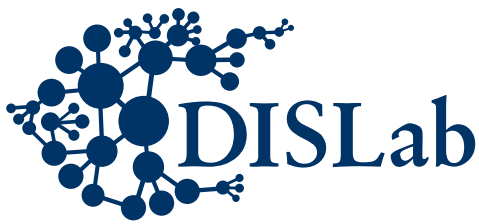
Расширение модели Charm++ для задач с мелкозернистым параллелизмом

А.С.Фролов

Лаборатория DISLab, АО «НИЦЭВТ»



«Технологии параллельной обработки больших графов» (GraphHPC-2016), Москва, 3 марта 2016 г.

Введение

Графовые задачи и HPC

- Потребности в высокоскоростной обработке больших графов:
  - Размеры представляющих интерес графов растут с каждым годом:
    - Интернет (WWW)

      На сентябрь 2015 – 50 миллиардов страниц[1]
    - Социальные медиа

      Facebook: 1.7 billion active users/month

      Twitter: 500 million tweets/day, 320 million users/month
    - Транспортные сети
    - Биоинформатика
    - Информационная безопасность

      Анализ TCP/IP траффика, лог-файлов и т.п.
  - Обработка графов в режиме offline и реальном времени
    - Размещение данных в оперативной памяти (in-memory)
    - Интенсивный трафик обмена сообщениями между узлами

[1] http://www.worldwidewebsize.com

Введение

Модель «Vertex-centric» (или « Think like a vertex ») и ее реализации

- $G = (V, E)$
  $Active(V)$ – множество активных вершин
  $F(v, S_v, M_{in}) \to (S_v, M_{out})$, где
  S_v – внутреннее состояние вершины v,
  M_{in} – входящие сообщения в вершину v,
  M_{out} – исходящие сообщения из вершины v
  while $Active(V) \neq \varnothing$
  forall $v \in Active(V)$
  $F(v, S_v, M_{in}) \to (S_v, M_{out})$

- Примеры реализаций VC-модели (в Big Data): Pregel, Apache Giraph, GraphLab, Kineograph, Trinity, Grace и др.

- **Возможно ли использовать VC-модель для обработки больших графов на суперкомпьютерах?**
  - Какая будет производительность в сравнении с “классическим” подходом (MPI + OpenMP)?
  - Какую вычислительную модель брать за основу?
  - Как это соотносится с проблемами “экзаскейла”? и др.

Обзор подходов к реализации графовых задач на суперкомпьютерах

MPI/Shmem + OpenMP/PThreads/TBB + CUDA/OpenCL

- Стандартные средства параллельного программирования
- Ориентированы на отражение архитектуры вычислительной системы
- Не отражают специфику графовых приложений (алгоритмов)
- Балансировка нагрузки требует предварительного анализа графа
- Оптимизации по агрегации сообщений, вершин выполняются вручную
- По сути – синхронная модель параллельного программирования, асинхронность может быть реализована, но с трудом
- Де факто – выбор практиков

The Graph 500 List

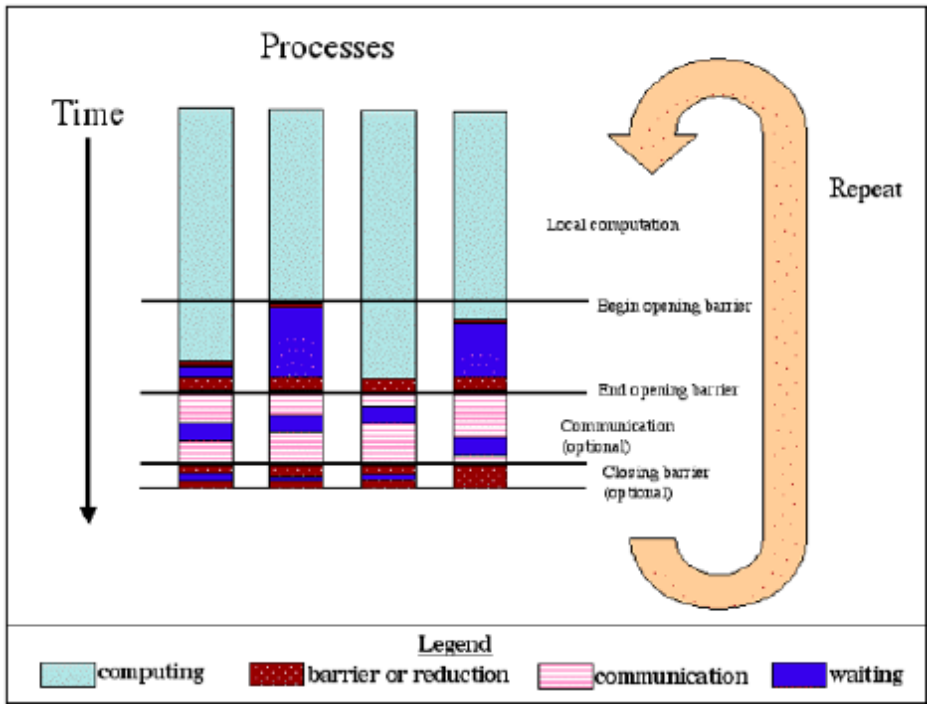
November 2015

No.	Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
1	1	K computer (Fujitsu - Custom)	RIKEN Advanced Institute for Computational Science (AICS)	82944	663552	40	38621.4
2	2	DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Lawrence Livermore National Laboratory	98304	1572864	41	23751
3	3	DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Argonne National Laboratory	49152	786432	40	14982
4	4	JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Forschungszentrum Juelich (FZJ)	16384	262144	38	5848
5	5	Fermi (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	CINECA	8192	131072	37	2567
6	6	Tianhe-2 (MilkyWay-2) (National University of Defense Technology - MPP)	Changsha, China	8192	196608	36	2061.48
7	7	Turing (IBM - BlueGene/Q, Power BQC 16C 1.60GHz)	CNRS/IDRIS-GENCI	4096	65536	36	1427
8	7	Blue Joule (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Science and Technology Facilities Council - Daresbury Laboratory	4096	65536	36	1427
9	7	DIRAC (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	University of Edinburgh	4096	65536	36	1427
10	7	Zumbrota (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	EDF R&D	4096	65536	36	1427

Обзор подходов к реализации графовых задач на суперкомпьютерах

Parallel Boost Graph Library (PBGL)

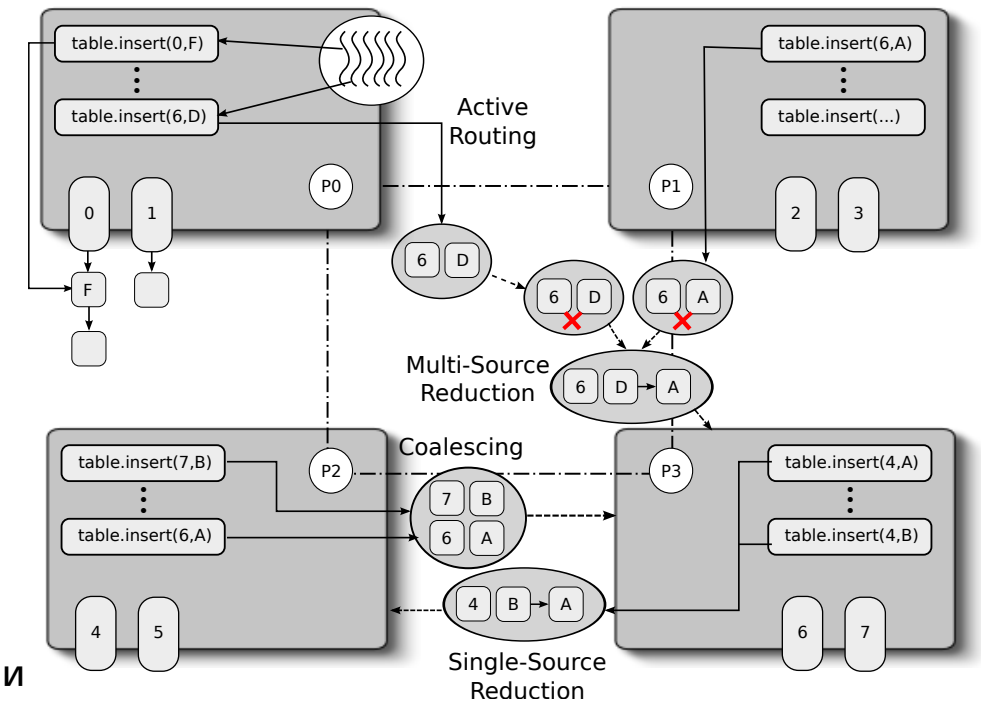
- Разработана в Indiana University
- API основан на использовании шаблонов, что дает гибкость в выборе алгоритмов и структур данных
  - ```
    typedef adjacency_list<listS,
      distributedS<mpi_process_group, vecS>,
      undirectedS,
      // Vertex properties
      no_property,
      // Edge properties
      property<edge_weight_t, double>
    > Graph;
    ```
- Основана на модели BSP (Bulk Synchronous Parallel)
  - `mpi_process_group`
- Содержит несколько десятков параллельных алгоритмов (breadth first search, sssp (Dejkstra, delta-stepping), betweenness centrality etc.)


    ```

Обзор подходов к реализации графовых задач на суперкомпьютерах

ActivePebbles (AM++), PBGL2

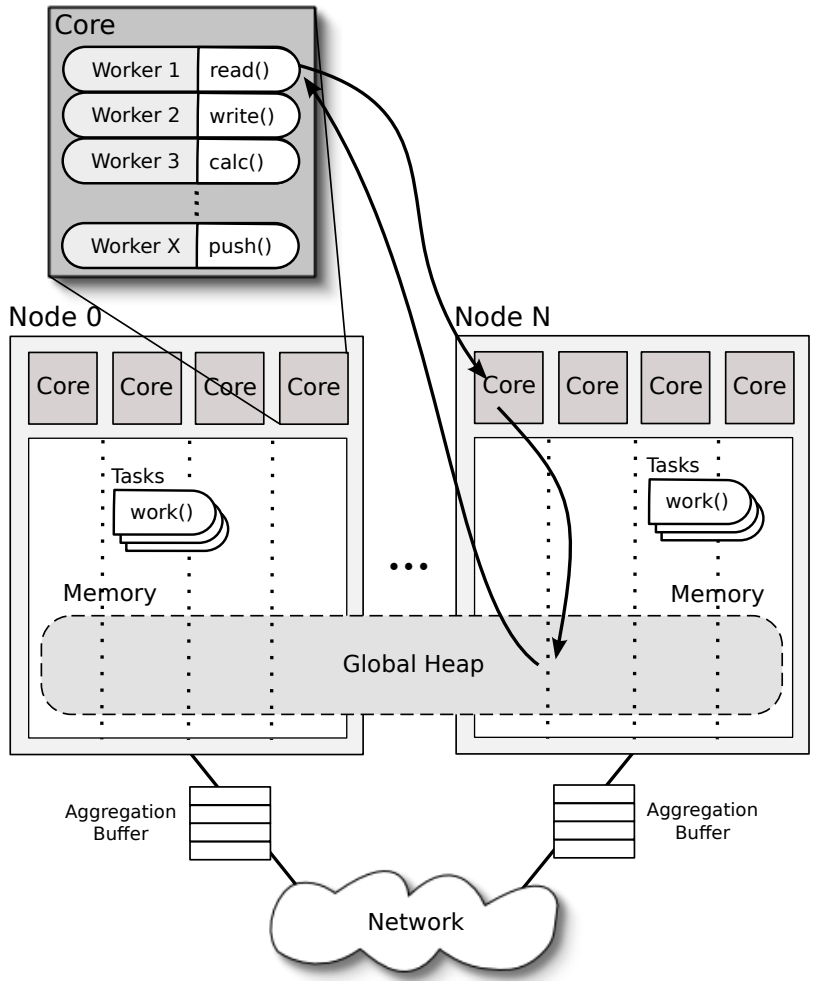
- Поддержка мелкозернистого параллелизма
- Асинхронная вычислительная модель на основе активных сообщений, два типа объектов
  - активные сообщения (*pebbles*)
  - узлы обработчики (*targets*)
- Глобальное адресное пространство, состоящее из *targets*-индентификаторов
- Статическое и динамическое распределение *targets*-объектов
- Автоматическая агрегация сообщений, программная маршрутизация сообщений
- Редукция сообщений в узлах-источниках и транзитных узлах
- Встроенные механизмы определения останова



Обзор подходов к реализации графовых задач на суперкомпьютерах

Grappa

- Разрабатывается в University of Washington
- Является библиотекой для C++11
- Runtime-система Grappa:
  - Tasking system: легковесные потоки (треды), используется work-stealing планирование
  - Communication layer: агрегация и передача сообщений
  - Distributed shared memory: поддержка глобального адресного пространства, операций чтения/записи, атомарных операций
- Реализация коммуникационного уровня:
  - MPI, GASNet

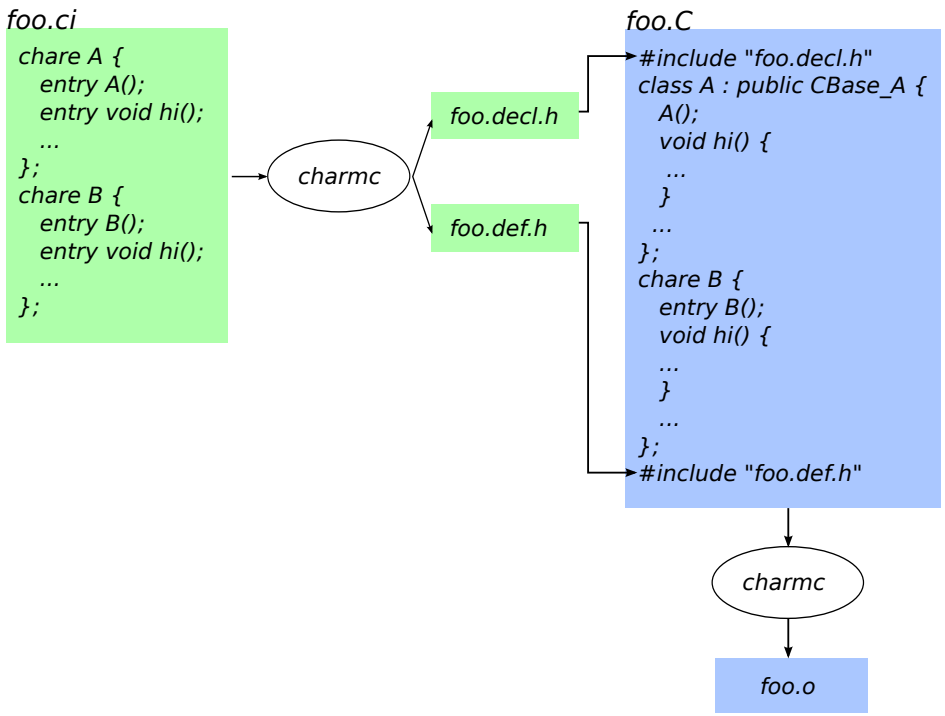


Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов
  - текущая версия 6.7.0
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)
- Поддерживаемые типы HPC-систем
  - SMP-узлы с NUMA памятью
  - кластеры с Infiniband, BlueGene/P, BlueGene/Q, Cray XK, Cray XC
  - ведутся работы по поддержке ускорителей (Xeon Phi, GPU)
- Приложения
  - NAMD, OpenAtom, ChANga, EpiSimdemics
  - BigSim, ClothSim, имитационная модель сети "Ангара"

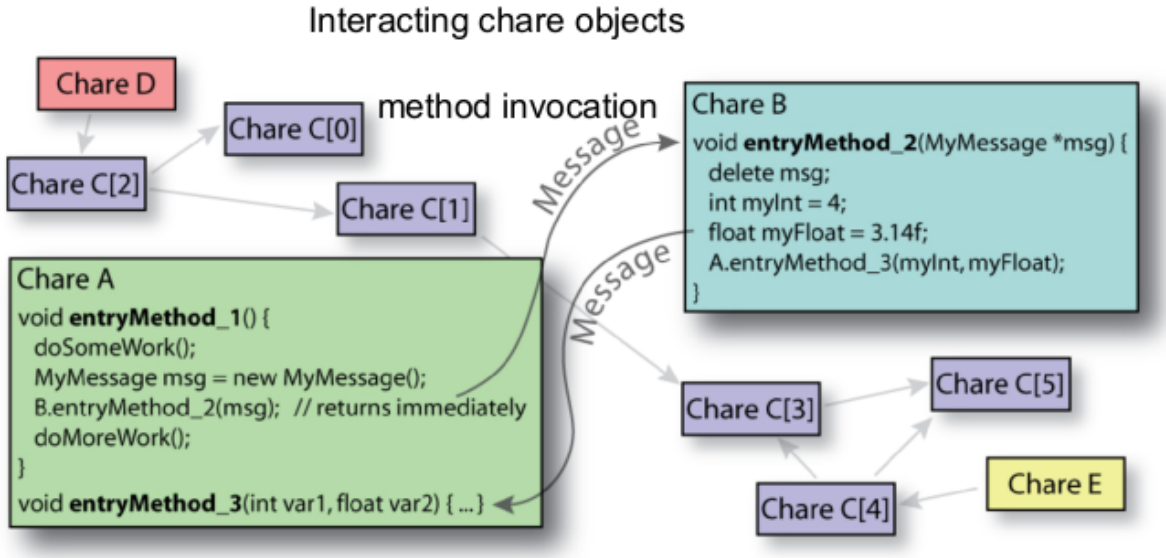
Программная модель Charm++ (1)

- Базовый объект Charm++ – *chare*
  - приложение состоит из множества *chare*-объектов
  - имеет набор *entry* методов, определенных .ci файле
  - *chare*-объекты обмениваются сообщениями, вызывая *entry*-методы друг друга
  - *entry*-методу доступны на запись только данные, принадлежащие соответствующему *chare*-объекту
  - выполнение *entry*-методов не может быть прервано, что гарантирует атомарность изменения данных внутри *chare*-объекта
  - могут быть объединены в "коллекции": 1D/2D-массивы

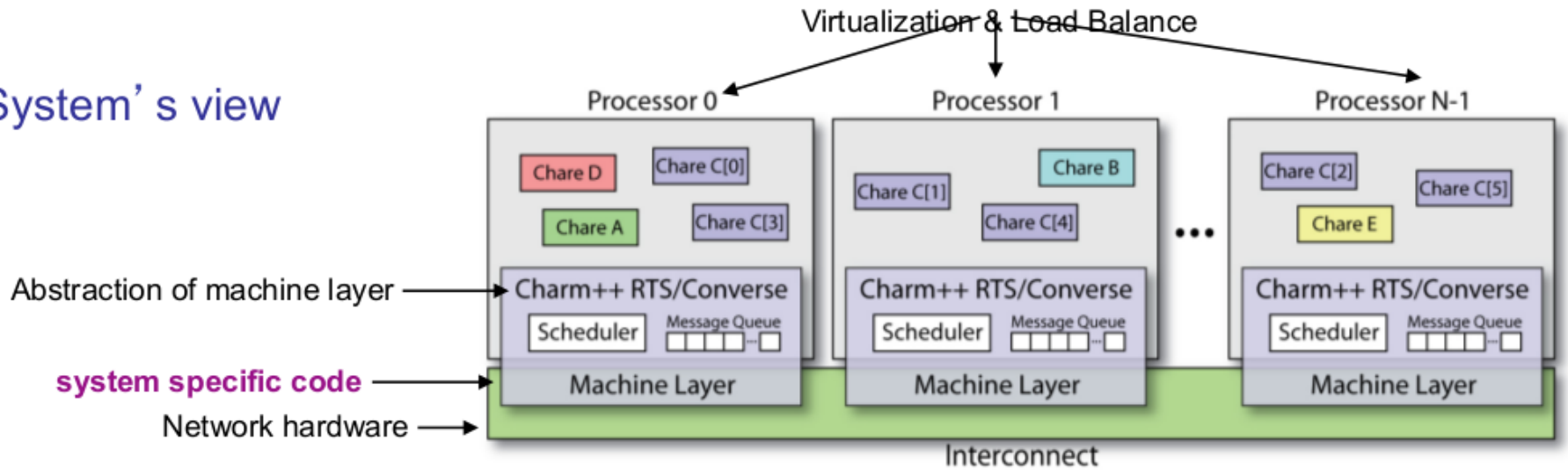


Программная модель Charm++ (2)

Application's view

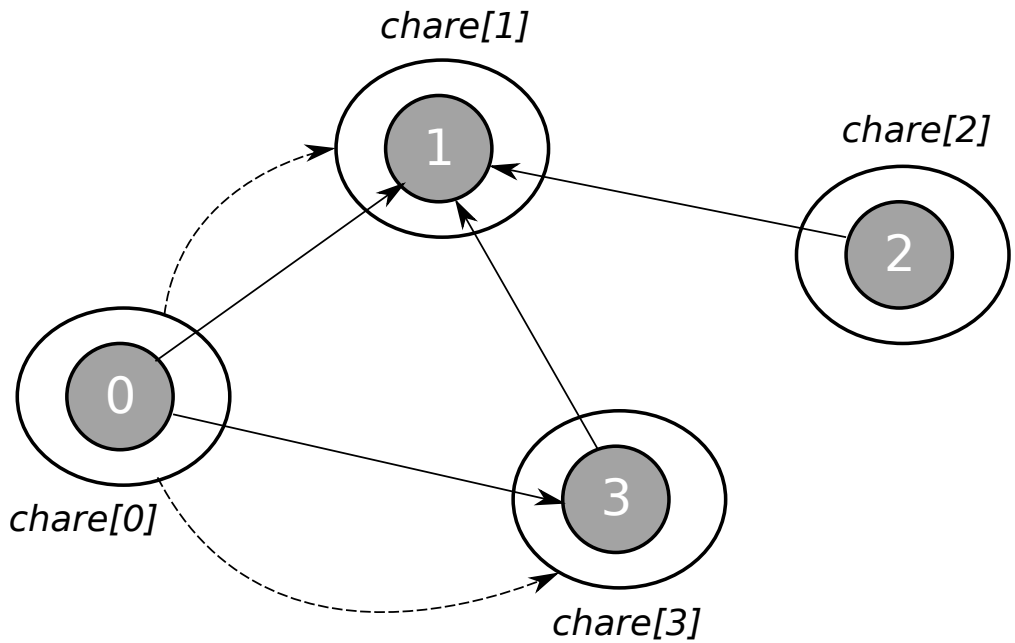


System's view



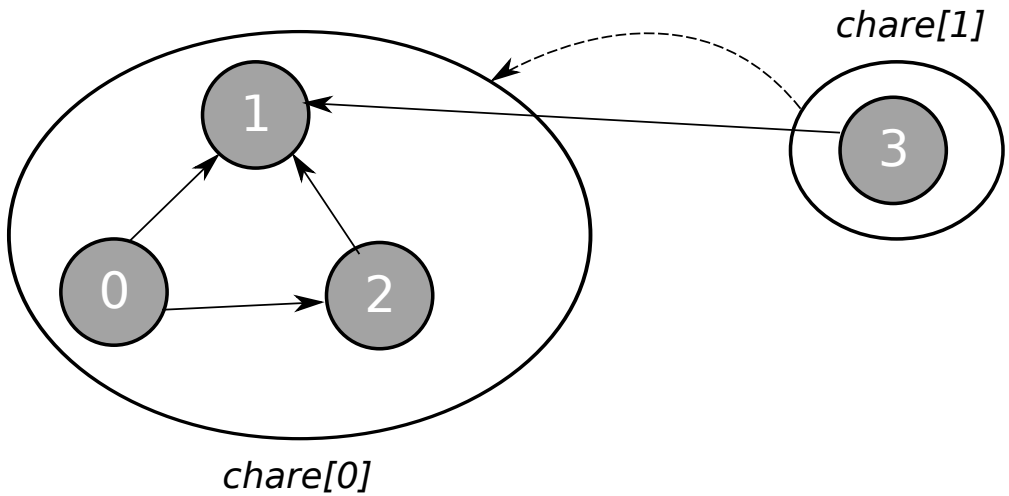
Подходы к реализации графовых алгоритмов на Charm++ (1)

- Подход “Vertex-centric”
  - Граф G – массив chare-объектов, распределенных по параллельным процессам (PE)
  - Каждая вершина – chare-объект
  - Вершины асинхронно обмениваются *активными* сообщениями (через вызовы entry-методов)
  - Окончание работы алгоритма соответсвует состоянию “тишины” в Charm++



Подходы к реализации графовых алгоритмов на Charm++ (2)

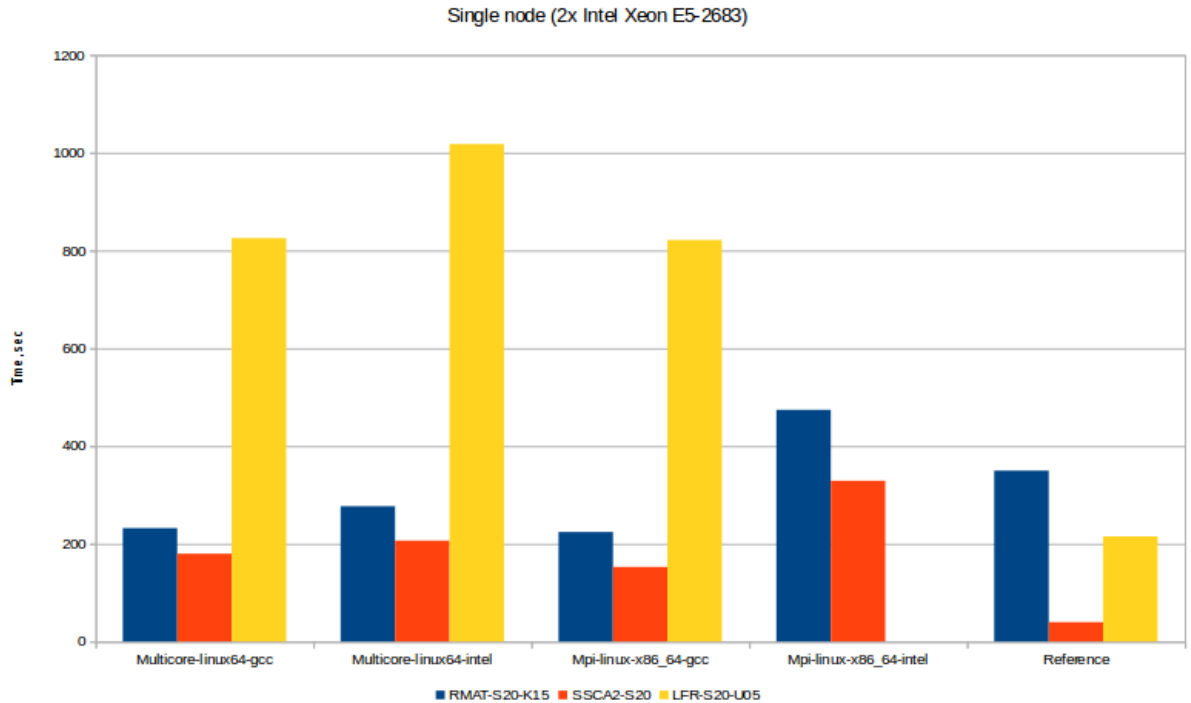
- Подход “Subgraph-centric”
  - Граф G – массив chare-объектов, распределенных по параллельным процессам (PE)
  - Каждый chare-объект содержит подграф графа G (представление подграфа в chare-объекте м.б. любым)
  - Алгоритмы состоят из последовательной [обработка локального подграфа] и параллельной частей [взаимодействие подграфов]
  - Подграфы асинхронно обмениваются *активными* сообщениями (через вызовы entry-методов)
  - Окончание работы алгоритма соответсвует состоянию “тишины” в Charm++

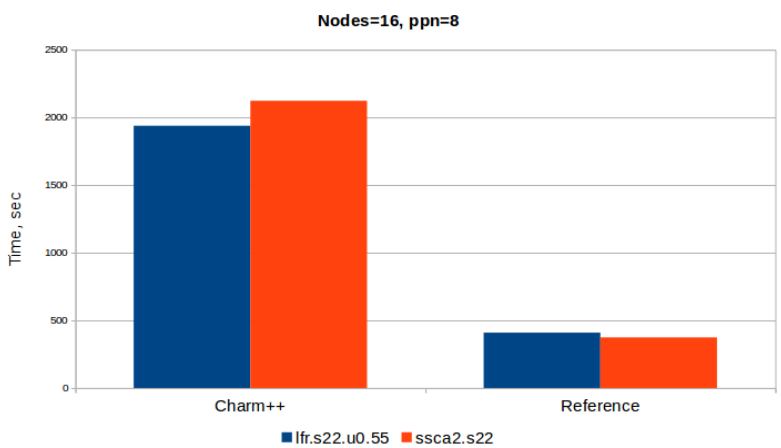


Проблемы Charm++ при реализации графовых алгоритмов в парадигме “Vertex-centric”

- Charm++ плохо поддерживает большое число chare-объектов на PE (1K и выше)
- В Charm++ нет встроенной агрегации сообщений (вызовов entry-методов)
- В Charm++ нет синхронизации подможеств chare-объектов:
  - механизм Quiescence Detection определяет “глобальную тишину” (т.е. для всех процессов/chare-объектов)
  - механизм Completion Detection требует знаний о количестве сообщений.
- Главная проблема – низкая производительность! Можно ли ее решить?

Производительность “наивной” реализации поиска сообществ в графе на Charm+ (GraphHPC-2016)

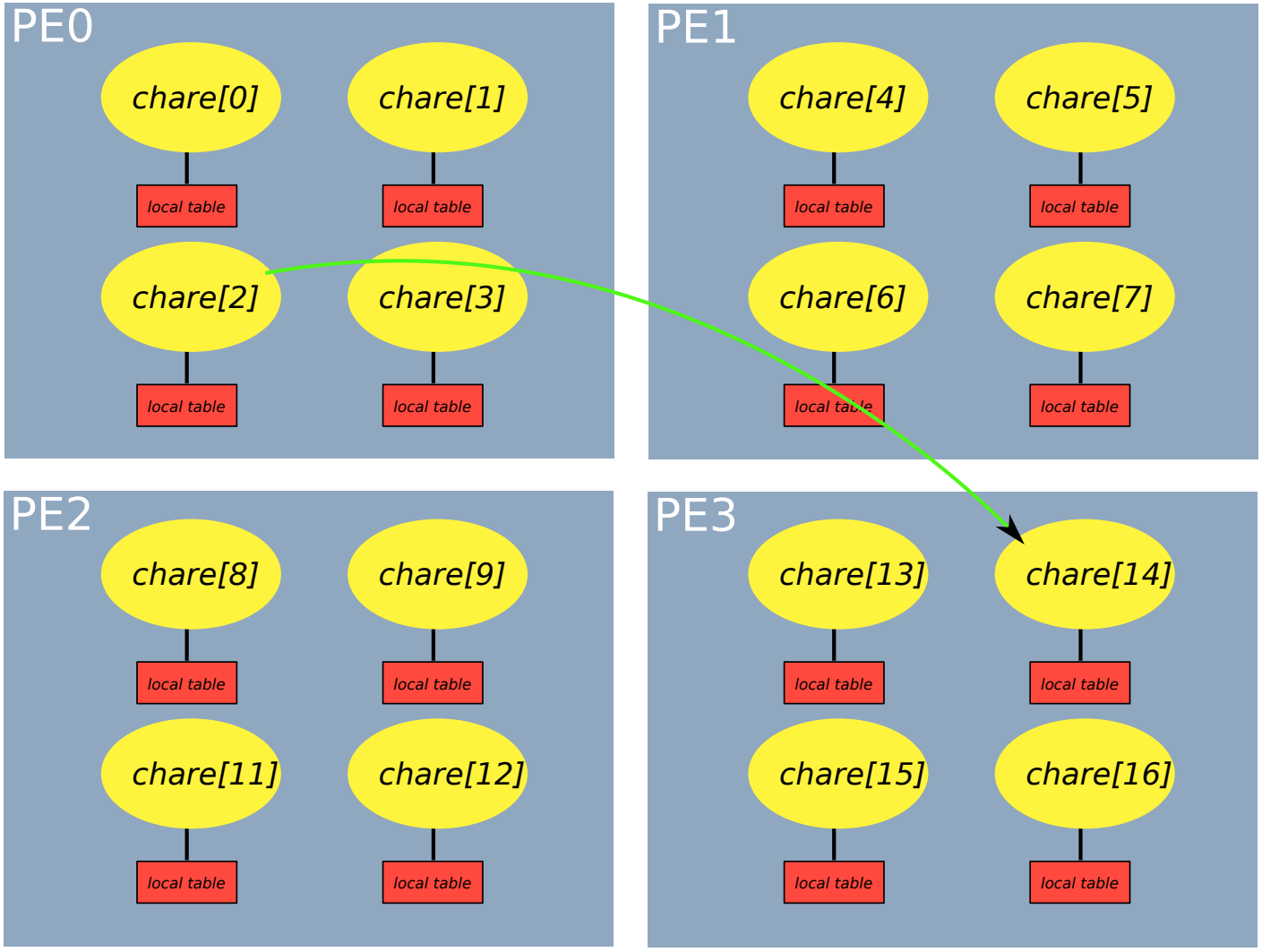




HPCC RandomAccess на Charm++

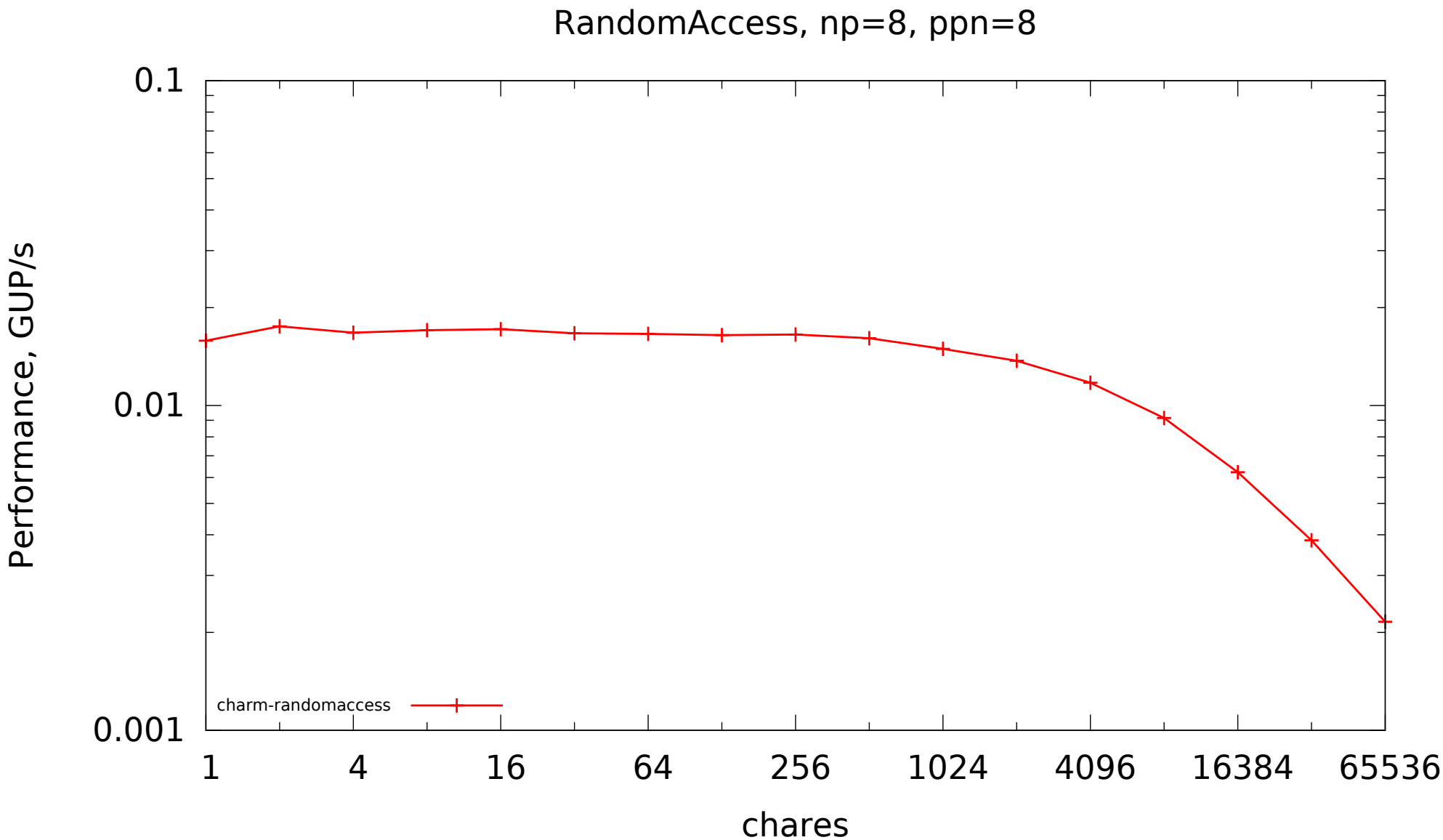
Algorithm RandomAccess

1: *Index* ← *Pseudo random indices*
2: **for all** $i \in Index$ **do**
3: $Table[i] \leftarrow Table[i] \oplus i$
4: **end for**



HPCC RandomAccess

Размер таблицы/PE: 2^{20} элементов, кластер на основе сети Ангара, MPI: MPICH 3.0.4



Но есть TRAM:

TRAM: Improving Fine-grained Communication Performance with Topological Routing and Aggregation of Messages

Presented by Lukasz Wesolowski

Оптимизация программной модели Charm++

Библиотека Topological Routing and Aggregation Module (TRAM)

- TRAM – библиотека для Charm++ приложений (NDMeshStreamer)
- Разработана для улучшения результата HPC Challenge (2011)
- Возможности TRAM:
  - агрегация коротких сообщений
  - создание виртуальных топологий (торовых) на которую отбражаются chare-объекты
  - узлы виртуальной топологии отображаются на узлы кластера, аллоцированные для приложения
  - агрегированные сообщения передаются peer-to-peer
  - для передачи данных с использованием TRAM используется специальный API (не стандартные вызовы entry-методов)

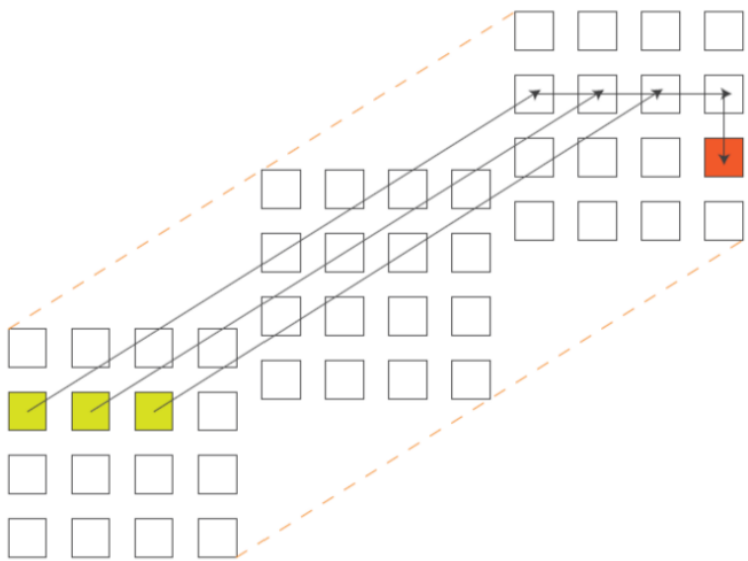
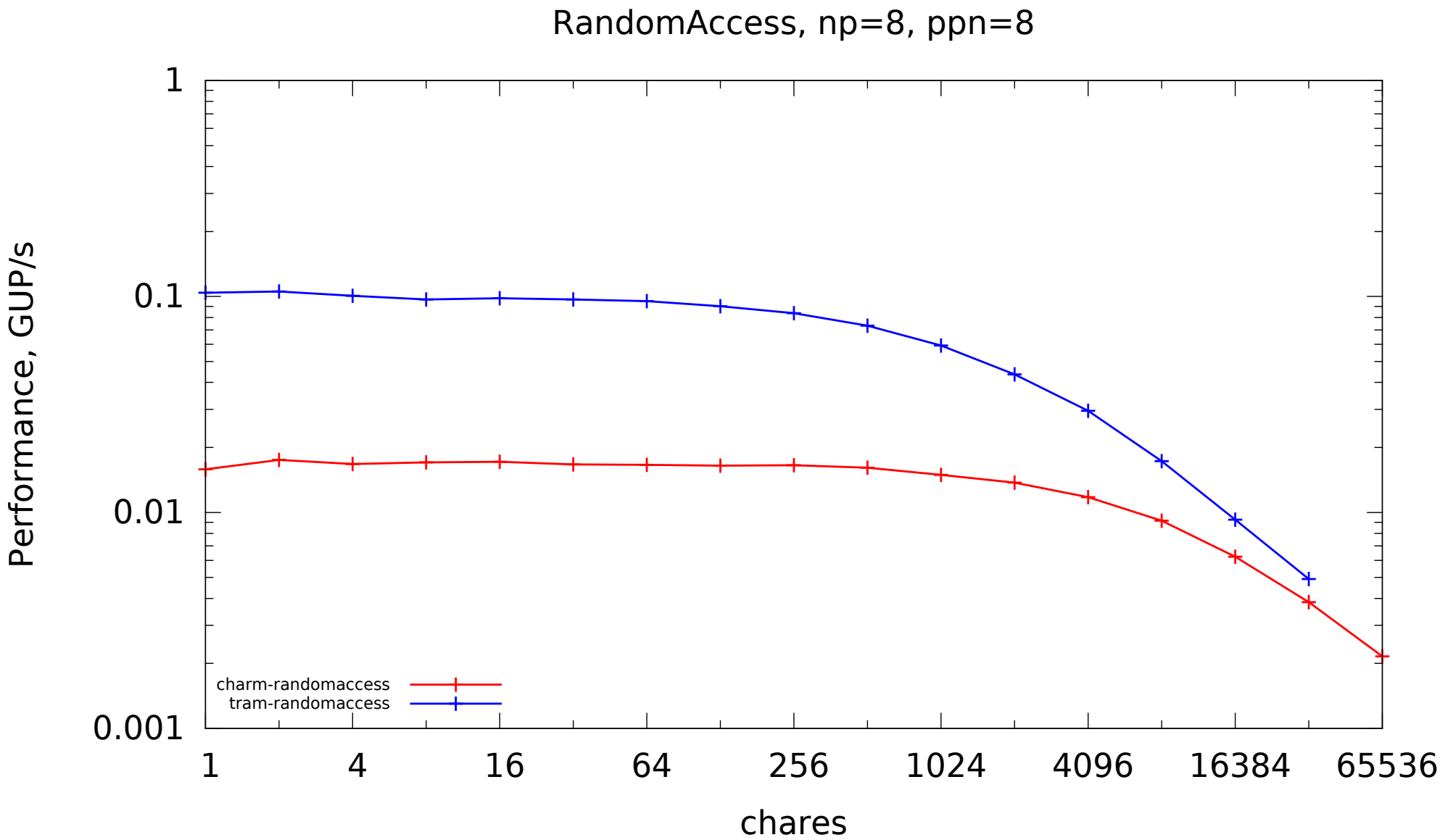


Рис. : Маршрутизация сообщений в TRAM (3D-тор)

HPCC RandomAccess

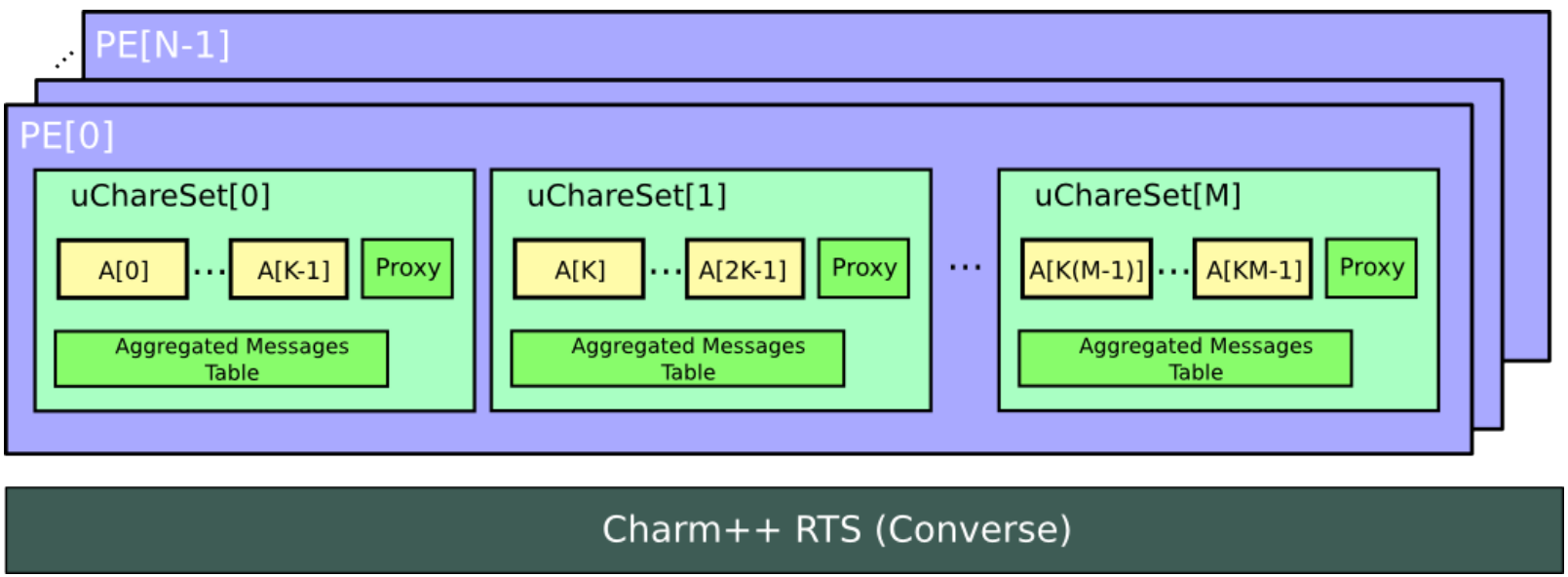
Размер таблицы/PE: 2^{20} элементов, кластер на основе сети Ангара, MPI: MPICH 3.0.4



Оптимизация программной модели Charm++

Библиотека uChareLib

- uChareLib (*микро*-Chare) – небольшое расширение языка Charm++, предоставляющее возможность уменьшить нагрузку на RTS Charm++ при высокой гранулярности параллелизма:
  - вводится дополнительный тип *uchare array* (определяется в .ci-файле)
  - *uchare*-объекты объединяются в *chare*-объекты
  - для передачи данных между *uchare*-объектами используются обычные вызовы entry-методов
  - сообщения между uchare-объектами агрегируются в *uChareLib*
  - введен новый тип entry-методов – reentrable (позволяет выполнять “повторный вход” в entry-метод одного объекта
  - во всем остальном программная модель uChareLib идентична Charm++



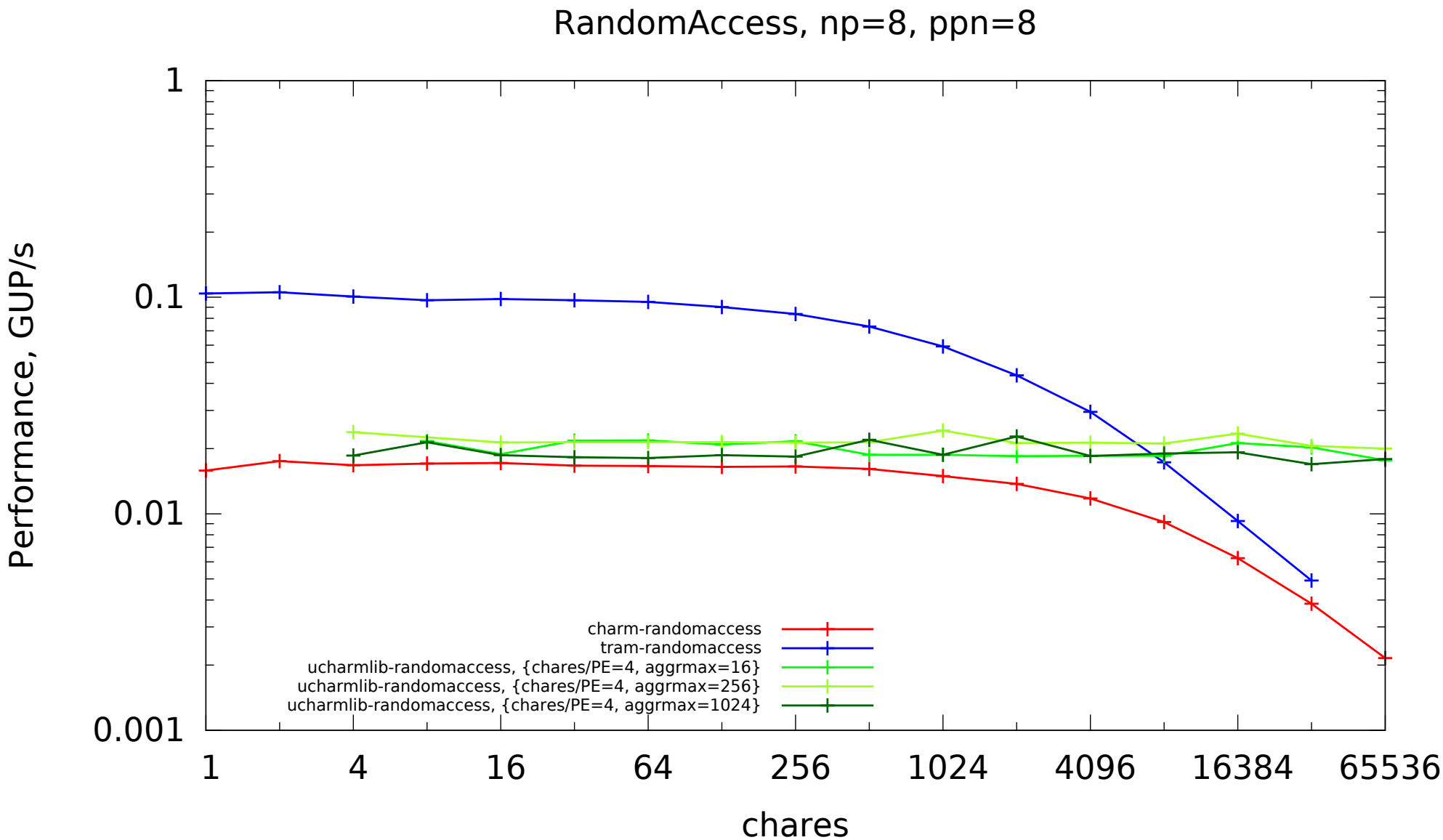
Библиотека uChareLib

Пример: Hello, World! (hello.ci)

```
1  mainmodule test_hello {
2    extern module uChareLib;
3    readonly CProxy_Main mainProxy;
4
5    mainchare Main {
6      entry Main(CkArgMsg *m);
7      entry [reductiontarget] void start();
8      entry void done();
9    };
10
11   uchare array [1D] Hello {
12     entry void hello(int callee);
13     entry [reentrable] void buybuy(int callee);
14   };
15   array [1D] uChareSet<Hello, CProxy_Hello, CBase_Hello>;
16 };
```

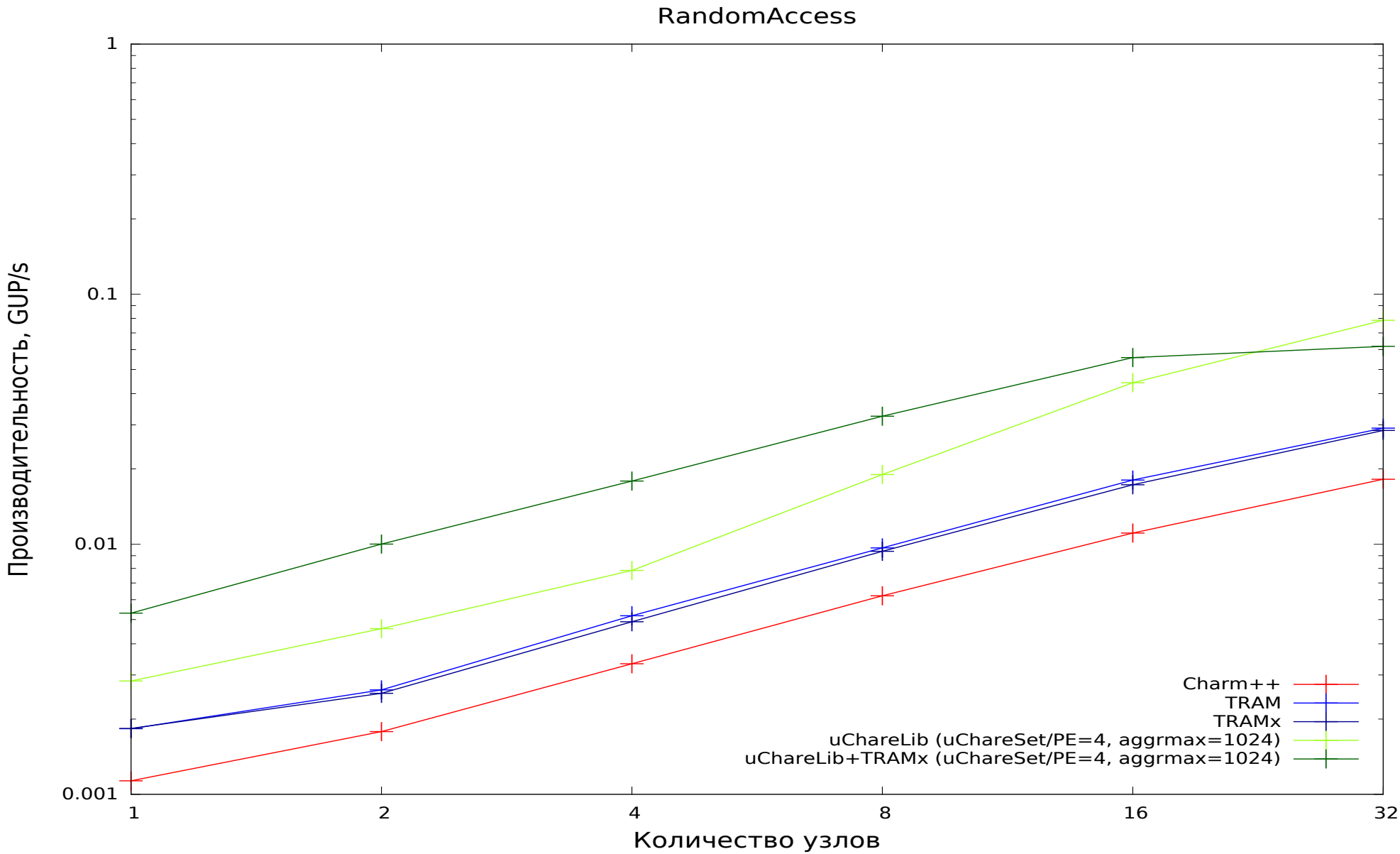
HPCC RandomAccess

Размер таблицы/PE: 2^{20} элементов, кластер на основе сети Ангара, MPI: MPICH 3.0.4



HPCC RandomAccess

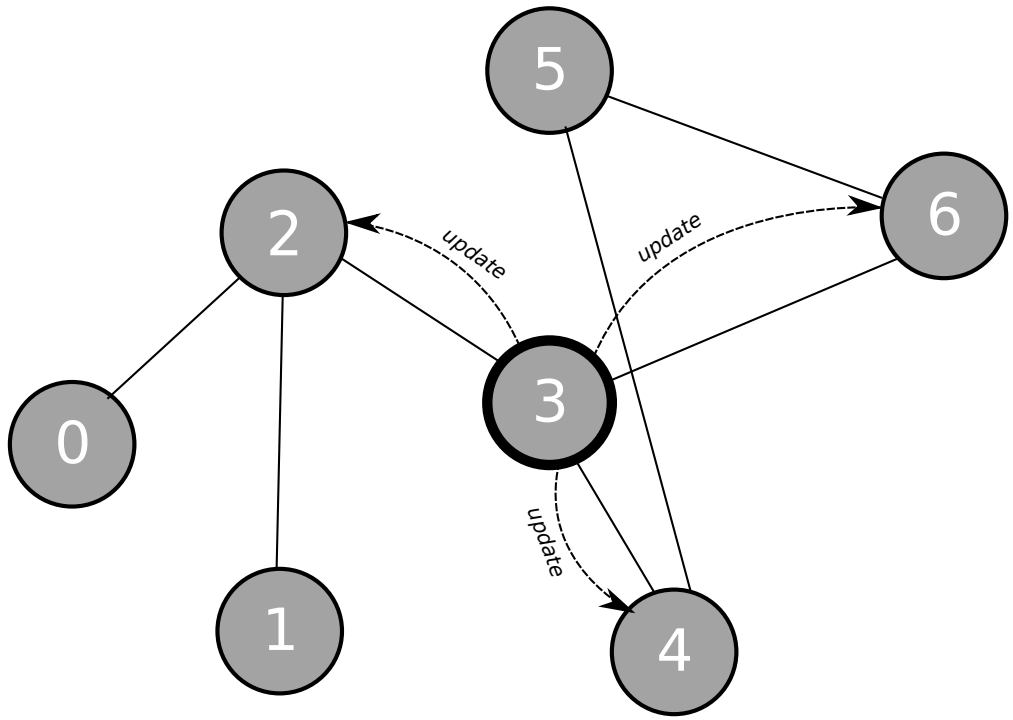
Размер таблицы/PE: 2^{20} элементов, кол-во chare(uchare)-объектов/PE: 16384, PE/узел (ppn) = 8, на основе сети Ангара, MPI: MPICH 3.0.4



Реализация асинхронного поиска вширь (asynchronous breadth-first search) на Charm++

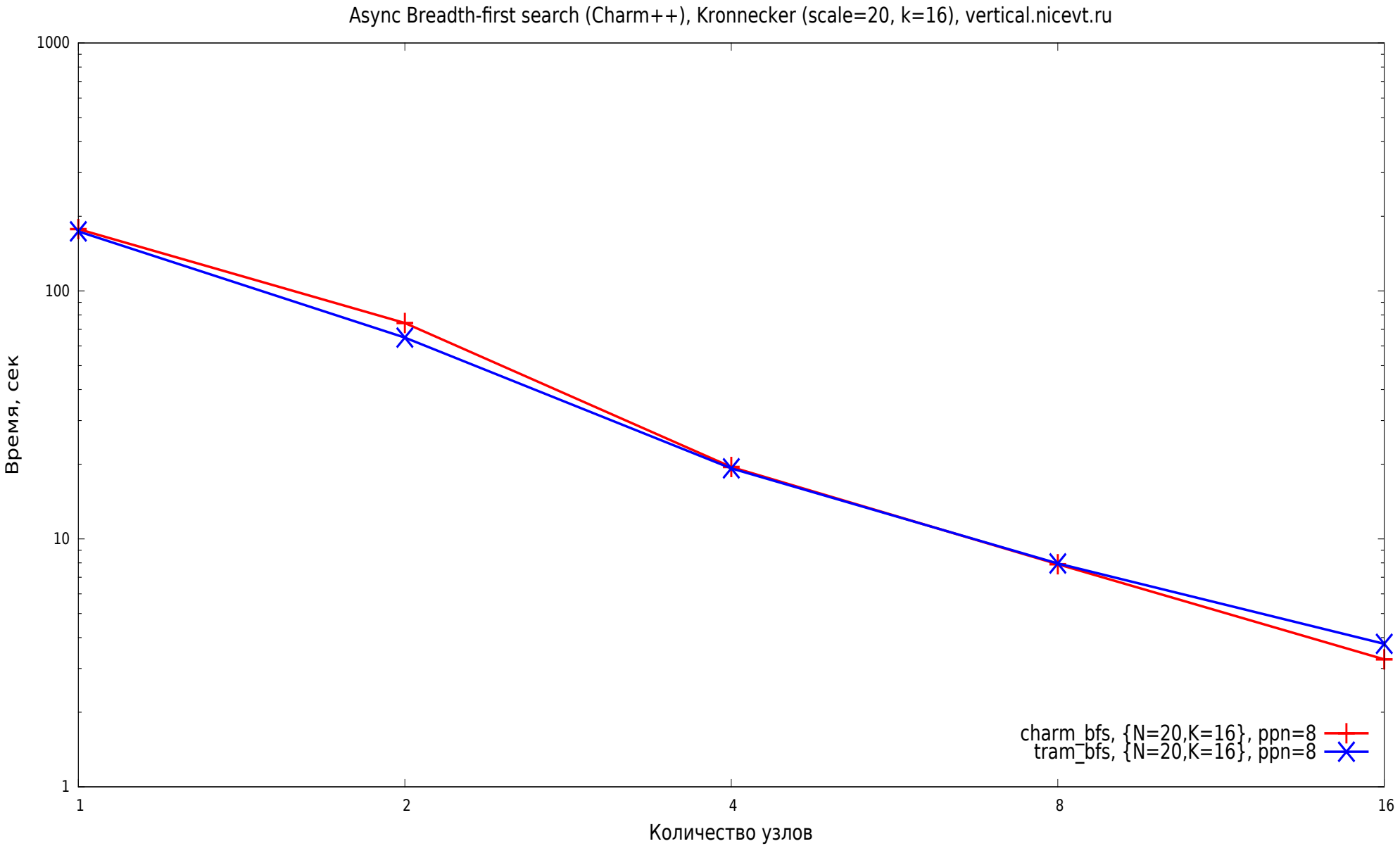
Algorithm Async BFS

```
1: function BFSVertex::Update
2:     if visited ≠ true then
3:         visited ← true
4:         for u ∈ AdjList do
5:             G[u].update()
6:         end for
7:     end if
8: end function
```



Async BFS

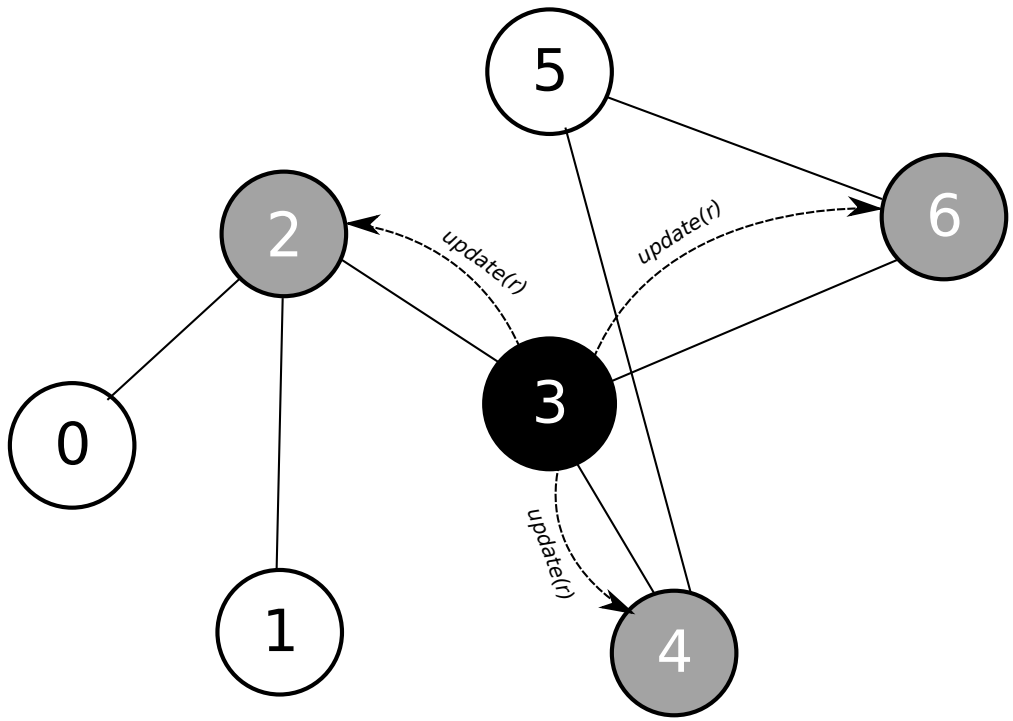
Граф: Kronecker (s=20,k=16), PPN=8,
система: кластер на основе сети Ангара, MPI: MPICH 3.0.4



Реализация асинхронного поиска вширь (asynchronous breadth-first search) на Charm++ (uChareLib)

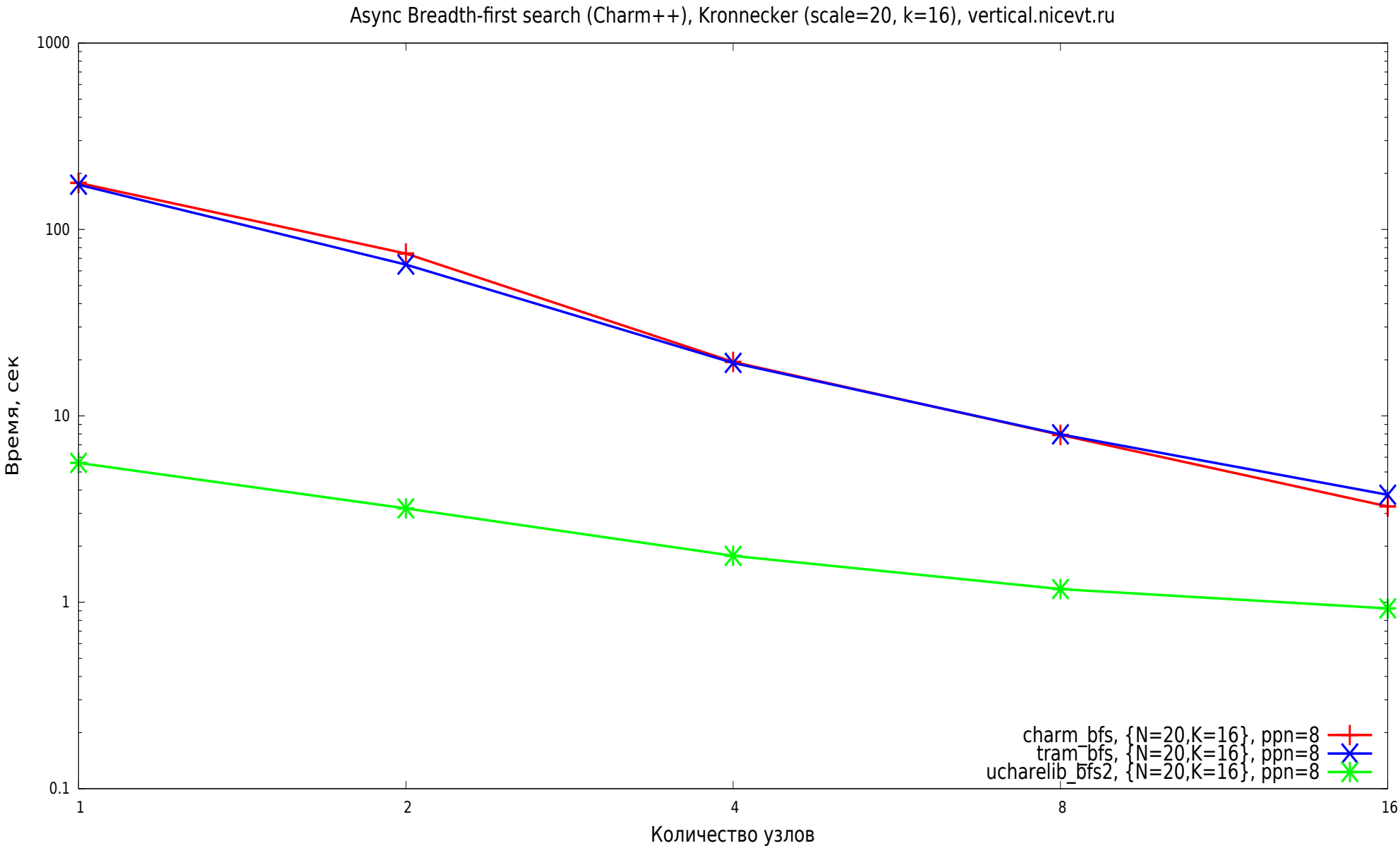
Algorithm Async BFS /w Radius

```
1:  function BFSVertex::Update(r)
2:      if state = White then
3:          if r > 0 then
4:              state ← Black
5:              for u ∈ AdjList do
6:                  G[u].update(r − 1)
7:              end for
8:          else
9:              state ← Gray
10:         end if
11:     end if
12: end function
13: function BFSVertex::Resume(r)
14:     if state = Gray then
15:         state ← Black
16:         for u ∈ AdjList do
17:             G[u].update(r − 1)
18:         end for
19:     end if
20: end function
```



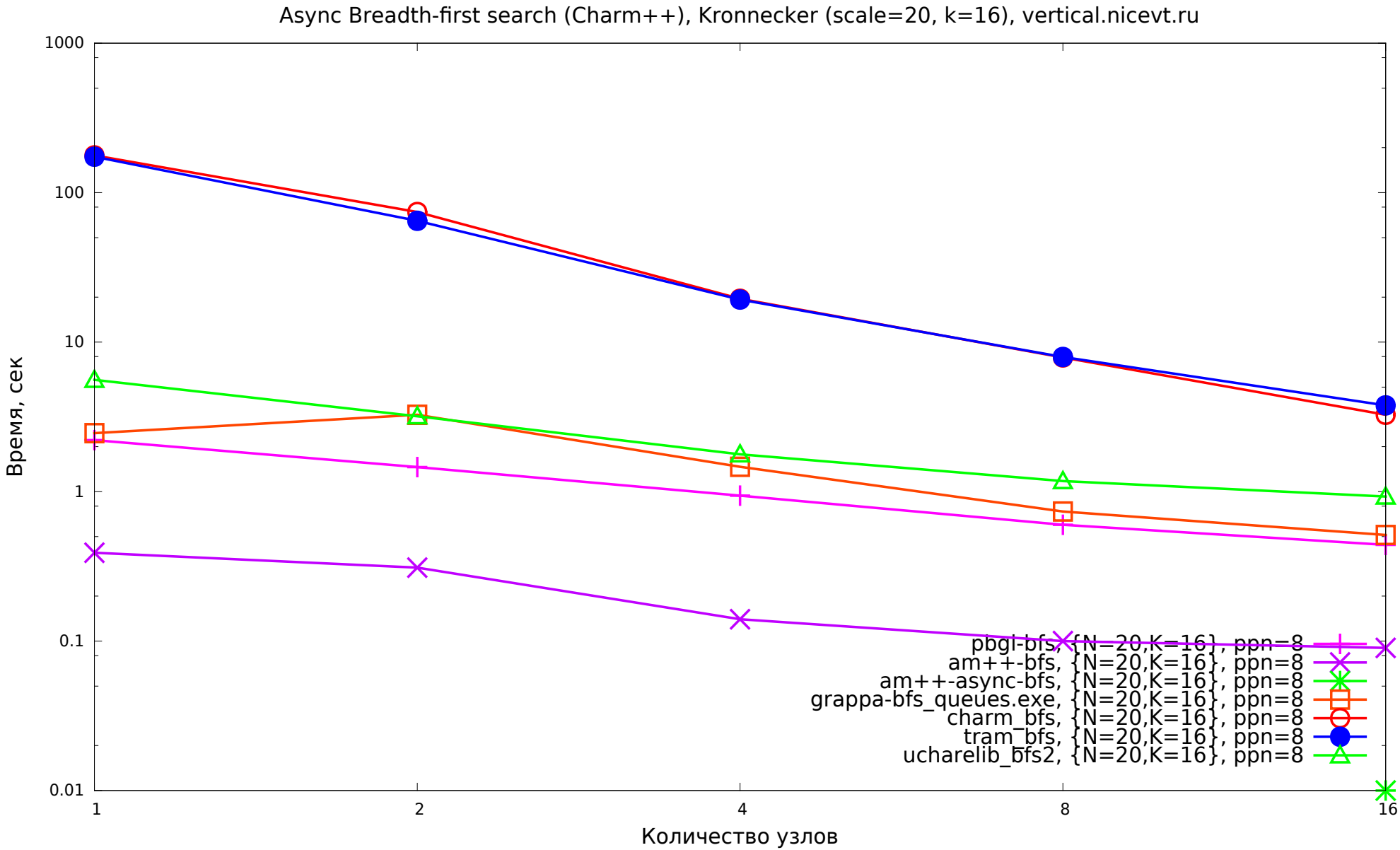
Async BFS

Граф: Kronecker (s=20,k=16), PPN=8,
система: кластер на основе сети Ангара, MPI: MPICH 3.0.4



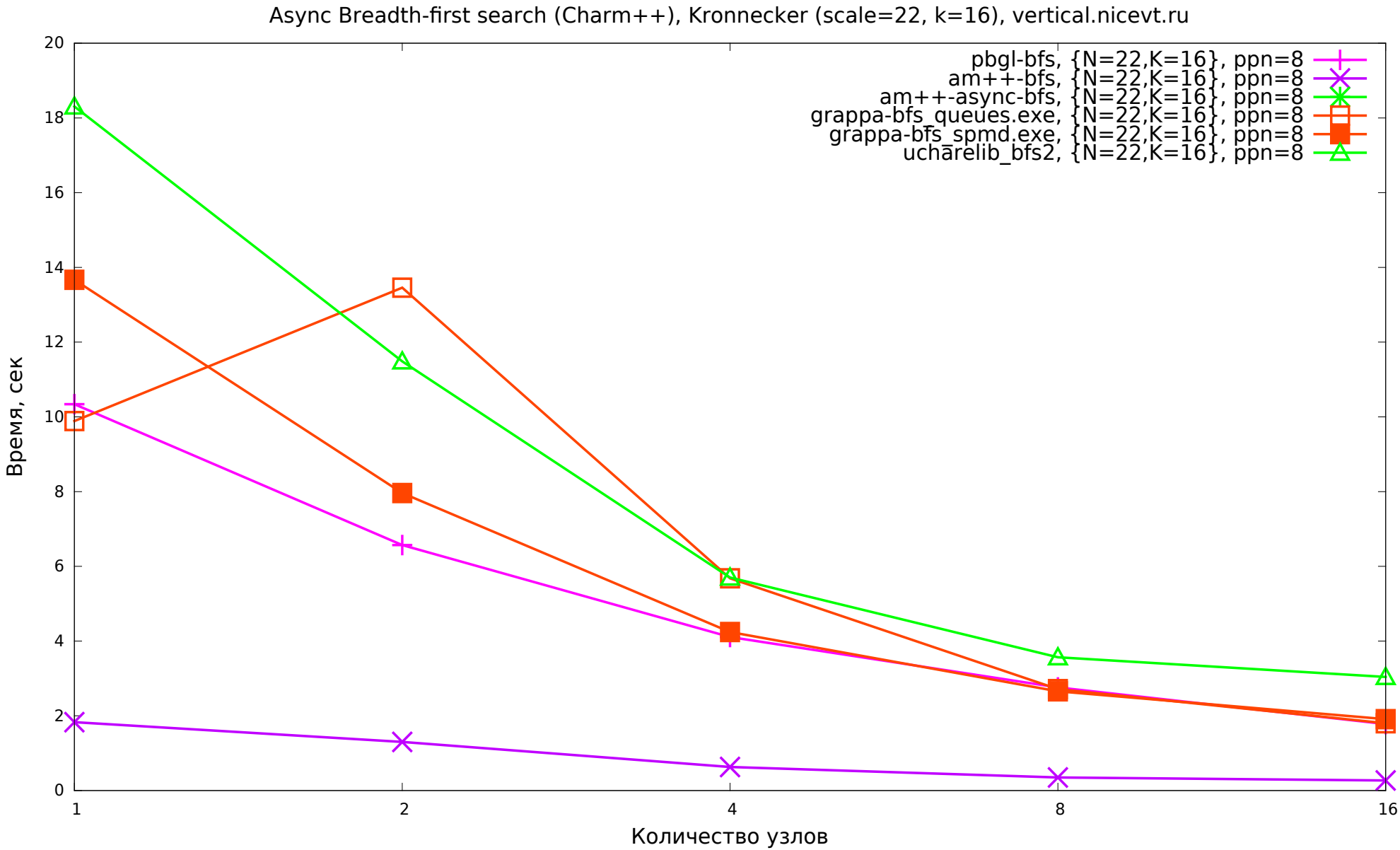
Сравнение различных реализаций BFS

Граф: Kronecker (s=20,k=16), PPN=8,
система: кластер на основе сети Ангара, MPI: MPICH 3.0.4



Сравнение различных реализаций BFS

Граф: Kronecker (s=22,k=16), PPN=8,
система: кластер на основе сети Ангара, MPI: MPICH 3.0.4

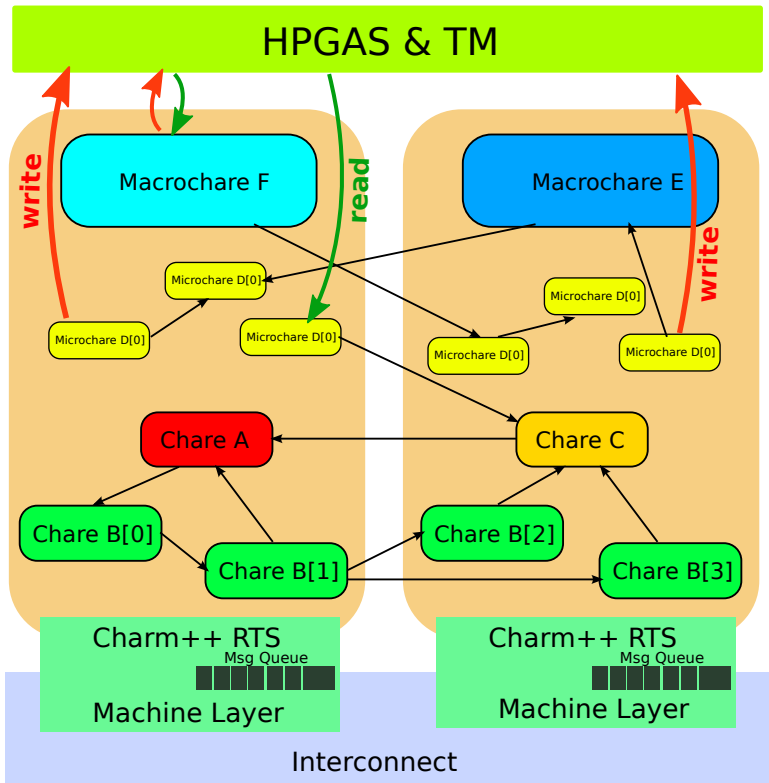


Выводы

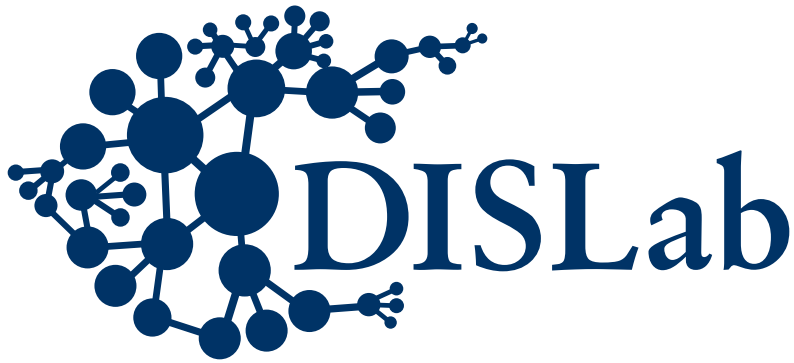
- Модель программирования Charm++ хорошо подходит для графовых алгоритмов, особенно описанных в vertex-centric парадигме.
- Отображение графовых алгоритмов 1:1 в Charm++ – путь к получению красивого решения и низкой производительности.
- Использование uChareLib должно обеспечить высокую производительность при сохранении стиля программирования Charm++.
  - На тестах RandomAccess и BFS удалось получить до 10X увеличения производительности при использовании uChareLib относительно Charm++
  - Исследование Charm++ и **uChareLib** на большем числе базовых графовых алгоритмов (PageRank, SSSP, Connected components и др.) в сравнении с AM++, Grappa, HPX-5 (ParalleX), а также более сложных тестах (MST, Community detection)
  - uChareLib может стать первым шагом к ...

XCharm: расширение модели Charm++ для систем с производительностью 10^{18} Flop/s

- Специализация chare-объектов по архитектуре процессорных элементов (PE)
  - Поддержка CUDA/OpenCL внутри методов chare-объектов, выделение классов chare-объектов для multi-core/many-core CPU, GPU и FPGA
- Специализация chare-объектов по гранулярности параллелизма
  - Выделение macrochare, chare, microchare в зависимости от количества вычислений, оптимицация планирования в RTS
- Динамические домены синхронизации
- Многоуровневая глобально адресуемая распределенная общая память (HPGAS)
- Поддержка проблемно-ориентированных языков программирования



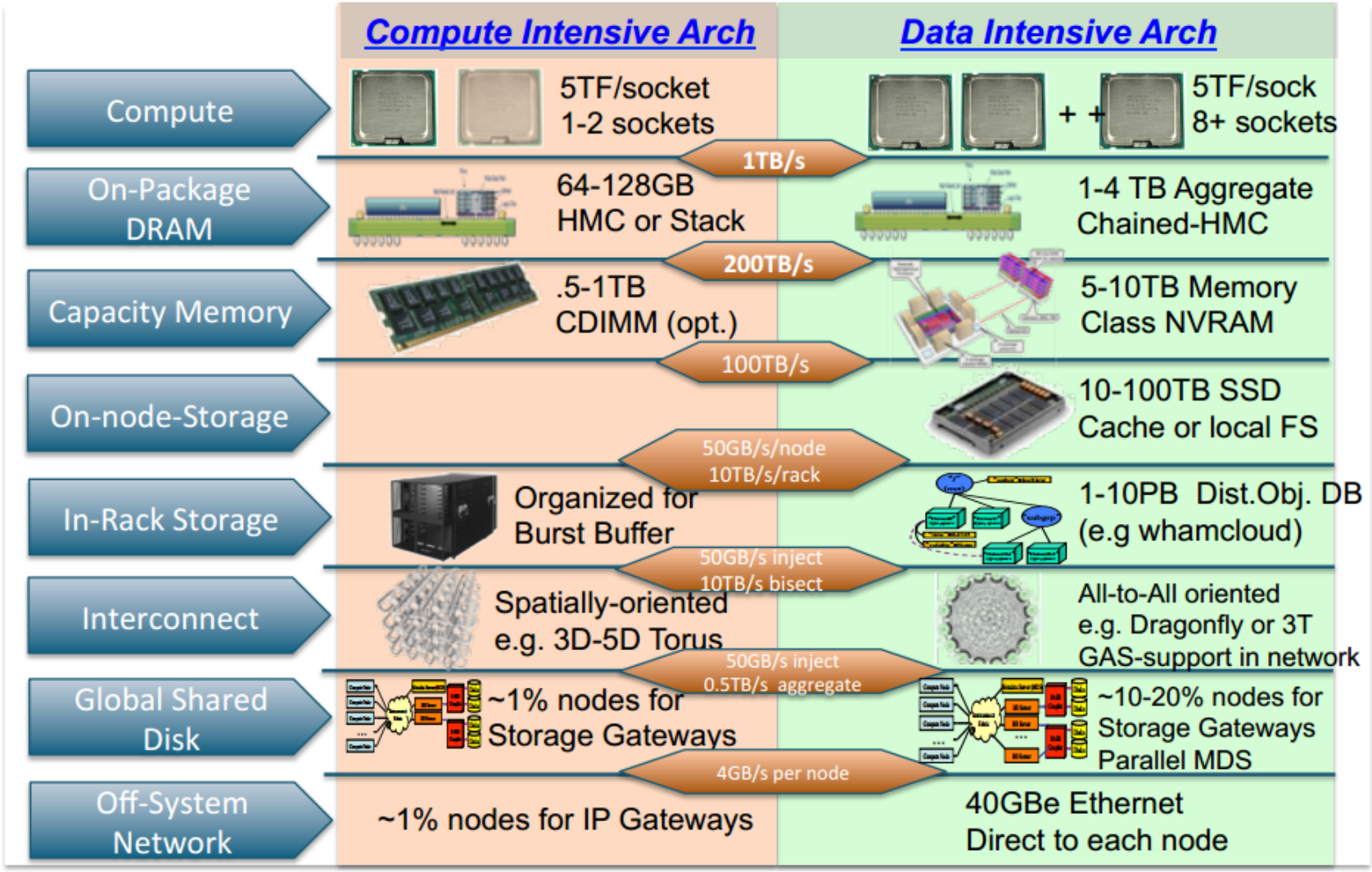
Спасибо! Вопросы?



http://dislab.org

Архитектурные тенденции экзафлосных систем

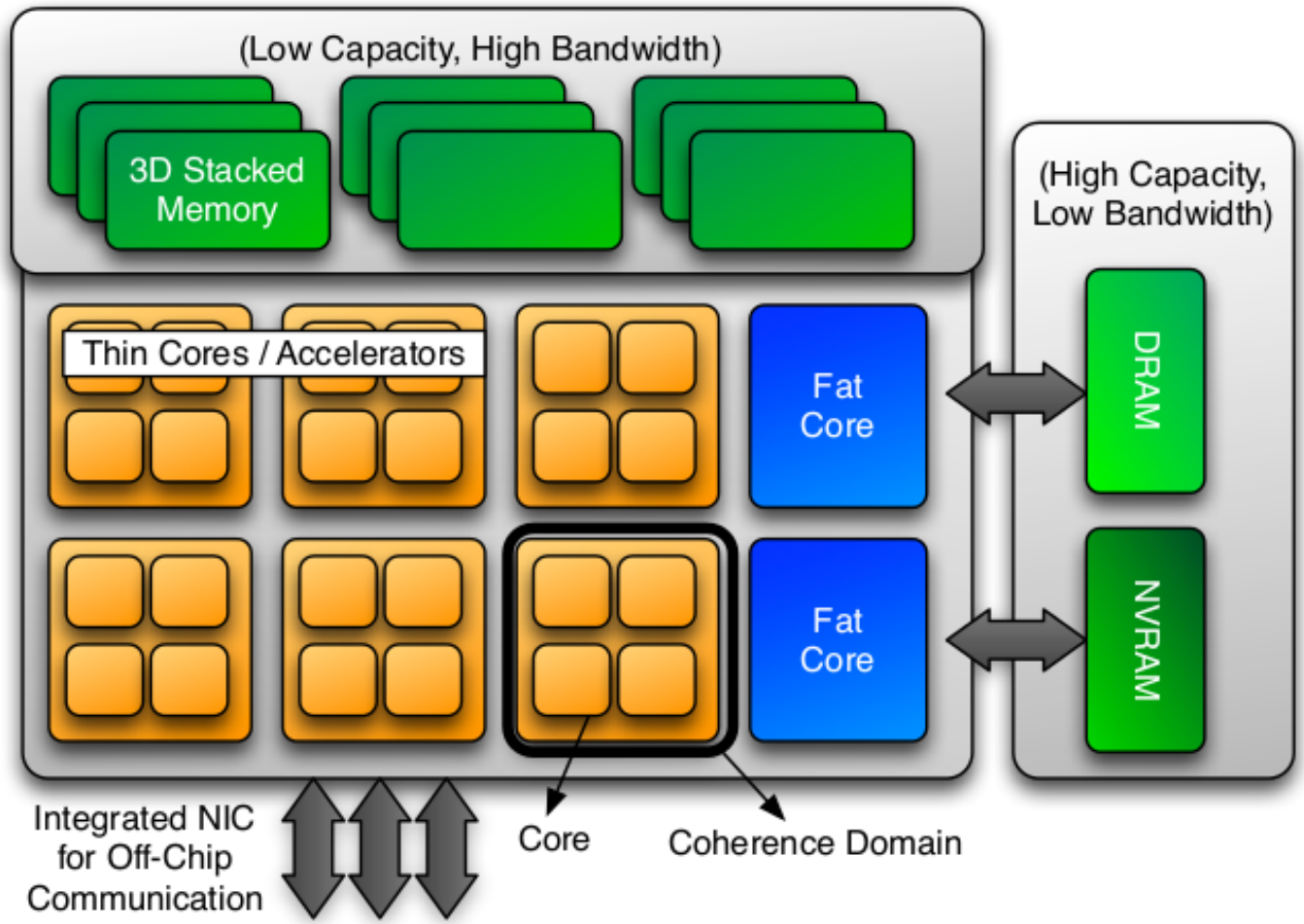
Strawman compute-intensive vs. data-intensive computer architectures in the 2017 [2]



[2] Synergetic Challenges in Data-Intensive Science and Exascale Computing, DOE ASCAC Report, 2013

Архитектурные тенденции экзафлосных систем

Архитектура (модель) узла экзафлопсного суперкомпьютера [3]



[3] J.A. Ang, R.F. Barrett, R.E. Benner, D. Burke, C. Chan, D. Donofrio, S.D... "Abstract Machine Models and Proxy Architectures for Exascale Computing Co–HPC2014 (to appear), New Orleans, LA, USA, IEEE Computer Society, November 17, 2014