Методы и языковые средства описания взаимосвязанных задач в распределенных пакетах прикладных программ

И.А. Сидоров, Е.И. Поздняк

Учреждение Российской академии наук
Институт динамики систем и теории управления Сибирского отделения РАН

В работе рассматриваются различные подходы к описанию схем решения задач, допускающих декомпозицию составных частей общей задачи на отдельные блоки и последующее решение набора полученных подзадач в распределенной вычислительной среде. Обсуждаются преимущества и недостатки каждого подхода, приводятся примеры их реализации в инструментальном комплексе DISCOMP.

1. Введение

Интенсивное развитие сетевых технологий и аппаратных средств, наблюдаемое в последние годы, позволило многократно повысить производительность вычислительных систем и обеспечило возможность организации эффективных параллельных вычислений. Качественные требования к реализации процесса параллельной обработки данных в прикладных программах (например, необходимость обеспечения эффективности, масштабируемости, переносимости и т.д.) породили большое многообразие систем для организации параллельных вычислений (см., например, работы В.Э. Малышкина, В.С. Бурцева, В.А. Васенина, В.В. Воеводина, А.Б. Жижченко, В.Н. Коваленко, В.В. Корнеева, Д.А. Корягина, А.О. Лациса, В.В. Топоркова и др.). Такие системы, как правило, требуют от специалиста-предметника достаточно высокого уровня программистской квалификации и навыков разработки параллельных программ (в их числе проблемы выявления внутреннего параллелизма алгоритма, необходимость следования тем или иным технологиям и моделям параллельного программирования, учета архитектуры и особенностей используемой вычислительной системы).

Однако существует широкий класс ресурсоемких задач, для решения которых не требуется существенной модификации реализующих алгоритмов и их адаптации к применению на высокопроизводительных вычислительных системах. Такие задачи могут характеризоваться необходимостью проведения многовариантных расчетов над полем независимых между собой входных данных различными программами для их обработки. Одним из инструментов организации такого рода вычислений являются пакеты прикладных программ (ППП), в составе которых выделяют три компонента: функциональное наполнение, высокоуровневые языковые средства описания исследуемой предметной области и системное программное обеспечение (ПО) для организации процесса решения исследовательской задачи. Сочетание в ППП разнообразных сложных моделей, алгоритмов и методик их исследования базируется на использовании принципа модульной организации функционального наполнения пакета. Использование принципа модульности позволяет заменить написание программы (в традиционном понимании) конструированием ее схемы на основе готовых программных блоков крупного размера.

Создание инструментальных средств, обеспечивающих разработку и применение ППП для параллельных и распределенных вычислительных систем (РВС), является одним из наиболее перспективных и актуальных направлений развития пакетной проблематики. Большинство инструментальных систем (например, OLYMPUS, ПРИЗ, САФРА, СПОРА и др.) разрабатывались, в основном, для создания традиционных ППП. Применение таких инструментариев для построения ППП, функционирующих в распределенных средах, является затруднительным. Известные системы организации распределенных вычислений (например, кластерная система Condor, программный комплекс BOINC, инструментарий X-COM и др.) позволяют осуществить в РВС решение не связанных между собой задач, допускающих распараллеливание по данным, но не обладают необходимыми возможностями для организации многофункциональ-

ных пакетов, поддерживающих выполнение взаимосвязанных прикладных программ.

В данной работе рассматривается инструментальный комплекс (ИК) DISCOMP [1], ориентированный на автоматизацию разработки и применения распределенных пакетов прикладных программ (РППП) в разнородных РВС. РППП ориентированы на класс задач, характеризующихся следующими свойствами:

- решение задачи требует проведения расчетов на ЭВМ с использованием больших объемов вычислительных ресурсов (процессорного времени, оперативной памяти, дискового пространства и др.);
- возможна декомпозиция общей сложной задачи на более простые (с вычислительной точки зрения) подзадачи;
- процесс решения общей задачи подразумевает распределенное решение набора ее взаимосвязанных подзадач;
- не предполагается интенсивного взаимодействия между параллельными процессами решения подзадач;
- задача допускает декомпозицию данных на блоки и независимую параллельную обработку этих блоков.

Ниже приведены характерные особенности среды функционирования, языковых средств, функционального наполнения и системной части РППП.

Средой функционирования РППП является РВС с разнородными вычислительными узлами, организованными на основе различных программно-аппаратных платформ и управляемыми разными операционными системами (ОС). Зачастую разнородные РВС имеют низкую степень отказоустойчивости.

Системная часть включает ряд распределенных подсистем и предоставляет средства для организации вычислений на основе удаленного запуска модулей и распределенного обмена данными. Все компоненты системной части являются платформо-независимыми и могут функционировать под управлением различных ОС (например, MS Windows, Linux, Mac OS X и др.).

Функциональное наполнение РППП составляют модули, представленные в виде исполняемых в пакетном режиме программ, реализованных на различных языках программирования (например, C, Fortran, Pascal). Модули размещаются в разных узлах РВС, в каждом узле РВС может быть установлено несколько модулей. Допустимо включение в состав функционального наполнения унаследованного ПО, а также нетиражируемых программных комплексов, жестко привязанных к узлам РВС. Обмен данными между модулями осуществляется через файлы.

Языковые средства РППП предоставляют возможности для описания концептуальной модели предметной области, ее объектов и основных свойств, задания начальных данных и пр. Главным назначением языковых средств РППП является предоставление разработчику пакета различных способов для формирования предметно-ориентированных параллельных схем решения задач в разнородных РВС. В общем случае, благодаря гибкости и универсальности языковых средств РППП, становится возможным расширить класс допустимых задач, более эффективно использовать ресурсы РВС и повысить надежность вычислений.

Данная работа посвящена подробному обсуждению различных подходов к описанию параллельных схем решения задач в ИК DISCOMP.

2. Модель распределенных вычислений

Формализованное описание предметной области РППП относится к классу вычислительных моделей [2] и представляет собой совокупность значимых параметров предметной области, а также модулей РППП, реализующих вычислительные операции с этими параметрами. В простейшем случае описание предметной области [3] РППП можно определить в виде структуры $S =< Z, T, M, Y >$, где Z – множество параметров, T – множество допустимых типов параметров, M – множество модулей, Y – множество вычислительных единиц РВС, которые способны выполнить тот или иной модуль из M. Связи между элементами множеств *Z, T, M* и *Y* заданы отношениями $ZT \subset Z \times T$, $IN \subset Z \times M$, $OUT \subset Z \times M$ и $MY \subset M \times Y$ (в общем случае типа «многие-ко-многим»). Для каждого объекта структуры *S* (параметра, типа параметра, модуля и вычислительной единицы) определен набор его атрибутов.

Множество *типов параметров T* включает: тип *file*, используемый для описания параметров неопределенной структуры (блоков произвольного текста большого размера); тип *filelist*, предназначенный для поддержки распараллеливания по данным (параллельный список параметров типа *file*). К основным атрибутам *параметра* относятся: имя параметра; тип параметра; ограничения на параметр (максимальное количество элементов, лимит предоставляемого дискового пространства). К основным атрибутам *модуля* относятся: имя модуля; тип; список входных параметров; список выходных параметров; команда запуска модуля (способ запуска модуля, аргументы командной строки); требования к среде выполнения (архитектура процессора, объем оперативной памяти, тип операционной системы, зависимые системные библиотеки); ограничения при выполнении (максимальное время выполнения, объем предоставляемого дискового пространства и оперативной памяти). К основным атрибутам *вычислительной единицы* относятся: имя; список установленных модулей; характеристики операционной системы, процессора, памяти, дискового пространства, и др.

Содержательно модуль $m_i \in M$ реализует возможность вычисления множества параметров $OUT^i \subset Z$ по множеству параметров $IN^i \subset Z$, а множества IN^i и OUT^i называются соответственно множествами входных и выходных параметров для модуля m_i. Поэлементная обработка параметра z_j типа *filelist* модулем m_i выполняется следующим образом: *k*-й элемент параметра z_{jk} обрабатывается *k*-м экземпляром модуля m_{ik}.

Постановка задачи для структуры *S* задается в процедурном виде и в общем случае формулируется следующим образом: «зная *S* выполнить *Q*», где $Q \subset M$ представляет собой частично упорядоченную последовательность модулей из *M*, которые необходимо выполнить для решения задачи.

Схемой решения задачи (СРЗ) в ИК DISCOMP называется модель крупноблочной программы, отражающей информационно-логическую структуру вычислений в терминах предметной области. Таким образом, вычислительный процесс (процесс интерпретации схемы решения задачи) предполагает выполнения ряда зависимых между собой подзадач (модулей).

Далее в работе представляются различные подходы к формированию СРЗ в порядке их реализации в ИК DISCOMP.

3. Ярусно-параллельная форма

Ярусно-параллельная форма является одним из самых естественных и простых подходов к описанию распараллеливания некоторой комплексной задачи [4]. Схему решения задачи, задаваемую в ярусно-параллельной форме, графически можно представить в виде направленного ациклического графа, включающего два непересекающихся множества вершин (рис. 1): множества вершин-параметров и вершин-модулей. Вершины-параметры, как и вершины-модули, являются попарно несмежными. На рис. 1 модули представлены овалами, параметры – кружками.

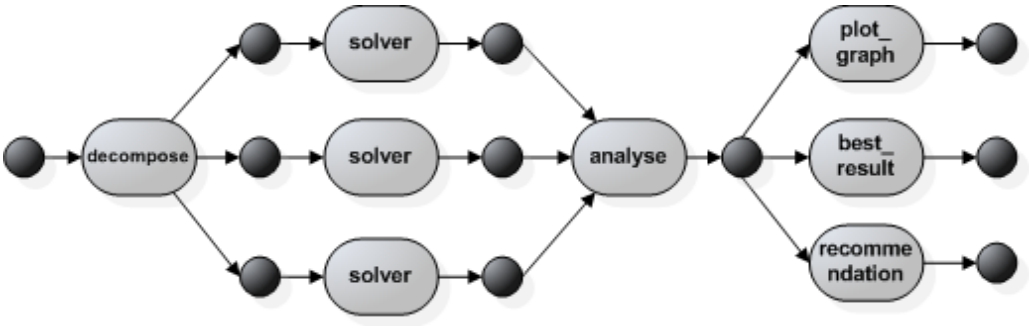

Рис 1. Ярусно-параллельная форма СРЗ

В терминах представленной выше модели при установлении частичного порядка множество *Q* разбивается на *k* непустых подмножеств. Упорядочение подмножеств осуществляется в зависимости от того, какие модули должны быть выполнены раньше. В рамках каждого *k*-го подмножества входящие в него модули могут выполняться независимо друг от друга в любой

последовательности или параллельно.

Описание частично упорядоченной последовательности в ИК DISCOMP осуществляется на специализированном языке, разработанном на основе расширяемого метаязыка разметки XML. Спецификация постановки задачи включает список ярусов (*stage*), на каждом из которых размещаются определенные модули (*module*). Модуль, предназначенный для поэлементной обработки параметра-списка, задается в СРЗ как модуль-список (*listmodule*).

Ниже приводится пример СРЗ, соответствующей приведенному на рис. 1 графу. На первом ярусе должен быть выполнен модуль *decompose*, формирующий элементы параметра типа *filelist*. На втором ярусе производится поэлементная (параллельная) обработка полученного параметра-списка экземплярами модуля *solver*. На третьем ярусе запускается модуль *analyse*, который анализирует полученные на предыдущем ярусе значения параметра-списка. И на четвертом ярусе одновременно (параллельно) могут быть запущены модули *plot_graph*, *best_result* и *recommendation*.

```
<scheme>
      <stage>
            <module name='decompose'/>
      </stage>
      <stage>
            <listmodule name='solver'/>
      </stage>
      <stage>
            <module name='analyse'/>
      </stage>
      <stage>
            <module name='plot_graph'/>
            <module name='best_result'/>
            <module name='recommendation'/>
      </stage>
</scheme>
```

Интерпретация СРЗ, заданной в таком виде, выполняется в соответствии с принципами ветвления и слияния FORK/JOIN [5]. Переход FORK моделирует «разветвление» – создание из одной ветви выполнения двух или более параллельных ветвей. Это, как правило, реализуется путем создания дополнительных ветвей вдобавок к существующим. Переход JOIN, в свою очередь, осуществляет «слияние» нескольких ветвей по завершению их работы (уничтожение созданных параллельных ветвей за ненадобностью).

3.1. Событийно-управляемые конструкции

Задание схемы решения задачи в виде типовой ярусно-параллельной формы накладывает ряд ограничений значительно сужающих допустимый класс решаемых задач. К числу наиболее существенных недостатков следует отнести отсутствие средств для описания логических операторов, необходимых для отсечения ненужных веток в процессе вычислений.

С целью повышения степени интеллектуализации средств управления вычислительным процессом в спецификацию схемы решения задачи заложена возможность включения управляющих конструкций для анализа текущего состояния вычислительного процесса и принятия решений о дальнейшем ходе вычислений. Применение таких конструкций основывается на системе событий, происходящих при интерпретации СРЗ. К событиям относятся:

- запуск модуля (или яруса);
- завершение работы модуля (или яруса);
- некорректное завершение модуля (или яруса);
- вынужденная остановка модуля (или яруса);
- периодическая проверка результатов вычислений для модуля (или яруса), выполняемая с определенным интервалом.

Для каждого структурного элемента СРЗ (модуля или яруса) при необходимости указывается тип обрабатываемого события и непосредственно сам обработчик, реализованный в виде

функции, содержащей блок операторов на высокоуровневом языке программирования *JavaScript*. Для взаимодействия с интерпретатором СРЗ используется специализированный программный интерфейс *DiscompAPI*, средства которого позволяют получать/изменять значения параметров, останавливать/запускать требуемый модуль (все модули на ярусе), и в некоторых случаях передавать управление на необходимый ярус.

Ниже приведен пример описания условия остановки вычислений при обработке параметра типа *filelist*. После завершения каждого экземпляра модуля *solver* запускается обработчик *checkListResult*, который выполняет анализ выходного элемента списка. Если значение выходного элемента содержит подстроку *«RESULT FOUND»*, то производится остановка всех модулей на текущем ярусе и процесс вычислений завершается. Применение данной конструкции позволяет избежать избыточных вычислений при решении комбинаторных задач большой размерности, основным назначением которых является поиск результата, удовлетворяющего определенным условиям.

```
<scheme>
      <stage>
            <listmodule name='solver'
                        onFinish='checkListResult($element_num)'/>
      </stage>
</scheme>
<control><![CDATA[
      function checkListResult (element_num) {
            // получить значение выходного элемента параметра-
            // списка result с порядковым номером element_num
            var res = DiscompAPI.getLPV('result', element_num);
            // если значение элемента содержит подстроку
            // "RESULT FOUND", то остановить вычисления на текущем
            // ярусе и перейти к следующему
            if ( res.match(/RESULT FOUND/) ) {
                  DiscompAPI.stopStageModules ();
            }
      }
]]></control>
```

Следующий пример иллюстрирует способ обработки события *onFinish* при параллельном запуске модулей, реализующих различные алгоритмы решения одной и той же задачи и идентичных по формату входных и выходных параметров. Применение подобной конструкции целесообразно, если заведомо неизвестно, какой из алгоритмов окажется наиболее эффективным для обработки заданных входных значений параметров.

```
<stage>
      <module name='m1' onFinish='DiscompAPI.stopStageModules();'/>
      <module name='m2' onFinish='DiscompAPI.stopStageModules();'/>
      <module name='m3' onFinish='DiscompAPI.stopStageModules();'/>
</stage>
```

Описанные в данном разделе языковые средства ИК DISCOMP успешно применены при решении задач моделирования логистических складских систем, исследования биоресурсов озера Байкал, построения множеств достижимости управляемых летательных объектов, при решении систем булевых уравнений. В качестве отдельного примера стоит выделить РППП D-SAT [6], реализующий технологию крупноблочного распараллеливания SAT-задач.

Однако, несмотря на все описанные усовершенствования, приведенные языковые средства не позволяют описывать сложные недетерминированные схемы, в которых порядок выполнения модулей в цепочке может задаваться неоднозначно. Построение таких схем требует наличия в составе инструментария более мощных языковых средств, предоставляющих возможности включения в алгоритм логических операторов, условных конструкций, операторов цикла, позволяющих динамически выстраивать схему вычислений на основе анализа текущих данных.

4. Компонентно-ориентированный подход

В данном разделе рассматривается подход к описанию схем решения задач на основе компонентно-ориентированной парадигмы, в основе которой лежит объектно-ориентированный подход с определенными ограничениями (в частности, отсутствие механизмов наследования). СРЗ в данном случае описывает взаимодействие независимых объектов через строго определенный набор интерфейсов. Каждый объект обладает определенными свойствами и поведением. Свойства – это элементы внутренней структуры объекта. Поведение – это совокупность действий, выполняемых в качестве реакции на принимаемые объектом сообщения. Реакция на сообщение реализуется в виде метода и представляет собой программу на объектно-ориентированном языке, составленную из посылок сообщений другим объектам, в том числе создаваемым временно в рамках одного метода.

Для работы с основными объектами вычислительной модели был разработан специализированный интерпретируемый язык *DCScript*, являющийся надмножеством объектно-ориентированного скриптового языка программирования *JavaScript*. При описании схемы решения задачи на языке *DCSript* разработчику пакета доступны для использования все возможности объектно-ориентированного языка *JavaScript*, основные объекты вычислительной модели (параметры и модули), а также ряд специализированных средств для работы с ресурсами РВС. Остановимся более подробно на описании двух базовых объектов вычислительной модели: параметров и модулей.

Основными методами параметра являются: получение значения параметра; установка значения параметра; добавление обработчика события. К числу событий относятся: параметр вычислен, изменены размерности для параметра с типом *filelist*.

Основными методами модуля являются: проверка готовности к запуску (все ли входные параметры определены); запуск модуля; остановка модуля; получение информации о состоянии модуля (в очереди, выполняется, завершен, остановлен или аварийно завершен); добавление обработчика события. К числу событий относятся: готовность к запуску, запущен, завершен, остановлен, завершен с ошибкой. Кроме того, в виду асинхронного режима работы интерпретатора, объект «модуль» включает методы синхронизации вычислений (к примеру, ожидание завершения модуля). Более подробно и полно все основные свойства и методы объектов языка *DCScript* рассматриваются в работе [7].

Ниже представлены примеры использования описанных объектов в СРЗ, задаваемой на входном языке *DCScript*.

```
//Для инициализации объектов используются конструкторы:
//DiscompModule, DiscompParameter.
module = new DiscompModule("module_name");

//Доступ к свойствам и методам объекта осуществляется в
//соответствии с  синтаксисом языка Javascript.
value = parameter.getValue();

//Добавление обработчика события для объекта
module.addEventListener("onStop", "moduleStopped()");

//Изменение хода вычислений с помощью условного блока:
if (parameter.getValue() == "1") {
      module1.start();
} else {
      module2.start();
}

//Синхронизация (барьер):
module1.start();
module2.start();
module1.waitForFinished();
module2.waitForFinished();
module3.start();
```

В качестве отдельного ниже приводится реализация метода простых итераций на языке *DCScript*. В данном примере отсутствуют параллельные ветви вычислений в теле цикла в виду особенностей алгоритма. Основное внимание сосредоточено на описании возможностей операторов языка.

```
// переменные программы
var e = 0.0001;  //точность
var t = 1;       //текущая погрешность
var n = 0;       //количество проведенных итераций
var max_n = 100; //максимально допустимое число итераций
var x0 = 0;      //начальное значение x
var param_x = new DiscompParameter("x");//объект для работы с пар.
//модуль, вычисляющий значение функции
var module_func = new DiscompModule ("func");
//выполнять цикл до тех пор, пока не будет достигнута
//заданная точность или превышено число итераций
while ( t > e || n++ > max_n ) {
      //запоминаем начальное значение x
      x0 = param_x.getValue();
      //запускаем модуль и дожидаемся его завершения
      module_func.start();
      module_func.waitForWinished();
      //вычисляем погрешность текущего и предыдущего значения
      t = Math.abs (param_x.getValue() - x0);
};
//печатаем найденный корень
DiscompAPI.logMessage ("Найден корень: " + param_x.getValue() );
//рисуем график
var module_graph = new DiscompModule ("graph");
module_graph.start();
//параллельно запускаем модуль анализа
var module_analyse = new DiscompModule ("analyse");
module_analyse.start();
//ожидание завершения модулей
module_analyse.waitForFinished();
module_graph.waitForFinished();
```

Рассмотренные языковые средства предоставляют возможности для описания логически-сложных схем решения задач с использованием механизмов динамического управления вычислительным процессом. Данные возможности позволили расширить класс задач, решаемых с применением инструментального комплекса DISCOMP. Например, стало возможным описывать алгоритмы итерационных процессов, задач сдваивания, сложных многометодных задач, а также определенных задач, в которых область данных не допускает разбиения на независимые блоки. Приведенные в данном разделе языковые средства ИК DISCOMP успешно применялись при решении задач поиска простых чисел в РВС и при реализации РППП CARMA[8].

Однако, как показывает практика, использование таких средств является для разработчиков пакета весьма затруднительным и требует навыков объектно-ориентированного программирования. Кроме того описание схемы решения задачи на языке *DCScript* существенно усложняет структуру распределенной программы, делает практически невозможным ее графическое представление, усложняет проверку целостности программы и значительно затрудняет ее модификацию.

5. Сети Петри

В предыдущих разделах были показаны два различных подхода к описанию связанных распределенных вычислительных процессов. Подход, основанный на ярусно-параллельной форме, не позволяет описывать логически-сложные схемы вычислений. Компонентно-ориентированный подход имеет значительно более широкие возможности, однако в виду изна-

чальной ориентированности скриптового языка на последовательные вычисления существенно усложняется структура распределенной программы. В настоящее время для ИК DISCOMP разрабатывается новый подход к описанию связанных вычислений в РВС, основанный на аппарате сетей Петри [9].

Сети Петри – инструмент моделирования динамических дискретных систем, являющийся одним из наиболее адекватных способов описания асинхронного выполнения параллельных процессов, в том числе в распределенных системах [4]. Сети Петри работают в терминах условий и событий, где первым сопоставлены позиции (особые узлы – емкости для хранения фишек), а последним – переходы (особые узлы-действия, связанные ориентированными дугами с позициями и перемещающие фишки из входных позиций в выходные).

В нашем случае позициям соответствуют параметры, а переходам – модули предметной области. Наличие фишки в позиции отражает состояние соответствующего параметра (определен или не определен). Условием срабатывания перехода (выполнения модуля) является наличие во всех входных позициях определенного количества фишек, кратного количеству входных дуг, соединенных с данным переходом. При срабатывании перехода из каждой входной позиции изымается, а в каждую выходную позицию добавляется некоторое количество фишек, равное кратности соответствующих дуг, что порождает новую маркировку сети. Если одновременно соблюдаются условия для срабатывания нескольких переходов, то выполниться может любой из них или любая их комбинация. Последнее в свою очередь определяет асинхронную природу сетей Петри.

На рис. 2 проиллюстрирован пример реализации конструкции FORK/JOIN на основе графового представления сети Петри. При срабатывании перехода *Fork* создаются четыре дополнительные ветви, и выполняется перемещение фишки из позиции *p1* в позиции *p21*, *p22*, *p23*, *p24*. После чего соблюдаются условия для срабатывания переходов *Solver*. На рисунке изображена маркировка сети для случая, когда первый, второй и четвертый переходы *Solver* завершили свою работу, а третий переход еще выполняется. Переход *Join* осуществляет барьерную синхронизацию и последующее слияние всех ветвей вычислений в одну.

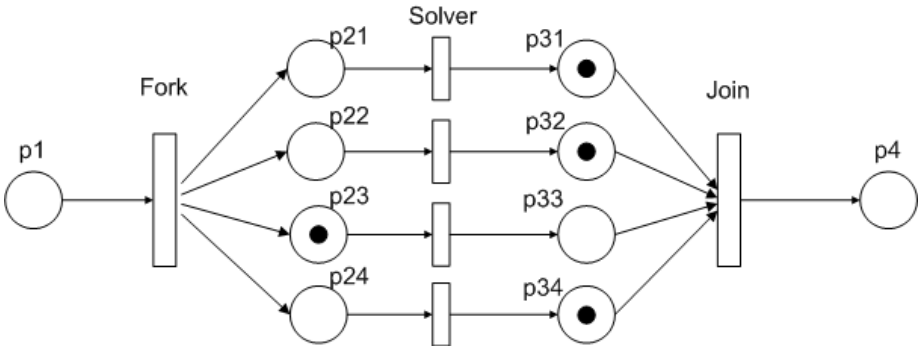

Рис. 2. Пример промежуточной маркировки сети Петри для конструкции FORK/JOIN

Однако стоит отметить, что применение сетей Петри в чистом виде является весьма затруднительным для описания вычислительных процессов и зачастую используются различные расширения в виде раскрашенных и иерархических сетей Петри. Первые позволяют более конкретно специфицировать условия срабатывания переходов, а вторые осуществлять иерархическую композицию или декомпозицию объектов сети. Было доказано, что сети Петри эквивалентны машине Тьюринга и составляют универсальную алгоритмическую систему [10]. Это дает право утверждать, что с использованием этого класса сетей Петри можно описывать достаточно сложные алгоритмы.

На рис. 3 изображена реализация метода простых итераций с использованием аппарата раскрашенных сетей Петри. Позиции n_{max}, n и e являются контролирующими и используются для срабатывания перехода *Check*. Двунаправленные дуги идущие от позиций к переходу *Check*, отражают сохранение фишек в этих позициях. Переход *Check* выполняет проверку количества допустимых итераций и сравнивает модуль разницы параметров x и x_0. В случае если превышено число допустимых итераций или найден корень, удовлетворяющий заданной погрешности, то осуществляется переход по дуге *true*. В противном случае осуществляется переход по дуге *false* и выполняется очередная итерация цикла.

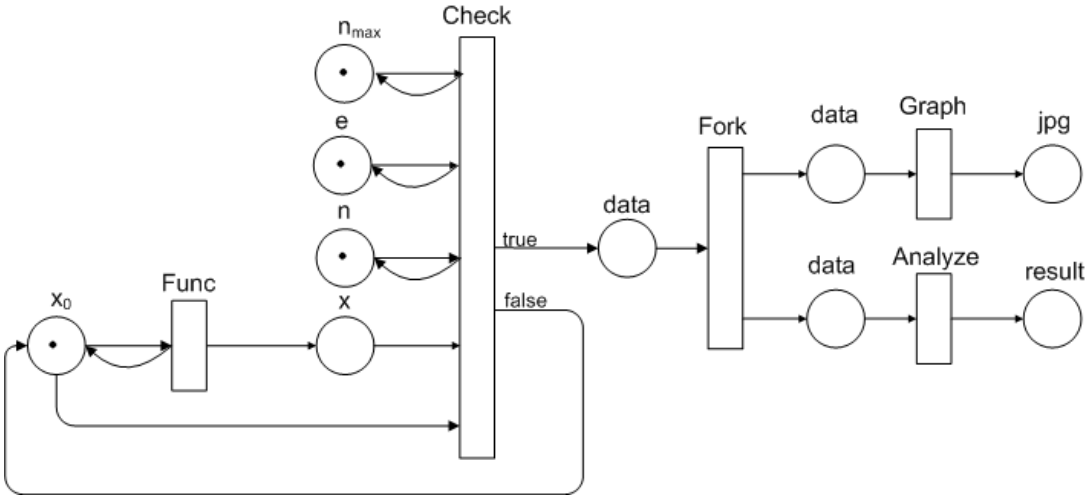

Рис. 3. Пример реализации метода простой итерации с применением раскрашенных сетей Петри

Для описания сети Петри в ИК DISCOMP используется модификация стандарта Petri Net Markup Language (PNML). PNML-описание передается в качестве входного параметра интерпретатору СРЗ, который в совокупности с описанием предметной области проверяет такие свойства как достижимость заданной разметки, живость переходов, ограниченность, безопасность. Для визуализации сети Петри, описанной с помощью PNML, планируется разработать графический редактор, который будет предоставлять возможности редактирования, моделирования и анализа цветных сетей Петри. Кроме того в составе данного графического редактора планируется реализовать средства для интерактивного управления ходом вычислительного процесса.

С использованием описанного в данном разделе подхода планируется реализовать РППП, предназначенный для решения задач бионформатики. В этой предметной области накоплено большое количество разрозненных программных продуктов, которые используются предметными специалистами на различных этапах вычислительного эксперимента. Проектируемые схемы решения задач для таких пакетов требуют развитых языковых средств в составе инструментального комплекса. Аппарат сетей Петри с нашей точки зрения является наиболее приемлемым подходом для описания схем такой сложности.

6. Вычислительный эксперимент

Одним из примеров применения рассмотренных в работе языковых средств является РППП CARMA, предназначенный для таксономической и функциональной классификации коротких метагеномных последовательностей. Пакет позволяет задавать управляющие параметры, влияющие на время вычисления, точность предсказания происхождения того или иного фрагмента последовательности и пр. Полное описание всех реализованных схем решения задач для данного пакета рассматривается в работах [7, 8].

В таблице 1 представлены результаты вычислительного эксперимента для двух наборов, состоящих из 40 и 80 файлов по 100 последовательностей. Вычисления проводились по последовательной схеме на 1 вычислительном ядре, по параллельной синхронной схеме (40 ядер), реализованной на основе параллельно-ярусной формы, и по асинхронной параллельной схеме (40 ядер), реализованной с использованием компонентно-ориентированного языка *DCScript*.

Таблица 1. Результаты вычислительного эксперимента

Кол-во последовательностей в файле x кол-во файлов	**Последовательная схема (1 выч. ядро)** (ЧЧ:ММ:СС)	**Синхронная параллельная схема (40 выч. ядер)** (ЧЧ:ММ:СС)	**Асинхронная параллельная схема (40 выч. ядер)** (ЧЧ:ММ:СС)
100 x 40	53:30:43	4:55:01	3:27:14
100 x 80	97:15:13	5:32:47	4:40:39

7. Заключение

На сегодняшний день с использованием подходов к описанию СРЗ в виде ярусно-параллельной формы и в виде программы на компонентно-ориентированном языке реализовано семь РППП для решения задач из различных предметных областей. Разрабатываемый подход на основе аппарата сетей Петри позволит существенно расширить класс задач, решаемых с использованием ИК DISCOMP, повысить надежность и эффективность вычислений.

Все представленные в данной работе подходы к организации вычислений в РППП допускают естественное развитие и обобщение базовых методов параллельных и распределенных вычислений для решения новых классов фундаментальных и прикладных исследовательских задач.

Литература:

1. Сидоров И.А., Опарин Г.А., Феоктистов А.Г. Разработка и применение распределенных пакетов прикладных программ // Программные продукты и системы, 2010. – № 2. – С. 108-111.
2. Тыугу Э.Х. Концептуальное программирование. – М.: Наука, 1984. – 256 с.
3. Опарин Г.А., Новопашин А.П. Булево моделирование планирования действий в распределенных вычислительных системах // Теория и системы управления, 2004. – № 5. – С.105-108.
4. Федотов И. Е. Некоторые приемы параллельного программирования: Учебное пособие. – М.: Изд-во МГИРЭА(ТУ), 2008. – 188 с.
5. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. – Новосибирск, 2006. – 296 с.
6. Заикин О.С., Семенов А.А., Сидоров И.А., Феоктистов А.Г. Параллельная технология решения SAT-задач с применением пакета прикладных программ D-SAT // Вестник ТГУ. – 2007. – № 23. – С. 83-95.
7. Сидоров И.А. Сластной К.А. Разработка объектно-ориентированных языковых средств распределенного программирования // Винеровские чтения: Материалы IV Всерос. конф. – Иркутск: Изд-во ИрГТУ, 2011. – С. 35-42.
8. Поздняк Е.И., Сидоров И.А., Галачьянц Ю.П. Генерализация алгоритма таксономического классификатора CARMA // Вестник ИрГТУ. – 2011. – № 9. – С. 11-15.
9. Котов В.Е. Сети Петри. – М.: Наука, 1984. – 160 с.
10. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации. – СПб.: Наука, 2000. – 780 с.