МЕТОД ПАРАЛЛЕЛЬНОГО ПОСТРОЕНИЯ ИЗОБРАЖЕНИЙ ПРИ ВИЗУАЛИЦАЗИИ НА BLUEGENE/P

К.В. Новиков, О.В. Джосан

В настоящее время активно развивается прикладное программное обеспечение для проведения научных исследований. В частности, разрабатываются алгоритмы и программные средства для проведения междисциплинарных научных исследований с использованием суперкомпьютеров. Важным вопросом для исследования в этой области является визуализация научных данных различной природы. При этом критичным оказывается как качество картинки, так и время, которое тратится на построение изображения. Если объект или процесс, который требуется визуализировать, является довольно сложным, а требования к качеству картинки довольно высоки, то процесс построения изображений требует значительных вычислительных ресурсов. По этой причине в настоящее время активно развиваются методы параллельного построения изображений.

Данная работа посвящена исследованию различных методов параллельного построения изображений для системы визуализации научных данных на многопроцессорных вычислительных комплексах. Проводится обзор различных стратегий параллельного построения изображений. Проводится исследование параллельного построения изображений на многопроцессорном вычислительном комплексе BlueGene/P, установленном в Московском государственном университете имени М.В. Ломоносова. Реализован алгоритм параллельной обратной трассировки лучей для построения изображений. Проведено исследование зависимости времени вычислений от различных параметров задачи: размера изображения, количества объектов, количества используемых процессоров и т.д. Исследована масштабируемость предложенного решения.

Методы трассировки лучей на сегодняшний день считаются наиболее мощными и универсальными методами создания реалистичных изображений. Известно много примеров реализации алгоритмов трассировки для качественного отображения самых сложных трехмерных сцен. Универсальность методов трассировки обусловлена тем, что в их основе лежат простые и ясные понятия, отражающие человеческий опыт восприятия окружающего мира. Будем полагать, что окружающие объекты обладают по отношению к свету следующими свойствами: излучают; отражают и поглощают; пропускают сквозь себя. Каждое из этих свойств можно описать набором характеристик. Например, излучение можно охарактеризовать интенсивностью, направленностью, спектром. В методе обратной трассировки лучи света рассчитываются и трассируются в обратном направлении. Через каждый пиксель экрана в направлении от наблюдателя проводится луч и трассируется до пересечения с объектом сцены. Далее рассчитывается отраженный луч (в направлении на источник света), чтобы учесть световую энергию, которая может быть принесена в направлении наблюдателя. Отраженный луч анализируется аналогично. Для определения цвета пикселя изображения нужно учитывать свойства объекта, а также то, какое световое излучение приходится на соответствующую точку объекта. Если объект зеркальный (хотя бы частично), испускается отраженный луч и трассируется рекурсивно. Результат трассировки отраженного луча учитывается при определении цвета пикселя пропорционально коэффициенту отражения. Когда выяснится, что текущий луч обратной трассировки не пересекает какой-либо объект, а уходит в свободное пространство, то на этом трассировка для этого луча заканчивается.

Классификация параллельных методов получения изображений является общепринятой. Суть задачи получения графического изображения – вычисление воздействия каждого примитива на каждый пиксель. В результате примитивы могут попасть на экран или нет. Следовательно, построение изображения может рассматриваться как проблема сортировки примитивов.

Таким образом, предлагается классификация по тому, в каком месте параллельного графического конвейера происходят сортировки. Во время геометрической обработки – “sort-first” сортировки выполняются вначале, в естественной точке разрыва конвейера – “sort-middle” в середине, во время растеризации – “sort-last” в конце. Выбор каждого из этих классов приводит к разным параллельным алгоритмам построения изображений с различными свойствами.

Sort-first. Целью этого подхода является распределение примитивов на самом раннем этапе графического конвейера – во время геометрической обработки. Область экрана разбивается на части, количество которых совпадает с количеством процессоров, и каждый процессор занимается построением одной области конечного изображения.

Алгоритм состоит из следующих шагов:1) Объекты распределяются по вычислительным узлам произвольным образом. 2) Каждый узел вычисляет области экрана, куда попадают соответствующие ему объекты. 3) Происходит перераспределение данных на нужные процессоры. 4) Выполняется геометрическая обработка и растеризация.

Графический конвейер целиком реализуется независимо от других процессоров, что является преимуществом данного подхода, также как низкие коммуникационные затраты. К недостаткам можно отнести чувствительность к неравномерно распределённым данным и сложность реализации.

Sort-middle. В “sort-middle” алгоритмах объекты перераспределяются в середине графического конвейера, то есть между геометрической обработкой и растеризацией. Таким образом, перед перераспределением объекты переведены в экранные координаты и готовы к растеризации.

Геометрическая обработка и растеризация обычно выполняются на разных процессорах. Геометрические процессоры производят необходимые вычисления с объектами, после чего классифицируют их по расположению на экране. Плоские объекты распределяются соответствующим образом между процессорами, где происходит их дальнейшая обработка.

Данный алгоритм считается основным, потому что перераспределение данных происходит в естественном месте графического конвейера.

Однако такой подход имеет и существенные недостатки:

- необходимость передачи большого объема данных между процессорами;
- отсутствие уравновешивания нагрузки на вычислительные узлы при неравномерно распределённых данных.

Sort-last. “Sort-last” алгоритм произвольным образом распределяет объекты между всеми вычислительными узлами. Каждый узел строит растровое изображение своего подмножества объектов независимо от того, в какой области экрана они будут расположены. Полученное изображение пересылается на специальный процессор, который совмещает все частично отрисованные кадры в один. Последний шаг алгоритма является очень трудоёмким, т.к. требует обработки огромных пиксельных данных.

Существует два различных метода реализации последнего шага алгоритма: 1) Передавать только ту часть изображения, которая участвовала в растеризации *(SL-sparce)*. Этот подход направлен на минимизацию коммуникационных затрат. 2) Передавать изображение целиком *(SL-full)*. Преимущество данного подхода состоит в том, что совмещение регулярных данных может быть легко реализовано на аппаратном уровне.

Реализация метода параллельного построения изображений на BlueGene/P. Для реализации был выбран метод обратной трассировки лучей. Распараллеливание вычислений на n вычислительных узлов осуществляется следующим образом:

- область экрана разбивается по высоте на n равных частей;
- данные копируются на все узлы;
- каждый процесс строит свою область изображения независимо от других;
- на процессе с номером 0 происходит совмещение всех частей в результирующее изображение.

Сбор данных на одном процессоре осуществляется при помощи метода рекурсивного удвоения. Для повышения качества изображения через каждый пиксель экрана трассируется не один, а c лучей, где c – коэффициент сглаживания. Аналогичного эффекта можно было бы добиться, построив изображение в c раз большее по размеру, разбив его на квадраты площади c и заменив каждый квадрат пикселем с усредненным значением.

Выбранный метод реализован на языке программирования C++ с использованием библиотеки MPI для распараллеливания вычислений. Программа принимает на вход XML-файл, который содержит данные, необходимые для визуализации (сцена). В качестве результата генерируется изображение в заданном разрешении в формате TGA[5]. Для импорта XML-файла используется сторонняя библиотека[6]. Входной файл имеет следующую структуру.

Листинг 1. Пример XML-файла, задающего структуру.

```
<?xml version="1.0" encoding="UTF-8"?>
<inputset>
 <image>
  <width>1280</width>
  <height>800</height>
  <objects>
   <object id="1" type="sphere">
    <shape>
     <center>(54.3058, 184.082, 9851.02)</center>
     <radius>53.345</radius>
    </shape>
    <materialproperty>
     <ambient>(0, 0, 0)</ambient>
     <diffusion>(0.69986, 0.25293, 0.829437)</diffusion>
     <specular>(0.110535, 0.0764465, 0.0798645)</specular>
     <shininess>(0, 0, 0)</shininess>
     <emission>(0, 0, 0)</emission>
     <reflection>0.571411</reflection>
     <refraction>0</refraction>
     <density>0</density>
     <power>97</power>
    </materialproperty>
```

```
    </object>
    ...
   </objects>
   <lights>
    <light id="1">
     <origin>(-7339.48, 3472.9, 2066.04)</origin>
     <intensity>(0.954651, 0.823792, 0.46286)</intensity>
    </light>
    ...
   </lights>
  </image>
</inputset>
```

Таким образом, входной файл содержит разрешение изображения, список объектов и список источников освещения. Описание каждого объекта включает в себя его тип (на данный момент реализована только сфера), форму (в зависимости от типа) и свойства материала. Каждый источник света задается его расположением и интенсивностью.

В программе реализованы следующие классы:

- SVector – простая реализация вектора (с точки зрения математики). Реализованы такие операции как сложение и вычитания векторов, умножение на число, скалярное произведение, норма, нормировка.
- TRay – луч; состоит из точки, задающей начало, и вектора, задающего направление.
- Свойства материала TMaterialProperty: цвет фонового отражения, цвет рассеянного отражения, цвет зеркального отражения, цвет собственного излучения, коэффициент блеска, коэффициент отражения, коэффициент преломления, полупрозрачность. В данной реализации алгоритма используется только фоновое и рассеянное освещение, а также коэффициент отражения.
- TShape – абстрактный класс, представляющий форму объектов. Для добавления новых типов объектов в программу достаточно создать новый класс, наследуемый от TShape, и реализовать в нем функцую пересечения с лучом.
- TSphere представляет сферическую форму объектов, наследуется от TShape.
- TObject состоит из формы и свойств материала.
- Класс TScene представляет собой данные для отображения.
- TViewport является основным классом в программе. Содержит все необходимые данные для построения изображения.

```
class TViewport
{
      int Width;
      int Height;
      COLOR *ImageMatrix;
      TScene *Scene;
      ...
      public:
      ...
      bool ConfigureFromFile(char *filename);
      bool Render(int rank, int size);
      bool DataCollection(int rank, int size);
      bool SaveToTGAFile(char *filename);
}
```

Используя описанный интерфейс, параллельное построение изображения сводится к следующему:

- с помощью метода TViewPort::ConfigureFromFile() считываются данные на каждом вычислительном узле;
- TViewPort::Render() отрисовывает часть изображения, соответствующую данному узлу;
- части построенного изображение собираются на узле с номером 0 с помощью метода TViewPort::DataCollection().

Результаты тестирования на BlueGene/P.

Представленная реализация параллельного построения изображений протестирована на архитектуре BlueGene/P, используя компилятор IBM XL C++. Для тестирования была сгенерирована случайная сцена, состоящая из 100 объектов и 4 источников освещения, а также сцены, состоящие из 90, 80, ..., 10 объектов, являющиеся подмножеством первой. Пример результата работы программы представлен на Рис.1.

Ускорение и эффективность. В целях изучения поведения программы были проведены измерения на 1, 2, 4, ..., 256 вычислительных узлах. Результаты для нескольких сцен показаны на Рис.2a и Рис.2b, на которых видно, что ускорение линейно зависит от количества узлов. При построении изображения размером 1280x800 пикселей на сцене, состоящей из 100 объектов, получено ускорение почти в 122 раза на 256 узлах (эффективность 47.5%).

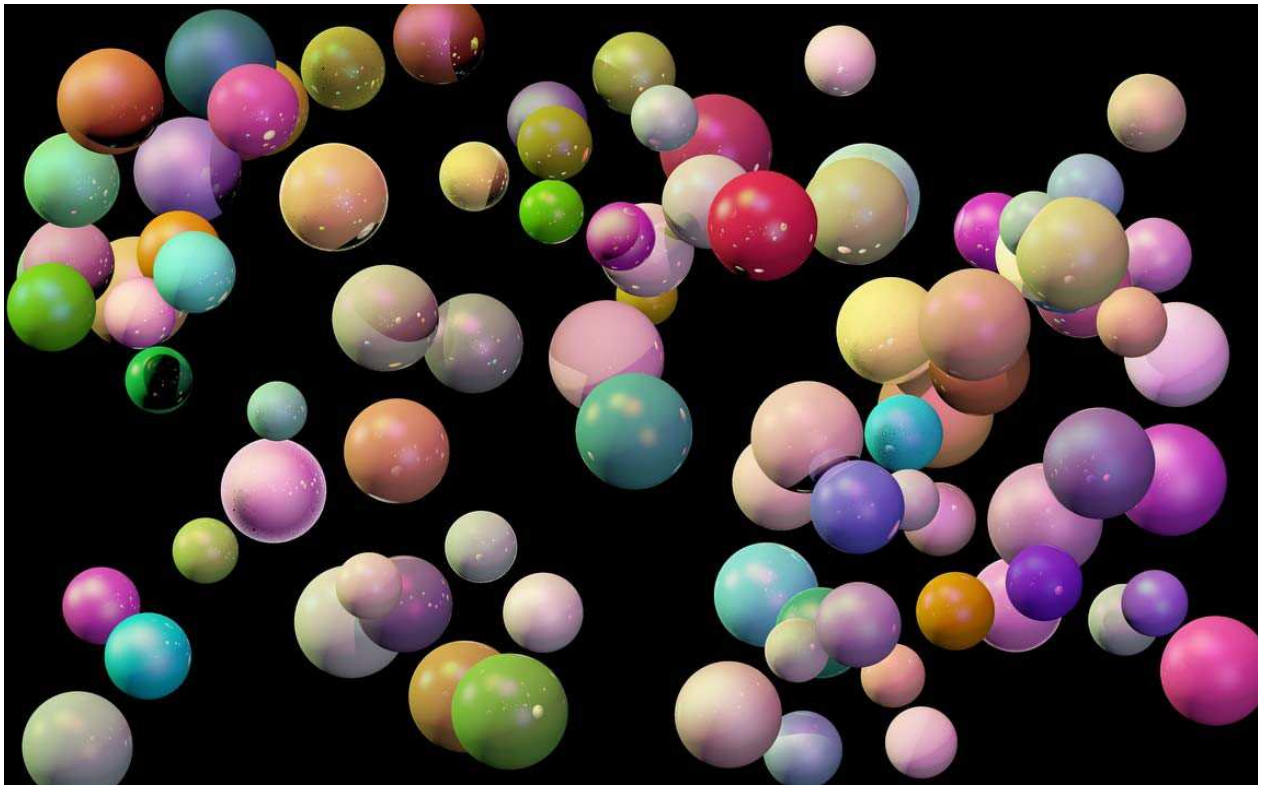

Рис.1 Изображение, построенное по выбранной для тестирования сцене со 100 объектами и 4 источниками освещения.

Масштабируемость по объему данных. Для анализа масштабируемости по объему данных программа была запущена для построения изображений размером 1280x800 пикселей на 10 различных сценах, количество объектов в которых колеблется от 10 до 100. Для получения более точного результата сцены сгенерированы таким образом, что каждая следующая получается из предыдущей добавлением 10 объектов. На Рис.2c можно проследить линейный рост времени построения изображения при увеличении объема данных.

Масштабируемость по размеру изображения. Проведены измерения времени работы программы при построении изображений различных размеров (от 320x200 до 6400x4000) фиксированной сцены, состоящей из 100 объектов, на 128 вычислительных узлах. На Рис.2d показаны результаты измерений времени работы программы, а на Рис.2e – отношения времени работы программы к количеству пикселей (оценка среднего времени трассировки одного луча).

На первом графике прослеживается парабола. Так как по оси x линейный размер изображения растет линейно, количество пикселей на изображении растет квадратично. Отсюда можно сделать вывод о линейной зависимости времени работы программы от количества пикселей в изображении. Это подтверждает Рис.2f, на котором видно, что время, приходящееся на один пиксель, приблизительно равно константе, при заданных параметрах равную $7{,}7 \cdot 10^{-6}$ секунд.

В представленной работе проведен обзор основных существующих методов параллельного построения изображений. Выполнена параллельная реализация метода обратной трассировки лучей для сцены, состоящей из сферических объектов. Проведено тестирование реализованного алгоритма на архитектуре BlueGene/P. Проанализированы полученные результаты и эффективность метода при различных конфигурациях.

Результаты тестирования показали, что представленный алгоритм почти линейно масштабируем по количеству вычислительных узлов, количеству объектов и размеру изображения. На 128 процессорах получено ускорение в 72 раза (55.9%), а на 256 процессорах – в 122 раза (47.6%).

В дальнейшем предполагается работа над следующими аспектами: 1) Оптимизация реализованного алгоритма. 2) Разработка распределенной модели данных и адаптация алгоритма к данным больших объемов. 3) Добавление различных типов объектов и текстур. 4) Реализация таких явлений как прозрачность и преломление.

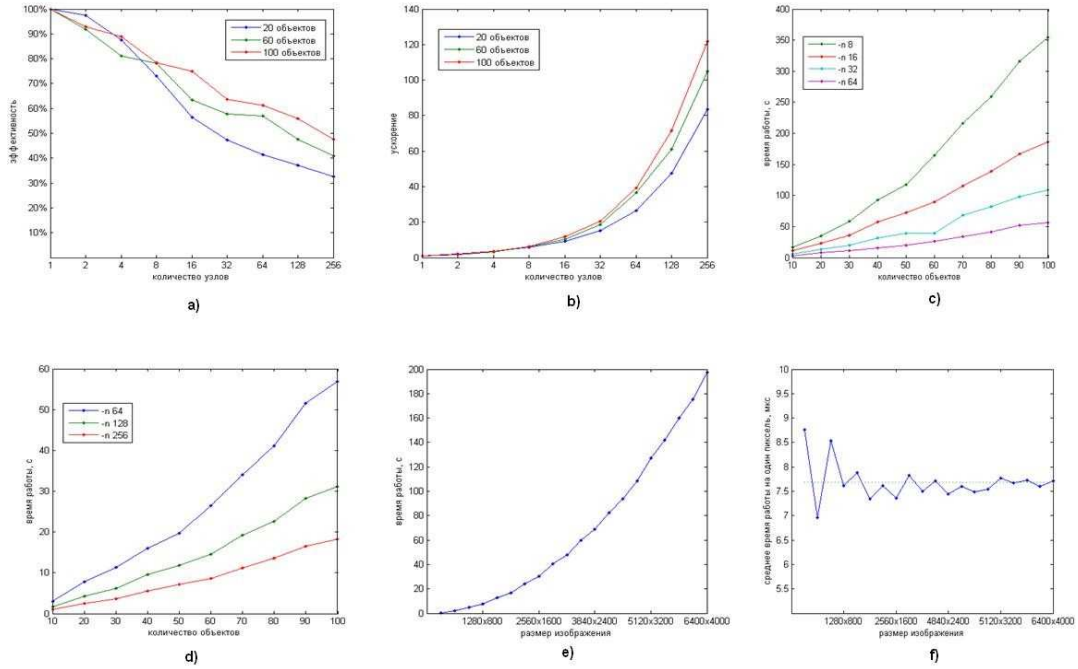

Рис.2 Результаты тестирования разработанного алгоритма построения изображений на системе BlueGene /P.

Данная работа выполнена в рамках НОЦ «Суперкомпьютерные технологии» при поддержке Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы и грантов РФФИ 08-07-00445-а, 09-07-12068-офи_м.

ЛИТЕРАТУРА:

1. S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, “A sorting classification of parallel rendering,” *IEEE Computer Graphics and Applications*, 1994.
2. A. Cedilnik, B. Gevici, K. Moreland, J. Ahrens, J. Favre, “Remote Large Data Visualization in the ParaView Framework,” *Eurographics Parallel Graphics and Visualization*, May 2006.
3. P. Carns, R. Ross, K. Iskra, S. Lang, “24/7 Characterization of Petascale I/O Workloads,” *Proceedings of 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, September 2009.
4. Д. Манаков, “Анализ параллельных визуальных технологий,” *Институт математики и механики УрО РАН*, Екатеринбург, Россия, 2007.
5. http://local.wasp.uwa.edu.au/~pbourke/dataformats/tga/ – TGA Format Specification
6. M. Choudrakis, C++ XML Library, http://www.turboirc.com/forum