УДК 681.3

А.Е. ДОРОШЕНКО, Н.В. КОТЮК, С.С. НИКОЛАЕВ, Г.Е. ЦЕЙТЛИН, Е.А. ЯЦЕНКО

РАЗВИТИЕ ИНСТРУМЕНТАРИЯ АЛГЕБРЫ АЛГОРИТМИКИ С ЦЕЛЬЮ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ЭВРИСТИЧЕСКИХ СРЕДСТВ

Ключевые слова: *алгебры алгоритмов, проектирование программ, синтез программ, оптимизация алгоритмов, эвристика, искусственный интеллект*.

ВВЕДЕНИЕ

Применение алгебраической методологии в программировании является одной из современных тенденций в информатике [1, 2]. В настоящей статье используются методы, основанные на алгебре алгоритмики (АА), развивающейся в рамках украинской алгебро-кибернетической школы [3, 4]. АА близка к концепциям алгебраической алгоритмики [2] и порождающего программирования [5] и предназначена для формализации знаний о предметных областях с использованием алгебраических средств. На основе АА была реализована инструментальная система программирования (названная интегрированным инструментарием проектирования и синтеза программ, ИПС [6, 7]). В системе совмещается три формы представления знаний: аналитическая (формула в алгебре алгоритмов), естественно-лингвистическая (текстовая) и графовая (граф-схемы). ИПС основан на методе диалогового конструирования синтаксически правильных алгоритмов, ориентированном на устранение синтаксических ошибок в процессе проектирования [3]. Для АА был разработан аппарат эквивалентных преобразований [3, 4], обеспечивающий возможность улучшения объектов проектирования по выбранным критериям (например, память, быстродействие и т.д.). В качестве компоненты ИПС был разработан диалоговый трансформатор регулярных схем, базирующийся на использовании соотношений и тождеств алгебр алгоритмов. В [7] исследовалось применение системы переписывания правил TermWare для преобразования схем алгоритмов. ИПС обеспечивает пошаговую разработку программ с использованием трансформаций, начиная от формальной спецификации и заканчивая программным кодом в целевом языке программирования. Однако при этом остается проблема производительности разрабатываемых программ, которая не может быть полностью решена ввиду высокой сложности генерации кода для быстро развивающихся аппаратных платформ. Таким образом, задача разработки программ состоит в совмещении трансформационной мощности формальной алгебраической методологии с адаптивными технологиями самонастраивающихся систем программирования.

В настоящей статье алгеброалгоритмические средства ИПС используются совместно с системой программирования MySHA [8], которая обеспечивает самообучающуюся генерацию кода и динамическое настраивание программ на целевую аппаратную и программную платформы. В случае использования MySHA высокоуровневый код, разработанный в ИПС, подлежит динамическому преобразованию для настройки на целевую платформу, в частности на многоядерный процессор или графический акселератор. MySHA включает автоматизированную систему развертывания ПО на вычислительном кластере и технологию интеллектуальной оптимизации кода. Система уравновешивает потенциальные недостатки разработанных ранее средств трансформации программ ИПС (например, слишком большую зависимость от структуры конкретной программы), используя эвристическое программирование.

1. ФОРМАЛИЗОВАННОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММ В ИНТЕГРИРОВАННОМ ИНСТРУМЕНТАРИИ

Разработанный ИПС основан на алгебре алгоритмики, которая представляет собой двухуровневую систему. В основу верхнего уровня положена теория клонов (семейств алгебр) [3, 9]. Под клоном понимается универсальная алгебра $C = \langle \{F_k\}; SUPER \rangle$, где $\{F_k\}$— совокупность основ, $k = 1, 2, \ldots, n$. Каждая основа состоит из множества операций определенных типов (логического, операторного, структур памяти и данных и др.); *SUPER* — сигнатура клона, содержащая операцию суперпозиции операций, принадлежащих основам. Классическим примером одноосновного клона является двузначная алгебра логики — алгебра Поста (PA) [9]:

$$PA = \langle L(2); SUPER \rangle,$$

где $L(2)$ — множество всех булевых функций.

Под алгоритмическим клоном (AC) понимается двухосновная система

$$AC = \langle \{OP, L(2)\}; SUPER \rangle,$$

где *OP* — операторная основа, состоящая из множества неинтерпретированных операторных схем. Выбор системы образующих AC определяет систему алгоритмических конструкций, присущую тому или иному методу конструирования алгоритмов и программ (структурному, неструктурному, объектно-ориентированному). В зависимости от поставленной задачи и выбранного метода программирования далее на основе AC осуществляется построение и исследование требуемой алгебры алгоритмов.

На нижнем уровне алгебры алгоритмики осуществляется построение вполне конкретной прикладной алгебры алгоритмов, ориентированной на представление алгоритмических знаний о выбранной предметной области. Схема образования прикладной алгебры базируется на интерпретации элементарных операторных и предикатных переменных для алгебры алгоритмов, построенной на верхнем уровне.

Примером алгебры алгоритмов являются системы алгоритмических алгебр (САА) В.М. Глушкова [3, 4], представляющие собой двухосновную алгебру $\text{САА} = \langle U, B; \Omega \rangle$, где U — множество операторов, B — множество логических условий (предикатов).

Операторы являются отображениями информационного множества (ИМ) в себя, логические условия являются отображениями ИМ в множество значений 3-значной логики $E_3 = \{0, 1, \mu\}$, где 0 — ложь, 1 — истина, μ — неопределенность. Сигнатура операций САА $\Omega = \Omega_1 \cup \Omega_2$ состоит из системы Ω_1 логических операций, принимающих значения в множестве B, и системы Ω_2 операций, принимающих значения в множестве операторов U. В САА $\langle U, B; \Omega \rangle$ фиксируется система образующих Σ, представляющая конечную функционально полную совокупность операторов и логических условий. С помощью этой совокупности посредством суперпозиции операций, входящих в Ω, порождаются произвольные операторы и логические условия из множеств U и B.

На САА базируется алгоритмический язык САА/1 [3], используемый для проектирования программ в ИПС. Данный язык предназначен для многоуровневого структурного проектирования и документирования последовательных и параллельных алгоритмов и программ. Преимущество его использования заключается в возможности описания алгоритмов в естественно-лингвистической форме, удобной для человека, что облегчает достижение требуемого качества программ. Основными объектами языка САА/1 являются абстракции операторов (преобразователей данных) и условий (предикатов). Условия и операторы могут быть базисными или составными. Базисный оператор (условие) в САА-схеме считается первичной атомарной абстракцией.

Составные условия строятся из базисных с помощью следующих логических операций САА:

- дизъюнкция: ‘условие1’ ИЛИ ‘условие2’;
- конъюнкция: ‘условие1’ И ‘условие2’;
- отрицание: НЕ ‘условие’.

Составные операторы строятся из элементарных посредством операций последовательного и параллельного выполнения операторов:

- “оператор1” ЗАТЕМ “оператор2” — последовательное выполнение операторов;
- ЕСЛИ ‘условие1’ ТО “оператор1” ИНАЧЕ “оператор2” КОНЕЦ ЕСЛИ — оператор ветвления;
- ПОКА НЕ ‘условие_окончания_цикла’ ЦИКЛ “оператор” КОНЕЦ ЦИКЛА — оператор цикла;
- ПАРАЛЛЕЛЬНО($(i) = (1), \ldots, (n)$)(“оператор”) — асинхронное выполнение n операторов (ветвей).

Идентификаторы операторов в САА-схемах заключаются в двойные кавычки, а идентификаторы условий — в одинарные. Уровни в САА-схемах определяются левыми частями равенств, а структура каждого уровня конкретизирована (в терминах САА) правой частью соответствующего равенства. Левая часть равенства отделяется от правой цепочкой символов =.

ИПС предназначен для проектирования алгоритмов и генерации программ в целевых языках программирования. Проектирование алгоритмов в ИПС основано на методе диалогового конструирования синтаксически правильных программ (ДСП-методе), обеспечивающем синтаксическую правильность схем [3]. При этом инструментарий использует упомянутые три различные формы представления алгоритмов в процессе их проектирования – естественно-лингвистическую (САА-схемы), алгебраическую (регулярные схемы) и граф-схемную. ИПС поддерживает генерацию последовательных и многопоточных программ на языках Java и C++, а также MPI программ на языке C. Для интеграции с системой MySHA инструментарий ИПС также был настроен на генерацию программ в языке MySHA M10.

Интегрированный инструментарий содержит следующие компоненты:

- ДСП-конструктор, предназначенный для диалогового проектирования синтаксически правильных схем последовательных и параллельных алгоритмов и генерации программ;
- редактор граф-схем;
- трансформатор, предназначенный для интерактивного преобразования схем алгоритмов;
- генератор САА-схем, базирующийся на основе схем более высокого уровня — гиперсхем [6], представляющих обобщение грамматик структурного проектирования;
- базу данных, содержащую описание конструкций САА и базисных понятий в трех упомянутых выше формах, а также их реализацию на целевом языке программирования.

ДСП-конструктор предназначен для поуровневого конструирования схем алгоритмов сверху-вниз с использованием конструкций САА, выбираемых из списка. Процесс построения алгоритма состоит в подстановке упомянутых конструкций одна в другую и имеет вид дерева конструирования алгоритма. Выбранные пользователем конструкции, а также операторные и логические переменные, входящие в них, отображаются в дереве с дальнейшей детализацией переменных. На каждом шаге проектирования система в диалоге с пользователем предоставляет на выбор лишь те конструкции, подстановка которых в формируемый текст не нарушает синтаксической правильности схемы. В зависимости от типа выбранной переменной (операторного или логического) система предлагает соответствующий список операций или базисных понятий. Далее дерево конструирования алгоритма используется для генерации текста САА-схемы, регулярной схемы, граф-схемы и кода в целевом языке программирования.

Пример 1. Рассмотрим последовательную САА-схему алгоритма нахождения простых чисел по методу решета Эратосфена, сконструированную в инструментарии ИПС.

```
СХЕМА  PRIMES_ERATOSFEN  ====
      "Нахождение  простых  чисел  методом  решета  Эратосфена"
  КОНЕЦ  КОММЕНТАРИЯ

"ОпределениеГлобальныхДанных"
====  "Определить  массив  (IS_SIMPL)  тип  (bool)";
      "Определить  массив  (ALL_NUM  )  тип  (int)  со  значениями
([-INF,  +INF])";
      "Определить  массив  (SIMPL_TAB)  тип  (int)"

"program"
====  "Установить  все  элементы  массива  (IS_SIMPL)  в  значение
(true)"
      ЗАТЕМ
      "Прочитать  целое  (MAX)"
      ЗАТЕМ
      ДЛЯ  '(cnt)  от  (1)  до  (+INF)  с  шагом  (2)'
      ЦИКЛ
        ЕСЛИ  '(IS_SIMPL[cnt])  =  (true)'
        ТО  ДЛЯ  '(scnt)  от  (1)  до  (+INF)'
            ЦИКЛ
              ЕСЛИ  '(cnt  *  scnt) > (MAX)'
              ТО  "Перейти  к  метке  (FINISH)"
              КОНЕЦ  ЕСЛИ
              ЗАТЕМ
              "(IS_SIMPL[cnt  *  scnt])  :=  (false)"
            КОНЕЦ  ЦИКЛА
         КОНЕЦ  ЕСЛИ
      КОНЕЦ  ЦИКЛА
      ЗАТЕМ
      "Метка  (FINISH)"
      ЗАТЕМ
      (SIMPLE_TAB  :=  "Выбрать  значения  из  массива  (ALL_NUM)
в  соответствии  с  маской  (IS_SIMPL)")
      ЗАТЕМ
      "Возвратить  значение  (SIMPL_TAB)"

КОНЕЦ  СХЕМЫ  PRIMES_ERATOSFEN
```

Для дальнейшего преобразования программ, сконструированных в ИПС с целью их настройки на платформу и оптимизации, используется система MySHA.

2. РАЗРАБОТКА ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ПЛАТФОРМЫ MYSHA

.Рассмотрим результаты распараллеливания приведенного алгоритма нахождения простых чисел для выполнения на многоядерной процессорной архитектуре. Система MySHA — многоязычная кросс-платформенная технология, которая дает возможность интегрировать компоненты, созданные с использованием языков программирования с разными парадигмами (императивная и декларативная) для реализации сложных научно-исследовательских проектов [8]. Платформа MySHA состоит из следующих элементов:

• операционной системы реального времени, которая может использоваться для выполнения программ на основе платформы MySHA;

• средств выполнения MySHA-программ на графических акселераторах AMD;

• технологии Universal Binary, обеспечивающей выполнение программы в оригинальном машинном коде без использования интерпретации/JIT-компиляции промежуточного кода;

• универсальной системы типов данных, оптимизированной для научных исследований;

• высокоуровневого промежуточного универсального языка программирования;

• технологии распараллеливания и оптимизации программ для многопоточного выполнения;

• средств разработки программ и их развертывания на кластерных системах;

• подсистемы управления памятью на основе использования символьных имен;

• автоматической подсистемы управления динамической памятью на основе мгновенного освобождения памяти, гарантирующего минимальный объем использования памяти и отсутствие объектов без ссылок в любой момент времени. (Финализация объектов в системе MySHA представляет детерминированный процесс в отличие от финализации в .NET и Java.)

Идеология создания программных систем на базе платформы MySHA получила название алгоритмически-ориентированного программирования (АОП), которое имеет много общего с алгеброалгоритмическим подходом к проектированию алгоритмов, рассмотренным в разд. 1. АОП дает возможность использовать в любой ситуации оптимальный подход к проектированию системы или отдельного алгоритма. Так, например, можно объединять традиционную структурную модель языка Паскаль, объектно-ориентированную модель в стиле .NET, технологию структурирования данных MySHA, а также эвристические методы и декларативное программирование.

Важным преимуществом платформы MySHA является высокоэффективная реализация библиотек и транслятора. Так, библиотеки для ЭВМ архитектур x86, AMD FireStream и МП реализованы на языке Ассемблер. Транслятор для платформы MySHA является трехуровневой системой, которая обеспечивает высокий уровень технологической совместимости и возможность создания новых трансляторов для многих языков программирования. Технологически на всех уровнях транслятор реализован как интеллектуальная нейронная сеть с использованием динамически генерируемых процедур и эвристик. Транслятор предоставляет средства профилирования и интеллектуальной оптимизации программного кода на уровне исходных текстов и ассемблерного кода. Важным свойством транслятора является возможность программных систем динамически изменять собственный высокоуровневый код; благодаря этому появляются широкие возможности для создания интеллектуальных программных комплексов.

Основу разработанной версии языка MySHA составляет язык программирования Паскаль. Значительная часть элементов языка MySHA заимствована из Алгола, Лиспа и Си. Файл программы на языке MySHA состоит из двух частей: зоны объявлений и зоны описания главного тела программы. В зоне объявлений описываются подпрограммы, модули, типы. С главного тела программы начинается и заканчивается ее выполнение. Программа нахождения простых чисел, использующая метод решета Эратосфена, может быть представлена языком MySHA M10 следующим образом:

```
// Объявление данных
ar<bool> IS_SIMPL; // Массив булевых значений
// Массив целых чисел заполняется значениями
ar<int> ALL_NUM = $R<int>([-INF, +INF]);
ar<int> SIMPL_TAB; // Массив целых
```

```
// Главное тело программы
program
{
     IS_SIMPL.SetAll(true);
     int MAX = Console.ReadLn("Enter MAX value").ToInt();
     forcnt(cnt, 1, +INF, step 2)
     if(IS_SIMPL[scnt]==true) forcnt(scnt, 1, +INF)
     {
            if(cnt*scnt>MAX) goto FINISH;
            IS_SIMPL[cnt*scnt] = false;
     }
     label(FINISH);
     SIMPL_TAB = ALL_NUM.SelectByBoolMask(IS_SIMPL);
     return SIMPL_TAB;
}
```

Язык MySHA предлагает много средств программирования алгоритмов. Так, к традиционной императивной модели (подпрограммы и руководящие конструкции языка Паскаль) добавлено управление памятью с помощью символьных имен (в MySHA каждая переменная может иметь символьное имя), эвристическое и декларативное программирование. Эвристикой является программа или подпрограмма, которая может оптимизировать свое поведение на основе расчета оптимальной стратегии путем изменения параметров-констант, заложенных в своем алгоритме, и на основе подбора оптимальных решений, входных данных и результатов. С помощью эвристик реализуются средства интеллектуализации при генерации кода и его выполнении (в частности, средства с нечеткой логикой), предусматривающие:

- рефлексию MySHA-программы для перестройки ресурсоемких базовых алгоритмов;
- рефлексию и динамическое генерирование эвристик для построения интеллектуальных систем на основе нейронных сетей и эвристических алгоритмов;
- синтаксические конструкции и технологии для описания и хранения знаний.

3. ГЕНЕРАЦИЯ ВЫСОКОПРОДУКТИВНЫХ ПРОГРАММ

Транслятор системы MySHA генерирует эффективный ассемблерный код. Использование технологии платформенного построения дает возможность создавать программные средства, не зависимые от типа аппаратного обеспечения, а также исследовать преимущества и недостатки различных процессорных архитектур для различных типов задач. MySHA содержит качественную систему измерения производительности программы, позволяющую определить скорость программы в целом и ее отдельных частей, а также показатель использования ОЗУ. Система измерения производительности интегрирована с транслятором, который использует данные системы в режиме самообучения для определения наилучшей оптимизационной стратегии. С целью оптимизации транслятор использует три набора динамических эвристик, позволяющих:

- определять «узкие места» программы и сравнивать производительность разных вариантов одного фрагмента кода на языке MySHA (основные аналитические эвристики);
- генерировать программу на Ассемблере в коде процессора;
- указать параллельные фрагменты программы для синхронизированного выполнения на процессоре с широким командным словом;
- выполнять выборку коротких последовательностей и сортировку команд с целью оптимизации программы для суперскалярных архитектур (поддерживаются модели SIMD, CISC и EPIC, а также наборы инструкций MMX, SSE, SSE2, 3DNOW, AMD64, EM64T, PHC и т.п.);

• осуществлять распараллеливание высокоуровневого программного кода на достаточно большие автономные блоки с целью уменьшения накладных расходов на передачу сообщений между процессами;

• распределять данные между потоками и подразделять подпрограммы в отдельные функции, выполняемые отдельными потоками;

• осуществлять оптимизацию памяти за счет неявного использования передачи указателя на структуру данных вместо передачи собственно структуры;

• выполнять оптимизацию памяти за счет построения специализированных эвристик очистки динамической памяти. Эта методика дает возможность избегать традиционных недостатков технологии освобождения памяти.

С целью исследования совместного использования системы MySHA и ИПС был проведен эксперимент, в котором ИПС был настроен на генерацию программ на языке MySHA M10. Сгенерированные таким образом программы передаются на вход системы MySHA, которая осуществляет необходимую трансформацию для оптимизации и параллелизации программ.

Специальный набор тестов научных задач был использован для определения реальной производительности MySHA в кластерной системе, имеющей следующую конфигурацию:

- 9: Intel Celeron 3,06 Ghz (512 K L2 cache), 248 MB RAM;
- 2: Intel Celeron 1,7 Ghz (128 K L2 cache). 128 MB RAM;
- 1: Intel Core 2 Duo E6400, 2 GB RAM;
- 8: Intel Celeron 533 MHz (128 K L2 cache), 64 MB RAM;
- 1: AMD Athlon 64 X2 4800+, 8 GB RAM;
- 1: Mobile AMD Turion 64 X2 MT-62, 2 GB RAM.

Таким образом, кластерная система имеет 22 узла кластера, 25 процессоров, 20 вычислительных узлов, 2 управляющих узла.

Пример 2. Рассмотрим результат распараллеливания программы нахождения простых чисел, сгенерированной в интегрированном инструментарии (см. пример 1). В системе MySHA было распараллелено тело конструкции IF упомянутой программы. Представим исходную конструкцию в языке MySHA M10 следующим образом:

```
if  (IS_SIMPL[cnt]  ==  true)
{
   forcnt(scnt,  1,  +INF)
   {
      <FOR_LOOP_BODY>
   }
}
```

В результате распараллеливания в системе MySHA вместо приведенной выше конструкции в программе дан следующий текст:

```
if(IS_SIMPL[cnt] == true)
{
   branch(X000)
   {
      forcnt(scnt, 1, +INF, step 2)
      {
         <FOR_LOOP_BODY>
      }
   }
   branch(X001)
   {
      forcnt(scnt, 2, +INF, step 2)
      {
```

```
            <FOR_LOOP_BODY>
        }
    }
    launch(ar<branch>() << X000 << X001);
    meet(ar<branch>() << X000 << X001);
}
```

В рассмотренном фрагменте тело конструкции IF (которым является цикл for), разделено на два цикла, которые выполняются двумя потоками. Параметры циклов были установлены таким образом, чтобы потоки обрабатывали независимые фрагменты массива IS_ SIMPL. Тело циклов for в сравнении с последовательным вариантом осталось без изменений.

Были получены результаты выполнения распараллеленной в системе MySHA программы нахождения простых чисел, которые сравнивались с результатами выполнения программы, реализованной вручную на других платформах.

Таким образом, среднее количество найденных простых чисел (в секунду) для различных программных платформ составляют 590 000 для языка MySHA M10 (автоматизированное распараллеливание), 475 000 для C++ с использованием MPI 1.1 (ручное распараллеливание), 273 000 для .NET (C#) (ручное распараллеливание), 191000 для языка Java (ручное распараллеливание). Во всех случаях вычислялись первые 10^{10} простых чисел и использовалось 18 потоков.

ЗАКЛЮЧЕНИЕ

В настоящей статье показано развитие инструментария для разработки параллельных алгоритмов на основе интеграции алгеброалгоритмических средств и самообучающихся методов трансляции. ИПС обеспечивает формализованные диалоговые средства конструирования программ, характерным для них являются простота в обучении и использовании, а также независимость от языка программирования и возможность переведения в произвольный целевой язык. Выполнена настройка алгеброалгоритмического инструментария на генерацию кода в языке MySHA M10, что позволяет выполнять дальнейшее автоматизированное преобразование программ в системе MySHA, в частности осуществлять их распараллеливание. Преимуществом системы MySHA в сравнении с разработанными ранее для ИПС средствами трансформации программ (базирующимися на алгебраических равенствах) является использование эвристик и самообучение. Результаты эксперимента показали, что программы, реализованные на платформе MySHA с использованием эвристических технологий, имеют большую продуктивность в сравнении с программами, разработанными для других платформ (C++, .NET, J2EE).

СПИСОК ЛИТЕРАТУРЫ

1. Algebraic methodology to software technology, http://www.amast.org
2. Ноден П., Китте К. Алгебраическая алгоритмика. — М.: Мир, 1999. — 720 с.
3. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. — Киев: Академпериодика, 2007. — 631 с.
4. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. — 3-е изд. — К.: Наук. думка, 1989. — 376 с.
5. Чарнецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. — СПб.: Питер, 2005. — 731 с.
6. Yatsenko O.A. Algebras of hyperschemes and integrated tools for synthesis of programs in modern object-oriented environments // Cybernetics and Systems Analysis. — 2004. — **40**, N 1. — P. 38–42.
7. Дорошенко А.Е., Жереб К.А., Яценко Е.А. Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. — 2007. — № 1. — С. 17–30.
8. Дорошенко А.Ю., Котюк М.В., Ніколаєв С.С. Програмна платформа для наукових досліджень // Там же. — 2007. — № 4. — С. 49–59.
9. Doroshenko A., Tseytlin G., Yatsenko O., Zachariya L. A theory of clones and formalized design of programs // Proc. Int. Conf. «Concurrency, Specification, and Programming», 2006. — P. 328–339.

Поступила 30.04.2009