На правах рукописи

Редькин Андрей Владимирович

МЕТОДЫ АСИНХРОННОГО УПРАВЛЕНИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫМИ ПАРАЛЛЕЛЬНЫМИ ВЫЧИСЛЕНИЯМИ

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Красноярск - 2009

Работа выполнена в Федеральном государственном образовательном учреждении высшего профессионального образования «Сибирский федеральный университет», г.Красноярск.

Научный руководитель: доктор технических наук, профессор
Легалов Александр Иванович

Официальные оппоненты: доктор технических наук, профессор
Рубан Анатолий Иванович

доктор технических наук, профессор
Опарин Геннадий Анатольевич

Ведущая организация: Институт систем информатики имени А.П. Ершова СО РАН

Защита состоится 26 июня 2009 года в 14 часов на заседании диссертационного совета ДМ 212.099.05 при Сибирском федеральном университете по адресу: г. Красноярск, ул. Киренского, 26, ауд. УЛК-115.

С диссертацией можно ознакомиться в научной библиотеке по техническим наукам Сибирского федерального университета.

Автореферат разослан 26 мая 2009 г.

Ученый секретарь диссертационного совета Вейсов Е.А.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность проблемы. Использование параллельных вычислений в настоящее время приобретает массовый характер. С одной стороны это связано с применением многоядерных процессоров, что ведет к параллелизму в персональных компьютерах. С другой, постоянно пополняется парк высокопроизводительных вычислительных систем для научных и инженерных расчетов. Разнообразие архитектур параллельных вычислительных систем и наличие множества альтернативных методов построения параллельных программ порождают различные подходы и парадигмы в параллельном программировании.

На сегодняшний день получили широкое распространение несколько, принципиально отличающихся друг от друга, систем параллельного программирования. Например, для многоядерных вычислительных систем (ВС) и параллельных ВС с общей памятью используются многопоточные библиотеки и система программирования OpenMP. При разработке программ для систем с распределенной памятью наибольшую популярность получил стандарт MPI. Следует отметить, что программы, созданные с применением этих средств, эффективно выполняются только на соответствующих им ВС. При переносе приложения на параллельную ВС с другой архитектурой его, как правило, приходится переписывать заново. Архитектурная зависимость параллельных программ порождается не только различиями в организации ВС. Даже в рамках одной архитектуры перенос программы с одной системы на другую может привести к разбалансировке вычислений из-за отличий в скоростных соотношениях процессоров и каналов передачи данных. Это ведет к использованию методов архитектурно-независимого параллельного программирования.

Методы архитектурно-независимого параллельного программирования основаны на разделении на отдельные этапы процессов формирования логики программы и ее архитектурной привязки, что позволяет не решать эти две задачи одновременно. При таком подходе программа может разрабатываться так, как будто она использует неограниченные вычислительные ресурсы. В этом случае она может служить в качестве базовой спецификации, обеспечивающей более быстрое решение вопросов, связанных с верификацией и отладкой, по сравнению с традиционным подходом.

Методы архитектурно-независимого параллельного программирования опираются на использование функциональных и потоковых языков, описывающих только информационный граф программы без привязки к системе исполнения. К таким языкам относятся Пифагор, Sisal, FPTL и ряд других. Управление вычислениями в этих языках, как правило, задается неявно, что позволяет использовать различные подходы к его реализации.

Следует отметить, что возможности функциональных языков и методов управления в них параллельными вычислениями недостаточно исследованы. Неизвестно, каким образом, набор и семантика операторов таких языков влияет на их возможности описывать различные параллельные алгоритмы, а также как должно осуществляться управление вычислениями на уровне виртуальных и реальных параллельных ВС. В связи с этим, актуальной является задача иссле-

дования и разработки новых методов функционально-потокового параллельного программирования и механизмов управления функционально-потоковыми параллельными вычислениями.

Цель работы: исследование и разработка новых методов функционально-потокового параллельного программирования и его инструментальной поддержки.

Для достижения указанной цели в работе решаются следующие задачи.

1 Разработка новых методов организации асинхронных вычислений по готовности данных и исследование их влияния на организацию функционально-потоковых параллельных программ.
2 Исследование принципов построения управляющих структур функционально-потоковых параллельных программ во время трансляции. Разработка методов построения управляющих структур программ во время трансляции, позволяющих изменять стратегию управления вычислениями.
3 Создание инструментальных средств обеспечивающих поддержку функционально-потокового параллельного программирования.

Методы исследования. В диссертационной работе использовались методы и понятия теории графов, теории алгоритмов, элементы теории множеств, теории языков и формальных грамматик. Для описания синтаксиса языка программирования использовались расширенные формы Бэкуса-Наура (РБНФ), диаграммы Вирта. В экспериментальной части работы применялись методы синтаксического анализа и компиляции, методы структурного и объектно-ориентированного программирования.

Научная новизна и положения, выносимые на защиту.

1 На основе анализа подходов к управлению вычислениями по готовности данных предложен метод управления данными с использованием асинхронных списков, обеспечивающий динамическую трансформацию параллелизма программы в зависимости от соотношений между временами выполнения операций и передачи данных между операциями. Это позволяет повысить гибкость функционально-потокового параллельного программирования и ведет к новому классу алгоритмов с неявным изменением параллелизма решаемой задачи.
2 Предложен метод трансляции функционально-потоковых параллельных программ, представленных в виде совокупности информационного графа и накладываемого на него графа управления, обеспечивающий гибкое изменение стратегий управления параллельными вычислениями.
3 Разработан событийный процессор, обеспечивающий повышение эффективности асинхронного управления выполнением функционально-потоковых параллельных программ за счет обработки событий, задаваемых управляющим графом.
4 Предложена архитектура инструментальной системы, обеспечивающей трансляцию и архитектурно-независимую отладку функционально-потоковых параллельных программ.

Практическая ценность работы.

1 На основе разработанной архитектуры реализована инструментальная система функционально-потокового параллельного программирования, обеспечивающая трансляцию, отладку и выполнение функционально-потоковых параллельных программ с применением предложенных подходов к организации функционально-потоковых параллельных вычислений.

2 Полученные научные и практические результаты использованы в учебном процессе по дисциплинам «Технология программирования» и «Модели параллельных вычислений» в ФГОУ ВПО «Сибирский федеральный университет».

Достоверность научных положений и выводов, полученных в диссертации, подтверждается исследованием методов разработки функциональных и потоковых параллельных программ, анализом существующих языковых и инструментальных средств, используемых для разработки параллельного программного обеспечения, корректным обоснованием постановок задач, точной формулировкой критериев, исследованием и сравнительным анализом существующих подходов к решению поставленной задачи.

Апробация диссертации. Основные положения диссертации докладывались и обсуждались на следующих конференциях и семинарах:

- третья Сибирская школа-семинар по параллельным вычислениям, Томск, 2006;
- шестая межрегиональная школа-семинар «Распределенные и кластерные вычисления», Красноярск, 2006;
- четвертая Российско-германская школа по параллельным вычислениям, Новосибирск, 2007;
- четвертая Сибирская школа-семинар по параллельным вычислениям, Томск, 2007;
- Всероссийские научно-технические конференции «Молодежь и наука — XXI век», Красноярск, 2005-2008;
- третья международная конференция «Параллельные вычисления и задачи управления» PACO '2006, Москва;
- международная научная конференция Параллельные вычислительные технологии (ПаВТ'2009), Нижний Новгород.

Сведения о внедрении. Результаты работы использованы при выполнении: научно-методического проекта Сибирского Федерального университета №10 «Решение некоторых задач прикладной математики и информатики для повышения потенциала образовательного процесса»; проекта Сибирского Федерального университета «Использование технологий параллельной обработки в научной, образовательной и организационной деятельности»; проекта №25 «Среда разработки для языка параллельного программирования Пифагор» в рамках «Программы развития СФУ на 2007–2010 годы».

Программный комплекс «Инструментальная система поддержки программирования на языке Пифагор с возможностями визуализации» используется в учебном процессе, что подтверждено актами внедрения.

Публикации. По теме диссертации опубликовано девять научных работ, из которых одна статья в издании, рекомендуемом ВАК.

Личный вклад автора. Автору принадлежат идеи работы, определение цели и постановка задачи, методы и способы реализации разработанных языковых конструкций. Им предложены новые виды программных объектов, поддерживающих функционально-потоковое параллельное программирование. Основные результаты, изложенные в работе, получены либо непосредственно автором, либо с его участием.

Структура диссертации. Диссертация состоит из введения, четырех глав, заключения и трех приложений. Робота содержит 129 страниц основного текста, 65 рисунков и 6 таблиц. Список литературы содержит 110 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении представлены цели и задачи исследования, раскрывается актуальность, новизна полученных результатов, практическая значимость. Приводится перечень вопросов, выносимых на защиту.

В первой главе приведен обзор моделей параллельных вычислений, в ходе которого анализируются реализованные в них механизмы управления асинхронными вычислениями. Предложен механизм использования асинхронных списков в функционально-потоковом параллельном программировании.

Существующие языки и модели описания асинхронных и параллельных вычислений, такие как FIFO-сети, сети потоков данных и другие позволяют учитывать асинхронное поступление данных, но в них отсутствуют возможности динамического изменения параллелизма алгоритма в зависимости от имеющихся ресурсов, что не позволяет писать переносимые параллельные программы, работающие с асинхронно поступающими данными. Наибольшая гибкость может быть достигнута при программировании с применением многопоточных библиотек, однако их использование предполагает явное управление вычислениями, что значительно усложняет программирование, ведет к большим затратам на написание программ. Использование данных библиотек не обеспечивает автоматического распределения ресурсов. Для более эффективной организации вычислений программист должен явно планировать потоки и писать дополнительный код для управления вычислениями. Стремление повысить эффективность такого кода обычно ведет к построению архитектурно-зависимых параллельных программ.

В отличие от рассмотренных решений в работе предлагается расширить модель функционально-потоковых параллельных вычислений новой операцией группировки в асинхронный список. Это обеспечивает не только обработку асинхронно поступающих данных, но и позволяет преодолеть некоторые внутренние ограничения модели, связанные с формированием конкурирующих

процессов. Ранее используемый в ней принцип синхронизации только по готовности данных снижает эффективность вычислений из-за необходимости ожидания получения всех элементов при формировании списка аргументов.

Применение асинхронных списков ведет к созданию программ нового типа, алгоритм работы которых зависит от соотношений между временем выполнения операций и передачи данных между функциями. Оператор группировки в асинхронный список обеспечивает упорядочение данных в соответствии с последовательностью их появления. Каждая из величин этого списка генерирует сигнал готовности, который может обрабатываться операцией интерпретации.

Обозначим асинхронный список через A^N, где N – количество порождаемых в нем элементов. Тогда ее структуру можно описать следующим рекурсивным выражением:

$$A^0 = (.), \qquad \text{если } N = 0$$
$$A^N = (d, A^{N-1}), \qquad \text{если } N > 0.$$

где d – головной элемент порождаемого списка, A^{N-1} – асинхронный список из оставшихся $N-1$ элементов, A^0 – пустой асинхронный список, $(.)$ – пустой список данных.

Таким образом, асинхронный список может рассматриваться как головной элемент и хвост. Головным является значение, полученное первым. Хвост является асинхронным списком, содержащим оставшиеся элементы, которые могут оказаться еще не сформированными к моменту появления первого значения. Однако появление головного элемента предполагает одновременное формирование и хвоста. Это позволяет трактовать один шаг вычисления асинхронного списка как порождение списка данных, состоящего и головного элемента и хвоста, являющегося асинхронным списком.

Показано, что использование асинхронных списков обеспечивает построение алгоритмов, параллелизм которых изменяется динамически в зависимости от соотношения времен выполнения операций и передачи данных между операциями. Таким образом, применение асинхронных списков позволяет отказаться от создания нескольких альтернативных алгоритмов, каждый из которых рассчитан на применение в зависимости от используемой в данный момент архитектуры ВС.

Наряду с динамическим изменением параллелизма внутри выполняемой функции асинхронные списки поддерживают взаимодействие нескольких разных функций, что еще больше повышает возможности динамического изменения параллелизма. Эту особенность функционально-потоковых параллельных программ можно рассмотреть на примере перемножения двух векторов:

$$c = A \times B = \sum_{i=1}^{N} a_i \times b_i$$

При традиционной реализации алгоритма суммирование выполняется только после выполнения функции попарного произведения двух исходных

векторов. Соответствующая программа на языке Пифагор выглядит следующим образом:

```
//суммирование элементов вектора
S_BinTreeSum << funcdef A {
  Len << A:|;
  return<< .^[((Len,2):[<,=,>]):?]^
  (   {A:[]},  {A:+},
      {  block {
           OddVec  << A:[(1,Len,2):..];
           EvenVec << A:[(2,Len,2):..];
           ([OddVec,EvenVec]: S_BinTreeSum):+ >>break;
         }
      }
  )
}
//перемножение пар из векторов
S_PairProduct << funcdef A {
  V1<< A:1; V2<< A:2;  return<<(V1,V2):#:[]:*:(.);
}
//скалярное перемножение векторов
S_VecScalProd << funcdef A{
  return<< A:S_PairProduct:S_BinTreeSum;
}
```

Временные соотношения между выполняемыми функциями представлены на рисунке 1.

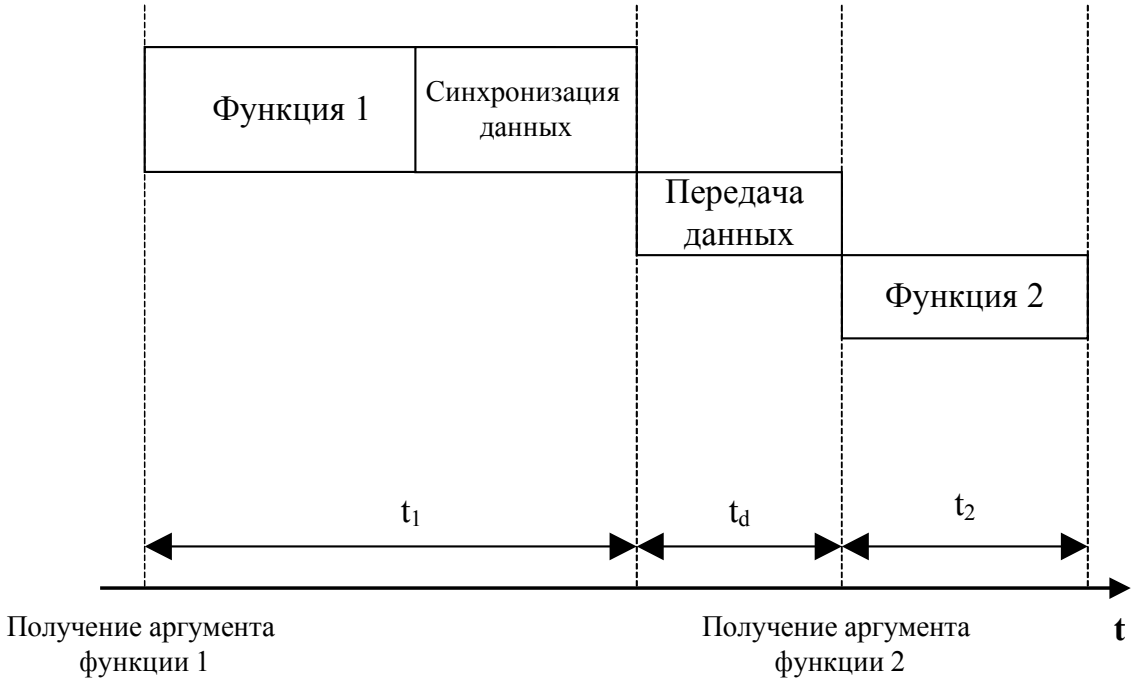


Рисунок 1 - Синхронное взаимодействие функций через список данных

Даже при частичной готовности результатов первой функции выполнение второй функции невозможно, что ограничивает скорость вычислений и парал-

лелизм. Если время выполнения первой функции t_1 , второй - t_2 , и время затрачиваемое на передачу данных - t_d , то общее время выполнения двух функций будет равно сумме временных затрат: $t_{общ}=t_1+t_2+t_d$.

Применение асинхронных списков позволяет реализовать программу с использованием следующих функций:

```
// Функция, возвращающая сумму элементов
// асинхронного списка
A_AsyncSum<< funcdef A {
// Формат аргумента: A=asynch(x1, x2, … , xn)
// где x1, x2, … , xn - числа
  x1<< A:1;     // Поступивший головной элемент данных
  tail_1<< A:-1; // Хвост асинхронного списка
  // Проверка на пустоту остатка списка
  [(tail_1:[IsEmptyDataList, IsNotEmptyDataList]:?]^
  (
     x1, // В списке, только один элемент,
         // определяющий сумму
     {   // Выделение второго аргумента
         // с последующим суммированием
        block {
           x2<< tail_1:1;      // второй аргумент
           s<< (x1,x2):+;         // сумма двух элементов
           tail_2<< tail_1:-1;  // "хвост" от "хвоста"
           // Рекурсивная обработка оставшихся элементов
           [(tail_2:[IsEmptyDataList,
                     IsNotEmptyDataList]:?]^
           (
              s,
              { asynch( tail_2:[], s):A_AsyncSum }
           ):. >>break
        }
     }
  ):. >>return;
}
//перемножение пар из векторов
A_PairProduct<<funcdef A{
  V1<<A:1;
  V2<<A:2;
  return<<(V1,V2):#:[]:*:asynch;
}
//скалярное перемножение векторов
A_VecScalProd<<funcdef A{
  return<<A:A_PairProduct:A_AsyncSum;
}
```

Функция перемножения A_PairProduct обеспечивает формирование на выходе асинхронного списка, элементы которого, по мере появления, поступают в функцию суммирования A_AsyncSum (рисунок 2).

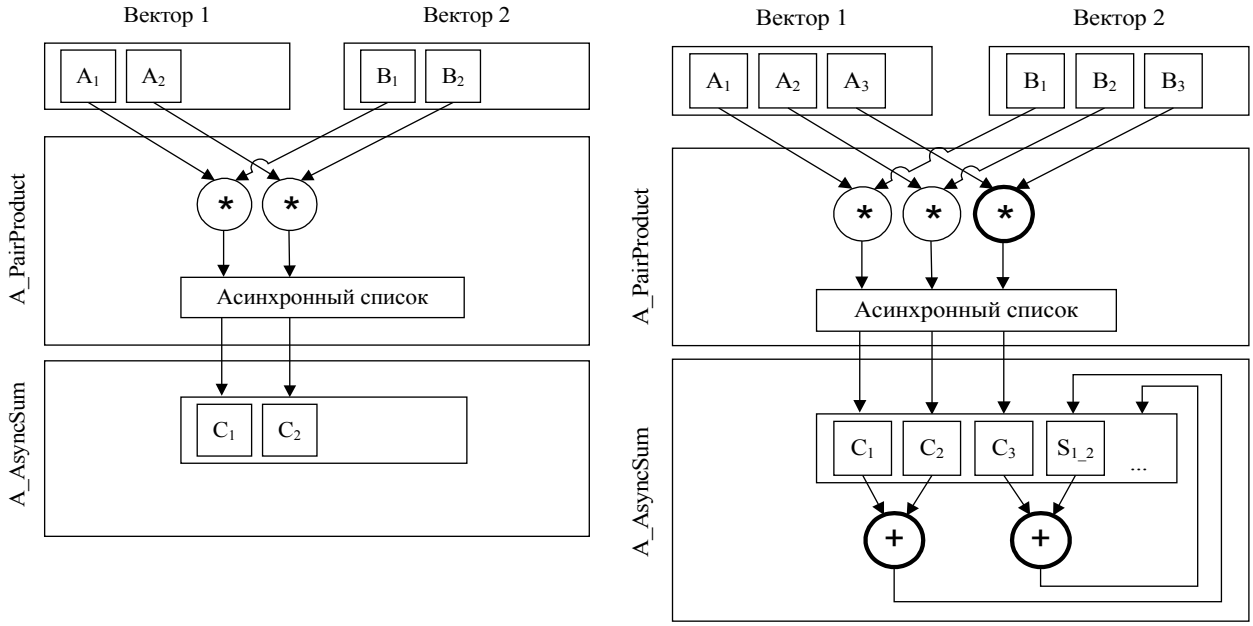


Рисунок 2 - Пример вычислений с применением асинхронных списков

Это ведет к совмещению во времени выполнения функций, обеспечивающему эффект конвейеризации. Уровень совмещения зависит от временных соотношений между операциями обработки и передачи данных. Диаграмма, показывающая совмещение выполнения функций при перемножении векторов представлена на рисунке 3.

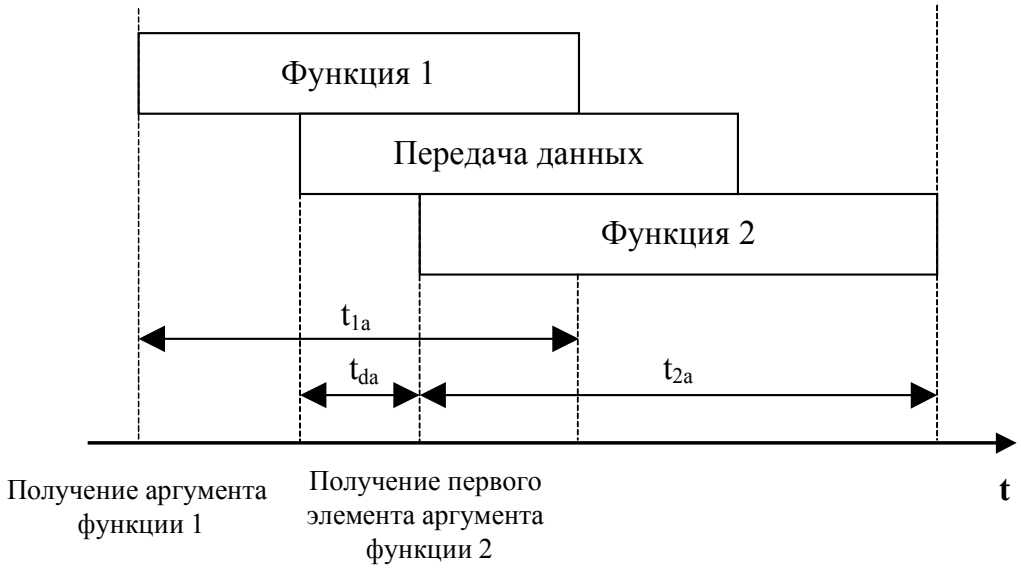


Рисунок 3 - Конвейерное взаимодействие функций при использовании асинхронного списка

Следует отметить, что конвейерное выполнение двух и более функций при помощи асинхронных списков осуществляется без дополнительных затрат на программирование самого процесса конвейерной обработки.

Для анализа возможностей использования асинхронных списков была проведена разработка различных алгоритмов. В диссертации представлены некоторые из алгоритмов сортировки данных. Для каждого варианта реализации сортировки приведены теоретические оценки максимального параллелизма и влияния задержек поступления данных на время работы программы. Показано что при использовании асинхронных списков осуществляется динамическое изменение параллелизма алгоритма решения задачи.

На основе анализа функционально-потоковых параллельных программ предложены дополнительные операции, расширяющие язык функционально-потокового параллельного программирования. Это ряд встроенных функций для работы со списками (insert, exchange, append, permute, drop, take) и операция «прямой интерпретации», позволяющая производить не доступные ранее действия с параллельными списками. Предложенные языковые средства позволяют создавать архитектурно-независимые параллельные программы с применением более мощных конструкций.

Во второй главе предлагается подход к организации асинхронного управления функционально-потоковыми параллельными вычислениями основанный на разделении информационного и управляющего графов.

Существуют различные варианты перехода от программ, написанных на функциональных и потоковых языках параллельного программирования к исполняемым программам. Это может быть как компиляция непосредственно в машинный код, так и трансляция в различные формы промежуточного представления. Например, для языка Sisal существуют трансляторы, обеспечивающие на выходе формирование промежуточного представления с возможностью последующей компиляции для различных архитектур параллельных ВС. Язык программирования FPTL непосредственно интерпретируется на параллельных ВС с использованием редукции. Подобный способ интерпретации неэффективен по затратам вычислительных ресурсов поскольку предполагает дополнительный анализ при редукции и выполнении вычислений на обратном ходе (свертке).

Отсутствие механизмов прямой интерпретации для функционально-потоковых языков требует поиска новых механизмов организации вычислений обеспечивающих более гибкие и эффективные подходы к управлению вычислениями.

В функционально-потоковой модели параллельных вычислений основным принципом управления является срабатывание операторов по готовности данных. Управляющий граф при таком управлении можно считать заданным неявно, что позволяет использовать редукционные методы для интерпретации программ. Тем не менее, использование управляющего графа в явном виде позволяет более детально описывать и изменять управление вычислениями, что открывает новые возможности для разработки программ и систем, обеспечивающих их выполнение.

Предложен новый формат информационного графа программы, который формируется в ходе трансляции. Управляющие графы программы могут порождаться в соответствии с различными правилами, определяющими, в конечном счете, стратегию управления вычислениями. Они могут генерироваться динамически в ходе вычислений, или порождаться статически по информационному графу программы.

В связи с этим, предлагается осуществлять трансляцию программы с формированием информационного и управляющего графа в два этапа. Схематично этапы трансляции показаны на рисунке 4.

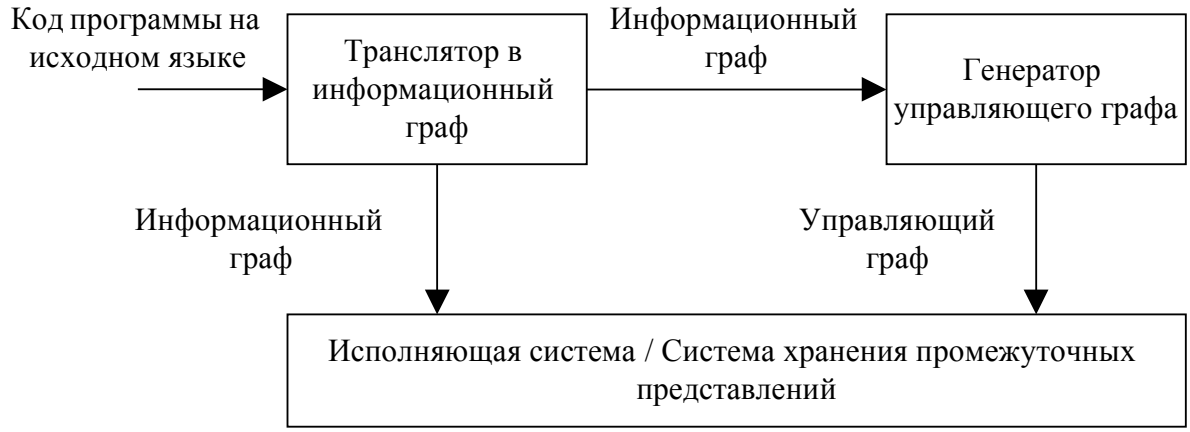
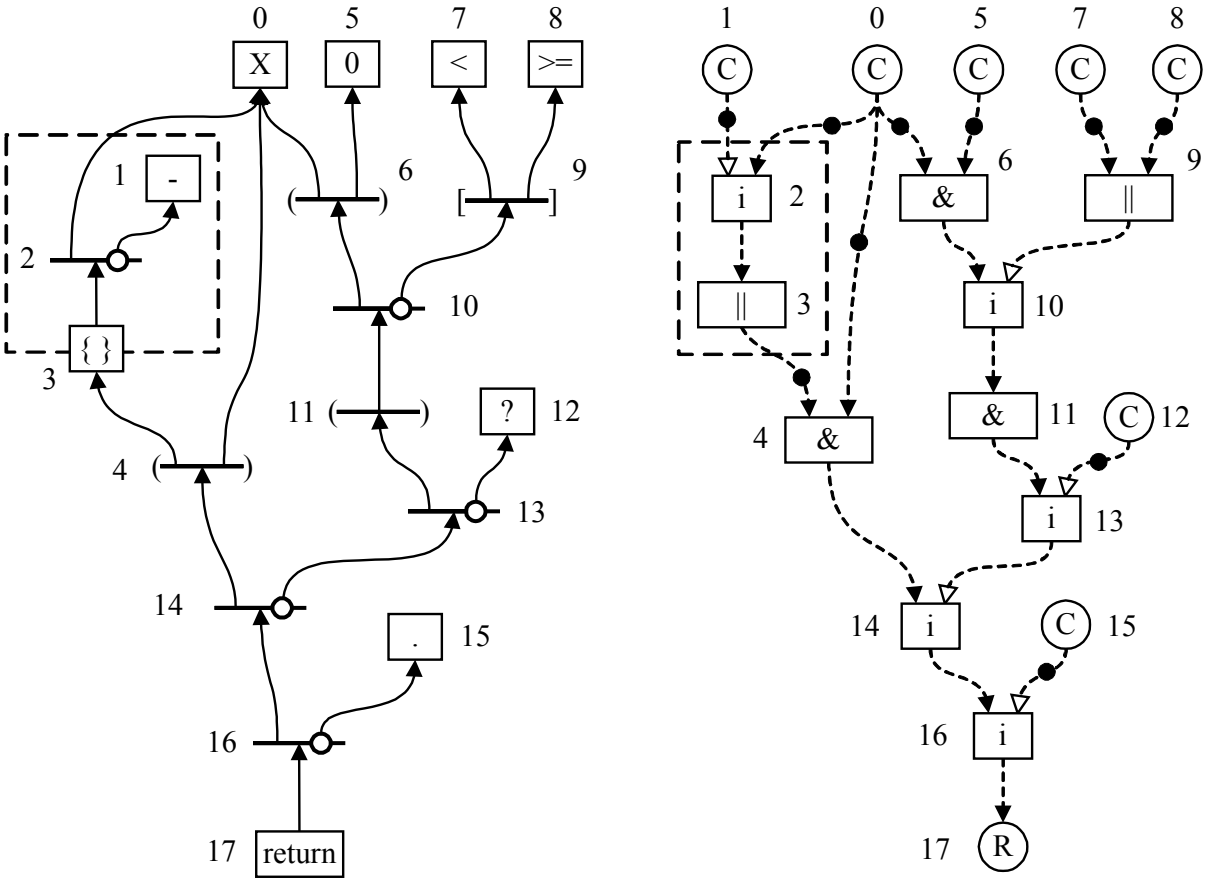


Рисунок 4 - Трансляция функционально-потоковых программ

В отличие от информационного графа функционально-потоковой модели параллельных вычислений, информационный граф, получаемый при трансляции, ориентирован на отражение зависимостей между операторными вершинами по чтению данных. Он является ациклическим ориентированным графом, вершины которого представляют операторы, а дуги - информационные связи между ними. Дуги направлены к вершинам - источникам информации. На рисунке 5а представлен информационный граф функции, вычисляющей абсолютную величину числа.

Управляющий граф функционально-потоковой параллельной программы, как и информационный, является ациклическим орграфом, вершинами которого определяют правила преобразования сигналов, а дуги указывают направления их передачи. Входящие дуги каждой вершины строго упорядочены, кроме того они могут быть многократными. Все выходящие дуги вершины имеют одинаковую кратность. Один из вариантов управляющего графа для заданного информационного представлен на рисунке 5б. Сама функция имеет следующий вид:

```
Abs<< funcdef X {
      [((X,0):[<,>=]):?]^
      ({X:-}, X):. >>return
}
```



а) информационный граф

б) управляющий граф

Рисунок 5 - Пример информационного и управляющего графа функции вычисления абсолютной величины числа

Управление вычислениями осуществляется посредством вызова вершин информационного графа из вершин управляющего графа. Такая связь обеспечивает активацию вычислений (рисунок 6).

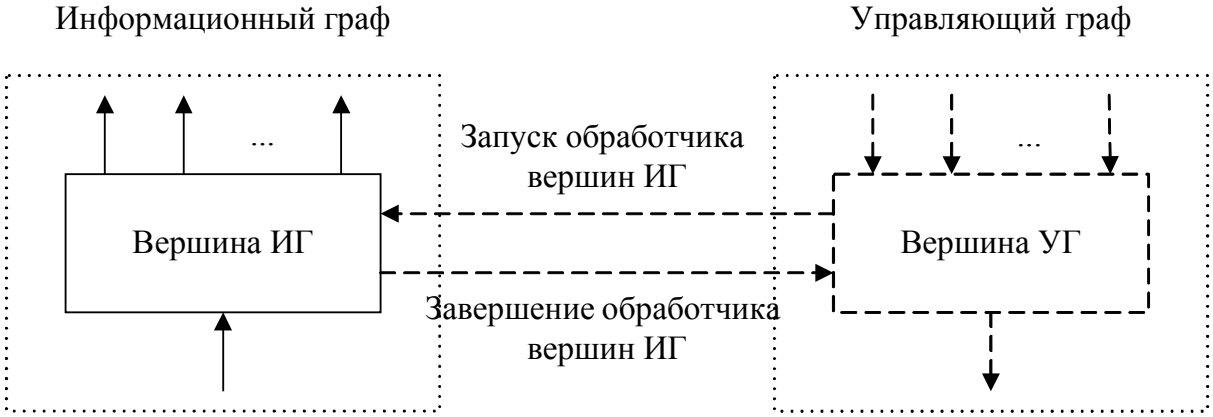


Рисунок 6 - Связь вершин информационного и управляющего графов

Управляющий граф может быть подвергнут модификациям во время трансляции, что позволяет оптимизировать стратегию управления вычислениями, за счет уменьшения числа операций и количества управляющих связей. В работе предложены обобщенные алгоритмы, обеспечивающие такую трансформацию. Для информационного и управляющего графов разработаны внутренние представления, используемые в системах трансляции и исполнения программ.

В третьей главе предложена архитектура событийного процессора исполняющего функционально-потоковые параллельные программы на архитектурно-независимом уровне.

В главе рассматривается организация системы программирования, обеспечивающей трансляцию и выполнение функционально-потоковых параллельных программ, написанных на языке «Пифагор» в виртуальной системе, названной событийным процессором. Событийный процессор обеспечивает обход управляющего графа, построенного по информационному графу программы. Разработка такого процессора поддерживает архитектурно независимую отладку параллельных программ, а отделение информационного графа от управляющего дает возможность повышения эффективности управления вычислениями за счет изменения управляющего графа после выполнения такой отладки.

Событийный процессор обеспечивает исполнение функционально-потоковой параллельной программы посредством механизмов обработки сигналов готовности данных - событий. Исходными данными, обеспечивающими его функционирование являются информационный граф (ИГ) и управляющий граф (УГ) программы. Функциональная схема событийного процессора приведена на рисунке 7.

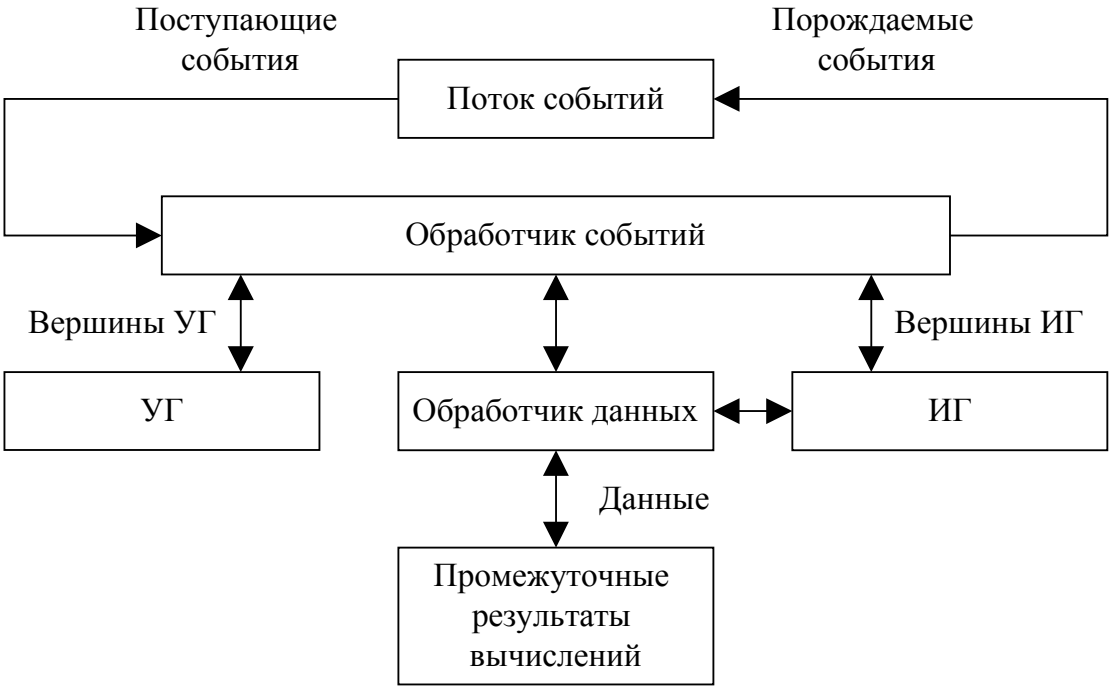


Рисунок 7 - Функциональная схема событийного процессора

Управляющий граф размещается в соответствующих внутренних структурах и используется событийным процессором для организации вычислений. Сигналы, определяющие начальную разметку управляющего графа, задают события, связанные с наличием данных, формируемых в ходе вычислений. Получение события активизирует управляющий узел, переводя его в новое состояние, определяемое как функцией связанного с ним узла информационного графа, так и своим текущим состоянием.

Функционирование событийного процессора осуществляется следующим образом. Исходные события, определяемые начальной разметкой управляющего графа, выставляются в поток событий, из которого они передаются обработчику событий в соответствии с дисциплиной обслуживания. В простейшем случае это может быть очередь. Обработчик событий выбирает указанный узел и, на основе анализа его состояния, может обратиться к информационному графу за кодом выполняемой операции. В случае, когда операция обработки данных должна быть выполнена, происходит обращение к обработчику данных, который осуществляет требуемые функциональные преобразования и сохраняет промежуточные результаты. После обработки данных управляющий узел переходит в новое состояние и при необходимости формирует новое событие, поступающее в поток.

Событийный процессор является частью разработанного комплекса инструментальных средств. Представлена экспериментальная реализация событийного процессора, позволяющая исполнять и отлаживать функционально-потоковые параллельные программы без их привязки к конкретным вычислительным ресурсам. Предложены варианты создания параллельных систем на основе использования нескольких событийных процессоров и дублирования его внутренних модулей.

В четвертой главе предложен комплекс инструментальных средств для поддержки функционально-потокового параллельного программирования. На основе разделения информационного и управляющего графов программы разработана система выполнения и отладки программ с использованием интерпретатора, порождающего множество событийных процессоров. Структура разработанной системы инструментальной поддержки представлена на рисунке 8.

Транслятор на входе получает исходные тексты функций, которые преобразуются во внутреннее представление и передаются в хранилище информационных графов. Внутренне представление каждой функции, помимо информационного графа, содержит информацию о связях с другими функциями и данные для отладки(в зависимости от исходного языка). Такая схема позволяет использовать различные входные языки пользователя, например, текстовый и графический, преобразуя их в одно и то же представление информационного графа.

Выделение генератора управляющих графов в отдельный этап обработки позволяет использовать различные стратегии управления вычислениями. Созданные управляющие графы поступают в хранилище вместе с информацион-

ными, поскольку их построение при каждом вызове функции или модуля нецелесообразно.

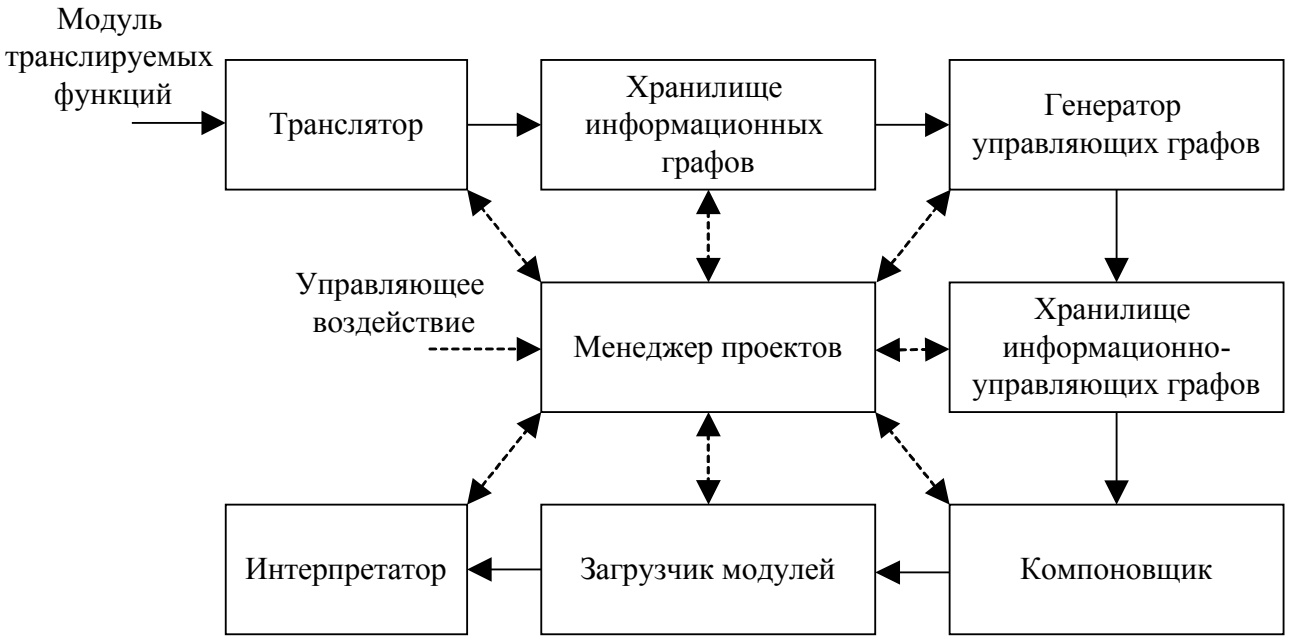


Рисунок 8 - Структура системы инструментальной поддержки

Компоновщик обеспечивает сборку связанных функций в единый модуль, который поступает в интерпретатор через загрузчик модулей. Загрузчик модулей оптимизирует процесс вызова функции в том случае, если они уже были обработаны ранее. Это дает возможность, с одной стороны, исключить повторную компоновку функций, а с другой - динамически производить перекомпоновку функций при их изменении. Такие возможности позволяют гибко производить обновление системы.

Интерпретатор в ходе работы динамически создает событийные процессоры для вызываемых функций. То есть, он обеспечивает вызов одних функций другими. Интерпретатор предусматривает два режима работы: обычный и отладочный. В отладочном режиме на его вход поступают не только информационные и управляющие графы, но и дополнительная информация, которая позволяет установить соответствие вершин и связей информационного и управляющего графов с участками программы на исходном языке. Это дает возможность вести отладку программы с использованием исходного текста.

Система реализована в виде отдельных исполняемых модулей для каждого компонента, осуществляющих обмен информацией на уровне системы хранения данных. Разработанная система дает возможность не только модифицировать каждый из компонентов по отдельности, но и включать дополнительные звенья в процесс трансляции и выполнения без необходимости модификации уже существующих программных объектов.

Разработан пользовательский интерфейс, позволяющий работать с хранилищем функций, производить запуск модульных программ с возможностями

событийной отладки и визуального представления информационного и управляющего графов программы

В приложении А перечислены синтаксические и семантические изменения языка Пифагор предлагаемые в данной работе.

В приложении Б описаны внутренние структуры представлений, используемые в системе инструментальной поддержки функционально-потокового параллельного программирования.

В приложении В приведены основные функции библиотеки для работы с внутренним представлением информационных и управляющих графов.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

1 Проведен обзор методов управления асинхронными вычислениями по готовности данных, на основе которого предложен метод управления данными с использованием асинхронных списков, позволяющий динамически трансформировать параллелизм программы в зависимости от соотношений между временами выполнения операций и передачи данных между операциями. В перспективе это позволяет создать новые методы планирования параллельных вычислений, использующих динамическое изменение параллелизма в зависимости от характеристик существующих вычислительных ресурсов.

2 Проанализированы особенности функционально-потоковой модели параллельных вычислений и существующих систем трансляции, позволившие сформировать новый метод трансляции функционально-потоковых параллельных программ в информационный граф, на который могут накладываться различные управляющие графы. Это позволяет гибко изменять стратегии управления вычислениями в зависимости от особенностей исходной программы и архитектуры используемой вычислительной системы.

3 На основе анализа особенностей управляющего графа и его взаимодействия с информационным графом функционально-потоковой параллельной программы разработан событийный процессор, обеспечивающий асинхронного управление вычислениями. Это обеспечивает более гибкую отладку программ.

4 Для комплексного объединения процессов разработки, отладки и выполнения функционально-потоковых параллельных программ разработана архитектура инструментальной системы. Разработанная архитектура реализована в виде инструментальная системы функционально-потокового параллельного программирования, обеспечивающая трансляцию, отладку и выполнение функционально-потоковых параллельных программ с применением предложенных к подходов к организации функционально-потоковых параллельных вычислений.

Результаты работы отражены в следующих публикациях:

1 Редькин, А.В. Событийное управление выполнением функционально-потоковых параллельных программ / А.В. Редькин, А.И. Легалов // Научный вестник Новосибирского государственного технического университета. – №3(32). – 2008. – С. 111-117.

2 Легалов, А.И. Функционально-потоковое параллельное программирование при асинхронно поступающих данных / А.И. Легалов, А.В. Редькин, И.В. Матковский // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта - 3 апреля 2009 г.). Челябинск: Изд. ЮурГУ, 2009. – С. 573-578.

3 Редькин, А.В. Промежуточное представление функционального языка потокового программирования / А.В. Редькин // Третья Сибирская школы-семинар по параллельным вычислениям ; под ред. проф. А.В. Старченко. Томск: изд-во Том. ун-та, 2006. – С. 121-125.

4 Легалов, А.И. Расширение асинхронного управления по готовности данных / А.И. Легалов, А.В. Редькин // Труды III Международной конференции «Параллельные вычисления и задачи управления» PACO '2006. Москва, Институт проблем управления им. В.А. Трапезникова РАН, – 2006. – С. 1272-1281.

5 Редькин, А.В. Событийная интерпретация функционально-потоковых параллельных программ // Четвертая Российско-германская школа по параллельным вычислениям. Научная сессия. Тез. докл. Новосибирск, 2007. – С. 29-30.

6 Редькин, А.В. Среда разработки для языка параллельного программирования ПИФАГОР / А.В. Редькин // Четвертая Сибирская школа-семинар по параллельным вычислениям. Тез. докл. Томск, 2007. – С. 20-22.

7 Редькин, А.В. Система инструментальной поддержки языка параллельного программирования ПИФАГОР / А.В. Редькин // Молодежь и наука: начало XXI века: Сб. материалов Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых: в 4 ч. – Ч. 1. Красноярск, 2007. – С. 26-29.

8 Редькин, А.В. Система интерпретации программ на языке ПИФАГОР / А.В. Редькин // Молодежь и наука: начало XXI века: Сб. материалов Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых: в 5 ч. – Ч. 2. Красноярск, 2008. – С. 100-102.

9 Редькин, А.В. Промежуточное представление информационного графа для языка Пифагор / А.В. Редькин // Информатика и информационные технологии. Красноярск. – 2004. – С. 205-209.