ТРЁХМЕРНАЯ ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ MPI-ТЕСТИРОВАНИЯ КОММУНИКАЦИОННОЙ СРЕДЫ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА

П.С. Банников, А.Н. Сальников

Разработчик программы для вычислительного кластера заинтересован в её эффективности. Эффективной программе необходимо учитывать коммуникационную среду того кластера, для которого она была написана, чтобы произвести оптимальное распределение вычислений по его узлам. Решением может являться ознакомление со спецификацией коммуникационной среды кластера. Однако, в спецификации указано лишь некоторое «идеализированное» описание (например, топология и/или пиковая пропускная способность каналов). На практике этого бывает недостаточно, и возникает потребность в знании пропускной способности каналов во время работы вычислительного кластера. Одним из способов удовлетворения данной потребности может служить проведение серии синтетических тестов. Суть тестов заключается в том, чтобы измерить задержку при передаче MPI-сообщения между узлами кластера. Среди входных параметров тестов — число задействованных узлов кластера и количество различных длин сообщений [2]. На выходе получается несколько матриц с усреднёнными по некоторому принципу значениями и одна матрица с величинами отклонений от средних значений. Матрицы являются *трёхмерными*: по первым двум измерениям откладываются номера узлов кластера, а третье измерение соответствует длине передаваемого сообщения. Размерность матриц задаётся в параметрах тестов *пользователем* (разработчиком программы для кластера), и в дальнейшем пользователь самостоятельно анализирует полученные результаты. При очень малых размерностях возможен анализ вручную, однако для анализа результатов, полученных с современных суперкомпьютеров, необходимы автоматизированные средства. В данной работе предлагается воспользоваться технологиями объёмной визуализации для наглядного отображения трёхмерных матриц. Ставятся две основные задачи: *наглядность* и *работа в режиме реального времени*. Для обеспечения наглядности разработано приложение со следующими возможностями отображения:

- перемещение, поворот, масштабирование трёхмерного объекта;
- установка так называемой «камеры» как внутри, так и за границами трёхмерного объекта;
- отображение в цветовое пространство только тех значений, которые попали в заданный пользователем диапазон;
- визуализация оболочек трёхмерного объёма задаваемой толщины (также служит для ускорения визуализации);
- построение объёмов «2-го уровня»: визуализация только тех клеток матрицы, которые попали в заданный пользователем диапазон значений.

Для обеспечения отображения в реальном времени привлекаются ресурсы графических процессоров (GPU). Для визуализации на GPU используется технология OpenCL. Однако, если видеокарта не поддерживает указанную технологию, визуализация автоматически переходит на центральный процессор (CPU). В этом случае задействуется многоядерность CPU с помощью технологии OpenMP. Таким образом, результаты визуализации возможно просмотреть на любом персональном компьютере.

Трёхмерные матрицы визуализируются в виде трёхмерных решёток, в узлах которых расположены *«точки»* — объекты, обладающие формой, цветом и прозрачностью. Значение в клетке матрицы однозначно соответствует прозрачности «точки». Поддерживаются несколько форм «точек»:

- куб как естественно получающаяся при визуализации трёхмерной решётки фигура;
- шар с постоянной интенсивностью;
- шар с интенсивностью, максимальной в центре и квадратично убывающей к границе.

Входные файлы с матрицами могут быть одного из двух форматов: бинарные (на основе NetCDF) и текстовые (структурированные определённым образом). В текущей реализации входной файл не считывается целиком в оперативную память персонального компьютера. Вместо изначальной матрицы со значениями задержек или дисперсий, представленных числами двойной точности, в памяти хранится трёхмерный массив цветов. В следствие того, что используются максимум две цветовые компоненты, достигается 4-кратный выигрыш по использованию памяти. Если для визуализации снова понадобилась исходная матрица (например, при смене диапазона отображаемых значений), входной файл считывается повторно.

Система визуализации работает в одном из следующих режимов:

- **Визуализация** только **матрицы задержек**.

Матрицей задержек назовём матрицу, содержащую усреднённые значения задержек при передаче MPI-сообщений.

Значения в клетках матрицы отображаются на спектр одного цвета.

- **Визуализация матрицы задержек и матрицы дисперсий** (с одинаковыми размерностями).

Матрицей дисперсий назовём матрицу, содержащую величины отклонений от средних значений.

Матрицы комбинируются друг с другом и визуализируются как одна матрица. Величины задержек отображаются на спектр одного цвета, величины отклонений — на спектр другого цвета. Если величина отклонения превышает величину задержки, данная клетка рисуется только цветом для отклонений.

- **Визуализация сравнения двух матриц задержек** одинаковых размерностей.

Опять же, матрицы комбинируются друг с другом и визуализируются как одна матрица. Положительная разность значений в клетках матриц с одинаковыми индексами отображается на спектр одного цвета, отрицательная — на спектр другого цвета.

Система визуализации разрабатывается как расширение функциональности кроссплатформенной программы Network Viewer 2 [2], входящей в проект PARUS [1], на трёхмерный случай. Здесь и далее система визуализации будет именоваться как «Network Viewer (2)».

В программной реализации для визуализации трёхмерных матриц используется алгоритм *рэйкастинга для объёмов* [3]. Данный алгоритм обеспечивает высокое качество получаемого изображения, поэтому хорошо подходит для решения первой основной задачи (наглядность). Выделяют следующие шаги алгоритма рэйкастинга:

1) «бросание» луча;
2) сэмплирование;
3) раскраска;
4) комбинирование цветов.

Заранее предполагается, что визуализируемый объём представлен в виде набора точек в узлах некоторой решётки и разбит на *слои*, выровненные по самому объёму.

На шаге (1) лучи испускаются из позиции так называемой «камеры» до каждого пикселя конечного изображения. Ищется пересечение луча с примитивом, ограничивающим визуализируемый объём. Если пересечение найдено, ищется пересечение луча с ближайшим к «камере» слоем объёма. Точка пересечения необязательно совпадает с какой-либо точкой слоя, поэтому на шаге (2) определяются точки объёма, ближайшие к точке пересечения. На шаге (3) точке пересечения присваивается цвет, получившийся в результате интерполяции цветов ближайших соседей этой точки. Если используются источники света, на цвет раскрашиваемой точки оказывается влияние посредством выбранной локальной модели освещения. После этого луч движется дальше, и ищется его пересечение со следующим слоем объёма. На шаге (4) цвета всех точек пересечения одного луча и слоёв объёма комбинируются в порядке от ближней точки к дальней на основе определённой функции смешивания; вовлекается полупрозрачность элементов объёма.

К недостаткам алгоритма рэйкастинга относится медленная скорость его работы из-за необходимости частого поиска пересечений. Для уменьшения количества поисков пересечений рекомендуется использование ускоряющих структур, таких как регулярная трёхмерная решётка. В данной работе сам визуализируемый объём выступает в роли ускоряющей структуры.

Достоинством алгоритма рэйкастинга, помимо высокого качества конечного изображения, является возможность совершенно независимо проводить расчёты для **каждого** пикселя изображения. Отсюда следует естественный параллелизм алгоритма, что широко используется в реализации Network Viewer 2.

Для осуществления прохождения луча по трёхмерной решётке используется алгоритм *3DDA* [4]. К достоинствам алгоритма относятся численная устойчивость, относительная простота реализации и достаточно высокая скорость работы. Под численной устойчивостью будем понимать, главным образом, корректную работу алгоритма даже в тех случаях, когда аппаратура выдаёт исключения для чисел вещественной арифметики с плавающей запятой.

Реализация на GPU выполнена через *«ядра»*. В спецификации OpenCL «ядро» определяется как функция, работающая на устройстве (в данном случае — на видеокарте). На текущий момент написано 6 «ядер»: 3 «ядра» для реализации построения оболочек объёма и 3 «ядра» для реализации построения объёмов «2-го уровня». Каждое из трёх «ядер» соответствует определённой форме «точки» объёма. Большое количество «ядер» объясняется намерением минимизировать количество команд потока управления (например, ветвлений), которые отрицательно сказываются на производительности. За выбор и запуск «ядер» отвечает центральный процессор.

Реализация на CPU выполнена через функции языка C++ с использованием технологии OpenMP. Количество функций совпадает с количеством OpenCL-»ядер» по той же причине, что и в реализации на GPU. Выбор активной функции осуществляется через указатели на функции.

Наконец, проводится сравнение качества получившегося изображения и скорости работы программы Network Viewer 2 с существующими кроссплатформенными программами, разработанными для схожих целей: ParaView 3.14.1 [5] и VisIt 2.4.2 [6]. Данные программы являются свободно распространяемыми, с открытым исходным кодом; обладают богатым функционалом для обеспечения наглядности визуализации, а также работают в режиме реального времени.

Машина, на которой проводились сравнения, имеет 4-ядерный процессор AMD Phenom II с 2 ГБ оперативной памяти и видеокарту NVIDIA GeForce GT240 с 1 ГБ памяти.

Всем трём программам на вход подавался NetCDF-файл, содержащий матрицу с результатами теста коммуникационной среды суперкомпьютера «Ломоносов», установленного в МГУ им. М.В. Ломоносова. В

тесте производился обмен MPI-сообщениями между 500 процессорами для 100 различных длин сообщений (матрица 500x500x100). Результаты работы программ Network Viewer, ParaView и VisIt представлены на рисунках [1], [2] и [3] соответственно:

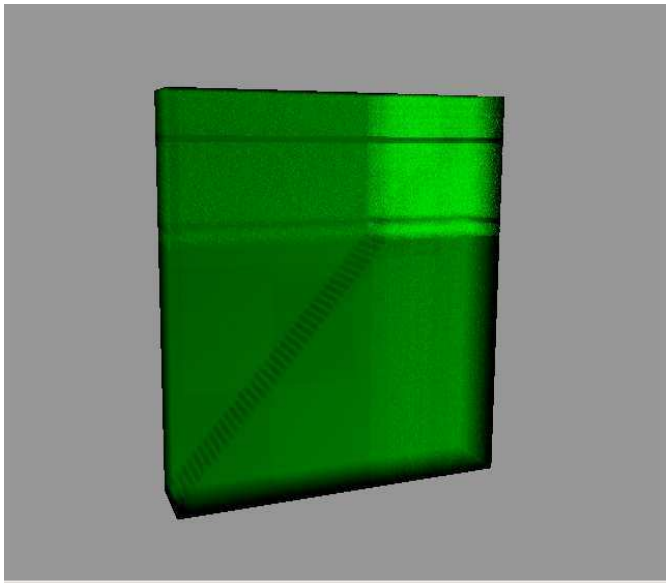

Рис. 1 "Ломоносов", 500x500x100. Network Viewer 2

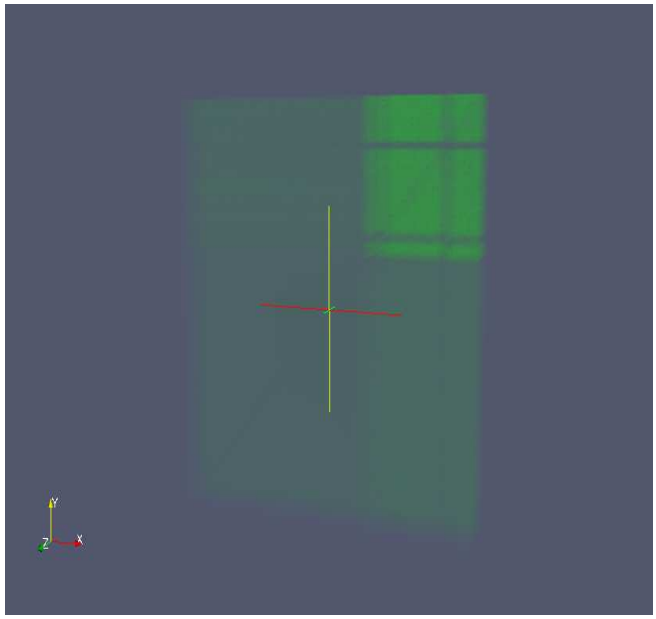

Рис. 2 "Ломоносов", 500x500x100. ParaView 3.14.1

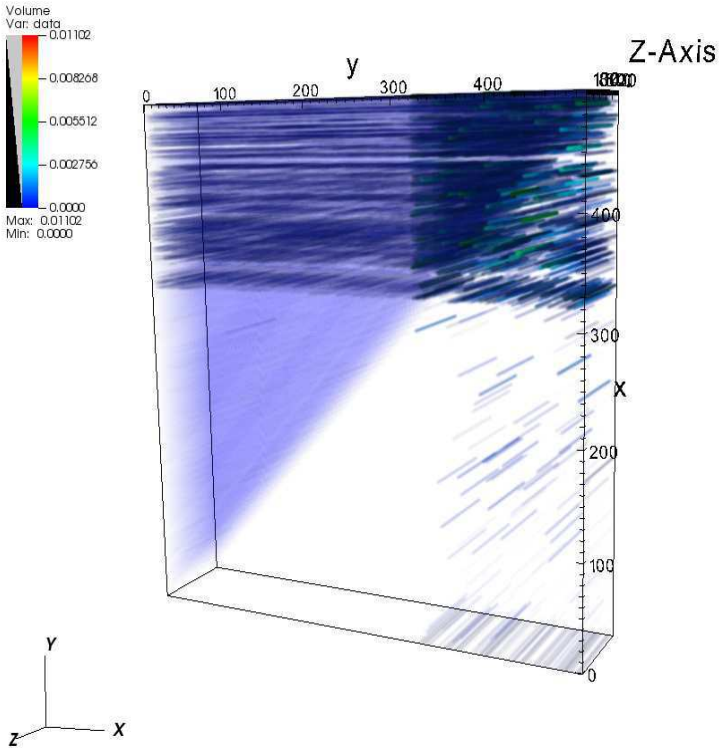

Рис. 3 "Ломоносов", 500x500x100. VisIt 2.4.2

VisIt 2.4.2 распознавал во входном файле не трёхмерную структуру, а двумерную, поэтому для корректности результатов сравнения было решено воспользоваться встроенными средствами самого VisIt 2.4.2: к двумерной структуре применялось геометрическое преобразование «Extrude» размером 100. В частности, это объясняет ярко выраженные полосы на рисунке [3].

Результаты работы программ сведены в следующую таблицу:

Таблица 1

	ParaView 3.14.1	VisIt 2.4.2	Network Viewer 2	
			OpenMP	OpenCL
Отрисовка (1 раз), FPS	1	60	11	37
Отрисовка, FPS	31	67		105
Подготовка (1 раз), с	8	?	1.39	
Подготовка, с	0.28	?	0.68	
Потребление памяти, МБ	200+	?	50+	

Программы запускались на одних и тех же данных несколько раз подряд, и было замечено, что время первого запуска сильно отличается от времени последующих запусков.

VisIt 2.4.2 не предоставляет данных о потреблении памяти и времени считывания входного файла, поэтому в таблице стоят символы «?».

Как видно из таблицы, Network Viewer 2 требует в 4 раза меньше памяти, чем ParaView 3.14.1, а визуализация на GPU в Network Viewer 2 проходит в 1.7-3 раза быстрее, чем в ParaView и VisIt. Реализация на CPU, однако, уступает обеим программам в 3-6 раз.

В следующем тесте программам ParaView и VisIt на вход подавался один NetCDF-файл, содержащий матрицу с результатами теста коммуникационной среды суперкомпьютера «Европа». Размеры матрицы — 2048x2048x25. Программе Network Viewer 2 на вход подавалось два файла такого размера и задавался режим работы «матрица задержек + матрица дисперсий». Ввиду того, что ни ParaView 3.14.1, ни VisIt 2.4.2 не поддерживают режим комбинирования матриц, представлен только результат работы Network Viewer 2 на рисунке [4]:

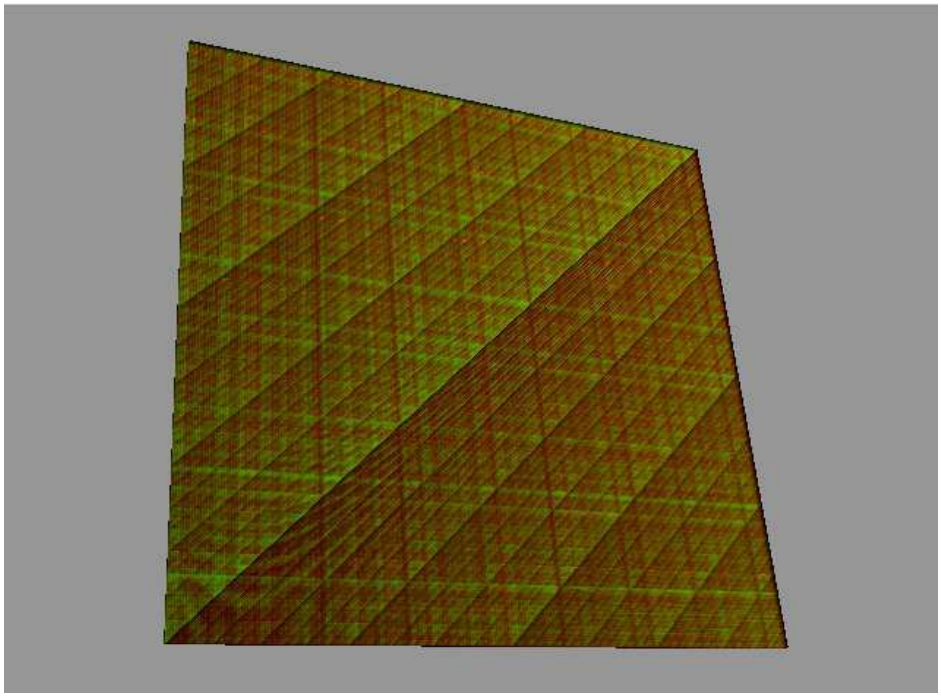

Рис. 4. "Европа", 2048x2048x25. Network Viewer 2

При работе в VisIt 2.4.2 снова было применено преобразование «Extrude» размером 25. Результаты работы программ сведены в следующую таблицу:

Таблица 2

	ParaView 3.14.1	VisIt 2.4.2	Network Viewer 2	
			OpenMP	OpenCL
Отрисовка (1 раз), FPS	-	37	6.5	5
Отрисовка, FPS	-	64		135
Подготовка (1 раз), с	19	?	27.2	
Подготовка, с	15	?	25.7	
Потребление памяти, МБ	800+	?	200+	

Комментарии те же, что и к первой таблице, с той лишь разницей, что в ParaView 3.14.1 не удалось получить изображение. Возросшее число кадров в секунду при реализации на OpenCL (135 против 105) объясняется малой «толщиной» матрицы в направлении взгляда.

Работа проводится при поддержке грантов РФФИ: 11-07-00756-а, 11-07-00614-а и госконтрактов по ФЦП "Научные и научно-педагогические кадры инновационной России": П873,П1258, П1317.

ЛИТЕРАТУРА:

1. A.N. Salnikov "PARUS: A Parallel Programming Framework for Heterogeneous Multiprocessor Systems" // 2006 – Lecture Notes in Computer Science, vol.4192, pp.408–409
2. А.Н. Сальников, Д.Ю. Андреев, Д.Ю. Лебедев "Инструментальная система для анализа характеристик коммуникационной среды вычислительного кластера на основе функций стандарта MPI" // 2012 – Вестник Московского университета, №1, Москва, изд-во МГУ, с.39–48
3. J. Pawasauskas "Volume Visualization With Ray Casting" // CS 563 - Advanced Topics in Computer Graphics, 1997 - [Электронный ресурс]. – URL: http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm
4. Алгоритм 3DDA // [Электронный ресурс]. – URL: http://www.ray-tracing.ru/articles182.html
5. ParaView // [Электронный ресурс]. – URL: http://www.paraview.org/paraview/project/about.html
6. VisIt // [Электронный ресурс]. – URL: https://wci.llnl.gov/codes/visit/about.html