ПАРАЛЛЕЛЬНЫЙ МЕТОД ВИЗУАЛИЗАЦИИ В ДИНАМИКЕ ВЫПОЛНЕНИЯ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА НА СУПЕРКОМПЬЮТЕРЕ

К.Н. Киселёв, О.В. Корж

Введение.

Одним из важнейших этапов научного исследования является наглядное представление полученных результатов. Сделать это позволяют системы визуализации, которых на данный момент имеется огромное количество: от узкоспециализированных до универсальных. Также, в любом исследовании визуализация результатов может послужить хорошим показателем качества и, помимо того, помочь выявить непонятные на первый взгляд особенности задачи. При моделировании на суперкомпьютере для этого обычно требуется действовать в рамках следующего алгоритма:

1. пишется и отлаживается программа, занимающаяся вычислениями и выдающая на выходе некоторый набор данных
2. запускается программа на реальных данных, описывающих некоторую исследуемую модель
3. после завершения счёта собираются данные и загружаются в систему визуализации
4. на выходе получается изображение или видео

В рамках такой последовательности действий визуализация выполняется лишь на последнем этапе. Хорошими примерами являются системы ParaView*[4]*, VisIt*[5]*, Equalizer*[6]*. Кроме того возможно написать визуализацию самостоятельно, используя VTK(Visualization ToolKit)*[2]* или просто OpenGL*[13]*. Однако существуют задачи, для которых такой подход становится неудобным или вообще невозможным. Узким горлышком здесь является то, что система визуализации опирается на полностью готовый набор данных, и исследователь может увидеть своими глазами результат только после завершения основных вычислений. Этот вариант неудобен, например, в следующих случаях:

- объём результатов вычислений очень велик — порядка сотен гигабайт или нескольких терабайт (нет возможности загрузить данные)
- не существует полного набора исходных данных (задачи, связанные с обработкой показаний каких-либо датчиков в реальном времени)
- интерактивное изменение параметров модели в процессе счёта

С помощью существующих средств визуализации алгоритм можно модифицировать следующим образом:

1. запуск вычислительной задачи в обычном режиме, когда результаты выводятся в файл
2. запуск визуализации, осуществляющей периодическое считывание из файла, либо из разных файлов

В данном случае имеется достаточно весомый недостаток — обмен данными через узлы ввода-вывода. Гораздо выгоднее использовать иной подход. Для осуществления визуализации в задачах с описанными выше свойствами требуется визуализировать данные в режиме реального времени (параллельно с основными вычислениями). В таком случае описанная последовательность действий изменится следующим образом:

1. написание и отладка вычислительного модуля;
2. написание и отладка визуализационного модуля;
3. запуск программы в рамках одной задачи вычислительного кластера;
4. наблюдение за результатами визуализации в режиме реального времени.

Основной особенностью описываемого подхода является параллельная работа процессов, отвечающих за визуализацию, и процессов, вычисляющих все необходимые для отображения данные в рамках одной вычислительной задачи. Очевидным является то, что область применения подобного подхода, где это будет действительно эффективным решением, обладает особенными свойствами, на которые можно опираться для достижения приемлемых результатов. Если попытаться формализовать специфику задачи, требующей визуализации в режиме реального времени, то можно выделить следующее:

1. Вычисления, описанные в задаче, опираются на источник данных, генерирующий последние постоянно и непрерывно в течение продолжительного времени. Примером может служить набор датчиков, измеряющих какие-либо физические показатели (температура, давление, скорость ветра и т. д.).
2. Задаче для получения приемлемых для оценки результатов требуется довольно большое время. В таком случае может быть необходим наглядный и эффективный контроль процесса вычислений во время выполнения на реальных исходных данных, чтобы идентифицировать ошибки на раннем этапе.
3. Задаче для работы требуется коммуникация с пользователем (например, изменение

параметров во время счета и т. п.)

Целью данной работы является исследование и реализация визуализации данных, предоставляемых основной вычислительной задачей в режиме реального времени, а также оценка масштабируемости параллельной компоновки изображений и качества результатов при такого рода подходе.

Общая схема работы визуализатора

Опираясь на выделенные во введении специфические свойства задач, требующих визуализации в режиме реального времени, можно определить критерии, которым должно соответствовать предлагаемое решение. Во-первых, так как вычисления и рендеринг производятся в рамках одной задачи (здесь говорится о задаче на кластера), требуется строгое отделение коммуникации между процессами, задействованными в визуализации, и процессами, выполняющими основные вычисления для получения данных. Во-вторых, необходимо решение проблемы синхронизации отображения и вычислений для предотвращения некорректных результатов работы визуализатора. Помимо этого, важной задачей является сокращение объёма пересылок между процессами визуализации и процессами, производящими расчёты. Последним критерием является доставка результатов визуализации конечному пользователю в качестве, приемлемом для исследования.

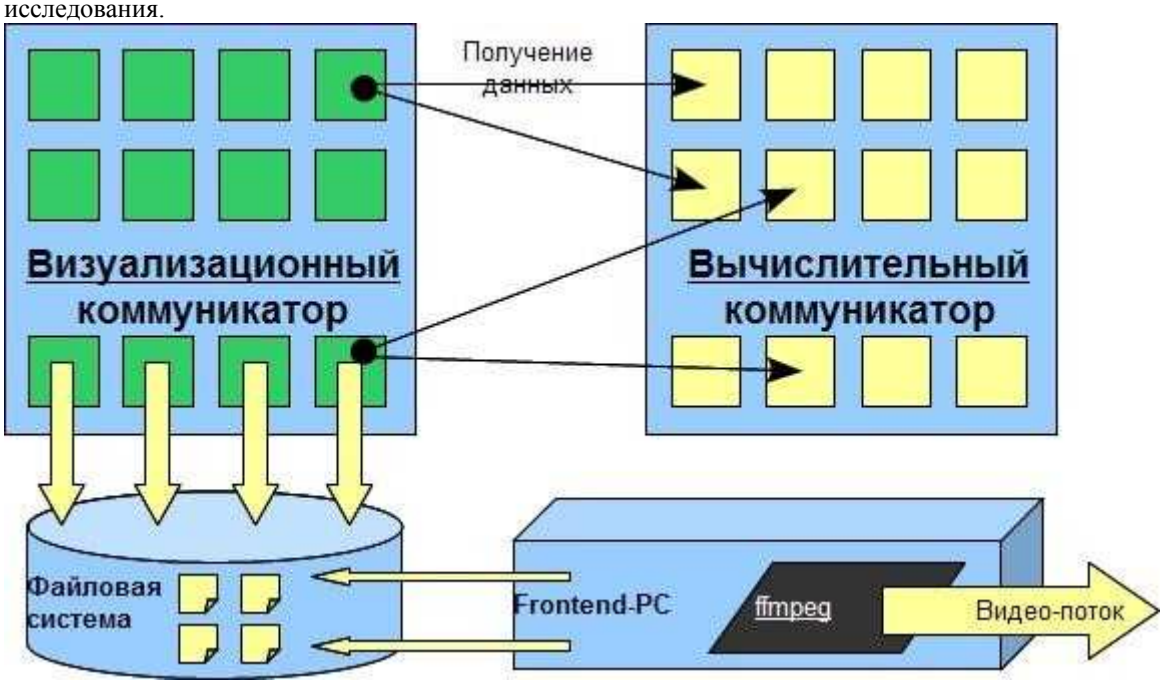

Рис 1. Общая схема работы визуализатора.

В данной работе для организации параллельной компоновки изображений используется библиотека IceT*[1]*. Она предлагает для использования собственные обёртки функций MPI, что в результате приводит к устранению проблем связанных с раздельной коммуникацией процессов, отвечающих за рендеринг и вычисления. Все дополнительные пересылки, связанные с визуализацией также проводятся в рамках коммуникатора IceT.

Для синхронизации процесса отображения и вычислений в данном решении предлагается использовать механизм делегирования. Все исходные данные для выполнения визуализации хранятся на «вычислительных» узлах. Обновление сопровождается отправкой соответствующего сообщения главному узлу визуализации, который в результате сам обращается к необходимым участкам данных удалённо. В данном случае используется библиотека MPI*[9,10]*, но не сложно реализовать и другие подходы (например, с помощью односторонних коммуникаций).

После компоновки результирующие изображения отправляются через узлы ввода-вывода в файловую систему кластера, где их обработкой занимается уже фронтенд. На этом компьютере производится кодирование входных изображений в видео-поток, который в дальнейшем можно просматривать любым удобным способом. Существует и другое возможное решение для создания видео-потока — кодирующая программа запускается не на отдельном компьютере, а на множестве узлов кластера. В этом случае пересылка изображений будет производиться не через узлы ввода-вывода, а через внутреннюю коммуникационную сеть между вычислительными узлами. Это может быть удобно и намного эффективнее, если имеется возможность отображать пользователю несколько видео-потоков (например, на разных мониторах, объединённых в «мониторную стену»).

Организация доступа к данным основной вычислительной задачи

В результате того, что вычисления и визуализация разделены по процессам, возникает проблема

доступа к данным, на основе которых предполагается строить изображение. Здесь возможно применение различных подходов: односторонние коммуникации (shmem *[11]*, MPI-2 *[9,10]*) и взаимодействие через сообщения (MPI *[9,10]*).

Предлагается разделить процессы на визуализационные и вычислительные. Задание количества возлагается на пользователя разрабатываемой системы, в виду различия преследуемых целей. В данной работе описывается реализация межпроцессного взаимодействия на основе MPI. Для предотвращения возможных коллизий во внутренних коммуникациях процессов визуализации и вычислительных процессов создаётся два различных коммуникатора. Пользователю разрешается использовать коммуникатор *MPI_COMM_WORLD* только для пересылки данных с узлов вычислений на узлы визуализации. Пользователю предлагается указать буферы данных, которые система затем при соответствующем вызове синхронизации пересылает на узлы визуализации. Все пересылки неблокирующие, за счёт чего достигается оказание минимального побочного эффекта на производительность пользовательских вычислений.

Описание геометрии сцены

Геометрию сцены задает пользователь. Обращение ко всем необходимым данным осуществляется удалённо через специальный интерфейс. Основной задачей на данном этапе является декомпозиция сцены любым возможным образом. Для полигональной геометрии проще всего использовать разбиение по объектам сцены. Другими словами, каждый процесс визуализации отрисовывает свои объекты или примитивы, не задумываясь о буфере глубины. Буфер глубины учитывается на этапе компоновки, и гарантируется, что результирующее изображение будет корректным.

Дело обстоит сложнее, если требуется осуществить *volume-rendering*, когда использование буфера глубины компоновщика становится невозможным. Проблема состоит в том, что невозможно автоматически определить порядок отрисовки полупрозрачных объектов. Однако сборка изображения всё-таки осуществима, но требуется указание порядка компоновки отдельных частей сцены, распределённых по узлам. Таким образом, задание геометрии в этом случае осуществляется через отрисовку отдельных непрерывных частей объёма с включённым альфа-каналом и последующей сортировке этих частей по направлению от точки, в которой расположена камера. Затем компоновщик IceT *[1]* на основе альфа-компоненты кусков сцены и порядка процессов сможет собрать результат.

С технической точки зрения организация рендеринга показана на рис. 2.

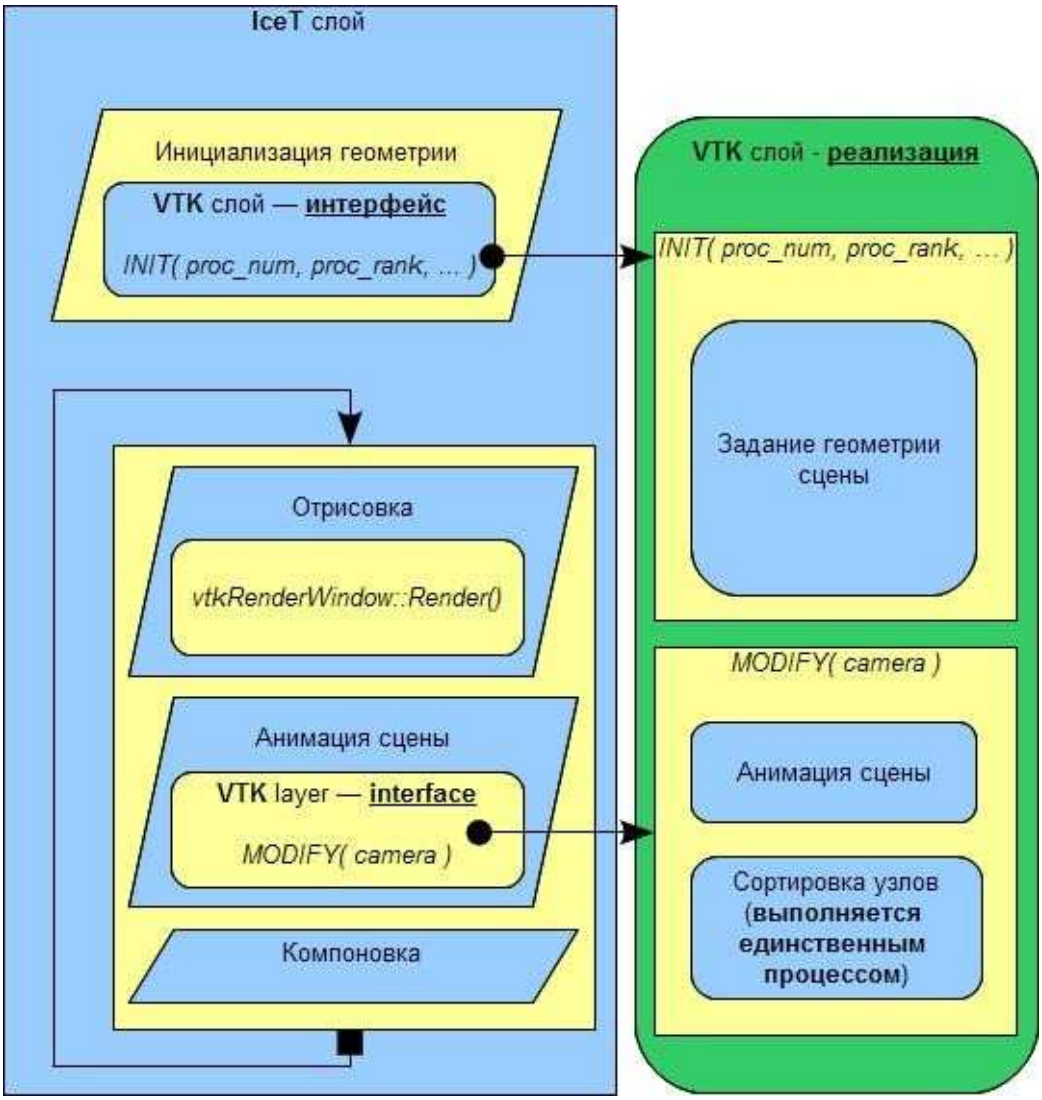

Рис. 2. Схема интерфейса задания сцены.

Параллельная компоновка результирующего изображения

В результате рендеринга отдельными процессами различных частей сцены (геометрических — в случае *volume*-рендеринга, или объектных — в случае полигональной графики) получаются изображения, которые требуется собрать в одно, опираясь в одном случае на буфер глубины, а в другом — на альфа-канал. В качестве такого компоновщика в данном решении используется библиотека IceT *[1,3]*, являющаяся основой параллельного рендеринга всем известной системы визуализации ParaView *[3,4]*.

Среди узлов, задействованных в компоновке, выделяется специальные, которые отвечают за хранение результирующего изображения. Такой узел может быть единственным, либо их может быть несколько. Использование нескольких узлов отображения выгодно, когда требуется построить картинку очень высокого разрешения, для чего необходимо намного большее количество оперативной памяти. Для параллельной компоновки применяется несколько стратегий. В данной работе используется схема рендеринга с единственным узлом отображения. В этом случае имеется возможность использовать две стратегии, которые есть по умолчанию: виртуальное дерево и бинарные обмены. Разработчики библиотеки IceT *[1,3]* утверждают, что для числа узлов меньше 8 первая стратегия более оптимальна. Бинарные же обмены эффективнее работают на большем числе процессов визуализации. Выбор стратегии происходит автоматически.

Прореживание кадров при интерактивной визуализации для синхронизации

Визуализация в реальном времени предъявляет дополнительные требования к скорости построения и компоновки изображений. Основной проблемой здесь является несбалансированность нагрузки на процессах визуализации во время рендеринга. Помимо этого, время отрисовки можно сократить, опираясь на человеческое восприятие результата.

Предлагается оптимизировать процесс рендеринга и компоновки, чтобы сократить время ожидания на основании следующих фактов. Во-первых, на общее время отрисовки влияет и рендеринг тех участков

сцены, которые с большой вероятностью не видны пользователю или видны на участке достаточно малого разрешения. Исходя из этого, можно сделать вывод о том, что различные процессы в разной степени влияют на качество результата, а следовательно, могут в разной степени отвечать за качество (разрешение и интерполяция) своей локальной части изображения. Для реализации сортировки в архитектуре системы имеется специальный класс ***processes_info_provider***, который осуществляет коллективный сбор информации с узлов визуализациии о параллелепипедах, ограничивающих геометрию части сцены.

Использование «дедлайнов» рендеринга.

Как уже было сказано, несбалансированность нагрузки процессов визуализации вызывает задержки на этапе компоновки изображения. Предлагается взять за основу описанную выше сортировку, но с поправкой на то, что от удалённости будет зависеть максимальное время, отводимое процессу на рендеринг. Если рендеринг не завершается в установленное время, то процесс будет отдавать компоновщику не настоящее изображение, а сгенерированное на основе информации, полученной за предыдущие отрисовки. Таких способов может быть несколько. Опишем некоторые из них.

Кэширование изображений.

Самым простым способом является использование хранилища полностью отрисованных изображений, куда программа после окончания рендеринга добавляет новый кадр. В таком случае после истечения отведённого процессу времени визуализатор не дожидается окончания рендеринга, а берет последнее изображение из кэша. У этого метода есть значительный недостаток, заключающийся в большом различии между правильным и предоставленным изображением. Однако при использовании сортировки по удалённости и введении нелинейной функции отводимого процессу на рендеринг времени (функция, зависящая от порядка процесса) результат получается приемлемым в большинстве случаев (особенно при визуализации на этапе отладки, когда важна общая картина происходящего, а не детали). Данный подход является обоснованным, когда требуется не «плавность» картинки, а отображение актуальных результатов. В случае, если мы визуализируем результаты работы задачи реального времени, то гораздо важнее видеть кадры реже, но с содержанием, соответствующим по времени текущим вычислениям. Самый простой пример — реализация интерактивного управления сценой с использованием компьютерной мыши. Для продолжительного движения курсором, намного выгоднее отображать картинку, соответствующую текущим координатам указателя, а не показывать всю цепочку кадров из прошлого.

Прогнозирование изображений.

Описанный выше метод достаточно прост в реализации и даёт хорошие результаты по времени обработки, но может оказаться неподходящим в случае, если от результирующего изображения требуется высокая точность и очень высокое разрешение. Предлагается вместо того, чтобы просто брать последнее изображение из хранилища, использовать все имеющиеся в кэше кадры и прогнозировать изображение на выходе. Самый простой прогноз можно сделать, если ввести для некоторых пикселей и небольших их окрестностей вектор-функцию, значение которой в следующей точке и будет прогнозироваться. В данном подходе можно опираться на то, что наиболее вероятным изменением геометрии является поворот, сдвиг и масштабирование. В результате после обучения на этих трёх видах функций могут получиться достаточно хорошие результаты. Опять же, здесь важным является то, что точность кадра на входе компоновщика зависит от порядка процесса после сортировки по удалённости. И так как функция времени, отводимого на рендеринг, не линейная, а например, квадратичная. Самый большой минус этого подхода — огромные затраты ресурсов, поэтому в разработанной системе он не реализован.

Схема реализации параллельного рендеринга с использованием кэширования.

Для реализации данной схемы требуется механизм асинхронных вызовов соответствующих функций рендеринга. Очевидным решением является использование двух потоков, в одном из которых работает основной цикл визуализации, а в другом производится непосредственная отрисовка кадра. В нашем случае при использовании VTK в дополнительном потоке вызывается блокирующая функция **vtkRenderWindow::Render()**, а в основном — **icetGLDrawFrame()**. Наглядная схема представлена на рис. 3.

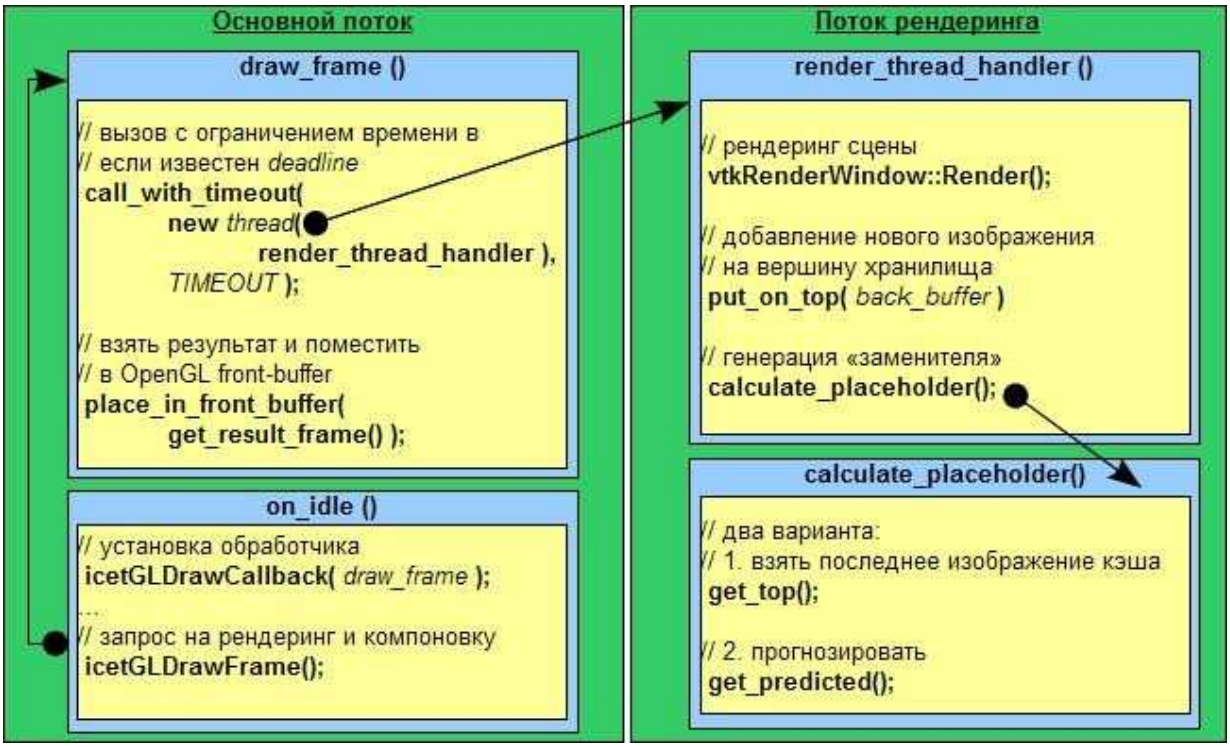

Рис. 3. Схема взаимодействия потоков при использовании рендеринга с ограничением по времени.

Соответствующая диаграмма классов подсистемы рендеринга с учётом ограничений времени выглядит следующим образом (рис. 4):

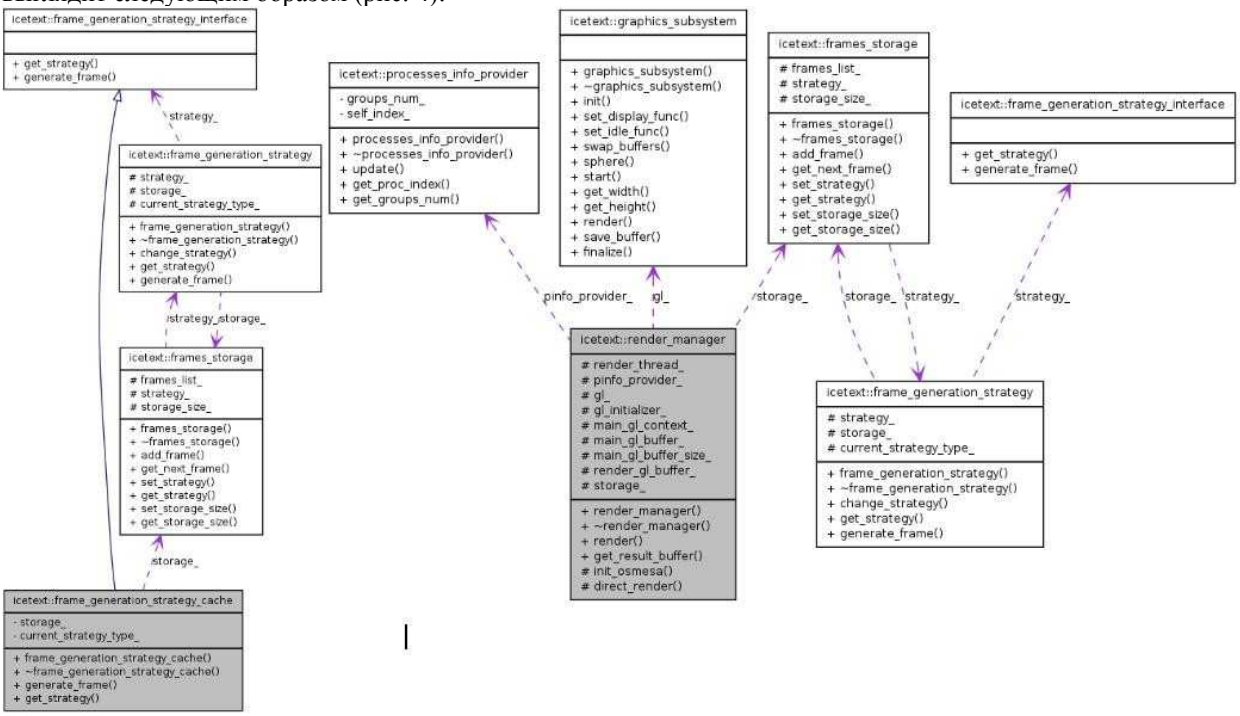

Рис. 4 Подсистема рендеринга с учётом ограничений времени.

Отображение результатов визуализации

В результате работы параллельного компоновщика на узлах, помеченных как узлы отображения, после вызова icetGLDrawFrame гарантируется сохранение соответствующих частей результирующего изображения в буфере OpenGL. Для того, чтобы возможно было видеть визуализацию в режиме реального времени множество исходных кадров требуется кодировать в видеопоток. Наиболее приемлемым в плане простоты, скорости и возможностей является ffmpeg.

Как упоминалось ранее, возможны два подхода к сборке результата в IceT:

1. параллельный компоновщик использует единственный узел отображения, на котором

располагается результирующий кадр

2. параллельный компоновщик использует разбиение экрана на части и, как следствие, несколько узлов отображения.

В первом случае можно использовать очень простую технологию. Необходимо создать в файловой системе файл типа fifo, который передать в качестве входного потока кодировщику. С узла отображения требуется всего лишь записывать очередной кадр в этот fifo-файл. В результате выходной видеопоток можно будет просматривать любым удобным способом.

Второй метод требует более сложного решения, так как каждый очередной кадр в этом случае представлен не одним, а множеством изображений. Можно было бы просто параллельно кодировать каждую часть в отдельный видео-поток. Но это неправильно, так как необходима синхронизация в случае несбалансированности отрисовки. Очевидное решение — использовать промежуточный модуль синхронизации. Таким образом, каждый узел отображения создаёт свой собственный fifo-файл, куда выводит часть очередного кадра, а модуль синхронизации, являющийся отдельным приложением, считывает входной поток. Далее возможно два варианта: объединить все части в одно изображение и использовать единственный кодировщик, либо параллельно кодировать несколько видео-потоков. В обоих случаях могут быть проблемы. В первом — это недостаток оперативной памяти при кодировании изображений огромного разрешения. Во втором — рассинхронизация выходных видео-потоков (зависит от скорости канала передачи). Таким образом, конкретный вариант кодирования результата следует выбирать в зависимости от задачи и преследуемых целей. В разработанной системе реализовано кодирование видео-потока из единственной входной очереди.

Результаты исследования масштабируемости и качества

Для анализа масштабируемости системы использовался частично синтетический тест, в котором была реализована параллельная система частиц. В результате ускорение системы на 10^5 частиц (количество треугольников порядка 10^8) с фиксированным числом вычислительных узлов и изменяемым числом узлов визуализации близко к линейному на участке от *2* до *64* процессов визуализации. В дальнейшем ускорение сильно отстаёт от линейного из-за затрат на компоновку большого числа маленький частей сцены. В эксперименте размер изображения был выбран *1024* x *1024*.

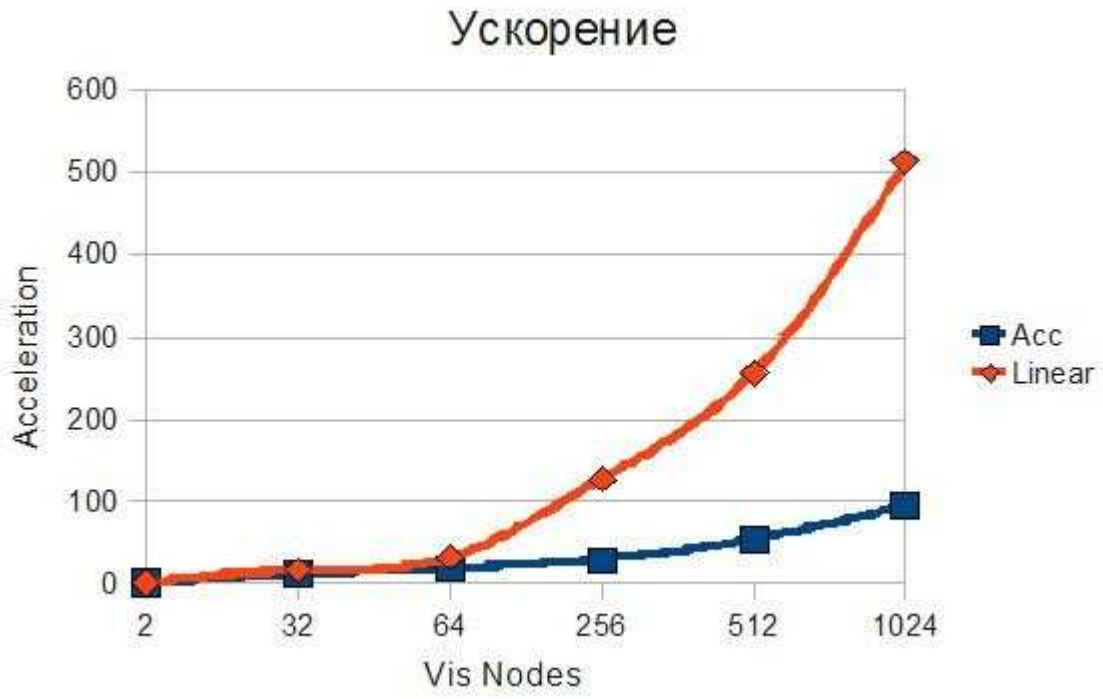

Рис. 5. График ускорения в сравнении с линейным.

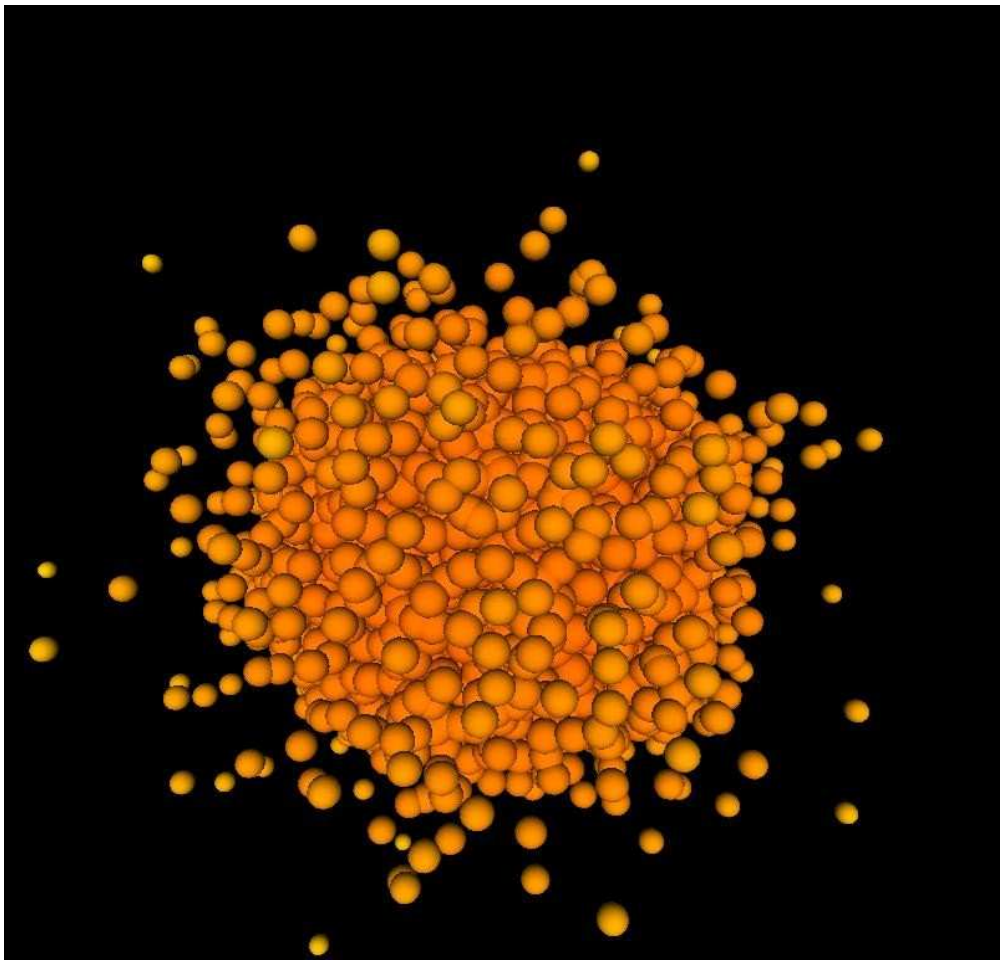

Рис. 6. Один из кадров визуализации модельной задачи (системы частиц).

Заключение.

В представленной работе выполнен анализ возможности визуализации в режиме реального времени параллельно с вычислительной задачей. Выделены основные особенности визуализации в режиме реального времени. Предложена общая схема работы визуализатора. Описана архитектура, в рамках которой разработана программа. Предлагаемая схема реализована на примере визуализации результатов модельной вычислительной задачи в режиме реального времени. Проведён анализ масштабируемости и качества реализованного визуализатора. Проведён анализ эффективности визуализации с использованием данной системы, который свидетельствует о том, что для задач, удовлетворяющих установленным требованиям, предложенный подход является хорошим, а иногда и единственным решением.

ЛИТЕРАТУРА:

1. Kenneth Moreland «IceT Users Guide 2.0» [HTML] (http://***icet***.*sandia.gov*)
2. Will Schroeder, Ken Martin, Bill Lorensen «Visualization Toolkit. An Object-Oriented Approach to 3D Graphics»: Pearson Education Inc., 2002.
3. Kenneth Moreland, Lisa Avila , Lee Ann Fisk «Parallel Unstructured Volume Rendering in ParaView »
4. «ParaView Guide»: Kitware Inc., 2008 [HTML] (http://www.kitware.com/products/books/paraview.html).
5. «VisIt User's Manual»: U.S. Department of Energy by Lawrence Livermore National Laboratory, 2005 [HTML] *https://wci.llnl.gov/codes/visit*.
6. Stefan Eilemann «Equalizer» // VisSIG meeting, 2006 [HTML] *http://www.equalizergraphics.com/index.html*.
7. OSMesa - библиотека off-screeen рендеринга для OpenGL [HTML] *http://www.mesa3d.org/osmesa.html*
8. Ffmpeg - библиотека для кодирования/декодирования видео [HTML] *http://www.ffmpeg.org/documentation.html*

9. MPICH2 [HTML] *http://wiki.mcs.anl.gov/mpich2/index.php/Main_Page*
10. William Gropp, Rusty Lusk, Rob Ross, and Rajeev Thakur «Advanced MPI: I/O and One-Sided Communication ».
11. «Shmem Programming Manual »: Quadrics Supercomputers World Ltd .
12. T. Porter and T. Duff, “Compositing digital images” //ACM SIGGRAPH 84, July 1984.
13. K. Moreland, B. Wylie, and C. Pavlakos, «Sort-last parallel rendering for viewing extremely large data sets on tile displays» // IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, October 2001.
14. М.Ву, Т.Девис, Дж. Нейдер, Д.Шрайнер «OpenGL. Руководство по программированию. 4-е издание»: Питер, 2006.