Интегрированная визуальная среда поддержки конструирования параллельных программ

Касьянов В.Н.

Институт систем информатики СО РАН

Введение

Параллельные вычисления в настоящее время являются одной из главных парадигм современного программирования и охватывают чрезвычайно широкий круг вопросов разработки программ. Ввиду значительно более сложной природы параллельных вычислений по сравнению с последовательными большое значение приобретают методы автоматизации разработки параллельного программного обеспечения, основанные на применении техники формальных моделей, спецификаций и преобразований параллельных программ [1, 3, 9, 13].

Фундаментальными проблемами организации параллельных вычислении являются: проблема увеличения производительности и эффективности использования многопроцессорных и распределенных вычислительных систем и проблема повышения уровня интеллектуализации программирования параллельных систем. Они не являются независимыми, ибо организация высокопроизводительных вычислений в многопроцессорной системе современной архитектуры оказывается слишком сложной для попыток ее решения без средств интеллектуализации программирования в такой системе. Трудность решения вопросов программирования параллельных систем определяется тем обстоятельством, что вопросы организации взаимодействий и синхронизации параллельных процессов существенно усложняют разработку параллельных алгоритмов и программ по сравнению с их традиционными (последовательными) вариантами. Имеющиеся средства автоматизации программирования многопроцессорных систем, в основном для языков семейства Фортран и С, не могут удовлетворить потребности все возрастающего количества пользователей, так как имеют ограниченные возможности как по предоставляемым средствам параллельного программирования, так и по кругу решаемых задач, характеризуемых в основном регулярными схемами вычислений.

Используя традиционные языки и методы, очень трудно разработать высококачественное, переносимое программное обеспечение для параллельных компьютеров. В частности, параллельное программное обеспечение не может быть разработано с малыми затратами на последовательных компьютерах и потом перенесено на параллельные вычислительные системы без существенного переписывания и отладки. Поэтому высококачественное параллельное программное обеспечение может разрабатываться только небольшим кругом специалистов, имеющих прямой доступ к дорогостоящему оборудованию. Однако, используя языки программирования с неявным параллелизмом, такие как функциональный язык Sisal [18], можно преодолеть этот барьер и предоставить широкому кругу прикладных программистов, не имеющих достаточного доступа к параллельным вычислительным системам, но являющихся специалистами в своих прикладных областях, возможность быстрой разработки высококачественных переносимых параллельных алгоритмов на своем рабочем месте.

Функциональная семантика языков программирования с неявным параллелизмом гарантирует детерминированные результаты для параллельной и последовательной реализации – то, что невозможно гарантировать для традиционных языков, подобных языку Фортран. Более того, неявный параллелизм языка снимает необходимость переписывания исходного кода при переносе его с одного компьютера на другой. Гарантировано, что программа с неявным параллелизмом, правильно исполняющаяся па персональном компьютере, будет давать те же результаты при ее исполнении на высокоскоростном параллельном или распределенном вычислителе.

Статья посвящена системе функционального программирования SFP, создаваемой в Институте систем информатики СО РАН при финансовой поддержке РФФИ (грант № 07-07-12050) [16]. Система SFP предназначена для поддержки разработки высококачественного переносимого программного обеспечения для параллельных вычислителей на недорогих персональных компьютерах. Разработанный в качестве входного языка системы язык функционального программирования Sisal 3.2 обладает неявным параллелизмом, гарантирует детерминированные результаты и поддерживает аннотированное программирование. Система SFP использует графовые промежуточные представления функциональных программ и предоставляет средства для написания и отладки Sisal-программ независимо от целевой архитектуры, а также для трансляции Sisal-программ в оптимизированные императивные программы, подходящие для целевых платформ исполнения.

Статья структурирована следующим образом. В п. 1 кратко описываются особенности языка Sisal 3.2. Общее описание системы SFP приводится в п. 2, п. 3 посвящен рассмотрению базовой части системы. Метатранслятор и front-end транслятор

системы представлены в п. 4, пп. 5 и 6 содержат описание используемых системой промежуточных представлений программ.

1. Язык Sisal 3.2

Название языка Sisal является аббревиатурой английского выражения «Streams and Iterations in a Single Assignment Language» (потоки и итерации в языке однократного присваивания) [18]. Создание языка — результат сотрудничества Ливерморской национальной лаборатории имени Лоренца, Университета штата Колорадо, Манчестерского университета и Digital Equipment Corporation (DEC). Язык ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка VAL.

По сравнению с императивными языками (подобными языку Фортран) функциональные языки, такие как Sisal, упрощают работу программисту. В функциональной программе программист должен только специфицировать результаты вычислений и может переложить большую часть работ по организации вычислений на компилятор, который отвечает за отображение алгоритма на определенную архитектуру вычислителя (включая планирование команд, передачу данных, синхронизацию вычислений, управление памятью и т.д.). По сравнению с другими функциональными языками Sisal поддерживает типы данных и операторы, присущие научным вычислениям, такие как циклы и массивы. Например, Sisal-программа умножения матриц может иметь вид:

```
type OneDim = array[ double_real ];
type TwoDim = array[ OneDim ];
function Main(A,B: TwoDim;
    M,N,L: integer returns TwoDim)
 for I in 1, M cross J in 1, L
 S := for K in 1, N
 R := A[I,K] * B[K,J]
 returns value of sum R
 end for
 returns array of S
 end for
end function
```

Sisal разрабатывается как язык функционального программирования, специально ориентированный на параллельную обработку в научных вычислениях и на замену языка Фортран на суперкомпьютерах [14]. О реальном вытеснении говорить еще рано, но Sisal как язык параллельного программирования достаточно интересен сам по себе и уже нашел свое применение в десятках организаций разных стран мира. Существует несколько реализаций языка Sisal (версии 1.2) для суперЭВМ, в частности на Denelcor HEP, Vax 11-780, Cray-1, Cray-X/MP, создан прототип оптимизирующего компилятора с языка Sisal 1.2 в распределенные программы для вычислителей, аппаратно или программно поддерживающих многонитевые вычисления, таких как, например, TERA, *T, TAM и MIDC.

Ливерморская национальная лаборатория и Манчестерский университет разработали усовершенствованную версию языка Sisal 90 [15], которая пока еще нигде не была реализована. К наиболее значительным нововведениям языка можно отнести поддержку функций высших порядков и их полиморфизма, операций над массивами и потоками разных размерностей, потенциально недетерминированных пользовательских редукций, а также введение развитых расширений для организации циклических вычислений и вызовов функций, написанных на других языках программирования.

Последние годы в Институте систем информатики СО РАН ведутся исследования по разработке входного языка создаваемой системы параллельного программирования SFP на базе языка Sisal 90 [6]. Текущим результатом этих работ явился язык Sisal 3.2 [8]. Язык Sisal 3.2 был получен путём развития языка Sisal 90 в сторону поддержки расширенных межмодульных взаимодействий, мультиязыкового и объектно-ориентированного программирования, а также возможностей предварительной обработки (preprocessing) и аннотированного программирования.

Для повышения уровня абстракции алгоритмов и возможности взаимодействия с другими языками программирования в язык Sisal 3.2 были введены новые концепции пользовательских типы с параметрами обобщенных процедур и инородных типов. В языке Sisal 3.2 впервые было дано точное описание семантики аннотаций-утверждений ([17]) — прагм, позволяющих пользователю управлять оптимизирующими преобразованиями и настройкой транслируемой программы на целевой вычислитель. Язык Sisal 3.2 поддерживает также возможность (с синтаксисом, развивающим синтаксис языка Sisal 2.0 в этой области) использования функций уже написанных программ на других языках программирования, таких как языки C++, C и Фортран.

Пользовательские типы с параметрами позволяют задавать составные типы со специфическими операциями. Например, можно определить тип матрицы произвольного типа и операцию перемножения этих матриц (не поэлементного). Операция перемножения матриц задаётся с помощью обобщенной процедуры. Каждой обобщенной процедуре сопоставляется ранее определённый контракт, в котором указаны, какие операции должны поддерживать типы, задаваемые параметрами обобщенной процедуры. Например, для элементов типа матрицы, это операции сложения и умножения. Далее приведён пример описания матрицы и операции их перемножения.

```
contract additive[T]
 operation + (T, T returns T)
 operation * (T, T returns T)
end contract
```

```
type matrix[T] = array [..,..] of T
operation * of additive[T] (matrix[T],
   matrix[T] returns matrix[T])
```

Инородные типы задаются некоторым строковым представлением на другом языке программирования. Значения инородных типов конструируются с помощью инородных операций и функций (написанных на другом языке программирования и расположенных в другом модуле со специальным интерфейсом на языке Sisal 3.2). С помощью инородных типов можно задавать архитектурнозависимые типы и определять операции неявного преобразования между ними и «машинно-независимыми» типами языка Sisal. Например, можно определить инородные типы целых и вещественных чисел фиксированного размера и определить операции неявного преобразования между ними и типами «integer» и «real» языка Sisal. Также на инородных типах основывается возможность использования языком Sisal 3.2 функций уже написанных программ на других языках программирования.

2. Система SFP

Система SFP разрабатывается с целью предоставить прикладному программисту на его рабочем месте удобную среду для разработки функциональных программ на языке Sisal, предназначенных для последующего исполнения на параллельных супервычислителях, доступных через телекоммуникационные сети [6, 16].

В рамках этой среды программист должен иметь возможность, с одной стороны, создавать и отлаживать Sisal-программу без учета целевой параллельной архитектуры, а с другой — производить настройку отлаженной программы на ту или другую целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ.

Процедура настройки состоит в реализации оптимизирующей кросс-трансляции разработанной функциональной программы в программу на языке супервычислителя (например, на языке C или C#), в процессе которой программа подвергается необходимым оптимизирующим и реструктурирующим преобразованиям под управлением пользователя. При этом степень участия программиста может быть различной — от предоставления дополнительной информации до прямого управления производимыми преобразованиями. В частности, программист должен иметь возможность визуальной обработки создаваемой Sisal-программы в рамках ее внутреннего представления.

Такие возможности делают супервычислители, включенные в сеть, более доступными для использования широкому кругу прикладных программистов, а также позволяют упростить работу прикладным программистам и повысить эффективность использования ими супервычислителей за счет переноса работ по конструированию и отладке программ с дорогих супервычислителей на более дешевые и привычные персональные компьютеры, а также за счет снятия необходимости выполнять эти работы для одной и той же задачи каждый раз заново при переходе с одного супервычислителя на другой.

Система SFP включает следующие компоненты: ядро, визуальный каркас, отладчик, метатранслятор, транслятор, ретранслятор, блоки промежуточных представлений IR1, IR2 и IR3 [10, 12], блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, генераторы выходного кода. IR1 и IR2 – это языки иерархических графов [5], представляющие функциональные программы в виде схем над распределенной и общей памятью [4, 7], а IR3 — язык для представления императивных программ.

Общая схема процесса кросс-компиляции в системе представлена на рис. 1. Исходная Sisal-программа («Source» на схеме) поступает на вход транслятора (front-end транслятора), который строит её первое внутреннее представление IR1. Далее граф IR1 подается на вход back-end части компилятора. Back-end часть включает следующие фазы (где фазы оптимизации являются необязательными): трансляция из IR1 в IR2 («IR2 Gen» на схеме); оптимизация IR2 («IR2 Opt» на схеме); трансляция из IR2 в IR3 с генерацией параллельного кода («IR3 Gen» на схеме); оптимизация IR3 («IR3 Opt» на схеме); понижение уровня IR3 (входит в блок «IR3 Opt»); трансляция IR3 в код целевой архитектуры («CodeGen» на схеме).

Целевой платформой для существующей реализации системы является платформа .NET. В ней в качестве кодогенератора используется транслятор внутреннего представления IR3 в программу на языке C#. Полученная программа транслируется в байт-код .NET компилятором языка C#. Разработана библиотека, содержащая систему классов C#, обеспечивающих поддержку периода исполнения для Sisal программ. Кроме того, пользователь может использовать эти классы для обеспечения

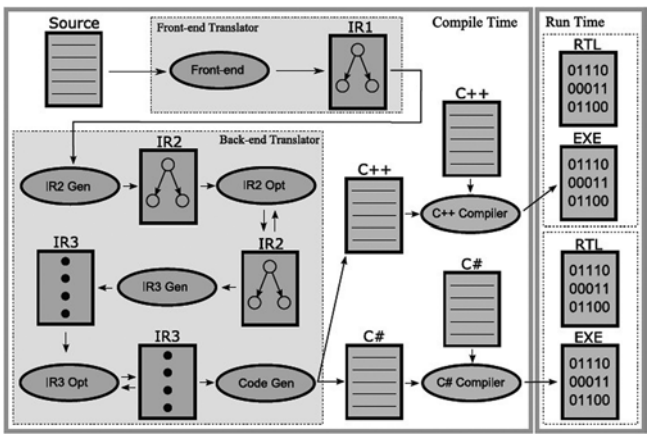

Рис. 1. Схема компиляции и исполнения Sisal программ

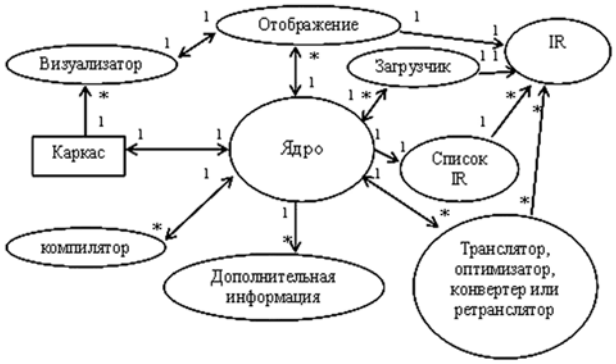

Рис.2. Общая схема связей компонентов системы SFP

взаимодействия Sisal-программы и кода на языке C# (например, организовывать ввод-вывод).

3. Базовая часть системы SFP

На основе разработанного подхода к интеграции средств поддержки функционального программирования создана базовая часть системы, состоящая из загружаемого ядра и визуального каркаса (рис. 2) [2].

Каркас предоставляет базовый интерфейс пользователя: главное окно, меню, строку состояния, панели управления и инструментов, дочерние окна документов и другие стандартные элементы пользовательского интерфейса. Функциональность этого интерфейса пользователя зависит от загруженных компонентов и их состояния. Например, пункты меню могут меняться динамически. При своей инициализации компоненты получают ссылку на интерфейс ядра, а у него можно запросить ссылку на интерфейс каркаса. Каркас позволяет модулям создавать окна, меню и элементы управления. Кроме того, каркас может сам создавать элементы пользовательского интерфейса и предоставлять программный интерфейс модулям для работы с этим интерфейсом. В задачи каркаса входит также создание в отдельном окне нужного элемента управления. Элементами управления могут являться, например, визуализаторы.

Визуализаторы позволяют отображать на экране информацию в виде графических изображений. Например, может быть визуализатор, рисующий граф представления программы. Способ задания графа визуализатору может быть любым и не обязан зависеть от представления программы, т.е. визуализатор может просто уметь рисовать графы некоторого типа. О том, как отобразить представление программы с помощью визуализатора, знает отображение. Отображения могут являться не отдельными компонентами, а лишь интерфейсами визуализаторов. Каждое отображение предназначено для связи внутреннего представления программы и визуализатора.

Компоненты представления и визуализации не обязаны знать о существовании друг друга, всё их взаимодействие осуществляется через компоненты отображений. Использование компонентов отображений позволяет увеличить повторное использование компонентов визуализации. Компоненты загрузки и сохранения позволяют хранить внутренние представления в файловом виде. Форматы хранения могут быть текстовыми или бинарными. Текстовые форматы могут быть полезны для ручного редактирования.

Компоненты взаимодействуют друг с другом и пользователем при помощи событий. Пользователь генерирует события при помощи вызова команд пользовательского интерфейса. Под событием понимается вызов одного компонента другим. События реализуются при помощи событий .NET. Вызывающий компонент имеет информацию только о сигнатуре вызова. Вызывающий компонент называется источником события, а вызываемый — подписчиком. Если компонент может обрабатывать события определённого типа, то он подписывается ко всем источникам данного события. О появлении нового источника оповещаются все компоненты, что позволяет организовать подписку. На одно событие могут подписаться сразу несколько компонентов (это не относится к пользовательским командам). Использование механизма событий позволяет упростить связи между модулями, а также интеграцию новых модулей, так как источники и подписчики могут работать независимо друг от друга.

4. Метатранслятор и front-end транслятор

В основу метатранслятора, предназначенного для автоматизации построения front-end трансляторов (трансляторов с входного языка системы в промежуточное представление IR1), положены предложенные понятия γ-автоматов и γ-схем, позволяющие наглядно описывать синтаксис языка в непосредственно исполняемом виде [11].

Модель γ-автомата основана на классической модели магазинного автомата и получается из неё за счет ограничений на вид функции перехода, позволяющих повысить наглядность графового представления автомата при сужении класса допустимых языков, и последующих расширений модели дополнительными состояниями и возможностями их обработки, позволяющими расширить класс допустимых автоматом языков при сохранении наглядности графового представления модели.

В γ-автомате допускается использование переходов следующих трёх видов: переходы, не зависящие от содержимого магазина и не меняющие его содержимое; переходы, не зависящие от содержимого магазина, но добавляющие к нему состояния; и переходы в состояние, вытолкнутое из магазина. Класс языков, распознаваемых детерминированными γ-автоматами, совпадает с классом LL_1-языков, но расширение автоматов контекстными состояниями позволяет задавать разбор более широких классов языков. Кроме того, модель γ-автомата расширяется средствами иерархической обработки неопределённостей, позволяющи-

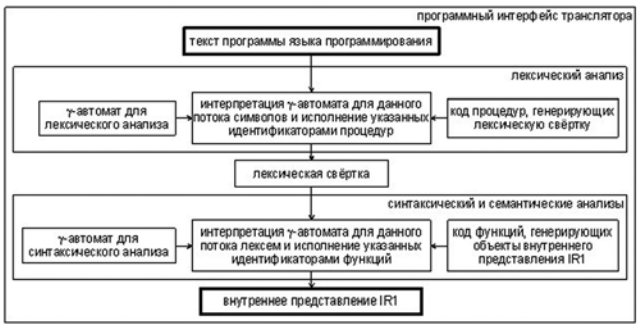

Рис. 3. Схема front-end транслятора, реализованного с применением метатранслятора

ми осуществлять обнаружение и нейтрализацию ошибок трансляции без накладных расходов полного определения γ-автомата.

С помощью γ-схем можно реализовать весь транслятор целиком, объединяя описания синтаксиса и семантики языка программирования. Для этого к вершинам и дугам γ-схем добавляются в качестве пометок имена трансдукторов (транслирующих процедур). Реализация этих процедур в формализме γ-схем не уточняется и может иметь общий модифицируемый контекст исполнения, позволяющий задавать трансляцию произвольной сложности.

Метатранслятор предполагает разделение спецификации конкретного транслятора на три части: графическую, текстовую и интерпретирующую. Графическая часть описывает γ-схему, которая задает γ-автомат, распознающий синтаксис языка. Текстовая часть содержит описание контекстно-зависимых переходов («семантики отношений») и транслирующих процедур («операционной семантики»), использующихся в γ-автомате. Интерпретирующая часть транслятора исполняет его графическую и текстовую части. Общая схема конструируемого транслятора приведена на рис. 3.

Разделение входа метатранслятора на графическую и текстовую части позволяет сочетать сильные качества обеих форм представления. Графические спецификации больше подходят для проектирования и документирования из-за богатства способов представления, легче усваиваемых кратковременной памятью человека. Текстовые спецификации лучше подходят для реализации программы по причине сочетания строгости, гибкости, компактности записи и переносимости.

К другим преимуществам разделения входа метатранслятора на графическую и текстовую части относится: простота модификаций входного языка путём наглядных изменений γ-схемы; высокая переносимость по причине интерпретируемости γ-автомата; упрощение реализации и тестирования процедур (проверки входных и выходных условий) за счёт их логической обособленности; автоматизация тестирования транслятора за счёт наличия исполняемого описания синтаксиса в виде γ-автомата; возможность использования динамических оптимизаций на уровне интерпретатора γ-автомата, настраивающих его на транслируемый язык или даже на конкретную программу языка в процессе её разбора; возможность легкой инструментации транслятора на уровне интерпретатора γ-автомата и сбора статистики программ транслируемого языка.

Front-end транслятор языка Sisal 3.2 системы SFP спроектирован с использованием рассмотренного разделения на графическую, текстовую и интерпретирующую части. Front-end транслятор осуществляет лексический и синтаксический разбор (совмещённый с семантическим) текста программ языка Sisal 3.2 , в процессе которого строится по входной программе ее IR1-представление. Разработанный транслятор обеспечивает простоту учёта модификаций языка, удовлетворительную скорость трансляции, качественные сообщения об ошибках и предупреждениях и развитые механизмы восстановления после ошибок разбора. Использование γ-схем позволило сократить объём текста front-end транслятора с языка Sisal 3.2 до десяти тысяч строк, по сравнению с тридцатью тысячами строк кода аналогичной части транслятора OSC 12.0 для более простой версии языка Sisal 1.2 [18].

5. Внутреннее представление IR1

Первый промежуточный язык IR1 является языком помеченных иерархических графов [5, 7], построенных из (вычислительных) вершин, дуг, портов и типов (рис. 4). Вершины могут быть простыми или составными. Простые вершины являются вершинами основного графа и обозначают литералы или операции, такие как сложение или деление. Составные вершины являются фрагментами (или подграфами) основного графа и представляют составные конструкции, такие как структурные выражения и циклы. Порты — это другой тип вершин основного графа, которые используются для представления операндов вычислений. Они делятся на входные порты (входы) и выходные порты (выходы). Дуги соединяют порты и изображают передачу значений от одного операнда к другому. Они идут от выходов к входам вычислительных вершин, содержащихся в одном и том же фрагменте, либо соединяют выходы вершин или фрагментов с выходами фрагментов, непосредственно их содержащих, или входы фрагментов с входами фрагментов или вершин, в них непосредственно вложенных. Типы — это пометки дуг, которые представляют типы значений, передаваемых по этим дугам.

На рис. 4 изображено IR1-представление следующей функции, которая осуществляет быструю сортировку целочисленного массива:

```
function QuickSort(Data: array[integer]
   returns array[integer])
 if size(Data) <= 1 then Data else
```

```
    let Pivot := Data[1];
    Low, Mid, High := for E in Data
    returns array of E when E < Pivot;
    array of E when E = Pivot;
    array of E when E > Pivot
    end for
    in QuickSort(Low) || Mid ||
      QuickSort(High)
    end let
    end if
  end function
```

В языке IR1 альтернатива явным образом реализуется неявными зависимостями между элементами составной вершины **Select**. Первый ее подфрагмент имеет один выход целого типа, значение которого определяет то, какой из других подфрагментов составной вершины **Select** будет использоваться для генерации её выходных значений. Литералы в графе IR1 задаются вершинами без входных портов и с одним выходным портом. Так как язык IR1 описывается иерархией ациклических графов, то вызовы функций задаются литералом функционального типа с именем функции, однозначно указывающим её граф. В языке IR1 имеются также составные вершины **LoopA**, **LoopB** и **Forall**, которые представляют циклы с постусловием, предусловием и диапазоном соответственно. Данные составные вершины содержат

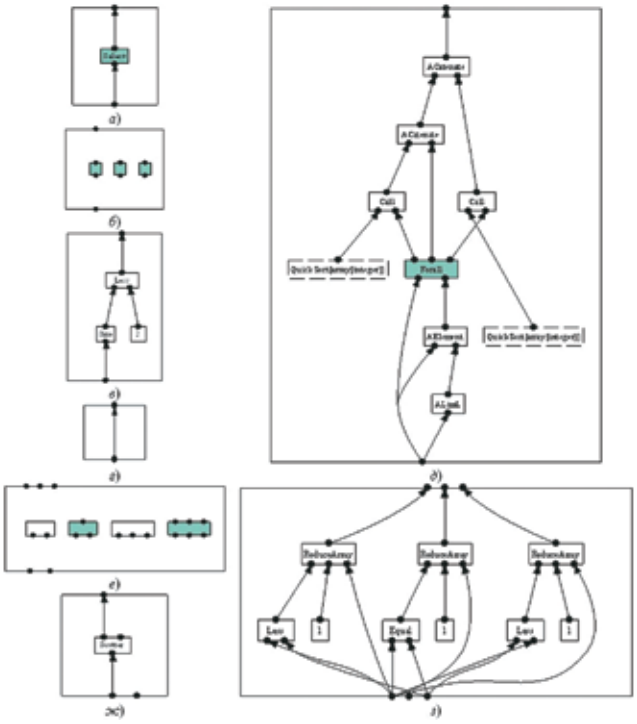

Рис. 4. Изображения графов, сгенерированные автоматически по IR1-представлению, построенному front-end транслятором для функции QuickSort: *а* – граф функции QuickSort, *б* – граф составной вершины **Select**, *в* – первый подфрагмент вершины **Select**, *г* – второй подфрагмент вершины **Select**, *д* – третий подфрагмент вершины **Select**, *е* – граф составной вершины **Forall**, *ж* – второй подфрагмент вершины **Forall**, *з* – четвёртый подфрагмент вершины **Forall**

фиксированное число элементов (подфрагментов), описывающих управление циклом, тело цикла и его предложение возврата.

6. Внутренние представления IR2 и IR3

Внутреннее представление IR2 отличается от IR1 введением общей памяти в виде множества переменных *VAR* и отображения σ_v: $E \rightarrow VAR$, задающего привязку переменных к дугам E иерархического графа, а также явным заданием отношения частичного порядка $\leq_\beta$ на вычислительных вершинах иерархического графа, ограничивающего возможные последовательности их исполнения. Предполагается, что если $N1 \leq_\beta N2$, то вычисление, представленное вершиной *N1*, должно обязательно происходить перед вычислением вершины *N2*. Если же вершины *N1* и *N2* не связаны отношением $\leq_\beta$, то выполнять операции для этих вершин можно в любом порядке; кроме того, возможно их параллельное исполнение.

Для внутренних представлений IR2 (и IR3) определяются такие объекты, как переменная и тип. При этом тип в IR2 (и IR3) служит уже для представления типов языка Sisal на уровне back-end транслятора. Тип содержит низкоуровневую информацию об объекте, с которым ассоциирован этот тип (такую как машинное представление типа и класс памяти).

Переменная описывает объекты языка Sisal на уровне IR2 и IR3. В представлении IR2 переменные ассоциируются с дугами графа, а в IR3 они служат операндами операций IR3, и имеют следующие атрибуты: уникальный идентификатор, уникальное имя, тип и дополнительную булеву переменную, отвечающую за свойство «is error». Переменные делятся на скалярные, массивные и записи. Каждая из этих групп предполагает наличие у переменных некоторых дополнительных атрибутов. У скалярных переменных — это размер в байтах. Переменные-массивы дополнительно содержат три вспомогательные переменные: переменная, описывающая элемент массива, нижняя граница массива и его размер. Переменные-записи содержат список переменных, описывающих поля данной записи.

IR2-представление строится из IR1-графа в два этапа. Первым этапом является построение отображения σ_v, т.е. аннотирование дуг иерархического графа переменными. Сначала оно строится так, чтобы дугам, выходящим из одного порта, приписывалась одна и та же переменная, а дугам, выходящим из разных портов, — разные переменные. Этим достигается выполнение требования единственности определения каждой переменной. В дальнейшем (при оптимизации IR2) распределение переменных по дугам графа может меняться. Заключительным этапом построения IR2 является упорядочивание вершин отношением приоритета исполнения $\leq_\beta$, которое строится исходя из ограничений, накладываемых дугами, связывающими

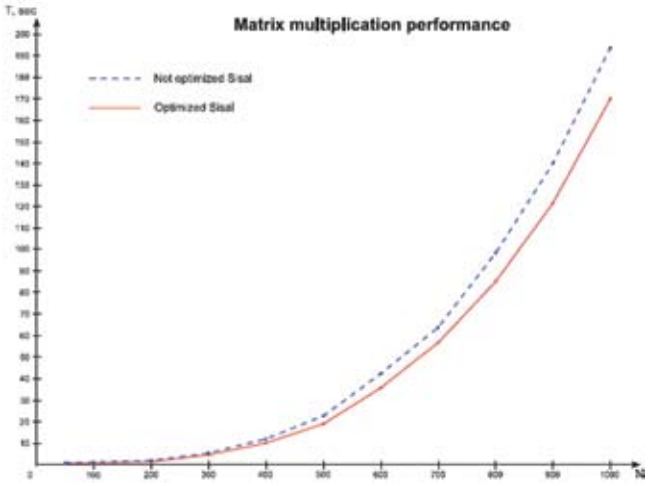

Рис. 5. Эффективность оптимизации

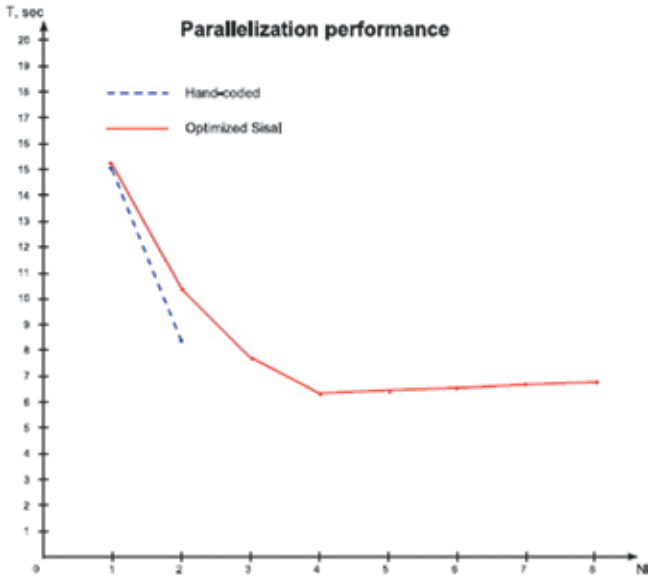

Рис. 6. Эффективность распараллеливания

операнды операторов, и правилами вычисления элементов (вершин) составных вершин, описывающих условные выражения и циклы.

Фаза оптимизации IR2 выполняет оптимизацию распределения переменных агрегатных типов для дуг графа и вынос инвариантных вычислений из циклов [4]. Оптимизация переменных агрегатных типов заключается в таком перераспределении переменных по дугам графа, которое позволяет избежать избыточного копирования объектов. Это преобразование выполняется в два прохода: первый проход — поиск вершин, которые являются единственными использованиями агрегатных типов; второй — привязывание входу и выходу каждой из таких вершин одной и той же переменной.

После оптимизации IR2 выполняется распараллеливание. На этом этапе производится раскраска внутреннего представления с целью выявления параллельной структуры алгоритма и нахождения фрагментов кода, для которых распараллеливание будет малоэффективным. После этого производится пометка циклов, итерации которых могут быть выполнены независимо (ParDo). В зависимости от определенных условий некоторые из этих циклов назначаются для параллельного исполнения.

Представление IR3 является среднеуровневым императивным представлением программы, состоящим из операторов и выражений. Операторы IR3 соединены между собой лексически (граф потока управления не строится).

В процессе трансляции из IR2 в IR3 для графа потока данных вычислений IR2 фактически строится императивная программа (последовательность операторов), выполняющая вычисления, заданные этим графом. Построение IR3 из графа IR2 включает генерацию последовательности операторов для каждой вершины и размещение полученного фрагмента в общей последовательности операторов IR3.

Блок оптимизации внутреннего представления IR3 выполняет втягивание переменных, удаление мертвых присваиваний и удаление бесполезных присваиваний [4]. Эффективность оптимизирующих преобразований на примере задачи умножения двух матриц показана на рис. 5.

После выполнения оптимизирующих преобразований выполняется «понижение уровня» IR3, которое заключается в замене «функций-интринсиков» (intrinsic functions) на последовательность простых операторов или «функций-интринсиков» более низкого уровня, которые заменяются уже на фазе кодогенерации.

Распараллеливание на уровне IR3 заключается в создании вспомогательных структур, относящихся непосредственно к целевому языку трансляции. На этом шаге создаются дополнительные переменные и классы, которые требуются для описания параллельной работы программы. Эффективность распараллеливания на примере задачи умножения двух матриц показана на рис. 6. Тесты были проведены на 4-х процессорной ЭВМ с SMP-архитектурой.

Заключение

В статье описана система SFP, предназначенная для поддержки разработки высококачественного переносимого программного обеспечения для параллельных вычислителей на недорогих персональных компьютерах, и ее основные компоненты в существующем виде. Вкратце описаны основные черты языка Sisal 3.2. Описаны внутренние представления IR1, IR2 и IR3. Рассмотрена схема front-end трансляции из языка Sisal 3.2 в первое внутренне представление IR1. Представлены результаты исследования эффективности оптимизирующих преобразований и результаты исполнения программ на SMP архитектуре.

Благодарности

Автор благодарен всем участникам проекта SFP, работа над которым выполняется при поддержке РФФИ (грант № 07-07-12050).

Литература

1. Воеводин В.В., Воеводин Вл.В., Параллельные вычисления. — СПб., БХВ-Петербург, 2002. — 609 С.
2. Глуханков М.П. Интегрированная среда визуального функционального программирования SFP // Молодая информатика. — Новосибирск, ИСИ СО РАН, 2005. — С. 21—30.
3. Евстигнеев В.А., Касьянов В.Н. Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. — 1996. — № 6. — С.12—26.
4. Касьянов В.Н. Оптимизирующие преобразования программ. — М.: Наука, 1988. — 336 С.
5. Касьянов В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. — Новосибирск, ИСИ СО РАН,1999. — С. 7 — 32.
6. Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А. Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск: ИСИ СО РАН, 2001. — С.54 — 67.
7. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003. — 1104 С.
8. Касьянов В.Н., Стасенко А.П. Язык программирования Sisal 3.2 // Методы и инструменты конструирования программ. — Новосибирск: ИСИ СО РАН, 2007. — С. 56— 134.
9. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. — Новосибирск, Изд-во НГТУ,2006. — 296 С.
10. Пыжов К.А. Внутренние представления среднего уровня для компиляторов языка Sisal // Методы и инструменты конструирования программ. — Новосибирск: ИСИ СО РАН, 2007. — С. 174— 185.
11. Стасенко А.П. Автоматная модель визуального описания синтаксического разбора // Методы и инструменты конструирования программ. — Новосибирск: ИСИ СО РАН, 2007. — С. 186–209.
12. Стасенко А.П. Система интерфейсов транслятора во внутреннее представление IR1 // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, ИСИ СО РАН, 2005. — С. 229—238.
13. Allen R., Kennedy K. Optimizing compilers for modern architectures. A dependence-based approach — Morgan Kaufmann Publishers, 2002. — 790 P.
14. Cann D.C. Retire Fortran? A debate rekindled // Commun. ACM. — 1992. — Vol. 35, № 8. — P. 81—89.
15. Feo J.T., Miller P.J., Skedzielewski S.K. and Denton S.M. Sisal 90 user's guide. Livermore, CA: Lawrence Livermore National Laboratory, Draft 0.96, 1995. — 80 P.
16. Kasyanov V.N., Stasenko A.P., Gluhankov M.P., Dortman P.A., Pyjov K.A., Sinyakov A.I. SFP — an interactive visual environment for supporting of functional programming and supercomputing // WSEAS Transactions on Computers. — 2006. — Vol. 5, Issue 9. — P. 2063— 2069.
17. Kasyanov V.N., Transformational approach to program concretization // Theoretical Computer Science. — 1991. — Vol. 90, № 1. — P. 37— 46.
18. McGraw J.R., Skedzielewski S.K., Allan S.J., Oldehoeft R.R., Glauert J., Kirkham C., Noyce B., Thomas R. Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.2. — Livermore, CA, 1985. — (Tech. Rep. / Lawrence Livermore National Laboratory; M-146, Rev. 1).