ПАКЕТ ПРОГРАММ ЛОГОС. МЕТОДЫ ФОРМИРОВАНИЯ ИЗОБРАЖЕНИЯ В ПАРАЛЛЕЛЬНОМ РЕЖИМЕ

И. В. Логинов, А. Л. Потехин, В. А. Никитин

ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Для развития наукоемких отраслей промышленности страны проводятся ОКР по теме «Разработка технологий проектирования и имитационного моделирования для суперЭВМ на основе базового программного обеспечения». Проект получил название «Пакет программ ЛОГОС».

С целью обеспечения проекта средствами графического анализа в ИТМФ РФЯЦ-ВНИИЭФ развивается параллельная система постобработки ScientificView [1], сегодня как отдельное приложение, а в ближайшей перспективе как основной модуль постобработки в составе пакета программ ЛОГОС.

Причиной развития системы ScientificView послужило то, что на момент начала ее создания ни одна программа визуализации (ParaView, LS-PrePost, Star-CD и т. д.) не могла полностью удовлетворить все потребности пользователей математического отделения, а реализация в таких программах новых возможностей сопряжена с различными трудностями.

Изначально система ScientificView была построена по схеме клиент-сервер. Серверная часть работает в параллельном режиме под операционными системами Windows или Linux на серверах с распределенной памятью. Сервер ScientificView обеспечивает считывание, предобработку и фильтрацию данных. Клиентская часть ScientificView – приложение, работающее под операционной системой Windows, содержащее пользовательский интерфейс и подсистему формирования изображения.

Распараллеливание процедур считывания и фильтрации данных позволило снизить время обработки средних по размеру задач до комфортных величин – в случае задач с числом ячеек регулярной сетки около 500 млн и использовании 64 процессоров для запуска серверной части время работы данных процедур составляет приблизительно от 40 до 80 с. Такое распараллеливание позволило быстро провести первичную реализацию системы, но привело к дисбалансу нагрузки – клиентская часть, при запуске на одном процессоре, должна обеспечить приемлемую скорость визуализации данных, полученных на большом числе процессоров сервера. Кроме того, ПЭВМ пользователя должна быть достаточно мощной, например, для обработки задач с 1 млрд ячеек может потребоваться более 8 ГБ оперативной памяти.

Поэтому в рамках системы ScientificView была начата работа по распараллеливанию процедур формирования изображения (реализация параллельного рендеринга). В докладе отражены основные задачи, решенные разработчиками системы при реализации режима параллельного рендеринга. Приведена зависимость времени работы разных алгоритмов от числа используемых процессоров при включении данного режима, описаны перспективы развития.

Преимущества использования параллельного рендеринга

Параллельный рендеринг позволяет решить сразу несколько актуальных задач, которые возникают при обработке расчетных данных большого объема в многопроцессорном режиме:

- снижение нагрузки на ПЭВМ пользователя, в особенности на оперативную память и видеокарту;
- снятие ограничения на объем данных – потенциально объем обрабатываемых данных ограничивается только мощностью параллельной ЭВМ;
- уменьшение времени загрузки и обработки данных, так как не нужно передавать графическую информацию на ПЭВМ;
- снижение нагрузки на ЛВС, так как нет необходимости передавать большие объемы данных.

Инициализация контекста OpenGL под ОС UNIX/LINUX

Для первичной реализации удаленного рендеринга (в системе ScientificView) необходимо было выбрать средство для создания контекста OpenGL под ОС UNIX/LINUX. Существует несколько вариантов инициализации контекста OpenGL под ОС UNIX/LINUX, в основном реализация сводится к использованию одного из инструментальных средств – GLX (расширение X Windows) или библиотеки Mesa (программная реализация OpenGL).

Как и во всех других операционных системах, в Unix X Windows не существует прямой поддержки OpenGL. В X Windows добавлено расширение GLX, для того чтобы можно было использовать OpenGL. GLX полностью отвечает за взаимодействие между OpenGL и X Windows, например, задает сетевой протокол для OpenGL команд, отвечает за создание и обработку OpenGL контекстов, т. е. обеспечивает корректную работу OpenGL в оконной системе X Windows. GLX имеет довольно богатый набор воз-

можностей, а также реализует специфические функции, характерные только для X Windows.

Mesa – это кросс-платформенная open-source библиотека, которая представляет собой программную реализацию библиотек OpenGL. В основном, Mesa применяется в приложениях, которые работают на различных узлах вычислительных кластеров, в т. ч. на узлах без мощных графических ускорителей.

В качестве средства для использования OpenGL под Unix разработчиками системы ScientificView были выбраны библиотеки Mesa. Выбор обусловлен несколькими причинами:

- нет ограничений на тип используемых узлов удаленных ЭВМ (с ускорителями или без них);
- нет зависимости от библиотек Unix X Windows;
- более простая и быстрая реализация кода для инициализации и использования контекста OpenGL.

Блокировка передачи топологии отображаемых сеток на клиент

Для того чтобы полностью переложить функции отрисовки с клиента на сервер (в режиме удаленного рендеринга), на клиентской стороне системы ScientificView необходимо было заблокировать вызов всех функций, так или иначе связанных с отображением данных, и заблокировать прием-передачу этих данных с клиента на сервер. Для блокировки передачи данных о топологии отображаемых объектов необходимо было учесть несколько основных моментов:

- запрет выделения памяти, необходимой для хранения «графических» данных на клиенте;
- запрет на клиенте вызовов функций, связанных с отображением данных;
- запрет отправки топологических данных с сервера на клиент.

В процессе написания кода для блокировки функций отображения на клиенте необходимо было корректно блокировать те или иные блоки программного кода в зависимости от трех возможных режимов работы системы ScientificView:

- скалярный режим на ПК;
- удаленный режим без параллельного рендеринга;
- удаленный режим с параллельным рендерингом.

Для локализации таких программных блоков был реализован соответствующий функционал, определяющий критерий выполнения того или иного участка кода в зависимости от приведенных режимов работы.

Стандартизация передачи параметров с клиента на сервер

Все настройки ScientificView, которые могут изменяться пользователем в процессе сеанса работы с системой, хранятся в отдельных наборах параметров. Для стандартизации процесса передачи различных настроек с клиента на сервер все наборы параметров были приведены к единому виду, что позволило упростить формирование массива данных для обмена между клиентом и сервером. Далее был реализован функционал, позволяющий единым образом передавать или получать сформированные массивы.

Обеспечение работы сервисных диалогов, связанных с отображением данных

Для управления различными настройками области отображения системы ScientificView необходимо было реализовать передачу параметров из диалогов клиентского приложения на сервер.

Основные настройки подсистемы отображения управляются при помощи диалогов, вызываемых из меню Параметры главного окна клиентского приложения системы ScientificView. Данное меню содержит пункты для вызова следующих диалогов:

- ***Настройка освещения*** – настройка параметров освещения сцены.
- ***Вспомогательная информация*** – задание критериев вывода подсказок, сообщений об ошибках, статуса выполняемых операций и др.
- ***Оптимизация*** – состоит из двух закладок для управления параметрами оптимизации процесса отрисовки геометрических объектов.
- ***Общие настройки***, состоит из закладок:
  - ***Отображение*** – задание цвета фона, цвета базовых примитивов области отображения – линии, точки, поверхности и др., задание размера шрифта для подписей шкалы, задание критериев отображения шкалы, координатных осей и др.
  - ***Сценарий*** – параметры обработки сценариев.
  - ***Информация по задаче*** – задание критериев отображения информации об открытом на данный момент разрезе.
  - ***Стартовые параметры*** – различные умолчания, задаваемые пользователем, – режим отображения, включение освещения и др.

Практически все приведенные параметры передаются путем применения механизма, описанного в разделе, посвященном стандартизации передачи параметров с клиента на сервер.

Манипуляции с отображаемой сценой

Изначально все манипуляции с изображением происходили на клиентской части программы. При переносе отображения сцены на серверную часть появилась проблема удаленной манипуляции с изображением.

Для решения проблемы был расширен функционал серверной части, по аналогии с клиентом, отвечающий за отображение сцена. Была реализована обработка новых команд от пользовательского

интерфейса, таких как: инициализация подсистемы отображения; обработка событий мыши и клавиатуры, перерисовка изображения и т. д.

Реализация механизма параллельного формирования изображений

Как уже отмечалось, ключевой особенностью режима удаленного рендеринга является выполнение процесса формирования изображения на серверной стороне программы. На первом этапе рассматривалась ситуация, когда сервер запускался на одном процессоре (т. е. без использования параллельного режима). Это значит, что при необходимости перерисовать кадр на клиенте (при изменении пользователем точки просмотра, изменении размеров экрана, выборе другой величины для отображения) единственный процессор сервера должен сформировать кадр и передать его на клиент.

После реализации алгоритма передачи кадра с серверной стороны на клиентскую, была начата работа по реализации параллельного алгоритма формирования кадра, который должен использоваться в процессе работы сервера в многопроцессорном режиме.

Ранее реализованные параллельные алгоритмы чтения и фильтрации данных используют принцип геометрической декомпозиции данных. Это значит, что каждому процессору серверной стороны при открытии файла доставался примерно одинаковый объем данных для обработки. В целях достижения приблизительно равномерной нагрузки и на модуль формирования изображения, было принято решение о том, что каждый процессор серверной части будет отображать множество данных, ранее полученных на нем в результате работы алгоритмов фильтрации.

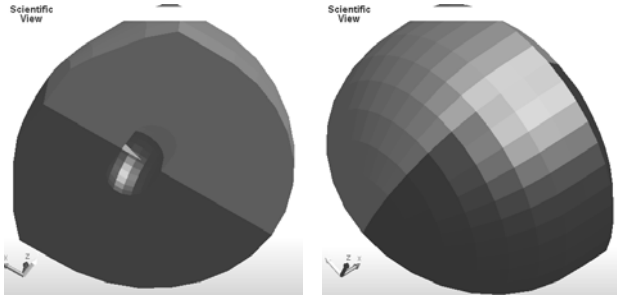

Рис. 1. Множества данных, обрабатываемые разными процессорами (разное положение объекта в сцене)

Очевидно, что при вращениях, масштабировании, смещениях и других операциях, приводящих к смене ракурса просмотра, множество пикселей экрана, занятых данными с конкретного процессора, будет меняться. Например, множество данных процессора из правого верхнего угла (рис. 1, слева) переместилось в левый верхний угол (рис. 1, справа).

Таким образом, невозможно «закрепить» за каждым процессором сервера свою часть экрана, потенциально любой процессор может при отображении своих данных закрасить любой пиксель экрана. Кроме того, возможна ситуация, когда пиксель экрана могут занимать данные с нескольких процессоров. Множество таких пикселей представлено на рис. 2 серым цветом, данные с каждого из процессоров представлены сеткой без заливки.

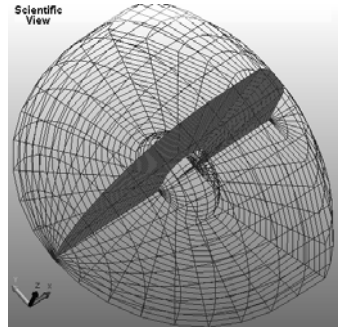

Рис. 2. Конфликтная ситуация: на некоторое множество пикселей «претендует» несколько процессоров

В таких ситуациях необходимо определить, данные с какого процессора находятся ближе к зрителю, и задать для рассматриваемого пикселя цвет именно с такого процессора (т. е. по сути провести *Z*-буферизацию).

На практике оказалось, что при обработке ряда задач средней сложности увеличение числа процессоров приводит к увеличению времени формирования кадра. Собственно формирование «своего» кадра на каждом из процессоров проводится в параллельном режиме и происходит достаточно быстро. Далее управляющий процессор последовательно проводит попиксельное сравнение с буферами от всех процессоров, следовательно, увеличение их числа приводит к увеличению времени работы. В то же время, при увеличении числа процессоров пиксельная зона ответственности каждого из них, как правило, уменьшается (рис. 3, зона «красного» процессора выделена рамкой).

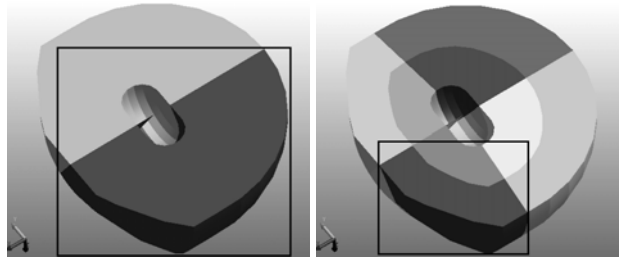

Рис. 3. Использование разного числа процессоров (слева – два, справа – шестнадцать), разница в величине пиксельных зон

Основываясь на этом, была проведена модернизация алгоритма формирования кадра в параллельном режиме. В результате модернизации удалось изменить зависимость времени формирования итогового кадра от числа процессоров с прямой на обратную (рис. 4).

Полученные данные говорят о том, что время формирования кадра вполне приемлемо для «статического» режима (когда нужно перерисовать сцену,

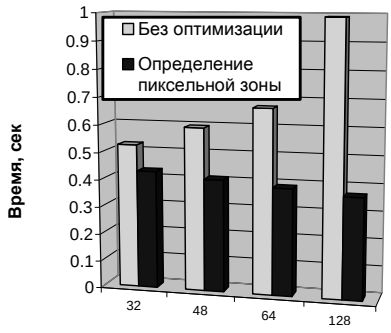

Рис. 4. Зависимость времени формирования итогового кадра от числа процессоров (первичный алгоритм и модернизированный)

например, при изменении размера экрана). Однако такое время формирования некомфортно при манипуляциях с отображаемой сценой (поворотах, смещениях и т. д.).

Здесь основную задержку уже вносит время приема-передачи данных с сервера на клиент через сеть. В данном случае, с учетом 100 мегабитного соединения и размера изображение около 3 мегабайт, получается, теоретическое минимальное время передачи около 0,25 с, что составляет более 67 % от времени формирования кадра. Очевидно, что нужно уменьшать именно время передачи, например, увеличивая пропускной канал (до 1 Гбита) или уменьшая объем передаваемых данных.

Поскольку при манипуляциях с отображаемой сценой важна высокая скорость, а качеством отображения можно несколько пренебречь, было решено использовать пиксельное огрубление. Его суть заключается в том, что вместо изображения (N, M) пикселей передается изображение, огрубленное в четыре, шестнадцать и т. д. раз, которое на клиентской стороне «натягивается» на экран размером (N, M) пикселей (рис. 5).

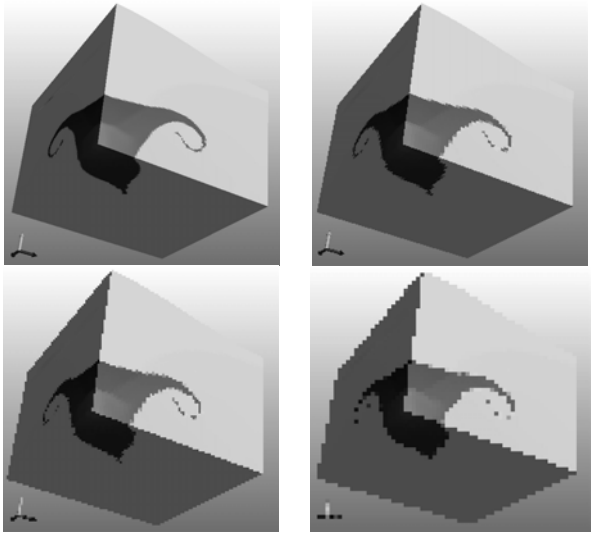

Рис. 5. Примеры использования алгоритма огрубления с разной степенью детализации (слева направо – исходное изображение, огрубление в 4 раза, 16 раз и в 64 раза)

Использование при вращении отображаемого объекта 16-кратного огрубления снижает время формирования кадра до 0,15 с, что соответствует вполне комфортному режиму с отображением 5–7 кадров в секунду.

Оценка характеристик работы режима параллельного рендеринга

Для проверки работоспособности ScientificView в режиме параллельного рендеринга было проведено тестирование наиболее часто используемых возможностей системы при запуске серверной части на визуализационном кластере. В качестве теста была выбрана задача, посчитанная по методике ЛЭГАК-3D [3]. Регулярная сетка задачи состоит из 2,12 млрд ячеек (размерностью 2687x229x3455), содержит 6 узловых и 7 ячеечных массивов. Разрез задачи распределенный, состоит из 260 файловых фрагментов.

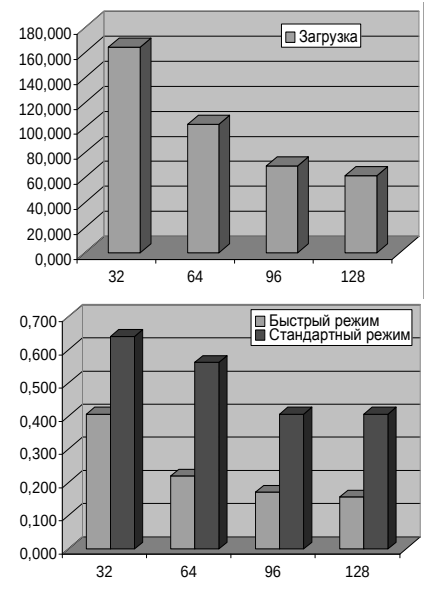

Рис. 6. Зависимость времени открытия файла на сервере (слева) и времени отображения (справа) от числа процессоров (быстрый режим – с пиксельным огрублением кадра, стандартный режим – построение полного кадра без огрубления)

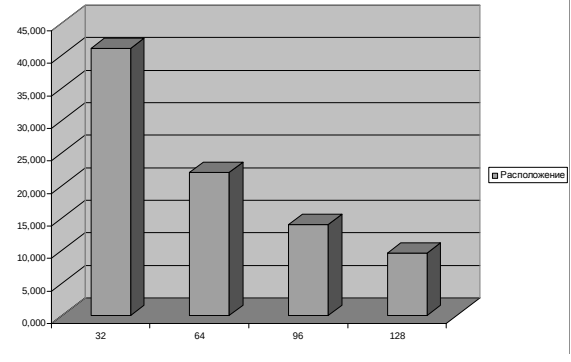

Рис. 7. Зависимость времени построения фильтра «Изоповерхность» от числа процессоров

Для сравнения режима удаленного рендеринга со стандартным режимом работы программы использовался то же тест при использовании 128 процессоров.

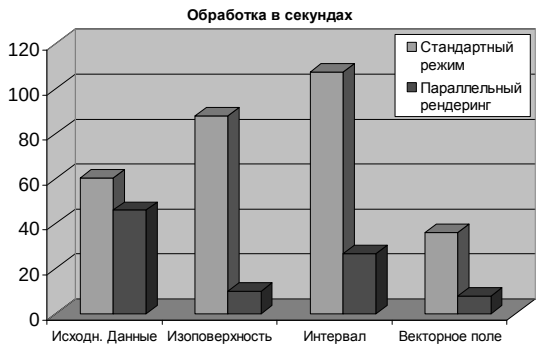

Рис. 8. Время работы алгоритмов фильтрации

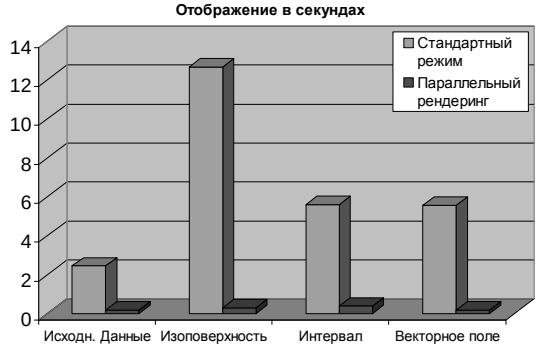

Рис. 9. Время отображения одного кадра

Рис. 10. Расход оперативной памяти на клиенте

Перспективы развития

Проведенное тестирование показало, что использование режима параллельного рендеринга крайне перспективно. По сути, он является единственным режимом, в котором ScientificView будет способен быстро обработать данные сверхбольшого объема (1–10 млрд ячеек или частиц). Однако до передачи версии ScientificView для полноценной практической эксплуатации необходимо решить ряд дополнительных задач:

• обеспечить работоспособность виджетов (интерактивных объектов, позволяющих задать параметры фильтров при помощи «мыши»). Реализовать методологию автовызова функций для обработки манипуляций с виджетами;

• реализовать кроссплатформенный метод отображения текста в контексте OpenGL;

• реализовать алгоритм формирования общего кадра в параллельном режиме, поддерживающий эффект прозрачности;

• добавить функции приема-передачи параметров для ряда диалогов;

• реализовать возможность инициализации контекста OpenGL при использовании протокола GLX для использования графических ускорителей.

Выводы

При использовании параллельного рендеринга значительно снижается нагрузка на ПЭВМ пользователя (так как клиентская часть системы не формирует изображений и не хранит топологию отображаемых объектов), также значительно уменьшается время между началом загрузки данных и выдачей изображения на экран (так как не требуется передачи данных о топологии с многопроцессорной ЭВМ на ПЭВМ). Максимальный объем обрабатываемых данных ограничивается возможностями многопроцессорной ЭВМ и практически не связан с мощностью ПЭВМ, на которой запущена клиентская часть ScientificView.

Представленные результаты тестирования показывают, что при обработке задачи 2 млрд регулярных ячеек от ПЭВМ пользователя требуется не более 50 Мбайт оперативной памяти. При этом выполнение алгоритмов фильтрации ускоряется до 5 раз по сравнению с ранее реализованным режимом отображения на ПЭВМ. Авторы предполагают, что режим параллельного рендеринга будет основным при обработке задач, состоящих из более чем 50 млн ячеек.

Литература

1. Потехин А. Л., Логинов И. В., Тарасов В. И. и др. ScientificView – параллельная система постобработки результатов, полученных при численном моделировании физических процессов // Вопросы атомной науки и техники. 2008. Вып. 4. С. 37–45.

2. Потехин А. Л. Методы быстрого формирования изображения в параллельной системе постобработки результатов научных вычислений ScientificView // Сборник трудов XX международной конференции по компьютерной графике и зрению Графикон. 2010. С. 273–279.

3. Циберев К. В., Артамонов М. В., Авдеев П. А. и др. Параллельный пакет программ ЛЭГАК-ДК для расчета задач гидрогазодинамики и прочности на неструктурированных сетках в лагранжево-эйлеровых переменных // Сборник трудов XI международного семинара «Супервычисления и математическое моделирование». Саров, 2009.