УДК 004.274

Р.В. Мальчева (канд. техн. наук, доц.), Мохаммад Юнис
ГВУЗ «Донецкий национальный технический университет», г. Донецк
E-mail: raisa@cs.dgtu.donetsk.ua

РЕАЛИЗАЦИЯ МОДИФИЦИРОВАННОГО АЛГОРИТМА ТРАССИРОВКИ ЛУЧЕЙ НА КЛАСТЕРЕ NeClus

Выполнен анализ аппаратных и программных средств кластера NeClus ДонНТУ. Рассмотрена реализация модифицированного алгоритма со строчной интерполяцией на кластере NeClus для изображений размером 600 на 600 и 1800 на 1200 пикселей. Проведен анализ результатов формирования изображений и приведены количественные оценки эффективности применения параллельной реализации на кластере. Также выполнена оценка влияния длины сегмента пикселей и коэффициента различия в цвете на время формирования изображения.
Ключевые слова: *трассировка лучей, кластер, интерполяция, длина сегмента, скорость.*

Введение

Более 50 лет развития систем визуализации, особенно, реального систем времени, привели к значительному разнообразию как алгоритмических, так и архитектурных решений. Реализации графических систем можно сравнивать по комплексному показателю – отношению качества полученного изображения к аппаратурной сложности системы, т.е. ее стоимости. Современные графические системы позволяют для каждого конкретного применения формировать свою архитектуру системы. Графические аппаратные и программные системы являются высокопроизводительными и, в то же время, достаточно дорогими [1]. В связи с этим актуальной задачей на современном этапе является правильный выбор конфигурации системы, т.е. достижение требуемого качества путем минимальных затрат. Метод трассировки лучей используется в компьютерной графике для формирования трехмерных сцен фотографического качества. Для сокращения времени работы алгоритма применяются как решения, направленные на ускорение анализа пересечения луча с объектами сцены, так и методы его распараллеливания. В [2-4] предложен модифицированный алгоритм трассировки лучей, который предусматривает выполнять формирование и отслеживание лучей для меньшего, чем разрешение изображения, количества пикселей, а цветовые значения внутренних пикселей получать при помощи строчной или блочной интерполяции.

В данной работе рассматривается реализация модифицированного алгоритма со строчной интерполяцией на кластере NeClus ДонНТУ для выбора длины строчного сегмента и оценки эффективности применения интерполяции.

Анализ аппаратно-программных средств для реализации алгоритма

Кластеры используются в вычислительных целях, в частности в научных исследованиях. Для вычислительных кластеров существенными показателями являются высокая производительность процессора в операциях над числами с плавающей точкой (flops) и низкая латентность объединяет сети, и менее существенными - скорость операций ввода-вывода, которая в большей степени важна для баз данных и web-сервисов.

Вычислительные кластеры позволяют уменьшить время расчетов, по сравнению с одиночным компьютером, разбивая задачу на ветви, которые параллельно выполняются и обмениваются данными по сети.

Кластер NeClus представляет собой параллельную вычислительную систему MIMD-архитектуры с распределенной памятью. Кластер состоит из 93-х вычислительных узлов: Node1 - Node93; одного узла управления - Front Node; системы коммутации в составе двух гигабитных Ethernet коммутаторов - HP Procurve. В качестве узла управления и вычислительных узлов используются стоечные 1U системные блоки фирмы NEC - NEC Express 5800 120RE-1 PCI-E. Технические характеристики этих блоков: процессорная часть - 2 процессора Intel Xeon 3.2GHz, 1 Mb L2 кеш, частота шины 800; оперативная память - 2 модуля оперативной памяти по 1 GB стандарта DDR 333; сеть – 1Gigabit Ethernet.

Программное обеспечение кластера: операционная система - ScientificLinux 5.4; пакет управления очередью заявок - Torque PBS; пакет управления выделением ресурсов - Maui; средства параллельного программирования MPI: Openmpi-1.2.4; Mpich-ch_p4-gcc-1.2.7; Lam-7.1.4. При разработке ускоренного алгоритма трассировки лучей были использованы блокирующие функции обмена информацией типа «Точка – Точка» (функции парного обмена). Для передачи данных использовалась функция MPI_Send (...). Для принятия данных использовалась функция MPI_Recv (...). Для синхронизации процессов была использована функция MPI_Barrier (...), которая блокирует вызовы процессов, пока все процессы группы не вызовут ее. В каждом процессе управления возвращается только тогда, когда все процессы в группе вызовут процедуру. Для измерения времени выполнения трассировки использовалась функция MPI_Wtime ().

Адаптация алгоритма для реализации на кластере

Алгоритм построен таким образом, что процесс с рангом 0 (master thread, MT) распределяет сроки пикселей в равном количестве между другими исполнительными процессами (slave thread, ST) и переходит в режим ожидания приема результатов от каждого исполнительного процесса Мастер поток определяет количество процессов в коммуникаторе MPI. Затем выполняется анализ количества потоков и формирование пакета, который содержит данные о задачах для каждого процесса. После этого выполняется рассылка пакетов всем вычислительным процессам в коммуникаторе MPI. Далее происходит инициализация начальных данных в каждом процессе, и начинается формирование своей части сцены. Отправка данных от вычислительных процессов выполняется по одной строке сцены. После выполнения всего алгоритма трассировки лучей проходит вычисление времени выполнения алгоритма, конец вычислений фиксируется в момент приема данных от последнего завершенного процесса.

Реализация модифицированного алгоритма трассировки лучей

Ускоренный алгоритм трассировки лучей для многопроцессорной системы был реализован на двух языках программирования - на Microsoft C # и C++. Язык Microsoft C # использовалась для разработки и отладки ускоренного алгоритма трассировки лучей на системе в среде Microsoft Compute Cluster Pack 2008. Язык C++ использовалась для конечной реализации алгоритма трассировки лучей для многопроцессорной системы, которая работает под управлением Scientific Linux 4.6. Обе программы были реализованы с помощью библиотек, которые представляют собой реализацию стандарта Message Passing Interface (MPI) - MPICH для C++ реализации и MPI.NET для программы, написанной на Microsoft C #.

Информационный блок служит для передачи ряда параметров от мастер - процесса другим процессам. Он реализован в виде следующей структуры:

```
struct SendingBlock
 {
  unsigned int _begin;// координата X, с которой начинается процесс трассировки;
  unsigned int _end;  // координата X, до которой выполняется процесс трассировки;
```

```
    char _AA;          // качество антиэлайсинга
    char _scene;       // сцена для тестирования
    char _step;        // шаг интерполяции
    char _koef;        // максимальное значения коэф. расхождения в цвете
    unsigned int _xy;  // результирующий размер изображения сцены в пикселях
};
```

Реализация мастер-части алгоритма. Основными задачами мастер - процесса являются: отправка информационного блока SendingBlock в начале выполнения программы слейв - процессам; принятие и обработка данных от процессов. Отсылка информационного блока SendingBlock выполняется через операцию MPI_Send. Это обусловлено тем, что данные в информационном блоке отличаются для каждого процесса, т.к. каждый процесс обрабатывает свою часть изображения. Формирование и отсылка информационных блоков приведены в листинге ниже.

```
    <чтение с клавиатуры параметров шага интерполяции, максимального
коэффициента различия в цвете >
    for (char i = 1; i < iTotalTasks; i++)
    {
        masterBlock->_begin = WIDTH / (tasks-1) * (i-1);
        masterBlock->_end = masterBlock->_begin + WIDTH /
    (tasks-1);
        MPI_Send(masterBlock,sizeof(*masterBlock),MPI_CHAR,
    i, 0, MPI_COMM_WORLD);
    }
```

Прием данных от слейв–процессов обеспечивается связкой двух функций MPI_Recv(…).

```
    MPI_Recv(&row_id,sizeof(row_id),MPI_CHAR, MPI_ANY_SOURCE, TAG_INFO,
MPI_COMM_WORLD, status);
    MPI_Recv(gathered_data, sizeof(*gathered_data) * (WIDTH + 2), MPI_CHAR, status-
>MPI_SOURCE, TAG_DATA, MPI_COMM_WORLD, status);
```

Первая функция MPI_Recv (...) принимает переменную row_id, которая представляет собой номер трассируемой сроки в изображении. Вторая функция MPI_Recv (...) принимает массив рассчитанных цветов пикселей.

Основной листинг мастер-части приведен ниже.

```
    double starttime = MPI_Wtime();
        if (rank == 0) // MASTER Part
        {
        cout<<"MASTER PART"<<endl;
        Colour* gathered_data = new Colour[WIDTH + 2];
        int row_id = -1;
        unsigned int* status_array = new unsigned int[tasks];
        for (char i = 0; i < tasks; i++)  status_array[i] = 0;
        for (unsigned int cnt_rcv = 0; cnt_rcv < (WIDTH+(tasks-1)); cnt_rcv++)
        {
        MPI_Recv(&row_id,sizeof(row_id),MPI_CHAR, MPI_ANY_SOURCE, TAG_INFO,
MPI_COMM_WORLD, status);
        MPI_Recv(gathered_data, sizeof(*gathered_data) * (WIDTH + 2), MPI_CHAR,
status->MPI_SOURCE, TAG_DATA, MPI_COMM_WORLD, status);
        status_array[status->MPI_SOURCE]++;
        if (status_array[status->MPI_SOURCE] == (WIDTH / (tasks - 1) + 1))
        {  double tend = MPI_Wtime ();
        cout<<"Thread "<<status->MPI_SOURCE<<" did his work, and now waiting other
threads at Barrier point. It worked "<<(tend - starttime)<<"s"<<endl;   }
```

Разработка слейв - части алгоритма. Основными задачами слейв - процесса являются: принятие информационного блока SendingBlock в начале выполнения программы от мастер - процесса; передача отработанных данных в мастер процесса. Принятие информационного блока SendingB lock выполняется операцией MPI_Recv. Далее идет инициализация сцены для трассировки в зависимости от того, какой номер сцены был принят от мастер-процесса.

```
MPI_Recv(rcvBlock,sizeof(*rcvBlock),MPI_CHAR,0,0,MPI_COMM_WORLD, status);
<ініціалізація сцени>
```

Далее с учетом принятого шага интерполяции идет настройка класса RayTracer и передача управления одной из функций трассировки - MPI_RayTraceScene (...) и MPI_RayTraceSceneIntr (...).

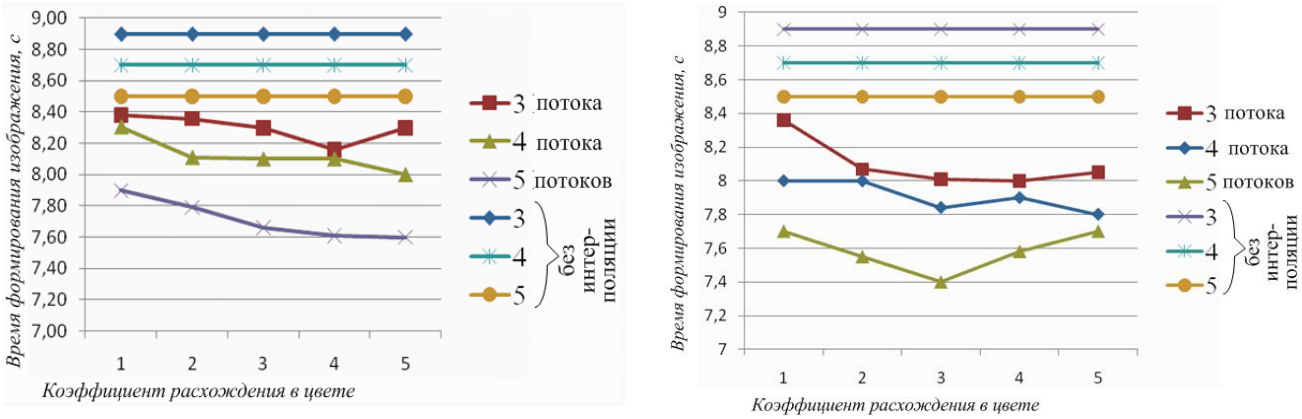
```
if (rcvBlock->_step == 1)
{//no interpolation
      RayTracer* rayTracer = new RayTracer(a_a,
                  true, true, true, true, true);
rayTracer->MPI_RayTraceScene (_xy, _xy, scene,
rcvBlock->_begin, rcvBlock->_end);
}    else   {
RayTracer* rayTracer = new RayTracer(a_a, true, true, true, true, true);
rayTracer->MPI_RayTraceSceneIntr (_xy, _xy, scene,
rcvBlock->_begin, rcvBlock->_end, rcvBlock->_step,
rcvBlock->_koef); }
```

После формирования одной строки изображения в функциях MPI_RayTraceScene и MPI_RayTraceSceneIntr происходит отправка данных в мастер - процесс.

```
MPI_Send (&y,sizeof(y),MPI_CHAR,0,TAG_INFO,MPI_COMM_WORLD);
MPI_Send (sending_colors,sizeof(*sending_colors)*(array_len + 2),
MPI_CHAR,0, TAG_DATA, MPI_COMM_WORLD);
```

Результаты тестирования модифицированного алгоритма трассировки лучей

Тестирование ускоренного алгоритма трассировки лучей для многопроцессорной системы проводилось в 2 режимах: для изображения размером 600 на 600 пикселей и для изображения размером 1800 на 1200 пикселей. Такой выбор размеров сцен обусловлено тем, что в настоящее время являются очень популярным формат FullHD (Full High Definition), для которого стандартный размер изображения 1820 на 1200 пикселей. Тестирование проводилось с использованием 4-кратного антиэлайсинга, для трассировки была выбрана сцена из [5]. При тестировании на одном процессе формирование изображения 600 на 600 пикселей происходило за 47,7 секунды. На рис.1 приведены графики сравнения быстродействия алгоритма трассировки лучей на 3-х, 4-х и 5-ти потоках с использованием линейной интерполяции с длиной пиксельного сегмента 2 и 3 пикселя для изображения размером 600 на 600 пикселей. Как видно из рисунка, оптимальным является использование 5-ти потоков и коэффициента максимального различия в цвете, равного 3. Такая комбинация дала прирост производительности около 11% по сравнению с 5-ю потоками без использования межпиксельной интерполяции.

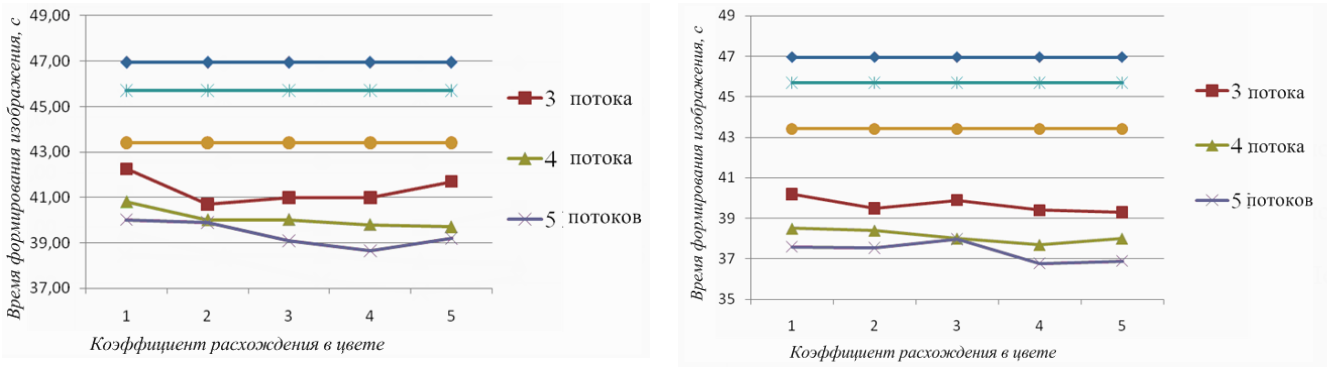


а) б)

Рисунок 1 – Сравнение скорости отображения сцены с использованием 3-х, 4-х и 5-ти потоков без интерполяции и с интерполяцией с шагом 2 (а) и 3 (б) пикселя (размер изображения 600×600)

Как видно из рисунка, оптимальным является использование 5-ти потоков и коэффициента максимального различия в цвете, равного 3. Такая комбинация дала прирост производительности около 11% по сравнению с 5-ю процессами без использования межпиксельной интерполяции. По сравнению с последовательным выполнением ускорение отображения с использованием интерполяции равно около 82%.

На рис.2 изображены графики сравнения быстродействия ускоренного алгоритма трассировки лучей на 3-х, 4-х и 5-ти потоках с использованием линейной интерполяции с длиной пиксельного сегмента 2 и 3 пикселя для изображения размером 1800 на 1200 пикселей (FullHD изображения).



а) б)

Рисунок 2 – сравнение скорости отображения сцены с использованием 3-х, 4-х и 5-ти потоков, шаг интерполяции 2 (а) и 3 (б) пикселя (FullHD изображение размером 1800×1200)

Как видно из рисунка 2, оптимальным для FullHD изображения является использование 5-ти потоков и коэффициента максимального различия в цвете, равного 4. Такая комбинация дала прирост около 16% по сравнению с 5-ю потоками без использования алгоритма интерполяции. По сравнению с выполнением на одном потоке (время отображения 98,7 с) ускорение отображения с использованием интерполяции составило 62%.

Выводы

В результате анализа алгоритма трассировки лучей и особенностей кластера разработана параллельная версия алгоритм трассировки лучей для MIMD системы. Реализован и протестирован кросс-платформенный алгоритм трассировки лучей для MIMD систем на языке высокого уровня C + + с использованием библиотеки MPI. Для изображения размером 600 на 600 пикселей максимальное ускорение (82%) получено при использовании 5-ти потоков, шага интерполяции в 3 пикселя и коэффициента различия в цвете, равного 3.

Для изображения размером 1800 на 1200 пикселей было получено ускорение на 62% (по сравнению с последовательной обработкой) для реализации на 5-ти потоках, шага интерполяции 3 пикселя и максимального коэффициента различия в цвете, равного 4.

Список использованной литературы

1. Foley J., Dam A. Computer Graphics. Principles and practice / J. Foley, A. Dam. - 2nd ed. In C. - AWPC, 1997. – 1175 pp.
2. Malcheva R. The improving of the Ray-tracing algorithm / R. Malcheva // Proceedings of ECCPM 2002. - Portoroz, 2002. – P. 537-538.
3. Мальчева Р.В. Применение строчной межпиксельной интерполяции для ускорения трассирования лучей / Р.В. Мальчева, Юнис М. // Материалы двенадцатого

международного научно-практического семинара «Практика и перспективы развития партнерства в сфере высшей школы». №12., Кн.2. – Донецк: ДонНТУ, 2011. – С. 72-74.
4. Мальчева Р.В. Применение блочной межпиксельной интерполяции для ускорения трассирования лучей / Р.В. Мальчева, С.А. Ковалев, Юнис М. // Материалы XVIII МНТК «Машиностроение и техносфера XXI века». Т2. – Донецк, 2011. – С. 189-191.
5. Мальчева Р.В. Разработка ускоренного алгоритма трассировки лучей / Р.В. Мальчева, А.А. Сереженко, Юнис М. // Інформатика та комп'ютерні технології. Матеріали VI міжнародної науково-технічної конференції студентів, аспірантів та молодих науковців. Т.1. – Донецьк: ДонНТУ, 2010. - С. 125–134.

Надійшла до редакції:
13.04.2013

Рецензент:
д-р техн. наук, проф. Скобцов Ю.О.

Р.В. Мальчева, М. Юніс
ДВНЗ «Донецький національний технічний університет»

Реалізація модифікованого алгоритму трасування промінів на кластері NeClus. *Виконаний аналіз апаратних і програмних засобів кластеру NeClus ДонНТУ. Розглянута реалізація модифікованого алгоритму з порядною інтерполяцією на кластері NeClus для зображень розміром 600 на 600 та 1800 на 1200 пікселей. Проведений аналіз результатів формування зображень й наведені кількісні оцінки ефективності застосування паралельної реалізації на кластері. Також дана оцінка впливу довжини сегменту пікселей й коефіцієнту різниці у кольорі на тривалість формування зображення.*
Ключові слова: *трасування промінів, кластер, інтерполяція, довжина сегменту, швидкість.*

R.V. Malcheva, M. Yunis
Donetsk National Technical University

The Realization of a Modified Ray-tracing Algorithm on the Cluster NeClus. *After the cluster NeClus (DonNTU) hardware and software analysis an adaptation of ray-tracing algorithm for implementation on a cluster is made. Algorithm is the following: the process with rank 0 (master thread, MT) sends an equal amount of pixel segments to every executive processes (slave thread, ST) and expect to begin receiving the array of pixels from each of the enforcement process. Master thread determines the number of processes in the communicator MPI and sends data to slaves. Then in every slave process the data initializations are started, slaves begin processing of corresponding part of the scene. Measuring of the algorithm execution time is evaluated after the forming of all pixels parameter. The end of the calculations is not fixed until master isn't receiving the data from the last completed process. As a result of analysis of the ray tracing algorithm and the characteristics of cluster a parallel version of the ray tracing algorithm for MIMD system is developed. A cross-platform ray tracing algorithm for MIMD systems on high-level language C++ using the library MPI is implemented and tested.*

To improve the system affectivity an algorithms for horizontal interpixel interpolation is proposed. A cross-platform modified ray tracing algorithm with a horizontal interpolation is implemented and tested for two scenes. For the image of 600 per 600 pixels the maximum acceleration (82%) was obtained for an implementation using 5-streams, 3-pixels segment for interpolation and the coefficient of color differences equal to 3. For the image of 1800 per 1200 pixels the maximum acceleration 62% (compared with the serial processing) was obtained for an implementation using 5-streams, 3-pixels segment for interpolation and the coefficient of color differences equal to 4.
Keywords*: ray-tracing, cluster, interpolation, length of segment, time.*