МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: МОСТ**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

**Тема:**

**«Оптимальные коммуникационные взаимодействия в высокопроизводительных вычислениях»**

Допущена к защите Выполнил:

студент группы 381706-1м

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гладков Дмитрий Олегович

ученая. степень, ученое звание, ф.и.о.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись подпись

Научный руководитель:

к.т.н. Сысоев Александр Владимирович

ученая степень, ученое звание, ф.и.о.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Рецензент:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Нижний Новгород  
2018

Содержание

[Аннотация 3](#_Toc515738360)

[Введение 5](#_Toc515738361)

[Цель работы 10](#_Toc515738362)

[Обзор предметной области 12](#_Toc515738363)

[InfiniBand Architecture 15](#_Toc515738364)

[Элементы InfiniBand Architecture 16](#_Toc515738365)

[Уровни InfiniBand Architecture 19](#_Toc515738366)

[Коммуникационные модели InfiniBand Architecture 23](#_Toc515738367)

[Intel® Omni-Path Architecture 25](#_Toc515738368)

[Элементы Intel® Omni-Path Architecture 26](#_Toc515738369)

[Уровни Intel® Omni-Path Architecture 28](#_Toc515738370)

[Высокопроизводительные коммуникационные интерфейсы общего назначения 41](#_Toc515738371)

[OpenFabrics Interfaces 41](#_Toc515738372)

[Архитектура 41](#_Toc515738373)

[Объектная модель 45](#_Toc515738374)

[Unified Communication X 50](#_Toc515738375)

[Архитектура 51](#_Toc515738376)

[Заключение 56](#_Toc515738377)

[Литература 57](#_Toc515738378)

Аннотация

За последние десятилетия были разработаны и представлены несколько высокоскоростных сетей, каждая из которых отличается от других показателями производительности, коммуникационными особенностями и возможностями. Однако инициативы по разработке новых сетевых решений сопровождаются увеличением расхождений в коммуникационных интерфейсах или “языках”, используемых каждой сетью. Соответственно, переносимость приложений на различных коммуникационных интерфейсах стала темой обширных исследований. В качестве основного средства для достижения переносимости приложений широко использовались такие модели программирования, как Sockets, Shared Memory, Message Passing Interface (MPI), Symmetric Hierarchical MEMory (SHMEM), Global Arrays (GA), Charm++ и другие.

В данной работе исследуются различные подходы к созданию унифицированного программного интерфейса доступа к сети, которая может быть использована различными параллельными программными моделями, такими как MPI, SHMEM и другими, в различных высокоскоростных сетевых средах (например, InfiniBand, Omni-Path Architecture, Ethernet, а также решения, основанные на Ethernet – iWARP, RoCE). В частности, исследования в данной работе посвящены:

1. анализу существующих программных моделей для параллельного программирования и их требований к коммуникационным интерфейсам;
2. рассмотрению существующих и широкоиспользуемых программных интерфейсов для работы с высокоскоростными сетями и возможности построения единой программной модели, способной обеспечить переносимость приложений;
3. анализу ограничений перспективных коммуникационных интерфейсов для мультисетевого программирования, рассмотрению и разработке возможностей для обеспечения наибольшей эффективности сетевого стека для параллельных программных моделей.

Введение

За последние несколько десятилетий достигнуты большие успехи в области компьютерных технологий. Компьютеры проникли во все области деятельности человека, будь то промышленность или исследования, и оказывают влияние на каждый аспект человеческой жизни. В начале 70-х гг XX века, когда обычные компьютеры набирали популярность среди исследователей, возникла необходимость в более мощных машинах, которые могли бы решить проблемы, слишком сложные для обычных компьютеров. Это осознание привело к развитию суперкомпьютеров – передовых и мощных машин, состоящих из нескольких обрабатывающих подсистем. Cray-1, разработанный Cray Research (ныне, Cray Inc.), был одним из таких мощных суперкомпьютеров. Развитие суперкомпьютеров и компьютерных технологий сопутствовало экспоненциальному росту требований приложений. Высокая стоимость проектирования, разработки и поддержки таких суперкомпьютеров для обеспечения высокой производительности приложений требовала, чтобы исследователи искали альтернативу этим суперкомпьютерам в виде кластерных систем (кластеров).

Кластеры состоят из недорогих коммерческих, готовых к использованию без дополнительных модификаций компьютеров (готовое коммерческое изделие, Commercial off-the-shell – COTS), соединенных между собой посредством сети. Кластеры становятся все более популярными в различных сферах применения, главным образом, из-за высокой производительности и относительно низкой стоимости. Такие системы теперь могут быть спроектированы для достижения различного уровня производительности из-за увеличения производительности процессоров, памяти и сетевых технологий.

Поскольку системы на основе кластеров опираются на возможности вычислительных узлов, взаимодействующих друг с другом по сети, характеристики сети (аппаратное обеспечение, а также связанное с ним программное обеспечение) являются важным компонентом в эффективности и масштабируемости кластеров. На рис. 1 показаны типичные среды, используемые параллельными и распределенными вычислительными приложениями. Среды могут варьироваться от локальных сетей внутри одного кластера до нескольких различных кластеров, подключенных через WAN или высокоскоростную магистральную сеть.

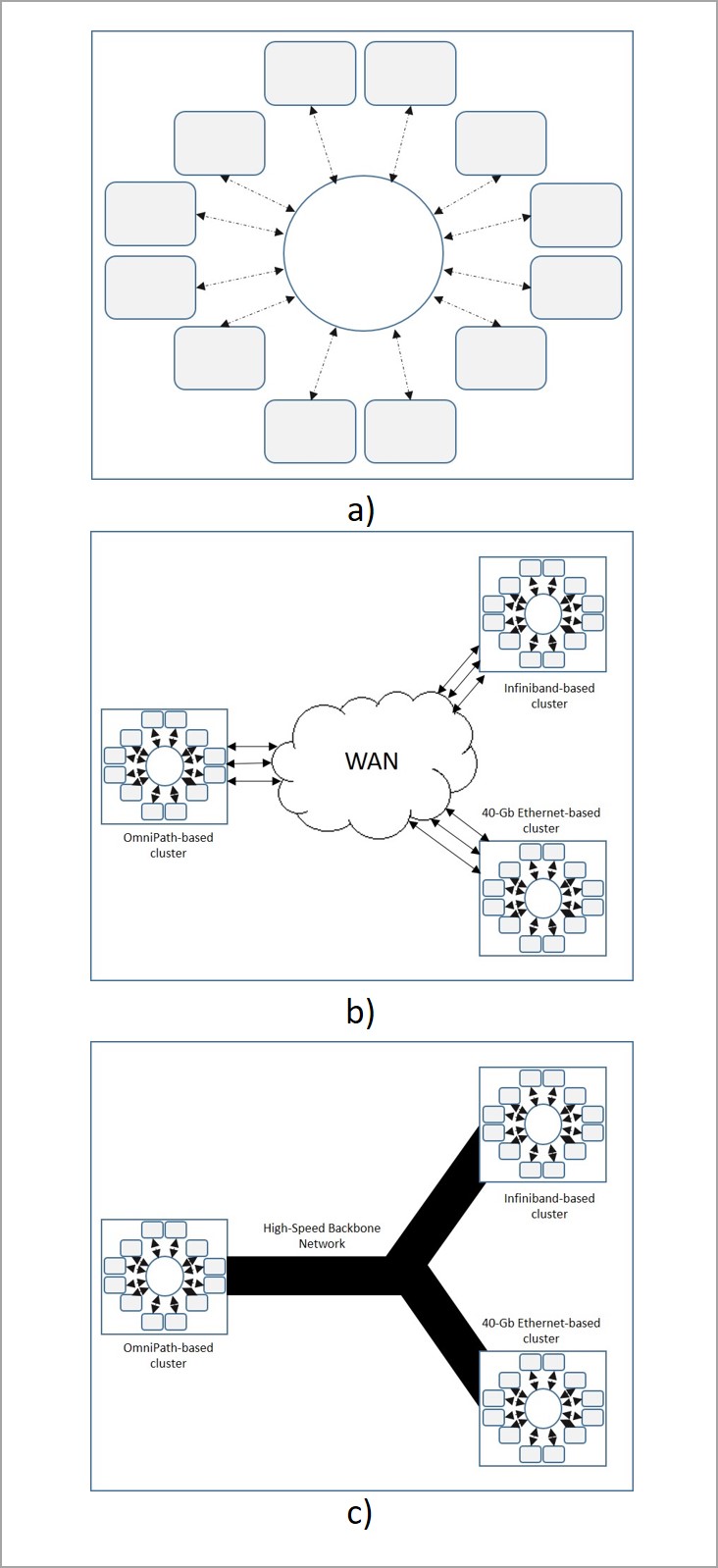


Рис. 1. Типичные коммуникационные среды для распределенных и параллельных приложений:

(a) Связь внутри локальной сети кластера;

(b) Межкластерная связь по WAN;

(c) Межкластерная связь по высокоскоростной магистральной сети.

За последние несколько лет на рынке был представлен ряд высокоскоростных сетей, включая 1/10/40/100-гигабитный Ethernet, InfiniBand, Myrinet, Omni-Path и т.д. С появлением таких сетей коммуникационные издержки в кластерных системах переходят от самой сети к сетевым протоколам на стороне отправителя или получателя. Сетевые протоколы предыдущего поколения, такие как TCP/IP, полагаются на ядро операционной системы для обработки сообщений. Это вызывает множество копирований и переключений контекста ядра на критическом пути обработки сообщения. Таким образом, накладные расходы на связь были высокими. В течение последних нескольких лет исследователи рассматривают альтернативы для увеличения производительности коммуникаций в кластерах посредством разработки протоколов низкого уровня с низкими задержками и высокой пропускной способностью, таких как FM и GM для Myrinet, EMP (Ethernet Message Passing) для Gigabit Ethernet и т.д. Чтобы стандартизировать эти решения, в конце 90-х годов была предложена архитектура виртуального интерфейса (VIA), но VIA не смог добиться успеха. В последние годы в индустрии была стандартизирована InfiniBand Architecture для разработки кластеров нового поколения. Все это приводит к сокращению отставания между функциональными возможностями физической сети и тем, что получают конечные пользователи.

Тем не менее, каждая новая сеть и протоколы пользовательского уровня предоставляют пользователю новый API (Application Programming Interface) или «язык» для взаимодействия с пользователем. Хотя эти новые API могут быть эффективно использованы для разработки новых приложений, они могут оказаться непригодными для уже существующих приложений, которые были разработаны несколько лет назад. Разработчики приложений нацелены на переносимость на различные будущие платформы и сети. Модель параллельного программирования MPI широко признана в качестве приемлемого подхода для достижений такой переносимости и является де-факто стандартом для научных приложений.

Для обеспечения переносимости основные реализации MPI основываются на широко используемом интерфейсе Sockets. Интерфейс Sockets является стандартом для написания сетевых приложений и поддерживается большим количеством современных системам, но не позволяет в полной мере использовать коммуникационные возможности сети. Каждая операция с пользовательскими данными несет значительные накладные расходы для обеспечения надежности, правильной последовательности, которые перекрывают низкие затраты сетевого интерфейса.

Цель работы

Имея ввиду поставленные проблемы интерфейса Sockets для построения высокоскоростных коммуникаций между объектами сети, которые не позволяют раскрыть полностью возможности сетевого оборудования, и проблемы специализированных API, разработанных для конкретных сетей, которые ограничивают переносимость приложений, в данном дипломном проекте предлагается рассмотреть перспективные подходы к созданию унифицированного интерфейса для различных параллельных технологий со следующими возможностями:

1. Позволять приложениям быть переносимыми на разные высокоскоростные сети без каких-либо изменений в программном продукте, имея возможность извлекать наилучшую производительность из сети.
2. Иметь возможность использовать предоставляемые функциональные способности сетей без ущерба производительности и переносимости приложений.
3. Обеспечивать надежность передачи данных, высокую масштабируемость приложений.

Цели данной работы состоят в том, чтобы:

1. Исследовать Open Fabric Interface (OFI), который предоставляет высокопроизводительное, многофункциональное, переносимое промежуточное ПО для существующих параллельных моделей, используемых для высокопроизводительных вычислений, таких как MPI, GlobalArrays, SHMEM и другие.
2. Рассмотреть и предложить возможные решения для текущих недостающих функциональных возможностей OFI, которые мешают обеспечивать низкие задержки, высокую пропускную способность, требуемую масштабируемость на известных сетевых технологиях.
3. Провести исследования, экспериментальные доказательства состоятельности и возможности дальнейшего применения предложенных решений на примере широко используемого программного интерфейса программирования высокопроизводительных приложений MPI.

Обзор предметной области

Последние десятки лет производительность вычислительных узлов увеличивается в два раза каждые 18 месяцев. Как отмечалось ранее, современные высокопроизводительные системы создаются путем объединения вычислительных узлов посредством сетевых интерфейсов в кластерные системы. Кластерные системы зависят от вычислительных мощностей узлов и производительности сети. Таким образом развитие сетевых интерфейсов и ПО, поставляемого вместе с сетевыми интерфейсами, влияет на эффективность кластерных систем.

В данной работе основные исследования связаны с MPI, поскольку большинство параллельных программ, разработанных для кластерных систем, использует именно эту технологию.

Стандарт MPI определяет четыре коммуникационных режима:

* Стандартный (Standard). В этом режиме решение о том, будет ли исходящее сообщение буферизовано или нет, принимает MРI. MРI может буферизовать исходящее сообщение. В таком случае операция посылки может завершиться до того, как будет вызван соответствующий прием. С другой стороны, буферное пространство может отсутствовать или MРI может отказаться от буферизации исходящего сообщения из-за ухудшения характеристик обмена. В этом случае операция отправки не будет завершена, пока данные не будут перемещены в процесс-получатель. Следовательно, в стандартном режиме посылка может стартовать вне зависимости от того, выполнен ли соответствующий прием. Она может быть завершена до окончания приема. Посылка в стандартном режиме является нелокальной операцией: она может зависеть от условий приема.
* Буферизованный (Buffered). Данный режим операции посылки может стартовать вне зависимости от того, инициирован ли соответствующий прием. Однако, в отличие от стандартной посылки, эта операция является локальной и ее завершение не зависит от обстоятельств приема. Следовательно, если посылка выполнена и никакого соответствующего приема не инициировано, тогда MPI обязан буферизовать исходящее сообщение, чтобы позволить завершиться соответствующему вызову операции отправки. Если не имеется достаточного объема буферного пространства, возникнет ошибка. Объем буферного пространства задается пользователем. Для того, чтобы буферизованный режим был эффективным, может потребоваться распределение буферов пользователем.
* Синхронный (Synchronous). Посылка, которая использует синхронный режим, может стартовать вне зависимости от того, был ли начат соответствующий прием. Однако, посылка будет завершена успешно, только если соответствующая операция приема стартовала. Следовательно, завершение синхронной передачи не только указывает, что буфер отправителя может быть повторно использован, но также и отмечает, что получатель достиг определенной точки в своей работе, а именно, что он начал выполнение приема. Если и посылка, и прием являются блокирующими операциями, тогда использование синхронного режима обеспечивает синхронную коммуникационную семантику: посылка не завершается на любой стороне обмена, пока оба процесса не выполнят синхронизацию (так называемый, рандеву) в процессе операции обмена. Выполнение обмена в этом режиме является нелокальным.
* По готовности (Ready). Посылка, которая использует режим обмена по готовности, может быть запущена только тогда, когда прием уже инициирован. В противном случае операция является ошибочной и результат будет неопределенным. На некоторых системах обмен по готовности позволяет устранить необходимость в синхронизации (рандеву), что улучшает характеристики обмена. Завершение операции посылки не зависит от состояния приема и в основном указывает, что буфер посылки может быть повторно использован. Операция посылки, которая использует режим по готовности, имеет ту же семантику, как и стандартная или синхронная передача. Это означает, что отправитель обеспечивает систему дополнительной информацией (а именно, что прием уже инициирован), которая может уменьшить накладные расходы.

Для разработки этих четырех коммуникационных режимов, обычно используются два внутренних протокола, Eager и Rendezvous. Эти протоколы обрабатываются механизмом процессинга (компонента в реализации MPI – Progress engine). Реализация Eager протокола обеспечивает передачу сообщений на удаленную (принимающую) сторону независимо от ее состояния. В Rendezvous протоколе, принимающая и отправляющая стороны участвуют в двухстороннем “рукопожатии” (handshake) через специальные сервисные сообщения перед тем, как данные будут переданы на принимающую сторону. Обычно Eager протокол используется для маленьких сообщений (< 16 килобайт), а Rendezvous – для больших.

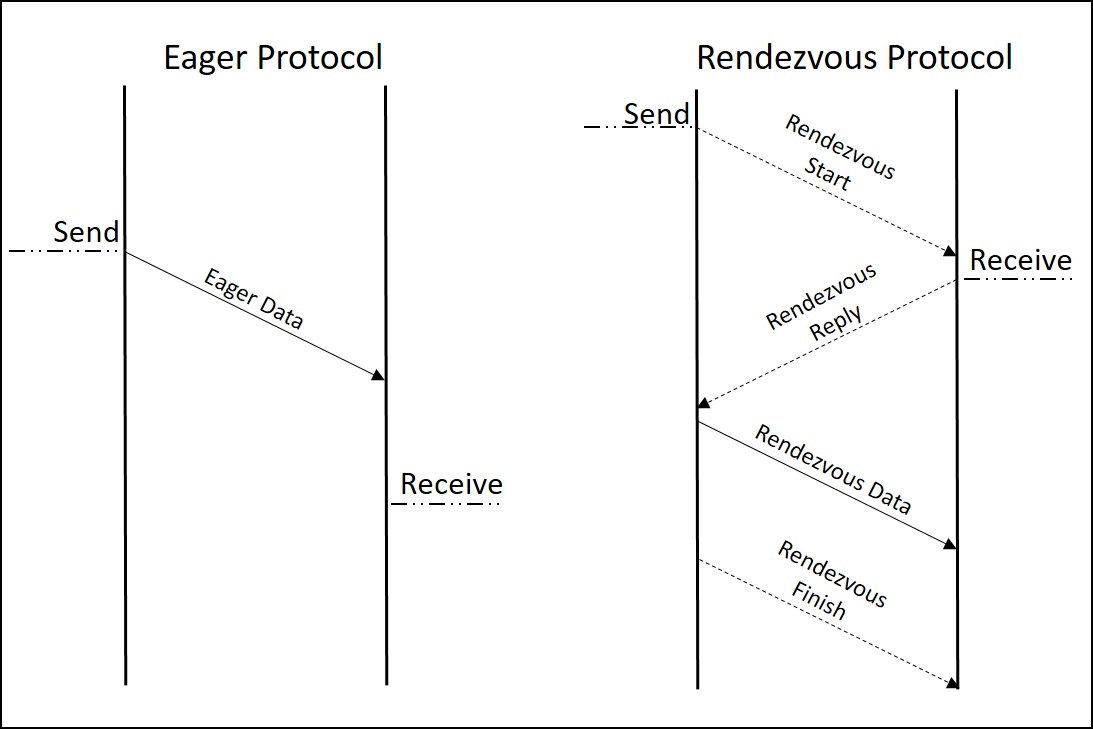


Рис. 2. Типичная реализация Eager и Rendezvous протоколов.

При передаче больших буферов с данными намного оптимальнее не выполнять дополнительных копирований данных, возникающих при использовании TCP/IP стека, который копирует пользовательские данные в ядро операционной системы и обратно.

InfiniBand Architecture

InfiniBand Architecture (IBA) была предложена как следующее поколение для I/O и межпроцессных взаимодействий. InfiniBand [1] становится все более популярным для кластерных вычислений, благодаря открытому стандарту и высокой производительности. До недавнего времени параллельные технологии, такие как MPI, разрабатывались поверх Ethernet, с использованием широко используемого TCP/IP стека. Но с развитием IBA, стали появляться MPI реализации, которые способны работать поверх InfiniBand для обеспечения высокой производительности взаимодействия процессов, развернутых на разных узлах.

InfiniBand определяет сетевую фабрику для взаимодействия вычислительных и I/O узлов. Она предоставляет сетевую инфраструктуру и инфраструктуру управления для обеспечения межпроцессорного взаимодействия и I/O. В InfiniBand сети вычислительные узлы и I/O узлы объединяются в фабрику посредством Channel Adapters (CA) – это конечные узлы в InfiniBand подсети, задача которых состоит в создании и потреблении трафика. Host Channel Adapter (HCA) – это CA, которые располагаются на вычислительных узлах, выполняет роль транспортного уровня и поддерживает Verbs интерфейс.

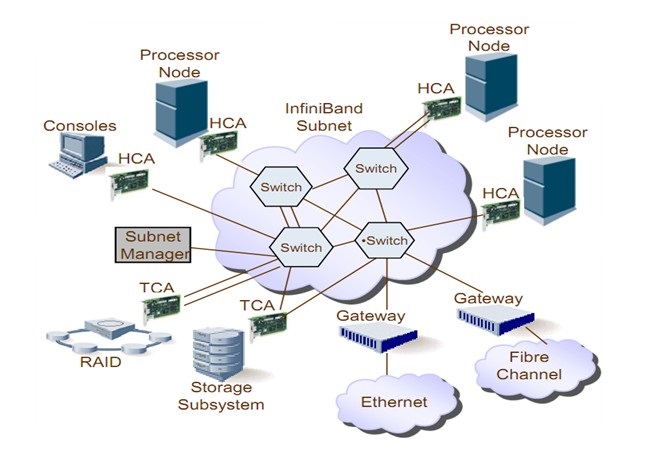


Рис. 3. Пример организации InfiniBand подсети.

Элементы InfiniBand Architecture

InfiniBand Architecture определяет несколько устройств для построения коммуникаций: канальный адаптер (channel adapter), коммутатор (switch), маршрутизатор (router) и менеджер подсети (subnet manager). Внутри подсети должен быть по-крайней мере один канальный адаптер для каждого конечного узла и менеджер подсети, который устанавливает и поддерживает каналы. Все канальные адаптеры и коммутаторы должны содержать Subnet Management Agent (SMA), который требуется для обработки взаимодействий с менеджером подсети.

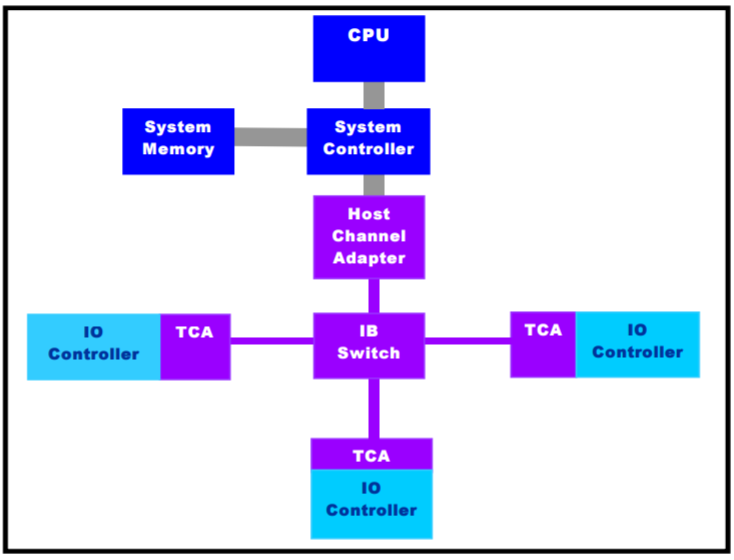


Рис 4. Элементы InfiniBand Architecture.

InfiniBand Architecture определяет следующие элементы:

* Канальные адаптеры (Channel Adapters – CA)

Канальные адаптеры соединяет InfiniBand к другим устройствам. Имеется два типа канальных адаптеров, Host Channel Adapter (HCA) и Target Channel Adapter (TCA).

HCA предоставляет интерфейс конечному устройству и поддерживает все программные Verbs, определенные InfiniBand Architecture. Verbs – это абстрактное представление, которая определяет требуемый интерфейс между ПО, предоставляемым клиенту, и функциональностью HCA. Verbs не определяют API для операционных систем, но определяет операции для производителей ОС для разработки, способного к использованию, API.

TCA предоставляет соединение к I/O устройствам из InfiniBand c некоторым необходимым подмножетсвом функциональностей из HCA для специфичных операций для каждого устройства.

* Коммутаторы (Switch)

Коммутаторы – это фундаментальные компоненты InfiniBand фабрики. Коммутатор содержит более чем один InfiniBand порт и перенаправляет пакеты из одного его порта в другой, основываясь на значении LID, содержащегося внутри LRH. Кроме сервисных пакетов, коммутаторы не потребляют и не генерируют пакетов. Как и CAs, требуется, чтобы SMA был реализован в коммутаторах, чтобы отвечать на пакеты от менеджера подсети (так называемые Subnet Management Packets). Коммутаторы могут быть сконфигурированы, чтобы перенаправлять одноадресные (unicast) пакеты (к одному получателю) или многоадрессные (multicast) пакеты (адресуемые к нескольким устройствам).

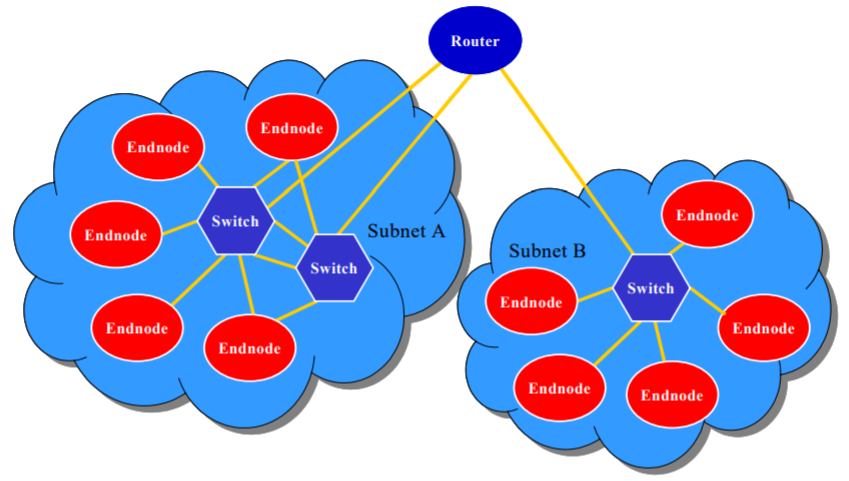


Рис 5. Связь компонент между двумя подсетями и внутри подсетей в InfiniBand Architecture.

* Маршрутизатор (Router)

InfiniBand маршрутизаторы перенаправляют пакеты из одной подсети в другую подсеть без потребления или генерации дополнительных пакетов. В отличие от коммутаторов, маршрутизатор читает Global Route Header (GRH), чтобы перенаправить пакет, основываясь на его IPv6 адресе Сетевого уровня. Маршрутизатор пересобирает каждый пакет с правильным LID следующей подсети.

* Менеджер подсети (Subnet Manager)

Менеджер подсети конфигурирует локальную подсеть и обеспечивает ее последующую работоспособность. В одной подсети обязательно должен быть по крайней мере один менеджер подсети, чтобы управлять всеми коммутаторами и маршрутизаторами и для реконфигурации подсети, когда поднимаются новые каналы или текущие каналы отключаются. Менеджер подсети может быть внутри любого устройства подсети. Менеджер подсети осуществляет связь между устройствами в подсети через SMA, который требуется для каждой компоненты InfiniBand.

Имеется возможность содержать несколько менеджеров подсети в подсети, но только один из них должен быть активным. Неактивный менеджер подсети (Standby SM) копирует информацию о пересылках активного менеджера подсети и проверяет, что активный менеджер подсети является работоспособным. Если активный менеджер подсети становится неработоспособным, резервный менеджер подсети перенимает ответственность обеспечения работоспособности фабрики внутри подсети.

Уровни InfiniBand Architecture

InfiniBand Architecture разбивается на несколько уровней, где каждый уровень работает независимо от других. Как показано на рисунке 4, InfiniBand разбивается на следующие уровни:

1. Физический уровень

InfiniBand является комплексной архитектурой, которая определяет оба электрические и механические характеристики для системы. Они включают кабели и емкости для волоконных и медных сред, разъемы для соединений и характеристики «горячей замены».

InfiniBand определяет три скоростных канала на физическом уровне, 1X, 4X, 12X. Каждый канал – это четырехпроводное последовательное дифференциальное соединение (два провода в каждом направлении), которое предоставляет полнодуплексное соединение со скоростью 2,5 Гбит/с.

1. Канальный уровень

Канальный уровень (совместно с транспортным уровнем) является основой InfiniBand Architecture. Канальный уровень выполняет функции компоновки пакетов, канальные операции типа точа-точка и транспортировку пакетов внутри локальной подсети.

* Пакеты

Имеется два типа пакета, определенные для канального уровня, пакеты управления и данных. Пакеты управления используются для конфигурации коммуникационных каналов и поддержки сервисов канального уровня. Пакеты данных могут транспортировать до 4 Кбайт полезных данных (payload).

* Транспортировка пакетов внутри одной подсети (Switching)
* QoS (Quality of Service)
* Управление потоком на кредитной основе (Credit Based Flow Control)
* Целостность данных

1. Сетевой уровень

Сетевой уровень обрабатывает маршрутизацию пакетов с одной подсети до другой (внутри одной подсети, сетевой уровень не требуется). Отправляемые между подсетями, пакеты содержат Global Route Header (GRH). GRH содержит 128 бит IPv6 адреса отправителя и получателя пакета. Пакета передаются между подсетями с помощью роутера, основываясь на 64 битном значении глобального уникального ID (Global Unique ID - GUID). Роутеры изменяют LRH (Local Route Header) на правильный локальный адрес внутри каждой подсети. Таким образом, последний роутер изменяет LID в LRH на LID удаленного порта. Внутри сетевого уровня, InfiniBand пакеты не требуют информации сетевого уровня и дополнительного заголовка, когда пакеты пересылается внутри одной подсети.

1. Транспортный уровень

Транспортный уровень требуется для доставки пакетов в правильном порядке, разбиение пакета на части, мультиплексирование канала (несколько логических соединений, основанных на одном физическом) и предоставление различных видов транспорта (сервис с надежными соединениями (reliable connection – RC), сервис надежных датаграмм (reliable datagram – RD), сервис с ненадежными соединениями (unreliable connection – UC), сервис ненадежных датаграмм (unreliable datagram – UD), сервис «сырых» датаграмм (raw datagram – RD)). Транспортный уровень обрабатывает сегментацию данных на стороне отправителя и восстановление на стороне получателя. Основываясь на Maximum Transfer Unit (MTU) пути, транспортный уровень разбивает данные на пакеты правильных размеров. Получатель повторно собирает пакет, основываясь на Base Transport Header (BTH), который содержит идентификатор удаленной QP (Queue Pair) и последовательный номер пакета. Получатель подтверждает пакеты и отправитель, получая эти подтверждения, обновляет очередь готовности (Completion Queue - CQ), записывая статус выполненной операции. InfiniBand Architecture предлагает значительное улучшение для транспортного уровня: все функциональности реализованы на аппаратном уровне. InfiniBand определяет несколько транспортных сервисов надежности отправки данных. Таблица 1 описывает каждый из поддерживаемых сервисов.

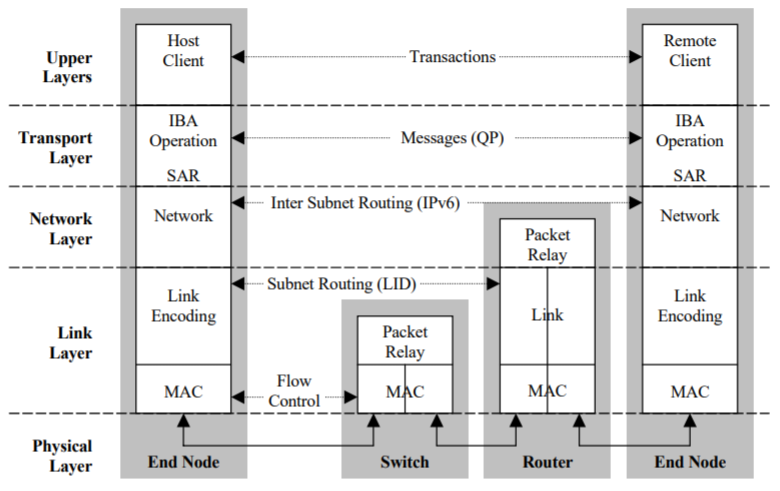


Рис 4. Уровни InfiniBand Architecture.

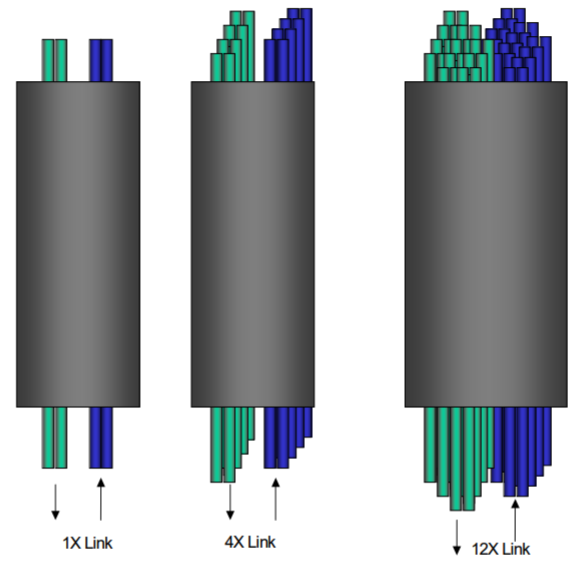


Рис 5. Три скоростных канала, определенных на физическом канале InfiniBand Architecture.

|  |  |
| --- | --- |
| Тип Сервиса | Описание |
| Reliable Connection | Подтверждающий пакеты, ориентированный на соединения |
| Reliable Datagram | Подтверждающий пакеты, мультиплексируемый |
| Unreliable Connection | Неподтверждающий пакеты, ориентированный на соединения |
| Unreliable Datagram | Неподтверждающий пакеты, неориентированный на соединения |
| Raw Datagram | Неподтверждающий пакеты, неориентированный на соединения |

Таблица 1. Типы сервисов транспортного уровня InfiniBand Architecture.

Коммуникационные модели InfiniBand Architecture

Основной целью InfiniBand является уменьшение накладных расходов системы на обработку сетевых данных посредством уменьшения количества копий, ассоциированных с передачей сообщений, и отказа отиспользования ядра операционной системы при отправке сообщения. Это достигается путем предоставления приложениям пользовательского уровня прямого, защищенного доступа к HCA.

InfiniBand Architecture поддерживает два типа коммуникационных семантик: канальную семантику (Send-Receive модель) и семантику доступа к памяти (RDMA модель).

В канальной семантике, каждый запрос на отправку имеет соответствующий запрос на прием на удаленной стороне. Такой подход отражает соответствие «один-к-одному» между каждой операцией отправки и приема. В семантике доступа к памяти используются Remote Direct Memory Access (RDMA) операции. Эти операции неявны для удаленной стороны, так как они не требуют, чтобы дескриптор операции приема был создан. Имеются два типа RDMA операций: RDMA Write и RDMA Read. В случае RDMA Write операций, инициатор напрямую пишет данные в пользовательский буфер, выделенный на удаленном узле. По аналогии, в случае RDMA Read операции, инициатор читает данные из пользовательского буфера, выделенного на удаленном узле.

InfiniBand был разработана с учетом VIA (Virtual Interface Architecture) архитектуры. VIA – это распределенная технология передачи сообщений, которая является аппаратно-независимой и совместима с текущими сетевыми высокоскоростными картами, предоставляет API, который может предоставить высокоскоростную, с низкими задержками связь между коммуницирующими сторонами в кластерных приложениях.

InfiniBand забирает обязанности по контролю траффика из программной части клиента с помощью использования исполняемых очередей. Эти очереди, называемые Work Queues (WQ) создаются клиентом и передаются в InfiniBand для последующего управления. Для каждого коммуникационного канала между устройствами назначается Work Queue Pair (WQP – очередь приема и отправки) для каждой конечной стороны. Клиент добавляет транзакцию в WQ (транзакция называется Work Queue Entry – WQE), которая обрабатывается канальным адаптером из очереди отправки и отправляется на удаленное устройство. Когда удаленное устройство отвечает, канальный адаптер возвращает статус клиенту, через очередь готовности (CQ - Completion Queue).

Клиенты могут добавлять несколько WQEs, а канальные адаптеры буду обрабатывать каждый из этих запросов. Затем канальные адаптеры генерируют CQE (Completion Queue Entry), чтобы предоставить статус для каждой WQE в правильном приоритезированном порядке. Это позволяет клиенту продолжать работу с другими активностями, пока транзакция обрабатывается.

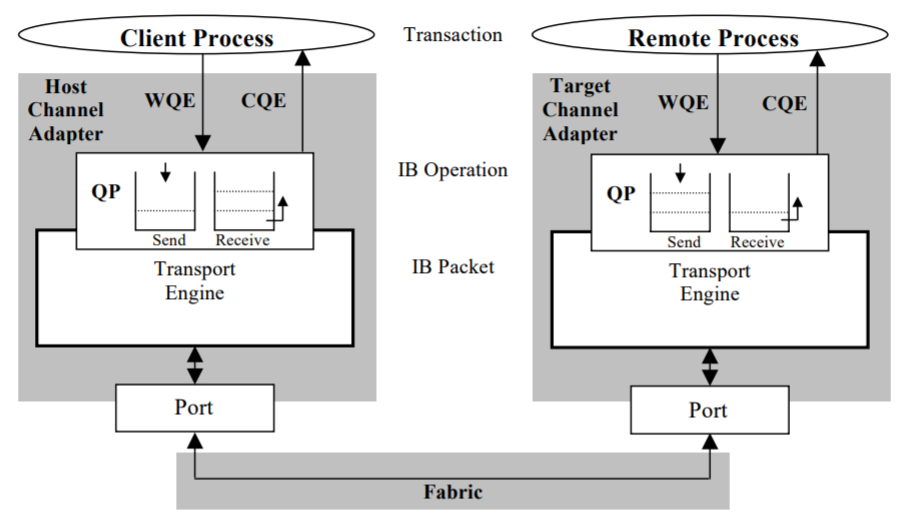


Рис 6. Коммуникационный стек InfiniBand.

Intel® Omni-Path Architecture

Intel® Omni-Path Architecture (OPA) представляет собой новое поколение сетевой фабрики с наследием от Intel® TrueScale продуктовой линейки, основанной на InfiniBand Architecture, и высокоскоростных сетевых карт Cray Aries. Intel OPA разработана для интеграции компонент фабрики с CPU и компонентами памяти, что позволяет обеспечивать низкие задержки и высокую пропускную способность, которые требуются для датацентров следующего поколения.

Intel® OPA определяет 1 и 2 уровни модели OSI, которые обеспечивают сетевое соединение между элементами для энергоэффективных суперкомпьютерных систем (к примеру, основанные на семействе продуктов Intel® Xeon Phi), критически важных корпоративных компьютерных систем (к примеру, основанные на Intel® Xeon™ процессорах) и недорогих серверов для датацентров (к примеру, использующие процессоры Intel® Atom™).

Чтобы обеспечить максимальную масштабируемость систем, как в HPC, так и для датацентров, существенно повышена надежность фабрики путем объединения механизмов повторной отправки на канальном уровне (обычно используемых в HPC фабриках) и end-to-end повторной отправкой (которые используются в традиционных сетях). Сетевая адресация 2 уровня модели OSI расширена для учета систем с более чем 10 миллионов конечных точек соединения (endpoint), что позволяет использовать высокую масштабируемость для датацентров на годы вперед. Чтобы обеспечить поддержку более широкого числа топологий, Intel® OPA предоставляет механизмы для изменения виртуальных полос (virtual lanes – VLs) по мере прохождения через фабрику. В дополнение, высокоприоритетные пакеты способны быть обработаны быстрее, чем пакеты с меньшим приоритетом для обеспечения наиболее предсказуемой производительности системы, в особенности при одновременном запуске нескольких приложений. Также предоставляется механизм разделения фабрики для изоляции трафика между запускаемыми задачами и между пользователями.

Элементы Intel® Omni-Path Architecture

Intel® OPA состоит определяет следующие коммуникационные элементы:

* Host Fabric Interface (HFI)

Каждый узел соединяется с фабрикой через HFI. HFI – это мост между процессором на узле и фабрикой. Минимально, HFI состоит из логики, которая необходима, чтобы реализовать физический и канальный уровни архитектуры фабрики, так что узел может быть присоединен к фабрике и способен обмениваться пакетами с другими серверами или устройствами. HFI может включать специализированную логику для исполнения и ускорения более высокоуровневых протоколов. HFI должен поддерживать логику, необходимую, чтобы отвечать на сообщения от сетевых компонент управления.

* Коммутатор (Switch)

Коммутаторы Intel® OPA являются устройствами 2-го уровня OSI модели и действуют как механизм перенаправления внутри одиночной Intel® OPA фабрики. Intel® OPA отвечают за QoS (Quality of Services) функциональности, такие как адаптивная маршрутизация (adaptive routing) и балансировка нагрузки (load balancing), а также реализуют Intel® OPA функциональности по управлению перегрузками (congestion management). Коммутаторы централизованно управляются ПО менеджера фабрики (Fabric Manager - FM) и каждый коммутатор включает Management Agent (MA) для обработки транзакций управления. Централизованное управление означает, что конфигурация каждого коммутатора настраивается менеджером фабрики, что включает в себя настройку таблиц маршрутизации для поддержки специфичных топологий фабрики, конфигурирование QoS параметров и предоставление альтернативных маршрутов для адаптивной маршрутизации. Все Intel® OPA коммутаторы должны содержать MA для поддержки коммуникаций с Intel® OPA FM.

* Менеджер фабрики (Fabric Manager)

Intel® OPA фабрика централизованно управляется и поддерживает модель избыточности (redundant) для менеджеров фабрики, которые управляют каждым устройством (то есть HFI, коммутаторами) в фабрике через MA, которые ассоциируются с теми устройствами. Основная FM в модели избыточности выбирается во время процесса инициализации фабрики. FM отвечает за:

1. Обнаружение топологии фабрики;
2. Установление идентификаторов фабрики и другие значения, необходимые для функционирования фабрики;
3. Создание и заполнение таблицы маршрутизатора коммутатора;
4. Поддержание базы данных Fabric Management;
5. Мониторинг использование фабрики, производительности и коэффициент ошибок.

Элементы фабрика управляется путем отправки пакетов управления через фабрику. Эти пакеты являются внутриполосными, т.е. отправляются через те же самые физические среды, как и обычные сетевые пакеты, используя специально отведенные буферы на специальной VL (VL15). Эта специальная VL для управления может быть сконфигурирована, чтобы функционировать с включенным или выключенным управлением потока. Без управления потока, пакеты управления не будут обработаны, если ресурсы очереди не доступны на порту. Двухсторонние протоколы, обеспечивающие надежность передачи данных, используются для обнаружения потерянных пакетов.

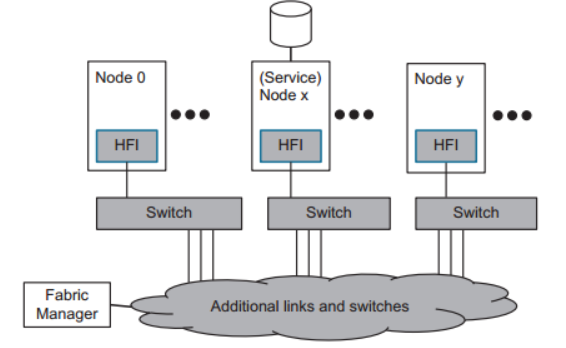


Рис 7. Элементы Intel® Omni-Path Architecture.

Уровни Intel® Omni-Path Architecture

Omni-Path Architecture разбивается на несколько уровней, где каждый уровень работает независимо от других. С сетевой точки зрения Omni-Path достаточно точно вписывается в эталонную модель OSI с одним небольшим нюансом: в описании протокола используется термин «полуторный уровень», отвечающий за надежную доставку объектов второго уровня, контроль потока и соединения. В ведении второго уровня остается адресация, коммутация и выделение ресурсов.

1. Физический уровень (Layer 1)

Использует существующие Ethernet и InfiniBand стандарты физического уровня.

1. Уровень канальной передачи (Layer 1.5)

Уровень канальной передачи (LT, Link Transfer) служит интерфейсов между Физическим и Канальным уровнями. LT уровень сегментирует пакеты фабрики (FP, Fabric Packets) на 64-битные сегменты, к ним присоединяется бит типа (65-ый бит) и получается, так называемый, флит (FLIT, Flow Control Digits). Затем 16 FLITs группируются в пакеты канальной передачи (LTP, Link Transfer Packet) для обеспечения надежности транспортировки 16 FLITs, на которые был разбит FP, и контрольной информации по каналу. Каждый LTP пакет защищен с помощью канального LCRC (Link Cyclic Redundancy Check), и выполняется повторная отправка LTP пакетов, когда происходят ошибки. LTP пакет может содержать FLITs с нескольких FP. Каждый LTP пакет может быть отправлен по одной из четырех полос, которые предоставляются Физическим уровнем.

Контрольные (Control) FLITs используются для управления протоколом повторной отправки. Командные (Command) FLITs используются для возврата кредитов управления потока VL передающей стороне.

FP FLITs и Командные FLITs могут быть отправлены в одном LTP пакете. Контрольные FLITs могут быть отправлены только в специальных нулевых LTP пакетах и не являются частью какого-либо пакета фабрики. Холостые (Idle) FLITs вставляются в постоянно отправляемые LTP пакеты, когда нет, доступных для отправки, FP FLITs.

LTP пакеты содержат 16 FLITs и, ассоциируемые с каждым FLIT, биты типа для отправки по каналу. В дополнении к 16 FLITs каждый LTP имеет два бита кредита канала VL и 14 битов LCRC, который охватывает весь контент всего LTP пакета. Общий размер LTP пакета составляет 128 байт полезной нагрузки (16 FLITs) и дополнительные 4 байта (16 битов типа для каждого FLIT, 14-битный CRC и 2 бита для кредита VL), что обеспечивает эффективность передачи данных, равную 64/66. Имеется два типа LTP пакетов. Надежные LTP пакеты содержат FP FLITs и VL кредиты вернувшихся FLITs и хранятся в буфере повторной отправки в течение периода времени, который достаточно длинный, чтобы гарантировать, что отсутствие запроса повторной отправки означает, что он был успешно принят удаленной стороной. Нулевые LTPs не потребляют буфер повторной отправки и никогда не передаются повторно. Они различаются с помощью Контрольного FLIT, который задает конкретную операцию в протоколе повторной передачи.

Механизмы, реализованные на Уровне 1.5:

1. Intel® Omni-Path Packet Integrity Protection (PIP) – Защита целостности пакетов

Intel® Omni-Path PIP используется для улучшения надежности передачи данных по каналу. Когда возникает ошибка CRC проверки, запрос на повторную отправку, содержащий последовательный номер LTP пакета с ошибкой, отправляется на удаленную сторону, информируя ее о повторной отправке указанного LTP пакета с ошибкой и всех последующих LTP пакетов. Принимающая сторона отбрасывает все LTP пакеты, пока не начнется прием пакетов повторной передачи, который указан в нулевом LTP пакете, который запрашивает повторную передачу

1. Intel® Omni-Path Traffic Flow Optimization (TFO) and Interleave – Оптимизация контроля трафика и чередование

Link Transfer Layer разрешает, чтобы FLITs из различных пакетов на различных VLs чередовались внутри LTP пакета и между LTP пакетами, когда они отправляются по каналу. Это дает более лучшее использование канала, меньшие задержки для высокоприоритетных пакетов. FP, использующий высокоприоритетный VL и поступающий на точку выхода канала, может вытеснить незавершенный FP, использующий менее приоритетный VL, чтобы минимизировать задержку приоритетного FP. Как только FP с более высоким приоритетом будет передан, передача приостановленного пакета возобновляется. FP с низким приоритетом может быть выгружен несколько раз в точке выхода канала. Точка выхода канала контролирует общее время, в течение которого FP с низким приоритетом задерживается путем приоритетного прерывания несколькими высокоприоритетными FPs, и позволяет выполнить FP с низким приоритетом, если превышен лимит, настроенный менеджером фабрики.

1. Virtual Lane Credit Management – Управление кредитам виртуальных каналов

Сторона передачи канала информируется об общем пространстве физического буфера (информируемый размер - в FLITs) на удаленной стороне во время процесса инициализации канала. Сторона приема рассматривает пространство как единый пул для всех VL. Передающая сторона управляет этим пространством на основе VL. В дополнение к одному VL управления поддерживается до 31 FP VL. Для каждого VL поддерживается выделенное пространство, а также общее пространство для всех VL. Соотношение фиксированного к общему пространству и выделенным размером пространства VL может быть динамически изменено менеджером фабрики. Для каждого VL подтверждения возвращаются в единицах по 8 FLITs как только они удаляются из пула буферов для конкретного VL. Эти подтверждения перемещаются по каналу в 2-битном поле VL-кредитования в каждом LTP. 2 бита из 4 последовательных LTP объединяются в 8-битное поле, которое используется для указания VL и номер 8 возвращаемых FLITs. Существуют также дополнительные Command FLITs, которые могут использоваться для возврата значения кредитов VL по каналу. 2-битное поле кредита VL LTP и Command FLITs передаются во время последовательности повторных передач, так что инкрементные значения кредита VL не теряются, когда возникают ошибки канала передачи.

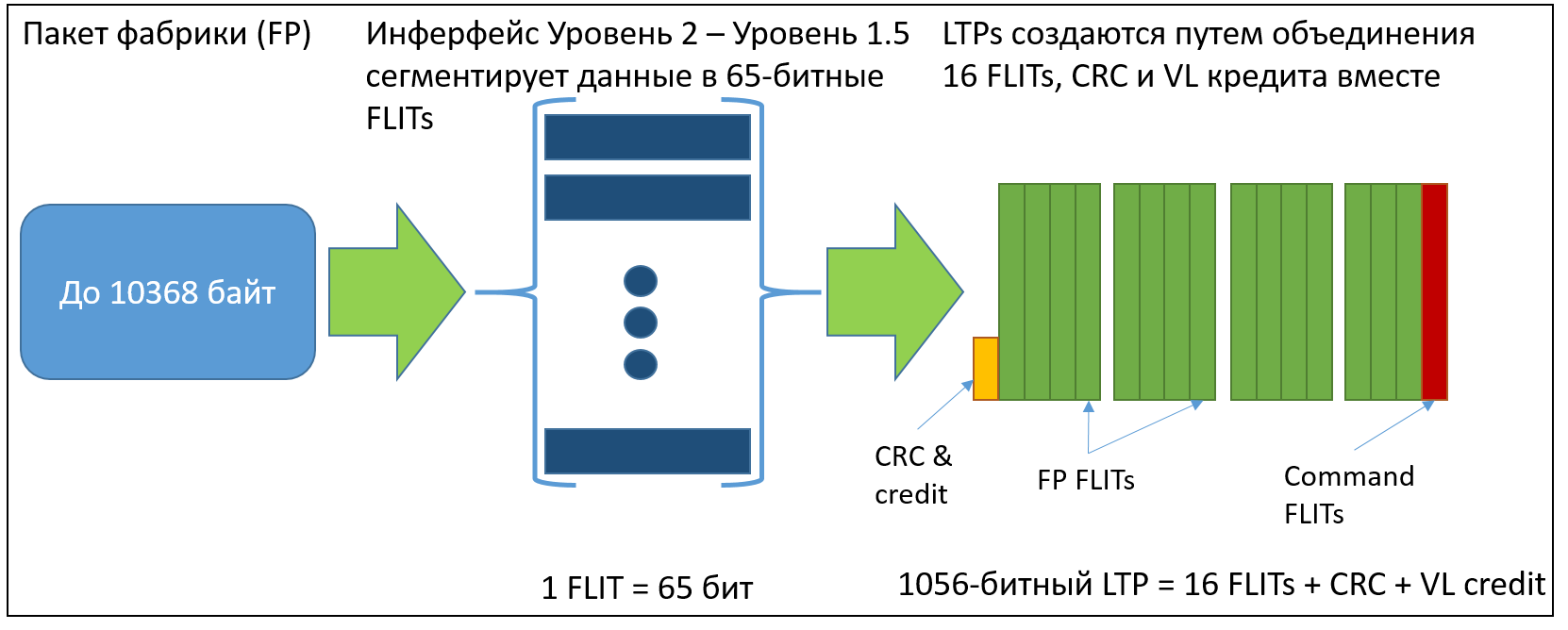


Рис 8. Сегментация FP – FLITs - LTP Intel® Omni-Path Architecture.

1. Канальный уровень (Layer 2)

Intel® OPA Канальный уровень разработан для огромных масштабируемых систем. Пакеты канального уровня используют 24-битную адресацию, а также оптимизированные форматы для небольших систем. До 10 килобайт может быть передано в одном пакете после учета самого большого заголовка транспортного уровня. Service Channels (SCs) и Virtual Lanes (VLs) предоставляют строительные блоки для поддержки широкого класса топологий, а также для реализации Quality of Service (QoS) функциональностей.

Quality of Service (QoS). В рамках Intel® OPA фабрики QoS предоставляет ряд возможностей, в том числе разделение задач/распределение ресурсов; разделение услуг/распределение ресурсов; разделение трафика приложений в рамках данной задачи; протокол (то есть, запрос/ответ); механизм избегания взаимоблокировки; приоритезация трафика и распределение полосы пропускания; оптимизация «дрожания» задержки путем приоритезации трафика.

Intel® OPA предоставляет очень гибкие механизмы для QoS через Виртуальные Фабрики (vFabrics – Virtual Fabrics), Классы Трафика (TCs – Traffic Classes), Уровни Сервиса (SLs – Service Levels) и Виртуальные Полосы (VLs – Virtual Lanes). В сердце QoS лежит SC механизм, который используется для дифференцирования пакетов (FP) в рамках фабрики. Для поддержки широкого спектра топологий фабрики и конфигураций, назначение SC управляется с помощью менеджера фабрики и SC конкретного пакета фабрики может изменится по мере прохождения через фабрику для маршрутизации в зависимости от возникновения взаимоблокировок.

Операции, выполняемые приложениями и системными администраторами, централизованы вокруг vFabrics. vFabric – это пересечение множества портов фабрики и один или более протоколов уровня приложений. Множество QoS функциональностей и политик безопасности устанавливается системным администратором для каждой vFabric. Каждая vFabric ассоциируется с классов трафика для QoS и разделение (Partition) для безопасности.

Классы трафика представляет группу уровней сервиса (SLs), которые будут использоваться данным транспортным уровнем или приложением. Некоторые транспортные уровни могут использовать несколько QoS уровней, чтобы избежать взаимную блокировку (такие как разделение запросов и ответов в определенных протоколах), пока другие могут использовать преимущество QoS уровней для разделения высокоприоритетного управляющего трафика от низкоприоритетного трафика с данными. Транспортные уровни могут ассоциировать классы трафик с единственным уровнем сервиса. Intel® OPA позволяет иметь до 32 классов трафика, но 4-8 является наиболее типичной конфигурацией. Классы трафика реализуются через сквозную (end to end) концепцию уровней сервиса. Классы трафика могут охватывать множество уровней сервиса, но уровень сервиса может быть назначен только к одному классу трафика. Уровни сервиса – это самый низкий уровень QoS концепции, который виден транспортному уровню OPA и приложениям.

Каналы Сервисов (SC, Service Channel), лежащие в основе уровней сервиса, дифференцируют пакеты различных уровней сервиса как только они проходят через фабрику. В некоторых топологиях фабрики, уровни сервиса могут охватывать множество каналов сервиса, но канал сервиса может быть назначен только на один уровень сервиса. Intel® OPA поддерживает 32 канала сервиса, однако, SC15 выделен под нужды внутриполосного менеджмента фабрики.

Для каждой конечной точки соединения (endpoint) HFI уровни сервиса отображаются на каналы сервиса через SL2SC таблицы при передаче (отображение каждого уровня сервиса на канал сервиса) и SC2SL таблицы (отображение входящих каналов сервиса на данный уровень сервиса с возможно несколькими каналами сервиса, отображенными на тот же уровень сервиса). Рисунок 9 показывает пример использования класса трафика, уровня сервиса и канала сервиса. В этом примере две конечные точки HFI соединены через коммутатор, предоставляющий маршрут из 8 сетей перехода (hop) через фабрику. Два класса трафика используются, один – для запрос/ответ протокола (к примеру, PGAS модель), назначенный на TC0, другой – для сетевой протокол хранилища данных, назначенный на TC1. Запрос/ответ протокол на TC0 требует два уровня сервисов (SL0 и SL1), сетевой протокол хранилища данных на TC1 требует только один уровень сервиса (SL2). Каждому уровню сервиса назначается пара каналов сервиса (SC0/SC1, SC2/SC3 и SC4/SC5) для избегания взаимной блокировки в фабрике, которая обычно используется в топологии тор. Когда пакеты переходят через фабрику, канал сервиса может изменять канал за каналом. Однако, уровень сервиса и класс трафика, видные для транспортного уровня и приложений, являются сквозными (end to end).

Внутри определенного канала, каналы сервиса разделяются на виртуальные полосы (Virtual Lanes). Виртуальные полосы предоставляют выделенные буфера приема для входящих пакетов фабрики. Виртуальные полосы также используются для разрешения взаимных блокировок маршрутизации. Intel® OPA поддерживает до 32 виртуальных полос, таким образом действительное количество поддерживаемых виртуальных полос будет зависеть от реализации. Intel® OPA поддерживает отображение каналов сервиса на виртуальные полосы, так что могут поддерживаться гетерогенные конфигурации фабрики. SC15 отображается на VL15 для внутриполосного менеджмента фабрики. Реализации могут выбрать поддержку менее, чем 32 виртуальных полос или могут быть сконфигурированы, оптимизировав количество буферизации для каждой виртуальной полосы путем уменьшения доступного количества виртуальных полос.

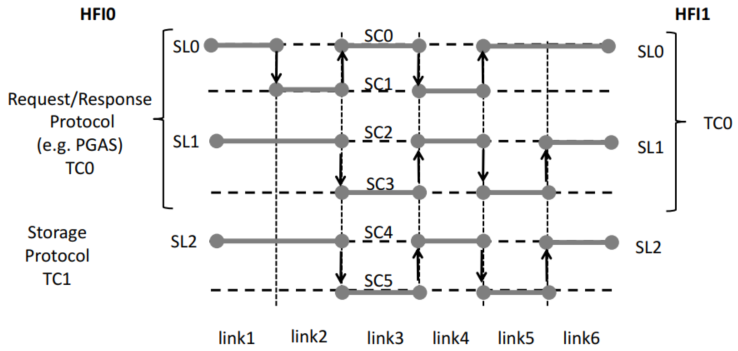


Рис 9. Пример использования классов трафика (TC), уровней сервиса (SL) и каналов сервиса (SC) в топологии с предотвращением взаимной блокировки маршрутизации из-за кредитов.

Каждый канал сервиса передает трафик одного уровня сервиса в одном классе трафика, и менеджер фабрики конфигурирует как каналы сервиса отображаются на ресурсы виртуальных полос на каждом порту. Таким образом, Intel® OPA обширно использует каналы сервиса для предотвращения взаимных блокировок маршрутизации и протокола и для предотвращения «head-of-line blocking» между классами трафика.

Управление перегрузками (Congestion Management). Intel® OPA использует многогранный подход к управлению перегрузками. Адаптивная маршрутизация и дисперсионная маршрутизация предоставляют балансировку нагрузки. Адаптивная маршрутизация идентифицирует перегруженные межсетевые каналы коммутатора и динамически настраивает маршрутизацию трафика, чтобы лучше использовать другие каналы в фабрике. Чтобы минимизировать влияние на транспортный уровень, которые не поддерживают обработку пакетов вне очереди.

В свою очередь, дисперсионная маршрутизация вероятностно распределяет трафик между несколькими маршрутами через фабрику или даже между несколькими виртуальными полосами внутри одного маршрута. Дисперсионная маршрутизация использует несколько путей (multipathing), которая определяет несколько путей между парами конечных точек соединения так, что источник может распространять его трафик между несколькими путями. Это уменьшает образование перегрузок, которые могут возникнуть в результате несбалансированного трафика. Поддержание правильной последовательности доставки не определено между пакетами, которые используют различные пути. Если правильная последовательность доставки необходима между пакетами, то источник должен посылать пакеты по одному пути или использовать другие механизмы, способные восстановить доставку вне очереди. Intel® OPA не определяет как конечные точки соединения должны выбирать путь для отправки пакета. Конечные точки соединения могут использовать любые из доступных путей, если они решают задачу предоставления правильной последовательности доставки.

Разделения (Partitions). Разделения – это механизм изоляции, который работает на канальном уровне, для каждого пакета фабрики, связанного с одни разделом. Разделения предоставляет изоляцию для группы конечных точек соединения, которые являются членами разделения для всех типов трафика (то есть, всех транспортных уровней). Однако, не мешает транспортному уровню обеспечивать более качественную защиту. Разделения в Intel® OPA фабрике содержит группу конечных точек соединения Intel® OPA. Коммуникации разрешены внутри разделения и не возможны для конечных точек, которые не находятся внутри разделения. Это позволяет использовать разделения для обеспечения изоляции между приложениями, запущенными в фабрике или между пользователями в фабрике. Отдельные конечные точки могут быть идентифицированы как полный или ограниченный член данного раздела. Полным членам разрешено общаться с любым членом разделения, но ограниченным членам разрешено общаться только с полными членами. Этот механизм позволяет фабрике иметь общие службы, такие как управление или общую глобальную файловую систему при сохранении изоляции между разделениями без обслуживания.

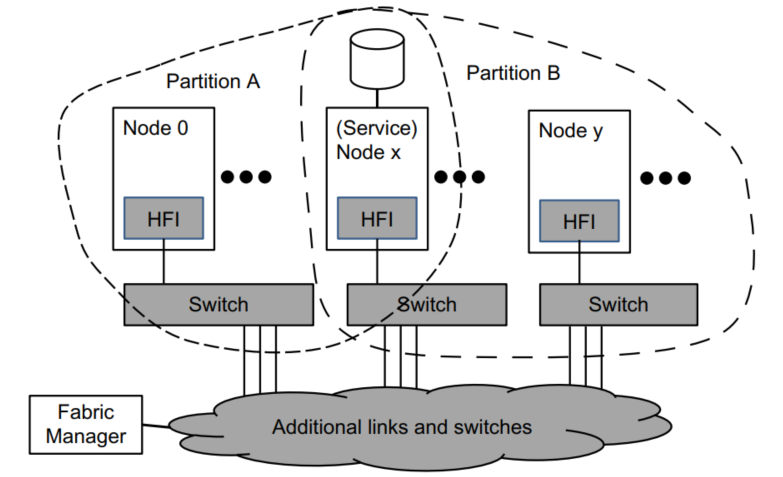


Рис 10. Intel® OPA разделения.

1. Транспортный уровень (Layer 4)

Несколько транспортных уровней могут быть инкапсулированы в пакеты канального уровня. Эти транспортные слои оптимизированы для общих моделей использования в HPC среде и датацентрах и несут ответственность за перенос семантики пользователя и предоставление сквозной (end-to-end) надежности. Все текущие транспортные уровни предоставляют инвариантный CRC (ICRC) для каждого пакета, который охватывает поля заголовка канального и транспортного уровня. При потере или повреждении пакетов протокол сквозной надежности будет повторно передавать пакеты. На транспортном уровне предоставляются программные модели для предоставления пользователю сетевого API. Три ключевых слоя программного обеспечения включают в себя PSM, OFED-Verbs и OpenFabrics Alliance (OFA) Open Fabrics Interface (OFI).

1. Performance Scaled Messaging (PSM)

PSM – это библиотека пользовательского уровня, которая предоставляет API для надежного транспортного уровень фабрики для Intel® Omni-Path HFI. PSM API предоставляет семантику согласованных очередей (Matched Queue - MQ), которая является строительным блоком для сопоставления тегов в MPI. PSM API был расширен для Omni-Path Architecture, чтобы предоставлять поддержку сопоставления тегов размером 96 бит, что позволяет поддерживать 32-битные пользовательские теги, 32-битную информацию об исходящем ранке и 32-битный контекст коммуникатора. Эти изменения позволяют повысить масштабируемость по сравнению с предыдущей версией библиотеки PSM, которая поддерживала теги размером до 64 бит. Примитивы для передачи сообщений, предоставляемые PSM, реализуют семантику точка-точка (point-to-point). Реализация PSM разработана для масштабирования до порядка миллионов MPI рангов.

Коллективные MPI операции получаются путем использования точка-точка примитивов отправки и приема, использую оптимизированные алгоритмы, которые могут быть выбраны из параметров, таких как размер сообщения, размер коллективного коммуникатора и топологии. Дополнительно, PSM предоставляет API для управления механизмом активных сообщения (Active Messages - AM), которые может быть использован для разработки произвольных протоколов связи с использованием парадигмы активных сообщений. AM используется для реализации модели программирования PGAS с использованием API OpenSHMEM, работающего поверх GasNET.

1. OFED Verbs

Intel® Omni-Path HFI полностью поддерживает Verbs интерфейс. Поддерживаются UD, UC и RC пары очередей (Queue Pairs – QP). Разделяемая очередь приема (Shared Receive Queue – SRQ) также поддерживается. Предоставляются стандартные Verbs интерфейсы к библиотекам уровня пользователя и ядра. Все стандартные Verbs протоколы управления и производительности поддерживаются. В дополнении к поддерживаемым 2 и 4 килобайтным MTU (Maximum Transfer Unit) у InfiniBand, Intel® OPA определяет 8 килобайтный MTU.

Высокопроизводительные коммуникационные интерфейсы общего назначения

OpenFabrics Interfaces

Интерфейсы OpenFabrics (OpenFabrics Interfaces – OFI) – это фреймворк, который фокусируется на предоставлении коммуникационных сервисов приложениям. OFI специально разработан для удовлетворения требований к произодительности и масштабируемости HPC приложений, таких как MPI, SHMEM, PGAS, DBMS и корпоративных приложений, работающих в тесно связанной сетевой среде.

Libfabric – это библиотека, которая определеяет и реализует API пользовательского пространства OFI и, как правило, является единственным программным обеспечение, с которым приложения напрямую связаны. API библиотеки libfabric не зависит от сетевых протоколов, а также от реализации определенных сетевых устройств, над которым он может быть реализован.

Libfabric предназначен для выравнивания сервисов фабрики с потребностями приложений, обеспечивая плотную семантическую подгонку между приложениями и нижележащим оборудованием фабрики.

Архитектура

Рисунок 11 демонстрирует архитектуру двух основных компонентов OFI, библиотеки libfabric и OFI провайдера, показанных между приложениями с поддержкой OFI и гипотетической сетевой картой (Network Interface Card – NIC), которая поддерживает обработку прямого I/O.

Библиотека libfabric определяет интерфейсы, используемые приложениями, и предоставляет некоторые общие службы. Однако основная часть реализации OFI находится в провайдерах. OFI провайдеры подключаются к libfabric и обеспечивают доступ к оборудованию фабрики и ее сервисам. Поставщики часто связаны с конкретным аппаратным устройством или сетевым адаптером.

Благодаря структуре libfabric, приложения получают непосредственный доступ к реализации провайдера для большинства операций, чтобы обеспечить минимально возможные задержки программного обеспечения.

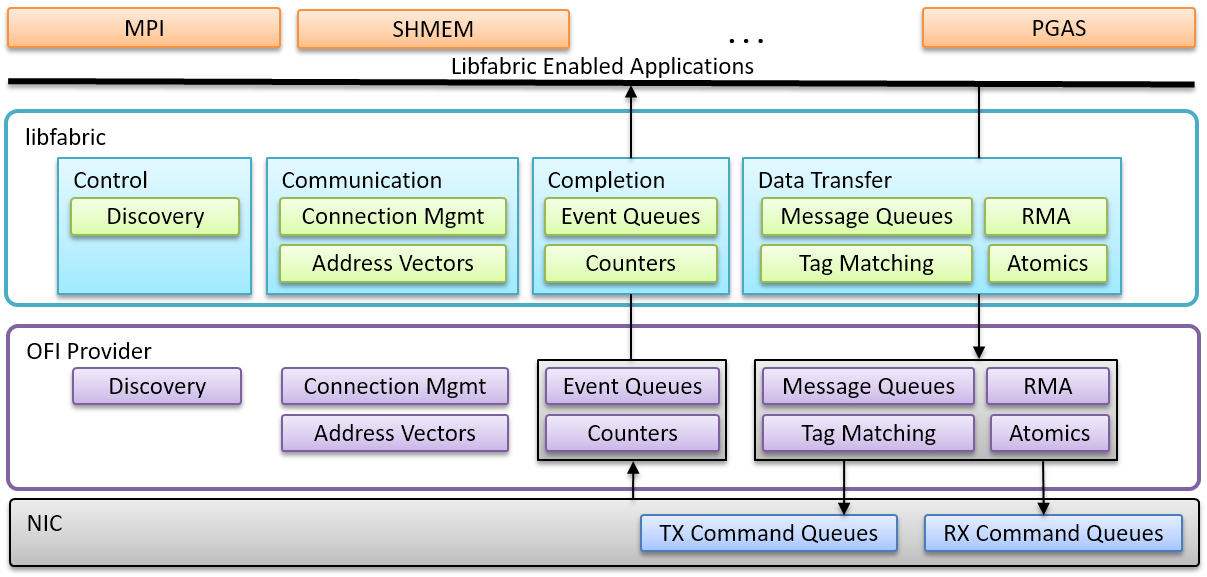


Рисунок 11. Архитектура libfabric и провайдера OFI, расположенная между приложениями и гипотетическим сетевым адаптером.

Как показано на рисунке 1, сервисы libfabric можно сгруппировать в четыре основные службы.

1. Контрольные службы

Эти службы используются приложениями для обнаружения информации о типах коммуникационных услугах, доступных в системе. Например, система обнаружения укажет какие фабрики достижимы с локального узла и какой тип коммуникаций каждая фабрика предоставляет.

Службы обнаружения позволяют приложению запрашивать определенные функции или возможности, например, желаемую модель коммуникаций от нижележащего провайдера. В ответ провайдер может указать, какие дополнительные возможности может использовать приложение, без отрицательного влияния на производительность или масштабируемость, а также требования о том, как приложение может наилучшим образом использовать базовое оборудование. Провайдер указывает последний, установив биты режима, которые кодируют ограничения на использование приложения интерфейсом. Такие ограничения обусловлены соображениями производительности, основанными на внутренних функциях реализации конкретного поставщика.

Результатом процесса обнаружения является то, что провайдер использует запрос приложения для выбора пути к программному обеспечению, который лучше всего подходит как для потребностей этого приложения, так и для ограничений провайдера.

1. Коммуникационные службы

Эти службы используются для установления связи между узлами. Они включают в себя вызовы для установления соединений (управление соединениями), а также функции, используемые для управлениями конечными точками без установления соединения (адресные векторы). Коммуникационные интерфейсы предназначены для абстрагирования специфических деталей фабрики и оборудования, используемых для подключения и настройки конечных точек соединений. Однако, векторы адресов предназначены для минимизации объема памяти, необходимой для хранения данных адресации для потенциально миллионов удаленных одноранговых узлов.

1. Службы оповещения о готовности

Libfabric предоставляет асинхронные интерфейсы, а службы завершения используются провайдером для прямого отчета о результатах предыдущих асинхронных операций. О завершении можно сообщить либо с помощью очередей событий, либо с помощью легковесных счетчиков.

Записи в очередях событий предоставляют сведения о завершенной операции в нескольких форматах, которые приложение может выбрать, чтобы свести к минимуму данные, которые должны быть предоставлены провайдеров. Счетчики просто сообщают количество выполненных операций.

1. Службы передачи данных

Службы передачи данных представляют собой наборы интерфейсов, разработанных по различным парадигмам связи. На рисунке 1 показаны четыре базовых набора интерфейса передачи данных. Эти службы передачи данных предоставляют приложениям прямой доступ к реализации провайдером соответствующей службы.

Интерфейсы передачи данных предназначены для устранения ветвей, которые произойдут в реализации поставщика, и уменьшают количество ссылок на память, например, позволяя ему перематывать командные буферы для дальнейшего сокращения количества команд, выполняемых при передаче.

1. Очереди сообщений предоставляют возможность отправлять и получать данные, в которых сохраняются границы сообщений. Они действуют как FIFO, причем сообщения, поступающие с удаленного отправителя, сопоставляются с очередью запросов на получение в порядке их получения провайдером на локальном узле.
2. Механизм соответствия меток (tag matching) аналогичен очередям сообщений, поскольку оно поддерживает границы сообщений, но отличается тем, что принятые сообщения направляются к очереди запросов на получение после выполнения сранения меток, которые переносятся в сообщении.
3. Передача с удаленным доступом к памяти (Remote Memory Access – RMA) позволяет приложению записывать данные из локальной памяти непосредственно в указанную ячейку памяти целевого процесса или считывать данные в локальную память непосредственно из указанной ячейки памяти целевого процесса.
4. Атомарные операции аналогичны передачам RMA, поскольку они позволяют осуществлять прямой доступ к указанному местоположению памяти целевого процесса, но отличаются тем, что позволяют манипулировать значением, записанным в этой памяти, например, увеличивать или уменьшать его.

Объектная модель

В основе архитектуры libfabric лежит объектно-ориентированная концепция. С высокоуровневой точки зрения индивидуальные службы фабрики связаны набором интерфейсов. Например, службы RMA доступны с использованием набора четко определенных функций. Наборы интерфейсов связаны с объектами, предоставляемые libfabric. Связь между объектом и набором интерфейсов примерно такая же, как и отношение между объектно-ориентированным классом и его функциями-членами, хотя фактическая реализация отличается по причинам производительности и масштабируемости.

Объект настраивается на основе результатов служб обнаружения. Чтобы обеспечить оптимальные пути программного обеспечения между программным и аппаратным оборудованием, провайдеры динамически связывают объекты с наборами интерфейсов на основе режимов, поддерживаемых провайдером, и возможностей, запрашиваемых приложением.

На рисунке 12 показан высокоуровневая модель отношений родитель-потомок между объектами libfabric.

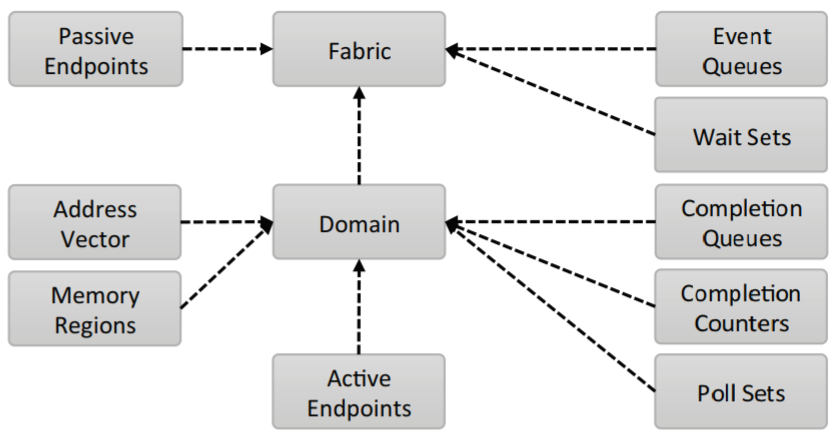


Рисунок 12. Объектная модель libfabric.

1. Фабрика (Fabric): Фабрика представляет собой набор аппаратных и программных ресурсов, которые имеют доступ к одной физической или виртуальной сети. Все сетевые порты в системе, которые могут взаимодействовать друг с другом через фабрику, принадлежащие к одному и тому же домену фабрики. Фабрика не только включает в себя локальные и удаленные сетевые карты, но и соответствующее программное обеспечение, коммутаторы, маршрутизаторы и любые необходимые компоненты управления сетью или подсетями.
2. Домен (Domain): Домен представляет логическое соединение в фабрике. Например, домен может отображаться в физический или виртуальный сетевой адаптер. Домен определяет границу, в пределах которой могут быть связаны ресурсы фабрики. Каждый домен принадлежит к одной фабрике.

Свойства домена описывают, как будут использоваться, связанные с ним, ресурсы. Атрибуты домена включают информацию о модели использования потоков приложением и о том, как ресурсы фабрики могут быть распределены между потоками. Он также определяет взаимодействия, которые происходят между конечными точками соединения (endpoints), очередями событий готовности, счетчиками готовности и адресными векторами. Цель заключается в том, чтобы приложение передавало достаточно информации, что провайдер мог выбрать наиболее оптимальную реализацию, адаптированную к потребностям приложения.

1. Пассивная конечная точка соединения (Passive Endpoint): Пассивные конечные точки соединения используются протоколами, ориентированными на соединение, для прослушивания входящих запросов на соединение, концептуально эквивалентных прослушивающим сокетам.
2. Активная конечная точка соединения (Active Endpoint): Активная конечная точка соединения (или, просто, конечная точка соединения) представляет собой коммуникационный портал и концептуально похожа на сокет.

Конечные точки обычно связаны с контекстом передачи и/или контекстом приема. Эти контексты часто реализуются с использованием аппаратных очередей, которые отображаются непосредственно в адресное пространство процесса, что позволяет обойти ядро операционной системы для передачи данных. Запросы передачи данных преобразуются нижележащим провайдером в команды, которые вставляются в контексты передачи и/или приема.

Более продвинутая модель использования конечных точек позволяет разделять ресурсы. Поскольку контексты передачи и приема могут быть связаны с ограниченными аппаратными ресурсами, libfabric определяет механизмы совместного использования контекстов между несколькими конечными точками. Общие контексты позволяют приложению или менеджеру ресурсов определять приоритеты распределения ресурсов и использования общих ресурсов оборудования.

В отличие от общих контекстов, имеется модель конечной точки соединения, известная как масштабируемая конечная точка соедениения. Масштабируемые конечные точки позволяют одной конечной точке использовать несколько базовых аппаратных ресурсов, имея несколько контекстов передачи и/или приема. Масштабируемые контексты позволяют приложениям выделять ресурсы, чтобы избежать ограничений синхронизации потоков или ограничений на порядок данных, не увеличивая объем памяти, необходимый для адресации.

1. Очередь событий (Event Queue – EQ): Очередь событий используется для сбора и отчета о завершении асинхронных операций и событий. Она обрабатывает управляющие события, которые напрямую не связаны с операциями передачи/приема данных, такими как запросы на соединение и ошибки асинхронных операций.
2. Очередь событий о готовности (Completion Queue – CQ): Очередь завершения – это высокопроизводительная очередь, используемая для сообщения о завершении операций передачи данных. Контексты передачи и приема связаны с очередями завершения. Формат событий, считываемых из очереди завершения, определяется приложением. Это позволяет создавать компактные структуры данных с минимальной записью в память.
3. Счетчики завершения (Completion Counter): Счетчик завершения - легковесная альтернатива очереди завершения, поскольку ее использование просто увеличивает счетчик, а не помещает запись в очередь. Подобно CQ, конечная точка соединения связывается с одним или несколькими счетчиками. Однако, счетчики обеспечивают более тонкую детализацию типов завершения, которые они могут отслеживать.
4. Множество ожидания (Wait Set): Множество ожидания предоставляет единый объект ожидания, который должен сигнализировать всякий раз, когда указанное условие возникает в очереди событий, очереди завершения или счетчике, принадлежащем данному множеству. Множества ожидания позволяют оптимизировать методы для приостановки и передачи сигналов. Приложения могут запрашивать использование определенного типа объекта ожидания, например дескриптор файла, или позволить провайдеру выбирать оптимальный объект.
5. Множество объектов опроса (Poll Set): Несмотря на то, что libfabric разработан для поддержки провайдеров, которые перекладывают обязанности по передаче данных напрямую на аппаратное обеспечение, он поддерживает провайдеров, которые используют центральный процессор для выполнения операций. Множество объектов опроса позволяет приложениям группировать очереди событий о готовности или счетчики готовности, позволяя за один вызов функции опроса, добиваться прогресса для нескольких объектов готовности.
6. Регион памяти (Memory Region – MR): Регион памяти определяет буфер локальной памяти приложения в терминах нижележащего провайдера. Для того, чтобы провайдер мог получить доступ к памяти приложения во время передачи данных, RMA или атомарных операций, приложение обязано предоставить соответствующие разрешения провайдеру, зарегистрировав объект региона памяти. Libfabric определяет несколько режимов для создания регионов памяти. Он поддерживает метод, который хорошо согласуемый с существующим оборудованием InfiniBand и iWARP, но для масштабирования на миллионы одноранговых коммуникационных сторон также имеется поддержка адресации с использованием смещений и пользовательских ключей.
7. Вектор адресов (Address Vector – AV): Векторы адресов используется конечными точками без установления соединения для сопоставления адресов более высокого уровня, которые могут быть более естественными для приложения, например, IP-адресов, в адреса, специфичные для конкретной фабрики. Это позволяет провайдерам сократить объем памяти, необходимый для поддержки больших таблиц адресов, а также устранить дорогостоящие методы разрешения адресов и поиска во время операций передачи данных.

Libfabric заимствовал и расширил концепции, определенные в других API, а также объединил их в расширяемую структуру. Дополнительные объекты могут быть легко введены или добавлены новые интерфейсы к существующему объекту. Тем не менее, определения объектов и интерфейсы разработаны специально для содействия масштабированию программного обеспечения и низкой задержке, когда это необходимо. Libfabric разрабатывался для обеспечения того, чтобы объекты обеспечивали правильный уровень абстракции, чтобы избежать неэффективности в приложениях и в провайдерах.

Unified Communication X

UCX – это фреймворк, предоставляющий сетевой API для современных сетевых адаптеров. Целью предоставляемого API является создание набора интерфейсов для реализации множества библиотек программных моделей и языков, которые являются переносимыми, масштабируемыми и эффективными.

Фреймворк разработан с учетом новых технологий для экзаскейла, таких как массивные параллельные вычислительные узлы, ускорители и иерархические запоминающие устройства. Архитектура UCX предоставляет программные конструкции для высокопроизводительных коммуникаций с высокой пропускной способностью, коммуникациях в гетерогенных иерархиях памяти и гибридных программных моделях, включая библиотеки ввода-вывода и библиотеки ориентированные на данные. Вместо того, чтобы создавать единый интерфейс как «подходящее для всех решение», UCX сообщество разрабатывает структуру, которая предоставляет необходимые компоненты для построения различных коммуникационных протоколов с использованием разных уровней абстракции. Такая архитектура обеспечивает высокую степень гибкости, позволяющую реализовать новые сетевые протоколы для новых моделей программирования.

К примеру, ожидается, что ускорители будут ключевым компонентом в проектировании систем, достигающих пиковой производительности в несколько едининиц exaFLOPS. Новые механизмы, такие как GPUDirect RDMA и GPUDirect Async, позволят практическим задачам, способным работать с GPU, работать с минимальным вмешательством CPU, оставляя другие, более общие задачи, для вычисления на CPU. UCX предоставляет отдельный транспорт для работы с памятью ускорителя, который позволяет адаптировать реализации к потребностям проектирования коммуникаций между GPU.

Архитектура

UCX состоит из трех основных компонент, которые предосталяют API и могут быть использованы поотдельности, или в совокупности, как отдельная библиотека (Рисунок 13):

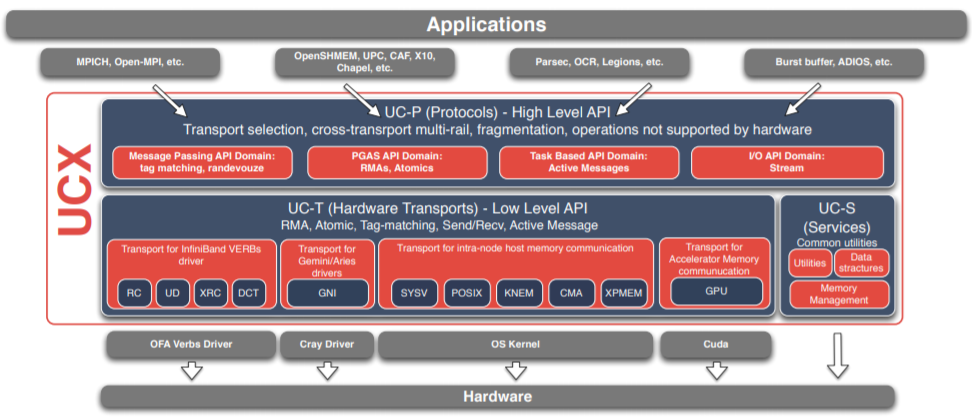


Рисунок 13. Архитектура UCX.

* UC-Services (UCS) – это уровень обслуживания, который обеспечивает необходимую функциональность для реализации портативных и эффективных утилит. Этот уровень предоставляет следующие службы:
  + Абстракции для доступа к функциональным возможностям платформы (атомарные операции, обеспечение потоковой безопасности и другие);
  + Инструменты для эффективного управления памятью (пулы памяти, распределители памяти и перехватчики памяти);
  + Широко используемые структуры данных (хеши, деревья, списки).
* UC-Transport (UCT) – это транспортный уровень, которые абстрагирует различия для разнообразных архитектур аппаратных средств и предоставляет низкоуровневый API, которые позволяет реализовывать протоколы связи. Основная цель этого уровня – это предоставление прямого и эффективного доступа к сетевым ресурсам аппаратных средств с минимальнымы накладными расходами, возникающими из-за программного обеспечения. Для реализации этой цели UCT полагается на низкоуровневые драйвера, предоставляемыми вендорами, такие как InfiniBand Verbs, Cray’s uGNI и другие. В отношении коммуникационного API, UCT определяет интерфейсы для коротких (*short*), буфферизированных скопировать-и-отправить (*bcopy*), без выполнения копирования, так называемых ноль копирований (*zcopy*), коммуникационных операций. *Short* операции являются оптимальными для маленьких сообщений, которые могут быть отправлены на месте. *Bcopy* опреации оптимальны для средних сообщений, которые обычно могут быть отправленые через, так называемые, перехватывающие буфера. *Zcopy* операции предоставляют семантику коммуникаций, которая выполняется без выполнения дополнительных копирований на уровне программного обеспения и сообщения сразу могут быть записаны в результирующую память стороны приема.
* UC-Protocols (UCP) реализует высокоуровневые протоколы, которые обычно используются MPI и PGAS программными моделями, путем использования низкоуровневых возможностей через UCT уровень. UCP ответственен за следующую функциональность: инициализация библиотеки, выбор транспорта для коммуникации, фрагментация сообщений, коммуникации через несколько сетевых интерфейсов (многорельсовая – multi-rail). UCP API имеет следующие классы интерфейсов: инициализация, удаленный доступ к памяти (Remote memory Access – RMA), атомарные операции с памятью (Atomic memory Operations – AMO), Активные сообщения (Active messages – AM), механизм сопоставления тегов (Tag-Matching – TM), коллективнные операции (Collectives), многорельсовая передача (Multi-rail).
  + Инициализация: Это подмножество интерфейсов определяет настройку коммуникационного контекста, запрашивает сетевые возможности и инициализирует локальные конечные точки связи. Коммуникационный контекст представляется UCX контекстом, которые является абстракцией сетевых ресурсов транспортного уровня. Интерфейсы инициализации конечной точки соединения инициализируют UCP конечную точку соединения, который является абстракцией всех необходимых ресурсов, ассоциированных с определенным соединением. Конечная точка соединения используется как вход для всех коммуникационных операций между определенным источником и получателем.
  + RMA: Это подмножество операций определяет односторонние коммуникационные операции, такие как PUT и GET, которые используются для реализации коммуникаций с прямым доступом к удаленной памяти, необходимым для распределнных программных моделей и моделей с общей памятью. UCP имеет отдельное множество интерфейсов для использования несмежных (non-contiguous) данных в коммуникациях. Эта функциональность была включена, чтобы поддерживать коммуникационные требования различных программных моделей и оперировать способности разброса/сбора (scatter/gather) современных сетевых аппаратных средств.
  + AMO: Это подмножество интерфейсов предоставляет поддержку для атомарно выполняемых операций в удаленной памяти, которые являются важным классом операций для PGAS программных моделей, в частности OpenSHMEM.
  + TM: Этот интерфейс поддерживает сопоставление тегов для операций приема и отправки, которые являются ключевой семантикой коммуникаций, определяемой спецификацией MPI.
  + AM: Это подмножество функциональностей, где входящий пакет генерирует вызов обработчика для, указанного отправителем, события на стороне получателя, чтобы быть обработанным соответствующим образом на стороне получателя. Например, одно- и двух-сторонние MPI операции могут быть реализованы на основании данной концепции. Однако, эти интерфейсы являются более общими и подходящими для других программных парадигм, в которых процесс получателя не подготавливает буфера для приема данных, но мождет отреагировать на полученный пакет напрямую. Также как интерфейсы RMA и сопоставлении тегов, интерфейс активных сообщений предоставляет отдельный API для различных типов сообщений и для работы с несмежными данными.
  + Коллективные операции: Это подмножество интерфейсов определяет групповые коммуникации и операции синхронизации. Коллективные операции включают в себя такие коллективные операции, как барьерная операци, все-к-одному, один-ко-многим, операции редукции. Где возможно, UCP использует преимущество аппаратного обеспечения для ускорения коллективных операций, к примеру, механизм коллективных операций на ускорителе InfiniBand коммутатора.

Исследование применимости OpenFabrics Interfaces для MPI

OpenFabrics Interfaces (OFI) был разработан для удовлетворения потребностей современных и перспективных программных моделей, используемых в высокопроизводительных вычислительных (High Performance Computing – HPC) средах, а также для разработки распределенных моделей, используемых для решения задач искусственного интеллекта (Artificial Intelligence – AI). В основном, приложения в HPC среде чувствительны к задержкам, возникающих при передаче маленьких сообщений между узлами. Распределенные модели, используемые для решения задач AI, в свою очередь, предъявляют требования к пропускной способности при передачи больших массивов данных.

Философия OFI заключается в том, чтобы обепечивать высокую эффективность передачи данных для любого размера данных с использованием любой поддерживаемой сетевой фабрикой, предоставляя универсальный интерфейс приложениям.

Для обеспечения низких задержек передачи сообщений для всего спектра размера поддерживаемых MPI сообщений необходимо учитывать возможность получения данных от разных отправителей и отправки данных разным получателям, обеспечивая эффективное потребление памяти и быстрый метод сопоставления MPI ранга c соотвествующим сетевым адресом, используемым в фабрике.

Размеры MPI сообщений можно разбить на следющие три группы:

* Маленькие (от 0 байт до 4 Кбайт), то есть размеры, соопоставимые с размером MTU (Maximum Transmission Unit)
* Средние (от 4 Кбайт до 512 Кбайт)
* Большие (от 512 Кбайт)

Для каждой группы размеров следует применять различные методы передачи данных, чтобы обеспечивать эффективность и высокую скорость.

*Eager* протокол

Практически все OFI провайдеры используют метод предвыделенных буферов, зарегистрированных в сетевом адаптаре и ожидающих получения данных с удаленной стороны. Размер предвыделенных буферов подбирается индивдуально для провайдеров. Предвыделенные буфера используются для реализации так называемого *Eager* протокола.

В предвыделенные буфера провайдеры получают данные с удальной стороны, если это не сервисные данные протокола, то происходит поиск, соответствующего по тегу и адресу назначения, буфера (Рисунок 14):

* Если соответствующий буфер был найден, то пользовательские данные копируются в предоставленный пользовательский буфер.
* Если соответствующий буфер не был найден, то буфер помечается как «неожиданный» («unexpected») и ожидается, когда будет предоставлен пользовательский буфер. Каждый раз, когда приложение предоставляет новый буфер, происходит поиск соответсвующего буфера в списке «неожиданных» буферов.



Рисунок 14. Два сценария работы MPI\_Recv

Количество предвыделенных буферов совпадает с максимально возможным количеством запросов приема данных, которые могут быть помещены в очереди приема данных драйвера сетевой карты. После того, как предвыделенный буфер, в который были получены данные из сети, был использован приложением, он заново помещается в очередь приема для ожидания новых данных. Такой механизм позволяет контролировать расход памяти библиотекой MPI, сохранять высокую скорость передачи сообщений и контролировать дисбаланс между отправкой и приемом данных для двух MPI рангов.

Размер предвыделенных буферов определяется на практике путем проведения экспериментов. Данный размер ограничивает максимальный размер данных, которые могут быть приняты с использованием *Eager* протокола. При выборе размера буфера следуют брать во внимание следующие факторы:

* Общее потребление памяти предвыделенными буферами, которое равняется произведению количества буферов и их размера;
* Число тактов центального процессора, затрачиваемое на копирование пользовательских данных из предвыделенного буфера библиотеки в буфер приложения;
* Анализ пересылаемых размеров данных приложением;
* Производительность других протоколов с более эффективным потреблением памяти для обработки рассматриваемого диапазона размеров сообщений.

*Rendezvous* протокол

Предвыделенные буферы могут быть использованы для получения служебных сообщений. Основным применением служебных сообщений является подготовка удаленной стороны для получения данных, размер которых превосходит размер предвыделенных буферов.

Одним из протоколов, который использует служебные сообщения, является *Rendezvous* протокол. Одной из его особенностей является возможность передачи любого размера данных, используя при этом небольшой размер предвыделенных буферов, тем самым обеспечивая низкое потребление памяти.

Современные сетевые интерфейсы, используемые в высокопроизводительных вычислениях, имеют механизмы удаленного доступа к памяти (RDMA), которые позволяют вычитывать данные из памяти или записывать данные в память удаленной стороны без вовлечения центрального процессора удаленной стороны в этот процесс. Механизм RDMA определяет два метода Read и Write, которые могут быть применены как к смежным (последовательный участок памяти), так и не к смежным данным (вектор последовательных участков памяти).

*Rendezvous* протокол состоит из трех этапов, которые выполняются один за другим:

* «Рукопожатие» («handshake»), при котором ранги обменивается изначальными данными и инициируют передачу данных с помощью протокола *Rendezvous*.
* Обмен данными. Механизм RDMA находит применение в Rendezvous протоколе. Данные могут быть записаны отправителем по указанным пользовательским адресам с помощью RDMA Write или же прочитаны из памяти отправителя по указанным пользовательским адресам с помощью RDMA Read.
* Завершение, при котором сторона, записывающая или вычитывающая данные, то есть, выполняющая RDMA Write или RDMA Read, соответсвенно, уведомляет удаленную сторону об окончании операции.

Возможны 4 реализации *Rendezvous* протокола:

1. Принимающая сторона посылает сервисное сообщение отправляющей стороне, уведомляющее о необходимости отправки данных. Отправляющая сторона посылает адреса пользовательской памяти, с которой данные должны быть прочитаны в пользовательский буфер принимающей стороны. Принимающая сторона выполняет операцию RDMA Read, вычитывая данные из полученных адресов. После окончания вычитывания данных принимающая сторона посылает сервисное сообщение, сообщающее о завершении транзакции (Рисунок 15а).
2. Принимающая сторона сообщает отправляющей стороне адреса пользовательской памяти, в которую необходимо записать данные. Отправляющая сторона, выполняет операцию RDMA Write, записывая данные из адресов пользовательского буфера отправляющей стороны в адреса удаленной памяти пользовательского буфера принимающей стороны. После окончания записи данных, отправляющий ранг отправляет сервисное сообщение, сообщающее о завершении транзакции (Рисунок 15б).
3. Отправляющая сторона посылет сервисное сообщение принимающей стороне, уведомляющее о необходимости приема данных. Принимающая сторона посылает адреса пользовательского буфера, в которые должны быть записаны данные. Отправялющая сторона выполняет операцию RDMA Write, записывая данные по указанным адресам. После окончания записи данных, отправляющая сторона посолыет сервисное сообщение, сообщающее о завершении транзакции (Рисунок 15в).
4. Отправляющая сторона посылает сервисное сообщение, в котором указывает адреса пользовательского буфера, из которого необходимо выполнить чтение данных. Принимающая сторона выполняет операцию RDMA Read, вычитывая данные из указанных адресов в предоставленный пользовательский буфер. После окончания операции чтения, принимающая сторона отправляет сервисное сообщение, уведомляющее об окончании транзакции (Рисунок 15г).

Адреса памяти, которые участвуют в RDMA операциях, должны быть предварительно зарегистрированы в драйвере сетевой карты. Драйвер сетевой карты возвращает два ключа: локальный и удаленный. Локальный ключ используется на локальной стороне для выполнения операций RDMA Write для записи зарегистрированых данных в удаленную память и RDMA Read для записи данных из удаленной памяти в зарегистрированный буфер. Удаленный ключ сообщается удаленной стороне для выполнения операций RDMA Write для записи данных в, зарегистрированную по этому ключу, память и RDMA Read для чтения данных из, зарегистрированной по этому ключу, памяти. После окончания RDMA операций, ключи могут быть освобождены для последующего использования.

Области память, фигурирующие на втором этапе (обмен данными) *Rendezvous* протокола, должны быть зарегистрированны на стороне получателя и отправителя. В зависимости от реализации данного этапа локальные или удаленные ключи должны быть сообщены либо отправителю, либо получателю для выполнения RDMA операции.

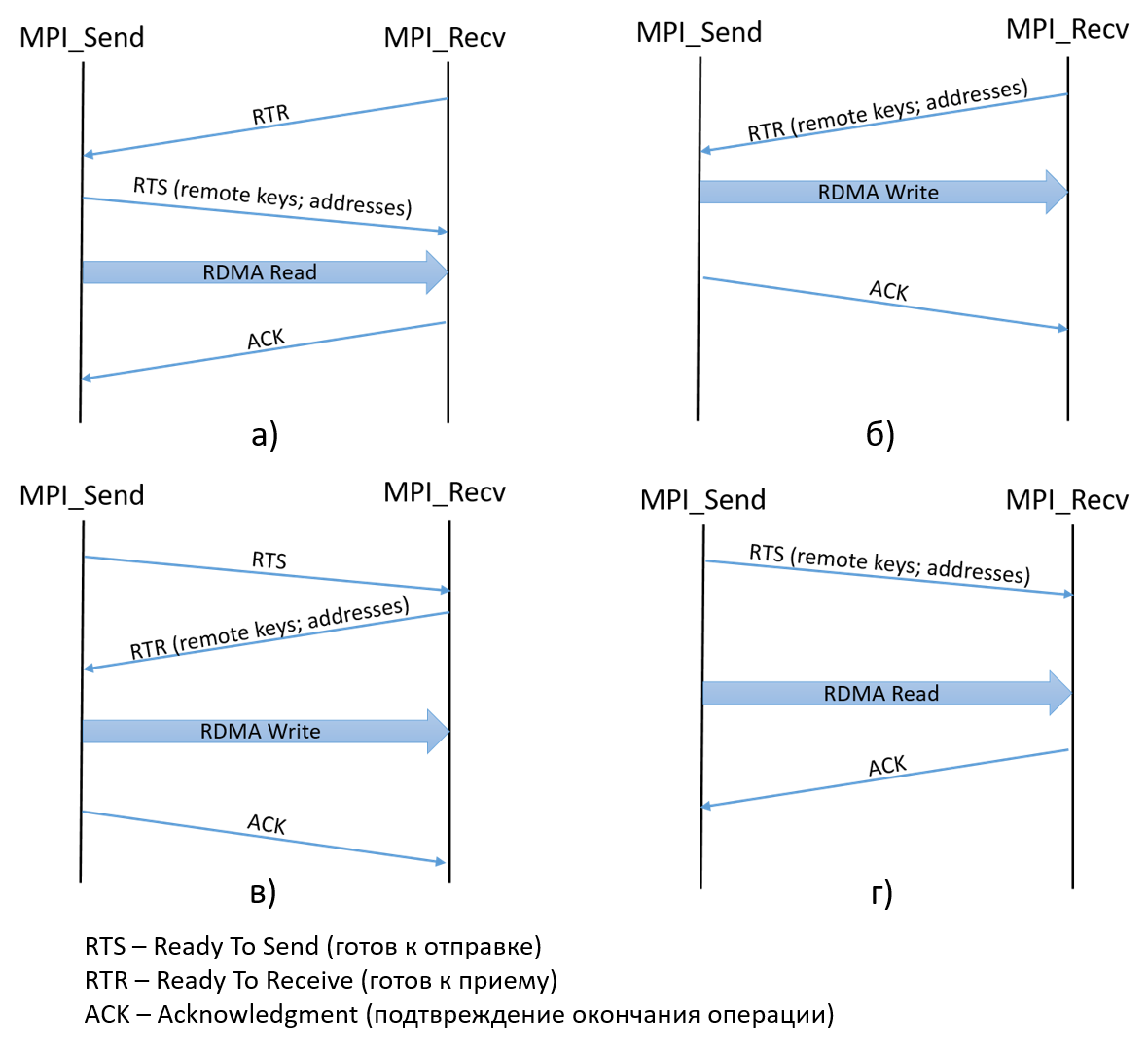


Рисунок 15. Реализации *Rendezvous* протокола.

Представленные реализации *Rendezvous* протокола с сообщением возможности получения данных с удаленной стороны (принимиающая сторона отправляет сервисное сообщение RTR) невозможно с использовать (Рисунки 15а и 15б), если MPI ранг отправителя не известен, то есть в качестве аргумента *source* функции *MPI\_Recv* был передан флаг *MPI\_ANY\_SOURCE*, так как неизвестен номер ранга, которому следует отправить сервисное сообщение RTR. Также стоит отметить, что реализации, которые в первом сервисном сообщениие (RTR или RTS) не сообщают удаленные ключи и адреса пользовательской памяти, над которой необходимо выполнить RDMA операции, являются неоптимальными, так как в таких реализацию двум рангам следует обменяться тремя сервисными (Рисунок 15а и 15в) вместо двух, как в других реализациях (Рисунок 15б и 15г).

Из представленного анализа, легко заметить, что наиболее универсальным и оптимальным является реализация, изображенная на Рисунке 15г.

Сравнение *Eager* и *Rendezvous* протоколов

Для проведения сравнения двух протоколов была использована следующее окружение:

* Аппаратное обеспечение:
  + CPU: Dual-socket Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  + Infiniband controller: Mellanox Technologies MT27700 Family (ConnectX-4)
* Программное обеспечение:
  + MPICH v3.3b2. Библиотека была собрана с поддержкой сетевого модуля CH4/OFI;
  + OFI v1.6.1. Библиотека была собрана с поддержкой Verbs провайдера, который может быть использован для InfiniBand и Intel® OPA фабрик, а также для Ethernet решений с поддержкой RDMA (iWarp, RoCEv1, RoCEv2), и RxM (RDM over MSG) провайдера, который используется для реализации Reliable Datagram (RDM) типа конечных точек соединений на основе базового провайдера с поддержкой Reliable Connected (MSG) типа конечных точек соединений, который непосредственно использует предоставляемые интерфейсы драйвера;
  + OSU Benchmarks из пакета MVAPICH v2.3b;
  + Intel® MPI Benchmarks 2019 Beta
  + Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux 4.3-1.0.1.0
  + RedHat Enterprise Server 7.4.

Запуск эксперимента для оценки производительности *Eager* протокола MPI библиотеки производился следующим образом:



Результаты, полученные для :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Количество байт** | **Задержка [микросекунды]** | **Пропускная способность [Мбайт/сек]** | **CPU cycles - memcpy** | **CPU cycles - fi\_send** | **memcpy/fi\_send, %** |
| 0 | 1.08 | 0 | 48 | 258 | 18.6 |
| 1 | 1.09 | 0.92 | 49 | 259.85 | 18.86 |
| 2 | 1.09 | 1.83 | 51 | 260.98 | 19.54 |
| 4 | 1.08 | 3.69 | 47.79 | 257.74 | 18.54 |
| 8 | 1.08 | 7.38 | 47.53 | 261.76 | 18.16 |
| 16 | 1.08 | 14.75 | 48.21 | 261.22 | 18.46 |
| 32 | 1.09 | 29.39 | 47.2 | 285.73 | 16.52 |
| 64 | 1.12 | 57.24 | 49.12 | 290.74 | 16.89 |
| 128 | 1.16 | 110.68 | 54.59 | 305.8 | 17.85 |
| 256 | 1.56 | 164.4 | 64.99 | 337.82 | 19.24 |
| 512 | 1.64 | 311.42 | 82.49 | 378.63 | 21.79 |
| 1024 | 1.8 | 567.4 | 152.1 | 420.38 | 36.18 |
| 2048 | 2.18 | 941.18 | 220.15 | 646.93 | 34.03 |
| 4096 | 2.87 | 1428.2 | 475.3 | 874.55 | 54.35 |
| 8192 | 4.07 | 2010.9 | 1110 | 1532.3 | 72.44 |
| 16384 | 6.21 | 2636.9 | 3113 | 3504.9 | 88.82 |
| 32768 | 9.96 | 3290.7 | 6649.9 | 7005.4 | 94.93 |
| 65536 | 16.62 | 3942.2 | 15555 | 15586 | 99.8 |
| 131072 | 34.21 | 3831.1 | 31820 | 32549 | 97.76 |
| 262144 | 67.63 | 3876.1 | 70333 | 70678 | 99.51 |
| 524288 | 132.88 | 3945.5 | 154044 | 155276 | 99.21 |
| 1048576 | 266.59 | 3933.3 | 295628 | 296120 | 99.83 |
| 2097152 | 513.12 | 4087 | 541785 | 585018 | 92.61 |
| 4194304 | 1043.7 | 4018.7 | 1E+06 | 1E+06 | 99.84 |
| 8388608 | 2079.8 | 4033.4 | 2E+06 | 2E+06 | 85.73 |
| 1.7E+07 | 4373.97 | 3835.7 | 4E+06 | 5E+06 | 88.11 |

Таблица 2. Результаты тестирования производительности *Eager* протокола.

Заключение

Текущее состояние развития сетевых технологий позволяет использовать современные сетевые среды и оптимальные механизмы для передачи/приема данных, но появлением новой сетевой технологии влечет за собой создание нового программного интерфейса для разработчиков. Такая тенденция ведет к созданию проприетарных реализаций MPI протокола, которые являются непортируемыми на другие системы.

Создание унифицированного интерфейса, который позволит оптимально, с минимальными накладными расходами обеспечить портируемость на все современные сетевые технологии, является важной задачей, стоящей перед разработчиками и исследователями современных HPC сообществ.

За последние пару лет появились два конкурирующим между собой интерфейса - OFI (Open Fabric Interfaces) [2], разработанный Open Fabric Alliance, и UCX (Unified Communication X)[3], разработанный OpenUCX сообществом. Современные реализации MPI переходят на использование этих интерфейсов для обеспечения лучшей переносимости.

Литература

1. Tom Shanley, MindShare, Inc. InfiniBand Network Architecture, Addison-Wesley Professional, 2003;
2. OpenFabrics Interfaces Working Group - <https://www.openfabrics.org/>; <https://github.com/ofiwg>;
3. Unified Communication X Working Group - <https://github.com/openucx>; <http://www.openucx.org/>.