УДК 681.3

В.Г. Акуловский, А.Е. Дорошенко, Е.А. Яценко

РЕАЛИЗАЦИЯ СРЕДСТВ ПРОЕКТИРОВАНИЯ И ГЕНЕРАЦИИ ПРОГРАММ НА ОСНОВЕ АЛГЕБРЫ АЛГОРИТМОВ С ДАННЫМИ

Предложено развитие алгеброалгоритмического инструментария проектирования и синтеза программ для конструирования спецификаций алгоритмов, сочетающих совместное описание данных и потоков управления в алгебре алгоритмов с данными. Применение предложенных алгебраического подхода и инструментария проиллюстрировано на примере разработки программ сортировки.

Введение

Актуальной проблемой современного программирования является его формализация, разработка высокоуровневых языков проектирования алгоритмов и программ на основе алгебраических средств. В Институте программных систем НАН Украины на протяжении длительного периода развивается теория, методология и инструментарий для формализованного проектирования параллельных программ, основывающиеся на средствах алгебры алгоритмики [1]. Алгебра алгоритмики представляет собой современное направление компьютерной науки, восходящее к фундаментальным работам академика В.М. Глушкова по теории систем алгоритмических алгебр (САА). Объектом исследования в ней являются высокоуровневые модели алгоритмов, представленные в виде схем. В предыдущих работах авторов [2–4] были предложены формальные средства для разработки эффективных программ в модифицированных САА (САА-М), ориентированных на формализацию последовательных и параллельных вычислений. На основе разработанной теории и методологии создан интегрированный инструментарий проектирования и синтеза программ (ИПС).

Учитывая принципиальную важность роли данных в процессе разработки алгоритмов и программ, в работах [5, 6] результате модификации модели ЭВМ Глушкова, предложена система алгоритмических алгебр с данными (САА\Д). Модификация заключается в дополнении модели внешней средой (ВС), представляющей собой с одной стороны память и внешние устройства, а с другой – множество данных D^{BC}. Сигнатура САА\Д включает все операции алгебры Глушкова. Принципиальное отличие предложенного формального аппарата от алгебры алгоритмов Глушкова состоит в использовании операторов вида $(D)O(D')$, на входе и выходе которых специфицированы обрабатываемые данные и которые, учитывая это, получили название Д-операторы. Данное отличие обеспечило возможность описания управляющих структур и структур обрабатываемых данных в процессе проектирования алгоритмов.

Цель данной работы состоит в настройке инструментальной системы ИПС на проектирование и синтез программ на основе схем, представленных в алгебре алгоритмов с данными. Применение алгебраического аппарата и инструментария проиллюстрировано на примере разработки последовательного и параллельного алгоритмов сортировки.

Предлагаемый подход близок к работам, относящимся к алгебраическому программированию [7] и синтезу программ на основе спецификаций [8, 9]. Отличие нашего подхода состоит в использовании вышеупомянутых Д-операторов, а также спецификаций алгоритмов, представленных не только в алгебраической, но и естественно-лингвистической форме, облегчающей понимание алгоритмов и достижение необходимого качества программ. Другим преимуществом разработанных инструментальных средств явля-

ется применение метода диалогового конструирования синтаксически правильных программ, исключающего возможность возникновения синтаксических ошибок в процессе проектирования схем.

1. Операции алгебры алгоритмов с данными

Алгебра алгоритмов с данными представляет собой трехосновную алгебраическую систему

$$\text{САА\Д} ::= \langle \{U, L, D\}; \Omega \rangle,$$

где U – множество Д-операторов, L – множество логических условий, D – множество данных, Ω – базовый набор операций алгебры, включающий операции Ω_1, определенные на множествах U, Ω_2, L, Ω_3 и D.

Д-операторы – это операторы вида $(D)O(D')$ со специфицированными на входе и выходе данными такими, что $D \subseteq D^{BC}$ и $D' \subseteq D^{BC}$. Д-оператор обрабатывает данные, т. е. анализирует и преобразует, изменяя их значения.

Предикат $(D^{\pi})P(\alpha)$ является одним из типов Д-операторов. Он осуществляет проверку заданного отношения D^{π} и продуцирует логическое условие $\alpha \in \{0, 1\}$, которое принимает истинное значение в случае, когда отношение D^{π} выполняется, и ложное в противном случае.

Представление любого Д-оператора посредством операций САА\Д и других Д-операторов называется регулярной схемой (РС) данного Д-оператора. Любой алгоритм может быть представлен в виде Д-оператора.

Рассмотрим операции САА/Д, определенные на множестве Д-операторов, и используемые в данной работе.

Композиция Д-операторов

$$(D_1)O_1(D_1') * (D_2)O_2(D_2') \qquad (1)$$

означает последовательное выполнение вначале Д-оператора $(D_1)O_1(D_1')$, а затем Д-оператора $(D_2)O_2(D_2')$.

Операция p_2-дизъюнкции

$$[(D^{\pi})P(\alpha)]((D_1)O_1(D_1') \vee (D_2)O_2(D_2')) =$$

$$= \begin{cases} (D_1)O_1(D_1'), \text{если } \alpha = 1; \\ (D_2)O_2(D_2'), \text{если } \alpha = 0, \end{cases}$$

состоит в выполнении Д-оператора $(D_1)O_1(D_1')$ в случае истинности условия α, или оператора $(D_2)O_2(D_2')$ в противном случае. Здесь $(D^{\pi})P(\alpha)$ – предикат, продуцирующий логическое условие α.

Операция последовательной фильтрации (p-фильтр) является производной от p_2-дизъюнкции и записывается в виде:

$$[(D^{\pi})P(\alpha)]((D)O(D')). \qquad (2)$$

Данная операция состоит в выполнении Д-оператора $(D)O(D')$ в случае истинности условия α.

Операция f-итерации

$$[(D_p)O^u(D_p'); (D_p')P(\alpha); (D_p)O^s(D_p')] \quad (3)$$
$$\{(D)O(D')\}$$

аналогична оператору for, который имеет место в большинстве языков программирования, и состоит в циклическом выполнении Д-оператора $(D)O(D')$ до тех пор, пока условие $\alpha = 1$, и завершается в противном случае. Здесь $(D_p)O^u(D_p')$ – Д-оператор, который устанавливает исходные значения данных D_p – параметров цикла; $(D_p)O^s(D_p')$ – Д-оператор, который изменяет их, как правило, с заданным шагом; $(D_p')P(\alpha)$ – предикат, который анализирует эти параметры, проверяя условие выполнения цикла; $(D)O(D')$ – тело цикла.

Пример 1. Частным случаем использования операции f-итерации может служить следующая РС:

$$[(0) := (i); i < n; (i, step) := (i)$$

$$\{(D_i)O(D_i')\},$$

где $(0) := (i)$ – базисный оператор, присваивающий переменной i исходное значение

0; $i < n$ – условие выполнения цикла; $(i, step) := (i)$ – оператор, увеличивающий значение переменной i на значение $step$; $(D_i)O(D_i')$ – тело цикла.

Для описания параллелизма в САА/Д используется операция асинхронной дизъюнкции Д-операторов $(D_i)O_i(D_i') \dot{\vee} (D_{i+1})O_{i+1}(D_{i+1}')$, состоящая в их параллельном асинхронном выполнении [6].

Для синхронизации параллельно выполняемых Д-операторов в РС используются синхронизаторы и контрольные точки.

Синхронизатор $(\phi)S$ задерживает вычислительный процесс до тех пор, пока условие ϕ, ложное в исходном состоянии, не станет истинным. Установка условия ϕ в истинное значение связана с прохождением контрольной точки, которая ассоциирована с данным синхронизатором.

Контрольная точка $(\phi)T(S)$, где S – ассоциированный с этой контрольной точкой синхронизатор, изменяет значение логического условия ϕ соответствующего синхронизатора, в результате чего "задержанный" синхронизатором вычислительный процесс продолжает выполнение.

В работе [6] сформулированы необходимые и достаточные условия для параллельного выполнения Д-операторов $(D_i)O_i(D_i') \dot{\vee} (D_{i+1})O_{i+1}(D_{i+1}')$ в РС:

1) в результате параллельного выполнения Д-операторов должны быть получены значения всех обрабатываемых данных $D_i, D_i', D_{i+1}, D_{i+1}'$ такие же, как и в случае их последовательного выполнения $(D_i)O_i(D_i') * (D_{i+1})O_{i+1}(D_{i+1}')$;

2) параллельно выполняемые Д-операторы должны быть синхронизованы, то есть завершение обоих этих Д-операторов должно быть синхронизовано с началом выполнения следующего за ними Д-оператора, а операции ввода-вывода, если они имеют место, в результате синхронизации должны принадлежать допустимой для данного алгоритма последовательности.

Учитывая приведенные условия, в работе [6] доказана следующая теорема о возможности параллельного выполнения Д-операторов.

Теорема. Два Д-оператора могут выполняться параллельно

$$(D_i)O_i(D_i') \dot{\vee} (D_{i+1})O_{i+1}(D_{i+1}'),$$

если они синхронизированы, и у них нет непосредственных $D_i' \cap D_{i+1} \neq \varnothing$ и обратных $D_i \cap D_{i+1}' \neq \varnothing$, $D_i' \cap D_{i+1}' \neq \varnothing$ связей, а последовательность операций ввода-вывода, если они имеют место, остается допустимой.

Определим далее операцию асинхронной f-итерации (параллельного цикла). Понятие параллельного цикла встречается во многих системах параллельного программирования (см., например, [10]), и, как правило, состоит в следующем. Пусть имеется цикл, витки (итерации) которого можно выполнять независимо, в любом порядке или одновременно. Тогда в параллельной программе можно распределить итерации такого цикла по отдельным параллельным процессам.

Запишем операцию асинхронной f-итерации в следующем виде:

$$\dot{\vee}[(start) := (i); i < end; (i, step) := (i)] \quad \{(D_i)O(D_i')\}. \qquad (4)$$

Данная операция состоит в параллельном выполнении витков f-итерации (цикла), указанной после символа $\dot{\vee}$. Здесь i – переменная цикла; $start$ – начальное значение переменной; $i < end$ – условие выполнения цикла; $(D_i)O(D_i')$ – тело цикла. Операторы $(D_i)O(D_i')$ должны удовлетворять условиям для параллельного выполнения операторов, рассмотренным выше.

Пример 2. Частным случаем использования операции асинхронной f-итерации может служить следующая схема алгоритма:

$$\dot{\vee}[(0) := (i); i < n; (i, 1) := (i)]$$

$$\{(D_i)O(D_i')\}.$$

Данная схема состоит в асинхронном выполнении совокупности Д-операторов

$(D_0)O(D'_0) \dot{\vee} (D_1)O(D'_1) \dot{\vee} ... \dot{\vee} (D_{n-1})O(D'_{n-1})$.

Целью данной работы является реализация средств САА\Д в инструментальной системе формализованного конструирования алгоритмов и программ, которая рассматривается в следующем разделе.

2. Инструментарий для проектирования алгоритмов и программ на основе САА\Д

В работах [1, 2] разработан инструментарий ИПС, обеспечивающий автоматизированное проектирование и синтез программ по формализованным спецификациям в САА-М. Инструментарий основывается на методе диалогового конструирования синтаксически правильных программ (ДСП-методе), и использует различные формы представления алгоритмов в процессе их проектирования – естественно-лингвистическую (САА-схемы), алгебраическую (регулярные схемы) и граф-схемную. В отличие от традиционных синтаксических анализаторов, ДСП-метод ориентирован не на поиск и исправление синтаксических ошибок, а на исключение возможности их появления в процессе построения алгоритма. Основная идея метода состоит в поуровневом конструировании схем сверху вниз путем суперпозиции языковых конструкций алгебры алгоритмов. На каждом шаге конструирования система в диалоге с пользователем предоставляет на выбор лишь те конструкции, подстановка которых в формируемый текст не нарушает синтаксическую правильность схемы. На основе построенной схемы алгоритма выполняется автоматическая генерация текста программы.

Отметим, что система ИПС может быть настроена на требуемый входной язык, представленный в алгебре алгоритмов, и выходной (целевой) язык программирования. В настоящее время система поддерживает генерацию последовательных и параллельных программ на языках C, C++, Java, а также термов на языке системы TermWare [11]. Для генерации параллельных программ были использованы различные средства: потоки с использованием функций WinAPI [2], операции MPI (Message Passing Interface) [3], операции языка C для CUDA [4] и др.

Архитектура инструментария показана на рис. 1. Основными компонентами системы являются:

• диалоговый конструктор синтаксически правильных программ (ДСП-конструктор), предназначенный для построения схем алгоритмов и генерации программ;

• редактор графового представления алгоритмов (граф-схем);

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

• база данных (БД), содержащая описание операций алгебры алгоритмов, базисных операторов и предикатов. Описание упомянутых элементов включает их представление в алгебраической и естественно-лингвистической формах, а также шаблон реализации на целевом языке программирования.

В основу функционирования ДСП-

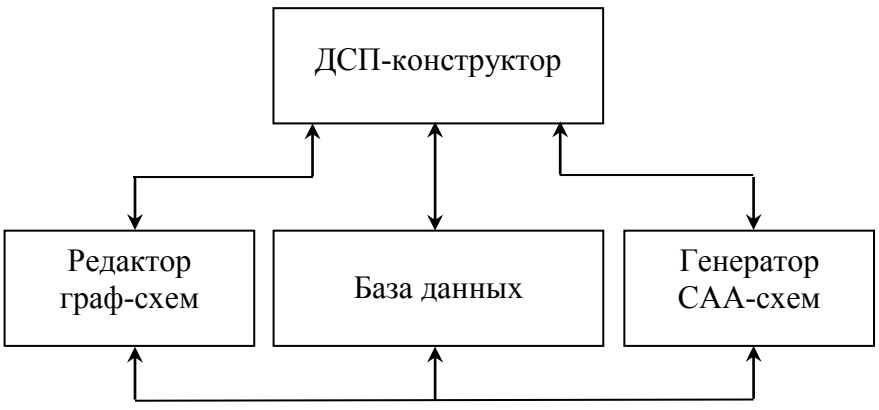

Рис. 1. Архитектура системы ИПС

конструктора положен диалоговый режим с использованием списка алгоритмических конструкций и дерева конструирования алгоритма. 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 (операторного или предикатного) система предлагает соответствующий список операций алгебры алгоритмов из базы данных. На основе полученного в процессе конструирования дерева алгоритма, инструментарий ИПС выполняет синтез кода на выбранном целевом языке программирования. В процессе синтеза управляющие конструкции схемы алгоритма и базисные операторы и предикаты отображаются в соответствующие операторы языка программирования в соответствии с их описанием (шаблоном) в БД. Составные операторы могут быть представлены как подпрограммы (функции).

В работах [1–4] система ИПС была применена для генерации многопоточных и распределенных параллельных программ на основе спецификаций алгоритмов, представленных в САА-М. В данной работе для ориентации инструментария на конструирование схем алгоритмов, представленных в САА\Д, в базу данных системы были добавлены описания операций, определения которых были рассмотрены в разделе 1. В табл. 1 приведены спецификации некоторых операторных операций САА\Д, которые были занесены в базу данных инструментария. Алгебраическая форма операций, по сравнению с приведенной в разделе 1 (см. формулы (1)–(4)), скорректирована с учетом текстового представления спецификаций в инструментарии. В частности, символ $\dot{\vee}$ в операции асинхронной f-итерации заменен на символы. Строками "условие1", “оператор1”, "переменная1" и т. п. обозначены различные операнды операций, вместо которых подставляются базисные предикаты, Д-операторы, идентификаторы переменных и прочее в процессе конструирования схемы алгоритма. Для каждой операции в таблице приведен шаблон

Таблица 1. Представление операторных операций САА\Д в алгебраической и естественно-лингвистической форме, а также их реализации на языке С++

№	Название операции	Алгебраическая форма	Естественно-лингвистическая форма	Шаблон реализации на языке С++
1	Композиция	“оператор1” * “оператор2”	“оператор1” ЗАТЕМ “оператор2”	^оператор1^; ^оператор2^

2	Последовательный фильтр	['condition1']
		"operator1"
			ЕСЛИ ‘условие1’
			ТО “оператор1”
			КОНЕЦ ЕСЛИ
				if (^условие1^)
				{
				^оператор1^
				};

3	f-итерация	["оператор1";
		'условие1';
		"оператор2"]
		{
		"оператор3"
		}
			ДЛЯ ("оператор1";
			'условие1';
			"оператор2")
			ЦИКЛ
			"оператор3"
			КОНЕЦ ЦИКЛА
				for (int ^оператор1^;
				^условие1^;
				^оператор2^)
				{
				^оператор3^
				}

4	Асинхронная f-итерация	//["оператор1";
		'условие1';
		"оператор2"]
		{
		"оператор3"
		}
			ПАРАЛЛЕЛЬНО ДЛЯ
			("оператор1"; 'условие1';
			"оператор2")
			"оператор3"
			КОНЕЦ ПАРАЛЛЕЛЬНО
				#pragma omp for
				for (int ^оператор1^;
				^условие1^;
				^оператор2^)
				{
				^оператор3^
				}

реализации на целевом языке программирования C++. Тексты реализаций содержат строки ^условие1^, ^оператор1^, ^переменная1^ и т. п., вместо которых в процессе генерации будут подставлены тексты реализаций операндов.

В последнем столбце четвертой строки таблицы приведен пример реализации операции асинхронной f-итерации на языке C++ с использованием средств OpenMP [10]. Отметим, что OpenMP представляет собой совокупность директив компилятора (прагм), библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. В данном случае для реализации параллельного цикла используется директива omp for, выполняющая распределение итераций цикла for между потоками.

Применение рассмотренных операций САА\Д для проектирования программ в системе ИПС продемонстрировано в следующем разделе.

3. Пример использования алгебраического аппарата и инструментария

Проиллюстрируем применение средств САА\Д и системы ИПС на примере распараллеливания алгоритма сортировки числовых массивов. В качестве исходного последовательного алгоритма рассматривается схема алгоритма сортировки методом пузырька.

В качестве обрабатываемых данных будем рассматривать массив целых чисел $\mathrm{M}[n]$, где n – константа, задающая размер массива. Доступ к элементам массива осуществляется с помощью индексных переменных, т. е. идентификатор элемента массива записывается в виде $\mathrm{M}[j]$, где M – идентификатор массива, j – индексная переменная. В частных случаях вместо индексной переменной может использоваться константа или выражение.

3.1. Последовательный алгоритм. Регулярная схема последовательного алгоритма пузырьковой сортировки, построенная в системе ИПС, имеет следующий вид:

$$
\begin{aligned}
&(\mathrm{M}[n])\ \mathrm{COPT}\ (\mathrm{M}[n]) = \\
&= [(0) := (i); (i < n); (i,1) + (i)] \\
&\quad \{ \\
&\qquad [(0) := (j); (j < n\text{-}1); (j,1) + (j)] \\
&\qquad \{ [\mathrm{M}[j] > \mathrm{M}[j+1]] \\
&\qquad\quad ((\mathrm{M}[j], \mathrm{M}[j+1]) \,\Pi\, (\mathrm{M}[j], \mathrm{M}[j+1]))\} \\
&\quad \}
\end{aligned}
$$

Данная схема представляет собой две вложенные операции f-итерации (два вложенных цикла). Кроме данных, специфицированных на входе алгоритма, использованы вспомогательные данные – индексные переменные i и j. Исходные значения переменных устанавливаются Д-операторами $(0) := (i)$ и $(0) := (j)$, на входе которых специфицированы константы, определяющие эти исходные значения. Индексные переменные изменяются с единичным шагом Д-операторами $(i,1) + (i)$ и $(j,1) + (j)$. В квадратные скобки заключены проверяемые отношения, а Д-оператор Π переставляет (меняет местами) элементы массива в случае, если в последовательном фильтре логическое условие принимает истинное значение.

Отметим, что по сравнению со схемами алгоритмов сортировки, представленных в САА (см. например, [1]), полученная выше РС содержит больший объем информации об алгоритме, и позволяет, в частности, более эффективно выполнять анализ информационных связей между операторами в них. Это преимущество обеспечивается за счет явного указания поименованных исходных и результирующих данных на входе и выходе Д-операторов.

3.2. Распараллеливание последовательного алгоритма. Для преобразования последовательного алгоритма COPT в параллельный выполним анализ информационных связей между Д-операторами $(\mathrm{M}[j], \mathrm{M}[j+1]) \,\Pi\, (\mathrm{M}[j], \mathrm{M}[j+1])$, последовательно выполняющимися во вложенном цикле по переменной j.

Рассмотрим, например, первые две итерации цикла, т. е. при $j = 0$ и $j = 1$. В

случае истинности условия $M[j] > M[j+1]$, при $j = 0$ будет выполнен Д-оператор $(M[0], M[1])\ П\ (M[0], M[1])$, а при $j = 1$ – $(M[1], M[2])\ П\ (M[1], M[2])$. Таким образом, множества входных и выходных данных этих операторов пересекаются по элементу массива $M[1]$. В соответствии с теоремой, приведенной в разделе 1, данные операторы имеют непосредственные и обратные связи, и не могут выполняться параллельно.

Одним из возможных вариантов распараллеливания последовательного алгоритма пузырьковой сортировки является его преобразование в алгоритм чёт-нечётной сортировки [12], а затем распараллеливание вложенного цикла [13, 14]. Суть модификации состоит в том, чтобы в зависимости от четности или нечётности номера итерации i внешнего цикла обрабатывать элементы с четными или нечётными индексами соответственно (при этом сравнение выделяемых значений осуществляется с их правыми соседними элементами).

Для преобразования алгоритма пузырьковой сортировки СОРТ в последовательный алгоритм чёт-нечётной сортировки выполним следующие замены (переинтерпретацию) базисных операторов:

$$(0) := (j) \rightarrow (i, 2)\,\%\,(j),$$
$$(j, 1) + (j) \rightarrow (j, 2) + (j),$$

где $(i, 2)\,\%\,(j)$ – Д-оператор, вычисляющий остаток от деления i на 2 и присваивающий полученное значение переменной j.

Полученная в результате указанных трансформаций РС имеет вид

$(M[n])$ СОРТ2 $(M[n]) =$
$[(0) := (i); (i < n); (i, 1) + (i)]$
$\{[(i, 2)\,\%\,(j); (j < n - 1); (j, 2) + (j)]$
$\{[M[j] > M[j+1]]$
$((M[j], M[j+1])\ П\ (M[j], M[j+1]))\}$
$\}$

Рассмотрим вновь данные, обрабатываемые операторами П при $j = 0$ и $j = 1$. В случае истинности условий фильтра, при $j = 0$ будет выполнен Д-оператор $(M[0], M[1])\ П\ (M[0], M[1])$, а при $j = 1$ – Д-оператор $(M[2], M[3])\ П\ (M[2], M[3])$.

Таким образом, множества входных и выходных данных в указанных Д-операторах не пересекаются, также как и в операторах при других значениях j во вложенном цикле. Операторы П являются независимыми и могут выполняться параллельно в рамках одной итерации внешнего цикла.

Для перехода к параллельному алгоритму чёт-нечётной сортировки необходимо выполнить следующие трансформации схемы СОРТ2:

1) заменить вложенную f-итерацию по переменной j на асинхронную f-итерацию;

2) добавить контрольные точки и синхронизаторы.

Результирующая параллельная РС имеет вид

$(M[n])$ СОРТ2_ПАР $(M[n]) =$
$[(0) := (i); (i < n); (i, 1) + (i)]$
$\{$
$\dot{\vee}[(i\,\%\,2) := (j); (j < n - 1); (j, 2) + (j)]$
$\{[M[j] > M[j+1]]$
$((M[j], M[j+1])\ П\ (M[j], M[j+1]))*$
$*(\text{ОБР\_ЗАКОНЧЕНА})\ T(j)(S)$
$\}*(\text{ВСЯ\_ОБР\_ЗАКОНЧЕНА})\ S$
$\}$

В приведенной схеме контрольная точка $(\text{ОБР\_ЗАКОНЧЕНА})\ T(j)(S)$ фиксирует момент завершения вычислений в текущей параллельной ветви. Синхронизатор $(\text{ВСЯ\_ОБР\_ЗАКОНЧЕНА})\ S$ выполняет задержку вычислений в теле внешнего цикла до тех пор, пока обработка во всех параллельных ветвях не будет завершена.

Естественно-лингвистическая форма данного алгоритма, построенного в системе ИПС, имеет следующий вид:

"(M[n]) СОРТ2_ПАР (M[n])" ====

ДЛЯ ((0) := (i); (i < n); (i, 1) + (i))
ЦИКЛ

```
ПАРАЛЛЕЛЬНО ДЛЯ
    ((i, 2) % (j); (j < n - 1); (j, 2) + (j))

  ЕСЛИ (M[j] > M[j+1])
  ТО
    "(M[j], M[j+1]) ПЕРЕСТАВИТЬ
     (M[j], M[j+1])"
   КОНЕЦ ЕСЛИ
   ЗАТЕМ
   ('Обработка в текущей ветви
    закончена') КТ(j) (S)

КОНЕЦ ПАРАЛЛЕЛЬНО
ЗАТЕМ
('Обработка во всех ветвях
  закончена') S
КОНЕЦ ЦИКЛА
```

На рис. 2 показана копия экрана ДСП-конструктора инструментария ИПС с построенной схемой СОРТ2_ПАР. Окно ДСП-конструктора состоит из трех вложенных элементов-окон. Верхнее левое окно предназначено для отображения и выбора алгоритмических конструкций и вставки их в дерево алгоритма. Дерево алгоритма отображается в окне, находящемся в правой части ДСП-конструктора. Третье окно, находящееся слева внизу, предназначено для отображения текста алгоритма в процессе его конструирования в естественно-лингвистической (САА-схема) или алгебраической форме (формула), в зависимости от установленной опции. В данном случае текст алгоритма представлен в алгебраической форме.

3.3. Генерация текста на языке программирования и результаты запуска подпрограмм. На основе схем СОРТ и СОРТ2_ПАР в системе ИПС была выполнена генерация кода подпрограмм на языке C++ – SORT и SORT2_PAR, соответственно. Текст подпрограммы SORT2_PAR сгенерирован с использованием средств OpenMP, и имеет следующий вид:

```
void SORT2_PAR(int *M, int n)
{
   #pragma omp parallel
   {
      for (int i = 0; i < n; i = i + 1)
      {
        #pragma omp for
```

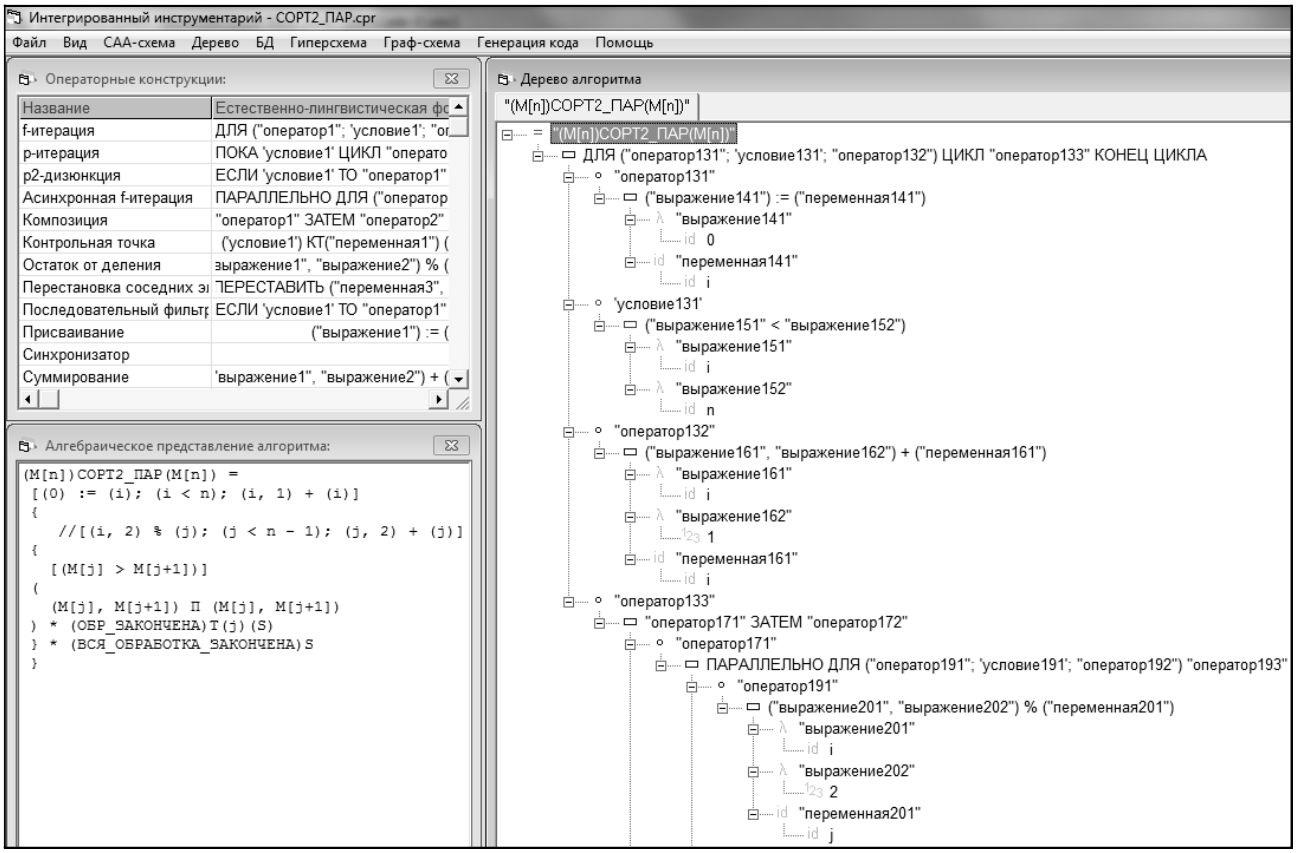

Рис. 2. Фрагмент копии экрана системы ИПС с алгоритмом СОРТ2_ПАР

```
        for (int j = i % 2; j < n - 1; j = j + 2)
        {
            if (M[j] > M[j+1])
            {
                int tmp = M[j];
                M[j] = M[j+1];
                M[j+1] = tmp;
            }
        }
    }
}
```

Для реализации асинхронной f-итерации в приведенной подпрограмме использована директива omp for, выполняющая распределение итераций цикла for между потоками в параллельной области программы. По умолчанию в этой директиве используется блочное распределение итераций цикла, т. е., если количество итераций цикла равно m, а количество параллельных потоков $-p$, то поток с номером 0 выполнит первые m/p итераций цикла, следующие m/p итераций будут назначены потоку 1, и т. д. Отметим, что в конце параллельного цикла OpenMP осуществляет неявную синхронизацию потоков, и поэтому специально реализовывать контрольные точки и синхронизаторы в данном случае нет необходимости. Параллельная область программы обозначена с помощью директивы omp parallel, выполняющей создание потоков [10]. Заметим, что в приведенном тексте директива omp parallel приведена перед внешним циклом, а не внутри его (перед директивой omp for), так как такой подход позволяет избежать системных издержек, связанных с многократным созданием потоков в этом цикле [13, 14].

Проведен эксперимент по выполнению разработанных подпрограмм SORT и SORT2_PAR на двух 8-ядерных процессорах Intel Xeon E5-2670 (2.60 ГГц), которые входят в состав суперкомпьютерного вычислительного комплекса СКИТ-4 Института кибернетики имени В.М. Глушкова НАН Украины [15]. Эксперимент выполнен для размера массивов n от 32000 до 576000 элементов.

На рис. 3 показан график зависимости времени выполнения (в секундах) от размера входного массива n для следующих подпрограмм:

1) подпрограммы последовательной пузырьковой сортировки SORT;

2) параллельной подпрограммы чёт-нечётной сортировки SORT2_PAR при количестве потоков 1;

3) подпрограммы SORT2_PAR, выполненной при количестве потоков 16.

Как видно из графика, программа SORT2_PAR в обоих случаях является бо-

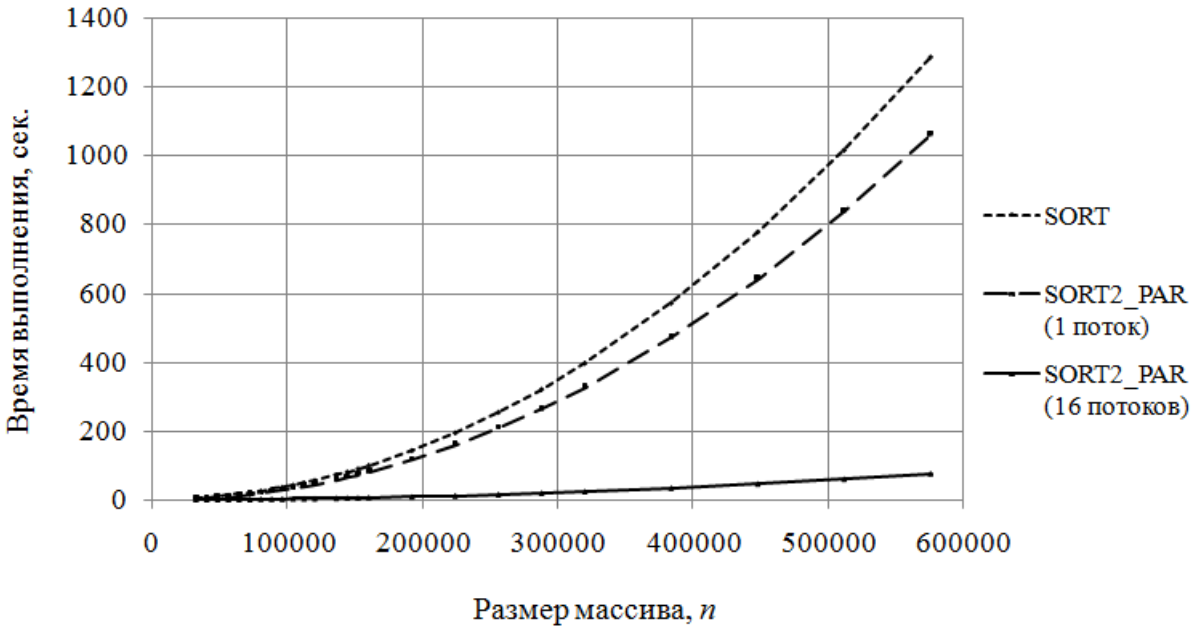

Рис. 3. Зависимость времени выполнения подпрограмм сортировки от размера массива n

лее эффективной по времени выполнения, чем пузырьковая сортировка SORT.

На рис. 4 показан график зависимости мультипроцессорного ускорения для параллельной подпрограммы SORT2_PAR от размера массива *n*. Ускорение вычислено по формуле $Sp = \frac{T_1}{T_{16}}$, где T_1 – время выполнения SORT2_PAR с 1 потоком; T_{16} – время выполнения SORT2_PAR с 16 потоками. Максимальное значение ускорения при $n = 576000$ составляет $Sp = 14.24$. Эффективность использования процессоров при этом равна $E = \frac{Sp}{16} = 0.89$.

Отметим также, что по сравнению с подпрограммой последовательной пузырьковой сортировки SORT, подпрограмма чёт-нечётной сортировки SORT2_PAR при использовании 16 потоков и указанном выше максимальном значении *n* выполняется быстрее в 17.26 раз.

Таким образом, система ИПС обеспечивает построение спецификаций последовательных и параллельных алгоритмов, представленных в алгебре алгоритмов с данными, а также выполняет генерацию соответствующего текста на целевом языке программирования. Преимуществом использования средств САА\Д в системе ИПС является возможность анализа связей между операторами в схемах для их последующей трансформации (например, распараллеливания). Так, в данном разделе на примере задачи сортировки показан переход от последовательного алгоритма к параллельному. Выполнены преобразования исходного алгоритма к его последующему варианту, содержащему меньшее количество связей, а затем введение параллельных конструкций.

Заключение

Предложено развитие алгеброалгоритмического инструментария проектирования и синтеза программ для конструирования спецификаций алгоритмов, сочетающих совместное описание данных и потоков управления на основе использования алгебры алгоритмов с данными. Преимуществом использования спецификаций, основанных на САА\Д, в сравнении со схемами, представленными в САА, состоит в использовании операторов, на входе и выходе которых специфицированы обрабатываемые данные. За счет этого в САА\Д более детально представлены информационные связи между частями алгоритма, позволяющие выполнять анализ и преобразования схем с целью их улучшения (в частности, распараллеливания).

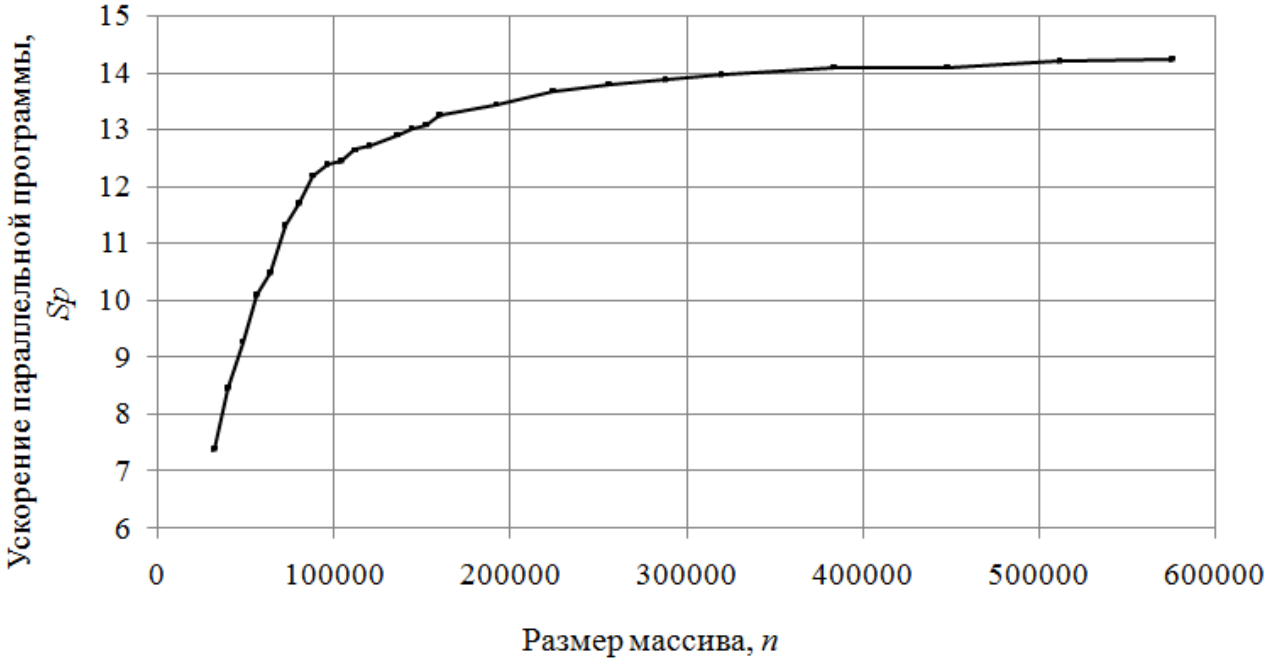

Рис. 4. Зависимость мультипроцессорного ускорения *Sp* параллельной подпрограммы SORT2_PAR от размера массива *n*

Применение предложенного в рамках САА\Д подхода к проектированию алгоритмов и инструментария проиллюстрировано на примере разработки подпрограмм сортировки массивов. Проведен эксперимент по выполнению разработанных подпрограмм на мультипроцессорном кластере, результаты которого демонстрируют хороший показатель эффективности распараллеливания вычислений.

1. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
2. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. – 2007. – № 1. – С. 17–30.
3. *Дорошенко А.Е., Яценко Е.А., Жереб К.А.* Средства синтеза параллельных MPI-программ // Проблеми програмування. – 2008. – № 2–3. – С. 595–604.
4. *Дорошенко А.Ю., Бекетов О.Г., Жереб К.А., Яценко О.А.* Формалізоване проектування та синтез паралельних програм для відеографічних прискорювачів // Проблеми програмування. – 2013. – № 3. – С. 38–46.
5. *Акуловский В.Г.* Алгебра алгоритмов, базирующаяся на данных // Кибернетика и системный анализ. – 2012. – № 2. – С. 151–166.
6. *Акуловский В.Г., Дорошенко А.Е.* Описание параллелизма в алгоритмах информационно-управляющих систем средствами алгебраического аппарата // Проблеми програмування. – 2013. – № 3. – С. 13–21.
7. *Sannella D., Tarlecki A.* Foundations of algebraic specification and formal software development. – Berlin: Springer-Verlag, 2012. – 594 p.
8. *Flener P.* Achievements and prospects of program synthesis // Lecture Notes in Artificial Intelligence. – 2002. – Vol. 2407. – P. 310–346.
9. *Gulwani S.* Dimensions in program synthesis // Proc. 12th Int. ACM SIGPLAN symposium on Principles and Practice of Declarative Programming, Hagenberg, Austria (26–28 July, 2010). – New York: ACM, 2010. – P. 13–24.
10. *OpenMP* Application Program Interface. – http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.
11. *Doroshenko A., Shevchenko R.* A rewriting framework for rule-based programming dynamic applications // Fundamenta Informaticae. – 2006. – 72, N 1–3. – P. 95–108.
12. *Odd–even* sort. – http://en.wikipedia.org/wiki/Odd–even_sort
13. *Pacheco P.* More about loops in OpenMP: sorting // An introduction to parallel programming. – Burlington: Morgan Kaufmann, 2011. – P. 232–236.
14. *Parallel* Computing Lecture 15: OpenMP III. – http://cs.nyu.edu/courses/spring14/CSCI-UA.0480-003/lecture15.pdf
15. *Суперкомп'ютер* ІК НАН України. – http://icybcluster.org.ua.

Получено 11.02.2015

Об авторах:

Акуловский Валерий Григорьевич,
кандидат технических наук,
доцент кафедры информационных систем и технологий,

Дорошенко Анатолий Ефимович,
доктор физ.-мат. наук, профессор,
заведующий отделом теории компьютерных вычислений,

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

Место работы авторов:

Академия таможенной службы Украины.
49000, Днепропетровск,
ул. Дзержинского 2/4.
E-mail: academy@amsu.dp.ua,
valeryakulovskiy@rambler.ru

Институт программных систем
НАН Украины.
03187, Киев,
проспект Академика Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: doroshenkoanatoliy2@gmail.com,
oayat@ukr.net