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](#_Toc508228708)

[Введение 5](#_Toc508228709)

[Цель работы 9](#_Toc508228710)

[Обзор предметной области 11](#_Toc508228711)

[InfiniBand Architecture 14](#_Toc508228712)

[Элементы InfiniBand Architecture 15](#_Toc508228713)

[Уровни InfiniBand Architecture 18](#_Toc508228714)

[Коммуникационные модели InfiniBand Architecture 22](#_Toc508228715)

[Intel® Omni-Path Architecture 24](#_Toc508228716)

[Элементы Intel® Omni-Path Architecture 25](#_Toc508228717)

[Заключение 27](#_Toc508228718)

[Литература 28](#_Toc508228719)

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:

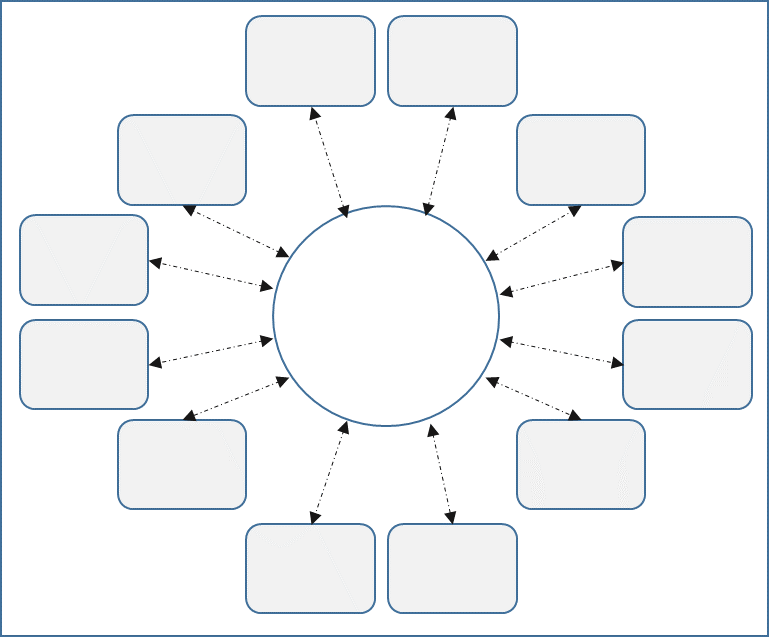
1. анализу существующих программных моделей для параллельного программирования и их требований к коммуникационным интерфейсам;
2. рассмотрению существующих и широкоиспользуемых программных интерфейсов для работы с высокоскоростными сетями и возможности построения единой программной модели, способной 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

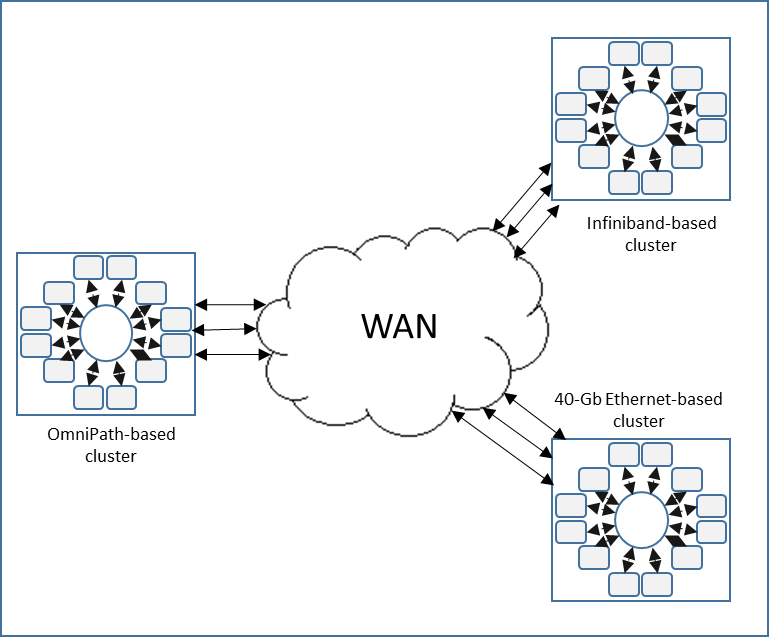
За последние несколько десятилетий достигнуты большие успехи в области компьютерных технологий. Компьютеры проникли во все области деятельности человека, будь то промышленность или исследования, и оказывают влияние на каждый аспект человеческой жизни. В начале 70-х гг XX века, когда обычные компьютеры 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.

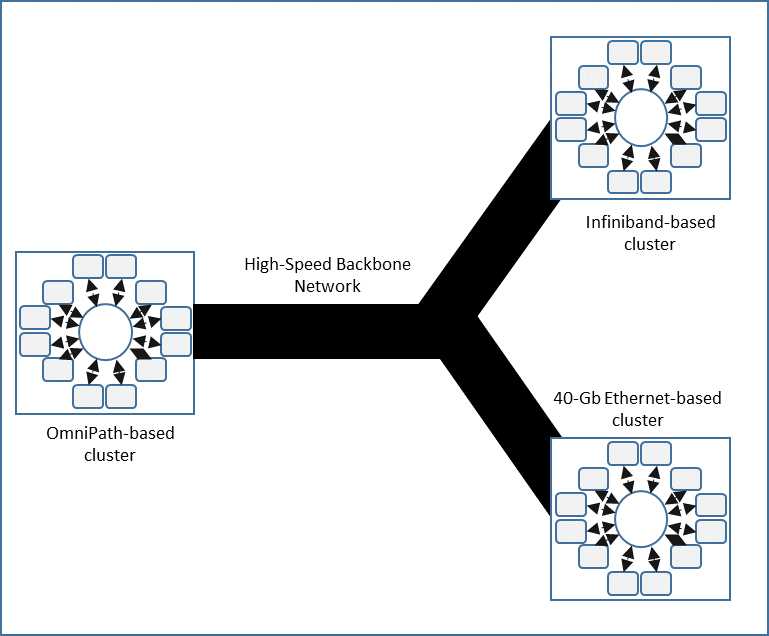
Поскольку системы на основе кластеров опираются на возможности вычислительных узлов, взаимодействующих друг с другом по сети, характеристики сети (аппаратное обеспечение, а также связанное с ним программное обеспечение) являются важным компонентом в эффективности и масштабируемости кластеров. На рис. 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.



(а)



(b)



(c)

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 он 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 новые 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 Sockets интерфейса 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 не только работать непосредственно в различных высокоскоростных сетях без каких-либо изменений, но также иметь возможность извлекать наилучшую производительность из сети.
2. Иметь возможность извлекать выгоду от предоставляемых способностей сетей, не ограничивая производительность и переносимость при переходе на другие сетевые технологии.
3. Обеспечивать надежность передачи данных, высокую масштабируемость приложений.

Цели данной работы состоят в том, чтобы:

1. Исследовать один из современных программных интерфейсов, 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 исследуемого ПО, 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, мощность вычислительных узлов увеличивается в два раза каждые 18 месяцев. В тоже время современные высокоскоростные сети, которые обеспечивают высокую пропускную способность и низкие задержки также являются неотъемлемой частью для построения высокопроизводительных вычислительных сред. Как отмечалось ранее, современные высокопроизводительные системы создаются путем объединения вычислительных узлов посредством коммуникационных 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.

Как отмечалось ранее, большинство научных приложений использует MPI для достижения распределенности вычислений, поэтому в данной работе основные исследования связаны с MPI.

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 протокол используется для больших сообщений.

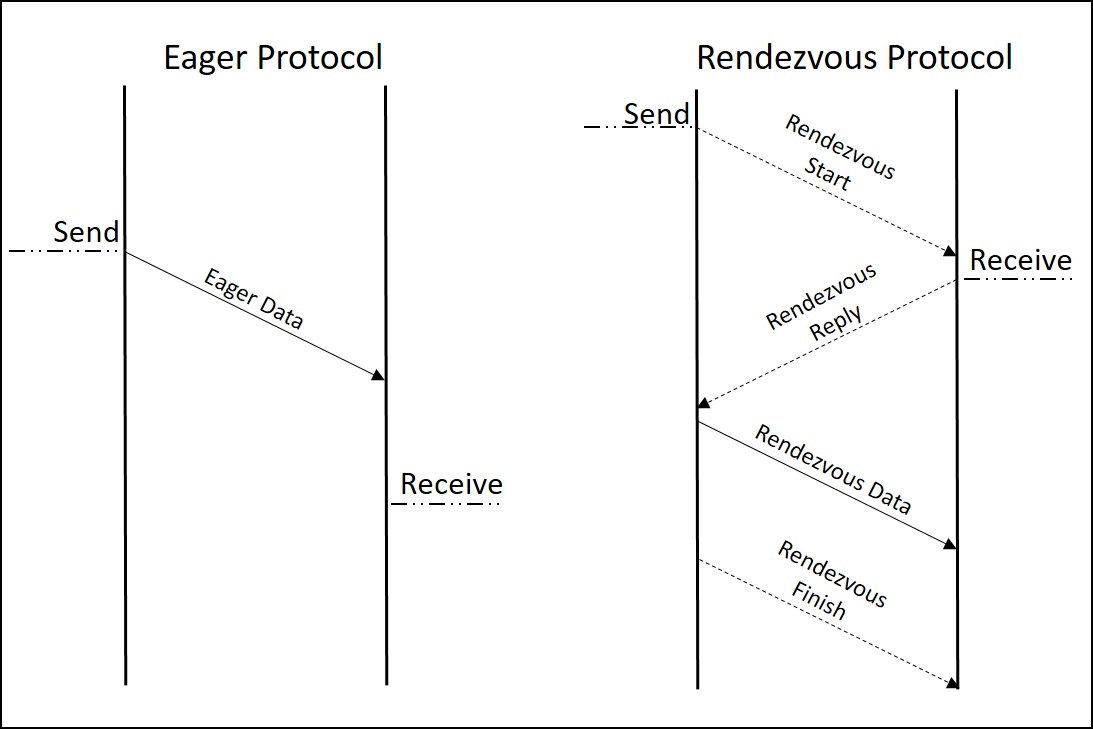


Рис. 2. Типичная реализация Eager и Rendezvous протоколов.

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 поодерживает Verbs интерфейс.

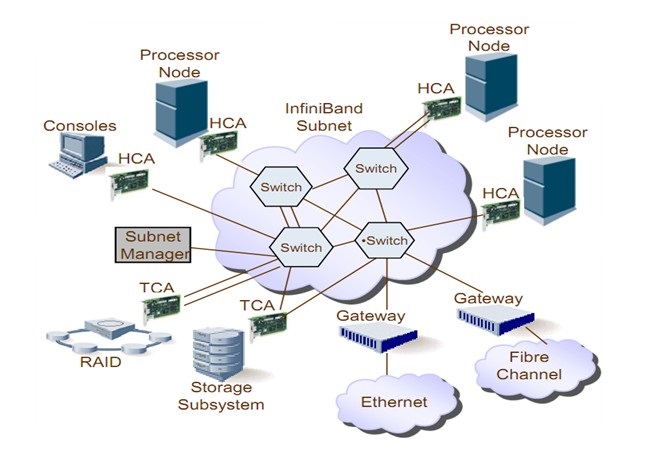


Рис. 3. Пример организации InfiniBand подсети.

Элементы InfiniBand Architecture

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

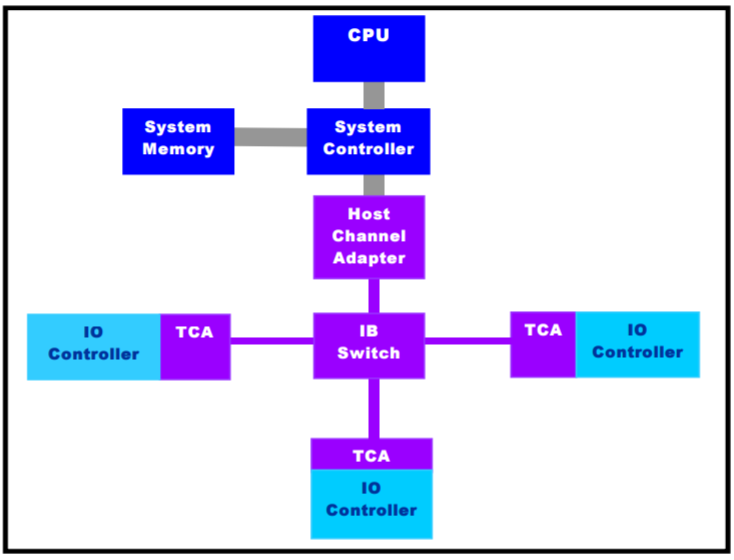


Рис 4. Элементы InfiniBand Architecture.

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 вендоров OS 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 сервсиных 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 сконфигурены, чтобы перенаправлять одноадрессные (unicast) пакеты (к одному получателю) или многоадрессные (multicast) пакеты (адресуемые к нескольким устройствам).

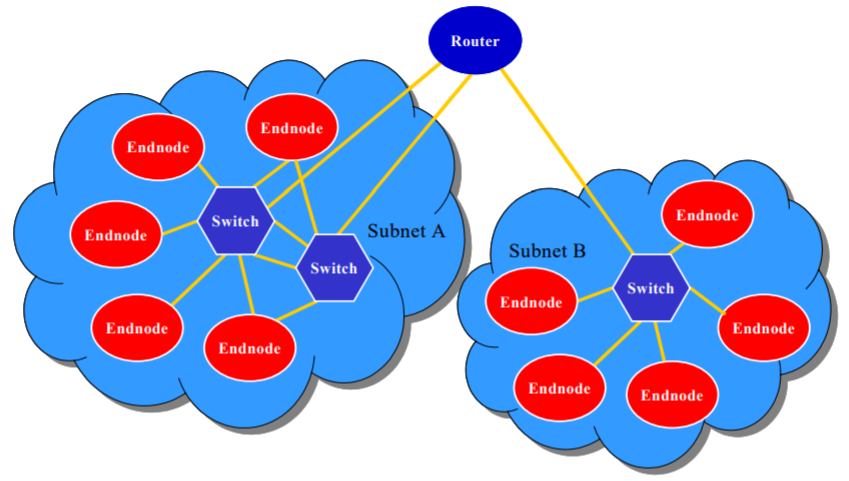


Рис 5. Связь компонент между двумя подсетями и внутри подсетей в InfiniBand Architecture.

* 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 адрессе 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 комлексной архитектурой, которая определяет оба электрические и механические характеристики для системы. Они вкючают 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 траспортировку 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 адресса отправителя и получателя пакета. Пакета передаются между подсеятми 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 для доставки пакетов в правильном порядке, разбиение пакета на части, мультеплексирование канала (несколько логических соединений, основанных на одном физическом) и предоставление различных видов транспорта(reliable connection (RC), reliable datagram (RD), unreliable connection (UC), unreliable datagram (UD), raw datagram (RD)). Транспортный уровень обрабатывает сегментацию данных на стороне отправителя и восстановаление 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), заисывая статут выполненной операции. InfiniBand Architecture предлагает значительное улучшение для транспортного уровня: все функциональности реализованы на аппаратном уровне. InfiniBand определяет несколько траспортных сервисов надежности отправки данных. Таблица 1 описывает каждый из поддерживаемых сервисов.

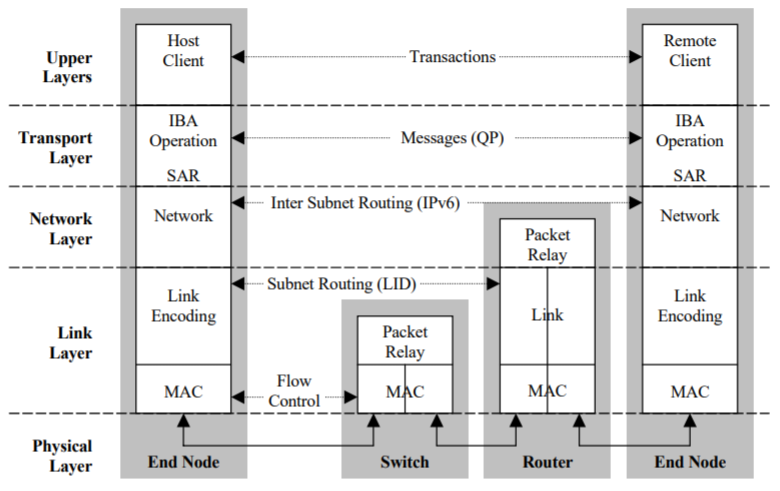


Рис 4. Уровни InfiniBand Architecture.

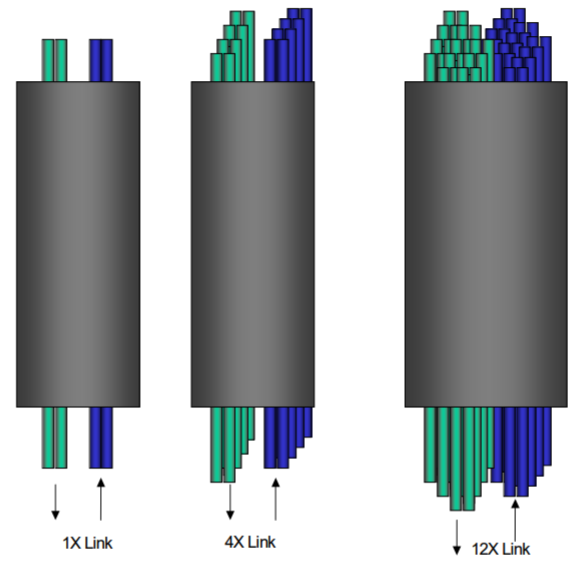


Рис 5. Три скоростных канала, определенных на физическом канале InfiniBand Architecture.

|  |  |
| --- | --- |
| Тип Сервиса | Описание |
| Reliable Connection | Подтверждающий пакеты, ориентированный на соединения |
| Reliable Datagram | Подтверждающий пакеты, мультеплексируемый |
| 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 уменьшенье накладных расходов системы на обработку сетевых данных посредством уменьшения количества копий, ассоцированных с передачей сообщений и невовлечения ядра операционной системы в критический путь отправки 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 один-к-одному соответствие 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 буффер, выделенный на удаленном узле. По аналогии, в случае RDMA Read операции, инициатор читает данные из пользовательского буффера, выделенного на удаленном узле.

InfiniBand был разработана с учетом VIA (Virtual Interface Architecture) архитектуры. VIA – это распределенная техноолгия передачи сообщений, которая является аппаратнонезависимой и совместима с текущими сетевыми высокоскоростными картами, предаставляет 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 адаптором из очереди отправки и отправляется на удаленное устройство. Когда удаленное устройство отвечает, канальный адаптер возвращает статус клиенту, через очередб готовности (CQ - Completion Queue).

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

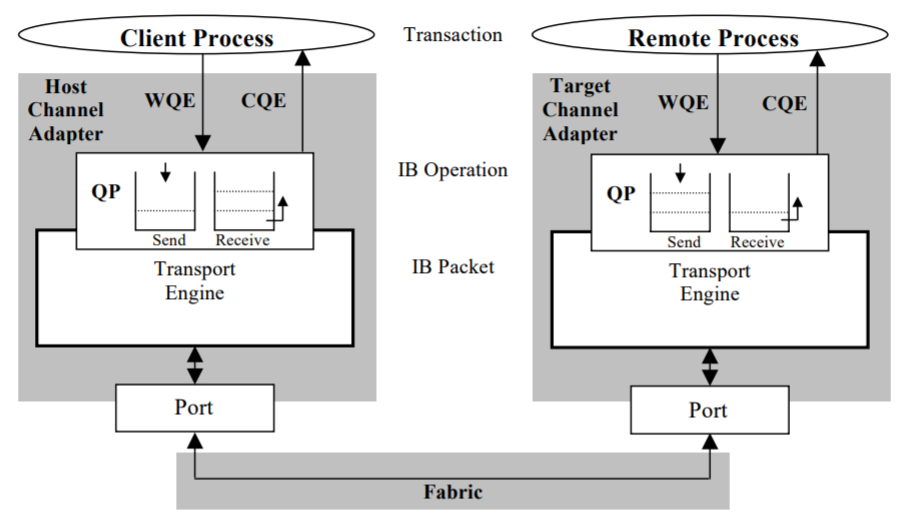


Рис 6. Коммникационный стэк InfiniBand.

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 Cray Aries высокоскоростных сетевых карт. 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. Интеграция фабрики использует преимущества подсистем обработки, кэширования и памяти, а также локальность инфраструктуры связи, обеспечивая быстрое внедрение аппаратных нововведений.

Первые реализации продуктов, основанных на Intel® OPA, фокусируются на суперкомпьютерные технологии, т.е. на выскопроизводительные вычислительные среды. Однако, Intel® OPA обычно применяется к классам датацентров, требующих масштабируемых и тесно связанных ресурсов CPU, памяти и хранилищей данных.

Intel® OPA определяет 1 и 2 уровни модели OSI, которые обеспечивают сетевое соединение между элементами для энергоэффективных суперкомпьютерых 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 Intel® Atom™ процессоры).

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 потвторной отправкой (которые используются в традиционных сетях). Сетевая адресая 2 уровня модели OSI расширена для учета систем с более чем 10 миллионов конечных точек соединения (endpoint), что позволяет использовать выскоую масштабируемость для датацентров на годы вперед. Чтобы обеспечить поддержку более широкого числа тополгий, Intel® OPA предоставляет механизмы для пакетов для изменения виртульных полос (virtual lanes - VLs) по мере и прохождения через фабрику. В дополнение, высокоприоритетные пакеты способны удерживать пакеты с меньшим приоритетов 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 фабриков через HFI. HFI – это мост между процессором на узле и фабрикой. Минимально, HFI состоит из логики, которая необходима, чтобы реализовать физический и канальный уровени 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, нгеобходимую, 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 вклоючает в себя настройку таблиц пересылки 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, коммутаторыми) в фабрике через MA, которые ассоциируются с теми устройствами. Основная FM в модели избыточности выбирается во время процесса инициализации фабрики. Основная FM отвечает за:

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 буфферы на специальной VL (VL15). Эта специальная VL для управления может быть сконфигурирована, чтобы функционировать с вклоюченным или выключенным управлением потока. Без управлением потока, пакеты управления будут откинуты, если ресурсы очереди не доступны на порту. Двухсторонние протоколы, обеспечивающие надежность, используются для обнаружения откинутых пакетов.

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

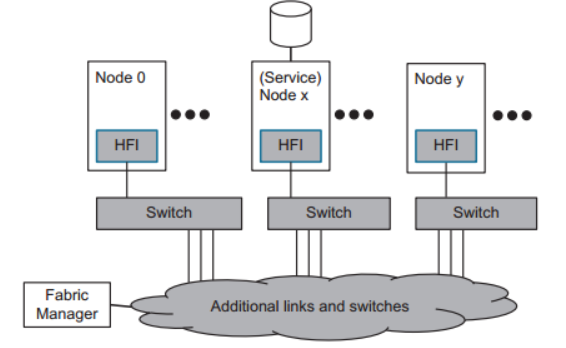


Рис 7. Элементы Intel® Omni-Path Architecture.

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 появлись два конкурирующим между сообой интерфейса - OFI (Open Fabric Interfaces)[2], разработанный Open Fabric Alliance, и UCX (Unified Communication X)[3], разработанный OpenUCX сообществом. Современные реализации MPI переходят на использованние 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/>.