УДК 681.3

Е.А. Яценко

ИНТЕГРАЦИЯ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ АЛГЕБРЫ АЛГОРИТМОВ И ПЕРЕПИСЫВАНИЯ ТЕРМОВ ДЛЯ РАЗРАБОТКИ ЭФФЕКТИВНЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Предложен подход к генерации термов по высокоуровневым спецификациям алгоритмов. Генерация выполняется в рамках совместного использования системы символьных вычислений и алгеброалгоритмического инструментария проектирования и синтеза программ. Система переписывания термов дополняет алгеброалгоритмический инструментарий средствами преобразования последовательных и параллельных алгоритмов, направленных на их улучшение.

Введение

В наши дни однопроцессорные системы практически полностью вытеснены многопроцессорными, позволяющими получать значительный прирост производительности программ. При этом использование многопроцессорных систем тесно связано с необходимостью распределения вычислений на различные процессоры и ядра, т. е., связано с распараллеливанием программ [1]. Существуют библиотеки, такие как pthreads [2], OpenMP [3], TBB [4] и др., позволяющие разработчикам создавать параллельные программы. При их использовании программист вручную производит разделение кода на независимые составляющие, обеспечивает обмен данными и синхронизацию между ними (подход pthreads), либо указывает компилятору, как и какие участки кода нужно распараллеливать (подход OpenMP). Однако у таких ручных методов есть существенные недостатки, в частности, связанные с внесением ошибок в программу и временем, требуемым на распараллеливание и отладку. Учитывая процесс распараллеливания последовательных программ целесообразно максимально автоматизировать, а в идеале – осуществлять полностью автоматически, без участия программиста [5].

В предыдущих работах автора [6–8] выполнено исследование проблемы автоматизации разработки эффективных программ для многоядерных архитектур с использованием алгеброалгоритмического Инструментария Проектирования и Синтеза программ (ИПС). В основу автоматизации проектирования и программирования в рамках разработанного инструментария положена высокоуровневая формализация конструирования параллельных алгоритмов и применение формальных трансформаций программ для оптимизации их производительности. Под оптимизацией программ понимается достижение требуемых качественных характеристик программы по заданным критериям (память, быстродействие, загрузка оборудования и др.). Система ИПС базируется на представлении алгоритмов в системах алгоритмических алгебр В.М. Глушкова (САА) [9] и выполняет синтез последовательных и параллельных (многопоточных) программ на языках программирования Java, C++, Cilk++ по схемам алгоритмов. Для автоматизации выполнения трансформаций алгоритмов (программ) в работах [7, 8] была использована система символьных вычислений (на основе парадигмы переписывания термов) TermWare [10], которая расширяет возможности системы ИПС и позволяет автоматизировать решение задач анализа и обеспечения надежности программного кода. Отметим, что в [7, 8] преобразование схем алгоритмов в термы производилось вручную. Данная работа посвящена разработке средств генерации термов по спецификациям алгоритмов, представленным в САА. Подход проиллюстрирован на параллельном алгоритме рекурсивной сортировки.

Материал работы имеет следующую структуру. В разделе 1 рассматриваются средства формализованного проектирования алгоритмов, которые используются в системе ИПС. В разделе 2 кратко охарактеризована система TermWare и возможности ее применения с ИПС. Раздел 3 посвящен рассмотрению процесса генерации текстов программ и термов по схемам алгоритмов в алгебро-алгоритмическом инструментарии.

1. Средства формализованного проектирования программ и алгеброалгоритмический инструментарий

В основу разработанного инструментария проектирования и синтеза программ положен аппарат САА и их модификаций [9]. Модифицированные САА (САА-М) предназначены для формализации процессов мультиобработки, которые возникают, в частности, при конструировании программного обеспечения в мультипроцессорных системах.

САА являются двухосновной алгеброй $<U, B; \Omega>$, где U – множество операторов, B – множество логических условий. Операторы представляют собой отображения информационного множества (ИМ) в себя, логические условия являются отображениями множества 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.

К логическим операциям системы Ω_1 относятся обобщенные булевы операции: дизъюнкция, конъюнкция, отрицание, а также операция левого умножения условия на оператор.

К основным операторным операциям системы Ω_2 относятся:

• `"оператор1" ЗАТЕМ "оператор2"` – последовательное выполнение операторов;

• оператор ветвления:

```
ЕСЛИ 'условие' ТО
   "оператор1"
ИНАЧЕ
   "оператор2"
КОНЕЦ ЕСЛИ;
```

• оператор цикла:

```
ПОКА НЕ 'условие_окончания_
        цикла'
ЦИКЛ "оператор"
КОНЕЦ ЦИКЛА;
```

• оператор цикла типа for:

```
ДЛЯ ("переменная1" ОТ
     "выражение1" ДО
     "выражение2")
ЦИКЛ
  "оператор1"
КОНЕЦ ЦИКЛА
```

• асинхронное выполнение двух операторов (данная операция называется также асинхронной дизъюнкцией):

```
("оператор1"
ПАРАЛЛЕЛЬНО
"оператор2")
```

• синхронизатор, выполняющий задержку вычислений до тех пор, пока условие `'условие1'` не станет истинным:

```
ЖДАТЬ 'условие1'
```

В работах [6, 9] аппарат САА-М применен при решении задач символьной мультиобработки: последовательных и параллельных сортировок, поиска, языкового процессирования и др.

В качестве примера далее приведена САА-схема быстрой сортировки входного массива *A* длины *n*. В данном алгоритме вначале выполняется составной оператор `"main"`, считывающий размер входного массива и вызывающий

затем функцию "qmain(n)". Оператор "qmain (n)" заполняет исходный массив *A* случайными значениями и вызывает функцию "sample_qsort". Данная функция сортирует подмассив, ограниченный указателями begin и end, и использует для этого стратегию “разделяй и властвуй” [11].

```
СХЕМА QUICKSORT_SERIAL

"main" ====
====“Определить переменную (n)
    типа (int) = (1000 * 10000)"
    ЗАТЕМ
    ЕСЛИ 'Количество параметров
       командной строки больше (1)'
    ТО "Присвоить переменной (n)
       значение (1)-го параметра
       командной строки"
    КОНЕЦ ЕСЛИ
    ЗАТЕМ
    "qmain(n)"

"qmain(n)"
==== ЛОКАЛЬНЫЕ_ПЕРЕМЕННЫЕ
   (
     "Определить массив (A)
     тип (int) размер (n)";
     "Определить переменную (i)
     тип (int)";
     "Определить переменную (end)
     тип (int)"
   )
   ЗАТЕМ
   ДЛЯ (i ОТ 0 ДО n - 1)
   ЦИКЛ
     "A[i] := i"
   КОНЕЦ ЦИКЛА
   ЗАТЕМ
   "Перемешать элементы массива
   (A) длины (n) случайным обра-
зом"
   ЗАТЕМ
   "Начать отсчет времени"
   ЗАТЕМ
   (end := A + n)
   ЗАТЕМ
 "sample_qsort(A, end)"
   ЗАТЕМ
   "Закончить отсчет времени
   и вывести результат"
   ЗАТЕМ
   "Проверить отсортированность
    массива (A) длины (n)"

"sample_qsort(begin, end)"
==== ЕСЛИ НЕ('begin = end')
     ТО "Уменьшить (end) на (1)"
       ЗАТЕМ
      "Переупорядочить массив (A)
с границами (begin)(end) так,
чтобы элементы, меньшие опорного
элемента (end), находились слева
от него, а большие находились
справа; сохранить позицию опорно-
го элемента в переменной
(middle)"
    ЗАТЕМ
    "sample_qsort(begin,
     middle)"
    "Увеличить (middle) на (1)"
     ЗАТЕМ
    "Увеличить (end) на (1)"
    ЗАТЕМ
    "sample_qsort(middle, end)"
    КОНЕЦ ЕСЛИ

КОНЕЦ СХЕМЫ QUICKSORT_SERIAL
```

Интегрированный инструментарий основывается на методе диалогового конструирования синтаксически правильных программ [6] и использует различные формы представления алгоритмов в процессе их проектирования – естественно-лингвистическую (САА-схемы), алгебраическую (регулярные схемы) и граф-схемную. Отметим, что интегрированный инструментарий может быть настроен на требуемый входной язык, представленный в алгебре алгоритмов, и выходной (целевой) язык программирования. В настоящее время он поддерживает генерацию последовательных и параллельных программ на языках C++, Java [6 – 8], Cilk++, а также термов (на языке системы TermWare [10]) по САА-

схемам алгоритмов. Основными компонентами интегрированного инструментария являются:

• ДСП-конструктор, предназначенный для диалогового проектирования синтаксически правильных схем алгоритмов и синтеза программ;

• редактор граф-схем;

• трансформатор, осуществляющий диалоговое преобразование регулярных схем;

• генератор САА-схем по параметризованным схемам более высокого уровня (гиперсхемам);

• база данных алгеброалгоритмических спецификаций, содержащая описание конструкций САА-М, базисных (элементарных) операторов и предикатов, а также их отображение в язык программирования.

В основу функционирования ДСП-конструктора положен диалоговый режим с использованием меню подстановок и дерева конструирования алгоритма (см. пример дерева конструирования в [6]). Меню состоит из конструкций САА-М, суперпозиция которых позволяет создавать алгоритмы в упомянутых ранее формах. Выбранные пользователем конструкции, а также операторные и логические переменные, входящие в них, отображаются в дереве с дальнейшей детализацией переменных. В зависимости от типа выбранной переменной (операторная или предикатная) система предлагает соответствующий список операций САА-М или элементарных понятий из базы данных. Отметим поуровневый стиль конструирования алгоритма, а также возможность переходов на различные уровни (узлы дерева) с продолжением процесса диалогового конструирования. Более подробно процесс генерации программного кода по схемам алгоритмов рассматривается в разделе 3.

2. Система TermWare

Система TermWare направлена на разработку высокодинамичных прикладных систем, к которым предъявляются повышенные требования, как к встроенному интеллекту для обеспечения интерактивности разрабатываемых систем, так и удешевлению разработки, сокращению сроков проектирования, улучшения характеристик повторного их использования (реинжениринга) и сопровождаемости [10]. Она отличается от большинства других систем этого класса как назначением и семантикой используемых средств, так и технологией их реализации. Язык TermWare это координационный язык-оболочка, предназначенный для написания предметно-ориентированных частей приложений, встраиваемых в прикладную систему для реализации функций взаимодействия этой системы с программным окружением.

TermWare оформлена как библиотека языка Java, встраиваемая в программное приложение. Для разработчика система термов выглядит как совокупность экземпляров класса ITermSystem, с возможностью управления набором правил и редуцирования термов. Наборы правил описаны на языке TermWare и хранятся в отдельных файлах. Основой языка являются термы, т. е. выражения вида $f(x_1,\ldots,x_n)$. В качестве атомарных термов используются переменные (которые записываются в виде $*identifier*), а также константы определенных типов данных (числовых, логических, строковых и атомарного – неизменяемые строки).

Набор правил содержит сами правила, а также дополнительную информацию (название системы правил, используемая база фактов, применяемая стратегия). Типичное правило в TermWare записывается следующим образом:

$$source\ [condition] \rightarrow$$
$$\rightarrow destination\ [action],$$

где используются четыре терма: source – входной образец; destination – выходной образец; condition – условие, определяющее применимость правила; action – дей-

ствие, выполняемое при срабатывании правила. Выполняемые действия и проверяемые условия в основном являются вызовами методов в БД фактов. Таким образом, осуществляется связь между правилами на языке TermWare и кодом на Java. Кроме того, возможно написание собственной стратегии, определяющей порядок применения правил.

В простейшем случае переписывающее правило это выражение типа $x \to y$, которое переписывает терм x в y. Пусть, например, имеется терм $plus(x, y)$, складывающий два числа. Тогда в результате применения к нему набора из двух правил

$$x \to y,$$

$$y \to z,$$

получим терм $plus(z, z)$.

Основой функционирования системы TermWare в рамках совместного использования с ИПС является создание и применение систем правил. Каждая система правил соответствует некоторому преобразованию схемы алгоритма. Кроме самих правил TermWare, система правил может включать дополнительную информацию, например, описание преобразования или ссылки на другие системы правил, которые могут применяться совместно с данной. Возможны также информационные системы правил, которые не осуществляют преобразование схемы, а вычисляют некоторые ее характеристики. Другие возможности совместного использования более подробно рассмотрены в работе [8]. Для интеграции с системой TermWare инструментарий ИПС был дополнен средствами генерации термов по схемам алгоритмов.

3. Генерация программ и термов по схемам алгоритмов

По полученному в процессе конструирования алгоритма дереву, а также в соответствии с реализациями элементарных операторов и условий на целевом языке, инструментарий ИПС выполняет генерацию кода на выбранном языке программирования или термов TermWare. В процессе синтеза управляющие конструкции САА-схемы алгоритма и базисные операторы и предикаты отображаются в соответствующие операторы языка программирования в соответствии с их описанием в БД. Составные операторы могут быть представлены как подпрограммы (методы). В случае генерации объектно-ориентированной программы на вход генератора поступает также файл, который содержит каркасное описание основного класса программы (без реализаций методов), в который выполняется подстановка сгенерированного кода.

Для реализации параллельных программ в БД используются различные средства:

- потоки с использованием функций WinAPI [7, 12];
- параллельные операции MPI [13], ориентированные на обмен сообщениями между процессами;
- операторы языка параллельного программирования Cilk++ [14].

Для распараллеливания рассмотренного в разделе 1 алгоритма быстрой сортировки был использован язык Cilk++, так как он облегчает программирование рекурсивных параллельных программ.

В табл. 1 приведено описание в БД ИПС основных операторных операций САА: композиции, альтернативы, цикла, асинхронной дизъюнкции и синхронизатора [9], реализации которых на языке TermWare используются в процессе генерации термов, а на языке Cilk++ – для генерации параллельных программ. Приведенные реализации содержат строки `^оператор1^` и `^оператор2^`, вместо которых в процессе генерации будут подставлены тексты реализаций операндов этих операций.

Таблица 1. Основные операторные операции САА и их реализации на языках TermWare и C++

Естественно-лингвистическая форма	Реализация на языке TermWare	Реализация на языке Cilk++
“оператор1” ЗАТЕМ “оператор2”	`then ( ^оператор1^, ^оператор2^)`	`^оператор1^; ^оператор2^`
ЕСЛИ “условие1” ТО “оператор1” ИНАЧЕ “оператор2” КОНЕЦ ЕСЛИ	`IF (^условие1^, ^оператор1^, ELSE (^оператор2^) )`	`if (^условие1^) { ^оператор1^ } else { ^оператор2^ }`
ПОКА НЕ ‘условие1’ ЦИКЛ “оператор1” КОНЕЦ ЦИКЛА	`WHILE_NOT (^условие1^, ^оператор1^ )`	`while (!(^условие1^)) { ^оператор1^ }`
("оператор1" ПАРАЛЛЕЛЬНО "оператор2")	`Parallel ( ^оператор1^, ^оператор2^ )`	`cilk_spawn ^оператор1^; ^оператор2^`
ЖДАТЬ 'условие1'	`WAIT (^условие1^)`	`cilk_sync;`

В табл. 2 приведены примеры описания в БД базисных элементов для рассмотренного алгоритма быстрой сортировки. Естественно-лингвистическая форма описания базисного элемента содержит в себе имена формальных параметров в скобках. Фактические параметры, которые задаются в САА-схемах, заменяют при генерации программы соответствующие формальные параметры в тексте реализации базисного понятия на языке программирования. Признаком формального параметра в реализации является символ “%”, за которым следует порядковый номер параметра в тексте базисного оператора или предиката. Например, в базисном операторе "Увеличить (var_id) на (value)" присутствует два параметра – идентификатор переменной и значение, на которое ее следует увеличить. Значение этих параметров будет подставлено вместо соответствующих формальных параметров в реализации этого оператора.

Таблица 2. Примеры описания в БД базисных операторов

Естественно-лингвистическая форма	Реализация на языке TermWare	Реализация на языке C++
"Определить массив (name) тип (type) размер (size)"	`Array(%1, %2, "%3")`	`%2* %1 = new %2[%3];`
"Определить переменную (x) типа (тип)"	`Variable(%1, %2)`	`%2 %1;`
"Увеличить (var_id) на (value)"	`Inc(%1, %2)`	`%1 = %1 + %2;`
"Перемешать массив (name) длины (size) случайным образом"	`ShuffleArrayRandomly (%1, %2)`	`std::random_shuffle (%1, %1 + %2);`

В результате интеграции системы TermWare и разработанного инструментария проектирования и синтеза программ, в TermWare передается термальное представление алгоритма, подлежащего трансформации и требуемые для этого равенства. Затем по преобразованному в TermWare алгоритму может быть произведена генерация соответствующей программы на языке программирования.

Далее в качестве примера приведен терм, сгенерированный по схеме последовательного алгоритма быстрой сортировки, рассмотренной в разделе 1.

```
Root (

main(
   then (
   Variable(n, int,
            "1000 * 10000"),
   then (
   IF (Greater(CmdLineParams,
               1),
       Assign(n, 1)),
   CALL(qmain(n)))),

qmain(Params(n),
   then (
   Locals
   (
     Array(a, int, "n"),
     Variable(i, int),
     Variable(end, int)
   ),
   then (
   ForLoop(i, 0, Minus(n, 1),
       AssignArrayElement
       (a, i, i)
   ),
   then (
   ShuffleArrayRandomly(a, n),
   then (
   START_TIME,
   then (
   then (
   Assign(end, Plus(a, n)),
sample_qsort(Params(begin, end),
   IF (NOT(Equal(begin, end)),
     then (
     Dec(end, 1),
     then (
     Partition(a, begin, end,
               end),
     then (
     CALL(sample_qsort(begin,
          middle)),
     then (
     Inc(middle, 1),
     then (
     Inc(end, 1),
     CALL(sample_qsort(middle,
                       end)
)))))))
)
)
```

Для распараллеливания приведенного алгоритма в функцию sample_qsort необходимо добавить операцию асинхронного выполнения операторов и синхронизатор (см. раздел 1). Для этого применяются следующие два правила:

```
1. then(CALL($x),
   then ($y, $z)) ->
   Parallel (
   CALL($x),
   then($y, $z))

2. then($x1,
   Parallel($x2, $x3)) ->
   then($x1,
   then(Parallel($x2, $x3),
   WAIT(AllThreadsCompleted(n))
   ))
```

В результате применения данных правил функция `sample_qsort` будет иметь вид:

```
sample_qsort( Params(begin,end),
IF(NOT(Equal(begin,end)),
   then (
   Dec(end,1),
   then(
   Partition(a,begin,end,end),
   then(

   Parallel(
     CALL(sample_qsort
         (begin,middle)),

     then(
     Inc(middle,1),
     then(
     Inc(end,1),CALL(sample_qsort
                  (middle,end))))
   ),

   WAIT(AllThreads
   Completed(n)))))))
```

Таким образом, в результате распараллеливания в первой ветви операции `Parallel` вызывается оператор `sample_qsort (begin,middle)`, а в другой – `sample_qsort(middle,end)`. Ветви создаются рекурсивно; их количество в алгоритме явно не указано, и задается при вызове программы. Синхронизатор `AIT(AllThreadsCompleted(n))` выполняет задержку вычислений до тех пор, пока все ветви не закончат вычисления. Как уже упоминалось, для реализации рассмотренного алгоритма сортировки был использован язык параллельного программирования Cilk++. Полученная программа была выполнена на многоядерной архитектуре Intel Core 2 Quad CPU, 2.51 ГГц. Время ее выполнения показано на рисунке. Ускорение при выполнении на 2, 3 и 4 процессорах составило 2; 2.9 и 3.8 соответственно, что свидетельствует о хорошей степени распараллеливаемости и масштабируемости вычислений.

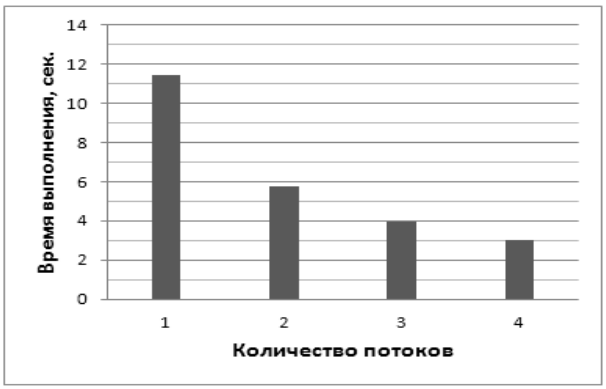

Рисунок. Время выполнения параллельной программы быстрой сортировки на 4-ядерном процессоре; размер входного массива $5 \cdot 10^7$ элементов

Заключение

Предложен подход к генерации термов по высокоуровневым спецификациям алгоритмов. Генерация выполняется в рамках совместного использования системы символьных вычислений TermWare и алгеброалгоритмического инструментария проектирования и синтеза программ ИПС. Система TermWare дополняет ИПС средствами преобразования последовательных и параллельных алгоритмов, направленных на их улучшение. Основой функционирования системы TermWare в рамках ИПС является создание и применение систем правил, каждая из которых соответствует некоторой трансформации схемы алгоритма. Отметим, что TermWare также позволяет расширить возможности ИПС в части автоматизации решения задач анализа и обеспечения надежности программного кода.

Перспективами дальнейших исследований в данном направлении является дополнение системы ИПС средствами обратного преобразования термов в схемы алгоритмов, а также настройка обоих систем на генерацию программ в других языках, поддерживающих распараллеливание.

1. *Система* автоматизации распараллеливания программ на промежуточном представлении LLVM. – http://www.dataved.ru/2011/06/llvmautoparallelizer.html.
2. *Multi-Threaded* Programming With POSIX Threads. – http://users.actcom.co.il/~choo./lupg/tutorials/multi-thread/multi-thread.html.
3. *OpenMP* Application Program Interface. – http://www.openmp.org/mp-documents/spec30.pdf.
4. *Intel* Threading Building Blocks. – http://threadingbuildingblocks.org
5. *Клинов М.С., Крюков В.А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер. – http://www.ict.edu.ru/vconf/files/11879.pdf.
6. *Дорошенко Е.А., Яценко Е.А.* О синтезе программ на языке Java по алгеброалгоритмическим спецификациям // Проблеми програмування. – 2006. – № 4. – С. 58–70.
7. *Дорошенко Е.А., Жереб К.А., Яценко Е.А.* Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. – 2007. – № 1. – С. 17–30.
8. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Об оценке сложности и координации вычислений в многопоточных программах // Проблеми програмування. – 2007. – № 2. – С. 41–55.

9. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
10. *Дорошенко А.Е., Шевченко Р.С.* Система символьных вычислений для программирования динамических приложений // Проблеми програмування. – 2005. – № 4. – С. 718–727.
11. *Кнут Д.* Искусство программирования для ЭВМ. – М.: Мир, 1978. – Т. 3. – 843 с.
12. *Guptha S.* Multithreaded Programming in a Microsoft Win32* Environment. – http://eng.harran.edu.tr/~nbesli/SP/senkronizasyon.pdf.
13. *Writing* Message-Passing Parallel Programs with MPI. – http://www.zib.de/zibdoc/mpikurs/mpicourse.pdf.
14. Intel Cilk++ SDK Programmer's Guide. – http://www.clear.rice.edu/comp422/resources/Intel_Cilk++_Programmers_Guide.pdf.

Получено 20.07.2012

***Об авторе*:**

Яценко Елена Анатольевна,
кандидат физико-математических наук,
старший научный сотрудник.

***Место работы автора*:**

Институт программных систем
НАН Украины.
Тел.: (044) 424 4972,
E-mail: oayat@ukr.net