Фролов А.С..[1], Семенов А.С.[2]

[1] АО «Научно-исследовательский центр электронной вычислительной техники», начальник отдела, frolov@nicevt.ru

[2] АО «Научно-исследовательский центр электронной вычислительной техники», начальник сектора, semenov@nicevt.ru

Использование Charm++ для решения графовых задач в масштабах экзафлопсных вычислений

КЛЮЧЕВЫЕ СЛОВА:

Обработка графов, языки параллельного программирования, программные модели, суперкомпьютеры, экзафлопс.

АННОТАЦИЯ:

В статье рассматривается проблема создания программной модели для экзафлопсных вычислительных систем, ориентированной на параллельную обработку больших графов. Предлагается в качестве основы программной модели использование модели параллельного языка Charm++ с специальными расширениями, ориентированными с одной стороны на поддержку архитектурных особенностей систем экзафлопсного класса, с другой стороны поддержка нерегулярных приложений с мелко-зернистым параллелизмом, характерным в том числе для задач обработки больших графов. Приводятся результаты экспериментов по введению многоуровневого параллелизма в программную модели Charm++ в виде библиотеки *uChareLib*. Результаты экспериментов получены на вычислительном кластере МВС10П на базе коммерческой коммуникационной сети Infiniband FDR и на 36-узловом кластере с отечественной коммуникационной сетью «Ангара».

ВВЕДЕНИЕ:

Одной из ключевых проблем создания суперкомпьютерных комплексов экзафлопсной производительности является вопрос разработки программных моделей и инструментальных средств программирования, позволяющих эффективно решать прикладные задачи в масштабах систем, состоящих из десятков тысяч вычислительных узлов, обладающих гибридной архитектурой, объединяющей разнородные вычислители (процессоры, ускорители на СБИС и ПЛИС) посредством единой высокоскоростной кэш-коггерентной сети, многоуровневой иерархической памятью, объединенных высокоскоростной коммуникационной сетью. [1,2]

Таким образом, достижение экзафлопсной производительности [с точки зрения архитектуры] сопряжено не только с экстенсивным наращиванием количества ядер и узлов вычислительной системы (относительно петафлопсного уровня количество ядер в системе будет x1000), но и с большим разнообразием (гибридностью) внутри вычислительного узла, более глубокой взаимной интеграцией вычислительных ядер друг с другим и с устройствами ввода-вывода (в том числе с сетевым адаптером), увеличением количества уровней иерархии памяти. Все эти изменения должны найти адекватную поддержку со стороны программных моделей и инструментальных средств программирования, используемых для экзафлопсных систем.

Развитие программных моделей (т. е. языков программирования) в значительной степени более инертно, чем аппаратных средств, что объясняется необходимостью поддерживать коды, написанные с использованием традиционных

средств параллельного программирования, таких как MPI, OpenMP, при чем развитие самих стандартов зачастую не отражает реальной ситуации с точки зрения их использования: коммуникационные операции MPI 1.0 по прежнему являются наиболее используемой практикой обменов данными между параллельными процессами. Такое явление характерно для прикладных областей, где имеется большой задел по количеству [legacy] прикладных задач и стоимость переписывания их с использованием новых программных моделей превышает возможный выигрыш для конечных пользователей.

Для относительно новых прикладных областей, таких как параллельная обработка больших графов, ситуация принципиально иная. Отсутствие проблемы поддержки legacy-кодов, позволяют использовать наиболее эффективные программные модели как с точки зрения производительности, так и с точки зрения продуктивности. Поэтому можно отметить, что именно в данных областях наиболее вероятно внедрение новых программных моделей

Кроме того, параллельная обработка графов характеризуется: во-первых, нерегулярной структурой обрабатываемых данных, как следствие интенсивной работой с памятью по непредсказуемым адресам и преимущественно каждое обращение считывает или записывает одно 64-разрядное слово, что не позволяет эффективно использовать пропускную способность памяти и кэши-данных; во-вторых, интенсивными обменами короткими сообщениями по коммуникационной сети с характерным шаблоном "все-всем"; в-третьих, преобладанием целочисленных адресных вычислений и операций обращения к памяти над арифметическими операциями с плавающей точкой; в-четвертых, неравномерной нагрузкой на вычислительные узлы.

Таким образом, для задач обработки больших графов на суперкомпьютерных комплексах экзафлопсной производительности необходимо создание новых альтернативных моделей вычислений отражающих как специфику графовых задач, так и архитектурные особенности экзафлопсных систем.

Данная работа посвящена созданию новой программной модели на основе языка программирования Charm++ [3] с управлением потоком сообщений, ориентированной на параллельную обработку графов в масштабах экзафлопсных систем. Далее приводится краткое описание Charm++ и предлагаемых расширений к модели программирования Charm++; приводятся результаты экспериментов по введению многоуровневого параллелизма в программную модели Charm++ в виде библиотеки *uChareLib*. В заключении приводится обобщение представленного материала и экспериментально полученных результатов, формулируются задачи дальнейшего исследования.

СТАНДАРТНАЯ И РАСШИРЕННЫЕ ПРОГРАММНЫЕ МОДЕЛИ CHARM++:

Язык параллельного программирования Charm++ является расширением языка C++ и основан на объектно-ориентированной модели с управлением асинхронным потоком сообщений [3].

Базовым объектом в Charm++ является *chare* (или *chare-объект*), обладающий помимо всех свойств объектов в C++ (инкапсулированными данными, множеством публичных и приватных методов и т. п.), интерфейсом из специальных *entry-методов*, вызовы которых соответствуют передаче данных (т. е. сообщений) между chare-объектами.

Приложение в Charm++ состоит из множества chare-объектов, обменивающихся между собой данными посредством вызовов entry-методов. Кроме того, вызов entry-метода порождает выполнение непосредственно самого метода на том узле, где находится chare-объект, entry-метод которого был вызван, то есть таким образом реализуется концепция *управления потоком данных* или *активных сообщений*.

В процессе выполнения entry-метода могут быть изменены только данные, принадлежащие chare-объекту. Одновременно у chare-объекта может выполняться не более одного entry-метода, что позволяет избежать проблемы обеспечения атомарности изменения данных, принадлежащих chare-объекту, и значительно упрощает программирование. Схематически программная модель Charm++ представлена на рисунке 1.

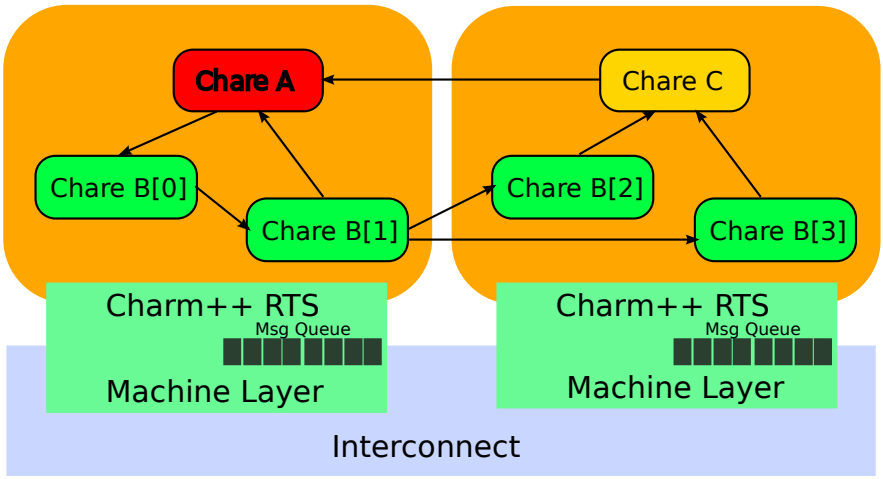

Рис.1. Классическая программная модель Charm++

Кроме простых chare-объектов Charm++ имеется возможность создавать массивы chare-объектов. При этом массивы бывают как одномерными, так и многомерными. Использование массивов позволяет создавать большое количество chare-объектов, управлять их распределением по вычислительным узлам, выполнять над ними коллективные операции.

Кроме того, в Charm++ поддерживается автоматическая динамическая балансировка нагрузки на вычислительные узлы, что реализуется на уровне runtime-системы за счет перемещения chare-объектов между вычислительными узлами. Для прикладного программиста данная возможность совершенна прозрачна, так как программная модель Charm++ не специфицирует расположение chare-объектов в вычислительной системе, и все объекты адресуются в едином адресном пространстве.

Основными причинами выбора Charm++ как основы для создания новой программной модели, ориентированной на обработку больших графов в масштабах экзафлопсных вычислительных систем, являлись: во-первых, асинхронная вычислительная модель Charm++ с управлением потоком данных; во-вторых, открытый исходный код; в-третьих, развитая инфраструктура Charm++ (наличие средств отладки и профилирования параллельных программ); в-третьих, активное сообщество разработчиков и пользователей.

Основные концепции модели Charm++ были разработаны почти 20 лет назад в то время, когда массово-параллельные системы (вычислительные кластеры) на основе коммерческих микропроцессоров только набирали популярность. Тем не менее, модель Charm++, вобравшая в себя как принципы dataflow-моделей, так и

многопоточных (мультитредовых) моделей вычислений, оказалась достаточно успешной и с момента создания почти не претерпевала изменений. Однако, за этот период архитектура вычислительных систем существенна изменилась, а с появлением экзафлосных суперкомпьютеров разница архитектур станет еще значительней

Таким образом, сохранение основных принципов программной модели Charm++ (управление потоком сообщений, асинхронность, инкапсулированность данных), введение расширений, отражающих новые архитектурные возможности суперкомпьютеров, а также соответствие классу задач обработки больших графов – суть методологии создания новой программой модели, описываемой в данной статье. Схематически предлагаемая расширенная программная модель Charm++ приведена на рисунке 2.

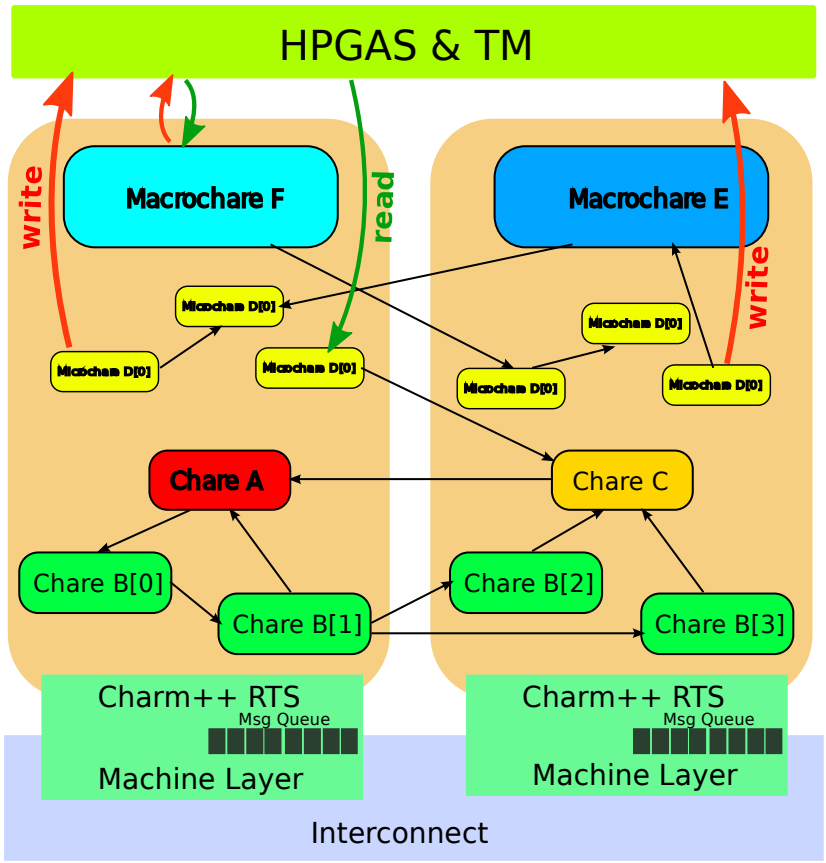

Рис.2. Расширенная программная модель Charm++

Ниже приведены описания предлагаемых расширений к Charm++.

Классификация chare-объектов по целевой архитектуре. Для соответствия гибридной архитектуре вычислительных узлов экзафлосных вычислительных систем, включающей в себя вычислительная ядра разного типа (CPU, GPU, FPGA), предлагается расширить тип chare-объекта Charm++ соответствующим спецификатором. Для реализации предлагается ввести сооветствующие ключевые слова в язык интерфейса Charm++, используемый в .ci файлах. Система поддержки выполнения Charm++ программ (runtime-система) будет автоматически распределять chare-объекты по соответсвующим вычислительным ядрам узла суперкомпьютера. Некоторые аспекты совместного использования Charm++ и CUDA для программирования оснащенных GPU кластеров представлены в [4].

Классификация chare-объектов по граннулярности параллелизма. Также предлагается разделять chare-объекты по степени гранулярности параллелизма (т. е. объему данных, соддержащихся внутри объекта, и количеству команд процессора,

требуемых для выполнения entry-методов) Предлагается ввести три типа объектов: *микро*, *нормальный* и *макро*. В соответствии с названиями микро-chare-объекты будут соответствовать самым малым вычислительным единицам параллельного алгоритма (таких объектов на одном ядре может выполняться до десятков тысяч), нормальные chare-объекты соответствуют обычным Charm++ объектам, макро-chare-объекты будут соответствовать наиболее "тяжелым" вычислениям, требующим большого количества данных, и включать entry-методы, состоящие из большого числа инструкций (например, содержать вложенные циклы и т.п.). Такая специализация позволит runtime-системе оптимизировать хранение chare-объектов и обеспечить наиболее эффективное планирование выполнения entry-методов. Для реализации предлагается ввести сооветствующие ключевые слова в язык интерфейса Charm++, используемый в .ci файлах.

Динамические домены синхронизации. Использование динамических доменов синхронизации позволит повысить эффективность и масштабируемость приложений, в которых предполагается выполнение какой-либо синхронизации (барьера, определения состояния завершения всех entry-методов и т.п.) по динамически определяемому подмножеству chare-объектов. Данная операция требуется во многих графовых алгоритмах и их асинхронных реализациях (например, в алгоритме Борувки поиска минимального остовного дерева графа).

Многоуровневая глобально адресуемая распределенная общая память (HPGAS). Изначально программная модель Charm++ не предполагает использования общей памяти как таковой. Все chare-объекты работают только с локальными данными, что позволяет перемещать chare-объекты между вычислительными узлами. Однако для некоторых алгоритмов введение общей памяти позволяет упростить реализацию и повысить эффективность. Предлагаемые уровни иерархии должны включать: уровень кэш-памяти (L1-L3 кэши), уровень сверхбыстрой памяти (HBM-памяти), уровень локальной памяти (DDR4), уровень локальной NVRAM, уровень удаленной памяти, доступной через, RDMA операции. Данные должны размещаться в подсистеме памяти в соответствии с указанным уровнем, что позволит обеспечить максимальную гибкость в управлении доступом к памяти. Некоторые аспекты совместимости программных моделей Charm++ и PGAS рассмотрены [5,6].

Поддержка проблемно-ориентированных языков программирования. Расширение возможностей программной модели Charm++ может привести к усложнению программирования. Данная проблема может быть решена за счет использования проблемно-ориентированных языков программирования (DSL). Для работы с графами может быть использован язык Green-Marl [7] или другие языки для работы с графами (в том числе языки запросов к графовым базам данных).

Представленный перечень расширения программной модели Charm++ не является окончательным, и его уточнения будет проводиться в ходе дальнейших исследований.

ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ:

Для демонстрации потенциала расширенной программной модели Charm++ рассмотрим введение микро-chare-объектов (или uchare-объектов) дополнительно к обычным chare-объектам, определенным в Charm++ .

Для реализации uchare-объектов была разработана библиотека *uChareLib* и в язык интерфейса Charm++ (язык описания Charm++ конструкций в .ci файлах) было

введено соответствующее ключевое слово – uchare. Пример описания массива из uchare-объектов представлен на рисунке 3.

```
mainmodule test_hello {
 extern module uChareLib;
 readonly CProxy_Main mainProxy;

 mainchare Main {
  entry Main(CkArgMsg *m);
  entry [ reductiontarget ] void start();
  entry void done();
 };

 uchare array [1D] Hello {
  entry void hello( int callee);
  entry void buybuy( int callee);
 };
 array [1D] uChareSet<Hello, CProxy_Hello
   CBase_Hello>;
};
```

Рис.3. Пример .ci файла с массивом uchare-объектов.

Использование uchare-объектов позволяет: во-первых, агрегировать объекты и хранить их более оптимально, чем обычные chare-объекты; во-вторых, автоматически агрегировать короткие сообщения, которые возникают при вызове entry-методов у uchare-объектов.

Для оценки проивзодительности были выбраны два оценочных теста: HPCC RandomAccess [8] и Asynchronous Breadth-First Search (ABFS). Тест HPCC RandomAccess реализует псевдослучайный доступ к большому массиву данных, единица измерения производительности GUP/s (Giga Updates Per Second). Тест ABFS реализует поиск достижимых вершин в графе от заданной вершины (корня), в отличие от стандартного BFS, используемого в Graph500 [9], в асинхронном BFS отсутствует глобальная синхронизация после обхода каждого уровня (т. е. решается задача только поиска достижимых вершин, без построения дерева предков). Единица измерения производительности GTE/s (Giga Traversed Edges Per Second).

Проводилось сравнение трех реализации: классической Charm++ реализации, реализаций, использующих библиотеку агрегации коротких сообщений TRAM [10] и uChareLib.

В качестве тестовых платформ использовались: суперкомпьютер МВС10П, установленный в МСЦ РАН, и 36-узловой прототипный кластер с отечественной выосокоскоростной коммуникационной сетью «Ангара» [11, 12], установленный в АО «НИЦЭВТ».

На рисунке 4 приведены результаты запуска теста RandomAccess на вычислительном комплексе МВС10П. На рисунке представлены зависимость производительности (GUP/s) от числа узлов. Количество используемых ядер – 16. Размер в таблицы – 2^{20} элементов (элемент – 64-разрядное слово) на одно ядро, то есть на один узел размер таблицы составляет 16 x 2^{20}. С увеличением числа узлов таблица увеличивается пропорционально (т. е. тест исследуется в режиме weak scaling). Количество используемых chare-объектов (для Charm++ и TRAM) и uchare-объектов (для uChareLib) – 16384, то есть имитируется приложение с мелкозернистым параллелизмом.

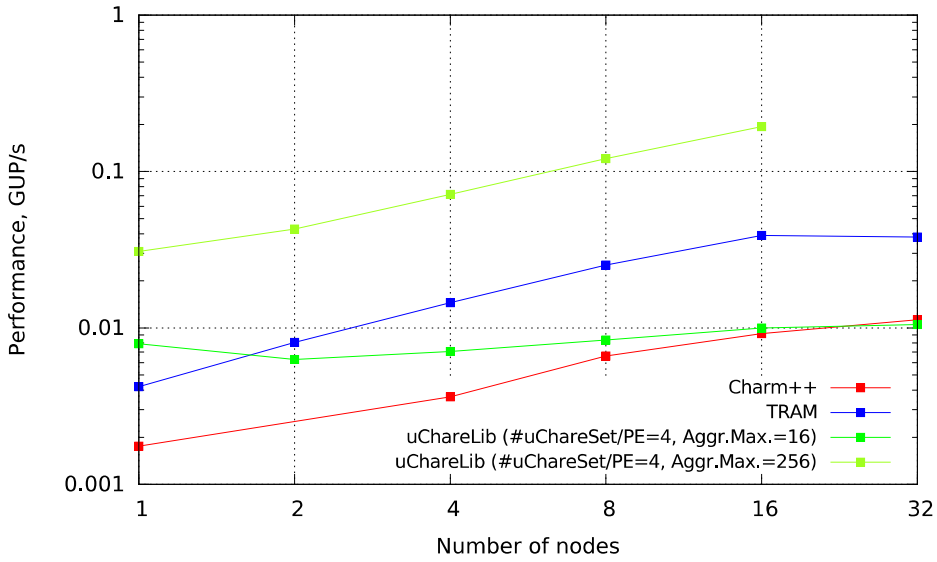

Рис.4. RandomAccess на МВС10П: количество chare, uchare-объктов/PE – 16384.

Как видно из рисунка, наименьшую производительность показала реализация теста RandomAccess на классической программной модели Charm++. Использование библиотеки агрегаций сообщений TRAM позволили увеличить производительность в 10 раз относительно Charm++. Вместе с тем результаты uChareLib превосходят TRAM также в 10 раз. При этом, важным параметром является размер агрегируемого сообщения (на рисунке Aggr.Max). Уменьшение этого параметра с 256 коротких сообщений в одном агрегированном до 16 приводит к падению производительности uChareLib-реализации до уровня Charm++. Для библиотеки TRAM размер буфер для агрегируемых сообщений равен 1024.

На рисунке 5 приведены результаты запуска теста ABFS на вычислительном комплексе МВС10П. На рисунке представлены зависимость производительности (MTE/S) от числа узлов. Количество используемых ядер – 16. Размер графа – 2^{16}, средняя степень вершин – 16. Граф генерируется с равномерно случайным распределением дуг между вершинами.

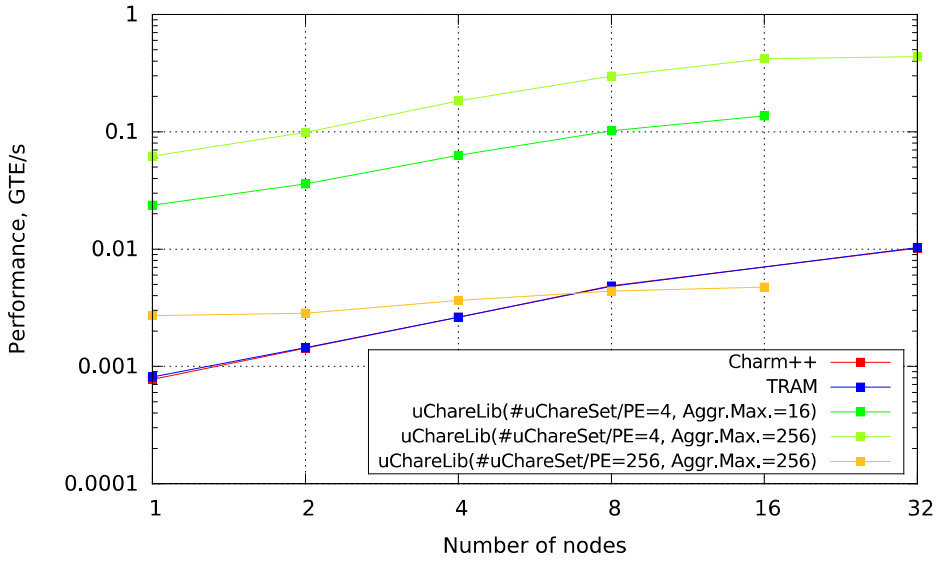

Рис.5. ABFS на МВС10П (равномерно случайный граф, размер графа $N=2^{16}$, $K=16$).

Как видно из рисунка, производительность реализаций ABFS на Charm++ и

TRAM одинакова. Использование uChareLib дает резкий прирост производительности в 44 раза. Также как и в случае с RandomAccess производительность uChareLib крайне чувствительна к параметрам: Aggr.Max – максимальный размер агрегируемого сообщения и количеству chare-объеквтов (uChareSet), используемых для агрегации uchare-объектов.

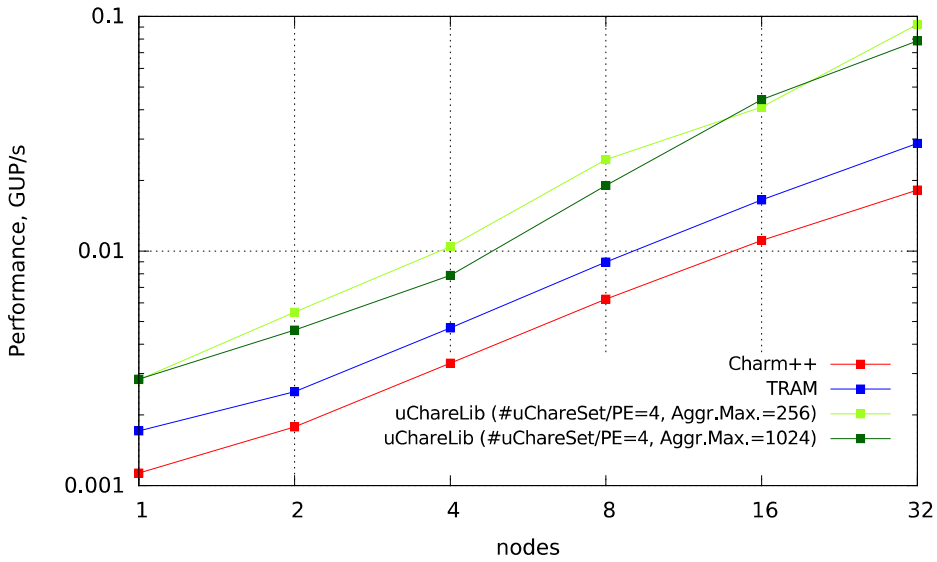

Рис.6. RandomAccess на 36-узловом кластере с сетью «Ангара».

На рисунке 6 приведены результаты запуска теста RandomAccess на 36-узловом вычислительном кластере на базе коммуникационной сети «Ангара». Узлы кластера объедирены в 3-х мерный тор – 3x3x4. Количество используемых ядер – 8.

Как видно из рисунка, соотношение производительности исследуемых реализаций теста RandomAccess аналогично результатам, полученным на МВС10П. Наименьшую производительность показал тест, реализованный на Charm++, далее TRAM и uChareLib.

ЗАКЛЮЧЕНИЕ:

В статье представлена расширенная программная модель Charm++, ориентированная на обработку больших графов на экзафлопсных системах. Предлагаемые расширения включают: специализацию chare-объектов по целевой архитектуре и граннулярности параллелизма, поддержку динамических доменов синхронизации, многоуровневой глобально адресуемой распределенной общей памяти, а также поддержку проблемно-ориентированных языков программирования.

Показан эффект от одного из предлагаемых расширений – поддержки микро-chare-объектов, на двух тестах HPCC RandomAccess и Asynchronous BFS. Ускорение, полученное от использования микро-chare-объектов, составило x5 для RandomAccess и x44 раза для ABFS на МВС10П. Для 36-узлового кластера с сетью «Ангара» на тесте RandomAccess ускорение uChareLib относительно библиотеки TRAM составило 3 раза.

Дальнейшие планы по данному направлению включают исследование uChareLib на разнообразных графовых задачах и большом числе узлов, реализацию остальных предложенных расширений модели Charm++.

Работа выполнена при поддержке гранта РФФИ №15-07-09368.

Литература

1. J. A. Ang, R. F. Barrett, R. E. Benner, D. Burke, C. Chan, J. Cook, D. Donofrio, S. D. Hammond, K. S. Hemmert, S. M. Kelly, H. Le, V. J. Leung, D. R. Resnick, A. F. Rodrigues, J. Shalf, D. Stark, D. Unat, and N. J. Wright. 2014. Abstract machine models and proxy architectures for exascale computing. In Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing (Co-HPC '14). IEEE Press, Piscataway, NJ, USA, 25-32. DOI=http://dx.doi.org/10.1109/Co-HPC.2014.4
2. Anrew Lumsdaine, New Execution Models are Required for Big Data at Exascale
3. Laxmikant V. Kale and Sanjeev Krishnan. Charm++: A portable concurrent object oriented system based on c++. SIGPLAN Not., 28(10):91–108, October 1993.
4. Lukasz Wesolowski, How to Write a Parallel GPU Application Using CUDA and Charm++, Charm Workshop 2010
5. Phil Miller, Aaron Becker, and Laxmikant Kalé. 2012. Using shared arrays in message-driven parallel programs. Parallel Comput. 38, 1-2 (January 2012), 66-74. DOI=http://dx.doi.org/10.1016/j.parco.2011.10.005
6. Aaron Becker, Phil Miller, Laxmikant V Kalé PGAS in the message-driven execution model 1st Workshop on Asynchrony in the PGAS Programming Model APGAS 2009
7. Sungpack Hong, Hassan Chafi, Edic Sedlar, and Kunle Olukotun. 2012. Green-Marl: a DSL for easy and efficient graph analysis. In Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII). ACM, New York, NY, USA, 349-362. DOI=http://dx.doi.org/10.1145/2150976.2151013
8. Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The hpc challenge (hpcc) benchmark suite. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC ’06, New York, NY, USA, 2006. ACM.
9. Graph500, http://www.graph500.org.
10. Lukasz Wesolowski, Ramprasad Venkataraman, Abhishek Gupta, JaeSeung Yeom, Keith Bisset, Yanhua Sun, Pritish Jetley, Thomas R. Quinn, and Laxmikant V. Kale. TRAM: Optimizing Fine-grained Communication with Topological Routing and Aggregation of Messages. In Proceedings of the International Conference on Parallel Processing, ICPP '14, Minneapolis, MN, September 2014
11. Симонов А.С., Слуцкин А.И., Макагон Д.В., Сыромятников Е.Л., Жабин И.А., Фролов А.С., Щербак А.Н. Опыт разработки отечественной высокоскоростной коммуникационной сети для суперкомпьютеров. – Доклад на Третьем Московском суперкомпьютерном форуме (МСКФ-2012), Россия, г. Москва, ВВЦ, 01 ноября 2012 г. http://www.ospcon.ru/files/media/mscf_2012_tesis.pdf.
12. И.Жабин, Д.Макагон, А.Симонов, Е.Сыромятников, А.Фрлоов, Е.Щербак Кристалл для “Ангары”, Суперкомпьютеры №4(16), 2013, 46-49.