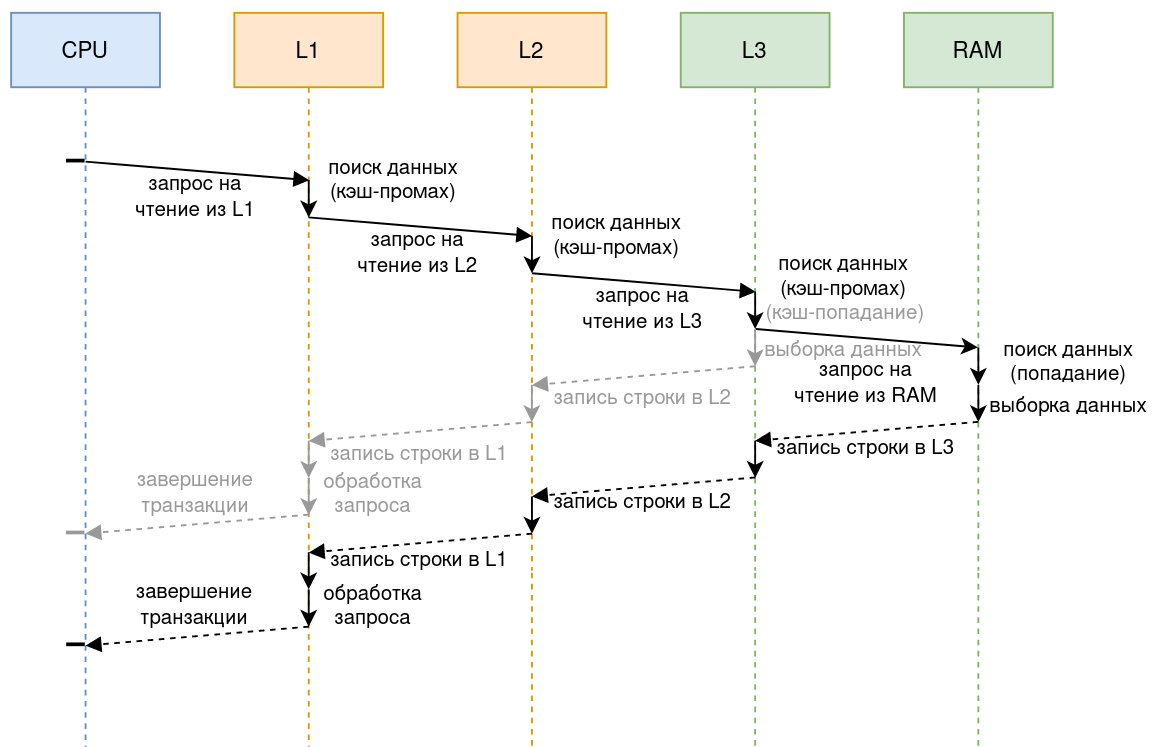


Рисунок 2.3 — Временная диаграмма, отражающая взаимодействие процессорного ядра с компонентами подсистемы памяти при выполнении операций чтения

Построенные временные диаграммы упрощены по сравнению с реальной последовательностью действий, например, не учтена матрица коммутации NoC и обращения к/от контролеров когерентности кэша (ССС) и контролеров памяти. Но несмотря на это, диаграммы отражают максимально возможную последовательность операций при записи и чтении новых данных, следовательно, с их помощью можно оценить предел времени записи и чтения сверху. Для более точной оценки следует учесть различные архитектуры вычислительных узлов кластера, политику кэша и используемые протоколы поддержания кэш-когерентности памяти, обеспечивающие решение конфликтов, происходящих при обращении к общим данным.

Архитектура многоядерного микропроцессора «RoboDeus» (1892ВМ248) была разобрана в предыдущей главе, поэтому на основе схемы из пункта 1.2 можно построить структурную схему (представлена на рис. 2.4), отражающую только те блоки, которые могут быть задействованы при передаче данных между процессами на одном узле кластера (но без учета контроллеров).

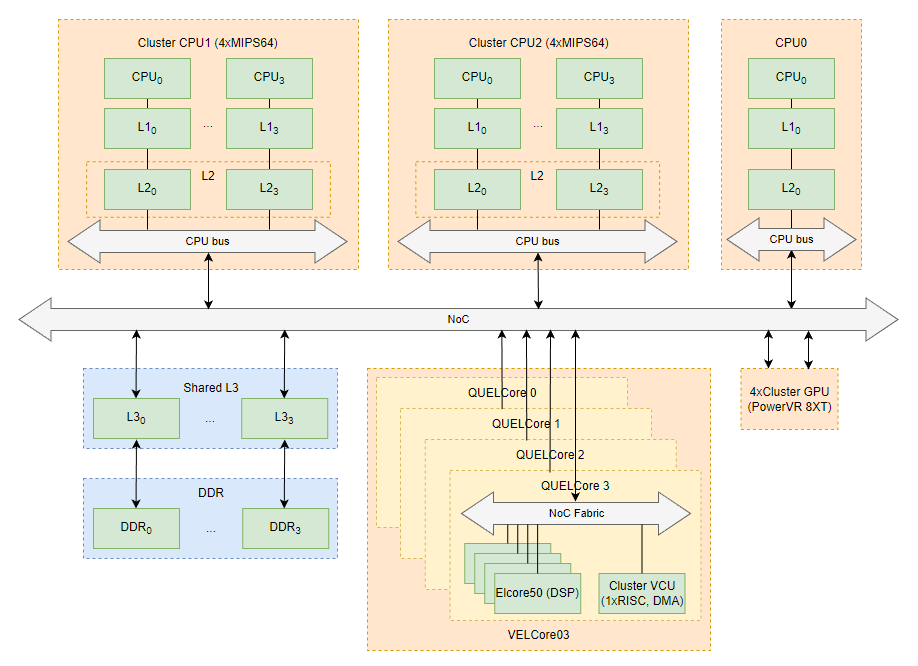


Рисунок 2.4 — Структурная схема многоядерного микропроцессора «RoboDeus»

Каждый кластер ядер CPU – это четырёхъядерный MIPS I6500 Daimyo с двухуровневой кэш-памятью, поддерживающей протокол когерентности MESI (modified, exclusive, shared, invalid) и защиту ECC+EDC (код с детектированием и исправлением ошибок) (30). Ядро I6500 поддерживает передачу данных из кэша в кэш (cache-to-cache) и такие атрибуты кэширования, как uncached, writeback, writeallocate и readmissesrequestshared. На диаграммах, описанных в ранее, предполагается, что запись и чтение осуществляются с промежуточным размещением в кэш-памяти (allocate), что поддерживается в исследуемом MIPS процессоре, но тем не менее, последовательность операций необходимо модифицировать с учетом того, что запросы на запись и чтение могут передаваться широковещательно (рис. 2.5).

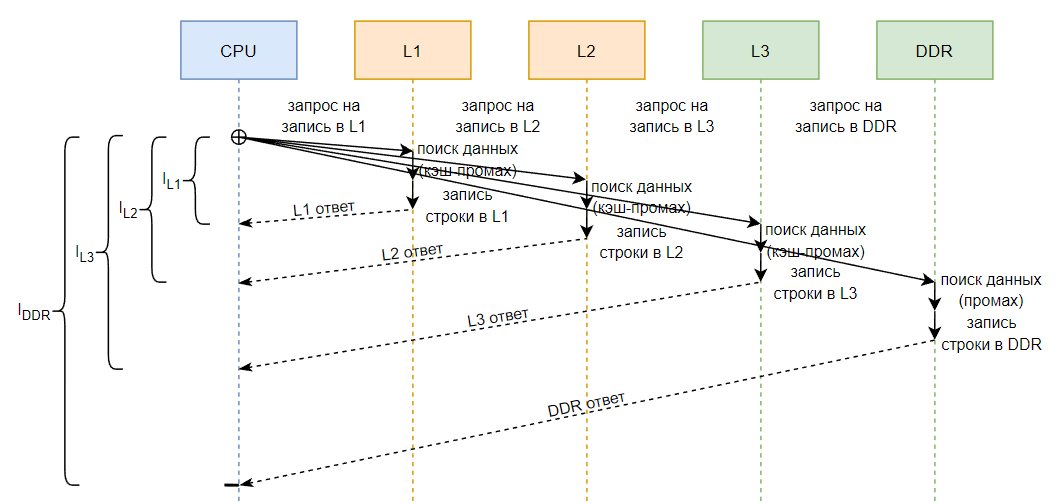


Рисунок 2.5 — Временная диаграмма, отражающая взаимодействие процессорного ядра с компонентами подсистемы памяти при широковещательных запросах

Протокол MESI поддерживает 4 состояния строки кэша, в том числе: строка может быть эксклюзивной (exclusive), т. е. отсутствовать в любом другом кэше, и разделяемой (shared), т. е. присутствовать в нескольких кэшах. Но в любом из 4х состояний, включая exclusive и shared, копия строки должна находится в основной памяти. Таким образом, часть передаваемого сообщения может быть соответственно записана в , но весь объем сообщения обязательно будет записан в , что и следует оценивать. При малом размере сообщения и передаче внутри одного кластера ядер CPU, возможно было бы использовать передачу данных cache-to-cache с использованием общего кэша L2, но далее будет рассматриваться именно передача между кластерами ядер CPU с использованием общей памяти.

В исследуемом стенде помимо отладочной платы, сконструированной на основе интегральной микросхемы «RoboDeus», используется хост-машина, которая представлена персональным компьютером с восьмиядерным микропроцессором архитектуры x86, например, IntelCorei7-9700KF. Указанный процессор основан на микроархитектуре Coffee Lake (CFL) и, как и большинство процессоров Intel, поддерживает протокол когерентности MESIF (modified, exclusive, shared, invalid, forward), являющийся модификацией протокола MESI с дополнительным состоянием F (31). Подробности о своей реализации обеспечения согласованности кэшей документация Intel не раскрывает. По словам разработчиков, CPU Intel может динамически адаптировать свою стратегию когерентности во время выполнения в соответствии с рабочей нагрузкой. Не будет одной фиксированной реализации когерентности кэша, их будет много. Какая из них используется во время выполнения, зависит от рабочей нагрузки.

Микроархитектура Coffee Lake (рис. 2.6) состоит из пяти основных компонентов: ядер CPU, подсистемы памяти (трехуровневая кэш-память + оперативная память DDR), кольцевого межсоединения, системного агента и встроенной графики GPU (32). Архитектура микросхемы «RoboDeus» без учета кластера обработки данных VELCore03, схожа с восьмиядерной вариацией CFL по используемым комплектующим. Основные различия заключаются в топологии ядер (каждое ядро по отдельности, а не в кластерах по 4 ядра), организации кэша третьего уровня (у каждого ядра собственный кэш L3, но он доступен другому ядру по общей шине) и коммутации (межсоединениеRing bus). Несмотря на это, если рассматривать последовательность действий при операциях записи и чтения не между всеми доступными ядрами, а только между двумя, то диаграммы получатся аналогичные.

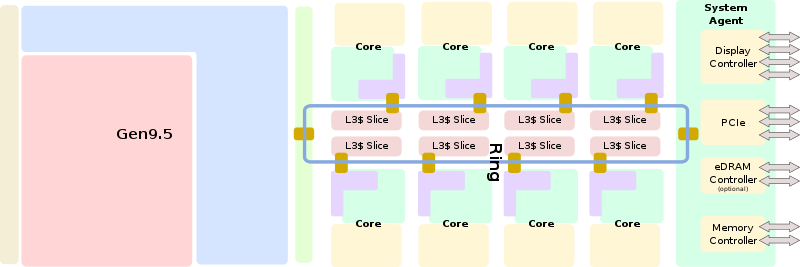


Рисунок 2.6 — Структурная схема восьмиядерного микропроцессора архитектуры Coffee Lake

Разработка математической модели передачи сообщений на одном узле кластера

На основе диаграмм последовательности, разработанных в пункте 2.1, можно построить математическую модель, отражающую время передачи сообщений между двумя процессами на одном узле кластера. При разработке будем считать, что узел — это именно микропроцессор «RoboDeus», поэтому все используемые обозначения будут соответствовать обозначениям его элементной базы (рис. 2.4). Несмотря на это, полученная модель также будет применима, если в узлах кластера находятся процессоры с архитектурой, схожей с архитектурой/комплектующими микропроцессора «RoboDeus» (например, IntelCorei7-9700KF).

Пусть на одном ядре из любого кластера ядер (clusterCPU1 или clusterCPU2) генерируется сообщение для передачи. Отношение объема передаваемого сообщения к пропускной способности CPU отражает суммарное время , затрачиваемое на генерацию сообщения (33):

|  |  |
| --- | --- |
|  | (1) |

В формуле используется пропускная способность CPU – , которая в общем случае определяется следующим образом:

|  |  |
| --- | --- |
| , | (2) |

где – тактовая частота, т. е. базовая частота с определенным множителем,

– разрядность или ширина системной шины,

– количество каналов передачи.

В дальнейшем пропускную способность потребуется рассчитывать отдельно для каждого уровня памяти или коммутации:

* – пропускная способность ядра CPU,
* – пропускная способность кэша первого уровня,
* – пропускная способность кэша второго уровня,
* – пропускная способность матрицы коммутации NoC,
* – пропускная способность кэша третьего уровня,
* – пропускная способность оперативной памяти.

Поэтому для облегчения дальнейших изысканий введем множество, включающее перечисленные выше пропускные способности от CPU до L3 (т.е. без учета оперативной памяти):

|  |  |
| --- | --- |
|  | (3) |

На временных диаграммах из пункта 2.1 можно заметить, что несмотря на то, какая операция осуществляется (запись или чтение), на каждом уровне системы памяти в любом случае будет произведен как поиск данных, так и запись строки (даже в случае чтения). Помимо этого, к некоторым уровням нельзя подключиться напрямую, миную предшествующие уровни, например, оперативная память DDR доступна только из кэша L3. Таким образом, чтобы построить математическую модель передачи сообщения следует учесть все уровни кэш-памяти, оперативную память и даже матрицу коммутации NoC.

Чтобы рассчитать количество данных, проходящих по шинам, необходимо количество кэш-промахов предыдущего уровня памяти умножить на размер пакета данных, проходящих по шине для одного запроса. Согласно пункту 2.1 при записи новых данных в область общей памяти запросы на любой уровень памяти заканчиваются кэш-промахом, следовательно, по каждой шине проходит сообщение в полном объеме. Таким образом, можно подсчитать время, затрачиваемое на передачу данных через собственный кэш L1, общую область кэша L2 и матрицу коммутации NoC:

|  |  |
| --- | --- |
|  | (4), (5), (6) |

Микросхема включает общий кэш третьего уровня (L3), который разделен на равные блоки по четырем каналам памяти. Каждый блок кэша L3 отвечает за свои собственные диапазоны памяти и такое разделение гарантирует, что разные кэши не будут содержать одни и те же строки кэш-памяти. Разбиение пространства адресов памяти по четырем блокам L3 производится на основе схем либо с чередованием, либо без. Следовательно, расчет затрачиваемого времени в общем виде выглядит следующим образом:

|  |  |
| --- | --- |
|  | (7) |

где - доля от общего объема сообщения , передаваемая по одному из четырех каналов памяти L3.

Поскольку кэш L3 разделен на равные по структуре блоки, то и пропускная способность у них одинаковая:

|  |  |
| --- | --- |
|  | (8) |

Отсюда вытекает следующее соотношение:

|  |  |
| --- | --- |
|  | (9) |

при условии, что

Каждому блоку L3 соответствует блок оперативной памяти DDR, поэтому время рассчитывается аналогичным образом:

|  |  |
| --- | --- |
|  | (10) |

Таким образом получается время, затрачиваемое на генерацию сообщения, обработку и прохождение через все уровни кэш-памяти и запись в основную память:

|  |  |
| --- | --- |
|  | (11) |

или

|  |  |
| --- | --- |
|  | (12) |

Время, затрачиваемое на операцию чтения рассчитывается аналогично, но с учетом того, что из DDR считывается только та часть от общего объема сообщения , которая не помещается в L3. Искомая часть находится в зависимости от среднего значения количества кэш-промахов при поиске строки кэша размером в L3 при операции чтения:

|  |  |
| --- | --- |
|  | (13) |

Прежде чем перейти к расчету общего времени , необходимо учесть суммарную задержку или латентность , иначе получившееся значение будет значительно меньше реального (34). Как правило, средние значения аппаратных задержек между различными узлами коммутации указываются в документации к микросхеме и выражены в тактах. Такт ядра процессора – это промежуток между двумя импульсами тактового генератора, который синхронизирует выполнение всех операций процессора. Время одного такта обратно тактовой частоте процессора. Таким образом получается время суммарной задержки от CPU до DDR или аппаратная составляющая латентности:

|  |  |
| --- | --- |
|  | (14) |

где – количество записываемых/считываемых строк сообщения,

– время одного такта,

– количество тактов до DDR согласно документации.

Также следует учесть программную составляющую латентности , которая накапливается из-за различных системных вызовов и подготовки как к началу передачи информации, так и к завершению её получения посредством интерфейса передачи сообщений MPI. Итак, расчет полного времени, затрачиваемого на передачу сообщения между ядрами CPU из разных кластеров (CPU1 и CPU2) в одну сторону выглядит следующим образом:

|  |  |
| --- | --- |
|  | (15) |

где - суммарная задержка или латентность,

- доля от объема передаваемого сообщения , доступная только из оперативной памяти при операции чтения,

-коэффициент пропускной способности.

Разработка математической модели передачи сообщений на разных узлах кластера

В предыдущем пункте сформулирована математическая модель, позволяющая оценить время, затрачиваемое на передачу данных между двумя процессами посредством интерфейса передачи сообщений MPI. Но для того, чтобы корректно воспользоваться данной моделью, необходимо придерживаться следующих ограничений:

1. Во-первых, модель не учитывает возможность передачи данных cache-to-cache через общий кэш L2, который используется при малом размере сообщений и передаче внутри одного кластера ядер CPU. Т.е. рассматривается передача данных только между разными кластерами ядер CPU с использованием общей памяти L3 или DDR.
2. Во-вторых, архитектура исследуемого процессора должна в некоторой степени соответствовать архитектуре микропроцессора «RoboDeus», при условии, что рассматриваются комплектующие и их топология, необходимые для выполнения последовательности действий при операциях записи и чтения не между всеми доступными ядрами, а только между двумя. Например, если система кэш-памяти двухуровневая и оценивается кластер не CPU, a DSP ядер, то использование сформулированной модели приведет к некорректным результатам.
3. В-третьих, предполагалось, что взаимодействие между процессами осуществляется только в рамках одного вычислительного узла путем передачи сообщений через область общей памяти, т.е. если процессы находятся на разных вычислительных узлах, потребуется учесть, что взаимодействие будет обеспечиваться через механизм сетевых сокетов.

Если для того, чтобы избавиться от первых двух ограничений, необходимо провести дополнительный анализ и учесть большее количество факторов (некоторые из которых относятся к категории случайных), влияющих на математическую модель, то для того, чтобы модель учитывала возможность передачи сообщений между процессами, находящимися на разных вычислительных узлах кластера, потребуется минимальная модификация. Тем более, поскольку технология MPI является одним из самым распространённых стандартов программирования для обеспечения взаимодействия узлов в системах с распределенной памятью, к которым, в частности, относятся гетерогенные кластеры, то бессмысленно оценивать время передачи сообщений только внутри одного вычислительного узла, а не при взаимодействии хотя бы двух.

Существует несколько способов организации межкомпьютерной связи, но при построении кластера, как правило, организуют локальную сеть (LAN), т.е. сеть, охватывающую небольшую территорию и использующуюся для обмена данными между узлами (компьютерами, рабочими станциями и т.д.) в пределах одного здания. Основные способы организации локальной сети (рис. 2.7):

1. Объединение рядом расположенных узлов сети через их коммуникационные порты, посредством специального кабеля.
2. Передача данных от одного узла к другому посредством маршрутизатора или модема с использованием проводных, беспроводных или спутниковых линий связи.

В дальнейшем рассматривается именно вариант с использованием проводного соединения между узлами, поскольку он считается наиболее распространенным способом организации локальной сети ввиду высокой скорости, стабильности работы и безопасности.

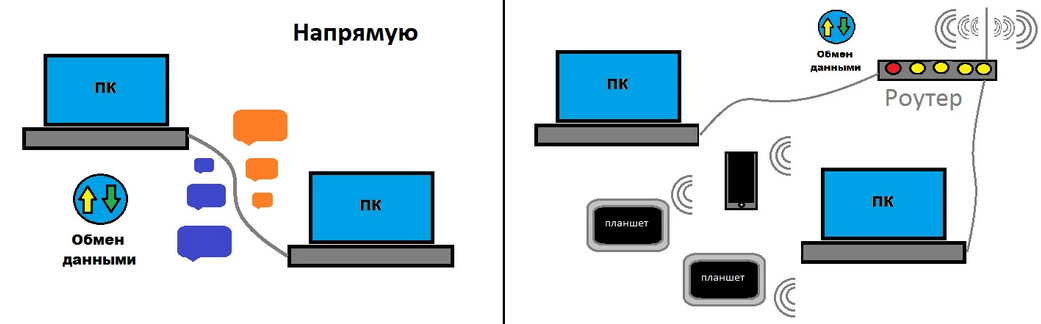


Рисунок 2.7 — Основные способы организации локальной сети

При организации даже самого простого вычислительного кластера, как правило, стараются физически разделить сеть как минимум на два сегмента (35). В одном сегменте работает сетевая файловая система (NFS), которая сильно нагружает коммуникационную сеть кластера, а в другом происходит обмен данными, который осуществляют между собой части параллельной программы, работающие на разных вычислительных узлах кластера. Таким образом, желательно, чтобы все узлы имели по два сетевых интерфейса (две сетевые карты) и два набора коммутаторов. На рисунке 2.8 изображена схема сети простого вычислительного кластера, в которой Ethernet-соединения используются для административных целей, а InfiniBand-соединения предназначены для вычислительного трафика. Поскольку оборудование InfiniBand зачастую отсутствует, то наибольший интерес представляет именно Ethernet-кабель, который может нести как административную, так и вычислительную нагрузку.

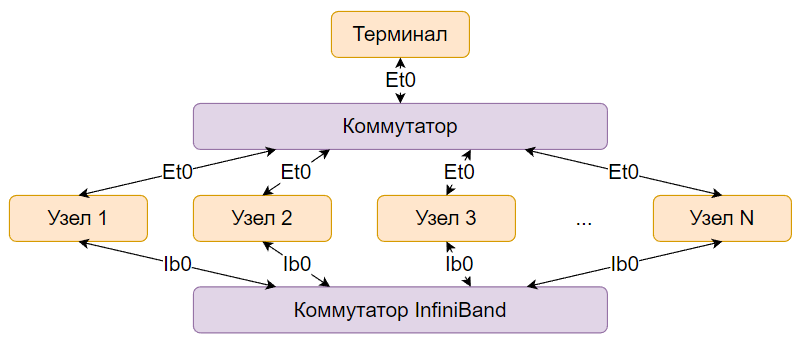


Рисунок 2.8 — Схема сети простого вычислительного кластера

После уточнения способа организации локальной сети между двумя вычислительными узлами, можно модифицировать математическую модель, построенную в предыдущем пункте таким образом, чтобы она отражала время передачи сообщений между двумя процессами на разных узлах кластера (узлы кластера могут быть, как однородные, так и гетерогенные, но для первоначального построения модели считаем, что они однородны). Допустим, приложения MPI\_1 и MPI\_2 на соответствующих однородных узлах кластера управляют процессом передачи данных путем обмена сообщениями (рис. 2.9).

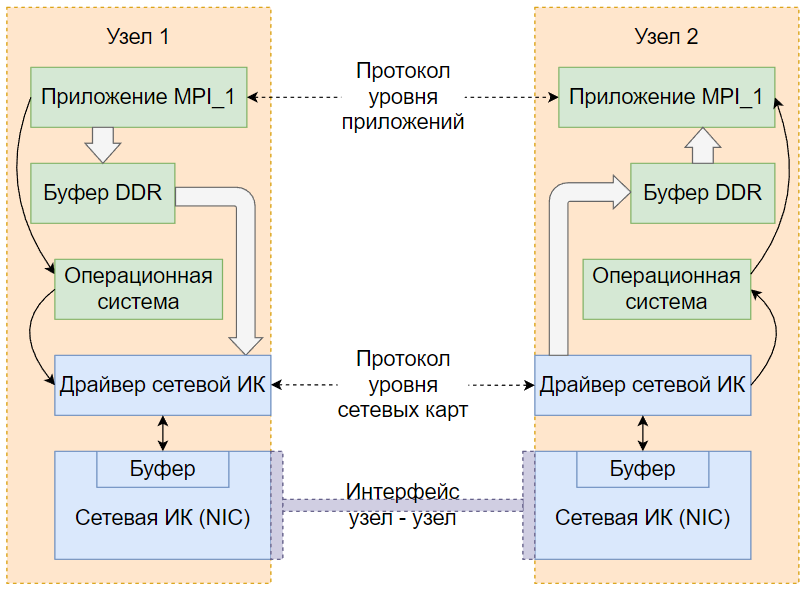


Рисунок 2.9 — Структурная схема обмена данными между двумя узлами

На стороне первого узла приложение MPI\_1 размещает сообщение, которое необходимо отправить, в буфере DDR. По аналогии с предыдущей математической моделью вычисляется время, затрачиваемое на генерацию сообщения, обработку и прохождение через все уровни кэш-памяти и запись в основную память:

|  |  |
| --- | --- |
|  | (1) |

или

|  |  |
| --- | --- |
|  | (2) |

Далее приложение MPI\_2 обращается к ОС с запросом на выполнение операции межкомпьютерного обмена данными. ОС запускает соответствующий драйвер сетевой карты, который загружает байт из буфера DDR в буфер интерфейсной карты NIC, после чего инициирует ее работу. Сетевая интерфейсная карта последовательно передает биты в линию связи, дополняя каждый новый байт стартовым и стоповым битами (в некоторых схемах аналогичную работу проделывает специальный сетевой процессор NPU). На стороне второго узла сетевая интерфейсная карта принимает биты, поступающие со стороны внешнего интерфейса, и помещает их в собственный буфер. После того как получен стоповый бит, интерфейсная карта устанавливает признак завершения приема байта. Факт корректного приема байта фиксируется драйвером сетевой интерфейсной карты второго узла. Драйвер переписывает принятый байт из буфера интерфейсной карты в заранее зарезервированный буфер DDR второго узла. Резюмируя сказанное, необходимо учесть время передачи данных посредством сетевой интерфейсной карты NIC:

|  |  |
| --- | --- |
|  | (3) |

где – объем передаваемого сообщения,

– пропускная способность сетевой карты.

Далее приложение MPI\_2 извлекает данные из буфера посредством операции чтения и интерпретирует их в соответствии со своим протоколом либо как сообщение, либо как данные. Будем считать, что время, затрачиваемое на операцию чтения рассчитывается также как в предыдущем пункте, т.е. с учетом того, что из DDR считывается только та часть от общего объема сообщения , которая не помещается в L3.

Значение суммарной латентности необходимо пересчитать, поскольку к программной составляющей латентности (время, затрачиваемое на системные вызовы и подготовку/завершение MPI вызовов) и аппаратной составляющей (время суммарной задержки от CPU до DDR) добавляется сетевая задержка . В сетевой задержке следует учесть пропускные способности Ethernet/USB разъемов и кабелей, используемых концентраторов, коммутаторов и переходников. Суммарная латентность рассчитывается следующим образом:

|  |  |
| --- | --- |
|  | (4) |

В итоге расчет времени, затрачиваемого при обмене данными между двумя узлами в одну сторону, выглядит следующим образом:

|  |  |
| --- | --- |
|  | (5) |

где - суммарная задержка или латентность,

- доля от объема передаваемого сообщения , доступная только из оперативной памяти при операции чтения,

- коэффициент пропускной способности.

Формализация процесса передачи сообщений между однородными и неоднородными узлами кластера

Модели, разработанные в пунктах 2.2 и 2.3, используются для анализа коммуникационных характеристик коммуникационной инфраструктуры гетерогенных кластеров. На основе этих моделей строится целевая функция , которая отражает время, затрачиваемое при отправке и получении сообщения объемом с одного процесса на другой, учитывая то, что процессы могут находится как на одном, так и на разных узлах исследуемого кластера, но обязательно однородных:

|  |  |
| --- | --- |
|  | (1) |

Параметры целевой функции :

* признак количества узлов ;
  + , если процессы находятся на одном узле кластера;
  + , если процессы находятся на разных узлах кластера;
* суммарная задержка или латентность ;
  + программная составляющая латентности ;
  + аппаратная составляющая латентности где – размер строки кэша, – тактовая частота процессора, – количество тактов до DDR;
  + сетевая задержка .
* коэффициент пропускной способности где принимает значения из множества . Множество содержит пропускные способности соответствующих уровней от CPU до L3;
  + каждая пропускная способность в общем случае рассчитывается следующим образом: , где – тактовая частота, – разрядность шины, – количество каналов передачи.
* пропускная способность оперативной памяти ;
* пропускная способность сетевой интерфейсной карты ;
* доля от объема передаваемого сообщения , доступная только из оперативной памяти при операции чтения , где - размер строки кэша L3, - среднее значение количества кэш-промахов при поиске строки в L3. Тогда объем эксклюзивных строк DDR рассчитывается как .

Для расчета времени, затрачиваемого при передаче сообщения объемом между неоднородными узлами исследуемого кластера, целевую функцию требуется модифицировать с учетом того, что часть вышеперечисленных параметров следует рассчитывать отдельно для каждого узла:

|  |  |
| --- | --- |
|  | (2) |

где

Цель диссертации заключается в повышении быстродействия процесса передачи сообщений между узлами гетерогенной кластерной системы, следовательно, для достижения положительного результата исследования целевая функция как для случая однородных узлов , так и неоднородных должна быть минимизирована:

|  |  |
| --- | --- |
|  | (3) |

Параметры, отражающие характеристики аппаратной составляющей, как правило, не подлежат изменению. Следовательно, минимизировать следует программную составляющую латентности и сетевую задержку .

Разработка методики расчета и анализа основных характеристик коммуникационной инфраструктуры кластера

Архитектура кластерной системы, как способ соединения процессоров друг с другом, в большей степени определяет ее производительность, чем тип используемых в ней процессоров, т.к. самым «узким местом» любого кластера всегда будет среда передачи данных между узлами (36). С помощью целевой функции можно оценить время передачи данных посредством MPI между двумя как однородными, так и неоднородными узлами кластера, например, между терминалом, через который осуществляется доступ к кластеру, и некоторым вычислительным узлом или просто между различными вычислительными узлами. Количество сочетаний узлов для оценки зависит от конкретной топологии сети, поэтому если их несколько, то целевую функцию следует усреднить , где – количество сочетаний узлов сети.

Помимо пропускной способности кабелей, коммутаторов и других составляющих, используемых для организации системной сети, важен сам вид сети, т. е. её топология. Существует целый ряд метрик, характеризующих сетевые соединения, но для того, чтобы оценить разницу между исследуемыми топологиями, можно рассчитать только основные характеристики сетевых топологий: диаметр (минимальное или максимальное расстояние между двумя узлами сети), связность сети, количество связей, приходящихся на один узел, и соответствующую среднюю пропускную способность узла.

Как правило, различают два типа топологий: статические и динамические. К статическим топологиям относят такие конфигурации сети, что остаются полностью неизменными в процессе эксплуатации (18). На физическом уровне между двумя узлами подобных топологий возможен только один фиксированный путь. Основным критерием для классификации статических сетей является размерность:

* 1-мерные (линейная);
* 2-мерные (кольцо, звезда, дерево, сеть, решетка, систолический массив);
* 3-мерные (полносвязный граф, хордальное кольцо, тор);
* n-мерные (гиперкуб).

Довольно продолжительное время основными сетевыми топологиями считались линейная, звездообразная и кольцевая. Считается, что наиболее эффективной конфигурацией из этой категории является топология «толстого дерева». Архитектура гиперкуба является второй по эффективности, но самой наглядной. На сегодняшний день чистые топологии встречаются все реже, как правило, используют различные гибридные комбинации. Выбор конкретной топологии для организации коммуникационных соединений между узлами кластера зависит от требуемого количества узлов.

В таблице 2.2 представлены формулы расчета коммуникационного расстояния или диаметра сети (в условных единицах) для рассмотренных в 1 главе статических сетевых топологий, где *N* — количество вычислительных узлов (как правило*, 2* и больше) (37). Можно заметить, что организация узлов конкретного кластера зависит от требуемого N: при малом N разница между рассмотренными топологиями не так существенна, поэтому выбор осуществляется ситуативно; при увеличении N следует рассчитать основные характеристики, а также учесть особенности построения конкретных топологий на физическом уровне, например, что количество приемников и передатчиков на один узел может быть ограничено или что большие концентраторы довольно дороги.

Таблица 2.2 — Основные характеристики сетевых топологий

|  |  |  |  |
| --- | --- | --- | --- |
|  | Минимальный диаметр | Максимальный диаметр | Средний диаметр |
| Линейка или Однонаправленное кольцо | *1* | *N - 1* | *min [N/2]* |
| Двунаправленное кольцо | *1* | *min [N/2]* | *min [N/4]* |
| Звезда | *2* | *2* | *2* |
| Полное двоичное дерево | *1* | *2\*(h - 1),*  *где h = max [log2N]* | *max [log2N] - 1* |
| Сегментированная сеть  , где  – количество сегментов, – узлов в сегменте | *min [M/2]*  *(в одном сегменте)* | *M*  *(в разных сегментах)* | *M - M/2S*  *(если S = 1,*  *то min [M/2])* |
| Решетка | *1* | *2\*( - 1)* | *- 1* |
| Полносвязный граф | *1* | *1* | *1* |
| Двумерная решетка (тор) | *2 (в двух кольцах)* | *2\*min [/2]* | *min [/2]* |
| Гиперкуб *2^p,* где *p =* *log2N* | *1* | *p* | *min [p/2]* |

Вторым типом топологий являются динамические конфигурации, которые изменяется в процессе вычислений при помощи программных средств. В отличие от статических, в узлах динамических топологий находятся коммутирующие элементы, а вычислительные устройства или терминалы подключаются к входам и выходам этой сети. Устройства могут подключаться и отключаться от сети в любое время, и данные могут передаваться по разным путям в зависимости от текущей нагрузки на сеть. В качестве элементов коммутирующих трактов, как правило, используются либо шины (одношинная или многошинная топологии), либо коммутаторы (матричные или многоярусные сети). Также как и со статическими топологиями, динамические редко применяю без различных модификаций. Например, размер одношинной топологии изменяется в диапазоне от 2 до 50 процессоров, поэтому её, как правило, используют не в чистом виде, а для объединения нескольких узлов в группы, которые в дальнейшем используются в качестве узлов в сети другой конфигурации.

Если построенная сеть характеризуется входами и выходами, то суммарное количество парных связей вычисляется как, а всего возможно связей. Таким образом, возможных конфигураций динамических сетей существует бесчисленное множество, что делает невозможным составление конкретных формул для каждой отдельной конфигурации. Поэтому среднее число шагов или диаметр сети должен рассчитываться по общей формуле следующим образом (39):

|  |  |
| --- | --- |
|  | (1) |

где – расстояние между узлами кластера,

– общее число узлов в сети,

– число узлов на расстоянии .

Еще одним важным показателем, влияющим на производительность и надежность сети, является связность , характеризующая наличие разных путей между узлами сети. Чем больше путей между узлами, тем более связная сеть. Недостаточная связность может привести к проблемам с передачей данных или даже к поломке всей сети. Поэтому при проектировании сети необходимо учитывать связность и выбирать топологию, которая обеспечит наилучшую связность для конкретных потребностей.

Конкретный вид данного показателя может быть определен, например, как минимальное количество дуг, которое надо удалить для разделения сети передачи данных на две несвязные области (39). Если представить исследуемую топологию в виде неориентированного графа , где – множество узлов кластера, – множество межузловых соединений, а (подобное определение позволяет учесть кратные рёбра), то связность сети определяется как реберная связность графа . Существует несколько алгоритмов для нахождения рёберной связности графа, но в основе их всех лежит следующая последовательность действий (рис. 2.11):

1. Каждому ребру сопоставляется единичная пропускная способность.
2. Перебираются все пары вершин и .
3. Находится количество непересекающихся путей из в (например, с помощью алгоритма Диница-Маллика для нахождения максимального потока).
4. Минимальное количество ребер, необходимых для удаления, равно минимальному из найденных значений на предыдущем шаге.

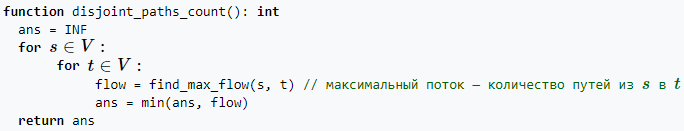


Рисунок 2.11 — Псевдокод алгоритма

Благодаря использованию слоистой сети, алгоритм Диница-Маллика часто работает значительно быстрее, чем другие алгоритмы нахождения максимального потока, такие как Форда-Фалкерсона и Эдмондса-Карпа. Поэтому этот алгоритм считается одним из наиболее эффективных – сложность его работы равнаили . Таким образом общая сложность алгоритма при использовании алгоритма Диница-Маллика будет равна .

Но одного показателя связности недостаточно, чтобы охарактеризовать сеть, поскольку, зачастую, если некоторым узлам в сети требуется пиковый трафик выше среднего, им выделяются дополнительные связи, да и в принципе количество связей, приходящихся на один узел сети, по ряду причин может быть распределено неравномерно между узлами. Следовательно, необходимо определить не только минимальное число дуг , но и среднее количество связей , приходящихся на один узел, а с учетом среднего числа шагов или диаметра также можно вычислить соответствующую среднюю пропускную способность по всем узлам: .

В заключение, рассчитанных показателей достаточно для того, чтобы предварительно охарактеризовать быстродействие исследуемой коммуникационной сети передачи данных. Если необходимо иметь фиксированную пропускную способность в среднем, то следует максимизировать параметр средней пропускной способности сети и общую связность сети . Если некоторым узлам периодически требуется пиковый трафик выше среднего, то им либо выделяются дополнительные связи, либо сокращается время передачи сообщений на самих узлах, следовательно, максимизировать следует . Общая формула выглядит следующим образом:

|  |  |
| --- | --- |
|  | (2) |

Таким образом, можно ввести коэффициент Коммуникационная инфраструктура считается улучшенной, если , где - коэффициент до модификации сети кластера, - после.

Таблица 2.3 — Методика расчета и анализа основных характеристик коммуникационной инфраструктуры кластера

|  |
| --- |
| **Этап 1.** Расчет времени передачи сообщений между узлами кластера  , (1)  где - время передачи сообщения объемом посредством MPI между двумя узлами кластера,  – количество различных сочетаний узлов сети. |
| **Этап 2.** Определение диаметра статической топологии сети   |  |  |  |  | | --- | --- | --- | --- | |  | Минимальный диаметр | Максимальный диаметр | Средний диаметр | | Линейка или Однонаправленное кольцо | *1* | *N - 1* | *min [N/2]* | | Двунаправленное кольцо | *1* | *min [N/2]* | *min [N/4]* | | Звезда | *2* | *2* | *2* | | Полное двоичное дерево | *1* | *2\*(h - 1),*  *где h = max [log2N]* | *max [log2N] - 1* | | Сегментированная сеть  , где  – количество сегментов, – узлов в сегменте | *min [M/2]*  *(в одном сегменте)* | *M*  *(в разных сегментах)* | *M - M/2S*  *(если S = 1,*  *то min [M/2])* | | Решетка | *1* | *2\*( - 1)* | *- 1* | | Полносвязный граф | *1* | *1* | *1* | | Двумерная решетка (тор) | *2 (в двух кольцах)* | *2\*min [/2]* | *min [/2]* | | Гиперкуб *2^p,* где *p =* *log2N* | *1* | *p* | *min [p/2]* | |
| **Этап 3.** Расчет диаметра динамической топологии сети  (2)  где – расстояние между узлами кластера,  – общее число узлов в сети,  – число узлов на расстоянии . |
| **Этап 4.** Нахождение реберной связности  Представление кластера в виде графа  , где – множество узлов,  – множество межузловых соединений,  (3)  Псевдокод алгоритма:  **function** disjoint\_paths\_count(): int  L = INF  for s∈V:  for t∈V:  flow = **find\_max\_flow**(s, t)  L = **min**(L, flow)  **return** L |
| **Этап 5.** Расчет средней пропускной способности сети  Средняя пропускная способность , (4)  где – среднее количество связей узла,  – средняя задержка между узлами. |
| **Этап 6.** Анализ основных характеристик  Введем коэффициент (5)  Коммуникационная инфраструктура считается улучшенной,  если , где - коэффициент до модификации сети, - после. |

В таблице 2.3 представлена методика, позволяющая перед непосредственным построением кластера рассчитать и оценить основные характеристики коммуникационной инфраструктуры кластера в зависимости от требуемого количества узлов N.

Разработка методики ранжирования стандартных интерфейсов межузловых коммуникаций

При передаче сообщений между разными узлами кластера значительный прирост времени происходит из-за латентности сети. Поскольку сетевая задержка учитывает пропускные способности Ethernet/USB разъемов и кабелей, используемых концентраторов, коммутаторов и переходников, то следует тщательно подходить к выбору типа межузловых коммуникаций (40).

При выборе способа организации узлов, можно воспользоваться стандартными интерфейсами коммуникационной инфраструктуры кластера: Fast Ethernet, Gigabit Ethernet, Myrinet, InfiniBand и т.д. Поскольку интерконнекты, как правило, определяются многими критериями, процесс их ранжирования можно представить в виде многомерной модели принятия решений (41). Под термином «принятие решений» полагаем действие над множеством альтернатив, в результате которого получается выбрать один лучший или несколько предпочтительных альтернатив.

На первом этапе следует сформировать совокупность исследуемых технологий межузловых коммуникаций. Пусть , где – множество альтернатив, из которых производится выбор. Универсальных методов формирования множества альтернатив не существует, т. к. решение о включении того или иного интерфейса зависит от особенностей и ограничений реальной ситуации, а также практического опыта задействованных в исследовании экспертов, поэтому и оставляется на усмотрение ЛПР, т.е. лица, принимающего решение. В рамках ранжирования сетевых технологий следует обратить внимание на интерконнекты, используемые в существующих крупных вычислителях, например, упоминаемые в проекте Топ500 (11), описывающем 500 самых мощных общественно известных вычислительных систем мира (рис. 2.12).

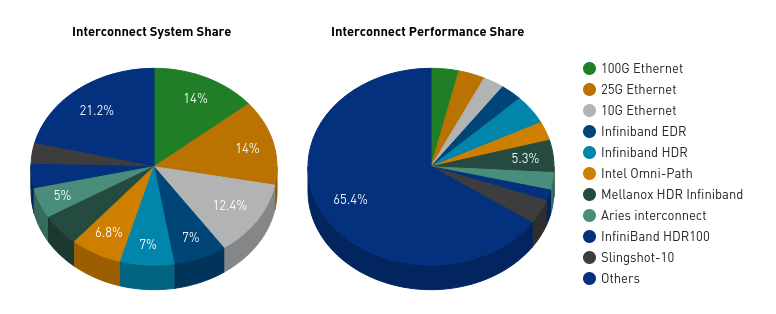


Рисунок 2.12 — Статистика используемых интерконнектов на момент последнего обновления проекта Топ500 (Ноябрь, 2022)

Но подобные сетевые технологии предполагают использование уже реализованных структур сетей передачи данных, поскольку в их стандартах изначально прописан ряд доступных топологий, состав устройств и правила их взаимодействия в сети. Например, с интерфейсами семейства Ethernet предполагается использование линейной или звездообразной топологий. Если же в рамках данной технологии воспроизводить кольцевое соединение между коммутаторами, то подобная сеть работать корректно не будет, потому что в ней могут возникать отрицательные явления, подобные широковещательному шторму. Таким образом, формирование множества альтернативных сетевых технологий, используемых при организации локальной вычислительной сети кластера, должно производиться с учетом заранее выбранной сетевой топологии.

Каждая альтернатива приводит к определенному исходу, последствия которого оцениваются по критериям, т. е. показателям, общим и измеримым относительно всех исследуемых альтернатив, а также характеризующим ценность решения (42). В общем случае , где – множество критериев выбора, используемых для оценки альтернатив из . Перед формированием конкретного множества критериев, следует учесть, что набор критериев должен быть полным (отбрасывание хотя бы одного из избранных показателей приводит к изменению результата решения задачи, а добавление, наоборот, не влияет на результат) и простым (критерии не должны повторятся, т. е. учитывать один и тот же аспект последствий, а также их должно быть минимальное количество).

Для сопоставления быстродействия различных видов сетей, как правило, используется критерий Сталлингса. Этот критерий предполагает вычисление битовой длины (максимальное число бит в сегменте), которая равна произведению максимальной длины сегмента (м) на полосу пропускания (бит/с), поделенному на скорость распространения сигнала в сегменте (м/с): . Тогда коэффициент использования сети рассчитывается как , где ( - размер передаваемых данных в кадре), а эффективная пропускная способность .

В предыдущем подходе не берутся в расчет издержки, связанные с передачей служебной информации и различной частотой следования кадров, которые дополнительно снижают эффективность сети. Таким образом, для более точной оценки следует использовать полный размер кадра с учетом не только полезной нагрузки, но и всей служебной информации (заголовка, преамбулы и т. д. в зависимости от стандарта исследуемой технологии), а также дополнительного межкадрового интервала , который вставляется после каждого кадра: . Т.к. доля служебной информации и интервала всегда одна и та же, то, чем меньше общий размер кадра, тем выше «накладные расходы». Зная частоту следования кадров и размер полезной информации , переносимой каждым кадром, нетрудно рассчитать полезную пропускную способность сети: .

Из данного рассмотрения может показаться, что чем больше пакет, тем лучше. С точки зрения пропускной способности так оно и есть, но с увеличением длины пакета увеличивается время отклика сети. Поэтому помимо теоретически рассчитанных коэффициентов на основе пропускной способности сети, следует учесть величину латентности , замеренную при передаче сообщений посредством MPI. Также стоит учесть, что использование в качестве критериев одновременно как эффективной пропускной способности , так и полезной можно считать избыточным, поскольку оба показателя рассчитываются на основеполосы пропускания сети. Выбор рассчитывать ли оба показателя и, например, усреднять их, или только один из них и какой именно зависит от решения ЛПР. Еще одним из критериев, необходимых для полноты набора, является стоимость сетевого оборудования , которую невозможно рассчитать без заранее известного количества вычислительных узлов и итоговой топологии сети. В состав оборудования, которое необходимо учесть, входят сетевые адаптеры для каждого узла кластера, один или несколько коммутаторов (в зависимости от реализуемой топологии) и, конечно, кабели LAN.

После формирования множества альтернатив и множества критериев необходимо составить матрицу принятия решений (МПР): , где , , а – оценка -той альтернативы по -му критерию. Теперь, когда все начальные значения рассчитаны, можно приступать к непосредственному решению задачи многокритериального выбора. Для этого существует целый ряд методов, однако все они в конечном итоге сводят многокритериальную задачу к одной или ряду задач с одним (иногда обобщенным) критерием (43). Рассмотрим некоторые из них:

1. Лексикографический метод.

При решении задач этим методом элементы из множества критериев ранжируются по степени важности, таким образом, чтобы первым оказался наиболее важный критерий, а последним – наименее. Определим , – вектор оценок -той альтернативы по критериям из множества . Предположим, все критерии максимизируются, тогда сравнение двух альтернатив производится на основании их векторных оценок - векторная оценка лексикографически больше векторной оценки , тогда и только тогда, когда выполняется одно из условий:

|  |  |
| --- | --- |
| 1.  2. ,  …  m. | (1) |

Если на последнем этапе не удается найти единственное решение, то либо вводится дополнительный критерий, либо из множества оставшихся решений делают субъективный выбор.

1. Метод последовательных уступок.

Также как в лексикографическом методе критерии должны быть упорядочены в порядке убывающей важности. Вначале определяется максимальное значение первого по важности критерия . Все последующие определяются уже с учетом того, что значения предыдущих должны отклоняться от соответствующих максимальных не более чем на величину допустимой уступки :

|  |  |
| --- | --- |
| 1.  2.  …  m. | (2) |

1. Метод главного критерия.

Одним из простейших способов решения является метод главного критерия, в котором в качестве в качестве целевой функции выделяется один из критериев (далее ), а на остальные критерии накладываются ограничения, чтобы они были не меньше (в задаче минимизации – не больше) пороговых значений . Таким образом, многокритериальная задача сводится к однокритериальной задаче вида:

|  |  |
| --- | --- |
|  | (3) |

На практике задача решается для нескольких наборов значений , а затем из полученных решений ЛПР определяет наиболее предпочтительный.

1. Линейная свертка.

Если векторные оценки по критериям являются аддитивными, т.е. их как минимум можно привести к общей шкале, то целевых функций можно заменить одной аддитивной функцией полезности:

|  |  |
| --- | --- |
| , | (4) |

где – вес -го критерия, , .

Ввиду того, что критерии зачастую характеризуют исследуемые альтернативы с разных сторон, то, скорее всего, они будут иметь различную размерность. При образовании обобщенного критерия вместо фактических значений следует использовать нормированные: – функция полезности -той альтернативы по критерию , .

1. Максиминная свертка

Здесь, в отличие от метода линейной свертки, на целевую функцию оказывает влияние только тот частный критерий, которому для данной векторной оценки соответствует наименьшее значение:

|  |  |
| --- | --- |
| , | (5) |

где – вес -го критерия, , , а – функция полезности -той альтернативы по критерию , .

Тогда многокритериальная задача сводится к максимизации .

1. Мультипликативная свертка.

Критериальные оценки также должны быть приведены к общей шкале, но в отличие от аддитивной свертки, мультипликативная свертка в основном работает не с абсолютными значениями критериев, а с относительными. Мультипликативная функция полезности с учетом весовых коэффициентов выглядит следующим образом:

|  |  |
| --- | --- |
| , | (6) |

где – вес -го критерия, , .

1. Метод смещенного идеала

Относится к целой группе методов, отличающихся формированием идеального объекта и процедурой отсеивания. Идеальной является та векторная оценка, в которой все мажорируемые критерии достигают своих максимальных значений, а минорируемые - минимальных:

|  |  |
| --- | --- |
| и | (7), (8) |

Далее все оценки нормируются относительно максимальной и минимальной:

|  |  |
| --- | --- |
| , | (9) |

Значения критерия в относительных единицах интерпретируются как расстояния -го объекта по критерию до идеального объекта. Идеальный объект имеет расстояние , а наихудший . Идея метода состоит в том, чтобы найти векторную оценку, наиболее близкую к идеальной . Мерой близости выступает некоторая функция расстояния:

|  |  |
| --- | --- |
| , | (10) |

где – вес -го критерия, , , а – степень концентрации, позволяющая переходить к различным метрикам (может быть любым целым положительным числом и ). И чем больше значение метрики , тем ближе объект находится к идеальному.

Таблица 2.4 — Плюсы и минусы различных методов решения задачи многокритериального выбора

|  |  |  |
| --- | --- | --- |
| **Название метода** | **Плюсы** | **Минусы** |
| Лексикографический метод  ,  , при одном из условий:  1. .  2. , .  …  m. . | * простота и быстрота получения результатов; * низкие требования к математической подготовке экспертов, ПО и вычислительным средствам; * возможность досрочного прекращения. | * необходимость обоснования иерархии критериев; * возможность потери эффекта совокупного влияния второстепенных критериев; * возможность получения слабо эффективных решений. |
| Метод последовательных уступок  1. .  2.  .  …  m.  . | * простота и быстрота получения результатов; * низкие требования к математической подготовке экспертов, ПО и вычислительным средствам. | * необходимость обоснования иерархии критериев; * необходимость обоснования ограничений; * возможность потери эффекта совокупного влияния второстепенных критериев; * возможность получения слабо эффективных и даже неоптимальных решений. |
| Метод главного критерия | * простота и быстрота получения результатов; * низкие требования к математической подготовке экспертов, ПО и вычислительным средствам. | * чрезмерное упрощение задачи; * итоговое решение сильно зависит от главного критерия; * необходимость обоснования ограничений; * возможность потери эффекта совокупного влияния второстепенных критериев; * возможность получения слабо эффективных решений. |
| Линейная свертка  , где  – вес , , ,  – функция полезности альтернативы по критерию ,  . | * компенсация низких оценок высокими; * учет значимости различных критериев; * установление взаимосвязей между критериями различной размерности. | * необходимость приведения к общей шкале и нормирования оценочных значений; * затруднения при определении весов; * не всегда низкая оценка по одному критерию может быть компенсирована высокой оценкой по другому. |
| Максиминная свертка  , где  – вес , , ,  – функция полезности альтернативы по критерию ,  .  . | * определение гарантированной нижней оценки; * учет значимости и различной размерности отдельных критериев. | * необходимость приведения к общей шкале и нормирования оценочных значений; * затруднения при определении весов; * учет только тех критериев, которые дают самые плохие оценки, все остальные критерии игнорируются. |
| Мультипликативная свертка  , где  – вес , , . | * компенсация низких оценок высокими; * учет значимости различных критериев; * отсутствие необходимости нормирования оценочных значений. | * необходимость приведения к общей шкале; * затруднения при определении весов; * не всегда низкая оценка по одному критерию может быть компенсирована высокой оценкой по другому. * тенденция к сглаживанию. |
| Метод смещенного идеала  ,  .  ,  .  , где  – вес -го критерия, , | * полиномиальная сложность; * установление взаимосвязей между критериями различной размерности; * результат получен в ранговой шкале. | * необходимость приведения к общей шкале; * затруднения при определении весов. |

В таблице 2.4 представлена сравнительная характеристика рассмотренных выше многокритериальных методов. В рамках решения задачи ранжирования стандартных интерфейсов коммуникационной инфраструктуры кластера наиболее подходящим методом является метод смещенного идеала ввиду полиномиальной сложности и эффективности получаемого решения. Минусы данного метода нивелируются тем, что критерии из множества уже являются количественными, т.е. не нуждаются в приведение к общей шкале.

Таким образом, методика ранжирования стандартных интерфейсов межузловых коммуникаций кластера, позволяющая выбрать подходящий интерконнект, представлена в таблице 2.5.

Таблица 2.5 — Методика ранжирования стандартных интерфейсов межузловых коммуникаций кластера

|  |
| --- |
| **Этап 1.** Формирование множества альтернатив  , где (1) |
| **Этап 2.** Формирование множества критериев  , где (2)  Пусть , тогда .  – латентность MPI,  или *)*, (3)  *= ,* или *),* (4)  – стоимость сетевого оборудования.  Полезная пропускная способность:  *, ,*  (5)  Эффективная пропускная способность:  *, ,*  *,* (6) |
| **Этап 3.** Составление матрицы принятия решений  , где , (7)  – оценка -той альтернативы по -му критерию,  - вектор оценок -той альтернативы по критериям из множества . |
| **Этап 4.** Применение метода смещенного идеала  1. Формирование идеального и неидеального объекта.  (8) и (9)  2. Нормирование критериальных оценок  , (10)  3. Составление системы важности критериев  *,* где(11)  Условие:, (12)  Пусть , тогда .  4. Расчет меры Минского  , где (13)  5. Упорядочивание альтернатив для различных значений  *,* если(14) |
| **Этап 5.** Ранжирование альтернатив  Исключение не доминирующей альтернативы из и включение в итоговое множество (упорядочено по возрастанию). Возвращение к 4 этапу, пока . |

## Выводы по главе 2

В ходе диссертационного исследования произведены следующие работы:

* проанализирована последовательность операций при передаче сообщений между двумя процессами;
* разработана математическая модель передачи сообщений на одном и нескольких узлах кластера;
* выполнена формализация процесса передачи сообщений между однородными и неоднородными узлами кластера;
* описана методика расчета и анализа основных характеристик коммуникационной инфраструктуры кластера;
* описана методика ранжирования стандартных интерфейсов межузловых коммуникаций.

# ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДИК И АЛГОРИТМА АНАЛИЗА ХАРАКТЕРИСТИК СЕТЕВОЙ ВЫЧИСЛИТЕЛЬНОЙ ИНФРАСТРУКТУРЫ ГЕТЕРОГЕННЫХ КЛАСТЕРОВ



Модель виртуального разделения основного процесса на субпроцессы для запуска MPI-программ без изменений

Резюмируя сказанное в предыдущем пункте, вместо первоначальной схемы адаптации MPI-модели, предлагается реализовать виртуальное разделение 64х-разрядного CPU-процесса на два 32х-разрядных DSP-процесса или субпроцесса, основным механизмом взаимодействия с которыми является общая память. Для всех DSP-процессов создаётся карта памяти (одна на всех), идентичная карте памяти CPU-процесса, что позволяет избежать изменения таблиц MMU – это решает проблему с необходимостью сбрасывать кэш. Идентичность карты памяти между DSP и CPU поддерживается драйвером, который перехватывает mmap и другие функции, потенциально изменяющие карту памяти (CPU-процесс этим заниматься не может, поскольку не имеет доступа к своей карте памяти). В итоге, все остальные механизмы взаимодействия строятся на основе общей памяти, сокетов, очередей сообщений, деревьев событий/прерываний, а системные вызовы работают через очереди сообщений DSP<=>CPU, хотя крупные блоки могут передаваться и напрямую из/в память субпроцесса.

Основным назначением субпроцессов является запуск существующих программ без внесения в них каких-либо изменений (при условии, что имеются все необходимые функции, если же каких функций не хватает, их набор расширяется) (46). Преимущество запуска программы целиком на DSP заключается в том, что при ускорении какой-либо функции будет получен положительный эффект для всех использующих её программ. Это возможно из-за отсутствия дополнительных расходов на вызов DSP функций со стороны CPU, в то время как подобные вызовы в ElcoreCL делают бессмысленным оптимизацию любых мелких функций (менее ~100К тактов на вызов), даже если такие функции вызываются миллионы раз. Если же рассматривать выполнение более долгих функций или даже целых сегментов кода, то проблема возникает, если сценарий использования не согласуется с моделью ElcoreCL (например, если функция вызывается для всех элементов некоторого списка, передать его сразу через ElcoreCL невозможно, т.к. предварительно его нужно проанализировать на предмет умещается ли он в 4Гб, а если нет, то придётся прибегнуть к копированию). Иными словами, субпроцессы позволяют оптимизировать для DSP популярные библиотеки, в то время как вызов оптимизированных функций через ElcoreCL может оказаться бессмысленным.

Предполагается, что собственные 4Гб памяти субпроцесса доступны DSP-ядру напрямую, а со стороны других DSP они могут быть прочитаны/записаны через VDMA (у CPU есть прямой доступ). Кроме "локальной" памяти (код, данные, куча, стек – см. рис. 3.3) в этих 4Гб будет и общая память (кэшируемая/некэшируемая), необходимая для управления и взаимодействия субпроцессов между собой. Например, там могут размещаться списки субпроцессов, списки потоков внутри конкретного субпроцесса, контексты потоков (субпроцессов), объекты синхронизации, очереди сообщений или «крупных» функций. Очередь «крупных» функций нужна для того, чтобы равномерно распределить нагрузку между простаивающими и работающими DSP или CPU, и универсальным способом это сделать является использование VDMA, поскольку через него модель субпроцессов позволяет получить доступ ко всей памяти CPU-процесса (47).

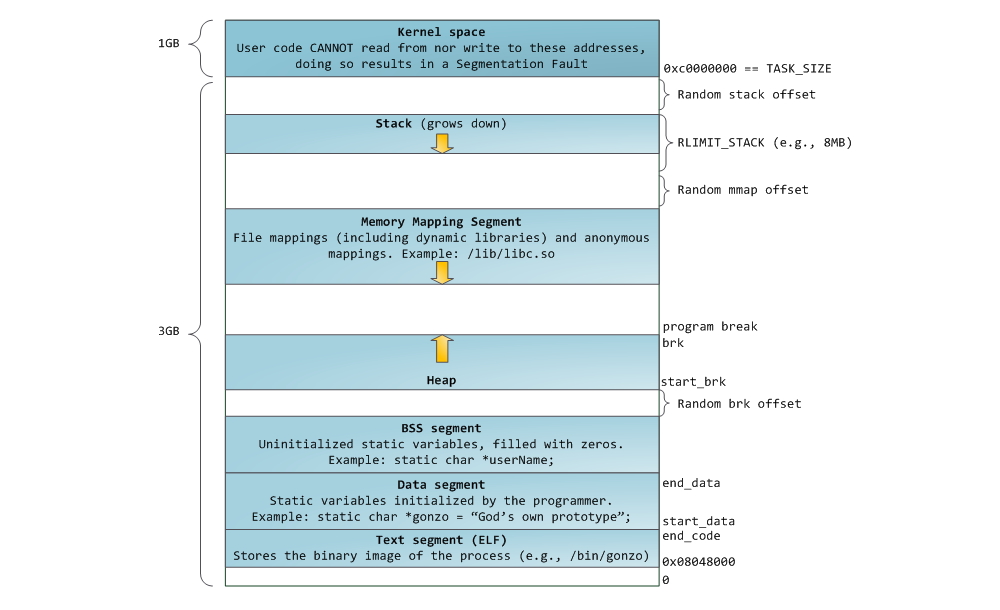


Рисунок 3.3 — Виртуальное адресное пространства процесса (Linux)

В рамках предыдущей модели добавление некоторой функции в очередь через ElcoreCL фактически соответствует запуску самостоятельного процесса. Единственный способ для этого процесса как-то взаимодействовать с другими процессами — это получить в качестве параметра общий буфер. Внутри этого процесса может быть реализована некая многопоточность, но тем не менее все потоки все равно будут выполняться на одном DSP. То есть другие DSP не могут участвовать в обработке локальных данных (как в OpenMP), поскольку не имеют к ним доступа. Чтобы убрать это ограничение и сделать возможной миграцию потоков между DSP, необходимо чтобы они работали в общем адресном пространстве. Реализовать это можно путём запуска на всех DSP некоторого загрузчика, который получит общую память, загрузит туда программу и начнёт её выполнение на ведущем DSP. Остальные DSP будут ждать в загрузчике и перейдут к выполнению указанной функции при создании в программе новых потоков. Но пока не работает когерентность, необходимо явно управлять кэшированием и аккуратно делить задачу, чтобы исключить конфликт записи в одну строку кэша:

* в случае явной многопоточной функций кэш ведущего ядра выгружается, массивы делятся на блоки – четные обрабатываются (кэш выгружается), процессы синхронизируются, затем обрабатываются нечётные блоки (кэш снова выгружается);
* если всю работу вести через DMA, то после выгрузки кэша ведущего ядра, нет необходимости в выгрузке 64 байт разделяемой строки с потенциальной порчей чужих записей;
* но если планируется, что потоки будут выполнять совершенно разные функции, то без когерентности нет возможности записывать глобальные переменные в любой момент (требуется синхронизация и управление кэшами).

Разработка методики ускорения выполнения MPI-приложений на кластере гетерогенного типа

В целях повышении быстродействия процесса передачи сообщений между узлами неоднородными кластера, помимо снижения сетевой задержки , путем применения методики ранжирования стандартных интерфейсов межузловых коммуникаций, также следует попытаться минимизировать аппаратную и программную составляющую латентности ( и соответственно). Для это предлагается предпринять следующие шаги:

**Этап 1.** Полная или частичная поддержка протокола когерентности кэшей.

Зачастую в системах с неоднородными узлами существуют проблемы, связанные с отсутствием когерентности кэшей между процессорами разной или специфической архитектуры. Когерентность должна гарантировать, что все ядра будут работать с актуальными данными при обращении к собственному кэшу, даже если обновление производилось другим ядром. Таким образом предотвращается возможность ошибок при обмене данными между ядрами процессора. Существуют различные протоколы когерентности (рис. 3.4), представляющие собой специальные алгоритмы для управления доступом к кэшу и обеспечения общности данных между ядрами процессора, но реализация этих протоколов может различаться в процессорах разной архитектуры, что и является источником проблемы.

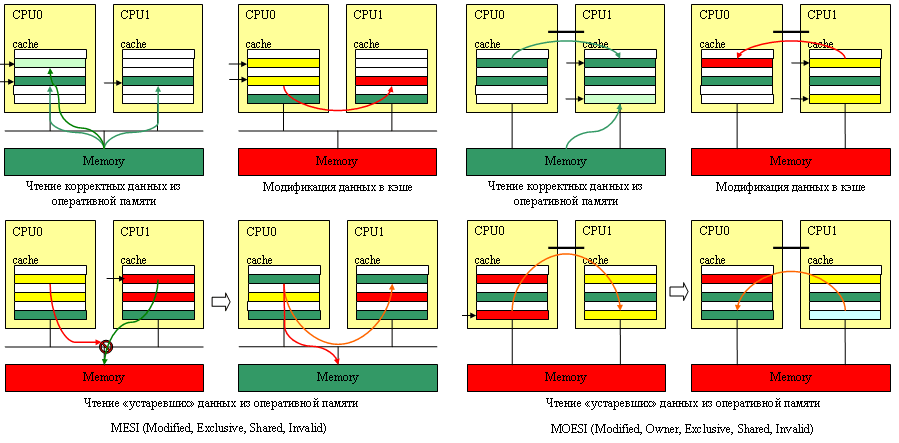


Рисунок 3.4 — Протоколы когерентности кэш-памяти (MESI, MESIF)

В отсутствии когерентности между кэшами при частичном изменении данных в одной из строк кэша есть опасность переписать устаревшими данными изменения от других ядер, поэтому данные могут в принципе не кэшироваться, что замедляет выполнение программы. Применимо для рассматриваемой конфигурации кластера существуют следующие решения данной проблемы:

1. Полная поддержка протокола когерентности, поскольку даже явные flush/invalidate эту проблему не решают.
2. Добавление в кэш строба записи на каждый байт, что может решить проблему с проприетарными DSP ядрами (необходимость частичных flush/invalidate остаётся), но при интеграции стандартных ядер, у которых только один флаг dirty на всю строку кэша, проблемы останутся.
3. Вычисления с кэшированием, но без когерентности.

Третий пункт довольной спорен, поскольку подобные вычисления корректны только при условии, что несколько ядер не пытаются в один момент времени записывать одну и ту же строку кэша. Хотя новый код может быть написан с учётом данного ограничения, существующий многопоточный код в большинстве случаев работать не будет, следовательно, он запускается либо в однопоточном режиме, либо переписывается. Здесь может работать схема один писатель-много читателей (1W+MR), иными словами, память DSP становится доступной для чтения другими DSP. В таком случае модель вычислений выглядит следующим образом: DSP что-то вычисляет, записывает в свою память и делает flush, после чего каким-то образом оповещает другие ядра о наличии новых данных; другие ядра DSP делают invalidate, после чего могут читать обновлённые данные и делать свои вычисления; после завершения вычислений писатель оповещается, что данные больше не нужны, т.е. он сможет освободить или повторно использовать свою память.

**Этап 2.** Виртуальное разделение основного процесса на субпроцессы.

В случае частичной поддержки протокола когерентности большинство проблем из пункта 3.1 все еще остается нерешенным:

* необходимость постоянно сбрасывать кэш CPU (на это затрачивается порядка 100К тактов);
* различная разрядность адресов на DSP и CPU (32 и 64 бита соответственно) приводит к лишнему копирование массивов указателей;
* дополнительные расходы на вызовы DSP функций со стороны CPU в ElcoreCL (делают бессмысленным оптимизацию мелких функций).

Размещение памяти субпроцессов (на DSP) в адресном пространстве основного процесса (на CPU) позволяет нивелировать влияние вышеперечисленных проблем, реализуя взаимодействие процессов через общую память без привлечения дополнительных механизмов. Для получения внешних ресурсов субпроцесс может обращаться к основному процессу, т.е. системные вызовы будут идти уже от имени CPU-процесса. Если субпроцесс приостановлен, то над его памятью основной процесс может спокойно выполнять send/recv/fread/fwrite, причем данные попадают в память субпроцесса без лишнего копирования. От ОС/драйвера требуется только поддержание когерентной основному CPU-процессу карты памяти, переключение потоков, а также приостановка при ошибках с сохранением состояния. Кратное 4Гб размещение субпроцессов позволяет назначить базовый адрес для переключения между субпроцессами внутри основного процесса путем определения значения регистра ASID, что также уменьшает объем сбрасываемых из кэша данных. При всем этом субпроцессы смогут использовать такое же представление указателей, как и у основного процесса, поскольку для адресации берутся только младшие 32 разряда. Если субпроцесс будет корректно заполнять старшие 32 разряда указателей (старшие 32 бита своего базового адреса), то основной процесс тоже сможет их использовать без преобразований. Если же создать общий блок памяти внутри субпроцессов (для него будет несколько резервов на основном процессе), то они смогут взаимодействовать напрямую без обращений к основному процессу.

Таким образом, субпроцессом считается любой DSP-процесс, имеющий совместную память с CPU. Для этого специальный драйвер VELcore строит общую для всех DSP-ядер карту памяти путем вызова специальной функции CPU-процессом. CPU-программа резервирует N\*4Гб памяти, где N - количество DSP-процессов, и 4Гб общей памяти для всех субпроцессов (Рисунок 3.5 — Субпроцессы внутри основного процесса). Карта памяти идентична карте памяти CPU-процесса и динамически изменяется. Объем памяти в 4Гб доступен напрямую из DSP-процесса, остальная память - через специальные механизмы, например, VDMA. Программа-загрузчик, выполняемая на CPU, загружает DSP-программу (Elf-файл) по фиксированным адресам.

**Этап 3.** Использование механизма прямого доступа к памяти (DMA).

Взаимодействие напрямую между субпроцессами или с памятью основного процесса также возможно через использование DMA, при этом дополнительных модификаций карты памяти не требуется. В некоторых случаях использование DMA даже необходимо – из-за отсутствия прямого доступа к памяти свыше 4Гб, с остальными данными можно работать в 64-х разрядном адресном пространстве CPU-процесса через DMA. Чтобы DSP и CPU могли совместно использовать сложные структуры данных, требуется единое понимание адресов и типов, что и обеспечивается общей картой памяти (таблица 3.1), в ином случае требовалась бы доработка компилятора DSP, чтобы адреса и связанные с ними типы занимали не 32, а 64 бита

Из-за проблем с когерентностью сделать поддержку OpenMP на уровне компилятора затруднительно, но в качестве аналога можно использовать DMA-функции, поскольку они не портят чужие данные как кэш. Фактически, DMA является заменой OpenMP, и делают возможным создание оптимизированных библиотек, где поток выполняемый на любом DSP или CPU может привлечь к вычислению «крупной» функции все свободные DSP ядра:

* если субпроцессу требуется, чтобы другие DSP-ядра помогли ему вычислить некоторую функцию, то он размещает задание в общей очереди, доступной другим субпроцессам;
* предварительно этот субпроцесс выгружает свои кэши, чтобы актуальные данные попали в память и через VDMA их можно было читать/записывать;
* поскольку речь идёт о DMA-функциях, для их работы другим DSP-ядрам требуется поток, использующий xyram и имеющий возможность обрабатывать задания из очереди;
* при этом сбрасывать собственные кэши L0/L1 другим DSP-ядрам не нужно, поскольку они работают с данными в другом субпроцессе;
* по завершении вычислений инициатор продолжает свою программу, а помощники могут переключиться на следующее задание или другой поток.

Предварительная схема соответствия адресов приведена в таблице 3.1. Столбец dspCoreN содержит карту памяти со стороны N-ого DSP-ядра, cpuCores – общая карта для основных CPU-ядер, а dmaCores представляет итоговую общую карту для всех DSP-ядер.

Таблица 3.1 — Карта памяти для DMA-функций

|  |  |  |  |
| --- | --- | --- | --- |
| **Address** | **dspCoresN** | **cpuCores** | **dmaCores** |
| 0-4Gb | dspShared | cpuLocal | dmaShared |
| dspLocal**N** |
| 4-8Gb | none | dspShared | dmaShared |
|  | none | none |
| 8-12Gb | none | none | dspShared |
| dmaLocal0 | dspLocal0 | dspLocal0 |
| 12-16Gb | none | none | dspShared |
| dmaLocal1 | dspLocal1 | dspLocal1 |
| … |  |  |  |
| 68-72Gb | none | none | dspShared |
| dmaLocal15 |  | dspLocal15 |
|  | dmaShared | dmaShared | dmaShared |
|  | none | cpuLocal | dmaShared |

**dspCoreN**:

* в нижних 4Гб размещается dspShared (общая для всех DSP) и dspLocal (собственная для каждого DSP);
* выше 4Гб отображается dspLocal всех DSP (для dma-функций) и dmaShared, которая может быть больше 4Гб;
* память cpuLocal со стороны DSP недоступна.

**cpuCores**:

* cpuLocal со стороны DSP доступна только при доработке драйвера, который будет поддерживать когерентность между DSP и CPU;
* dspShared в адресное пространство CPU отображается только один раз, чтобы у кэша не было проблем с alias;
* в адресное пространство CPU отображаются все dspLocal, чтобы была возможность прямого обмена с памятью субпроцессов;
* без DMA-функций DSP может помочь CPU (даже со сложными структурами), только если основной процесс размещает данные в dspLocal;
* dmaShared используется в случаях, когда DMA-функции помогают обрабатывать блоки данных свыше 4Гб;

**dmaCores**:

* драйвер поддерживает когерентность памяти между DSP и CPU при любых изменениях;
* всю память CPU-процесса можно трактовать как dmaShared, поскольку за счёт одинаковых карт памяти через DMA можно добраться везде;
* локальная память DSP теперь начинается не c 0 адреса, поэтому работа с ней осуществляется за счёт загрузки правильного ASID
* по этим же причинам dspShared отображается несколько раз, чтобы каждый DSP мог с ней работать напрямую;
* все dspLocal теперь присутствуют только в одном экземпляре, и ниже 4Гб отображать их не требуется;
* такая карта памяти позволяет переключать DSP ядро между процессами путём перезагрузки ASID;
* во избежание проблем с когерентностью один процесс должен выполняться не более чем одним ядром;
* несколько ядер могут работать в одном процессе, только если на уровне кода исключить запись в одну строку кэша.

**Этап 4.** Улучшение характеристик аппаратных средств кластера.

Поскольку сетевая задержка учитывает пропускные способности Ethernet/USB разъемов и кабелей, используемых концентраторов, коммутаторов и переходников, то следует выбрать тот тип интерконнекта, который был признан наилучшим при применении методики ранжирования стандартных интерфейсов межузловых коммуникаций

В целях минимизации аппаратной составляющей латентности может потребоваться некоторая модернизация аппаратных средств кластерной системы. Поскольку где – размер строки кэша, – тактовая частота процессора, – количество тактов до DDR, то одним из самых простых и очевидных способов является повышение тактовой частоты процессора как на одном узле кластера, так и на многих.

На рисунке 3.6 видно, что скорость чтения из L1-кэша растет линейно с ростом частоты CPU (ref\_clk). Это означает, что скорость чтения ограничена производительностью CPU и ее можно повысить либо путем повышения рабочей частоты процессора, либо путем оптимизации обращений к памяти (например используя SIMD-инструкции) (48).

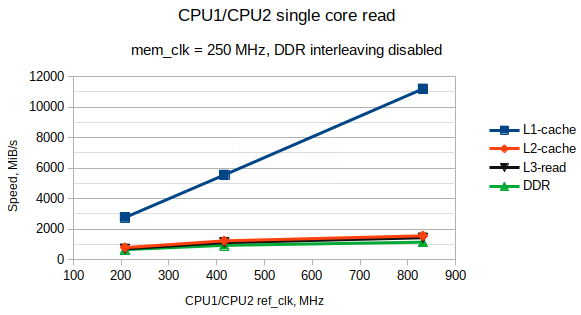


Рисунок 3.6 — Зависимость скорости чтения с различных уровней памяти от частоты

**Этап 5.** Сборка MPI с OpenSHMEM.

Модель программирования SHMEM (от англ. Shared memory) состоит из библиотечных подпрограмм, которые обеспечивают связь с малой задержкой и высокой пропускной способностью для использования в масштабируемых программах с высокой степенью параллелизма (49). Подпрограммы API OpenSHMEM предоставляют модель программирования для обмена данными между взаимодействующими параллельными процессами. Программа OpenSHMEM выполнена в стиле SPMD (Single Program Multiple Data), т. е. все процессы выполняют одну и ту же программу, но с различными данными. В этой модели каждый процесс выполняет свою часть программы независимо от других процессов, но периодически связывается с другими для обмена информацией, от которой зависят последующие вычисления.

Благодаря применению PGAS-модели, в которой процессы имеют доступ не только к собственной локальной памяти, но также и к разделенной между всеми процессами глобальной области, каждый процесс может читать и записывать данные в эту область, что позволяет обмениваться информацией между процессами без использования сетевых протоколов. Т.е. данные хранятся в локальной памяти каждого процесса, а глобальная память используется для обмена данными между процессами, причем операции записи в глобальную память автоматически синхронизируются между всеми процессами. SHMEM также использует модель «односторонней связи», что означает, что каждый процесс может напрямую обращаться к памяти других процессов без необходимости синхронизации. Это позволяет достичь высокой производительности при выполнении операций чтения и записи.

Подпрограммы OpenSHMEM минимизируют накладные расходы, связанные с запросами на передачу данных, максимально увеличивают пропускную способность и минимизируют задержку данных. Для работы с SHMEM используется набор функций, позволяющих создавать, инициализировать и управлять общей памятью: операции put поддерживают удаленную передачу данных другому процессу, операции get получают эти данные, а удаленные указатели разрешают прямые ссылки на объекты данных, принадлежащие другим процессам. Другими поддерживаемыми операциями являются широковещательные broadcast и reduction, барьерная синхронизация и операции с атомарной памятью, которые позволяют выполнить несколько операций над общей памятью как единое целое.

Программы, написанные с использованием функций SHMEM, похожи по стилю на программы интерфейса передачи сообщений (MPI), поскольку оба стандарта предназначены для организации параллельных вычислений на распределенных системах. Таким образом, API OpenSHMEM можно использовать отдельно или в сочетании с MPI в одной и той же параллельной программе (рис. 3.7). Но на MPI сложно реализовывать управляемые данными приложения, т. е. приложения с нерегулярными схемами обмена данными. Модель SHMEM как раз-таки предоставляет альтернативный подход, но из-за того, что OpenSHMEM является новым стандартом, маловероятно, что с ним будут переписаны целые приложения. Благодаря появлению специальных Unified Communication Runtimes платформ, например, MVAPICH2-X, возможно построение нового класса гибридных приложений, использующих преимущества как MPI, так и SHMEM моделей.

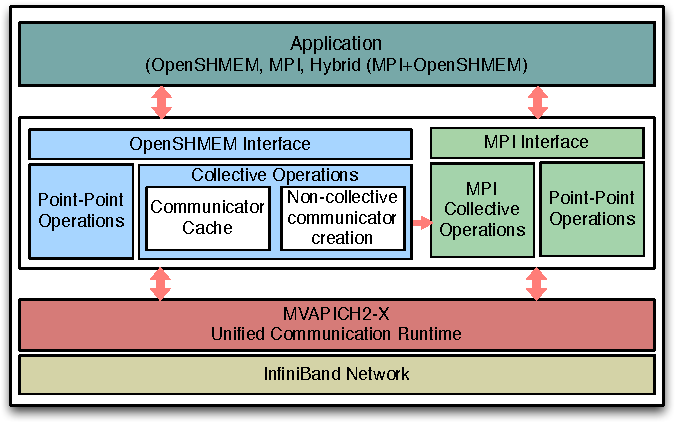


Рисунок 3.7 — Диаграмма компонентов гибридного приложения с использованием MPI и OpenSHMEM

Таким образом, при сборке реализации MPI (например, OpenMPI) следует в конфигурационном файле включить сборку реализации OpenSHMEM (--enable-oshmem), если по умолчанию она отключена. Также следует контролировать параметры, отвечающие за создание либо статических, либо разделяемых библиотек. По умолчанию OpenMPI и OpenSHMEM создают разделяемые библиотеки, и все компоненты включаются как часть этих библиотек. Поскольку включение статических библиотек (--enable-static) и отключение разделяемых библиотек (--disable-shared) — это два независимых варианта, для создания статических библиотек и статической компоновки следует использовать их вместе.

**Этап 6.** Оптимизация MPI-программ на пользовательском уровне.

Для улучшения характеристик латентности и пропускной способности при пересылке сообщений посредством MPI можно использовать следующие подходы (5):

1. Организация асинхронного взаимодействия с помощью неблокирующих примитивов send и receive. Конечно, по сравнению с асинхронным взаимодействием синхронное легче реализовать (рис. 3.8). Оно также более надежно, так как гарантирует процессу-отправителю, возобновившему свое выполнение, что его сообщение было получено. Главный же недостаток синхронного выполнения — ограниченный параллелизм и возможность возникновения ситуаций, когда процесс-отправитель блокируется навсегда. Если все же используются блокирующие транзакции, требуется обеспечить исключение тупика посредством тщательной разработки используемой коммуникационной стратегии.

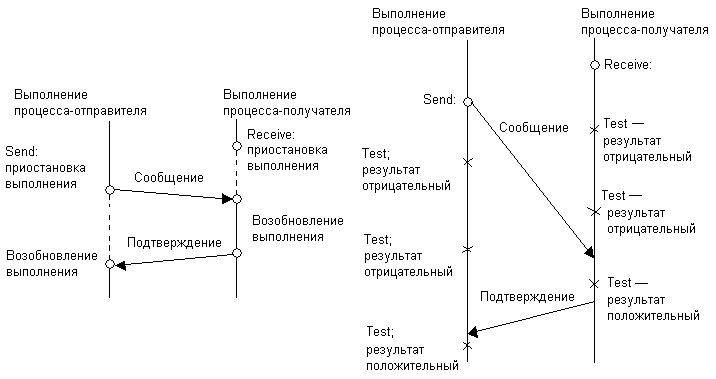


Рисунок 3.8 — Синхронное и асинхронное взаимодействие

2. Использование неблокирующих операций ввода/вывода (non-blocking I/O) позволяет избежать блокировки процессов на этих операциях и продолжить выполнение программы во время ожидания завершения операции (аналогично предыдущему пункту).

3. Использование коллективных операций для сбора (MPI\_Gather) и рассылки данных (MPI\_Scatter) является более эффективным способом взаимодействия, поскольку устраняет вызовы MPI\_Send и MPI\_Recv каждый раз при коммуникации процессов.

4. Определение номера принимающего процесса при вызове подпрограммы MPI позволяет избежать использования константы MPI\_ANY\_SOURCE, которая, как правило, увеличивает латентность.

5. Оптимизированное использования памяти. Необходимо учитывать, что каждый процесс имеет свою локальную память, поэтому есть смысл минимизировать количество обменов данными между процессами и использовать локальные копии данных. А для больших сообщений можно использовать архивацию данных, а также по возможности использовать стандартные типы данных вместо MPI\_PACK и MPI\_UNPACK.

6. Использование алгоритмов распределения нагрузки в большинстве случаев позволяет равномерно распределить задачи между процессами (предварительно следует подсчитать количество доступных каналов передачи данных) и увеличить эффективность параллельной обработки. Может получиться, что использование коммуникаторов с меньшим количеством процессов уменьшает количество обменов данными между процессами, что существенно ускоряет выполнение программы.

7. Реализация алгоритмов и структур данных, подходящих для распараллеливания. Необходимо учитывать, что некоторые алгоритмы и структуры данных могут быть неэффективными в параллельном исполнении, поэтому необходимо выбирать наиболее подходящие для конкретной задачи.

Структура системы сборки дистрибутива ОС Linux Buildroot

Для наиболее эффективного использования ресурсов 1892ВМ248 «Robodeus» необходима операционная система. Одной из самых распространенных ОС для встраиваемых систем является UNIX-подобная GNU/Linux (или просто Linux). Центральной частью ОС GNU/Linux является ядро Linux, обеспечивающее приложениям координированный доступ к ресурсам системы. Основные особенности ядра Linux:

* Переносимость. Linux запущен на большинстве существующих архитектур.
* Безопасность. Код ядра Linux редактируется многими экспертами.
* Стабильность и надежность.

ОС GNU/Linux распространяется в виде дистрибутивов. Для встраиваемых систем существует большой выбор утилит систем сборки: бинарные Debian/Ubuntu/Fedora, системы сборки Yocto/OpenEmbedded, Buildroot и т.д. Преимущества использования Buildroot заключаются в возможности сборки системы кросс-компиляции самим Buildroot’ом, простоте конфигурации (make menuconfig), простоте устройства (представляет из себя в основном набор Makefile’ов), высокой скорости сборки (несколько минут для сборки rootfs минимальной конфигурации), небольшом размере выходного образа (2 МиБ в минимальной конфигурации) и большом количеством уже существующих рецептов сборки приложений.

Buildroot это система автоматической сборки полноценного GNU/Linux окружения для различных платформ (51). Она позволяет создать минимальный и оптимизированный образ системы для оборудования с ограниченными ресурсами. Buildroot использует пакетный менеджер и конфигурационные файлы для определения того, какие пакеты и компоненты должны быть собраны в целевом дистрибутиве. Таким образом, Buidlroot позволяет одной командой make собрать целый ряд образов (рис. 3.9).

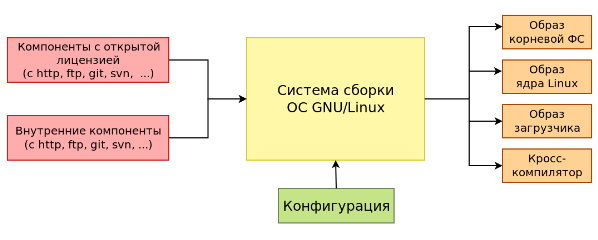


Рисунок 3.9 — Основные принципы системы сборки Buildroot

Состав архива дистрибутива для запуска на отладочном комплекте Semantic Server Processor Bring-up Board (SSP BuB) с СнК «Robodeus» следующий:

1. buildroot: исходные коды системы сборки Buildroot (некоторые рецепты пакетов Buildroot изменены).
   1. arch/ - файлы конфигурации для различных архитектур;
   2. package/- рецепты сборки пакетов;
   3. fs/ - файлы для генерации образов файловых систем;
   4. output/ - финальные артефакты сборки;
   5. toolchain/ - файлы для генерации или использования кросс-компилятора.
2. buildroot/dl: директория, содержащая архивы исходных кодов всех пакетов данной конфигурации. В директории содержатся исходные коды Boot-приложений (/ddrinit и /solaris-boottools), U-Boot (/uboot-tools), Linux (/linux).
3. external-common: внешнее дерево пакетов прикладного ПО для вычислительного кластера без использования ELcore-ядер;
4. external-solaris: внешнее дерево пакетов специального ПО для поддержки «RoboDeus».
5. external-elcore50: внешнее дерево пакетов прикладного ПО для вычислительного кластера с использованием ядер Elcore-50;
6. Makefile: скрипт сборки Buildroot. Скрипт устанавливает переменную BR2\_EXTERNAL с указанием пути до директорий external-\* и вызывает make в директории buildroot. Таким образом, при вызове make в корневой директории дистрибутива доступны все стандартные цели Buildroot (например, make help – вывод справки по целям Buildroot).

Boot-приложения для 1892ВМ248 «Robodeus» выполняют начальную инициализацию процессора MIPS I6400 Samurai (CPU0). Boot-приложения состоят из начального загрузчика IBL, который выполняет начальную инициализацию процессора CPU0; инициализатора контроллеров DDR; Daimyoboot или инициализатора процессоров MIPS I6500 Daimyo (CPU1/CPU2).

Загрузчик U-Boot — это системное программное обеспечение для выполнения базовой инициализации аппаратных устройств (в том числе DDR-памяти), загрузки и исполнения бинарных файлов приложений (в том числе ядра Linux), а также декомпрессии бинарных файлов приложений. Кроме того, загрузчик U-Boot предоставляет оболочку для запуска различных команд для выполнения загрузки данных с внешнего носителя или сети, тестирования работоспособности различных аппаратных устройств, DDR-памяти и т.д.

Ядро Linux выполняет задачи по управлению всеми аппаратными ресурсами (CPU, память, ввод/вывод) и обработке конкурентных запросов на использование аппаратных ресурсов от различных пользовательских приложений. Дистрибутив ОС GNU/Linux на базе Buildroot для 1892ВМ248 поддерживает варианты расположения ядра Linux на SD/eMMC-карте и TFTP-сервере.

Файловые системы используются для организации данных в директории и файлы. В UNIX-подобных системах пользователям и приложениям доступна единая глобальная иерархия файлов и директорий, которая в общем случае может состоять из нескольких файловых систем. Корневая файловая система (rootfs) — это система, которая примонтирована в корень (root) глобальной иерархии, обозначаемый /. Корневая файловая система монтируется при загрузке ОС ядром Linux (для этого используется специальный параметр ядра root=). Для 1892ВМ248 корневая файловая система может располагаться на SD/eMMC-карте и NFS-сервере.

Для перепрограммирования ПО «Robodeus» необходимо на ПК собрать новый образ дистрибутива, записать его на SD-карту и заменить SD-карту на печатном узле вычислителя. Порядок сборки дистрибутива следующий:

1. Разархивировать архив дистрибутива (package-name - имя архива без расширения tar.gz):

tar zxvf <package-name>.tar.gz

1. Сменить текущую рабочую директорию в распакованную директорию:

cd <package-name>

1. Если сборка выполняется в ОС рабочего ПК, то выполнить команду для сборки Buildroot и образа SD-карты (длительность сборки составляет около 120 минут и зависит от производительности CPU рабочего ПК):

make

Сборка в контейнере Docker предназначена для организации воспроизводимой сборки (например, непрерывной интеграции) и не рекомендуется для разработки и отладки скриптов сборки. В случае если сборка Buildroot была выполнена в контейнере, то повторные запуски сборки (после реконфигурации, изменения исходных кодов пакетов) также должны выполняться в контейнере. Повторный запуск сборки Buildroot будет завершаться ошибкой. Подобная ситуация сложилась из-за того, что рабочая директория Buildroot на файловой системе хост-машины и на файловой системе контейнера имеет различные пути, а при сборке скрипты Buildroot устанавливают абсолютные пути.

Для сборки в контейнере Docker необходимо:

1. Установить и настроить сервис Docker на хост-машину.
2. Добавить текущего пользователя в группу docker.
3. Настроить прокси для сервиса Docker
4. Для очистки генерируемых промежуточных файлов Buildroot необходимо выполнять команду:

make clean

1. Запустить сборку Buildroot и образа SD-карты в контейнере:

docker-build.sh make

Результаты сборки ПО располагаются в директории buildroot/output/images. Результатом сборки дистрибутива являются:

* ibl\* – образы загрузчиков IBL;
* ddrinit\*/ddrinit.\* – образы загрузчика DDR Initializer;
* daimyoboot-\* – образы загрузчика Daimyoboot;
* u-boot.bin – образы загрузчика U-Boot;
* vmlinux.gz.itb – скомпилированное ядро Linux;
* spi-\*.img – образы прошивки SPI-памяти;
* solaris-buildroot-sdcard.img – образ SD-карты, содержащий ОС GNU/Linux;
* rootfs.tar – образ корневой файловой системы.

Пакетное добавление в сборку пользовательских программ и библиотек

Хранение пользовательских рецептов и конфигураций внутри основного дерева Buildroot не всегда является удобным: во-первых, невозможно четко разделить части ПО с закрытой и открытой лицензией, а во-вторых, это повышает сложность обновления Buildroot. Механизм BR2\_EXTERNAL позволяет хранить пользовательские рецепты и конфигурации, а также другие артефакты вне основного дерева Buildroot. Механизм может использовать только для пакетного добавления в сборку Buildroot программ и библиотек пользователя. Модификация существующих пакетов таким образом невозможна.

«RoboDeus» Buildroot SDK позволяет добавлять и собирать новые пакеты для запуска на целевой платформе (52). Пакет в Buildroot это набор файлов конфигурации и сборки для приложений, библиотек, модулей ядра, входящих в состав корневой файловой системы. Для сборки пакета следует использовать одну из следующих инфраструктур сборки:

* autotools-package - для пакетов, сборка которых основана на autotools;
* python-package - для Python-пакетов, использующих для сборки distutils, setuptools;
* perl-package для Perl-пакетов;
* cmake-package - для пакетов, сборка которых основана на CMake.
* generic-package - для пакетов, для которых невозможно использовать вышеперечисленные инфраструктуры.

Наиболее часто используемой инфраструктурой является autotools-package. Каждый пакет в дистрибутиве Buildroot имеет свою собственную поддиректорию в директории package. В рамках инфраструктуры autotools-package директория пакета может состоять из следующих файлов описания пакетов:

* Config.in - файл конфигурации пакета. Содержит описание опций конфигурации пакета. Наличие файла является обязательным;
* <pkg>.mk - файл сборки пакета, содержащий информацию о том, как пакет следует собирать. Наличие файла является обязательным;
* <pkg>.hash - файл, содержащий хэш-суммы загружаемых файлов. Наличие файла не является обязательным.

Пакеты, необходимые для сборки и установки проприетарного Elvees OpenMPI добавлены в external-elcore50, т.е. во внешнее дерево пакетов Buildroot для поддержки DSP Elcore-50 Quad. В связи с тем, что OpenMPI 2.1.6 собирается под несколько аппаратных платформ (mips, elcore-50) требуется следующий ряд пакетов: аvatar (фактический SysCall для запуска OpenMPI на Elcore50), elvees-openmpi-dsp (сборка OpenMPI 2.1.6 под MCom-03/Solaris DSP), elvees-openmpi-cpu (сборка OpenMPI 2.1.6 под MCom-03/Solaris CPU) и проприетарный multicore-mpi, т.е. собственная версия OpenMPI в рамках адаптации модели параллельного программирования MPI. Проприетарные библиотеки openmpi и elcore-mpi реализуют программный интерфейс MPI для CPU и DSP соответственно. Реализован ряд наиболее часто используемых объектов и вызовов (порядка 25 функций).

Для добавления нового пакета в дистрибутив «RoboDeus» Buildroot необходимо (на примере elvees-openmpi-dsp):

1. В директории package создать поддиректорию с названием пакета:

buildroot/external-elcore50/package/elvees-openmpi-dsp

2. В директории пакета создать файл Config.in с описанием опций конфигурации пакета. Файл конфигурации пакета Config.in:

config BR2\_PACKAGE\_DSP\_ELVEES\_OPENMPI

bool "dsp-elvees-openmpi"

select BR2\_PACKAGE\_AVATAR

help

A build of OpenMPI 2.1.6 for MCom-03/Solaris DSP

3. Добавить в файл конфигурации пакетов верхнего уровня buildroot/external-elcore50/Config.in ссылку на файл конфигурации добавляемого пакета.

source "$BR2\_EXTERNAL\_ELCORE50\_PATH/package/dsp-elvees-openmpi/Config.in"

4. В директории пакета создать файл <pkg>.mk c описанием правил сборки пакета. Файл сборки пакета dsp-elvees-openmpi.mk:

ELVEES\_OPENMPI\_VERSION = 2.1.6

ELVEES\_OPENMPI\_SITE=https://www.open-mpi.org/software/ompi/v2.1.6/downloads

ELVEES\_OPENMPI\_SOURCE = openmpi-2.1.6.tar.bz2

ELVEES\_OPENMPI\_DEPENDENCIES = avatar

ELVEES\_OPENMPI\_BIN\_ARCH\_EXCLUDE = $(AVATAR\_TARGET\_DIR)/dsp

ELVEES\_OPENMPI\_CONF\_OPTS += --prefix=$(AVATAR\_TARGET\_DIR)/cpu

и т.д.

Хотя пакет test-mpi включен в состав пакетов внешнего дерева external-elcore50, в отличие от полноценного Elvees OpenMPI, он содержит только ряд тестов для замера производительности и проверки корректности выполнения MPI-функций проприетарной реализации программного интерфейса передачи сообщений multicore-mpi. Таким образом, необходимо добавить ряд эталонных MPI-бенчмарков в репозиторий research/test-mpi, а затем заменить его в суперрепозитории solaris/buildroot. Для редактирования исходников пакетов Buildroot используется механизм local override. Чтобы собрать Buildroot с изменённым исходным кодом research/test-mpi необходимо:

1. Клонировать research/test-mpi:

git clone "ssh://<user>@gerrit.elvees.com:29418/research/test-mpi test-mpi"

2. Создать файл local.mk для переопределения пакета (local override):

echo TEST\_MPI\_OVERRIDE\_SRCDIR=$(pwd)/test-mpi > buildroot/buildroot/local.mk

Имя пакета TEST\_MPI должно соответствовать имени пакета в рецепте Buildroot. Рецепт содержится в одном из внешних деревьев external-\*.

3. Редактировать исходные коды в выгруженном репозитории test-mpi.

4. Пересобрать пакет, выполнив следующие команды в корне выгруженного суперрепозитория solaris/buildroot (результат сборки появится в директории buildroot/output/images):

make test-mpi-reconfigure

make test-mpi-build

Структура репозитория test-mpi после учета исходных файлов бенчмарков в системе автоматической сборки CMake выглядит следующим образом:

├── build

├── ci

│ └── build

├── Makefile

├── CMakeLists.txt

├── 3dparty

│ └── openmpi-2.1.6

├── tests

│ └── mmpi-tests

│ └── <function-name>

└── benchmarks

│ └── <benchmark-name>

В директории ci (continuous integration) лежат специальные shell-скрипты. Основной скрипт build настроен таким образом, чтобы запустить выполнение нескольких целей из Makefile верхнего уровня в параллельном режиме на доступном числе процессорных ядер:

#! /bin/bash -xe

NUMBER\_OF\_PROCESSORS=$(nproc)

make -j$NUMBER\_OF\_PROCESSORS

Makefile уже сообщает make каким образом скомпилировать и скомпоновать программу. Список возможных целей указан после специального обозначения .PHONY. а цели, которые необходимо выполнить, перечислены после общей цели all:

TOP\_DIR := $(CURDIR)

BUILD\_DIR = $(TOP\_DIR)/build

BUILD\_E50 = $(BUILD\_DIR)/dsp

BUILD\_X86 = $(BUILD\_DIR)/cpu

.PHONY: build-cpu test-cpu build-dsp

all: build-cpu build-dsp

build-cpu:

mkdir -p $(BUILD\_X86)

cd $(BUILD\_X86) \

&& cmake $(TOP\_DIR) -DCMAKE\_INSTALL\_PREFIX=$(BUILD\_DIR) -DOUTDIR=cpu \

&& $(MAKE) all install

test-cpu:

cmake --build $(BUILD\_DIR)/cpu --target test

build-dsp:

make --file ci/Makefile.env

mkdir -p $(BUILD\_E50)

source ci/activate.env \

&& cd $(BUILD\_E50) \

&& cmake $(TOP\_DIR) -DCMAKE\_INSTALL\_PREFIX=$(BUILD\_DIR) \

-DCMAKE\_TOOLCHAIN\_FILE=$(ELCORE50\_TOOLCHAIN) \

-DLD\_SCRIPT=$(TOP\_DIR)/avatar/openmpi/openmpi.ld \

-DOUTDIR=dsp \

&& $(MAKE) all install

Цели build-cpu и build-dsp создают специальные директории (/cpu и /dsp соответственно) в основной временной директории сборки build и командой cmake для CMakeLists.txt верхнего уровня запускают сценарий полной сборки проекта. В файле CMakeLists.txt указываются: версия CMake, опции компилятора, директории bin для бинарных файлов, lib для файлов библиотек (статических .a, динамических .so и описаний .la), include для заголовков, необходимые пакеты и библиотеки (mpi open-rte open-pal pthread rt m), а также задаются конкретные настройки под необходимую аппаратную платформу (elcore-50, mips, x86 и sim3x). Далее CMakeLists.txt собирает все исходники в своём каталоге и в подкаталогах (подкаталоги также содержат дочерние файлы CMakeLists.txt):

add\_subdirectory(avatar/openmpi)

add\_subdirectory(benchmarks)

В директории avatar/openmpi собирается аппаратнозависимая версия openmpi из исходных файлов директории 3dparty/openmpi-2.1.6. А директория benchmarks содержит исходники бенчмарков для замера эффективности MPI-функций, пропускной способности и латентности (набор тестов определен в 1 главе):

add\_subdirectory(mpi-bench-suite-2003)

add\_subdirectory(netpipe-3.7.2)

add\_subdirectory(phloem-1.4.5)

add\_subdirectory(b-eff-3.6.0.1)

add\_subdirectory(intel-mpi-benchmarks-2021.3)

В итоге собирается порядка 13 эталонных тестов. Пример сборки через CMakeLists.txt одного из бенчмарков (netpipe-3.7.2):

cmake\_minimum\_required(VERSION 3.1)

project(NPMPI)

set(CMAKE\_C\_STANDARD 11)

set(CMAKE\_CXX\_STANDARD 11)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

enable\_language(CXX)

enable\_language(C)

set(GCC\_COVERAGE\_COMPILE\_FLAGS "-DMPI")

add\_definitions(${GCC\_COVERAGE\_COMPILE\_FLAGS})

add\_executable(NPmpi mpi.c netpipe.c netpipe.h)

add\_dependencies(NPmpi openmpi)

install(TARGETS NPmpi

RUNTIME DESTINATION ${OUTBIN})

Разработка алгоритма измерения основных сетевых характеристик кластерной системы

Для оценки коммуникационной инфраструктуры кластерной системы следует иметь возможность измерить ее основные характеристики. Коммуникационную среду можно достаточно полно охарактеризовать двумя параметрами: латентностью - временем задержки при посылке сообщения, и пропускной способностью - скоростью передачи информации. Все перечисленные характеристики, как правило, измеряются специальными программами или бенчмарками. Обязательно использование именно эталонных программ, иначе сравнение характеристик, измеренных разными программами, будет считаться некорректным. Свободно распространяемые эталонные бенчмарки, в которых используется интерфейс передачи сообщений (MPI), рассмотрены в 1 главе. На базе их исходного кода необходимо предварительно создать исполняемые файлы.

Поскольку узлы гетерогенного кластера неоднородны, то исполняемые файлы должны быть скомпилированы под соответствующую аппаратную платформу (x86, mips, elcore-50 и т.д.). Таким образом, исполняемые файлы бенчмарков под mips и elcore-50 должны быть обязательно скомпилированы в окружении системы сборки Buildroot. Конечно, перед этим Buildroot должен быть настроен (например, установлены proxy и docker), а пакет test-mpi с эталонными программами либо добавлен в директорию одного из внешних деревьев external-\*, либо переопределен с использованием механизма local override. Параллельно можно произвести настройку окружения для компиляции исполняемых файлов под x86, которые собираются непосредственно в репозитории test-mpi. При возникновении проблем со сборкой проекта можно использовать готовые исполняемые файлы со страницы релиза Elvees OpenMPI 2022.09.0.9 (https://box.elvees.com/index.php/s/ GTggLeBjWaCwa5C).

Вычислительный кластер, как правило, работает под управлением одной из разновидностей ОС Unix, в частности, исследуемый стенд с СнК «Robodeus» работает под управлением ОС GNU/Linux. Система сборки Buildroot поддерживает пакетное добавление в сборку программ и библиотек пользователя, поэтому сборка и установка Elvees OpenMPI под платформу «Robodeus» осуществляется в составе дистрибутива Buildroot отдельными пакетами: avatar, elvees-openmpi-cpu, elvees-openmpi-dsp, multicore-mpi. Предполагается, что образ дистрибутива операционной системы (вместе с образом rootfs) собран заранее на другой машине, затем записан на SD-карту и установлен через неё на стенд, поэтому без существенных изменений пересобирать его заново не требуется.

После установки удаленного SSH-соединения между рабочим ПК и хост-машиной стенда, для запуска исполняемых файлов на плате «Robodeus» необходимо произвести включение и загрузку собранного в Buildroot дистрибутива ОС GNU/Linux с SD-карты или через NFS. Бинарные файлы бенчмарков должны быть либо продублированы на всех узлах кластера, либо если узлов много, то загружены в общую директорию. В случае запуска бенчмарка между несколькими узлами вычислительной системы обязательно создание конфигурационного файла с указанием ip-адресов машин и количества процессов на каждом узле. Конфигурационный файл hosts также как и исполняемые файлы должен быть загружен в общую директорию.

Эталонная MPI-программа запускается посредством командной строки mpirun c необходимыми аргументами (количество задействованных процессов, имена узлов кластера, имя исполняемого файла и т.д.). Методика измерения пропускной способности соответствует паттерну PingPong (рис. 3.10).



Рисунок 3.10 — Паттерн PingPong

Алгоритм измерения пропускной способности между двумя узлами кластера представлен на рисунке 3.11. На узле <node-0> генерируется сообщение необходимого объема (как правило, размер соответствует степеням числа 2). Замеряется текущее время time1, а после начинается процесс отправки и приема сообщения. Процесс с номером 0 (узел <node-0>) посылает процессу с номером 1 (узел <node-1>) сообщение длины Imsg байт. Процесс 1, приняв сообщение от процесса 0, посылает ему ответное сообщение той же длины. Используются блокирующие (blocking) вызовы MPI (MPI\_Send, MPI\_Recv). Эти действия повторяются N раз с целью минимизировать погрешность за счет усреднения. Процесс 0 измеряет общее время time, затраченное на все эти обмены. Пропускная способность P(Imsg) определяется по формуле: P(Imsg) = 2\*N\*Imsg/time(Imsg). Латентность lat измеряется как время, необходимое на передачу сигнала, или сообщения нулевой длины.

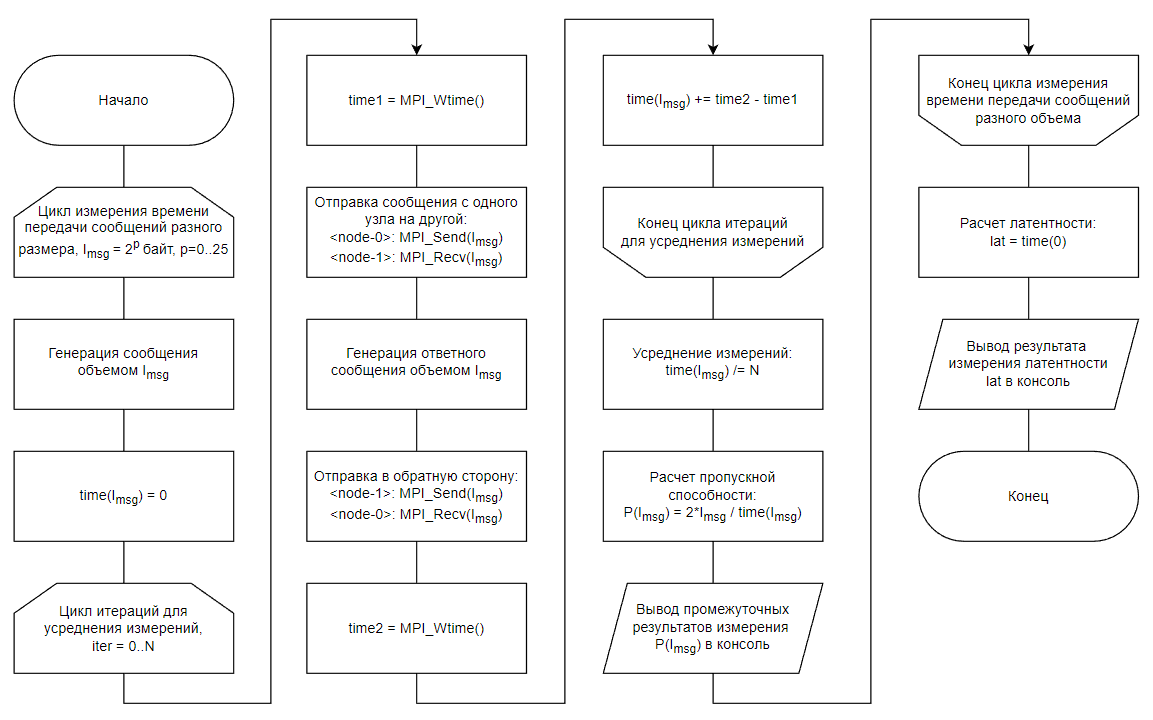


Рисунок 3.11 — Алгоритм измерения пропускной способности между двумя узлами кластера

Во всех конфигурациях тест считается успешно завершенным, если на экран монитора не выведен код ошибки. После выполнения теста результаты записываются в лог-файл, а работа с вычислительным кластером завершается: осуществляется выход со всех терминалов, питание стенда отключается, а SSH-соединение с хостом разрывается. Окончательная версия алгоритма представлена на рисунке 3.12.

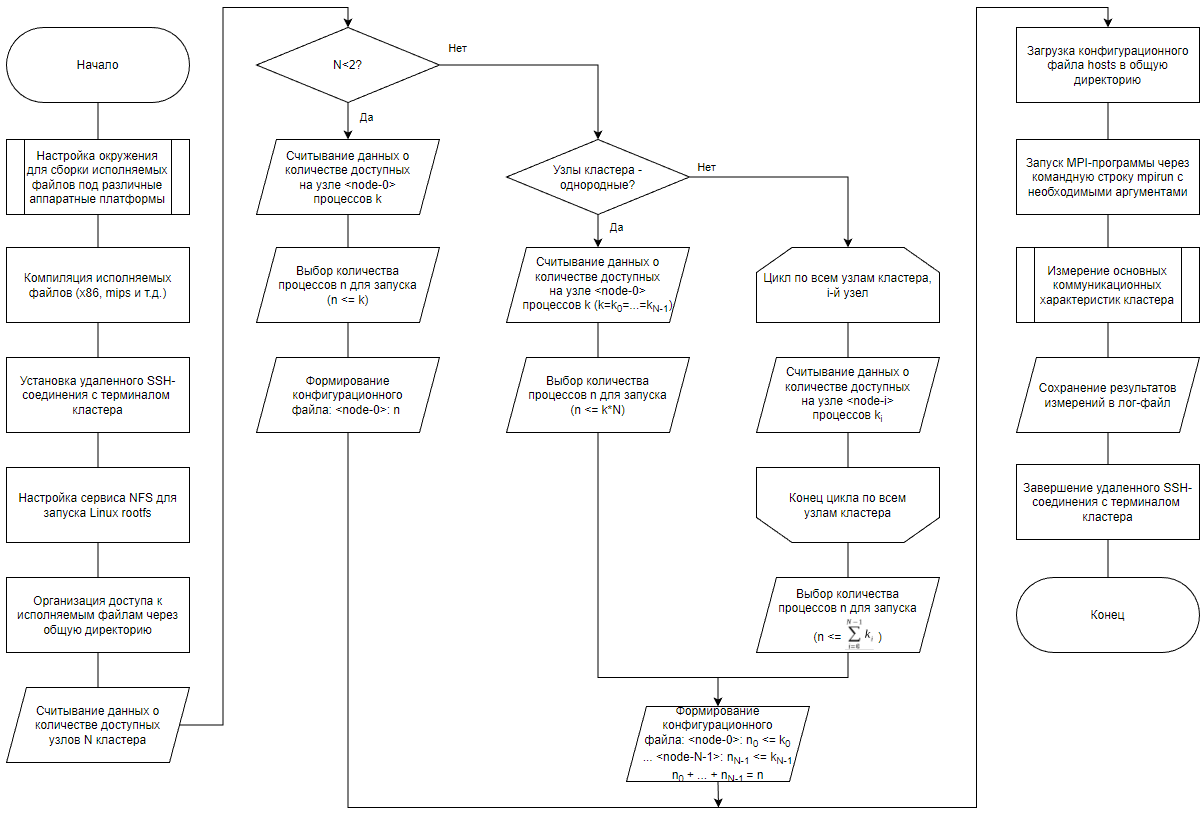


Рисунок 3.12 — Алгоритм измерения основных сетевых характеристик кластерной системы

## Выводы по главе 3

В ходе диссертационного исследования произведены следующие работы:

* изучены преимущества модели виртуального разделения основного процесса на субпроцессы для запуска MPI-программ;
* разработана методика ускорения выполнения MPI-приложений на кластере гетерогенного типа;
* описана структура системы сборки дистрибутива ОС Linux Buildroot с последующим добавлением пользовательских пакетов в сборку;
* разработан алгоритм измерения основных сетевых характеристик кластерной системы с использованием алгоритма измерения пропускной способности между двумя узлами кластера.

# ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНОЕ ПОДТВЕРЖДЕНИЕ РЕЗУЛЬТАТОВ ИССЛЕДОВАНИЯ

2. 1. Описание стенда для проведения экспериментальных исследований

В данной разрабатывается концепция гетерогенного HPC (High-performance computing) кластера, в вычислительных узлах которого находятся серверные комплекты RoboDeus SDV (Solo Deep Vision), состоящие из материнской платы RoboDeus SHB (Single Head Board) и комплектующих для сервера (блоки питания, резервирования, система охлаждения, память, BMC устройство и др.), что позволит совместно со стеком программного обеспечения создавать серверные системы различного назначения.

Cерверная плата RoboDeus SHB содержит процессор, шину данных, память, порты ввода/вывода, широкий набор периферийных интерфейсов, а также дисплейные и мультимедийные интерфейсы. Вычислительные процессы запускаются на соответствующих узлах, управление же осуществляется посредством терминального доступа с управляюшей хост-машины кластера, представленной эталонным сервером Supermicro 2xXeon Gold.

На рисунке 4.1 схематично изображен состав кластера.

Важно заметить, что данная конфигурация представляет собой объединение нескольких кластеров (серверных комплектов) вместе с дополнительными устройствами, что обеспечивает независимую оперативную память для каждого узла.

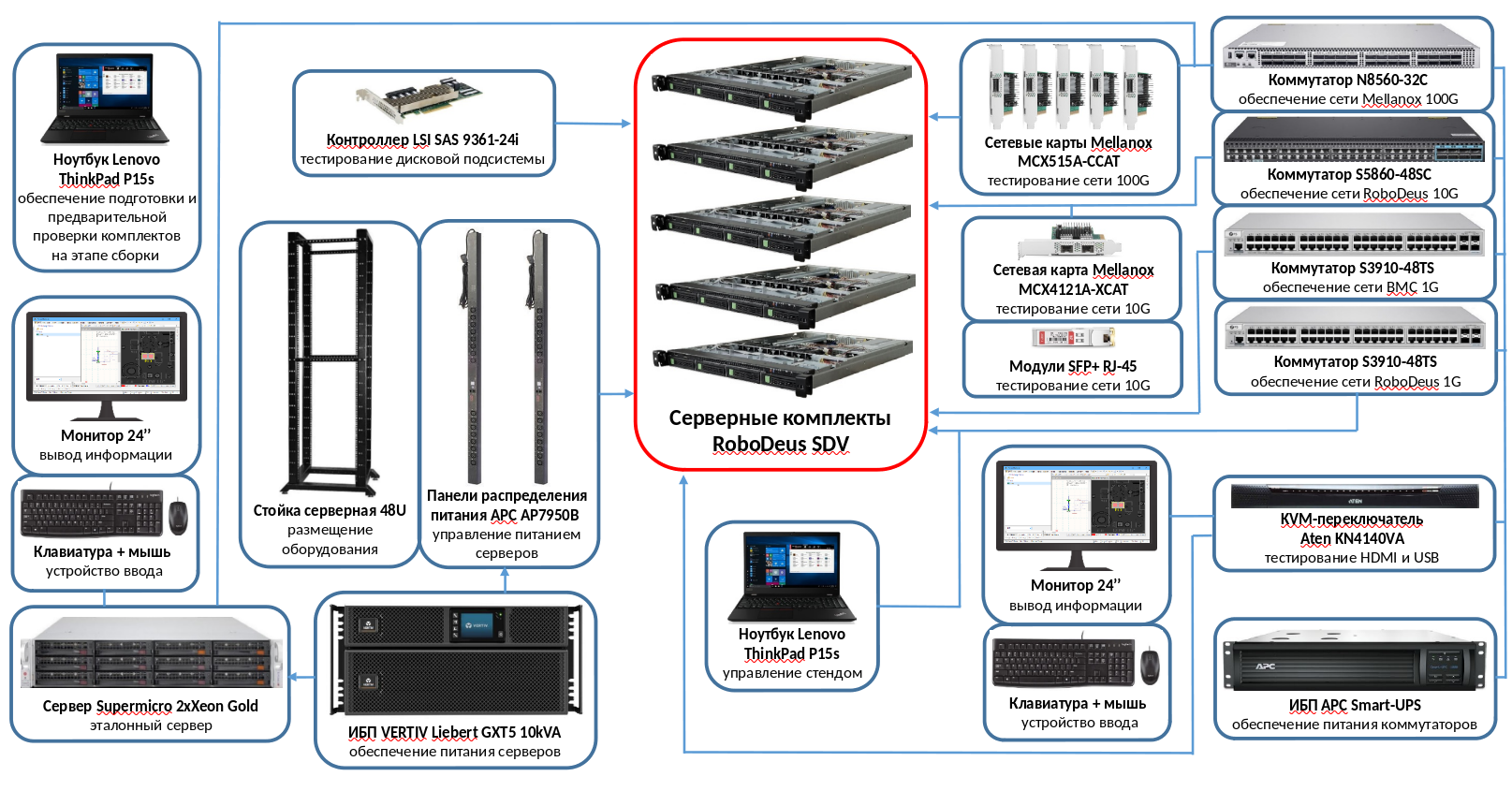


Рисунок 4.1 — Состав стенда «RoboDeus» SDV

Для организации коммуникационной инфраструктуры разрабатываемого кластера могут использоваться как передовые высокоскоростные сетевые технологии, например, 100 Gigabit Ethernet (100 GbE), так и менее эффективные, но более распространенные версии - 1 GbE и 10 GbE, поскольку доступны коммуникаторы на 1G, 10G и 100G с соответствующими сетевыми картами Mellanox.

При наличии проблем с доступом к вычислительному кластеру, экспериментальные исследования могут проводиться на стендах, сконструированных на основе отладочных комплектов SSP BuB или Semantic Server Processor Bring-up Board и управляющих хост-машин стендов, в большинстве случаев, представляющих собой персональные компьютеры с 8-ми ядерными x86 процессорами (например, Intel Core i7-9700KF и AMD Ryzen 5 3400G). Состав отладочного комплекта SSP BuB представлен в таблице 4.1. Основной его составляющей является интегральная микросхема «Robodeus» (1892ВМ248) с кластерами ядер разных архитектур.

Таблица 4.1 — Состав отладочного комплекта SSP BuB

|  |  |
| --- | --- |
| **№** | **Наименование** |
| 1 | Материнская плата SSP BuB с процессором «RoboDeus»  (1892ВМ248) и модулем SPI-flash SOIC16WMEM |
| 2 | Процессорный кулер DEEPCOOL CK-11508 |
| 3 | Открытый стенд Streacom BC1 |
| 4 | Блок питания ATX 650Вт Corsair RM650x |
| 5 | Модуль памяти UDIMM DDR4 3200 16GB Kingston |
| 6 | Карта памяти microSD 32GB SanDisk High Endurance |
| 7 | Кабель microUSB 2.0 1,8м |
| 8 | Сетевой адаптер Ethernet USB 3.0 DUB-1312 |
| 9 | Переходник microUSB 3.0 B(m) - USB 3.0 A(f) |

Для работы с отладочным комплектом SSP BuB (рис.4.2) используются следующие аппаратно-программные средства:

1. Управляющий ПК на ОС CentOS 8 x86-64 и установленными пакетами iperf3, minicom, putty и правами суперпользователя.
2. Коммутатор или маршрутизатор Ethernet для организации тестовой локальной сети.
3. DHCP-сервер в тестовой локальной сети.

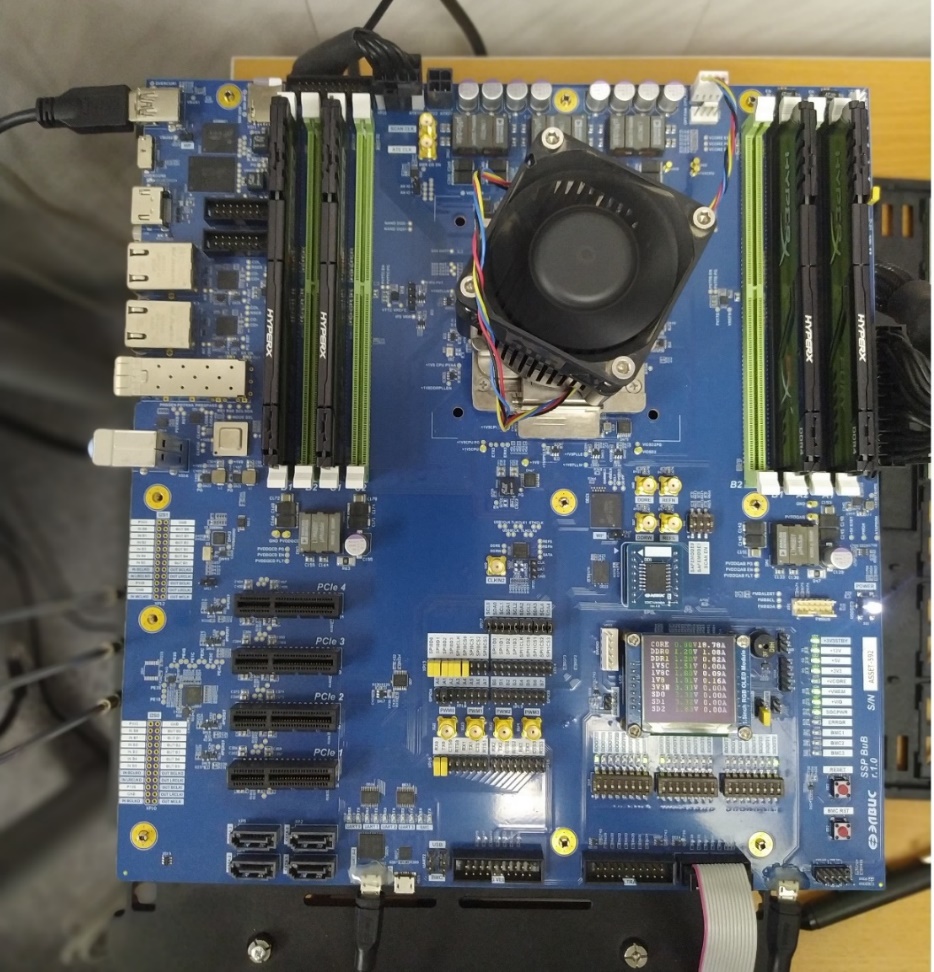


Рисунок 4.2 — Экземпляр отладочного комплекта SSP BuB для проведения экспериментальных исследований

Схема подключения тестового стенда с отладочным комплектом SSP BuB (рис. 4.3):

1. Кабель microUSB 2.0 к разъему XS14 BMC&UART 0-2 материнской платы (для работы с «RoboDeus») и управляющему ПК.
2. Кабель microUSB 2.0 к разъему XS17 BMC USB материнской платы (для работы с BMC) и управляющему ПК.
3. Патч-корд от сетевого адаптера Ethernet USB к тестовой локальной сети.

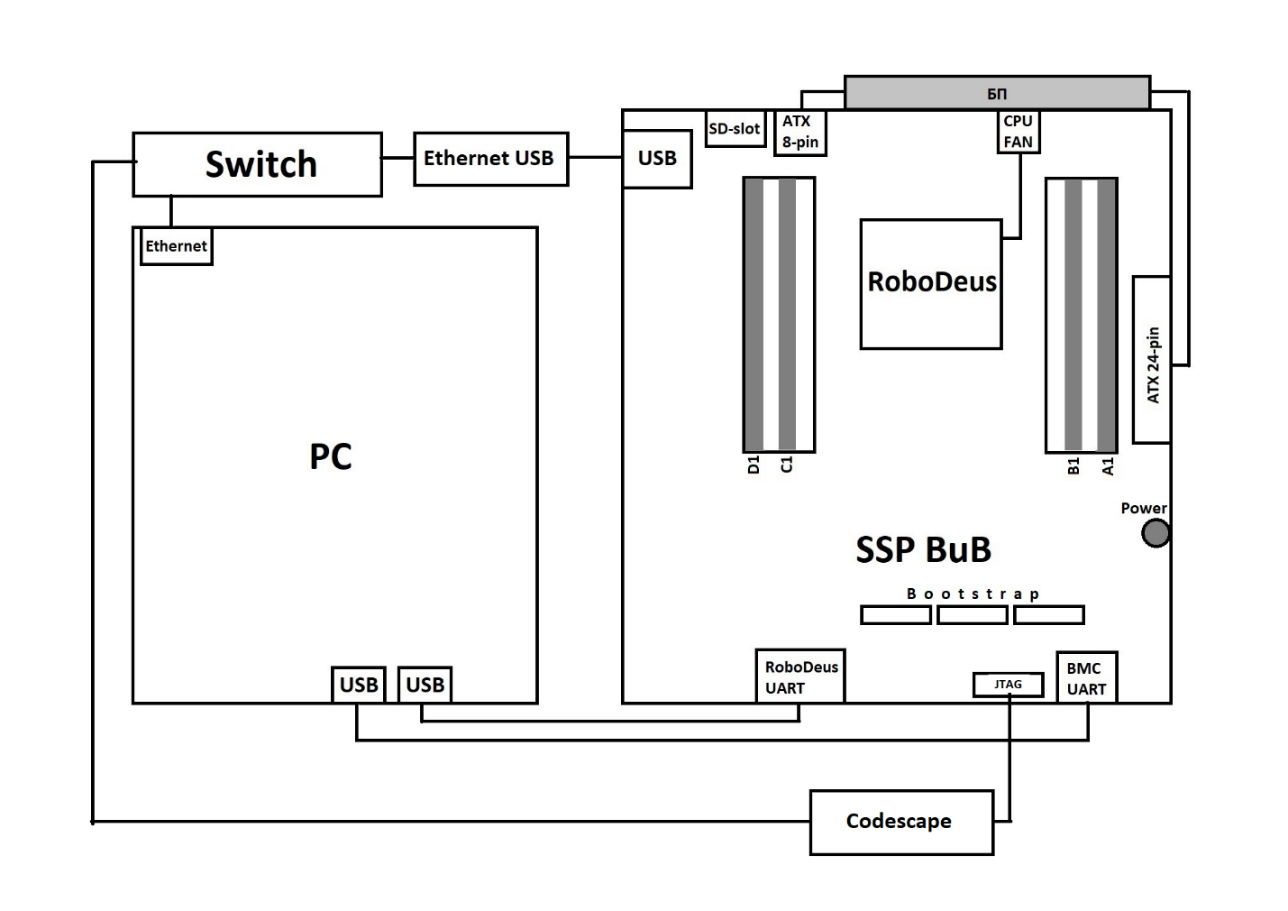


Рисунок 4.3 — Схема подключения стенда с отладочным комплектом SSP BuB (с JTAG-отладчиком)

На рисунке 4.3 можно заметить, что при подключении стенда со стороны отладочного комплекта SSP BuB вместо Ethernet разъема используется разъем USB. Использование кабеля Ethernet/LAN является довольно очевидным выбором, поскольку даже самый дешевый Gigabit Ethernet считается высокоскоростным, т.к. поддерживает скорость передачи до 1 Гбит/с, но зачастую для тестирования коммуникационной сети узлов кластера используются соединения через USB.

Перед непосредственной работой на стенде следует его включить и убедится в корректности настроек. Для включения нужно перевести выключатель питания БП ATX тестового комплекта в положение ON. Настройки используемого сетевого адаптера должны соответствовать DHCP-серверу тестовой локальной сети. Также следует произвести проверку наличия четырех интерфейсов cp210x converter (ttyUSB0, ttyUSB1, ttyUSB2, ttyUSB3) и интерфейса ttyACM0 USB ACM device, выполнив команду

dmesg | grep tty

Диаграмма последовательности (рис. 4.4) отражает процесс удаленного подключения пользователя с рабочего ПК (web-интерфейс) на хост-машину стенда (Host) через SSH. Подключение к отладочному комплекту SSP BuB возможно несколькими способами:

1. Подключиться к BMC SSP BuB через консольный терминал minicom с использованием значения нового порта USB ACM device (например, ttyACM0).

minicom -D /dev/ttyACM0

Включить SSP BuB через консольную команду, а после убедиться, что кулер на процессоре Robodeus включился.

power.power\_on()

По окончанию работы на плате выключить SSP BuB и выйти из терминала minicom (Ctrl A, затем Q).

power.power\_off()

2. Подключиться к Robodeus SSP BuB через консольный терминал minicom с использованием наименьшего значения порта cp210x (например, ttyUSB0).

minicom -D /dev/ttyUSB0

Нажать кнопку Power на SSP BuB, а после убедиться, что кулер на процессоре Robodeus включился. В терминале minicom должны начать выводиться информационные сообщения загрузки Linux. По окончании загрузки Linux, войти, используя логин root без пароля.

Выяснить и зафиксировать текущий IP-адрес SSP BuB.

ifconfig eth0

При необходимости, произвести настройку сети Robodeus SSP BuB в конфигурационных файлах /etc/network/interfaces на корневой файловой системе (дефолтные настройки – DHCP). Для изменения имени хоста необходимо отредактировать конфигурационные файлы /etc/hostname и /etc/hosts на корневой файловой системе.

По окончанию работы на плате выключить SSP BuB, нажав кнопку Power на SSP BuB, и выйти из терминала minicom (Ctrl A, затем Q).

3. Подключиться к Robodeus SSP BuB через SSH по стандартному порту. Необходимо использовать сетевое имя (дефолтное имя – solaris) либо ip-адрес устройства (имя пользователя – root).

ssh root@solaris или ssh root@<ip-адрес-SSP-BuB>

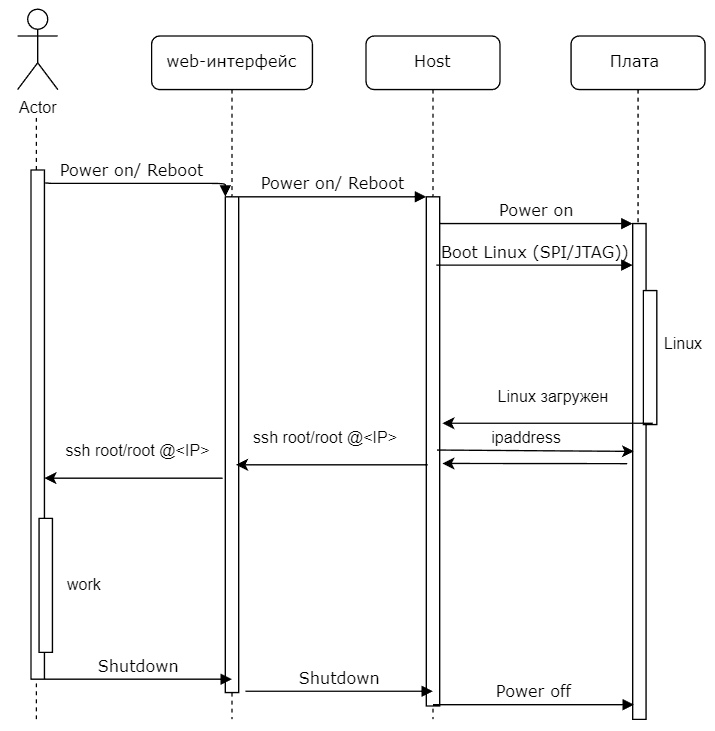


Рисунок 4.4 — Диаграмма последовательности, отражающая процесс подключения к отладочному комплекту SSP BuB

Порядок проведения экспериментальных исследований

В соответствии с алгоритмом измерения основных сетевых характеристик кластерной системы (пункт 3.6) следует настроить окружение сборки исполняемых файлов под различные аппаратные конфигурации. Поскольку время работы на вычислительном кластере, как правило, ограничено, можно заранее скомпилировать исполняемые файлы на рабочем ПК, но для этого необходимо провести некоторую первоначальную настройку (предполагается, что хост-машина кластера настроена аналогичным образом):

1. Поскольку репозиторий с исходными файлами бенчмарков и Elvees OpenMPI хранится в сети, следует настроить proxy для стабильного подключения к интернету. Можно прописать следующие строчки в файл ~/.bashrc (через bash-терминал):

cat >> ~/.bashrc << EOF

# Proxy settings

export http\_proxy=http://proxy.elvees.com:3128

export ftp\_proxy=http://proxy.elvees.com:3128

export https\_proxy=http://proxy.elvees.com:3128

export no\_proxy=127.0.0.1,localhost,elvees.com

EOF

Для вступления настроек в силу без перезагрузки:

source ~/.bashrc

2. В этом же файле следует добавить загрузку модулей (Environment modules), необходимых для установки переменных окружения для использования компилируемого вручную ПО с NFS-сервера /usr/corp:

module use /usr/corp/Projects/ipcam-vip1/modules

module use /usr/corp/eltools/modules

module load cmake

module load git

module load texlive

module load picutils

module load toolchain/aarch64/linaro/linux/2019.02

module load automake

3. Репозиторий test-mpi сохранен в системе управления версиями git, поэтому если SSH-ключ рабочего ПК не добавлен в профиль пользователя Gerrit, необходимо это сделать. По умолчанию пользовательские SSH-ключи хранятся в каталоге ~/.ssh, поэтому стоит проверить, присутствует ли там открытый ключ id\_rsa.pub. И если его там нет, то сгенерировать и вывести в консоль (путь к файлу оставить по умолчанию, запрос passfrase можно пропустить):

if [ ! -f ~/.ssh/id\_rsa.pub ]; then ssh-keygen -N "" -f ~/.ssh/id\_rsa; fi

cat ~/.ssh/id\_rsa.pub

Далее следует скопировать ключ и добавить в настройках профиля https://gerrit.elvees.com/settings/#SSHKeys:

User → Settings→ SSH keys → Add new SSH key, нажать Save Changes

Окружение для сборки настроено, теперь можно скомпилировать исполняемые файлы в формате x86. Для начала следует клонировать проект research/test-mpi.

git clone "ssh://<user>@gerrit.elvees.com:29418/research/test-mpi"

Перейти в корневую директорию проекта.

cd test-mpi

Поскольку подмодули могут быть не синхронизированы с сервером репозитория, их следует обновить. Также следует внести последние коммиты в локальное рабочее дерево Git.

git submodule update --init --recursive

git submodule update –remote

Собрать проект.

ci/build

После сборки исполняемые файлы, файлы библиотек и заголовков располагаются по пути /home/<user>/test-mpi/build/cpu. При возникновении проблем со сборкой проекта можно использовать готовые исполняемые файлы со страницы релиза Elvees OpenMPI 2022.09.0.9 (https://box.elvees.com/index.php/s/ GTggLeBjWaCwa5C). Для этого следует загрузить архив research-test-mpi-x86.zip и распаковать его в созданную для размещения тестов директорию /home/<user>/test-mpi/build/cpu/bin, а также предоставить права на данную папку.

chmod -R ugo+x /home/<user>/test-mpi/build/cpu/bin

Параллельно можно произвести настройку окружения для компиляции исполняемых файлов под mips и elcore50 на этом же ПК, поскольку сборка будет производиться с помощью BuildrootSDK в составе пакета test-mpi. Для сборки в контейнере Docker необходимо настроить прокси клиента.

mkdir -p ~/.docker

cat > ~/.docker/config.json <<EOF

{

"proxies":

{

"default":

{

"ftpProxy": "http://proxy.elvees.com:3128",

"httpsProxy": "http://proxy.elvees.com:3128",

"httpProxy": "http://proxy.elvees.com:3128",

"noProxy": ".elvees.com"

}

}

}

EOF

Затем убедиться, что пользователь добавлен в группу docker. Пользователи с правами sudo могут сделать это самостоятельно.

sudo usermod -a -G docker $(whoami)

Перед компиляцией бенчмарков следует единоразово собрать Buildroot. Для начала следует клонировать проект solaris/buildroot.

git clone "ssh://<user>@gerrit.elvees.com:29418/solaris/buildroot"

Перейти в корневую директорию проекта.

cd buildroot

Обновить подмодули проекта.

git submodule update --init --recursive

Запустить docker-сервис на Linux.

sudo systemctl start docker

export DOCKERFILE=Dockerfile.centos8stream

export ENABLE\_NETWORK=1

Собрать buildroot в конфигурации solaris\_defconfig и применением фрагмента hpc для высокопроизводительных вычислений согласно инструкции README.rst

./docker-build.sh make solaris\_defconfig FRAGMENTS=hpc

./docker-build.sh make

В этой же директории отдельно собрать пакет test-mpi.

make test-mpi-reconfigure

После сборки исполняемые файлы и библиотеки располагаются по пути /home/<user>/buildroot/buildroot/output/target/elvees-openmpi/cpu или /home/<user>/ buildroot/buildroot/output/target/elvees-openmpi/dsp. А в директории /home/<user>/ buildroot/buildroot/output/images находятся образы Buildroot для загрузки ОС GNU/Linux через JTAG. При возникновении проблем, аналогично сборке под x86, можно воспользоваться готовыми бинарными файлами, загрузив и распаковав архив research-test-mpi-mips.zip. Но нужно предоставить права на данную папку.

chmod -R ugo+x /home/<user>/buildroot/buildroot/output/target/elvees-openmpi/cpu

После установки удаленного SSH-соединения между рабочим ПК и хост-машиной стенда, оператору следует под своей учетной записью установить ряд ПО и подготовить среду терминала.

ssh -p 22 <username>@<host-ip>

Аналогично настройке рабочего ПК необходимо установить proxy для доступа в сеть (если они настроены глобально, это можно пропустить), настроить docker и зарегистрировать публичный SSH-ключ на Gerrit. Убедиться, что имя пользователя добавлено в группу dialout на хост-машине стенда. Если нет, то следует добавить пользователя в группу (заменить в команде <user> на имя пользователя). Для вступления прав в силу необходимо перезайти в свою учетную запись.

cat /etc/group

sudo usermod -a -G dialout <user>

Для загрузки некоторых утилит настраивается среда Python 2 и Python 3. После устанавливаются тулы управления Solaris BuB BMC bubctl и тулы сборки BIMG-образов и JTAG-скрипты загрузки boottools.

pip3 install --user git+ssh://gerrit.elvees.com:29418/tools/bubctl

pip2 install --user git+ssh://gerrit.elvees.com:29418/solaris/boottools

Например, с помощью утилиты bubctl можно удаленно управлять вычислительным узлом SSP BuB.

bubctl sysinfo

bubctl power on

bubctl power off

Для запуска Linux rootfs на вычислительном узле с процессором «Robodeus» необходимо установить и запустить NFS сервис. Исходный код окружения доступен путем клонирования репозитория docker/boot-server.

git clone ssh://gerrit.elvees.com:29418/docker/boot-server

cd /home/<user>/boot-server

Следует убедиться, что на ПК запущена служба rpcbind, остановлена служба nfs и загружен модуль ядра nfsd (актуальная инструкция по подготовке запуска NFS указана в README.rst репозитория):

sudo systemctl enable rpcbind

sudo systemctl start rpcbind

sudo systemctl disable nfs-server

sudo systemctl stop nfs-server

sudo modprobe nfsd

Собрать на хост-машине стенда docker-образ сервера NFS (достаточно однократное выполнение команды).

docker build --tag boot-server-nfs .

После запуска NFS-сервера экспортируемая директория становится доступной по адресу <HOST\_ADDRESS>:/exports, где <HOST\_ADDRESS> — имя хоста или его IP-адрес.

Поддерживаются следующие сценарии экспорта по NFS:

1. Экспорт файлов с уровнем привилегий выше уровня привилегий активного пользователя. Например, раздача корневой файловой системы из rootfs.tar с файлами пользователя root:

docker run --detach --privileged --rm --name boot-server-nfs --network host --tmpfs /exports --volume /var/run/rpcbind.sock:/var/run/rpcbind.sock boot-server-nfs

alias docker-exec="docker exec --interactive boot-server-nfs"

docker-exec tar -x -C /exports < rootfs.tar

2. Экспорт существующей директории хоста <EXPORTS> (путь к <EXPORTS> должен быть абсолютным):

docker run --detach --privileged --rm --name boot-server-nfs --network host --volume /var/run/rpcbind.sock:/var/run/rpcbind.sock --volume <EXPORTS>:/exports boot-server-nfs

Для запуска исполняемых файлов на вычислительном узле SSP BuB необходимо произвести включение и загрузку собранного в Buildroot дистрибутива ОС GNU/Linux (образы Solaris Buildroot скачиваются с сервера или собираются из исходных кодов) либо с SD-карты, либо через NFS. Загрузить ОС можно, воспользовавшись средствами solaris/boottools. Для этого необходимо запустить скрипт solaris-boot.py, указав вместо 00000 действительный номер JTAG-отладчика (для низкоуровневой отладки), а вместо /dev/ttyUSBx UART-порт ПК к которому подключен UART0 стенда или Solaris на плате RoboDeus SHB:

solaris-boot.py -j 00000 \

--xtal 25000000 \

--ibl ${IMG\_PATH}/iblns.bin \

--ddrinit ${IMG\_PATH}/ddrinit-solarisbubns/ddrinit.bin \

--dts ${IMG\_PATH}/macadamia/solaris-bub-r1.0.dtb \

--daimyoboot ${IMG\_PATH}/daimyoboot-ns.bin \

--mtpd ${IMG\_PATH}/mtpd.bin \

--uboot ${IMG\_PATH}/u-boot.bin \

--kernel ${IMG\_PATH}/vmlinux.gz.itb \

--uart-port /dev/ttyUSBx

Дождавшись завершения скрипта solaris-boot.py, запустить minicom или tio, указав вместо /dev/ttyUSBx UART-порт ПК к которому подключен UART0 стенда или Solaris на плате RoboDeus SHB. Залогиниться как root.

minicom -D /dev/ttyUSBx

tio /dev/ttyUSBx

По завершении работы с контейнером следует остановить его выполнение. Удаление контейнера выполняется автоматически.

docker stop boot-server-nfs

А при завершении работы на стенде необходимо отключать питание платы и выходить со всех подключенных терминалов к плате.

bubctl power off

killall minicom

Для запуска бенчмарков на нескольких узлах посредством mpirun необходимо, чтобы бинарные файлы на каждом узле располагались по одному и тому же пути. Поэтому необходимо организовать доступ к исполняемым файлам либо через общую директорию, либо продублировав директорию с подготовленными или скачанными файлами тестов с рабочего ПК на хост-машину стенда (в формате x86) и на его узлы (в формате mips).

cp -r /home/<user>/test-mpi/build/cpu/bin/\* /home/<user>/common/test-mpi/bin

scp -r /home/<user>/test-mpi/build/cpu/bin/\* <user>@<host-ip>:/home/<user>/

common/test-mpi/bin

scp -r /home/<user>/buildroot/buildroot/output/target/elvees-openmpi/

cpu/bin/\* <user>@<robodeus-ip>:/home/<user>/common/test-mpi/bin

Не забыть предоставить права на скопированные файлы.

chmod -R ugo+x /home/<user>/common/test-mpi/bin/

Перед запуском тестов следует произвести проверку доступных сетевых интерфейсов.

netstat -nr

В командах тестов, в ключе --mca btl\_tcp\_if\_exclude необходимо исключить неиспользуемые сетевые интерфейсы, перечислив их через запятую, а в переменной NCCL\_SOCKET\_IFNAME - с символом ^ перед названием интерфейса.

mpirun --mca btl\_tcp\_if\_exclude docker0,lo,virbr0 -x NCCL\_SOCKET\_IFNAME=^docker0,^lo,^virbr0

В случае запуска бенчмарка между несколькими узлами вычислительной системы обязательно создание конфигурационного файла с указанием ip-адресов машин и количества процессов на каждом узле. Если запуск бенчмарка производится на двух процессах (-np 2), то адреса рабочих машин указываются в теле команды после флага –host. Если бенчмарк требует более двух процессов (-np 4, -np 8), то необходимо создать в директории /home/<user>/common/test-mpi конфигурационный файл my-hosts с указанием hostname или ip-адресов рабочих машин и количества запрашиваемых процессов через параметр slots, и вместо флага --host использовать флаг --hostfile, после которого указать путь до созданного файла конфигурации. Суммарное количество процессов, указываемых в параметре slots должно соответствовать количеству процессов, которые запрашивались при запуске данной команды на одном ПК.

Пример содержимого файла my-hosts:

<hostname-or-ip-node#1> slots=2 max\_slots=20

<hostname-or-ip-node#2> slots=2 max\_slots=20

Также следует организовать беспарольный доступ по ключу RSA, скопировав публичный ключ того узла, с которого планируется запускать все тесты, на остальные узлы вычислительного кластера.

ssh-copy-id -i ~/.ssh/id\_rsa.pub <user>@<node-ip>

mpirun по умолчанию отказывается запускаться от имени пользователя root.

Чтобы переопределить это значение по умолчанию, необходимо добавить параметр --allow-run-as-root в командную строку mpirun. Но для обеспечения выполнения тестов x86-mips и mips-mips лучше создать пользователя с паролем, аналогичного пользователю хост-машины стенда (учетные данные должны быть идентичны актуальной доменной учетной записи).

adduser <user>

Описание возможных конфигураций узлов вычислительного кластера для измерение пропускной способности и латентности сети разобрано в пункте 4.1. Во всех конфигурациях тест считается успешно завершенным, если на экран монитора не выведен код ошибки.

Пример запуска эталонного теста на одном узле кластера в конфигурации x86 (например, это может быть хост машина стенда):

# NetPipe 3.7.2

/home/<user>/common/openmpi/bin/mpirun --mca btl\_tcp\_if\_exclude docker0,lo,virbr0 -x NCCL\_SOCKET\_IFNAME=^docker0,^lo,^virbr0 -np 2 /home/<user>/common/test-mpi/bin/NPmpi

Запуск этого же теста, но на вычислительном узле SSP BuB в конфигурации mips из терминала minicom (или tio):

# NetPipe 3.7.2

/home/<user>/common/openmpi/bin/mpirun --allow-run-as-root -np 2 /home/<user>/common/openmpi/bin/NPmpi

Пример запуска между двумя вычислительными узлами в конфигурации x86-x86 (рабочим ПК и хост-машиной стенда):

# NetPipe 3.7.2

/home/<user>/common/openmpi/bin/mpirun --mca btl\_tcp\_if\_exclude docker0,lo,vibr0 -x NCCL\_SOCKET\_IFNAME=^docker0,^lo,^vibr0 --host <pc-ip>,<host-ip> /home/<user>/common/openmpi/bin/NPmpi

# Intel MPI Benchmarks 2021.3

/home/<user>/common/openmpi/bin/mpirun --mca btl\_tcp\_if\_exclude docker0,lo,vr0 -x NCCL\_SOCKET\_IFNAME=^docker0,^lo,^vibr0 --hostfile /home/<user>/common/openmpi/bin/my-hosts /home/<user>/common/openmpi/bin/IMB-MPI1

<pc-ip> slots=2 max\_slots=20

<host-ip> slots=2 max\_slots=20

Запуск бенчмарков в конфигурациях mips-mips, как и x86-mips осуществляется аналогично конфигурации x86-x86. При этом в команде теста или в содержимом hostfile необходимо указать корректные ip-адреса узлов.

Экспериментальные исследования, подтверждающие достоверность  
полученных результатов

Описание возможных конфигураций узлов вычислительного кластера для измерения пропускной способности и латентности сети представлено на рисунке 4.5. В качестве узла для проведения экспериментальных исследований, сконфигурированных под архитектуру MIPS64, используется отладочный комплект SSP BuB, основной составляющей которого является гетерогенный многоядерный процессор «RoboDeus» (1892ВМ248). Для проведения измерений под архитектуру x86-64 в качестве узлов используются управляющие хост-машины стендов (см. пункт 4.1), представляющие собой персональные компьютеры с 8-ми ядерными x86 процессорами (например, Intel Core i7-9700KF и AMD Ryzen 5 3400G).

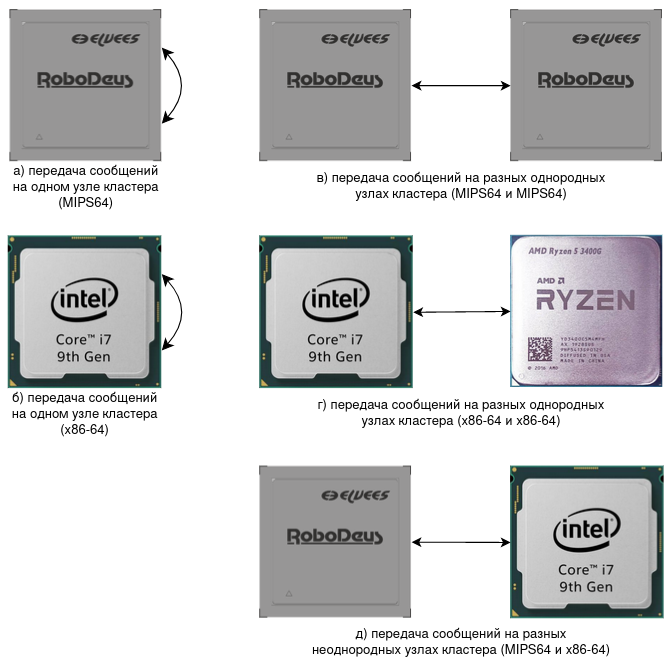


Рисунок 4.5 — Описание возможных конфигураций процессоров для передачи сообщений между узлами вычислительного кластера

Для того, чтобы теоретически рассчитать время передачи сообщений между узлами кластера, можно воспользоваться математической моделью из пункта 2.4. С помощью целевой функции рассчитывается время, затрачиваемое при отправке и получении сообщения объемом с одного процесса на другой, как на одном узле (рис. 4.5 а, б), так и на разных узлах (рис. 4.5 в, г) исследуемого кластера. Но для расчета времени передачи сообщений между неоднородными узлами (рис. 4.5 д) следует применить формулу . Для последующей оценки теоретических и экспериментальных результатов исследований, рассчитаем время передачи сообщений между узлами кластера в случаях на рисунке 4.5 а, б и д.

При расчете времени передачи сообщений между процессами на вычислительном узле «RoboDeus» (1892ВМ248) сразу примем , поскольку процессы находятся на одном узле кластера. Далее следует рассчитать ряд параметров, зависящих от аппаратных характеристик микропроцессора. Например, в формуле целевой функции используются пропускные способности соответствующих уровней памяти или коммутации. Согласно математической модели, каждая пропускная способность в общем случае рассчитывается следующим образом: , где – тактовая частота, – разрядность шины, – количество каналов передачи. В таблице приведены рассчитанные по приведенной формуле пропускные способности от CPU до DDR.

Таблица 4.2 — Пропускные способности соответствующих уровней памяти или коммутации микропроцессора «RoboDeus» (1892ВМ248)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Тактовая частота , МГц | Разрядность шины, бит | Количество каналов передачи | Пропускная способность , ГБ/с |
| Ядро CPU | 832 | 256 | 1 | 26 |
| Кэш первого уровня L1  (64 КБ) | 832 | 32 | 4\*16 | 208 |
| Кэш второго уровня L2  (1 МБ) | 600 | 128 | 4\*4 | 150 |
| Матрица коммутации NoC | 600 | 256 | 4 | 75 |
| Кэш третьего уровня L3  (16 МБ) | 3200 | 64 | 4 | 100 |
| Оперативная память DDR | 3200 | 64 | 4 | 100 |

На основе множества пропускных способностей можно рассчитать коэффициент пропускной способности:

|  |  |
| --- | --- |
|  | (1) |

Далее необходимо заняться расчетом суммарной задержки или латентности . Поскольку , учитываются только программная и аппаратная составляющие латентности, т.е. . Для расчета аппаратной задержки  дополнительно к тактовой частоте процессора и количеству тактов до DDR нужно знать размер строки кэша . Система кэшей микропроцессора «RoboDeus» сконфигурирована таким образом, чтобы иметь строки по 64 байта. В ином случае требовалось бы либо усреднять размер строки, либо рассчитывать задержку до каждого уровня по отдельности. Для расчета количества тактов до DDR универсальных формул нет, поскольку у этого параметра сильная зависимость от архитектуры микропроцессора. Если в документации к микросхеме отсутствует стандартное значение , то его требуется экспериментально замерить специальными тестами доступа в оперативную память.

Программная составляющая латентности измеряется как время, необходимое на передачу сигнала, или сообщения нулевой длины. На диаграмме (рис. 4.6) можно увидеть значения латентности MPI для различных конфигураций узлов, замеренные с помощью специального теста скорости межпроцессорных обменов (MPI Transfer Test) из пакета бенчмарков MPI Bench Suite 2003 (НИВЦ МГУ). 4й столбец отражает задержу при пересылке сообщения на узле «Robodeus» со стандартной тактовой частотой процессора ().

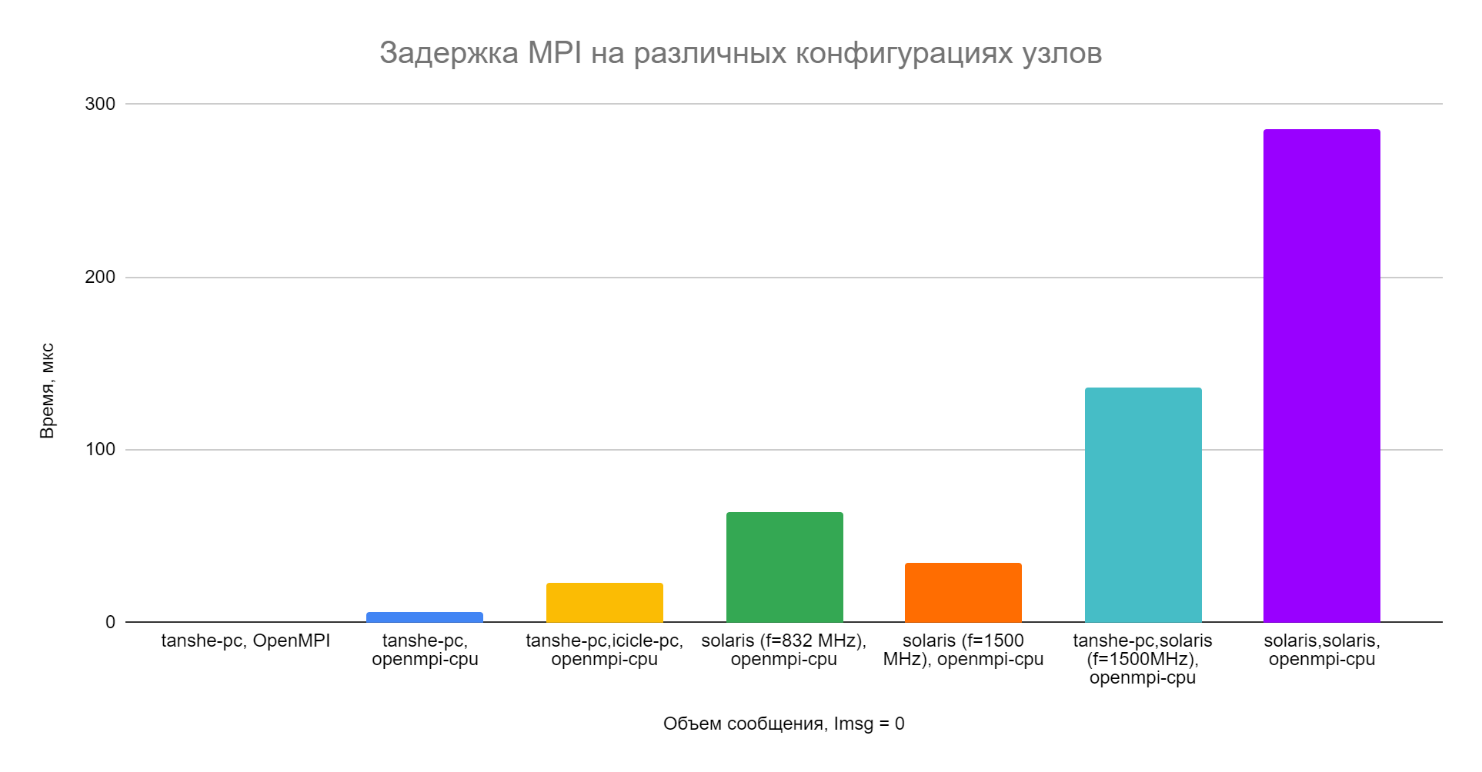


Рисунок 4.6 — Столбчатая диаграмма, отражающая задержу MPI на различных конфигурациях узлов

В таблице 4.3 представлены все необходимые параметры для расчета суммарной латентности (расчет аппаратной задержки однократно невозможен, т.к. зависит от объёма передаваемого сообщения ).

Таблица 4.3 — Параметры для расчета суммарной латентности

|  |  |  |  |
| --- | --- | --- | --- |
| Программная задержка MPI  , мкс | Размер строки кэша  , байт | Тактовая частота процессора  , МГц | Количество тактов до DDR |
| 91,39 | 64 | 832 | 30 |

Итоговое время передачи сообщений между процессами на одном узле кластера рассчитывается следующим образом:

|  |  |
| --- | --- |
|  | (2) |

Можно заметить, что целевая функция немного видоизменена для удобства вычислений. Также требовалось учесть, что в результате эксперимента с помощью бенчмарка Intel MPI1 PingPong пересылка совершается в соответствии с паттерном PingPong. Т.е. процесс с номером 0 посылает процессу с номером 1 сообщение длины байт. Процесс 1, приняв сообщение от процесса 0, посылает ему ответное сообщение той же длины. Поскольку с помощью математической модели, разработанной в пункте 2.4, можно рассчитать время передачи сообщения только в одну сторону, для того чтобы можно было сравнить результаты, некоторые параметры были удвоены. В таблице 4.4 представлены результаты теоретических и экспериментальных исследований.

Таблица 4.4 — Результаты теоретических и экспериментальных исследований на одном узле вычислительного кластера (mips)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| , байт | , МБ/с | , мкс | , мкс | , мкс |
| 0 | 0 | 91,39 | 0 | 91,39 |
| 1 | 0,01 | 92,52 | 0,0005634014423 | 91,39258668 |
| 2 | 0,02 | 92,75 | 0,001126802885 | 91,39517337 |
| 4 | 0,04 | 92,39 | 0,002253605769 | 91,40034673 |
| 8 | 0,09 | 92,06 | 0,004507211538 | 91,41069346 |
| 16 | 0,17 | 93,33 | 0,009014423077 | 91,43138692 |
| 32 | 0,34 | 93,06 | 0,01802884615 | 91,47277385 |
| 64 | 0,68 | 93,49 | 0,03605769231 | 91,55554769 |
| 128 | 1,37 | 93,44 | 0,07211538462 | 91,72109538 |
| 256 | 2,68 | 95,67 | 0,1442307692 | 92,05219077 |
| 512 | 5,2 | 98,54 | 0,2884615385 | 92,71438154 |
| 1024 | 9,81 | 104,38 | 0,5769230769 | 94,03876308 |
| 2048 | 17,8 | 115,05 | 1,153846154 | 96,68752615 |
| 4096 | 28,32 | 144,64 | 2,307692308 | 101,9850523 |
| 8192 | 46,89 | 174,72 | 4,615384615 | 112,5801046 |
| 16384 | 60,9 | 269,04 | 9,230769231 | 133,7702092 |
| 32768 | 72,4 | 452,59 | 18,46153846 | 176,1504185 |
| 65536 | 116,46 | 562,73 | 36,92307692 | 260,9108369 |
| 131072 | 186,54 | 702,64 | 73,84615385 | 430,4316738 |
| 262144 | 269,33 | 973,33 | 147,6923077 | 769,4733477 |
| 524288 | 328,97 | 1593,73 | 295,3846154 | 1447,556695 |
| 1048576 | 372,63 | 2814,02 | 590,7692308 | 2803,723391 |
| 2097152 | 393,96 | 5323,24 | 1181,538462 | 5516,056782 |
| 4194304 | 397,7 | 10546,47 | 2363,076923 | 10940,72356 |

На рисунке 4.7 изображены графики функций времени пересылки сообщений разного объема между ядрами CPU микропроцессора «Robodeus». Красный график построен по значениям, рассчитанным теоретически по вышеприведенной формуле, а синий отражает время, полученное в результате эксперимента с помощью бенчмарка Intel MPI PingPong пакета Intel MPI Benchmarks 2021.3.



Рисунок 4.7 — Время пересылки сообщений на одном узле вычислительного кластера (mips)

Аналогичным образом следует подготовить параметры для расчета времени пересылки сообщений на узле, с отличной от предыдущей архитектуры, а именно Intel Core i7-9700KF. Часть характеристик микропроцессора можно узнать с помощью команды терминала lscpu (рис. 4.8).

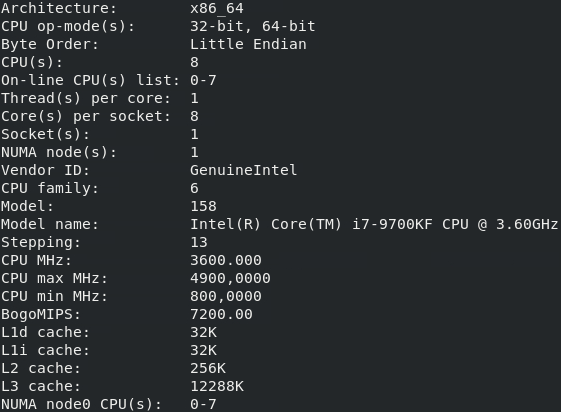


Рисунок 4.8 — Характеристики Intel Core i7-9700KF

Остальная информация об аппаратных составляющих, как правило, указывается в официальный документации. Явно указаны пропускные способности CPU () и DDR (). Для всех уровней кэш-памяти приводится скорость, которую необходимо перевести из Б/такт в ГБ/с (табл. 4.5). Это делается с помощью умножения на актуальное значение тактовой частоты процессора .

Таблица 4.5 — Пропускная способность уровней кэш-памяти

|  |  |  |
| --- | --- | --- |
|  | Скорость  , Б/такт | Пропускная способность  , ГБ/с |
| L1 (512 КиБ) | 64 | 230,4 |
| L2 (2 Мб) | 64 | 230,4 |
| L3 (12 Мб) | 32 | 115,2 |

Скорость матрицы коммутации указана в ГП/с, что означает количество миллиардов пересылок в секунду. Одна передача содержит 16 бит (для процессоров Intel), следовательно, поскольку для процессора указана скорость в 8 ГП/с, то теоретическая суммарная пропускная способность одного соединения рассчитывается так: . Следует учесть возможность передачи в обе стороны (значение удваивается), а также того, что может быть несколько соединений (разрядность шины ):

|  |  |
| --- | --- |
|  | (3) |

В таблице 4.6 представлены все недостающие параметры для расчета суммарной латентности .

Таблица 4.6 — Параметры для расчета суммарной латентности

|  |  |  |  |
| --- | --- | --- | --- |
| Программная задержка MPI  , мкс | Размер строки кэша  , байт | Тактовая частота процессора  , МГц | Количество тактов до DDR |
| 5,18 | 64 | 3600 | 10 |

Адаптированная под пересылки в обе стороны функция подходит и для расчетов теоретического времени передачи сообщений на 64-разрядном микропроцессоре Intel Core i7-9700KF с архитектурой x86. Результаты проведенных расчетов и экспериментальных исследований приведены в таблице 4.7.

Таблица 4.7 — Результаты теоретических и экспериментальных исследований на одном узле вычислительного кластера (x86)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| , байт | , МБ/с | , мкс | , мкс | , мкс |
| 0 | 0 | 5,18 | 0 | 5,18 |
| 1 | 0,19 | 5,27 | 0,0000004340277778 | 5,180186063 |
| 2 | 0,38 | 5,32 | 0,0000008680555556 | 5,180372126 |
| 4 | 0,76 | 6,99 | 0,000001736111111 | 5,180744252 |
| 8 | 1,14 | 7,03 | 0,000003472222222 | 5,181488504 |
| 16 | 2,28 | 7 | 0,000006944444444 | 5,182977009 |
| 32 | 4,57 | 7 | 0,00001388888889 | 5,185954017 |
| 64 | 9,14 | 7,04 | 0,00002777777778 | 5,191908034 |
| 128 | 18,19 | 5,51 | 0,00005555555556 | 5,203816068 |
| 256 | 46,47 | 5,56 | 0,0001111111111 | 5,227632137 |
| 512 | 92,16 | 5,67 | 0,0002222222222 | 5,275264274 |
| 1024 | 180,51 | 5,67 | 0,0004444444444 | 5,370528547 |
| 2048 | 263,9 | 7,76 | 0,0008888888889 | 5,561057094 |
| 4096 | 503,78 | 8,13 | 0,001777777778 | 5,942114188 |
| 8192 | 917,25 | 8,93 | 0,003555555556 | 6,704228376 |
| 16384 | 1557,5 | 10,52 | 0,007111111111 | 8,228456752 |
| 32768 | 2351,5 | 13,93 | 0,01422222222 | 11,2769135 |
| 65536 | 2030,2 | 32,28 | 0,02844444444 | 17,37382701 |
| 131072 | 2682,3 | 48,87 | 0,05688888889 | 29,56765402 |
| 262144 | 4005,37 | 65,45 | 0,1137777778 | 53,95530803 |
| 524288 | 5770,22 | 90,86 | 0,2275555556 | 102,7306161 |
| 1048576 | 7364,62 | 142,38 | 0,4551111111 | 200,2812321 |
| 2097152 | 7528,92 | 278,55 | 0,9102222222 | 395,3824643 |
| 4194304 | 6721,02 | 624,06 | 1,820444444 | 785,5849285 |

На графике (рис. 4.9) можно наглядно увидеть, что функция, рассчитанная теоретически, в достаточной мере близка к экспериментальной. Также следует заметить, что значения, полученные для узла с микропроцессором «Robodeus», почти на порядок отличаются от текущих. Это связано с более высоким значением тактовой частоты, а, следовательно, и пропускных способностей и более низкой аппаратной задержкой на узле с условной x86 архитектурой.

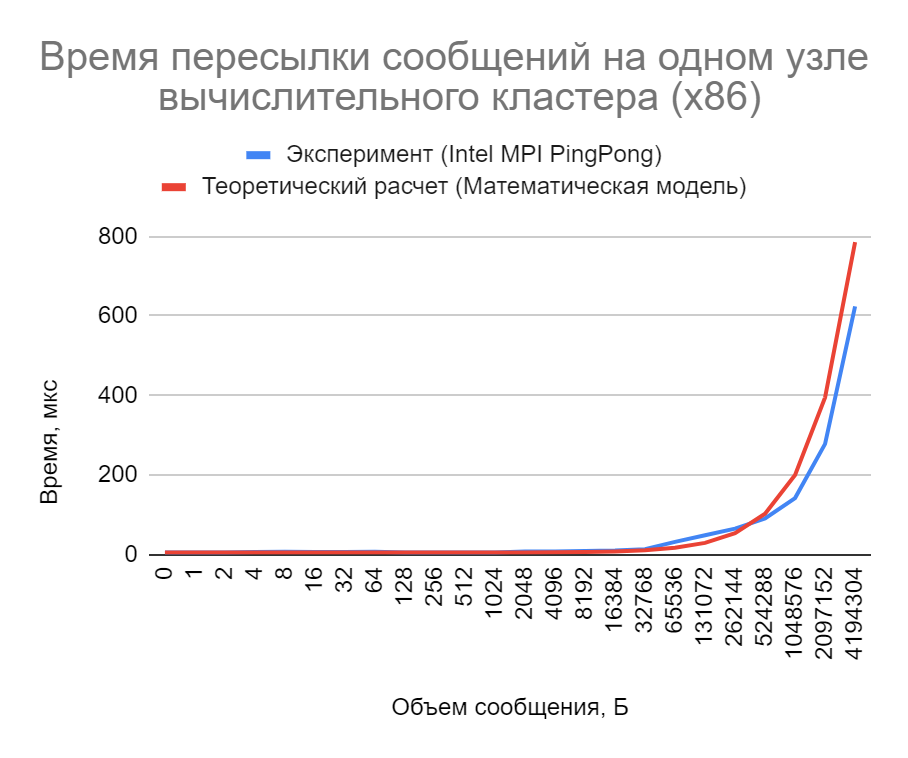


Рисунок 4.9 — Время пересылки сообщений на одном узле вычислительного кластера (x86)

Для теоретического расчета времени пересылки сообщений между двумя неоднородными узлами кластера можно использовать целевую функцию . Почти все необходимые параметры для неоднородных узлов «RoboDeus» (1892ВМ248) и Intel Core i7-9700KF уже рассчитаны, поэтому требуется уточнить лишь:

* сетевую задержку ;
* пропускную способность сетевой интерфейсной карты .

В случае передачи между двумя узлами кластерной системы сетевая задержка (вместе с ) замеряется как латентность MPI с помощью специального теста скорости межпроцессорных обменов (MPI Transfer Test). Пропускная способность сетевых карт, как правило, усредняется по всем задействованным в передаче узлам. В рассматриваемой конфигурации Таким образом, итоговое время передачи сообщений между процессами на неоднородных узлах кластера рассчитывается следующим образом:

|  |  |
| --- | --- |
| где | (4)  (5) |

В таблице 4.8 представлены результаты измерения времени во время эксперимента, а также расчеты времени на одном узле кластера () и суммарное теоретическое время передачи на двух узлах .

Таблица 4.8 — Результаты теоретических и экспериментальных исследований на одном узле вычислительного кластера (x86-mips)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| , байт | , МБ/с | , мкс | , мкс | , мкс | , мкс |
| 0 | 0 | 5,84 | 0 | 0 | 5,84 |
| 1 | 0,2 | 5,05 | 0,00008431089744 | 0,00004651575855 | 5,840150827 |
| 2 | 0,4 | 4,99 | 0,0001686217949 | 0,00009303151709 | 5,840301653 |
| 4 | 0,8 | 5,03 | 0,0003372435897 | 0,0001860630342 | 5,840603307 |
| 8 | 1,6 | 5 | 0,0006744871795 | 0,0003721260684 | 5,841206613 |
| 16 | 3,2 | 5 | 0,001348974359 | 0,0007442521368 | 5,842413226 |
| 32 | 6,41 | 4,99 | 0,002697948718 | 0,001488504274 | 5,844826453 |
| 64 | 12,81 | 5 | 0,005395897436 | 0,002977008547 | 5,849652906 |
| 128 | 25,35 | 5,05 | 0,01079179487 | 0,005954017094 | 5,859305812 |
| 256 | 50,16 | 5,1 | 0,02158358974 | 0,01190803419 | 5,878611624 |
| 512 | 99,06 | 5,17 | 0,04316717949 | 0,02381606838 | 5,917223248 |
| 1024 | 192,89 | 5,31 | 0,08633435897 | 0,04763213675 | 5,994446496 |
| 2048 | 355,19 | 5,77 | 0,1726687179 | 0,0952642735 | 6,148892991 |
| 4096 | 631,49 | 6,49 | 0,3453374359 | 0,190528547 | 6,457785983 |
| 8192 | 1157,01 | 7,08 | 0,6906748718 | 0,381057094 | 7,075571966 |
| 16384 | 1898,58 | 8,63 | 1,381349744 | 0,762114188 | 8,311143932 |
| 32768 | 2573,99 | 12,73 | 2,762699487 | 1,524228376 | 10,78228786 |
| 65536 | 2261,42 | 28,98 | 5,525398974 | 3,048456752 | 15,72457573 |
| 131072 | 3574,34 | 36,67 | 11,05079795 | 6,096913504 | 25,60915145 |
| 262144 | 5283,81 | 49,61 | 22,1015959 | 12,19382701 | 45,37830291 |
| 524288 | 7184,47 | 72,98 | 44,20319179 | 24,38765402 | 84,91660581 |
| 1048576 | 8553,62 | 122,59 | 88,40638359 | 48,77530803 | 163,9932116 |
| 2097152 | 7580,98 | 276,63 | 176,8127672 | 97,55061607 | 322,1464232 |
| 4194304 | 6809,36 | 615,96 | 353,6255344 | 195,1012321 | 638,4528465 |

Итоговые графики, отражающие экспериментальное и теоретическое время передачи сообщений, в случае трех конфигураций (mips, x86, x86-mips) представлены на рисунке 4.10. Из графиков видно, что данные, предсказанные с помощью построенной математической модели, почти полностью совпадают с результатами эксперимента.



Рисунок 4.10 — Время пересылки сообщений на узлах вычислительного кластера

Для верификации результатов исследований используется одна из универсальных мер близости многомерных объектов одинаковой размерности, а именно Евклидово расстояние :

|  |  |
| --- | --- |
|  | (6) |

где - полученный в результате эксперимента вектор времени пересылки сообщений объема ,

- вычисленный по математической модели вектор времени пересылки сообщений объема ,

а – вес измерения времени при заданном объеме .

Веса выбираются в зависимости от того, какие измерения важнее для конкретного случая оценки: либо более важным считается время при пересылке малых сообщений, либо при отправлении большого объема данных, которые определенно не поместятся в L3. Результаты расчета Евклидова расстояния для случаев передачи сообщений разного объема внутри одного узла (robodeus и x86) и для случая пересылки между гетерогенными узлами (robodeus<=>x86) представлены в таблице 4.9.

Таблица 4.9 — Расчеты метрик оценки достоверности

|  |  |  |  |
| --- | --- | --- | --- |
| Показатель | «RoboDeus» (1892ВМ248) | Intel Core i7-9700KF | «RoboDeus» ⬄  Intel Core i7 |
|  | 722,32 мкс | 30,77 мкс | 69,25 мкс |
|  | 0,07 | 0,05 | 0,11 |
| Средняя абсолютная погрешность | 86,97 мкс | 17,34 мкс | 6,73 мкс |
| Средняя относительная погрешность | 6,05 % | 6,58 % | 7,16 % |

Приведено отношение получившегося расстояния к максимальному экспериментальному измерению, а также получены средние значения абсолютных и относительных погрешностей.

Средняя абсолютная погрешность:

|  |  |
| --- | --- |
|  | (7) |

Средняя относительная погрешность:

|  |  |
| --- | --- |
|  | (8) |

Практическое применение методики ускорения выполнения MPI-приложений на кластере гетерогенного типа

Применение методики ускорения выполнения MPI-приложений на кластере гетерогенного типа позволяет повысить быстродействие процесса передачи сообщений между неоднородными узлами кластера, путем минимизации аппаратной и программной составляющей латентности ( и соответственно), а сетевая задержка снижается путем применения методики ранжирования стандартных интерфейсов межузловых коммуникаций. Для этого на практике предпринимались шаги, упомянутые в разработанных методиках.

Во-первых, реализована частичная поддержка протокола когерентности, поскольку несоответствие данных между кэшами может привести к ошибкам и непредсказуемому поведению системы. Отсутствие когерентности ранее не позволяло свободно использовать общую область памяти для передачи данных, например, между DSP и CPU, ввиду необходимости контролировать работу кэша со стороны пользователя. Вычисления без когерентности, но с кэшированием корректны только при условии, что несколько ядер не пытаются в один момент времени записывать одну и ту же строку кэша, что накладывало существенные ограничения, в том числе понижающие эффективность и скорость работы программы.

С помощью бенчмарка RAMspeed (специальная утилита для измерения производительности оперативной и памяти быстрого доступа) наглядно показана разница при обращении на разные уровни памяти микропроцессора «RoboDeus» (1892ВМ248). На рисунке 4.11 представлена столбчатая диаграмма, отражающая скорость записи при поддержке кэширования. Тактовая частота внутренней и внешней памяти mem\_clk и частота опорного тактового сигнала ref\_clk (она же частота CPU) указаны на графике.

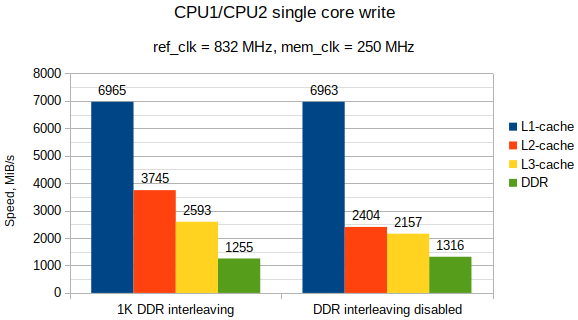


Рисунок 4.11 — Диаграмма скорости записи на разные уровни памяти

На рисунке 4.12 представлен аналогичный график, но уже для проверки скорости чтения с разных уровней памяти. Также была проверена возможность повышения скорости чтения/записи в память за счет равномерного распределения адресов памяти по банкам памяти (DDR interleaving). Замечено, что включение DDR interleaving не повлияло на скорость чтения из DDR, но существенно повысило скорость чтения из L2-кэша (скорость выросла ~ в 2.7 раза).

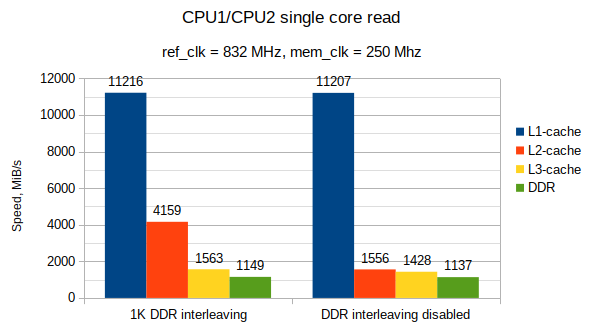


Рисунок 4.12 — Диаграмма скорости чтения с разных уровней памяти

В соответствии со вторым этапом реализована модель виртуального разделения процессов на субпроцессы. Субпроцессы живут и работают только пока существует CPU-процесс, именно он решает в какой момент и в каком количестве создать или завершить субпроцессы. Чтобы это было возможным, CPU-процесс собирается с библиотекой avatar-cpu, а субпроцессы с avatar-dsp. Задачей avatar-dsp является создание у субпроцесса иллюзии того, что он работает в привычной среде, где имеются требуемые библиотеки, функции, системные вызовы. Задачей avatar-cpu является управление субпроцессами и реализация тех функций, которые силами DSP сделать нельзя, например, переключиться на другой субпроцесс или вызвать функцию ОС.

Помимо библиотеки avatar-cpu, также требуется драйвер, который строит для DSP карту памяти идентичную CPU (за исключением системных страниц) и поддерживает её при изменениях (например, при вызове mmap). Это позволяет CPU-процессу выделить память для субпроцесса, создать структуры необходимые для управления и обмена данными, записать код программы или некоторого загрузчика, после чего обратиться к драйверу для физического запуска DSP. Помимо запуска/остановки DSP, другими задачами драйвера являются обработка прерываний/исключений и переключение DSP-ядра на выполнение субпроцессов другого CPU-процесса.

Текущая схема работы с OpenMPI представлена на рис. 4.13. Avatar или SysCall для Elcore-50 предназначен для запуска на DSP более широкого набора приложений, чем позволяют библиотеки компилятора, а именно MPI-программ. Фактически, это avatar-процесс (для CPU), который запускает на DSP приложения, собранные c libavatar. В libavatar добавляются те структуры и функции, которых не хватает в библиотеках компилятора, реально функции выполняются на CPU. Обмен между DSP и CPU выполняется через кэшируемую общую память. Avatar-процесс может запустить только одно приложение, для запуска нескольких требуется запустить несколько avatar-процессов.

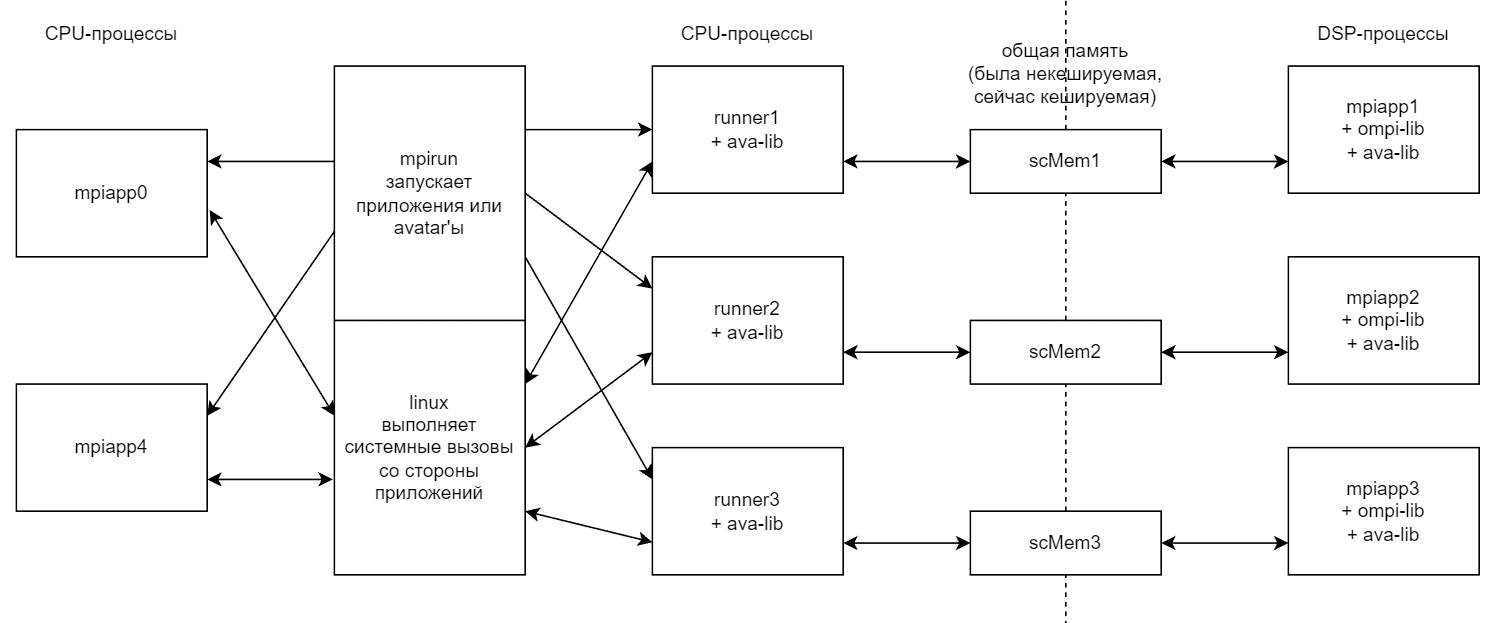


Рисунок 4.13 — Текущая схема работы с OpenMPI

В соответствии с 4 этапом методики для ускорения выполнения MPI-приложений следует повысить тактовую частоту процессора, например с 832 МГц до 1500 МГц. Порядок действий следующий:

1. Проверить текущую частоту процессора с помощью команды:

cat /sys/kernel/debug/clk/cpu\_1\_ref/clk\_rate

1. Склонировать проект \*\*solaris/bootapps\*\*.
2. Именить файл <bootapps-root-path>/common/ucg.c следующим образом:

{UCG\_S, CHANNEL\_CPU\_DAIMYO0\_REF, 3, DIV\_ROUNDUP(UCG\_S\_PLL3\_FREQ\_MHZ, 832), 1},

поменять на

{UCG\_S, CHANNEL\_CPU\_DAIMYO0\_REF, 1, DIV\_ROUNDUP(UCG\_S\_PLL1\_FREQ\_MHZ, 1500), 1},

1. Аналогично для CHANNEL\_CPU\_DAIMYO1\_REF:

{UCG\_S, CHANNEL\_CPU\_DAIMYO1\_REF, 3, DIV\_ROUNDUP(UCG\_S\_PLL3\_FREQ\_MHZ, 832), 1},

поменять на

{UCG\_S, CHANNEL\_CPU\_DAIMYO1\_REF, 1, DIV\_ROUNDUP(UCG\_S\_PLL1\_FREQ\_MHZ, 1500), 1},

1. Переопределить путь к пакету в файле <buildroot-root-path>/buildroot/local.mk,

добавив SOLARIS\_BOOTAPPS\_OVERRIDE\_SRCDIR=<bootapps-root-path>.

Таким образом, получившиеся после пересборки Buildroot образы уже будут содержать сведения о том, на какой частоте работать.

Повышение частоты процессора способствует увеличению быстродействия в целом, так как это позволяет обрабатывать больше информации за меньшее время. На рисунке 4.14 отображена зависимость пропускной способности при передаче сообщений разного объема посредством MPI (бенчмарк Intel MPI1 PingPong). Диаграмма нормирована, что позволяет увидеть соотношение пропускных способностей при различных значениях частот процессора.

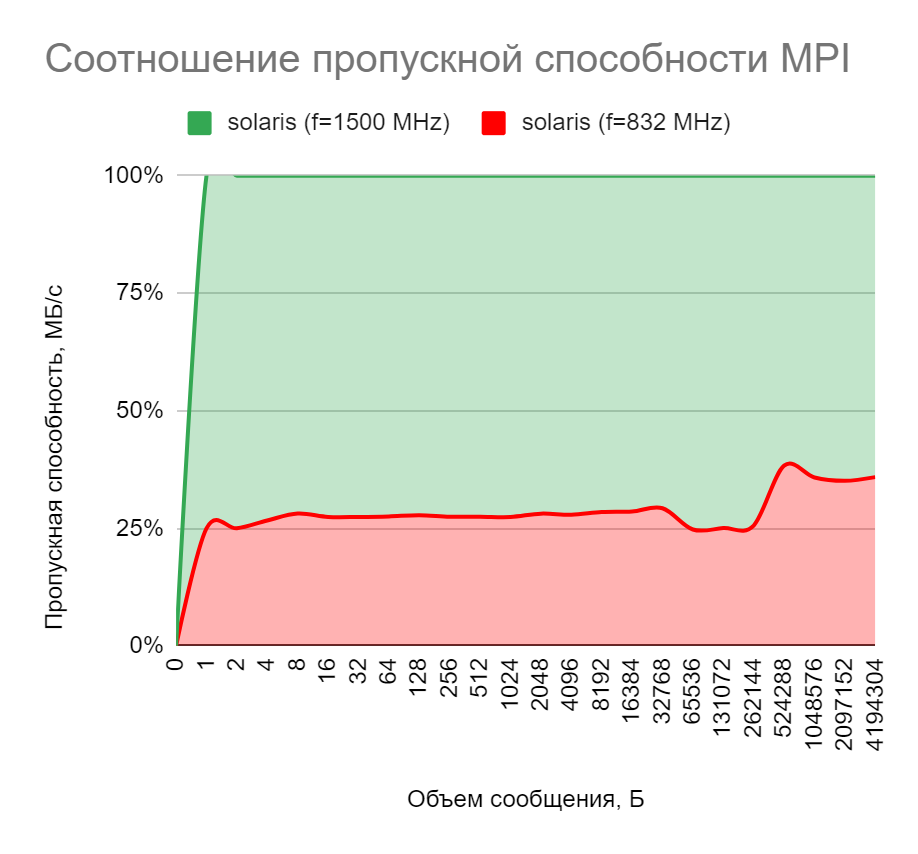


Рисунок 4.14 — Соотношение пропускных способностей при повышении частоты CPU

На рисунке 4.3 можно заметить, что при подключении стенда со стороны отладочного комплекта SSP BuB вместо Ethernet разъема используется разъем USB. В результате применения методики ранжирования стандартных интерфейсов межузловых коммуникаций, принято решение о замене коммуникационных соединений через USB на Ethernet, поскольку максимальная скорость передачи данных по Ethernet-кабелю в несколько раз выше USB (USB 2.0 поддерживает скорость только до 480 Мбит/с). Соединение через USB дополнительно может потребовать целый пакет драйверов и программ, например, до модификации стенда использовался промежуточный сетевой адаптер Ethernet USB 3.0, что отрицательно влияло на латентность сети. На рисунке 4.15 показан график времени передачи сообщений между узлами через USB и Ethernet.

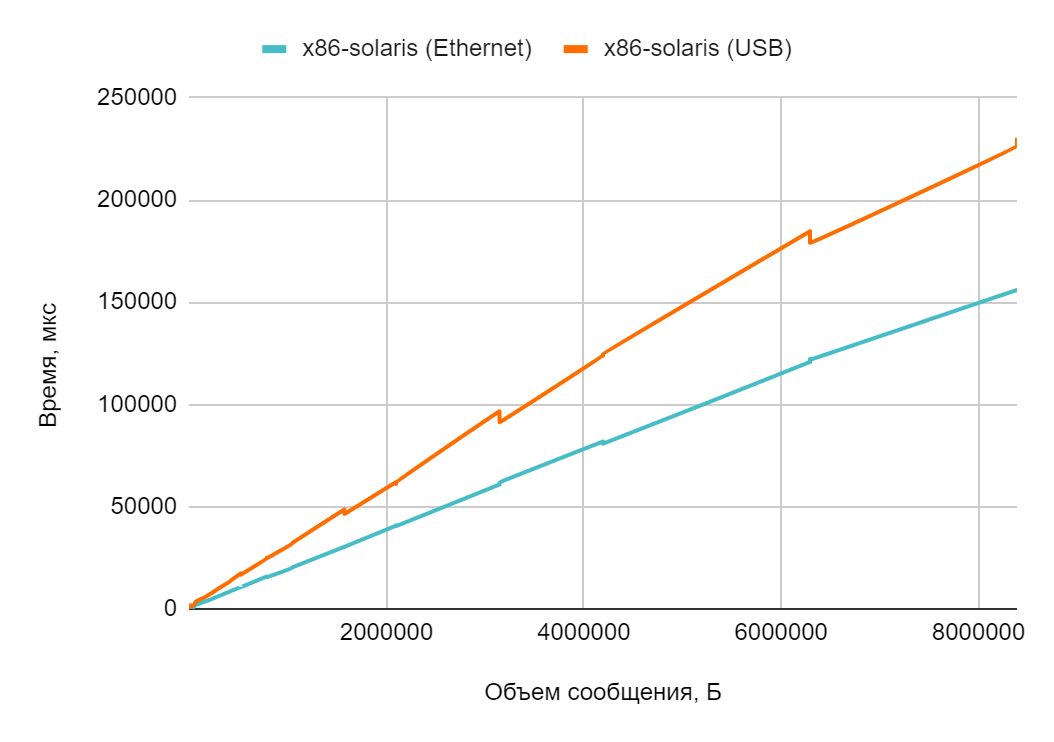


Рисунок 4.15 — Время передачи сообщений между узлами через USB и Ethernet

Согласно 5 этапу методики, использование MPI вместо с технологией OpenSHMEM позволит улучшить производительность программы и сократить время её выполнения. Для этого была выбрана специальная реализация MPI – OpenMPI, поддерживающая обе модели параллельного программирования. Реализация OpenSHMEM включает следующие компоненты:

* atomic – атомарные операции;
* memheap – аллокаторы памяти, поддерживающие модель памяти PGAS;
* scoll – коллективные операции;
* spml – односторонние point-to-point операции;
* sshmem – поддержка разделяемой памяти.

В итоге MPI используется для обмена сообщениями между процессами в распределенной памяти, а OpenSHMEM - для обмена сообщениями между процессами в общей памяти. На рисунке 4.16 отображен график выполнения бенчмарка NetPipe 3.7.2, который выполняет простые ping-pong тесты, пересылая сообщения увеличивающегося размера между двумя процессами. Пропускная способность в случае использования общей памяти через SHMEM заметно выше.

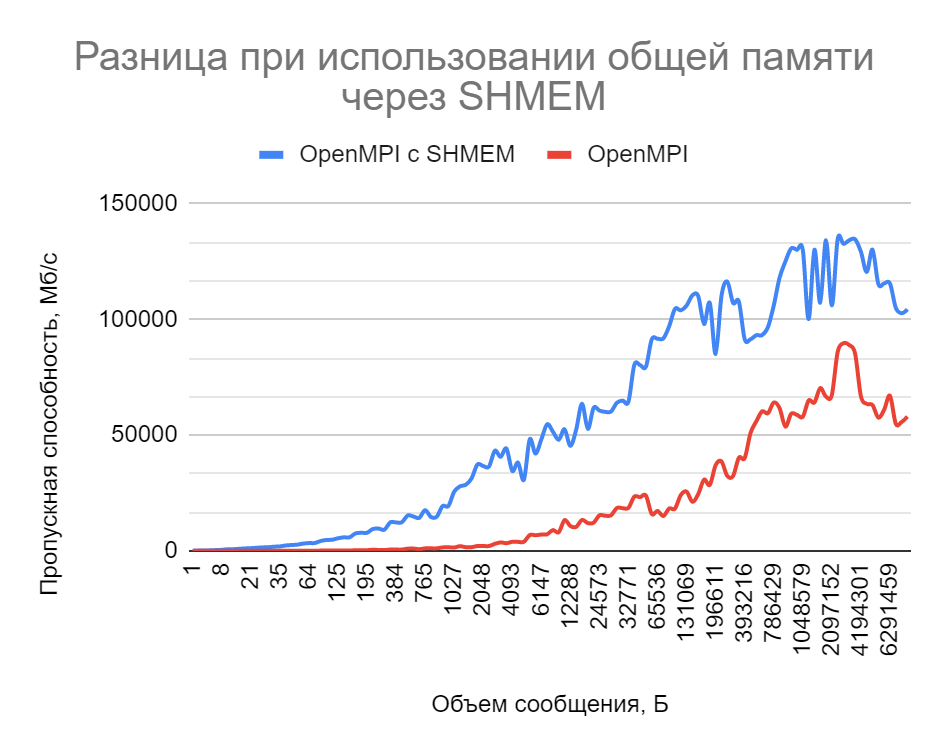


Рисунок 4.16 — Разница при использовании общей памяти через SHMEM

Перспективы практического приложения результатов диссертационного исследования

Производительность параллельных вычислений прежде всего зависит от каналов связи. Фактически, пропускная способность и латентность не только характеризуют кластер, но и ограничивают класс задач, которые могут эффективно решаться на нем. Таким образом, если пропускная способность важна при передаче большого объема информации, то наличие латентности определяет тот факт, что максимальная скорость передачи по сети не может быть достигнута на сообщениях с небольшой длиной. Если задача требует частой передачи небольших пакетов данных, то при высоком значении латентности, большая часть времени будет уходить не на передачу данных между процессами, а на установление связи, в то время как узлы будут простаивать. Для улучшения коммуникационных качеств кластерной системы необходимо как уменьшить величину латентности, так и максимально увеличить значение пропускной способности сети. При этом важны не столько пиковые характеристики, заявленные в стандарте или рассчитанные теоретически, сколько реальные, достигаемые на уровне пользовательских приложений, например, на уровне MPI-приложений. Поэтому эти характеристики считаются основными и при анализе кластерной системы замерять требуется именно их.

Таким образом, можно выделить два ключевых механизма, приводящих к ускорению передачи данных - уменьшение латентности (времени, затрачиваемом программным обеспечением и устройствами сети на подготовку к передаче информации по каналу связи) и увеличение пропускной способности (количества данных, которое может быть передано по шине за единицу времени). Полная латентность складывается из программной (латентность сигнала, на которую влияют используемые библиотеки передачи сообщений, протоколы TCP/IP, а также расходы на копирование из буферов) и аппаратной (задержка сигнала, задержки физического прохождения на портах коммутаторов и сетевых платах) составляющих (рис. 4.17). Чем короче порции передаваемых данных, тем чаще возникает задержка, а чем длиннее передаваемые данные, тем она реже возникает, и скорость передачи будет характеризоваться максимальной пропускной способностью канала.



Рисунок 4.17 — Латентность информации = Задержка сигнала + латентность сигнала

Применение методики ускорения выполнения MPI-приложений на кластере гетерогенного типа позволяет повысить быстродействие процесса передачи сообщений между однородными и неоднородными узлами кластера, путем минимизации аппаратной и программной составляющей латентности ( и соответственно), а сетевая задержка снижается путем применения методики ранжирования стандартных интерфейсов межузловых коммуникаций. С помощью методики расчета и анализа основных характеристик коммуникационной инфраструктуры кластера можно оценить коммуникационную инфраструктуру кластера до и после модификации.

Латентность, например, уменьшается за счет реализации некоторых ресурсоемких операций на аппаратном уровне, организации многоуровневой кэшируемой памяти, использования специальных механизмов доступа к общей памяти, а пропускная способность повышается за счет организации наиболее эффективной для заданного количества узлов сети топологии, выделения дополнительных связей узлам и т.д. Например, если необходимо иметь фиксированную пропускную способность в среднем, то следует использовать дерево коммутации с поразрядной коммутацией на каждом уровне и максимально большими коммутаторами для минимизации задержек. Если некоторым узлам периодически требуется пиковый трафик выше среднего, то им выделяется несколько дополнительным связей. При интеграции узлов с коммутаторами возможно получить большую пиковую пропускную способность, хотя средняя может деградировать.

## Выводы по главе 4

В ходе диссертационного исследования произведены следующие работы:

* описан стенд для проведения экспериментальных исследований;
* определен порядок проведения экспериментальных исследований;
* проведена оценка достоверности теоретических и экспериментальных результатов исследований;
* описаны примеры применения методики ускорения выполнения MPI-приложений на кластере гетерогенного типа;
* описаны перспективы практического применения результатов.

# ЗАКЛЮЧЕНИЕ

В ходе исследования выполнены поставленные задачи:

1. Разработано формализованное представление процесса передачи сообщений между однородными и неоднородными узлами кластера.
2. Разработана методика расчета и анализа основных характеристик коммуникационной инфраструктуры кластера.
3. Разработана методика ранжирования стандартных интерфейсов межузловых коммуникаций.
4. Разработана методика ускорения выполнения MPI-приложений на кластере гетерогенного типа.
5. Разработан алгоритм измерения основных сетевых характеристик кластерной системы.
6. Проведена оценка эффективности полученных результатов.

Также достигнута основная цель работы, а именно удалось повысить быстродействие процесса передачи сообщений между узлами гетерогенной кластерной системы.

# СПИСОК ЛИТЕРАТУРЫ

1. Филин А.В. Суперкомпьютеры и суперкомпьютинг: состояние проблемы параллельных вычислений / Системы и средства информ. – 2007. – № 17. – С. 391 – 467.
2. Сухих А.В., Васяева Н.С. Исследование классификаций кластерных систем / Кибернетика и программирование. – 2016. – № 2. – С. 20 – 27.
3. Максимов Н.В., Партыка Т.Л., Попов И.И. Архитектура ЭВМ и вычислительных систем: учебник / – 5-е изд., перераб. и доп. – М.: ФОРУМ: ИНФРА-М. — 2013. — 512 с.
4. Богданов А.В., Корхов В.В., Мареев В.В., Станкова Е.Н. Архитектуры и топологии многопроцессорных вычислительных систем / – М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», – 2004. — 176 с.
5. Пьянзин К. Недорогие кластеры / Журнал сетевых решений/LAN – 2000. – № 06.
6. Чернышева Л.П., Харитонов Д.П. Эффективное использование гетерогенных вычислительных систем / «Вестник ИГЭУ» – 2012. – Вып. 5. – С. 47 – 51.
7. Кривов М.А., Притула М.Н., Гризан С.А., Иванов П.С. Оптимизация приложений для гетерогенных архитектур. Проблемы и варианты решения / ИТиВС. – 2012. – № 3. – С. 72–81.
8. Summit and Sierra Supercomputers. An Inside Look at the U.S. Department of Energy’s New Pre-Exascale Systems / NVIDIA Corporation. – 2014. – 10 с.
9. Mitsuhisa Sato, Yuetsu Kodama, Miwako Tsuji, Tesuya Odajima. Co-Design and System for the Supercomputer “Fugaku” / Center for Computational Science (R-CCS), RIKEN. – 2019. – 9 с.
10. SX-Aurora TSUBASA Architecture Guide Revision 1.1 / NEC CORPORATION. – 2018. – 509 c.
11. Рейтинг суперкомпьютеров мира Top500 URL: https://top500.org (дата обращения: 06.06.2023).
12. Микросхема RoboDeus // Официальный сайт АО НПЦ «ЭЛВИС» URL: https://elvees.ru/chip/processors-multicore/robodeus (дата обращения: 06.06.2023).
13. Мачарадзе Г.Т., MPI-модель организации параллельных вычислений в кластерах гетерогенного типа / АО НПЦ «ЭЛВИС» . – 2021. – С. 106 – 111.
14. Микросхема интегральная 1892ВМ248. Руководство пользователя / АО НПЦ «ЭЛВИС» . – 2021. – 659 с. /
15. Структурная схема ПО BMC // ПО RoboDeus SHB URL: https://docs.elvees.com/pages/viewpage.action?pageId=106112443 (дата обращения: 06.06.2023).
16. Основы локальных сетей // Национальный Открытый Университет «ИНТУИТ» URL: https://intuit.ru/studies/courses/57/57/info (дата обращения: 06.06.2023).
17. Топольский Н.Г., Мосягин А.Б., Коробков В.В., Блудчий Н.П. Информационные технологии управления в ГПС: Учебное пособие. – М.: Академия ГПС МВД России, 2001. – 188 с.
18. Мысли о развитии DSP // Технологические решения URL: https://docs.elvees.com/pages/viewpage.action?pageId=116230507 (дата обращения: 06.06.2023).
19. Стригунов В.В. Введение в компьютерные сети : учеб. пособие /]. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2016. – 103 с.
20. Блинова И.В., Попов И.Ю. Теория информации. Учебное пособие. – СПб:Университет ИТМО, 2018. – 84 с.
21. Тест производительности межпроцессорных обменов // Лаборатория параллельных информационных технологий НИВЦ МГУ URL: https://parallel.ru/testmpi/transfer.html (дата обращения: 06.06.2023).
22. Математический аппарат для построения компьютерных сетей : Методические указания к практическим занятиям / Набережные Челны, – 2020, – 175 с.
23. NAS Parallel Benchmarks // NASA Advanced Supercomputing (NAS) Division URL: https://www.nas.nasa.gov/software/npb.html (дата обращения: 06.06.2023).
24. Piotr Luszczek, Jack Dongarra, David Koester, Rolf Rabenseifner. Introduction to the HPC Challenge Benchmark Suite / 2004. – 12 с.
25. A Network Protocol Independent Performance Evaluator // NetPIPE URL: http://bitspjoule.org/netpipe/ (дата обращения: 06.06.2023).
26. Pallas MPI Benchmarks - PMB, Part MPI-1 // Intel MPI Benchmarks URL: https://parallel.ru/sites/default/files/ftp/benchmarks/pmb/PMB-MPI1.pdf (дата обращения: 06.06.2023).
27. Чередов А.Д., Мальчуков А.Н. Организация ЭВМ и систем: учебное пособие / А.Д. Чередов, А.Н. Мальчуков; Томский политехнический университет. – 4-е изд., перераб. и доп. – Томск:Изд-во Томского политехнического университета, 2016. – 236 с.
28. М. С. Комар, “О скоростях передачи данных на шинах между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью в современных компьютерах”, Модел. и анализ информ. систем, 24:4 (2017), 434–444
29. Антонов А.А., Ключев А.О., Комар М.С., Кустарев П.В., Кучерявый Е.А., Молчанов Д.А., Петров В.И., Платунов А.Е. Разработка протокола множественного доступа для процессоров с многоуровневым кэшированием // Научно-технический вестник информационных технологий, механики и оптики. 2015. Т. 15. № 3. С. 463-469.
30. MIPS64® I6500 Multiprocessing System Programmer’s Guide, Revision 1.00 // © Wave Computing. – 2017. – 236 с.
31. Andrew Willam Hay. MESIF Cache Coherence Protocol / The University of Auckland. – 2012. – 251 с.
32. Intel® Core™ i7-9700KF Processor // Intel | Data Center Solutions, IoT, and PC Innovation URL: https://www.intel.co.uk/content/www/uk/en/products/sku/190885/intel-core-i79700kf-processor-12m-cache-up-to-4-90-ghz/specifications.html (дата обращения: 06.06.2023).
33. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. –СПб: БХВ – Петербург, 2004.-608 с.: ил.
34. Анализ скоростей передачи данных между процессами в параллельных вычислительных системах // Современные информационные технологии/Вычислительная техника и программирование URL: http://www.rusnauka.com/8\_DNI\_2009/Informatica/43356.doc.htm (дата обращения: 06.06.2023).
35. Ечмаев А.И. ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА НА БАЗЕ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ УЧЕБНЫХ АУДИТОРИЙ // Материалы V Международной студенческой научной конференции «Студенческий научный форум» URL: https://scienceforum.ru/2013/article/2013005909 (дата обращения: 06.06.2023).
36. Справочник Novell Netware 4 / С.Б. Орлов, по заказу ИИЦ "Попурри", 1994 г.
37. Архитектуры и топологии многопроцессорных вычислительных систем // Национальный Открытый Университет «ИНТУИТ» URL: https://intuit.ru/studies/courses/45/45/info (дата обращения: 06.06.2023).
38. Построение сетей на базе коммутаторов и маршрутизаторов // Национальный Открытый Университет «ИНТУИТ» URL: https://intuit.ru/studies/courses/45/45/info (дата обращения: 06.06.2023).
39. Алексеев В.Е, Захарова Д.В. ТЕОРИЯ ГРАФОВ: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. –119 с.
40. Пластун И.Л. Сети ЭВМ / Учеб. пособие. Саратов: Сарат. гос. техн. ун-т, 2005. 75с.
41. Шихалёв А.М. Многокритериальные задачи принятия решений в управлении качеством: учеб. пособие / А.М. Шихалёв, И.И. Хафизов. – Казань: Издательство Казанского университета, 2020. – 92 с.
42. Лотов А.В., Поспелова И.И. Многокритериальные задачи принятия решений: Учебное пособие. – М.: МАКС Пресс, 2008. – 197 с.
43. Саата Т. Принятие решений. Метод анализа иерархий / Перевод с английского Р. Г. Вачнадзе. – Москва «Радио и связь». – 1993. – 278 с.
44. Мачарадзе Г.Т. MPI-модель организации параллельных вычислений в кластерах гетерогенного типа / АО НПЦ «ЭЛВИС», г. Москва. – 2021. – С. 106 – 111.
45. Имитатор MPI для Elcore50 // HPC and HPDA URL: https://docs.elvees.com/pages/viewpage.action?pageId=55315929 (дата обращения: 06.06.2023).
46. Субпроцессы и общая память // HPC and HPDA URL: https://docs.elvees.com/pages/viewpage.action?pageId=58459280 (дата обращения: 06.06.2023).
47. Avatar или SysCall для Elcore-50 // Multicore MPI URL: https://docs.elvees.com/pages/viewpage.action?pageId=106112312 (дата обращения: 06.06.2023).
48. Производительность CPU1/CPU2 // ПО Solaris URL: https://docs.elvees.com/pages/viewpage.action?pageId=75628851 (дата обращения: 06.06.2023).
49. OpenSHMEM(3) man page (version 3.0.6) // Open MPI: Open Source High Performance Computing URL: https://www.open-mpi.org (дата обращения: 06.06.2023).
50. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI: Пособие / Мн.: БГУ, 2002. –323 с.
51. The Buildroot user manual // Buildroot URL: https://buildroot.org (дата обращения: 06.06.2023).
52. Buildroot how-to // Статьи how-to URL: https://docs.elvees.com/display/NTO7/Buildroot+how-to (дата обращения: 06.06.2023).