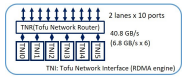
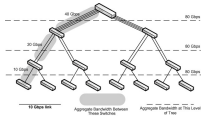
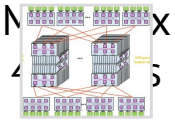
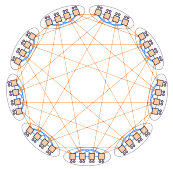
Курс «Архитектурно-ориентированная оптимизация кода»

Масштабируемость интерфейса MPI и его реализаций

Михаил Курносов

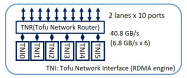
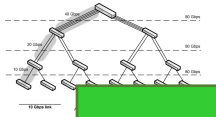
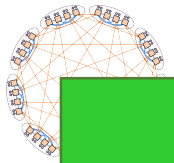
High-Performance Computing Systems

Top500 (#58, Nov 2021)

#	System	HPL Rmax, PFLOPS	HPCG, PFLOPS	Interconnection Network (topological levels)				
1	**Fugaku** Fujitsu **7 630 848 cores**	442.0	16.0	 **Tofu interconnect D** *6D mesh/torus topology* 158 976 nodes			**Network on Chip** (ring bus) 4 CMG 48 + 2 cores	**Shared memory** Core Mem Group, NUMA node 12 cores Fujitsu A64FX ARMv8.2
2	**Summit** IBM Power System AC922 2 414 592 cores	148.6	2.9	 **Mellanox EDR 100G** *Non-blocking fat tree* 4 608 nodes (2 x Power 9, 6 x NVIDIA Volta V100)			**X-BUS** 2 x Power 9 3 x NVIDIA V100	**Shared memory** 22 cores IBM Power9 3 x NVIDIA V100
4	**Sunway TaihuLight** **41 932 800** cores	93	0.48	**Switch network** 	**Supernode network** fully connected	**Sunway Network** PCIe 3.0 107 520 nodes	**Network on Chip** 6 core groups 390 cores	**Shared memory** 1 MPE + 64 CPE (mesh 8x8, RISC)
5	**Perlmutter** HPE CRAY EX235 761 856 cores	70.9	1.9	 **Slingshot** *Dragonfly* > 1 536 nodes			**AMD Infinity Fabric** NUMA-nodes	**Shared memory** 64 cores AMD Milan 3 x NVIDA A100

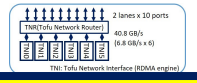
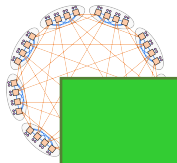
High-Performance Computing Systems

Top500 (#58, Nov 2021)

#	System	HPL Rmax, PFLOPS	HPCG, PFLOPS	Interconnection Network (topological levels)				
1	**Fugaku** Fujitsu **7 630 848 cores**	442.0	16.0	 **Tofu interconnect D** **MPI**: Fujits MPI (Open MPI), RIKEN-MPICH, XMP			**Network on Chip** (ring bus) **Threading**: OpenMP	**Shared memory** Core Mem Group, NUMA node ujits **Vectorization** SVE 512 bit 2
2	**Summit** IBM Power System AC922 2 414 592 cores	148.6	2.9	 **Mellanox EDR 100G** *Non-blocking fat tree* (2 /100) **MPI**: IBM Spectrum (Open MPI)			**X-BUS** 2 x Power 9 **Threading**: OpenMP, CUDA	**Shared memory** 22 cores I 3 x **Vectorization** AltiVec
4	**Sunway TaihuLight** **41 932 800** cores	93	0.48	**Switch network** N x **MPI**: MVAPICH	**Supernode network** fully	**Sunway Network** PCIe 3.0 107 520 nodes	**Network on Chip** 6 core 3 **Threading**	**Shared memory** 1 MPE + 64 CPE (m **Vectorization**
5	**Perlmutter** HPE CRAY EX235 761 856 cores	70.9	1.9	 **Slingshot** **MPI**: Cray MPI (MPICH)			**AMD Infinity** N **Threading**: OpenMP, CUDA	**Shared memory** c 3 x **Vectorization** AVX

High-Performance Computing Systems

Top500 (#58, Nov 2021)

#	System	HPL Rmax, PFLOPS	HPCG, PFLOPS	Interconnection Network (topological levels)				
1	Fugaku 7 6						**Network on Chip** (ring bus) **Threading**: OpenMP	**Shared memory** Core Mem Group, NUMA node ujits 2 **Vectorization** SVE 512 bit
2	IBM 2 4						**X-BUS** 2 x Power 9 **Threading**: OpenMP, CUDA	**Shared memory** 22 cores I 3 x **Vectorization** AltiVec
4	**Sunway TaihuLight** **41 932 800** cores	93	0.48	**Switch network** N x 4 **MPI**: MVAPICH	**Supernode network** fully	**Sunway Network** PCIe 3.0 107 520 nodes	**Network on Chip** 6 core 3 **Threading**	**Shared memory** 1 MPE + 64 CPE (m **Vectorization**
5	**Perlmutter** HPE CRAY EX235 761 856 cores	70.9	1.9	 **Slingshot** **MPI**: Cray MPI (MPICH)			**AMD Infinity** N **Threading**: OpenMP, CUDA	**Shared memory** 64 co 3 x **Vectorization** AVX

2021 ACM Gordon Bell Prize

Closing the "Quantum Supremacy" Gap: Achieving Real-Time Simulation of a Random Quantum Circuit Using a New Sunway Supercomputer

(China, https://doi.org/10.1145/3458817.3487399)

MPI + Threading + Vectorization

High-Performance Computing Systems

Top500 (Nov 2022)

NOVEMBER 2022

			SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Frontier**	HPE Cray EX235a, AMD Opt 3rd Gen EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-10	DOE/SC/ORNL	USA	8,730,112	1,102.0	21.1
2	**Fugaku**	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
3	**LUMI**	HPE Cray EX235a, AMD Opt 3rd Gen EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-10	EuroHPC/CSC	Finland	2,174,976	304.2	5.82
4	**Leonardo**	Atos Bullsequana intelXeon (32C, 2.6 GHz), NVIDIA A100 quad-rail NVIDIA HDR100 Infiniband	EuroHPC/CINEC	Italy	1,463,616	174.7	5.61
5	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1

Frontier:

- x86 Cores: 591 872
- GPU Cores: 8 138 240
- Interconnect: Slingshot-11

2022 ACM Gordon Bell Prize (https://awards.acm.org/bell)

- Project "*Pushing the Frontier in the Design of Laser-Based Electron Accelerators with Groundbreaking Mesh-Refined Particle-In-Cell Simulations on Exascale-Class Supercomputers*" (Particle-in-Cell simulation)

top50.supercomputers.ru

26.09.2022

№	Название Место установки	Узлов Проц. Ускор.	Архитектура: кол-во узлов: конфигурация узла сеть: вычислительная / сервисная / транспортная	Rmax Rpeak (Тфлоп/с)	Разработчик *Область применения*
1	«Червоненкис» Яндекс, Москва	199 398 1592	199: CPU: 2x AMD EPYC 7702 , 1024 GB RAM Acc: 8x NVIDIA A100 HDR InfiniBand / нд / 100 Gigabit Ethernet	21530.0 29415.17	**Яндекс** **NVIDIA** *IT Services*
2	«Галушкин» Яндекс, Москва	136 272 1088	136: CPU: 2x AMD EPYC 7702 , 1024 GB RAM Acc: 8x NVIDIA A100 HDR InfiniBand / нд / 100 Gigabit Ethernet	16020.0 20636.1	**Яндекс** **NVIDIA** *IT Services*
3	«Ляпунов» Яндекс, Москва	137 274 1096	137: CPU: 2x AMD Epyc 7662, 512 GB RAM Acc: 8x NVIDIA A100 HDR InfiniBand / нд / 100 Gigabit Ethernet	12810.0 20029.19	
4	«Кристофари Нео» SberCloud (ООО «Облачные технологии») , СберБанк, Москва	99 198 792	99: CPU: 2x AMD EPYC 7742, 2048 GB RAM Acc: 8x NVIDIA A100 HDR InfiniBand / 10 Gigabit Ethernet / 200 Gigabit Ethernet	11950.0 14908.6	**NVIDIA** **SberCloud (ООО «Облачные технологии»)** *Облачный провайдер*
5	«Кристофари» SberCloud (ООО «Облачные технологии») , СберБанк, Москва	75 150 1200	75: NVIDIA DGX-2 CPU: 2x Intel Xeon Platinum 8168 24C 2.7GHz, 1536 GB RAM Acc: 16x NVIDIA Tesla V100 EDR Infiniband / 100 Gigabit Ethernet / 10 Gigabit Ethernet	6669.0 8789.76	**SberCloud (ООО «Облачные технологии»)** **NVIDIA** *Облачный провайдер*
6	«Ломоносов-2» Московский государственный университет имени М.В.Ломоносова, Москва	1696 1696 1856	1536: CPU: 1x Intel Xeon E5-2697v3, 64 GB RAM Acc: 1x NVIDIA Tesla K40M 160: CPU: 1x Intel Xeon Gold 6126, 96 GB RAM Acc: 2x NVIDIA Tesla P100 FDR Infiniband / Gigabit Ethernet / FDR Infiniband	2478.0 4946.79	**Т-Платформы** *Наука и образование*

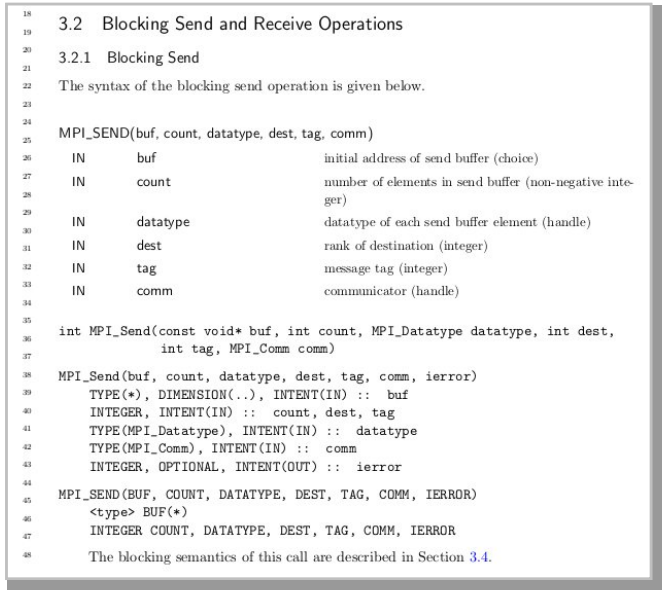
Развитие коммуникационных технологий для НРС в РФ

1. Коммуникационная сеть **«Ангара»**, разрабатываемая ОАО «НИЦЭВТ» (г. Москва) [1]
2. Коммуникационная сеть **СМПО-10G**, разрабатываемая ФГУП «РФЯЦ-ВНИИЭФ» (г. Саров) [2]
3. Коммуникационная сеть **«МВС-Экспресс»**, разрабатываемая ФГУП «НИИ Квант» и ИПМ РАН (г. Москва) [3]
4. Коммуникационные сети **СКИФ-Аврора**, Паутина, разработанные ИПС РАН (г. Переславль-Залесский) [4]

1. А.И. Слуцкин, А.С. Симонов, И.А. Жабин, Д.В. Макагон, Е.Л. Сыромятников. Разработка межузловой коммуникационной сети ЕС8430 «Ангара» для перспективных российских суперкомпьютеров // Успехи современной радиоэлектроники, 2012, №1, с. 6-10.
2. В.Г. Басалов, В.М. Вялухин. Адаптивная система маршрутизации для отечественной системы межпроцессорных обменов СМПО-10G // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов 2012. Вып.3. С. 64-70.
3. В.К. Левин, Б.Н. Четверушкин, Г.С. Елизаров, В.С. Горбунов, А.О. Лацис, В.В. Корнеев, А.А. Соколов, Д.В. Андрюшин, Ю.А. Климов. Коммуникационная сеть МВС-Экспресс // Информационные технологии и вычислительные системы, 2014, №1, с. 10–24.
4. С.М. Абрамов, В.Ф. Заднепровский, Е.П. Лилитко. Суперкомпьютеры «СКИФ. ряда 4», Информационные технологии и вычислительные системы, 2012, №1, с. 3–16.
5. Степаненко А.С. Мультипроцессорные среды суперЭВМ. Масштабирование эффективности. — 2016.

Стандарт MPI

- **MPI (Message Passing Interface)** – программный интерфейс передачи сообщений (API) для программирования вычислительных систем
- **Основные разделы MPI**
  - ❑ Point-to-point Communications
  - ❑ One-sided Communications
  - ❑ Collective communications
  - ❑ Derived datatypes
  - ❑ Process topologies and management



3.2 Blocking Send and Receive Operations

3.2.1 Blocking Send

The syntax of the blocking send operation is given below.

MPI_SEND(buf, count, datatype, dest, tag, comm)

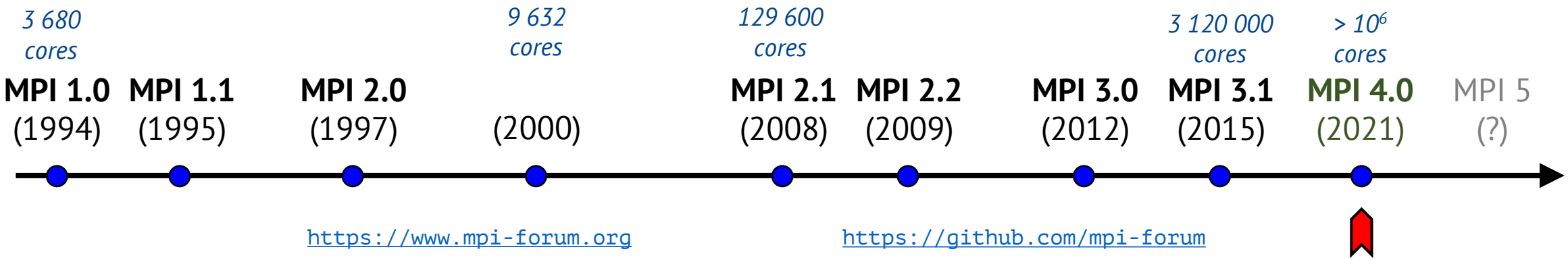
IN	buf	initial address of send buffer (choice)
IN	count	number of elements in send buffer (non-negative integer)
IN	datatype	datatype of each send buffer element (handle)
IN	dest	rank of destination (integer)
IN	tag	message tag (integer)
IN	comm	communicator (handle)

```
int MPI_Send(const void* buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)

MPI_Send(buf, count, datatype, dest, tag, comm, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN) :: buf
    INTEGER, INTENT(IN) :: count, dest, tag
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR
```

The blocking semantics of this call are described in Section 3.4.



https://www.mpi-forum.org

https://github.com/mpi-forum

Проблемы масштабируемости интерфейса MPI

- **Масштабируемости** (scalability) – способность системы сохранять заданный уровень производительность при росте числа p процессорных ядер
- **Масштабируемости MPI** – зависимости размера требуемой памяти и/или числа выполняемых процессорных/сетевых операций от количества процессов p растет медленнее линейной (в каждом процессе)
- **Плохо масштабируемая спецификация** (API) – приводит к реализациям, время и/или потребление памяти которых линейно ($\Omega(p)$) зависит от числа p процессов
- **Плохо масштабируемая реализация MPI** – время и/или потребление памяти линейно (или хуже) зависит от числа p процессов

Open MPI (BSD license)
https://www.open-mpi.org

- **Mellanox HPCX**: HCOLL, InfiniBand
- **IBM Spectrum MPI**: libcollectives, POWER9 and CUDA IPC
- **Fujitsu Post-K MPI**: Fujitsu A64FX, TofuD
- **Bull Open MPI**: hierarchical collectives, BXI

MPICH (BSD-like license)
https://www.mpich.org

- **MVAPICH:** InfiniBand
- **Intel MPI:** Intel Omni-Path
- **Tianhe2 TH-MPI**: GLEX
- **Cray MPI:** torus, SMP-aware
- **IBM PE MPI**

Проблемы масштабируемости интерфейса MPI

1. Нерегулярные коллективные операции (irregular collectives)

- **Нерегулярные коллективные операции** (irregular collectives) требуют передачи массивов длины p – размеры сообщений для каждого процесса

  - `MPI_GATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts[p], displs[p], recvtype, root, comm)` — **2p**
  - `MPI_SCATTERV(sendbuf, sendcounts[p], displs[p], sendtype, recvbuf, recvcount, recvtype, root, comm)`
  - `MPI_ALLGATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts[p], displs[p], recvtype, comm)`
  - `MPI_ALLTOALLV(sendbuf, sendcounts[p], sdispls[p], sendtype, recvbuf, recvcounts[p], rdispls[p], recvtype, comm)` — **4p**
  - `MPI_ALLTOALLW(sendbuf, sendcounts[p], sdispls[p], sendtypes[p], recvbuf, recvcounts[p], rdispls[p], recvtypes[p], comm)` — **6p**

- При p = 2^20 каждый процесс потребляет порядка 4 MiB только для передачи параметров; обработка массивов требует порядка $\Omega(p)$ операций

Проблемы масштабируемости интерфейса MPI

1. Нерегулярные коллективные операции (irregular collectives)

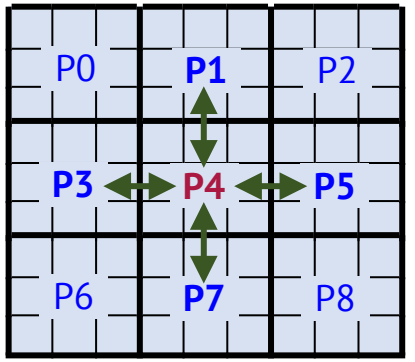
- Используются (в том числе) для реализации обменов по фиксированному шаблону (stencil code, PETSc)
- В Alltoallv counts[i] = 0 для процессов, не участвующих в обменах

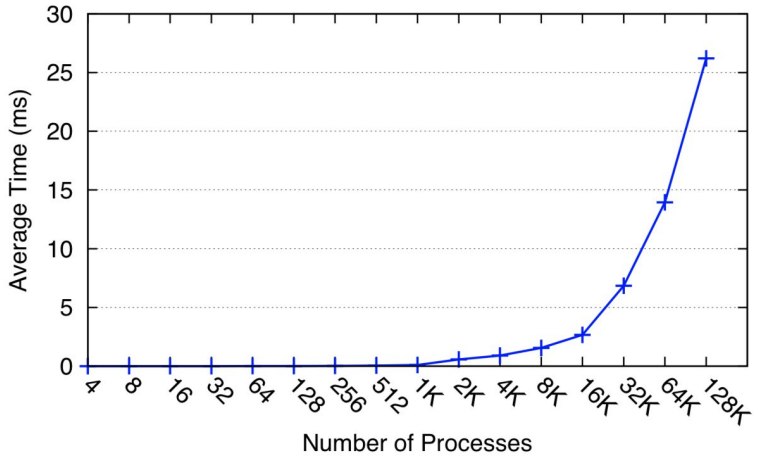
```
MPI_ALLTOALLV(
      sendbuf, sendcounts[p], sdispls[p], sendtype,
      recvbuf, recvcounts[p], rdispls[p], recvtype, comm)
```

- Алгоритмы Alltoallv требуют $\Omega(p)$ операций для анализа массивов

Шаблон “крест”, процесс 4:

```
counts[] = {0, 10, 0, 10, 0,
            10, 0, 10, 0}
```





Время Alltoallv для сообщений нулевой длины (counts[] = 0), BlueGene/P [Balaji, 2010]

Альтернативы

- Использовать Send/Recv для реализации обменов с соседними процессами (PETSc)
- MPI 3.1 Neighborhood Collectives (2015)

```
MPI_NEIGHBOR_ALLGATHER(
      sendbuf, sendcount, sendtype, recvbuf,
      recvcount, recvtype, comm)
```

Проблемы масштабируемости интерфейса MPI

2. Виртуальные топологии (virtual topologies)

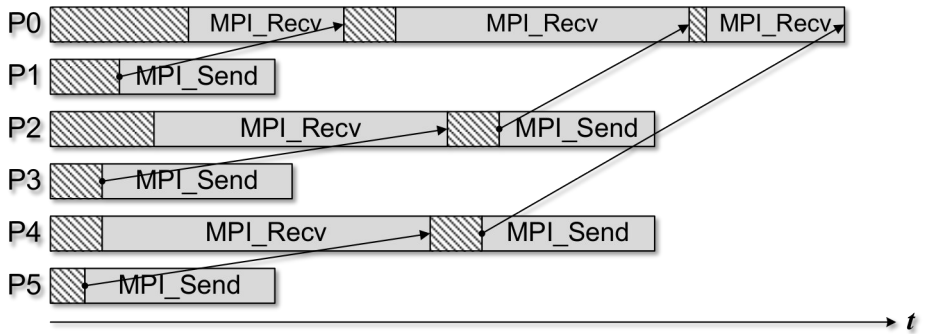
- Позволяют задать шаблон информационных обменов между MPI-процессами
- Библиотека может выполнить переупорядочивание процессов для сокращения времени информационных взаимодействий (process mapping)
  - **MPI_CART_CREATE**(comm_old, ndims, dims, periods, reorder, comm_cart)
  - **MPI_GRAPH_CREATE**(comm_old, nnodes, **index**[], **edges**[], reorder, comm_graph)
    потребление памяти каждым процессом: $\Omega(p+e)$, $O(p^2)$
- **Альтернативы**
- MPI 2.2 (2009): Механизм задания графа в распределенном виде, каждый процесс задет свою локальную окрестность

```
MPI_DIST_GRAPH_CREATE(comm_old, n, sources[], degrees[], destinations[],
                      weights[], info, reorder, comm_dist_graph)
```

Проблемы масштабируемости интерфейса MPI

3. Коллективные операции (collective operations)

- Время выполнения **блокирующих коллективных операций** (MPI_Bcast, MPI_Allreduce, MPI_Allgather) в значительной степени зависит от сбалансированности моментов их вызовов в процессах MPI-программы
- Причины
  - Неравномерная загрузка процессов
  - “Шум” операционной системы и сети (system noise)



- **Альтернативы**
- MPI 3.1 (2015): неблокирующие коллективные операций (nonblocking collectives)

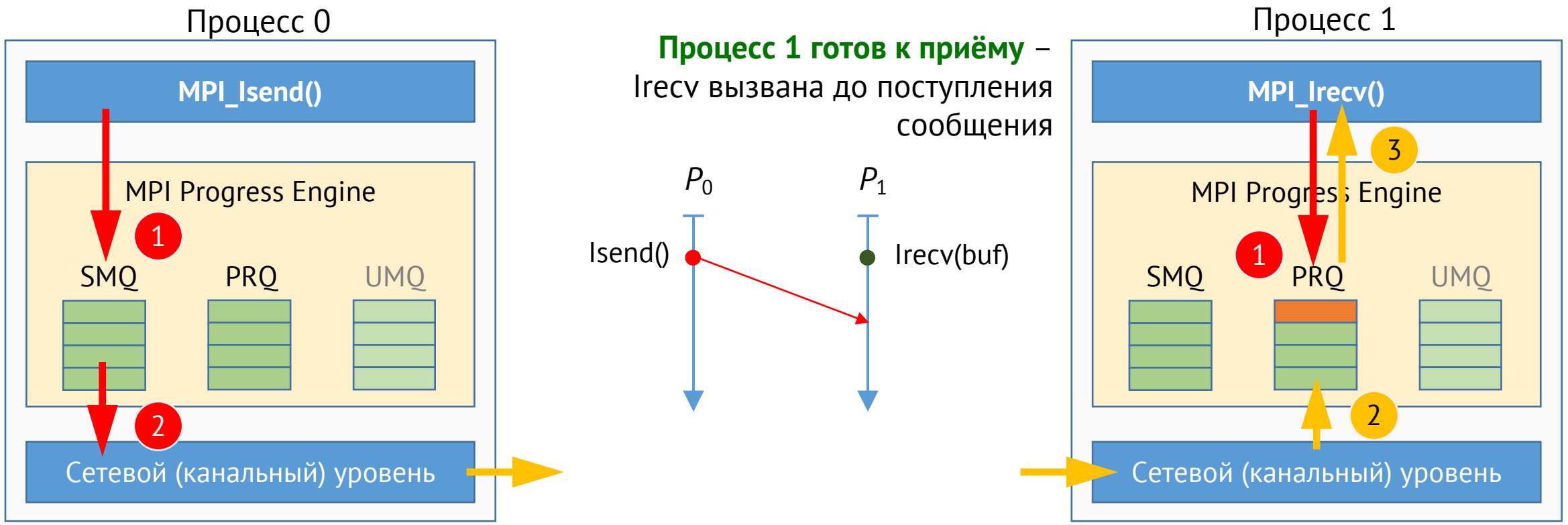
```
MPI_Ibcast(buf, …, req) … Computations … MPI_Test(req) … MPI_Wait(req)
```

- Аппаратная реализация NBC: NVIDIA (Mellanox) HPC-X

[*] P. M. Widener, S. Levy, K. B. Ferreira, T. Hoefler. ***On noise and the performance benefit of nonblocking collectives //*** The Int. J. of High Performance Computing Applications. 2016. Vol 30, Nr. 1, pp. 121-133

Проблемы масштабируемости реализаций MPI

4. Двусторонние обмены (point-to-point collectives)



Isend() + Wait()

1. Помещение сообщения в очередь SMQ (Send Message Queue)
2. Отправка сообщения через сетевой интерфейс

Irecv()

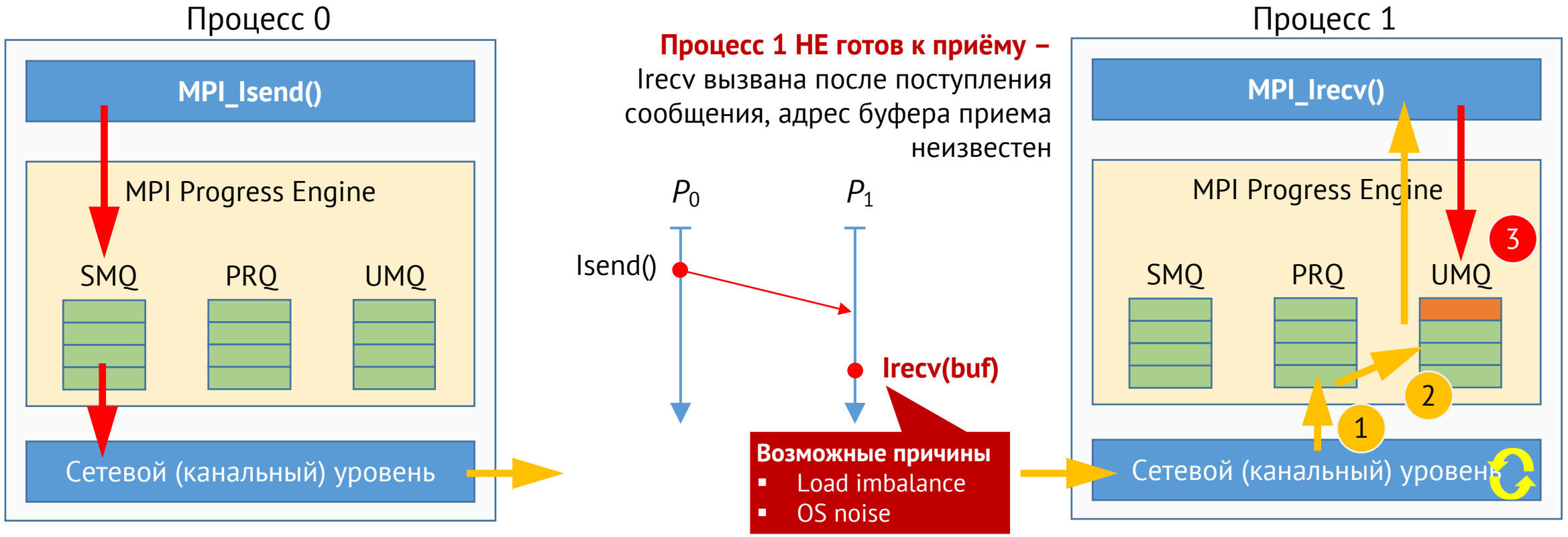
1. Помещение запроса в очередь PRQ (адрес буфера)

Прием сообщения из сети

1. Поступило сообщение, поиск в очереди PRQ
2. Копирование сообщения в буфер пользователя

Проблемы масштабируемости реализаций MPI

4. Двусторонние обмены (point-to-point collectives)



Isend() + Wait()

1. Помещение сообщения в очередь SMQ (Send Message Queue)
2. Отправка сообщения через сетевой интерфейс

Прием сообщения из сети:

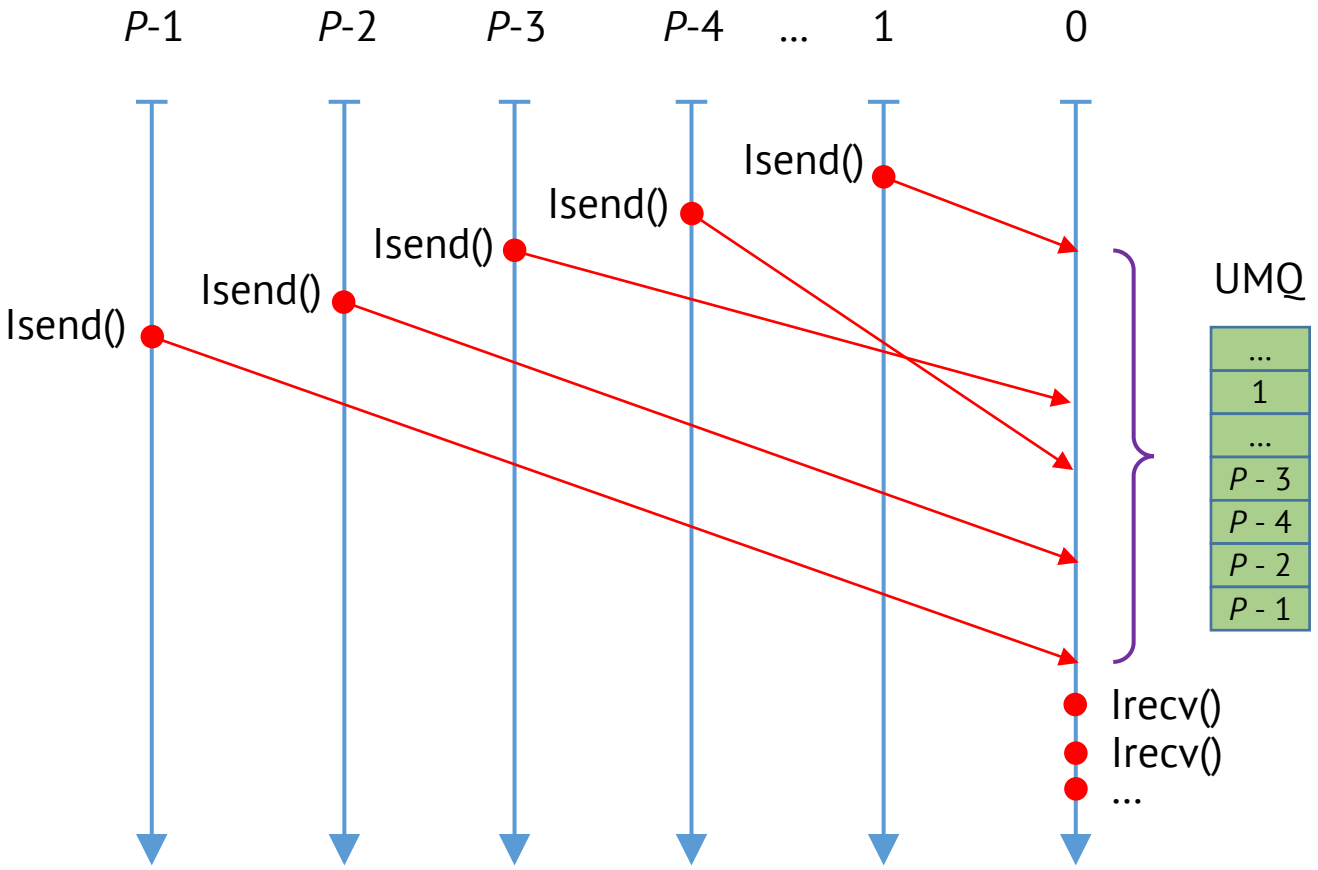
1. Поиск запроса на приём в PRQ
2. Если в PRQ нет запроса, сообщение помещается в UMQ – копируется во временный буфер (eager protocol vs. rendezvous)

Irecv(): поиск сообщения в очереди UMQ, копирование сообщения в буфер пользователя

Проблемы масштабируемости реализаций MPI

4. Двусторонние обмены (point-to-point collectives)

MPI_Gather (linear algorithm)



Худший случай –
корень 0 вызвал все Irecv после прихода сообщений от $p - 1$ процесса

- Длина UMQ – $O(p)$, необходимо $O(p)$ временных буферов
- Каждый Irecv в корне требует порядка $O(p)$ операций для поиска сообщения и копирований в буфер пользователя

- M. Flajslik, J. Dinan, K. D. Underwood. ***Mitigating MPI Message Matching Misery*** // *International Conf. on High Performance Computing, 2016.*
- K. Raffenetti et. al. ***Why is MPI so slow?: analyzing the fundamental limits in implementing MPI-3.1*** // *SC-2017*

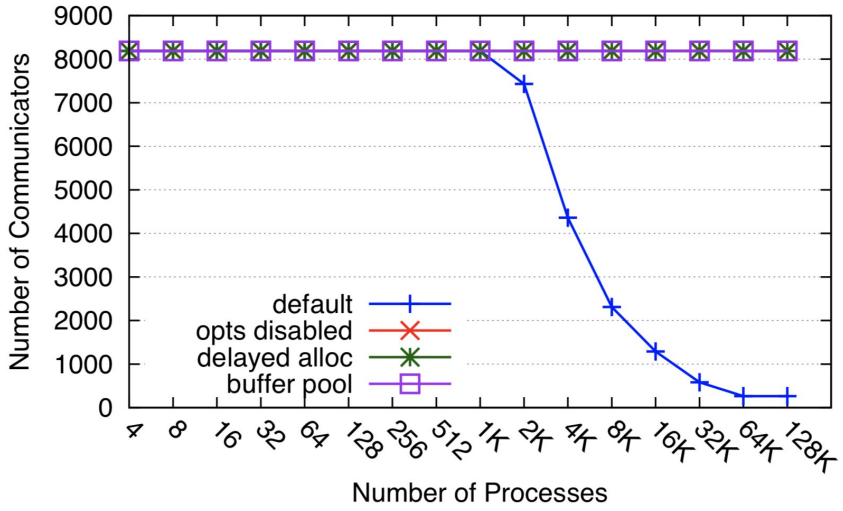
Проблемы масштабируемости реализаций MPI

5. Нумерация процессов (process mappings)

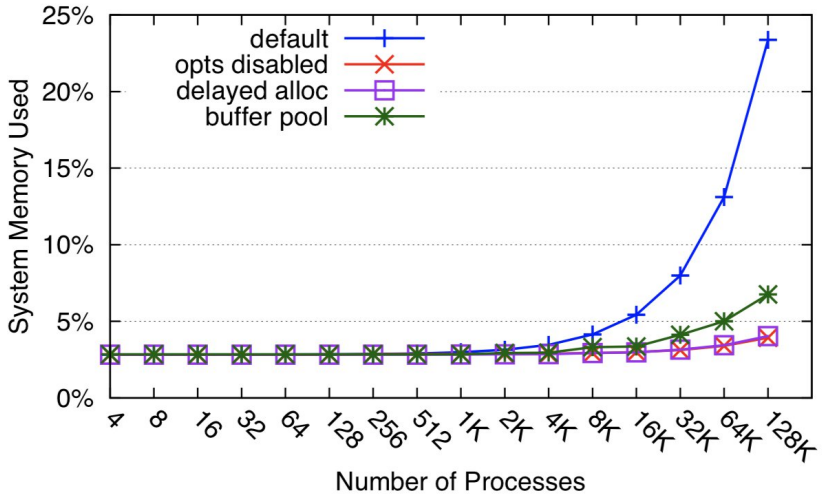
- Коммуникатор MPI содержит таблицу соответствия номеров процессов (ranks) идентификаторам процессоров вычислительной системы: $f: rank \rightarrow id$
- Основной подход (Open MPI, MPICH, MVAPICH) – массив длины p, обеспечивающий за время $O(1)$ поиск процессора по ранку; сложность по памяти $O(p)$

- `MPI_GROUP_UNION(group1, group2, newgroup)`
- `MPI_GROUP_INTERSECTION(group1, group2, newgroup)`
- `MPI_GROUP_DIFFERENCE(group1, group2, newgroup)`

- `MPI_COMM_CREATE(comm, group, newcomm)`
- `MPI_COMM_DUP(comm, newcomm)`
- `MPI_COMM_RANK(comm, rank)`



Количество коммуникаторов помещающихся в память узлов системы BG/P (MPI_Comm_dup) [Balaji, 2010]



Потребление памяти на BG/P при 32 вызовах MPI_Comm_dup [Balaji, 2010]

Проблемы масштабируемости реализаций MPI

5. Нумерация процессов (process mappings)

- Коммуникатор MPI содержит таблицу соответствия номеров процессов (ranks) идентификаторам процессоров вычислительной системы: $f: rank \rightarrow id$
- Основной подход (Open MPI, MPICH, MVAPICH) – массив длины p, обеспечивающий за время $O(1)$ поиск процессора по ранку; сложность по памяти $O(p)$

- `MPI_GROUP_UNION(group1, group2, newgroup)`
- `MPI_GROUP_INTERSECTION(group1, group2, newgroup)`
- `MPI_GROUP_DIFFERENCE(group1, group2, newgroup)`

- `MPI_COMM_CREATE(comm, group, newcomm)`
- `MPI_COMM_DUP(comm, newcomm)`
- `MPI_COMM_RANK(comm, rank)`

Альтернативы

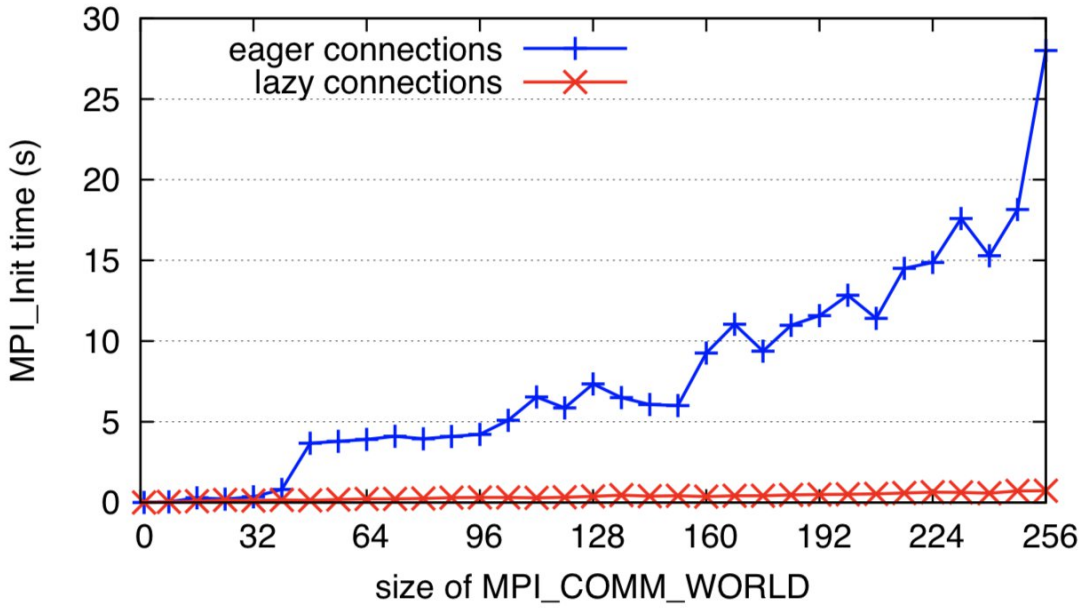
- Аналитическое задание 1-1 отображения $f: rank \rightarrow id$ для специальных случаев: $f(rank) = (rank * a + b) \% p$

- J. L. Traff. ***Compact and efficient implementation of the MPI group operations*** // 17th European MPI Users' Group Meeting, 2010, Vol 6305 of LNCS, pages 170–178.
- H. Kamal, S. M. Mirtaheri, and A. Wagner. ***Scalability of communicators and groups in MPI*** // In ACM International Symposium on High Performance Distributed Computing (HPDC), 2010.

Проблемы масштабируемости реализаций MPI

6. Масштабируемость MPI_Init

- В случае использования MPI с протоколами ориентированными на установление соединения каждый процесс может устанавливать в **MPI_Init** порядка $\Omega(p)$ связей
- **Жадный вариант** (eager) – предварительно в MPI_Init устанавливаются соединения со всеми процессами
- **Отложенная инициализация** (lazy) – соединение устанавливается, только при первом взаимодействии пары процессов
- Отложенная инициализация увеличивает время выполнения первого вызова коммуникационной операции



Время выполнения MPI_Init
(кластер на базе 8 ядерных узлов, сеть TCP/IP)

Проблемы масштабируемости реализаций MPI

7. Масштабируемость коллективных операций

Flat point-to-point based (send/recv)

Only number of processes and message size are known

- **One-to-all (Broadcast, Scatter)**
  - Binary tree, Binomial tree, *k*-nomial tree $O(\log(p))$
  - Flat tree $O(p)$, *k*-chain $O(p)$, pipeline
  - Scatter binomial tree + allgather recursive doubling
- **All-to-one (Gather, Reduce, Scan)**
  - Binomial tree, *k*-nomial tree, binary tree,
  - *k*-chain, pipeline, linear
  - Rabesifner's algorithm
- **All-to-all (Allgather, Alltoall, Allreduce, Reduce_scatter, Reduce_scatter_block)**
  - Bruck, recursive doubling, recursive halving algorithm, neighbor exchange
  - Linear, ring, gather + scatter
  - Rabenseifner's algorithm, Butterfly

Hardware-accelerated algorithms

- NVIDIA/Mellanox SHARP (Allreduce, Reduce, Barrier, Bcast)
- Mellanox multicast (Bcast)
- IBM BG tree network (Bcast)

Topology-aware algorithms

Topology of network is known (routes, distances, process placement)

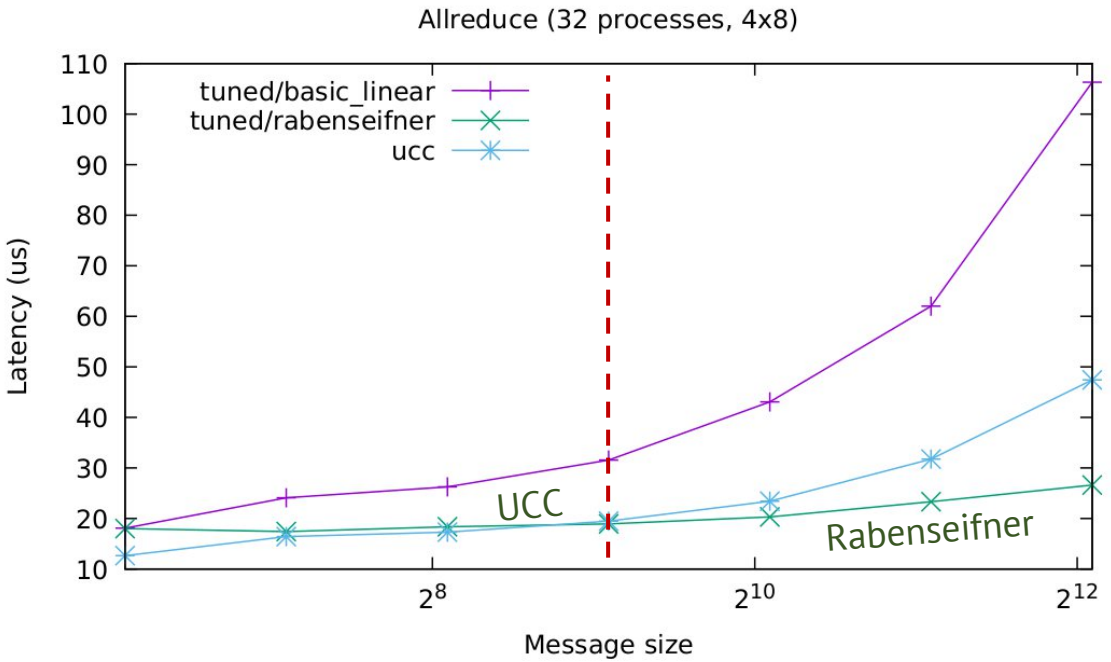
- Algorithms for Torus/Dragonfly networks
- Algorithms for Fat-tree topology
- Hierarchical collectives (SMP-aware): network + shared memory (intra-node)

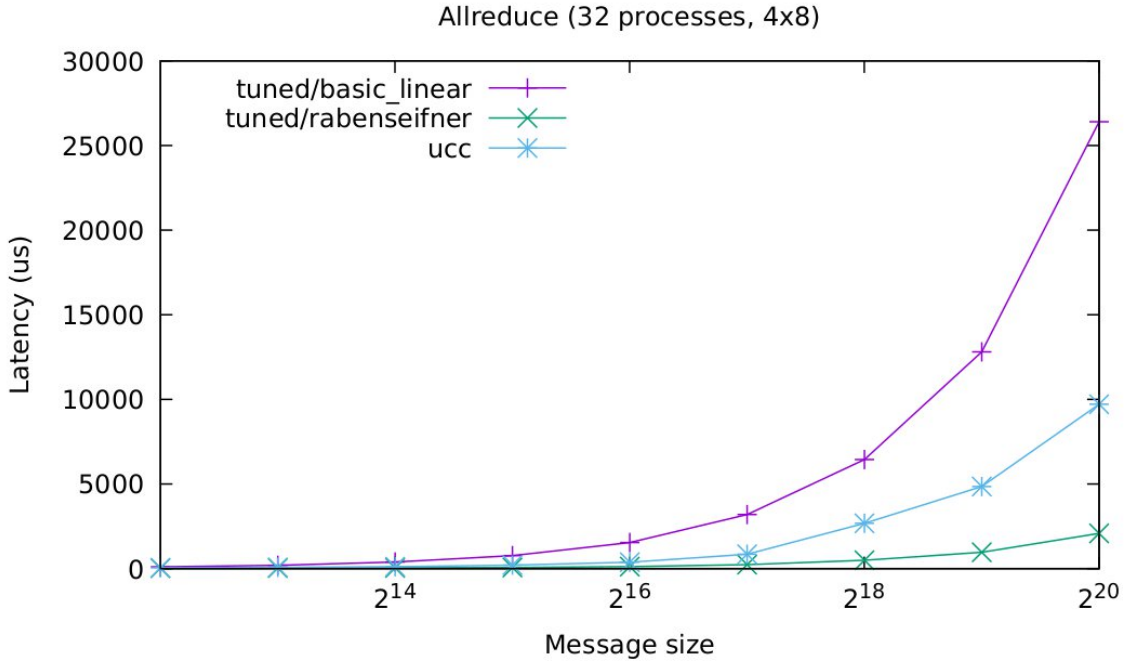
- MPI 4.0 Standard defines 17 (51) collective ops
- > 50 algorithms (MPICH/MVAPICH, Open MPI, Intel MPI, HPC-X)

Выбор алгоритма коллективной операции Allreduce

```
$ cat task.job
#SBATCH --nodes=4 --ntasks-per-node=8

mpiexec --mca coll_tuned_priority 100 \
        --mca coll_tuned_verbose 0 \
        --mca coll_tuned_use_dynamic_rules true \
        --mca coll_tuned_allreduce_algorithm rabenseifner \
        ./osu_allreduce
```





HPC Communication Software Stack

Applications

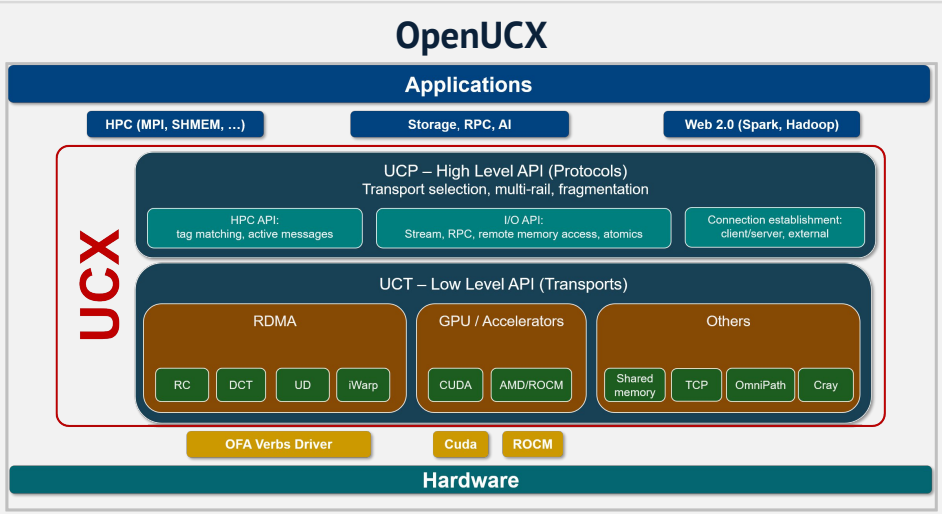
HPC Applications | AI Applications | BigData Workloads

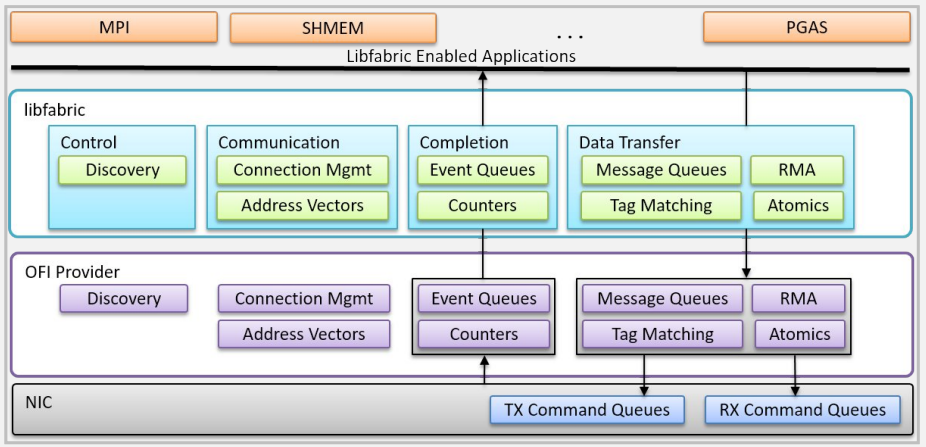
MPI, SHMEM libraries

NVIDIA HPC-X | IBM Spectrum MPI | Fujitsu MPI | MVAPICH | Intel MPI | Cray MPI

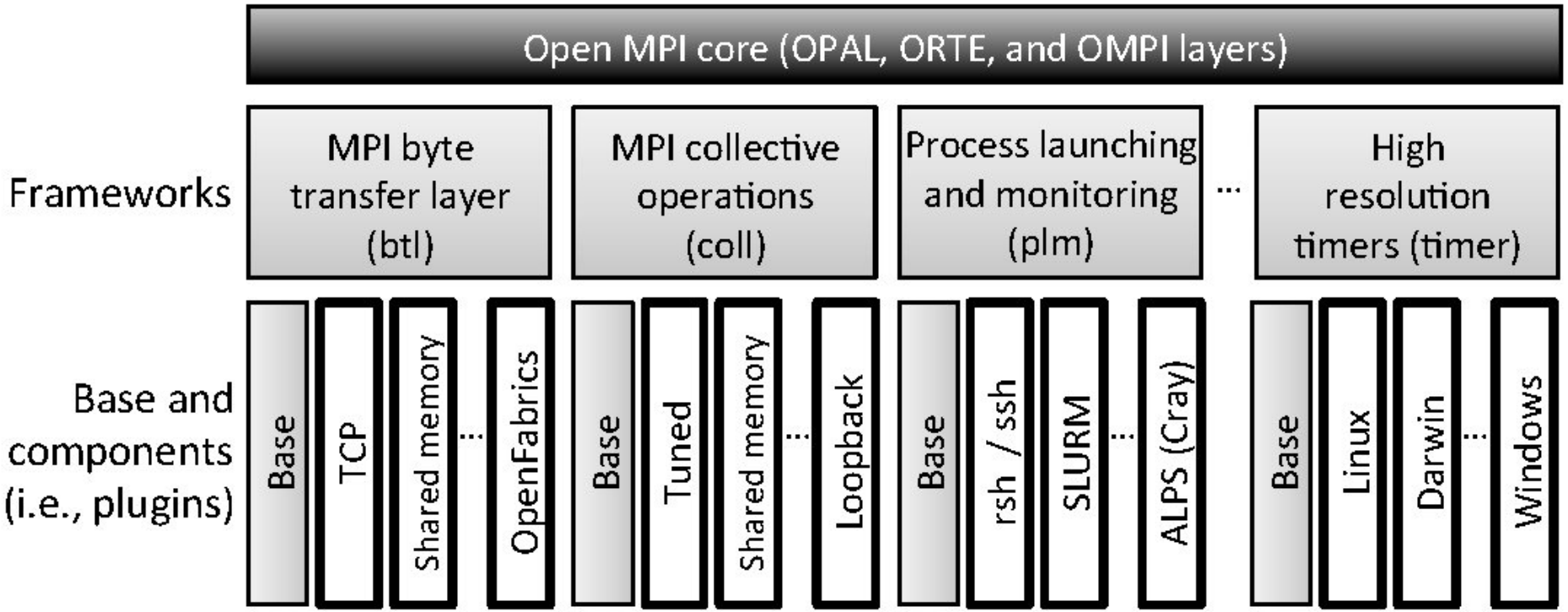
Open MPI | **MPICH**

Transport inter-node, shared-memory, GPU memory





Open MPI



- **btl:** MPI point-to-point Byte Transfer Layer (MPI point-to-point messages on some types of networks)
- **coll:** MPI collective algorithms
- **pml:** MPI point-to-point management layer: UCX (ob1, ucx)

`https://www.open-mpi.org/faq/?category=tuning`

Open MPI

```
$ ompi_info --param btl all --level 9

MCA btl: self (MCA v2.1.0, API v3.1.0, Component v4.1.4)
MCA btl: vader (MCA v2.1.0, API v3.1.0, Component v4.1.4)
MCA btl: tcp (MCA v2.1.0, API v3.1.0, Component v4.1.4)
MCA btl: ofi (MCA v2.1.0, API v3.1.0, Component v4.1.4)

MCA (-) btl vader: parameter "btl_vader_segment_size"
    (current value: "4194304", data source: default, level: 5 tuner/detail, type: int)
    Maximum size of all shared memory buffers (default: 4M)

MCA (-) btl vader: parameter "btl_vader_single_copy_mechanism"
    (current value: "cma", data source: default, level: 3 user/all, type: int)
    Single copy mechanism to use (defaults to best available)
    Valid values: 1:"cma", 4:"emulated", 3:"none"

MCA (-) btl ofi: parameter "btl_ofi_eager_limit"
    (current value: "0", data source: default, level: 4 tuner/basic, type: size_t)
    Maximum size (in bytes, including header) of "short" messages (must be >= 1).

...
```

Open MPI

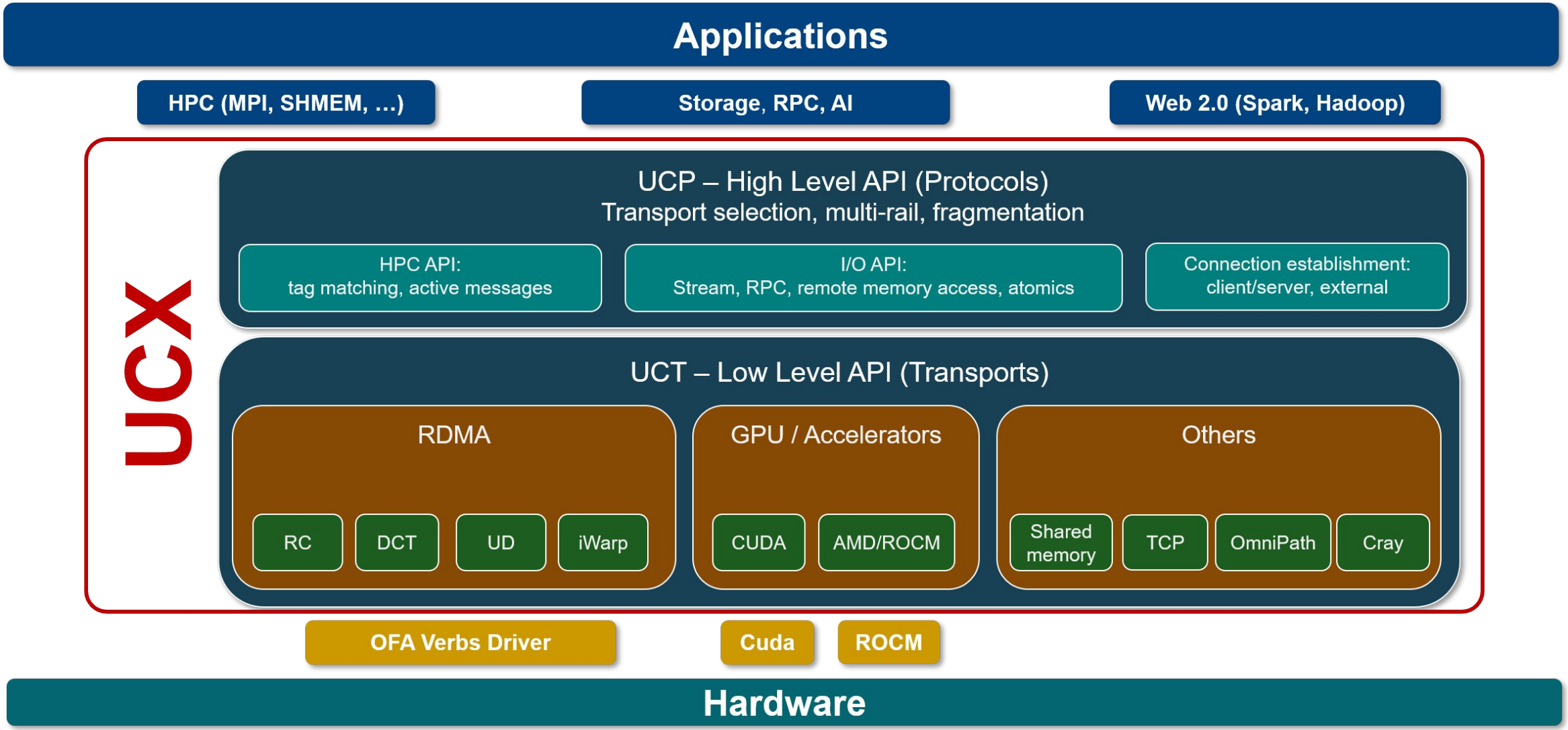
```
$ ompi_info --param pml all --level 9

MCA pml: v (MCA v2.1.0, API v2.0.0, Component v4.1.4)
MCA pml: monitoring (MCA v2.1.0, API v2.0.0, Component v4.1.4)
MCA pml: cm (MCA v2.1.0, API v2.0.0, Component v4.1.4)
MCA pml: ucx (MCA v2.1.0, API v2.0.0, Component v4.1.4)
MCA pml: ob1 (MCA v2.1.0, API v2.0.0, Component v4.1.4)
```

```
$ mpirun -np 4 -hostfile ./hostfile --mca rmaps_rank_file_path rankfile ./mpi_app
```

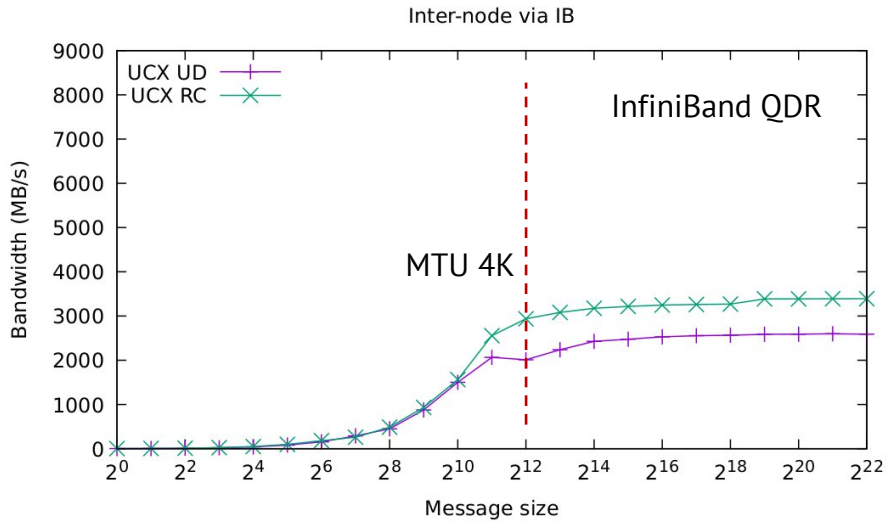
```
$ mpirun --mca pml ucx --mca btl ^vader,tcp,openib,uct \
         -x UCX_NET_DEVICES=mlx5_0:1 ./mpi_app
```

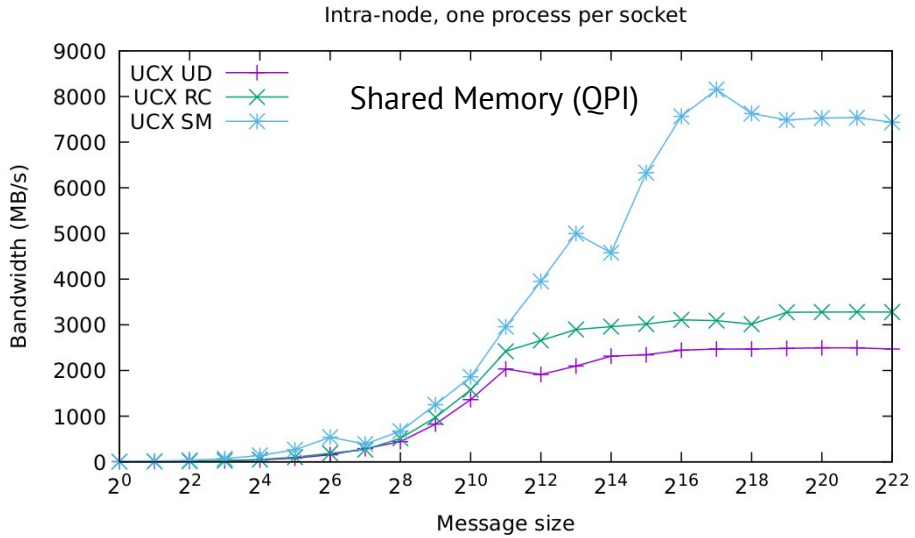
OpenUCX


Applications
HPC (MPI, SHMEM, …)
Storage, RPC, AI
Web 2.0 (Spark, Hadoop)
UCX
UCP – High Level API (Protocols)
Transport selection, multi-rail, fragmentation
HPC API:
tag matching, active messages
I/O API:
Stream, RPC, remote memory access, atomics
Connection establishment:
client/server, external
UCT – Low Level API (Transports)
RDMA
RC
DCT
UD
iWarp
GPU / Accelerators
CUDA
AMD/ROCM
Others
Shared memory
TCP
OmniPath
Cray
OFA Verbs Driver
Cuda
ROCM
Hardware

Open MPI 4.1.4 + OpenUCX 1.13.1

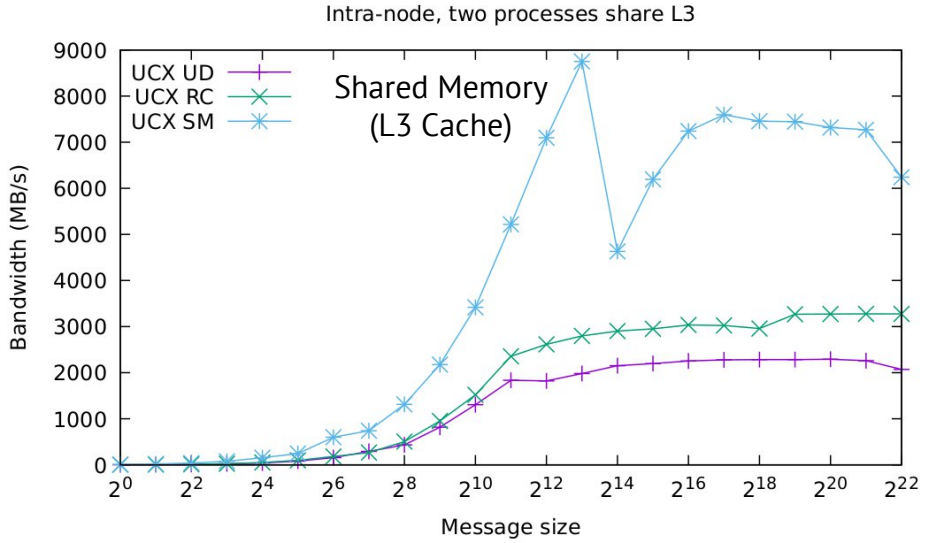
InfiniBand QDR (Mellanox MT26428 + InfiniScale IV IS5030 QDR)






```
# InfiniBand
# Два узла, один процесс на узел
$ mpiexec -mca pml ucx --mca btl ^vader,tcp,openib,uct \
          -x UCX_NET_DEVICES=mlx4_0:1 \
          -x UCX_TLS=rc ./osu_bw
```

```
# Intra-node, QPI (shared memory)
# Один узел, один процесс на процессор (socket)
$ mpiexec -mca pml ucx --mca btl ^vader,tcp,openib,uct \
          -x UCX_NET_DEVICES=mlx4_0:1 -x UCX_TLS=sm \
          --map-by socket --display-map ./osu_bw
```



Open MPI Collective Framework

- Каждый компонент реализует все или часть коллективных операций
- Компоненты имеют приоритет, задаваемый пользователем

openmpi/ompi/mca/coll

```
adapt
base          # Algorithms library (used by tuned)
basic
cuda
fca           # Mellanox FCA
han           # Hierarchical collectives (intra-npde + inter-node)
hcoll         # Mellanox HCOLL
inter
libnbc        # Nonblocking collectives
monitoring
portals4      # Portals API (Sandia NL)
self
sm            # Shared Memory collectives (depricated)
sync
tuned         # Основной компонент с логикой выбора алгоритмов из base/
ucc           # Open UCX collectives
```

Open MPI coll/tuned

$ ompi_info --param coll tuned --level 9

```
MCA coll tuned: parameter "coll_tuned_priority" (current value: "30", data source: default, level: 6 tuner/all, type: int)
  Priority of the tuned coll component

MCA coll tuned: parameter "coll_tuned_use_dynamic_rules"
  (current value: "false", data source: default, level: 6 tuner/all, type: bool)
  Switch used to decide if we use static (compiled/if statements) or dynamic (built at runtime) decision function rules
  Valid values: 0: f|false|disabled|no|n, 1: t|true|enabled|yes|y

MCA coll tuned: parameter "coll_tuned_dynamic_rules_filename"
  (current value: "", data source: default, level: 6 tuner/all, type: string)
  Filename of configuration file that contains the dynamic (@runtime) decision function rules

MCA coll tuned: parameter "coll_tuned_allreduce_algorithm"
  (current value: "ignore", data source: default, level: 5 tuner/detail, type: int)
  Which allreduce algorithm is used. Can be locked down to any of: 0 ignore, 1 basic linear, 2 nonoverlapping (tuned reduce
+ tuned bcast), 3 recursive doubling, 4 ring, 5 segmented ring. Only relevant if coll_tuned_use_dynamic_rules is true.
  Valid values: 0:"ignore", 1:"basic_linear", 2:"nonoverlapping", 3:"recursive_doubling", 4:"ring", 5:"segmented_ring",
                6:"rabenseifner"

MCA coll tuned: parameter "coll_tuned_allreduce_algorithm_segmentsize"
  (current value: "0", data source: default, level: 5 tuner/detail, type: int)
  Segment size in bytes used by default for allreduce algorithms.
  Only has meaning if algorithm is forced and supports segmenting. 0 bytes means no segmentation.

MCA coll tuned: parameter "coll_tuned_allreduce_algorithm_tree_fanout"
  (current value: "4", data source: default, level: 5 tuner/detail, type: int)
  Fanout for n-tree used for allreduce algorithms.
  Only has meaning if algorithm is forced and supports n-tree topo based operation.
...
```

Open MPI coll/tuned

```
$ cat task.job
#SBATCH --nodes=4 --ntasks-per-node=8

mpiexec --mca coll_tuned_priority 100 \
        --mca coll_tuned_verbose 0 \
        --mca coll_tuned_use_dynamic_rules true \
        --mca coll_tuned_allreduce_algorithm rabenseifner \
        ./osu_allreduce
```