Визуальное программирование и PGRAPH

Способы разработки параллельных программ

Существует множество графических моделей описания алгоритмов параллельных вычислений. В области описания вычислительных алгоритмов наиболее удобной представляется форма, близкая описанию блок-схемам, которая реализована в параллельной версии технологии ГСП [1].

Технология графосимволического программирования GRAPH, созданная на кафедре ПС СГАУ, является одним из способов наглядного представления алгоритмов программы в виде графа управления. При разработке и реализации двухфазного алгоритма глобальной оптимизации модифицированным методом половинных делений использовалась средство моделирования параллельных вычислений PGRAPH 2.0.

Явный параллелизм и автоматическое распараллеливание

Работы в области моделирования и построения параллельных алгоритмов можно разделить на два больших направления:

* Неявный параллелизм. Это направление изучает методы автоматической генерации параллельных алгоритмов на оснfове их последовательных прототипов (автоматического распараллеливания последовательных алгоритмов).
* Явный параллелизм. Разработка методов организации вычислений, изначально ориентированных для реализации на ЭВМ с параллельной архитектурой.

Исследования в области автоматического распараллеливания вычислительных алгоритмов необходимы в связи с наличием большого объема ранее разработанных методов, алгоритмов и программ для решения различных задач на последовательных ЭВМ. Их реализация на параллельных ЭВМ требует модификации, связанной с распределением данных и вычислений по узлам параллельной ЭВМ, а также с адаптацией под особенности архитектуры конкретной ЭВМ. Этот процесс важно автоматизировать, чтобы максимально сократить его длительность и избавить исследователей - специалистов в различных областях, которые часто являются авторами и пользователями вычислительных программ, от знакомства со спецификой конкретной ЭВМ, на которой программа будет исполняться.

Автоматическое распараллеливание имеет большое значение и при создании новых вычислительных программ. Последовательные алгоритмы удобны и естественны для человека в силу того, что люди привыкли думать и действовать последовательно. Вместе с тем, любая современная ЭВМ обладает определенной степенью параллелизма.

Наряду с несомненными достоинствами, такими как возможность использования ранее разработанных и хорошо отлаженных последовательных программ, сохранение привычного для человека последовательного стиля разработки вычислительных алгоритмов, обеспечение переносимости программ, автоматическое распараллеливание обладает недостатками. Основным недостатком является ограниченная область применения. К сожалению, не все последовательные алгоритмы допускают эффективное распараллеливание. Иногда сам численный метод, на основе которого построен алгоритм, не допускает распараллеливания.

Для достижения максимальной производительности необходимо уже на этапе разработки алгоритма учитывать параллелизм и явно выделять участки, которые должны выполняться одновременно. Более того, необходимо учитывать архитектуру и особенности конкретной параллельной ЭВМ, на которой будет исполняться программа.

Основные сложности, с которыми сталкиваются исследователи в области построения параллельных алгоритмов с явным параллелизмом, в первую очередь связаны с наглядным представлением алгоритма.

Текстовая нотация, традиционно используемая в математике и программировании, удобна для представления последовательных процессов. Однако последовательная природа самого текста значительно затрудняет восприятие текстового описания параллельных вычислений. На первый план выдвигаются графические способы описания параллелизма.

Графические модели параллельных процессов

Основой подавляющего большинства графических способов представления параллельных процессов является форма представления в виде графа, то есть совокупности вершин (узлов), соединенных между собой дугами (ребрами). В отличие от текстовой формы записи, в которой объекты (символы и слова) образуют *последовательность*, а каждый объект связан только с левым и правым «соседом», графовая форма позволяет наглядно изображать более сложные *взаимосвязи*, поскольку в ней каждый объект может соединяться с несколькими другими объектами. В этом смысле текстовая форма одномерна, в то время как графовая форма – многомерна. Возможность варьировать геометрические размеры, форму и цвет вершин, внешний вид и толщину дуг, изменять взаимное расположение вершин без изменения топологии графа значительно увеличивают выразительные возможности графовой формы представления.

Графические модели обычно являются ориентированными графами, в которых дуги определяют направление передачи данных или зависимость между вершинами. Вершины и дуги обычно снабжаются текстовыми аннотациями, которые именуют их, перечисляют их содержимое или свойства. Различные графические модели отличаются друг от друга семантикой вершин и дуг. Основные классы графических моделей параллельных процессов:

* сети Петри [2];
* графы переходов, используются в SWITCH-технологии [3] и графическом языке Statecharts [4];
* диаграммы потоков данных, используются в системах CODE 2.0 [5], Paralex [6];
* диаграммы потоков управления, примерами моделей этого типа являются графический язык системы HeNCE [7], технология графо-символического программирования.

Графическая нотация является более наглядной и компактной, по сравнению с текстовым описанием. За счет использования графических моделей удается не только сократить время разработки параллельных алгоритмов, но и повысить их качество, т.к. графическая нотация допускает формальное математическое описание модели, по которому может быть проведена ее автоматическая верификация и оптимизация.

Концептуальная модель организации параллельных вычислений в ГСП

Технология ГСП в работе [1] определена как технология проектирования и кодирования алгоритмов программного обеспечения (ПО), базирующаяся на графическом способе представления программ, преследующую цель полной или частичной автоматизации процессов проектирования, кодирования и тестирования ПО.

Модель представляется четверкой <D, F, P, G>, где D – множество данных некоторой предметной области, F – множество операторов, определенных над данными предметной области, Р – множество предикатов, действующих над структурами данных предметной области, G = {A, Ψ, Φ, R} – ориентированный помеченный граф. A = {A1, A2, …, An} – множество вершин графа. Каждая вершина Ai помечена локальным оператором   
fi(D) ∈ F. На графе задано множество дуг управления Ψ = { Ψ1i1, Ψ1i2, …, Ψjm} и множество дуг синхронизации Ф = {Ф1i1, Ф1i2, …, Фj*l*}. R – отношение над множествами вершин и дуг, определяющее способ их связи. Дуга управления, соединяющая любые две вершины Ai и Aj, имеет три метки: предикат pij(D) ∈ P, приоритет k(Ψij) ∈ N и тип дуги T(Ψij) ∈ N. Каждая дуга синхронизации Фij помечена сообщением μij ∈ N. В классической (последовательной) модели вычислительного процесса, используемой в технологии ГСП, отсутствуют дуги синхронизации, которые вводятся в модель параллельного вычислительного процесса для решения задач синхронизации между его различными участками.

Развитие вычислительного процесса, описываемого моделью, ассоциируется с переходами из вершины в вершину по дугам управления. При этом переход по дуге управления возможен лишь в случае истинности предиката, которым она помечена. Если несколько предикатов, помечающих исходящие из вершины дуги, одновременно становятся истинными, переход осуществляется по наиболее приоритетной дуге. Функционирование модели начинается с выполнения оператора f0(D), помечающего начальную вершину A0.

Тип дуги Ψij определяется как функция T(Ψij) ∈ {1,2,3}, значения которой имеют следующую семантику:

T(Ψij) =1 – последовательная дуга (описывает передачу управления на последовательных участках вычислительного процесса);

T(Ψij) =2 – параллельная дуга (обозначает порождение нового параллельного вычислительного процесса);

T(Ψij) =3 –терминирующая дуга (завершает параллельный вычислительный процесс).

Для описания параллелизма вводится понятие параллельной ветвиβ - подграфа графа G, начинающегося параллельной дугой (тип этой дуги   
T(Ψij) = 2) и заканчивающегося терминирующей дугой (тип этой дуги   
T(Ψij) = 3). β = <Aβ, Ψβ, Rβ>, где Aβ – множество вершин ветви, Ψβ – множество дуг управления ветви, Rβ – отношение над множествами вершин и дуг ветви, определяющее способ их связи.

Дуги, исходящие из вершин параллельной ветви β, принадлежат также ветви β. При кодировании алгоритма, описанного с помощью предлагаемой модели, каждая параллельная ветвь порождает отдельный процесс – совокупность подпрограмм, исполняемых последовательно на одном из процессоров параллельной вычислительной системы.

Графическая модель обычно содержит несколько параллельных ветвей, каждая из которых образует отдельный процесс. В этом смысле модель параллельных вычислений можно представить как объединение нескольких параллельных ветвей:

G = U βk,

где βk – параллельные ветви графа G.

Таким образом, распараллеливание вычислений возможно только на уровне граф-модели. Вычисления в пределах любого актора выполняются последовательно.

Число параллельных ветвей в модели фиксируется при ее построении, при этом максимальное количество ветвей не ограничивается. Каждая ветвь имеет ровно один вход и один выход, для обозначения, которых в граф-модели используются два типа дуг: параллельная дуга и терминирующая дуга (рисунок 1):

- параллельная дуга

- терминирующая дуга

Рисунок 1 – Обозначение параллельных и терминирующих дуг в графической модели параллельных вычислений

Для описания правил построения граф-модели введем следующую систему обозначений:

To(Ai) – список дуг, входящих в вершину Ai,

Fr(Ai) – список дуг, исходящих из вершины Ai,

H(βj) – начальная вершина ветви βj,

| L | – число элементов в списке L.

Переход по параллельной дуге начинает работу параллельной ветви, переход по терминирующей дуге – заканчивает ее работу. Параллельная дуга не содержит предиката, т. е. переход по ней происходит безусловно.

∀Ψij ∈Ψ, T(Ψij) = 2 : Pij ≡ 1.

Из вершины может исходить не менее двух параллельных дуг. Если из вершины исходит параллельная дуга, то дуги других типов (обычная или терминирующая) не могут исходить из данной вершины.

T(Ψij0) = 2 → ∀j ≠ j0 : T(Ψij) = 2, | Fr(Ai) | ≥ 2.

Входящие в вершину дуги не могут быть одновременно параллельными и терминирующими:

T(Ψi0j0) = 2 → ∀i : T(Ψij0) ≠ 3.

Функционирование модели начинается с запуска единственной ветви, называемой мастер-ветвью (мастер-процессом). Обозначим мастер-ветвь β0. В вершинах мастер-ветви, имеющих исходящие параллельные дуги, порождаются новые параллельные ветви. Вершины этих ветвей также могут иметь параллельные дуги, таким образом, допускается вложенность параллельных ветвей. Ветви, породившиеся в одной вершине некоторой ветви, должны терминироваться также в одной вершине этой же ветви:

∀k: H(βk) = Ak0, Ψik0 ∈ Fr(Ai), Ai ∈ β1 →

→ ΨkN ∈ Т0(Aj), T(ΨkN) = 3, Aj ∈ β1.

В ветви 1 управление из вершины Аi после запуска ветвей k, k = k1, k2, …, K, передается вершине Aj. Вершина Aj запускается на выполнение после завершения работы ветвей k1 … K. Таким образом, в каждый момент времени в любой ветви выполняется ровно одна вершина.

Переход между двумя вершинами, принадлежащими различным параллельным ветвям, возможен только по параллельным и терминирующим дугам, то есть, запрещены условные переходы между вершинами различных параллельных ветвей:

∀i, j, Ai ∈ 1, Aj ∈ 2 → T(Ψij) ≠ 1.

Если некоторая ветвь породила новые параллельные ветви, то вычисления в ней приостанавливаются до завершения работы порожденных ветвей. Таким образом, вложенные параллельные ветви выполняются последовательно относительно друг друга.

Управление вычислительными процессами. Граф-машина

В технологии ГСП для объектов – агрегатов используется мониторная схема организации вычислений. В основу способа положено централизованное управление процессом вычислений, осуществляемое специальной программой – граф-машиной.

Граф-машина универсальна для любого алгоритма. Исходной информацией для граф-машины служит, описанная выше, модель графа управления вычислительным процессом. Анализируя его графическую модель, представленную в виде структур на смежной памяти [8], она выполняет в соответствующем порядке акторы и агрегаты, вычисляет значения предикатов и управляет синхронизацией. Для каждой параллельной ветви запускается по одному экземпляру граф-машины, которая представляет собой отдельный процесс в вычислительной системе.

Работа граф-машины начинается с выполнения актора в корневой вершине. Затем строится список дуг, исходящих из текущей вершины. Этот список просматривается граф-машиной последовательно, начиная с самой приоритетной дуги. Вычисляется значение предиката, помечающего дугу, и в случае его истинности, происходит переход к обработке следующей вершины. В результате обработки параллельной дуги в отдельном процессе запускается другая граф-машина, обрабатывающая порождаемую данной дугой параллельную ветвь. После запуска всех параллельных ветвей происходит переход в вершину, в которой они терминируются. Родительская граф-машина ожидает завершения выполнения всех дочерних граф-машин, если не задано альтернативное условие.

Централизация функций управления в рамках одной программы (граф-машины) на самом деле очень удобное решение, поскольку позволяет:

* контролировать вычислительный процесс в целом. И, в случае нештатных ситуаций, принимать системные решения;
* реализовать сбор статистической информации о характеристиках надежности каждого из модулей; вычислительной сложности модулей; маршрутах развития вычислительного процесса и т.п;
* выполнять пошаговую отладку программы в реальном времени и симулировать вычисления.

Межмодульный интерфейс параллельного обмена данными

Проблема передачи информации от одной программы к другой традиционно представляет собой сложную проблему, из-за которой возникают серьезные трудности в процессе автоматизации порождения кодов программ. Особенно остро эта проблема стоит в системах с распределенной памятью, таких как суперкомпьютерные кластеры. Для сохранения очевидных преимуществ, возникающих при использовании модели общей памяти, в системе PGRAPH последняя эмулируется за счет использования возможностей объектно-ориентированной парадигмы программирования: понятия класса языка С++.

Стандарт хранения и использования данных в ГСП

В технологии ГСП используется стандарт при организации межмодульного информационного интерфейса. Стандарт обеспечивается выполнением семи основных правил:

1. Вводится единое для всей предметной области программирования (ПОП) хранилище данных, актуальных для всей области. Полное описание данных размещено в словаре данных ПОП. Любые переменные, не описанные в словаре данных, считаются локальными данными для тех объектов ГСП, где они используются.
2. В пределах ГСП описание типов данных размещается централизовано в архиве типов данных.
3. Данные, актуальные для формируемого программного приложения, объединяются в единую универсальную структуру - класс TPOData.
4. В базовых модулях в качестве механизма доступа к данным допускается только передача параметров по адресу, ссылающемуся на универсальную структуру данных.
5. Привязка данных объектов ПОП к формальным параметрам базовых модулей реализована в паспортах объектов ПОП.
6. В технологии ГСП не рекомендуется использовать иные способы организации межпрограммных связей по данным.
7. Данные в ПОП могут быть общими и локальными. Память под общее данное выделяется в менеджере памяти, и все процессоры имеют доступ к этой переменной. Память под локальную переменную выделяется на каждом процессоре, и только этот процессор может читать и изменять её значение.

Данных подход к организации межмодульного информационного интерфейса приводит к тому, что формируемые (автоматически или автоматизировано) программные коды и информационные связи «пространственно» отделены друг от друга. Модификация любого из объектов (актора, предиката или агрегата) не требует переделки кодов других объектов, входящих в ПОП. Физически данные ПОП хранятся в общей области памяти базового компьютера. Параллельный вариант граф модели при использовании технологии MPI предполагает распределенное размещение данных на разных компьютерах некоторого кластера. Для соблюдения вышеназванных условий в параллельной версии технологии ГСП программно реализована модель общей памяти для аппаратной архитектуры с распределенной памятью.

Способ реализации общей памяти в ГСП

В среде выполнения программы выбирается машина, на которой будет запущен процесс, отвечающий за хранение переменных ПОП. Учитывая аппаратные особенности и топологию ВС, это может быть узел с наибольшим объемом оперативной памятью или центральный узел, имеющий минимальное время доступа от любого из остальных узлов кластера. Преимущество данного подхода в том, что значительно экономится ресурс памяти на вычислительных узлах, т.к. на узлах память выделяется только под те переменные, которые используются.

Описанная идея организации хранения и обмена данными между параллельными процессами ориентирован на модель передачи сообщений, в которой каждый процесс работает с локальными данными. Например, стандарт MPI подразумевает, что процессы обмениваются данными только в результате пересылки их в виде сообщений.

Предлагаемый способ обмена данными требует введения понятия диспетчера данных – подпрограммы, выполняющей функции хранения, чтения и модификации данных предметной области.

Диспетчер памяти

Диспетчер данных исполняется в отдельном процессе параллельной программы. Он порождает объект, описываемый классом TPOData, который хранит значения данных предметной области. В каждом из процессов, содержащих параллельные ветви граф-модели, также порождается объект класса TPOData. Однако функции доступа к членам-данным у объекта диспетчера данных и у объектов параллельных ветвей различаются. Диспетчер данных хранит все данные в локальной памяти и для обращения к ним использует обычные указатели. На остальных процессах используется ленивая инициализация памяти под переменную при первом доступе.

В параллельных ветвях граф-модели для чтения или записи некоторого данного осуществляется обращение к диспетчеру памяти с помощью совокупности сообщений. В первом сообщении пересылается запрос на чтение или запись конкретного данного. Каждая переменная из ПОП получает уникальный номер, по которому диспетчер памяти может ее идентифицировать. В случае чтения параллельная ветвь переходит к ожиданию ответа от диспетчера данных. При записи во втором сообщении пересылается новое значение переменной. Диспетчер данных циклически принимает и обрабатывает запросы параллельных ветвей (рисунок 2).

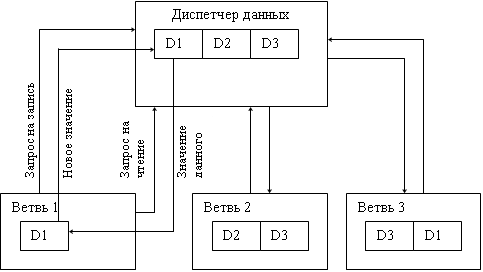


Рисунок 2 – Модель общих данных системы PGRAPH

Обзор класса TPOData

Для сохранения очевидных преимуществ, возникающих при использовании модели общей памяти, в системе PGRAPH последняя эмулируется за счет использования возможностей объектно-ориентированной парадигмы программирования: класса TPOData.

Класс TPOData – это ядро механизма хранения и передачи данных в технологии ГСП. Класс TPOData инкапсулирует все данные ПОП и предоставляет доступ к ним через поля-свойства. Свойство — способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Обращение к свойству объекта выглядит так же, как и обращение к полю объекта, но, в действительности, реализовано через вызов функции. При попытке задать значение данного свойства вызывается один метод (*setter*), а при попытке получить значение данного свойства — другой (*getter*). Полей ровно столько, сколько переменных в ПОП, и их названия совпадают с названиями переменных. Для переменных простых типов в классе TPOData описано по одному методу доступа для чтения и установки значения. Для массивов определено по два метода чтения и установки: доступ ко всему массиву и к элементу по индексу.

Доступ к данным и акторов

Класс TPOData и экземпляр класса напрямую недоступен для разработчика граф-программ и программист не обязан знать о его существовании. Когда программист обращается к переменной ПОП, на самом деле он обращается к полям-свойствам объекта D.

Рассмотрим использование свойств на примере создания inline-актора. Описание данных и переменных ПО приведено в таблице 1 и 2

Таблица 1 – Описание типов данных ПО

| Название | Определение | Описание |
| --- | --- | --- |
| int |  | Встроенный тип целых чисел |
| double |  | Встроенный тип чисел с двойной точностью |
| array | typedef int array[100] | Массив целых чисел длины 100 |

Таблица 2 – Описание данных ПО

|  |  |
| --- | --- |
| Название | Тип |
| A | int |
| B | double |
| C | array |

Пример доступа к переменным:

a = 100; //устанавливает значение a в 100

double \_b = b; //читает значение b в локальную переменную \_b

b+=1; //инкрементирует значение b

for (int i = 0; i < c.length; i++)

c[i] = i; //инициализирует каждый элемент массива

С помощью компилятора объектов ГСП данный код перед сохранением, скрытно от пользователя будет транслирован в следующий:

D->a = 100; //устанавливает значение a в 100

double \_b = D->b; //читает значение b в локальную переменную \_b

D->b+=1; //инкрементирует значение b

for (int i = 0; i < D->c.length; i++)

D->c[i] = i; //инициализирует каждый элемент массива

В примере показана дополнительная возможность массивов – это получение их длины, как значения поля length.

Ограничения использования свойств

В целом, с плеч программиста снимается тяжелый груз забот об управлении данными с помощью функций MPI, и обмен данными между процессами становится “похож” на обмен данными между нитями внутри одного процесса. Однако реализованный механизм полей-свойств накладывает ряд ограничений на использование данных.

1. Переменная ПОП не может использоваться в качестве параметра функции с переменным числом аргументов (например printf(const char \*, …)) и не может передаваться по указателю в функцию (например scanf(const char \*, void \*)). В подобных случаях необходимо создать локальную буферную переменную, через которую читать и писать в переменную ПОП. Это ограничение касается использования свойств только в inline-модулях.
2. Переменные в общей памяти не могу иметь тип “указатель”. Следствием этого является то, что в общей памяти нельзя создавать массивы переменной длины.
3. В качестве переменных предметной области нельзя использовать многомерные массивы, как следствие запрета 2. Необходимо представить многомерный массив в виде одномерного.

Программное средство моделирования и разработки алгоритмов параллельных вычислений

Архитектура программного комплекса моделирования и анализа алгоритмов параллельных вычислений

На рисунке 3 приведена структура программного комплекса моделирования и анализа алгоритмов параллельных вычислений.

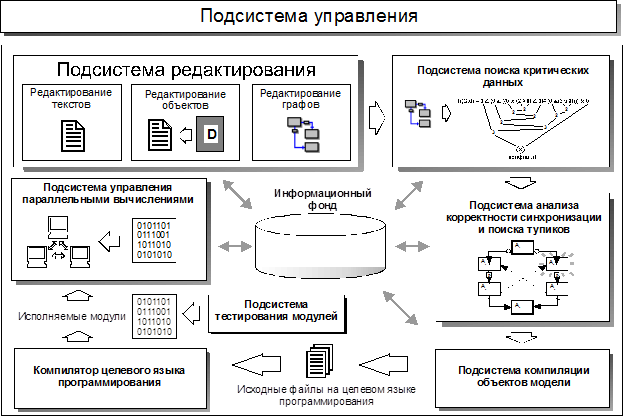


Рисунок 3 – Структура программного комплекса моделирования и анализа алгоритмов параллельных вычислений

Ядром системы является информационный фонд, который содержит информацию о переменных предметной области, объектах и созданных пользователем системы граф-моделях. Эта информация поступает в базу данных в результате работы пользователя с редакторами подсистемы редактирования. Исходный текст базовых модулей и предикатов создается в редакторе текстов, который представляет собой обычный текстовый редактор, дополненный средствами проверки синтаксиса базовых модулей.

Редактор объектов предназначен для создания акторов и предикатов на основе базовых модулей с помощью операции паспортизации. Агрегаты создаются пользователем в редакторе графов – специализированном графическом редакторе, оперирующем визуальными объектами модели, такими как вершины и дуги.

После создания графической модели алгоритма, перед запуском процесса вычислений, осуществляется проверка корректности модели.

Информация о разработанных пользователем акторах, предикатах и агрегатах используется подсистемой компиляции объектов модели. На основе данных информационного обеспечения она строит исходные файлы на некотором целевом языке программирования. В настоящее время реализована поддержка языка C и C++, однако систему можно расширить для работы с другими языками программирования. Таким образом, пользователю предоставляется возможность использования наиболее удобного для данной задачи языка или языка, наиболее известного пользователю, в котором он имеет большую библиотеку написанных и отлаженных БМ.

Для работы с каждым целевым языком программирования создается отдельный компилятор объектов и при желании – отдельный редактор текстов, в качестве которого может применяться, например, специализированный редактор среды программирования на целевом языке.

Для преобразования исходных текстов в исполнимый модуль используется компилятор целевого языка программирования. Как правило, это программный продукт, созданный сторонним разработчиком и входящий в состав среды программирования на целевом языке. Такой подход придает системе гибкость, повышает переносимость и надежность создаваемых программ. Исполняемые модули по желанию пользователя системы могут генерироваться для различных платформ и операционных систем (например, для SMP-компьютеров под управлением операционной системы Windows 2007 или Unix-кластеров).

Подсистема управления параллельными вычислениями работает с исполняемыми модулями и выполняет такие действия, как распределение программ по узлам вычислительной системы, запуск программ на выполнение и сбор статистических данных об их работе. В состав подсистемы управления параллельными вычислениями могут входить, например, средства анализа трассировочной информации о выполнении программы для оптимизации ее производительности. Чаще всего эта подсистема входит в состав операционной системы или является сторонним продуктом, предустановленным на кластере, например, система управления заданиями Torque [9].

Программный комплекс моделирования и анализа алгоритмов параллельных вычислений PGRAPH 2.0

В рамках настоящей работы разработан и реализован программный комплекс PGRAPH 2.0, предназначенный для визуального построения граф-моделей параллельных алгоритмов и автоматической генерации программ на основе этих моделей.

Программный комплекс ориентирован на работу в модели передачи сообщений. Генерируемые программы могут исполняться как на вычислительных системах с общей памятью, так и в распределенных системах. Механизм передачи сообщений между параллельными процессами базируется на технологии MPI (Message Passing Interface), которая представляет стандартизованные средства передачи сообщений в различных операционных системах на различных аппаратных платформах.

Программный комплекс универсален и может работает под управлением операционной системы Windows или Linux (с установленной графической оболочкой). Генерируемые параллельные программы могут работать в любой операционной системе, для которой имеется реализация MPI. Например, для создания программы, ориентированной на Linux-кластер, необходимо лишь наличие библиотеки MPI и компилятора для соответствующей версии Linux. Выбор языка программирования для описания базовых модулей также ограничен лишь наличием реализации MPI для этого языка. В настоящее время реализована версия программного комплекса PGRAPH 2.0, использующая для написания базовых модулей язык С++.

Архитектура программного комплекса соответствует архитектуре, представленной на рисунке 3. Для взаимодействия с пользователем используется графический оконный интерфейс. Подсистема управления параллельными вычислениями интегрирована с графическим редактором граф-моделей. Благодаря этому, при выполнении различных действий в системе, пользователь всегда видит текущий агрегат, над которым эти действия выполняются (рисунок 4).

Информационный фонд программного комплекса хранится в базе данных, поддерживаемой СУБД MySQL [10]. Выбор этой СУБД обусловлен тем, что она имеет реализации для различных операционных систем, поддерживает язык структурированных запросов к базам данных SQL (Structured Query Language) и распространяется свободно.

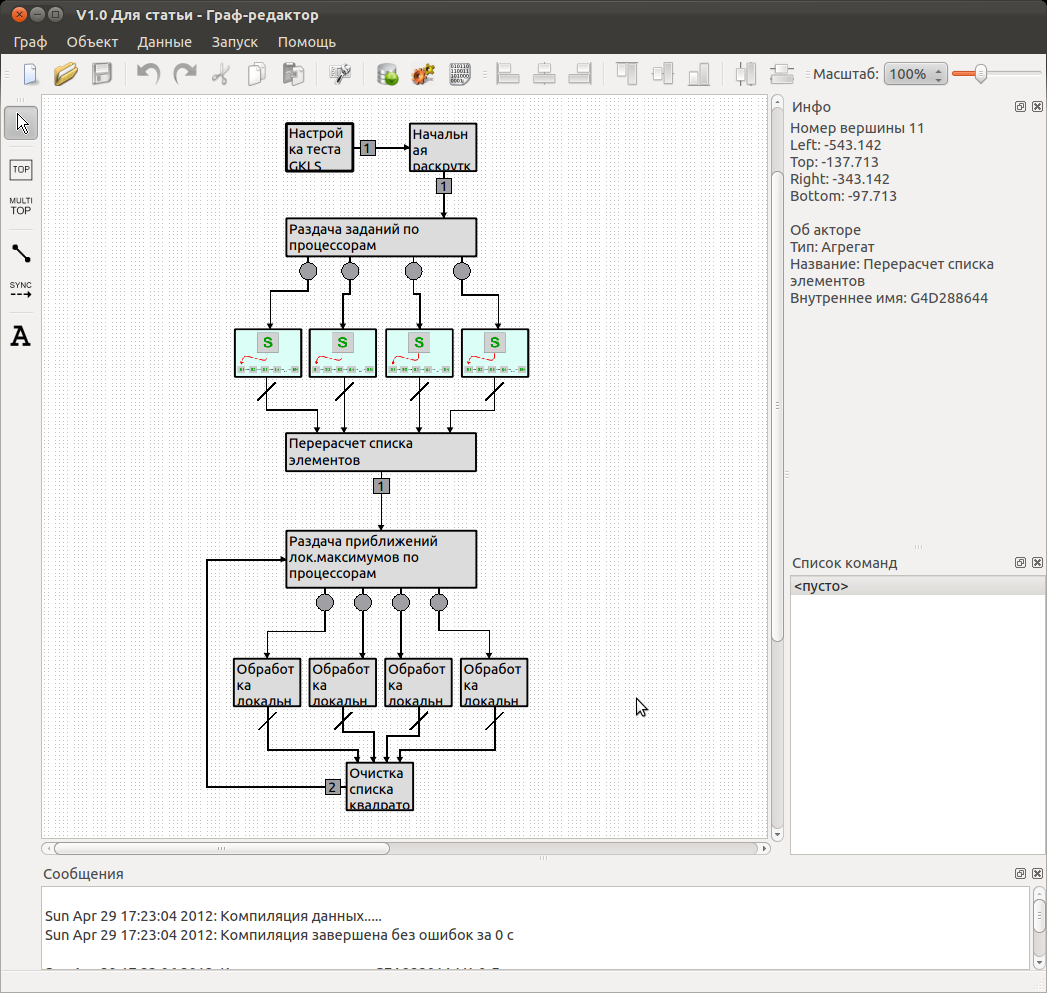


Рисунок 4 – Главное окно программного комплекса моделирования и анализа алгоритмов параллельных вычислений PGRAPH 2.0

В систему интегрирован редактор текстов, предназначенный для написания исходных текстов базовых модулей и выполняющий простейшие операции по проверке их синтаксической корректности, а также редактор объектов технологии ГСП. Компилятор объектов выполнен в виде отдельного модуля. Он позволяет на основании описания объектов, хранящегося в информационном фонде системы, генерировать исходные тексты на языке С++. В качестве компилятора исходных текстов используется компилятор для целевой платформы, поддерживающий конкретную библиотеку MPI. Для Linux кластеров имеется большое число различных реализаций MPI и компиляторов. Для Windows систем наиболее типичен вариант сочетания библиотеки MPICH2 и компилятора Microsoft Compiler.

Подсистема редактирования

В данной работе было продолжено совершенствование граф-редактора, разработанного в бакалаврской работе Аболмасова П.В. [11].

Кроме исправлений незначительных ошибок в редакторе реализованы следующие новые функции:

* интеграция с подсистемой компиляции и подсистемой запуска;
* автоматическое геометрическое выравнивание объектов модели;
* сохранение истории действий пользователя;
* масштабирование холста модели;
* особая вершина, реализующая паттерна «все делают одно и тоже».

Генерация исходных текстов параллельных программ на языке С++

Для построения исполнимых файлов программный комплекс генерирует исходный текст программы на языке С++. Это происходит скрыто от пользователя в автоматическом режиме. В процессе генерации используется описание объектов технологии ГСП, хранящееся в информационном фонде. Полученные исходные тексты подаются на вход компилятора C++, который автоматически вызывается программным комплексом. Генерируемая программа на языке С++ состоит из нескольких модулей (рисунок 8).

Для каждого объекта технологии ГСП компилятор объектов генерирует отдельный файл с исходным текстом. Для агрегатов этот файл состоит из двух частей:

1. заголовка, содержащего структуры данных, описывающих граф-модель;
2. вызова отдельной процедуры, реализующей работу граф-машины.

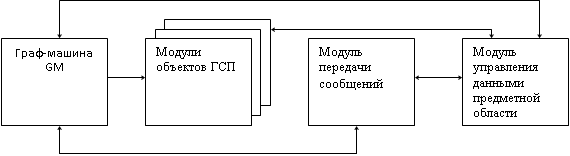


Рисунок 8 – Структура программы, автоматически генерируемой на основе граф-модели алгоритма

Для акторов и предикатов генерируется файл с исходным текстом реализующего их базового модуля, в котором все обращения к данным предметной области заменены на специальные конструкции в соответствии с межмодульным информационным интерфейсом параллельной программы.

Компилятор данных

Компилятор данных собирает из типов и данных ПОП класс TPOData. В основном при создании класса TPOData решается задача определения типа переменной.

Выходные файлы строятся компилятором на основе шаблонов. В шаблоне определены теги, которые заменяются компилятором на соответствующие конструкции. Всего имеется три файла шаблона:

* utypes.h.template – шаблон, в котором описаны типы данных;
* tpodata.h.template – заголовочный файл класса TPOData;
* tpodata.cpp.template – исходный текст класса TPOData.

По количеству шаблонов создается 3 выходных файла. Для каждого определенного типа данных в файле utypes.h имеется запись. Если речь идет о программе на MPI, то в файле utypes.h создается соответствующий тип MPI для каждого пользовательского типа. В соответствии со стандартом MPI каждый новый пользовательский тип должен быть зарегистрирован в программе перед его использованием. Код для регистрации нового типа генерируется автоматически.

Для каждой переменой из ПОП в классе TPOData создается поле-свойство, для доступа к переменной и метод установки и получения значений, работающие с этим свойством. Класс TPOData описываются в файле tpodata.h. а его реализация, соответственно, помещается в файл tpodata.cpp.