# Адаптивный Конструктор Для Интерактивных Задач На Масс-Параллельных Машинах

Д.В. Манаков, М.Р. Шагубаков
ИММ УрО РАН, г. Екатеринбург
manakov@imm.uran.ru

## АННОТАЦИЯ

Развитие параллельной техники ставит перед компьютерной графикой ряд новых задач. Одна из наиболее важных – это реализация on-line визуализации для параллельных вычислений. В статье описан ряд подходов к реализации интерактивной параллельной графики. Предлагается разработка адаптивного конструктора (интегрированной системы редактирования, компиляции и запуска интерактивных задач, включающей функции терминала).

## КЛЮЧЕВЫЕ СЛОВА

on-line визуализация, интерактивная графика, метафайл, модель клиент-сервер, модульный подход, интерфейс пользователя, адаптивный конструктор.

## ВВЕДЕНИЕ

При создании средств графического вывода для масс-параллельных ЭВМ основное внимание, как правило, уделяется разработке систем off-line визуализации, в которых данные обрабатываются и выводятся после окончания многочасового (или даже многосуточного) счета. Вместе с тем в рамках параллельных вычислений существует необходимость реализации средств интерактивной графики, в частности, для поддержки начальных этапов моделирования, когда требуется вмешательство пользователя в процесс счета. Подобные средства on-line визуализации необходимы также для создания средств визуализации параллельного программного обеспечения, например, визуальных отладчиков правильности и эффективности.

В данной работе описываются некоторые результаты разработки систем интерактивной графики на масс-параллельных вычислителях типа МВС–100 и МВС–1000 [1], [2].

## ПОДХОДЫ К РЕАЛИЗАЦИИ ИНТЕРАКТИВНОЙ ПАРАЛЛЕЛЬНОЙ ГРАФИКИ

Наш опыт реализации интерактивной параллельной графики показывает, что наиболее просто можно реализовать передачу информации между параллельно работающими процессами и графической программой через файлы, с использованием флагов файла. Но более предпочтительно использование для этого двухзвенной модели клиент-сервер (клиент находится на управляющей машине параллельного вычислителя), с применением X-терминала.

Наиболее узким местом данного взаимодействия является организация обмена сообщениями между управляющей машиной и вычислителем (при отсутствии или неприменимости соответствующих системных средств). Для организации обмена можно использовать именованные каналы (не самый лучший вариант) или вставлять обмены в загрузочный модуль или в операционную систему параллельной машины.

Исследования показали преимущества трехзвенной модели реализации конструктора, предполагающей размещение программ комплекса на рабочей станции, на управляющей ЭВМ параллельного вычислителя и на его процессорах. При этом формирование изображения должно осуществляться на мощной рабочей станции. Как правило, математические данные занимают меньший объем, чем данные результирующего трехмерного изображения. Однако часть счета может выполняться и на рабочей станции. В тоже время, очевидно, что такие операции как вращение, перемещение и сжатие графического объекта, надо выполнять на рабочей станции. Реализацию обменов между рабочей станцией и управляющей машиной традиционно проводят с использованием Java.

Очевидно, что в условиях работы на параллельных вычислителях с ограниченными возможностями передачи информации по медленным сетям формирование растрового изображения на параллельных процессорах неэффективно. Довольно часто для передачи изображения в такой ситуации используются аппаратно-независимые графические протоколы или метафайлы. При этом на параллельных процессорах работает некоторая стандартная графическая система или графическая библиотека.

Правильно выбрав графическую библиотеку, мы обеспечиваем решение широкого класса задач. Плюсом является и то, что текст графической задачи, размещенной на одном из процессоров параллельного вычислителя, практически идентичен тексту последовательного варианта.

При разработке библиотеки, создающей метафайл, как правило, используются именованные функции. При этом передается номер функции, потом ее аргументы, а возвращается код ответа. Для графических функций код ответа часто не важен, поэтому используется буферизация, когда в одной посылке сообщения передается максимальное число функций. В многопроцессорном варианте используется идентификатор (номер) процесса для согласования момента рисования. Кроме того, в объектно-ориентированных библиотеках используется идентификатор объекта. Недостатком этого подхода является большое количество графических функций, для создания которых можно использовать автогенерацию, то есть синтаксический анализатор, который по тексту конкретной графической библиотеки создает библиотеку генерации метафайла.

Наиболее часто встречается и эффективно используется передача отфильтрованных математических данных путем создания графического сервера приложений, таких как, например, AVS или параллельная библиотека MATLAB [5]. По существу это тот же метафайл, только проблемно-ориентированный. Недостатком этого подхода является то, что он ориентирован на конкретные задачи.

Несмотря на разнообразие подходов к разработке задач on-line визуализации они не являются универсальными

и обладают рядом существенных недостатков. Разработка интерактивных средств, специально предназначенных для каждой отдельной задачи, трудоемка, а объединение таких средств, предназначенных для выполнения нескольких различных задач, в одну систему приводит к значительному увеличению используемых библиотек. Поэтому, для ускорения реализации интерактивных графических средств, обеспечивающих параллельное моделирование, нами предлагается создание интегрированной системы редактирования, компиляции и запуска интерактивных задач, включающей функции терминала, или адаптивного конструктора интерфейсов [2].

Наши работы ведутся в рамках обеспечения средств визуализации для отечественного параллельного вычислителя МВС-1000, установленного в ряде вычислительных центров страны.

## РОЛЬ АДАПТИВНОГО КОНСТРУКТОРА

Конструктор является средой для редактирования, компиляции и запуска двух задач, выполняющихся на разных компьютерах, в частности, на рабочей станции и параллельном вычислителе. Второй задачей конструктора является установление связи между работающими на различных компьютерах программами через их переменные и массивы. Также конструктор включает в себя функции терминала для удаленной машины и поддерживает автоматизированную разработку интерфейсных элементов и визуальных компонент для программы клиента (Рис 1).

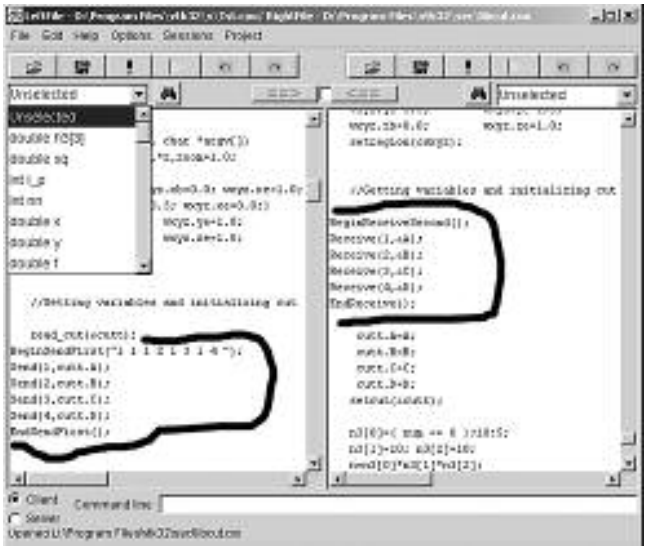

**Рисунок 1. Окно конструктора с редактируемыми исходными текстами**

В реализации конструктора используются две основополагающие идеи:

Во-первых, использование модели клиент-сервер [5], обеспечивающей ввод-вывод во время счета параллельной задачи.

Во-вторых, модульный подход [3], обеспечивающий переход от специализированных систем визуализации к унифицированным. В свою очередь этот подход приводит к попыткам разбить задачу на неизменяемые и варьируемые модули.

При установлении связей между модулями мы получаем возможность:

– автоматизировать этот процесс;
– отказаться от фиксированного формата данных по вводу-выводу;
– настраиваться на различные источники данных.

Поэтому возникает идея создания редактора межмодульного взаимодействия, который предназначен для быстрого перехода от задач off-line визуализации к интерактивным задачам on-line визуализации.

Можно выделить следующие модули интерактивной параллельной задачи:

1. параллельная часть – сервер;
2. графическая часть – клиент;
3. интерфейс пользователя;
4. организация обмена сообщениями между клиентом и сервером - конструктор.

Связь между клиентом и сервером предлагается осуществлять через переменные. Так как конструктор включает полный цикл **edit-compile-run**, после компиляции получается одна полноценная задача, готовая к выполнению. Перекомпиляция, использование языка Java, настройка на различные системы программирования снимает ряд проблем с переносимостью – таким образом, адаптивный конструктор является инструментом метакомпьютинга для программистов.


**Рисунок 2. Общая схема работы с конструктором**

Перед началом работы необходимо настроить конструктор на используемые трансляторы и библиотеки, указать местонахождение программ клиента и сервера. Настройка осуществляется в меню “Options\Directories”. Теперь можно начинать работу:

Открываем файлы, которые необходимо будет связать. В левом окне – программа с рабочей станции, в правом – программа, работающая на сервере. После этого необходимо зайти в пункт меню «Project\Save». Проекту будет присвоено имя Project[первое свободное число]. В дальнейшем, чтобы работать с этими файлами достаточно будет открыть сохраненный проект.

Расставляем передачи. Для этого выбираем в каждом окне позицию передачи (передача будет вставлена в текущую строку) и передаваемые переменные. Затем, нажав одну из кнопок, изображающих стрелки влево или вправо, определяем направление. Перед этим будет проверено, используется ли в настраиваемых программах библиотека для

осуществления обмена, и в случае необходимости, будет добавлена.

Можно задавать передачи в несколько этапов. Для этого достаточно сохранять проект. При открытии проекта уже существующие передачи будут восстановлены из файла. После того, как все передачи установлены, необходимо расставить сессии. Для этого надо выбрать пункт меню «Sessions/Add». В случае, если вам необходимо добавить передачи, сначала нужно удалить сессии, а после добавления снова расставить. По сути дела, сессии являются скобками, для объединения однотипных передач.

Теперь можно компилировать программы и запускать.

## РЕАЛИЗАЦИЯ ОБМЕНА СООБЩЕНИЯМИ

Для создания обмена сообщениями между задачами используется конструктор, как инструмент установления соответствия между переменными и моментов передачи данных между задачами. Преимуществом является то, что данные можно передать из любой части программы, в том числе осуществить промежуточную передачу для отслеживания хода программы или сохранения состояния программы. Кроме того, отсутствует привязка к конкретному протоколу передачи сообщений (в данное время используется MPI). Строгое соответствие между выбранными переменными двух задач и разбиение на сессии (парное событие, объединяющее группу операторов по вводу или выводу) по передаче данных обеспечивают:

– буферизацию, т.е. значения всех переменных в рамках одной сессии пересылаются одним сообщением;

– упреждение в блокирующем сообщении, т.е. посылка сообщения осуществляется в начале сессии, а прием в конце.

Рассмотрим на примерах случаи, возникающие при передаче данных:

Нарушение порядка

```
I=10;
J=25;
Send(I);        Receive(&J1)
Send(J);        Receive(&I1)
```

```
I=10;
J=25;
BeginSendFirst("1 2 1 1")     BeginReceiveSecond();
Send(1,I);                    Receive(1,&J1);
Send(2,J);                    Receive(2,&I1);
EndSendFirst()                EndReceiveSecond();
                              R=J1/100+I1*100;
```

Если бы мы считывали переменные без учета соответствия, то получили бы R=2500.1, а правильным будет R=1000.25.

Переда из **if**. Если значение переменной не было передано, то при получении значение переменной не изменится.

Передача из цикла – переданные значения воспринимаются как массив. При получении необходимо запросить переменную с этим номером столько раз, сколько было передано значений. Количество автоматически отслеживается и, в случае превышения, функция Receive возвращает 0.

## РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ ДЛЯ ПРОГРАММЫ КЛИЕНТА

В дальнейшем предполагается возложить на конструктор разработку интерфейса пользователя с механизмом связи от переменных к визуальным компонентам. Введем понятие обратной или адаптивной визуальной компоненты. Как правило, сначала выбирается компонента, например строка для ввода или кнопка, определяются ее свойства, а затем идет привязка к конкретным данным или функциям программы (непосредственное манипулирование). Предлагается, анализируя часть текста или данные программы, автоматически выбрать соответствующие компоненты и определить их свойства. На наш взгляд, такой подход упростит и ускорит разработку интерфейса пользователя. Так по группе операторов "case" можно автоматически построить компоненту меню, а анализируя тип данных – выбрать соответственно строку для ввода, список или таблицу. Конечно, свойства компонент надо задавать, но часть их можно определить автоматически. Предлагаемую модель интерфейса можно назвать автогенерацией по спецификации программиста [4].

## ОПИСАНИЕ ДЕМОНСТРАЦИОННОГО ПРИМЕРА

Для анализа работоспособности конструктора и определения ближайших перспектив развития была рассмотрена следующая тестовая задача. В трехмерном пространстве задана сетка со значениями функции в узлах. Так как объем данных очень большой, для его сокращения предполагается использовать сечение заданной плоскостью, в результате объем данных сокращается на размерность. По результирующим данным графическая программа строит двухмерную поверхность и ее линии уровня, используя библиотеку VTK (Рис 3). При переходе к интерактивному режиму для этой задачи можно выделить два этапа.

Сначала графическая (клиентская) часть передает значения плоскости в счетную (серверную) часть, а потом получает результирующие данные и визуализирует их. Меняя значения плоскости, можно организовать цикл. На рисунке 1 можно увидеть конструктор с редактируемыми исходными текстами, расставленными передачами и сессиями (обведены линиями), также показан механизм выбора переменной для установления обменов между программами «через переменные». После выполнения компиляции становятся доступными кнопки для запуска программ. В случае необходимости можно внести в тексты изменения, сохранить, перекомпилировать и запустить снова.


**Рисунок 3. Пример использования адаптивного конструктора для создания интерактивных графических программ**

## ПЕРСПЕКТИВЫ И РАЗВИТИЕ КОНСТРУКТОРА

Анализ реализации тестовой задачи показал работоспособность конструктора, Так же стало возможным более четко определить ближайшие развитие конструктора:

1. организация передачи более сложных типов данных (структур, функций, объектов);
2. разработка настройки взаимодействия с использованием различных протоколов обмена данными (в настоящее время реализованы обмен данными через файлы и с использованием стека протоколов TCP/IP);
3. разработка обратных визуальных компонент;реализация интерактивных задач на МВС-1000/16, возможно с переходом от двухзвенной модели клиент-сервер к трехзвенной.

Работа выполнена при поддержке РФФИ, гранты № 01-07-90210, № 01-07-90215.

## СПИСОК ЛИТЕРАТУРЫ:

1. Авербух В.Л., Исмагилов Т.Р., Манаков Д.В. Подходы к реализации интерактивной графики на МВС-1000 // Материалы Всероссийской конференции "Высокопроизводительные вычисления и их приложения", 2000 г., С. 241-244. Подмосковный филиал МГУ им. М.В.Ломоносова, Институт Проблем Химической Физики РАН,
2. Авербух В.Л., Манаков Д.В., Шагубаков М.Р. Разработка конструктора для интерактивных задач на масс-параллельных машинах // Материалы III Всерос. молодежной школы "Суперкомпьютерные вычислительно-информационные технологии в физических и химических исследованиях", 2001 г., С. 13-17. Черноголовка. Подмосковный филиал МГУ им. М.В.Ломоносова, Институт Проблем Химической Физики РАН, 2001,
3. Березин С.Б., Пасконов В.М. Компонентная система визуализации результатов расчетов на многопроцессорных вычислительных системах // Материалы Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения" ,2000 г., С. 202-203.
4. Вельтмандер П.В. Архитектуры графических систем \ Учебное пособие // http://ermak.cs.nstu.ru/kg_rivs/kg03.htm
5. Husbands P., Isbell Ch. The parallel problem server: a client-server model for large-scale scientific computation // www.mit.edu/parry/ppserver.ps