УДК 510.66:004.272.2

А.А. ЛЕТИЧЕВСКИЙ, В.Н. ГЕРМАН, М.К. МОРОХОВЕЦ, Н.Н. ЩЕГОЛЕВА

ПАРАЛЛЕЛЬНЫЙ ПОИСК ВЫВОДА В ЛОГИЧЕСКОМ ИСЧИСЛЕНИИ НА ОСНОВЕ СИСТЕМЫ АЛГЕБРАИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Ключевые слова: *система алгебраического программирования, правила переписывания, алгоритм очевидности, поиск логического вывода, параллельные вычисления, кластер*.

ВВЕДЕНИЕ

Появление все более мощных современных электронных средств вычислений, в частности таких, как параллельные вычислительные машины, оказывает влияние на развитие математического обеспечения решения различных задач, в том числе тех, которые традиционно относятся к области искусственного интеллекта. Одним из направлений этой области является автоматизация поиска доказательств теорем (АДТ). Поскольку оно возникло раньше, чем появились параллельные ЭВМ, ко времени внедрения средств параллельных вычислений были созданы программные системы автоматизированного доказательства теорем, функционировавшие на последовательных ЭВМ. Перед разработчиками АДТ встала задача создания таких программных систем, которые бы эффективно работали на параллельных машинах.

В рамках двух базовых стратегий распараллеливания поиска решения задач — И-параллелизма и ИЛИ-параллелизма — исследователями в области АДТ рассматривались разные схемы параллельных вычислений. В настоящее время различается распараллеливание вычислений при поиске доказательства теорем на уровне: обработки термов, обработки предложений, организации поиска в целом. Одним из видов распараллеливания вычислений на уровне обработки термов является параллельное выполнение унификации (например, [1, 2]) обработки предложений — применение одновременно нескольких правил вывода исчисления при поиске логического вывода или одного правила вывода к нескольким совокупностям формул-посылок (например, [3]) организации поиска — одновременное использование для решения одной задачи нескольких стратегий поиска доказательства теорем (одинаковых или различных) в режиме сотрудничества или режиме соперничества (например, [4]). Опыт разработки и использования параллельных систем АДТ показал, что на данном этапе развития вычислительных средств наиболее перспективным является распараллеливание на уровне организации поиска [5].

В настоящей статье описаны средства параллельного поиска доказательств теорем, разработанные на базе системы алгебраического программирования (АПС). Они создавались с учетом особенностей доступного технического обеспечения параллельных вычислений — кластерного комплекса СКИТ-1. Исследования имели целью, с одной стороны, создать параллельную версию доказывателя теорем (для формул логики высказываний), разработанного ранее на базе АПС, с другой — найти класс задач, схема решения которых путем параллельных вычислений хорошо сочеталась бы с архитектурой вычислительного комплекса СКИТ-1, и, кроме того, изучить экспериментально зависимость времени решения задач рассматриваемого класса от числа используемых процессоров. В работе анализируются результаты экспериментов с программой параллельного поиска доказательств теорем. Дальнейшее изложение организовано следующим образом. Приведены краткие сведения об АПС, описано базовое исчисление, использованное для построения системы АДТ, даны краткие сведения о кластерном комплексе СКИТ-1, на базе ко-

торого проводились эксперименты, представлено обоснование выбора класса задач для проведения экспериментов, описана организация параллельного доказательства теорем, приведены результаты экспериментов и их анализ.

СИСТЕМА АЛГЕБРАИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Система алгебраического программирования [6] служит средой для разработки прототипов прикладных программных систем. Она наиболее приспособлена для предметных областей, представленных алгебро-логическими моделями. Техника программирования в АПС базируется на переписывании термов. В отличие от традиционного подхода, ориентированного на использование канонических систем правил переписывания с «очевидной» стратегией их применения, в АПС возможно сочетание любых систем правил переписывания и разнообразных стратегий переписывания. Такой подход значительно расширяет возможности техник переписывания, поскольку возрастает их гибкость и выразительность. В структуре АПС можно выделить три способа структуризации используемых объектов:

- основные типы системных объектов;
- базовые вычислительные механизмы;
- допустимые языковые конструкции, объединенные в семейство языков АПЛАН.

Основным типом данных в системе является терм (представленный в виде дерева) в алгебре $T_\Omega(Z)$, порожденной множеством первичных объектов Z и операциями сигнатуры Ω. Эта алгебра рассматривается как абсолютно свободная. В системе три основных типа объектов: алгебраические программы (ап-модули), алгебраические модули (а-модули) и интерпретаторы. Алгебраические программы — это тексты в языке АПЛАН. Каждая программа содержит описание сигнатуры Ω с синтаксисом для построения алгебраических выражений. В ней определяются множества имен X и атомов A. Алгебраическая программа определяет также начальные значения имен.

Алгебраические модули содержат внутренние представления структур данных, определенных в ап-модулях. Понятие алгебраического модуля имеет динамический характер: модуль находится в состоянии, которое со временем может меняться; изменение состояния а-модуля происходит в том случае, когда при интерпретации выполняется какая-либо его процедура.

Интерпретаторы используются для интерпретации процедурного подмножества языка АПЛАН.

Техника программирования в АПС базируется на переписывании термов. В АПС реализовано несколько стратегий переписывания, кроме того, арсенал стратегий можно пополнять.

Система алгебраического программирования организована таким образом, что пользователь имеет одновременный доступ ко всем уровням программирования, начиная с языка конкретной предметной области и заканчивая уровнем расширения языка си, которое называется L2C. Таким путем достигается эффективное распределение сложности разрабатываемой программной системы по различным уровням. Поддерживается эволюционность процесса программирования, начиная с выполняемой алгебраической спецификации через оптимизирующие преобразования до эффективной программы на языке си. Самый верхний и специализированный уровень образуют конструкции, связанные с некоторой конкретной предметной областью. Они имеют локальный характер, но также активно используются в других предметных областях. Это связано с тем, что в базовом АПЛАНе нет типов данных. Строго говоря, существует единственный тип данных — алгебраический терм. В то же время имеются возможности для использования строгого типизирования, например при программировании в объектно-ориентированном расширении АПЛАНа.

В качестве примера использования техники переписывания для решения задач, а также языковых конструкций языка АПЛАН рассмотрим простую задачу функционального программирования — вычисление чисел Фибоначчи. Известное рекурсивное определение n-го числа Фибоначчи задается следующей системой соотношений:

$$F(0)=1,\ F(1)=1,\ F(n)=F(n-1)+F(n-2).$$

Данную систему можно рассматривать как (рекурсивное) определение функции F или систему правил переписывания, которую можно использовать для вычисления $F(n)$. Чтобы записать ее на языке АПЛАН, следует импортировать модуль `std.ap`:

```
INCLUDE  <std.ap>.
```

Эта команда дает возможность включить некоторые стандартные определения, в частности, обеспечивает использование арифметических операций «+», «−», отношения «=», а также некоторых других синтаксических понятий, определенных в модуле `std.ap`. Затем следует определить имя системы (с помощью предложения `NAME R;`) и присвоить начальные значения:

```
R:=rs(n)(
    F(0) = 1,
    F(1) = 1,
    F(n) = F(n - 1) + F(n - 2)
);
```

Первая строка присваивания указывает, что значением имени `R` является система правил переписывания (`rs`), а `n` — единственная переменная, используемая системой.

Систему `R` можно применить, например, к выражению $T=F(10)$ и преобразовать его таким образом:

$$F(10)=F(10-1)+F(10-2)=F(9)+F(8).$$

В АПС данное преобразование выполняется за один шаг, так как арифметические операции являются интерпретируемыми и выполняются по мере возможности на каждом шаге переписывания. Следующий шаг переписывания может быть выполнен одним из двух способов, в зависимости от того, к которому из вхождений выражения $F(n)$ применяется третье правило системы. Рассмотрим случай, когда выбирается первое вхождение, т.е. $F(9)$. Тогда имеем новое выражение

$$T=(F(8)+F(7))+F(8).$$

Продолжая подобным образом, т.е. выбирая всякий раз самое левое (самое внутреннее) вхождение, получаем

$$T=((F(7)+F(6))+F(7))+F(8)=$$
$$=(((F(6)+F(5))+F(6))+F(7))+F(8)=$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$=(\ldots((F(1)+F(0))+F(1))+\ldots)+F(8).$$

Теперь применимы как третье, так и второе правила. Будем действовать по принципу, который используется во всех основных стратегиях переписывания в АПС. Он состоит в том, что правило, которое фактически применяется, является первым из правил, используемых на текущем этапе вычислений. Таким образом, последующие два шага преобразований приведут к выражениям

$$T=(\ldots((1+F(0))+F(1))+\ldots)+F(8)=(\ldots((1+1)+F(1))+\ldots)+F(8).$$

В соответствии с левосторонней стратегией в первую очередь следует выполнить сложение. В результате последовательности переписываний согласно левосторонней стратегии получим

$$T = 55 + 34 = 89.$$

Для данной системы тот факт, что при переписывании выбирается лишь самое левое вхождение, не существен. Чтобы получить требуемый результат с помощью системы правил R, достаточно использовать любую стратегию переписывания (алгоритм переписывания), удовлетворяющую следующим условиям.

1. На каждом шаге переписывания применяется одно из правил системы либо выполняется арифметическая операция.

2. Выбор правила осуществляется согласно последовательности, в которой записаны правила.

3. Переписывание продолжается до тех пор, пока это возможно, т.е. пока существуют вхождения, к которым применимы правила системы, либо возможны выполнения арифметических операций над числами.

Стратегия, удовлетворяющая приведенным условиям, называется финальной. АПС позволяет использовать встроенные стратегии переписывания, а также писать собственные стратегии, которые наиболее подходят для решаемой задачи. Вызов встроенной стратегии является вызовом внутренней процедуры. Как правило, он имеет вид $s(T, R)$, где s — имя стратегии, T — имя алгебраической структуры данных, к которой применяется система, R — имя системы правил переписывания.

В рамках системы АПС разработан целый ряд прикладных программ: средства обработки формул логики высказываний (приведение пропозициональных формул к КНФ, ДНФ, проверка тождественной истинности формулы), приведение полиномов, решатель задач общего назначения.

Первоначально АПС разрабатывалась в среде MS DOS. В настоящее время существуют версии АПС, предназначенные для работы под управлением различных современных операционных систем — как Linux, Solaris, так и Windows.

ЛОГИЧЕСКОЕ ИСЧИСЛЕНИЕ

Средства поиска доказательств теорем в АПС основаны на исчислении алгоритма очевидности [7]. В основу исчисления положена идея В.М. Глушкова об алгоритме очевидности [8] как формализации рассуждений практического математика. При реализации этого исчисления использовано представление, предложенное в [9]. Элементарными объектами исчисления являются пропозициональные формулы и простые секвенции. Пропозициональные формулы рассматриваются с точностью до эквивалентности, которая определяется с помощью всех соотношений булевой алгебры за исключением дистрибутивности (поскольку последняя является источником экспоненциальной сложности). В системе АПС существует функция Can, которая используется для приведения логических формул к нормальной (но не канонической) форме, что сохраняет эту эквивалентность. Ассоциативность, коммутативность и идемпотентность конъюнкции и дизъюнкции, а также законы противоречия, исключенного третьего и законы для пропозициональных констант применяются неявным образом, будучи встроенными в стратегию Can_ord. Для построения простых секвенций используется синтаксис $x \Rightarrow y$, где x и y — пропозициональные формулы.

Исчисление алгоритма очевидности задается в виде комбинации двух исчислений. Одним из них является исчисление вспомогательных целей (ИВЦ), другим — исчисление условных секвенций (ИУС). Вывод в исчислении вспомогательных целей имеет финитный характер, он всегда завершается и используется как один шаг в исчислении условных секвенций, которое является основным. Заметим, что в некотором смысле исчисление алгоритма очевидности обобщает поиск вывода в логическом программировании, который можно рассматривать как поиск вариантов сведения задачи к подзадачам (вспомогательным целям). Отличие состоит лишь в том, что вспомогательные цели в логическом программировании извлекаются из

хорновских дизъюнктов, а в исчислении алгоритма очевидности они могут извлекаться из произвольных формул.

Исчисление условных секвенций. Условная секвенция — выражение вида (w,Q), где w — конъюнкция литералов (для пропозиционального исчисления литерал — пропозициональная переменная либо ее отрицание), Q — конъюнкция простых секвенций.

Аксиомы исчисления:

$$(w,1);\ \ (w,u\Rightarrow 1);\ \ (w,0\Rightarrow P);\ \ (0,Q).$$

Здесь P — конъюнкция простых секвенций, единица означает пустую конъюнкцию.

Правила вывода:

(ИУС.1) $(w,u\Rightarrow 0)\vdash(w,1\Rightarrow\neg u)$,

(ИУС.2) $(w,u\Rightarrow x\wedge y)\vdash(w,(u\Rightarrow x)\wedge(u\Rightarrow y))$,

(ИУС.3) $(w,u\Rightarrow x\vee y)\vdash(w,\neg x\wedge u\Rightarrow y)$,

(ИУС.4) $(w,x\wedge y\Rightarrow z)\vdash(w\wedge x,y\Rightarrow z)$, где x — литерал,

(ИУС.5) $\dfrac{(w,F)\vdash(w',F')}{(w,F\wedge H)\vdash(w,H)}$, где (w',F') — аксиома,

(ИУС.6) $\dfrac{\mathrm{aux}(1,w\wedge u\Rightarrow z,1)\vdash\mathrm{aux}(v,z\wedge y\Rightarrow z,P)}{(w,u\Rightarrow z)\vdash(w\wedge\neg z,P)}$,

где z — литерал, P — конъюнкция простых секвенций (правило вспомогательной цели).

Исчисление вспомогательных целей. Вспомогательной целью назовем выражение вида $\mathrm{aux}(v,u\Rightarrow z,P)$, где z — литерал, u,v — пропозициональные формулы, P — конъюнкция простых секвенций (пустая конъюнкция равна 1). Правила исчисления вспомогательных целей:

(ИВЦ.1) $\mathrm{aux}(v,x\wedge y\Rightarrow z,P)\vdash\mathrm{aux}(v\wedge y,x\Rightarrow z,P)$,

(ИВЦ.2) $\mathrm{aux}(v,x\vee y\Rightarrow z,P)\vdash\mathrm{aux}(v,x\Rightarrow z,(v\Rightarrow\neg y)\wedge P)$.

В этих правилах конъюнкции и дизъюнкции рассматриваются с точностью до коммутативности, поэтому нет необходимости рассматривать альтернативные правила, которые образуются путем перестановок x и y.

Свойство завершимости исчисления вспомогательных целей следует из утверждения $\mathrm{aux}(v,u\Rightarrow z,P)\vdash\mathrm{aux}(v',u'\Rightarrow z,P')$, где u' — литерал.

Основным свойством исчисления алгоритма очевидности является следующее:

P — тавтология тогда и только тогда, когда $(1,1\Rightarrow P)\vdash Q$, где Q — аксиома.

Пример построения вывода в исчислении алгоритма очевидности. Пусть требуется доказать, что в исчислении алгоритма очевидности выводима формула

$$(a2\wedge a1\vee\neg a2\wedge a1\vee\neg a1)\wedge(a1\vee\neg a1). \qquad (1)$$

Согласно основному свойству этого исчисления достаточно показать, что $(1,1\Rightarrow(a2\wedge a1\vee\neg a2\wedge a1\vee\neg a1)\wedge(a1\vee\neg a1))\vdash Q$, где Q — аксиома. Запишем вывод, указывая в скобках справа от формулы название правила, которое применяется для ее вывода. При выводе учитывается, что пропозициональные формулы рассматриваются с точностью до некоторой эквивалентности. Итак, имеем

$(1,1\Rightarrow(a2\wedge a1\vee\neg a2\wedge a1\vee\neg a1)\wedge(a1\vee\neg a1))$ (исходная условная секвенция)

$(1,(1\Rightarrow(a2\wedge a1\vee\neg a2\wedge a1\vee\neg a1))\wedge(1\Rightarrow(a1\vee\neg a1)))$ (ИУС.2)

Далее применимо ИУС.5, т.е. если будет доказано, что $(1,1\Rightarrow(a2\wedge\wedge a1\vee\neg a2\wedge a1\vee\neg a1))\vdash(w',F')$, где (w',F') — аксиома, то останется проверить выводимость условной секвенции

$$(1,1\Rightarrow(a1\vee\neg a1)) \qquad (2)$$

$(1,1\Rightarrow(a2\wedge a1\vee\neg a2\wedge a1\vee\neg a1))$ (ИУС.5, начало)

$(1,\neg(a2\wedge a1)\Rightarrow\neg a2\wedge a1\vee\neg a1)$ (при $x=a2\wedge a1$, $y=\neg a2\wedge a1\vee\neg a1$ ИУС.3)

$(1,\neg a2\vee\neg a1\Rightarrow\neg a2\wedge a1\vee\neg a1)$ (замена $\neg(a2\wedge a1)$ на $\neg a2\vee\neg a1$)

$(1,(\neg a2\vee\neg a1)\wedge(a2\vee\neg a1)\Rightarrow\neg a1)$ (ИУС.3)

$\mathrm{aux}\,(1,(\neg a2\vee\neg a1)\wedge(a2\vee\neg a1)\Rightarrow\neg a1,1)$ (ИУС.6, начало)

$\mathrm{aux}\,((\neg a2\vee\neg a1),a2\vee\neg a1\Rightarrow\neg a1,1)$ (ИВЦ.1)

$\mathrm{aux}\,((\neg a2\vee\neg a1),\neg a1\Rightarrow\neg a1,\neg a2\vee\neg a1\Rightarrow\neg a2)$ (ИВЦ.2)

$(a1,\neg a2\vee\neg a1\Rightarrow\neg a2)$ (ИУС.6, конец)

$\mathrm{aux}\,(1,a1\wedge(\neg a2\vee\neg a1)\Rightarrow\neg a2,1)$ (ИУС.6, начало)

$\mathrm{aux}\,(a1,\neg a2\vee\neg a1\Rightarrow\neg a2,1)$ (ИВЦ.1)

$\mathrm{aux}\,(a1,\neg a2\Rightarrow\neg a2,a1\Rightarrow a1)$ (ИВЦ.2)

$(a1\wedge a2,a1\Rightarrow a1)$ (ИУС.6, конец)

$\mathrm{aux}\,(1,a1\wedge a2\wedge a1\Rightarrow a1,1)$ (ИУС.6, начало)

$\mathrm{aux}\,(1,a1\wedge a2\Rightarrow a1,1)$ (замена $a1\wedge a2\wedge a1$ на $a1\wedge a2$)

$\mathrm{aux}\,(a2,a1\Rightarrow a1,1)$ (ИВЦ.1)

$(a1\wedge a2\wedge\neg a1,1)$ (ИУС.6, конец; аксиома)

Последняя секвенция является аксиомой. Осталось проверить выводимость условной секвенции $(1,1\Rightarrow(a1\vee\neg a1))$. Имеем:

$(1,(1\Rightarrow(a1\vee\neg a1)))$ (условная секвенция (2))

$(1,a1\Rightarrow a1)$ (ИУС.3)

$\mathrm{aux}\,(1,a1\Rightarrow a1,1)$ (ИУС.6, начало)

$(\neg a1,1)$ (ИУС.6, конец; ИУС.5, конец; аксиома)

Таким образом, вывод формулы (1) построен.

ТЕХНИЧЕСКИЕ СРЕДСТВА И ОПЕРАЦИОННАЯ СРЕДА ПРОГРАММЫ ПАРАЛЛЕЛЬНОГО ПОИСКА ЛОГИЧЕСКОГО ВЫВОДА

Базой для экспериментов с программой поиска доказательств послужила мультипроцессорная суперкомпьютерная система кластерного типа СКИТ (суперкомпьютер для информационных технологий) [10], разработанная в Институте кибернетики имени В.М. Глушкова НАНУ для решения задач, требующих вычислений большого объема и обработки массивов данных больших размеров.

Кластерная вычислительная система представляет собой набор стандартных распределенных программно-аппаратных компонентов, объединенных для решения общих задач. В кластере в качестве процессорных элементов используются стандартные компьютеры, а для коммуникации — стандартные интерфейсы для сетей. Вычислительная мощность кластерной системы определяется быстродействием вычислительных узлов и межузловых коммуникаций.

Система СКИТ обеспечивает надежное функционирование аппаратно-программных средств разработки интеллектуальных информационных технологий и прикладного программного обеспечения и предназначена для выполнения сложных заданий в области экономики, науки, безопасности, обороны и других сферах народного хозяйства. Функциональные компоненты существующего вычислительного комплекса предназначены для круглосуточного непрерывного удаленного управления потоком заданий как в рамках локальной вычислительной сети, так и через Интернет.

Вычислительный комплекс состоит из двух кластерных суперкомпьютеров: СКИТ-1 и СКИТ-2.

Эксперименты проводились на суперкомпьютерной системе СКИТ-1, являющейся 32-процессорным 16-узловым кластером на микропроцессорах Intel Xeon 2,67 ГГц (имеет разрядность 32 бита и возможность производить вычисления с 64- и 128-битовыми данными). В каждый кластер входят: управляющий сервер, сервер доступа, общий коммутатор кластерного комплекса и система хранения данных (файловый сервер).

Управляющий сервер используется для компиляции задач, постановки задач в очередь, выделения ресурсов задаче и запуска задач.

В составе кластера имеется три сети передачи данных: файловая среда, сеть управления узлами кластера и сеть, которая используется для обмена сообщениями между узлами.

Узлы кластера не имеют собственных дисков, поэтому каждый узел во время начальной загрузки образует корневую файловую систему сети NFS (Network File System). Затем создается раздел файловой системы с рабочими данными заданий. Ввод-вывод файлов осуществляется по сети обмена данными.

Индивидуальный пользовательский каталог (так называемый домашний каталог пользователя) содержит все пользовательские области ввода-вывода, в которых могут размещаться программы заданий пользователя.

Каждая задача имеет свой паспорт, в котором указана информация для системы управления задачами, ведущей очереди задач пользователей, назначающей ресурсы задаче, контролирующей выполнение задач и освобождающей использованные ресурсы после завершения задачи. Кластерный ресурс определяется количеством процессоров и временем решения задачи. Выходные файлы и файлы стандартного вывода создаются в рабочем каталоге задачи, если нет специальных указаний. Для пользователей обеспечивается индивидуальная защита его домашнего каталога от других пользователей и защита самого комплекса от несанкционированного доступа, в том числе от зарегистрированных пользователей.

Логическая система параллельного программирования обеспечивает старт программы пользователя на первом процессоре на первом узле из списка узлов, выделенных для задания, с последующим динамичным стартом порождаемых пользовательской программой процессов на следующих процессорах этого списка узлов кластера. Связь порожденных пользовательских процессов между собой и с порождающим процессом поддерживается через коммуникационные библиотеки логической системы, функционально соответствующие реализованной в ней версии интерфейса MPI (Message Passing Interface). Все узлы, выделенные одной задаче, могут связываться между собой по схеме «каждый с каждым».

Запуск задания на выполнение на узлах кластера может осуществляться разными способами в зависимости от того, является ли задание MPI-задачей или обычной непараллельной программой. Если одна задача получила в качестве ресурса процессор некоторого узла, другие задачи не могут пользоваться процессорами этого узла. Предполагается, что собственно работа пользователя осуществляется лишь в рамках его домашнего каталога на сервере доступа, а для контроля доступа к вычислительным узлам достаточно использовать механизмы управления запуском программ с помощью специальной команды.

Выбирая очередное задание из очереди, планировщик ресурсов читает паспорт задания, определяет список вычислительных модулей, на которых будет выполняться параллельное задание, открывает к ним доступ для псевдопользователя — владельца логической системы параллельного программирования, в рамках которой будет выполняться задание, запускает на них эту логическую систему, если она неактивна, и выполняет командный файл старта параллельного задания. Командному файлу передаются такие параметры, как количество и имена выделенных вычислительных модулей (в качестве конфигурационных данных для библиотеки обмена сообщениями), и задание запускается на вычисление, после чего планировщик ресурсов переходит к обработке следующего задания, находящегося в очереди.

При расчетах эффективности следует учитывать, что межпроцессорный обмен данными приводит к уравниванию эффективности каждого процесса по скорости с самым медленным процессом.

Взаимодействие пользователя с системой поддерживается средствами диалоговых окон с терминала клиента. Сеанс работы начинается с соединения вычислительной машины пользователя с сервером доступа и запуска интерактивной программы.

Задача, требующая компиляции, становится в очередь компиляции. Компиляция использует `Makefile`, который может быть готовым или генерироваться системой управления задачами. Кроме текстов программ, пользователь может задавать свои опции для файлов, которые включаются, и библиотек.

ВЫБОР ЗАДАЧ ДЛЯ ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТОВ С ПРОГРАММОЙ ПАРАЛЛЕЛЬНОГО ПОИСКА ВЫВОДА

С учетом особенностей среды (программных средств и оборудования) функционирования программы параллельного поиска вывода системы АПС для экспериментирования с этой программой были выбраны следующие задачи.

1. Является ли тавтологией пропозициональная формула вида $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$?

2. Является ли тавтологией пропозициональная формула вида $F \to F_1 \wedge \ldots \wedge F_n$?

Решение задач первого типа на кластерной системе сводится к решению n задач вида «является ли тавтологией формула F_i?» ($i \in \{1,\ldots,n\}$), поскольку $(F_1 \wedge \ldots \wedge F_n)$ — тавтология тогда и только тогда, когда F_i — тавтология для каждого i, $i \in \{1,\ldots,n\}$.

Решение задач второго типа на кластерной системе сводится к решению n задач вида «является ли тавтологией формула $F \to F_i$?» ($i \in \{1,\ldots,n\}$), поскольку $F \to F_1 \wedge \ldots \wedge F_n$ — тавтология тогда и только тогда, когда $F \to F_i$ — тавтология для каждого i, $i \in \{1,\ldots,n\}$.

Таким образом, каждый доступный процессор работает с более простой формулой, чем та, что задана изначально.

ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНОГО ПОИСКА ВЫВОДА НА БАЗЕ ИСЧИСЛЕНИЯ АЛГОРИТМА ОЧЕВИДНОСТИ

Решение задачи параллельного поиска вывода на базе алгоритма очевидности в среде СКИТ-1 осуществляется в режиме «хозяин-работник». Входные данные — это формула логики высказываний вида $F_1 \wedge \ldots \wedge F_n$, количество требуемых процессоров $k+1$ ($k \le n$), из которых один процессор выполняет функции «руководителя работ» («хозяина»), остальные — «исполнители» («работники»). Процессор-руководитель посылает сообщения процессорам-работникам и принимает от них сообщения. Каждый доступный (рабочий) процессор загружен программой поиска вывода формул логики высказываний, основанной на исчислении алгоритма очевидности, и i-й процессор-работник начинает работать с i-й формулой (т.е. с F_i), $i \in \{1,\ldots,n\}$, которая указывается ему процессором-руководителем в соответствующем сообщении. Если формула F_i является тавтологией, то i-й процессор-работник формирует об этом сообщение «Prover result_1» и, если еще остались необработанные формулы $F_m, F_{m+1}, \ldots, F_n$, принимает сообщение, содержащее указание на очередную формулу F_m из оставшихся, которую следует обрабатывать. Процесс продолжается до тех пор, пока имеются необработанные формулы. Формула является тавтологией тогда и только тогда, когда все процессоры прислали сообщение «Prover result_1». Таким образом, отдельный процессор-работник может за один сеанс работы программы параллельного поиска вывода работать с несколькими разными формулами перечня $F_1, \ldots, F_n$. Программа, которая осуществляет управление поиском — загрузку рабочих процессоров входными данными (пересылку сообщений процессорам) и анализ сообщений, поступающих от рабочих процессоров, использует MPI функции `MPI_Send` и `MPI_Receive`.

РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

С программой параллельного поиска вывода проведены две серии экспериментов. Для первой серии выбраны следующие формулы логики высказываний (приведены на языке АПЛАН).

Для эксперимента 1: Exp 1 = `( a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_7) & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_6) & a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_5) & a_4 & a_3 & a_2 & a_1 |/ ~(a_4) & a_3 & a_2 & a_1 |/ ~(a_3) & a_2 & a_1 |/ ~(a_2) & a_1 |/ ~(a_1)) &`

`( a_6 & a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_6) & a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_5) & a_4 & a_3 & a_2 & a_1 | / ~(a_4) & a_3 & a_2 & a_1 |/ ~(a_3) & a_2 & a_1 |/ ~(a_2) &`

```
a_1 |/ ~(a_1)) &
     ( a_5 & a_4 & a_3 & a_2 & a_1 |/ ~(a_5) & a_4 & a_3 &
a_2 & a_1 |/ ~(a_4) & a_3 & a_2 & a_1 |/ ~(a_3) & a_2 & a_1
|/ ~(a_2) & a_1 |/ ~(a_1)) &
     ( a_4 & a_3 & a_2 & a_1 |/ ~(a_4) & a_3 & a_2 & a_1 |/
~(a_3) & a_2 & a_1 |/ ~(a_2) & a_1 |/ ~(a_1)) &
     ( a_3 & a_2 & a_1 |/ ~(a_3) & a_2 & a_1 |/ ~(a_2) & a_1
|/ ~(a_1)) &
     ( a_2 & a_1 |/ ~(a_2) & a_1 |/ ~(a_1)) &
     ( a_1 |/ ~(a_1)).
```

Для эксперимента $i+1$: Expi = `(a_f(i) & a_(f(i)-1) & … & a_1 |/ ~(a_f(i)) & a_(f(i)-1) & … & a_1 |/ … |/ ~(a_2) & a_1 |/ ~(a_1)) & (a_(f(i)-1) & … & a_1 |/ ~(a_(f(i)-1)) &… & a_1 |/ … |/ ~(a_2) & a_1 |/ ~(a_1)) & Exp(i-1).`

Здесь $f(i)=7+2(i-1)$, i — целое, $1\le i\le 6$, $f(7)=18$.

Формула Expi.j совпадает с Expi с точностью до коммутативности и ассоциативности. Как видно из правил построения формул Expi, сложность первых конъюнктивных членов формулы значительно превосходит сложность последних ее конъюнктивных членов, поэтому при первоначальном распределении заданий между процессорами-работниками нагрузка на них неравномерна. Формула Expi.j строится из формулы Expi путем перестановки конъюнктивных членов и расстановки скобок таким образом, чтобы сделать первоначальную нагрузку процессоров-работников примерно одинаковой. Результаты экспериментов приведены в табл. 1 (время вычислений указано в секундах).

Таблица 1

Число процессоров	Результаты экспериментов для Expi													
	Exp1	Exp1.1	Exp2	Exp2.1	Exp3	Exp3.1	Exp4	Exp4.1	Exp5	Exp5.1	Exp6	Exp6.1	Exp6.2	Exp7
2	7	9	15	15	31	34	65	90	141	142	274	296	293	2091
3	7	6	11	10	15	9	34	51	68	78	136	166	137	1748
4	5	6	8	9	15	10	26	37	48	50	93	133	98	1733
5	7	7	10	10	15	14	24	40	43	62	82	113	86	1734
6	7		10		15	15	24	41	45	46	82	100	86	1716
7	10		12		17	18	25	41	45	66	66	95	92	1758
8	9		11		17		24		46	58	63	90	90	1711
9	10		12		19		25		45	63	63	88	92	1724
10			13		17		26		47	51	87	85	89	1748
11					19		28		51	71	86	89	96	1762
12					18		28		49	60	91	90	89	1755
13					19		28		51		91	89	89	1757
14					20		29		52		104	97	93	1736
15					22		31		53		90	95	93	1759
16			17		21				51		90	92	89	1772
17									141		96	97	98	1749
18											98	94	86	1765
19											99	98	98	1746
20											95	97	97	1754
21											101	100	100	1775

В табл. 2 показано, при каком количестве процессоров достигнуто максимальное ускорение в экспериментах первой серии по отношению к двум процессорам.

Таблица 2

Характеристики	Показатели ускорения для Expi													
	Exp1	Exp1.v	Exp2	Exp2.v	Exp3	Exp3.v	Exp4	Exp4.v	Exp5	Exp5.v	Exp6	Exp6.1	Exp6.2	Exp7
Время на двух процессорах	7	9	15	15	31	34	65	90	141	142	294	296	293	2091
Минимальное время	5	6	8	9	15	9	24	37	43	46	63	85	86	1711
Число процессоров	4	3	4	4	3	3	5	4	5	6	8	10	5	8
Ускорение	1,8	1,5	1,9	1,7	6,2	3,8	2,7	2,4	3,3	3	4,7	3,5	3,4	1,2

Таблица 3

Число процессоров	Результаты экспериментов для Ti									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
2	15	22	52	125	237	440	652	1054	1343	2234
3	11	19	31	62	126	250	310	541	746	1261
4	9	14	23	49	90	155	220	366	532	814
5	10	16	23	35	66	110	173	261	409	587
6	8	14	18	27	54	88	152	232	331	481
7	9	13	21	33	51	80	128	221	287	424
8	9	19	22	30	50	67	116	163	244	377
9	11	16	24	27	45	64	94	155	254	334
10	11	17	25	25	43	57	97	143	209	300
11			19	27	45	58	82	129	174	292
12			19	27	44	58	78	123	162	241
13			23	32	36	61	81	102	160	228
14			20	30	37	49	80	100	164	224
15			22	30	37	51	72	100	132	211
16				25	37	48	64	99		188
17				27	38	49	65	97		188
18				26	41	50	60	94		186
19				27	39	54	63	82		171
20				28	43	55	65	77		174
21					42	51	63	89		179
22					35	55	65	82		169
23					38	54	67	83		161
24					35	54	69	93		153
25					38	54	67	80		135
26						42	70	93		144
27						48	63	95		132
28						45	68	80		133
29						52	67			133
30						46	65			142
31						47	63			134
32						55	63			136
33						52	61			154
34							61			145

Для второй серии экспериментов выбрана задача определения существования путей между заданной вершиной ориентированного графа и каждой из остальных его вершин, кроме тех, которые смежны с заданной вершиной. Рассмотрены два вида орграфов: ориентированные деревья и орграфы с выделенной вершиной. В первом случае заданной вершиной является корень дерева, во втором — выделенная вершина.

Дерево T1 описывается формулой логики высказываний следующего вида (на языке АПЛАН):

```
T1 = (~(a_0) |/ a_1) & (~(a_0) |/ a_2) & (~(a_0) |/ a_3) & (~(a_0) |/ a_4) & (~(a_0) |/ a_5) & (~(a_1) |/ a_6) & (~(a_2) |/ a_7) & (~(a_3) |/ a_8) & (~(a_4) |/ a_9) & (~(a_5) |/ a_10).
```

Путь между корнем дерева (вершиной a_0) и вершиной a_9, например, существует, если формула T1 $\rightarrow$ (a_0 $\rightarrow$ a_9) является тавтологией. Дерево Ti ($1 \le i \le 10$) имеет $10+5(i-1)$ вершин. Орграф Gri ($1 \le i \le 4$) имеет $10+5(i-1)$ вершин и образуется из дерева Ti путем изменения направления дуги (a_(5(i –1)+1), a_(5i +1)). Выделенной является вершина a_0. Результаты экспериментов приведены в табл. 3, 4 (время вычислений указано в секундах).

Таблица 4

Число процессоров	Результаты экспериментов для Gri			
	Gr1	Gr2	Gr3	Gr4
2	19	28	64	146
3	13	21	43	93
4	12	16	29	75
5	19	21	38	72
6		25	31	50
7		42	33	45
8		17	29	51
9		28	29	42
10		24	30	52
11		32	28	44
12			37	47
13			37	44
14			34	43
15			37	45
16			36	53
17				41

Таблица 5

Характеристики	Показатели ускорения для Ti									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Время на двух процессорах	15	22	52	125	237	440	652	1054	1343	2234
Минимальное время	8	13	18		36	42	60	77	132	132
Число процессоров	6	7	6		13	26	18	20	15	27
Ускорение	1,9	1,7	2,9		6,7	10,5	10,9	13,9	10,2	16,9

Таблица 6

Характеристики	Показатели ускорения для Gri			
	Gr1	Gr2	Gr3	Gr4
Время на двух процессорах	19	28	64	146
Минимальное время	12	16	28	41
Число процессоров	4	4	11	17
Ускорение	1,6	1,8	2,3	3,6

В табл. 5 показано, при каком количестве процессоров достигнуто максимальное ускорение для ориентированных деревьев, в табл. 6 — при каком количестве процессоров достигнуто максимальное ускорение для орграфов с выделенной вершиной.

ЗАКЛЮЧЕНИЕ

Время решения задач во всех случаях существенно уменьшается при использовании более чем одного рабочего процессора, а ускорение вычислений нелинейно зависит от количества рабочих процессоров. Результаты экспериментов свидетельствуют, что лучшие временные показатели при использовании программы параллельного поиска вывода, функционирующей на кластерной системе, для формул вида

$F_1 \wedge \ldots \wedge F_n$ наблюдаются тогда, когда рабочих процессоров больше, чем один, но число их невелико по сравнению с числом n конъюнктивных членов формулы. Таким образом, чтобы достичь наибольшего ускорения при решении задач рассматриваемого класса на кластерной системе средствами АПС, не нужно использовать максимальное количество процессоров.

Отметим, что эксперименты проводились в режиме удаленного доступа к средствам вычисления. Используемая вычислительная система является ресурсом коллективного пользования. Особенности входных данных (формул) в экспериментах обусловили неравномерную загрузку процессоров. АПС — система, ориентированная на создание не столько высокоскоростных программных продуктов, сколько на построение прототипов прикладных программ, что сказывается на временных показателях работы программы.

В дальнейшем планируется исследовать возможности параллельных АПС-программ для решения других классов задач на вычислительном комплексе СКИТ.

СПИСОК ЛИТЕРАТУРЫ

1. Yamaguchi T., Tezuka Y., Kakusho O. Parallel processing of resolution // Proc. of 9th Joint Conf. on AI (Los Angeles, 1985). — San Francisco: Morgan Kaufmann Publ. Inc., 1985. — **2**. — P. 1178–1180.
2. Vitter J.S., Simons R.A. Parallel algorithms for unification and other complete problems // Proc. of ACM'84 Ann. Conf. "The Fifth Generation Challenge," 1984. — New York: ACM, 1984. — P. 75–84.
3. Schumann J., Letz R. PARTHEO: A high-performance parallel theorem prover // Proc. of CADE'90 (Kaiserlautern). — Berlin; New York: Springr-Verlag, 1990. — P. 40–56.
4. Wolf A., Letz R. Strategy parallelism in automated theorem proving // IJPRAI. — 1999. — **13**, N 2. — P. 219–245.
5. Bonacina M.P. Ten years of parallel theorem proving: a perspective // Notes of the Workshop on Strategies in Automated Deduction, Sec. Feder. Logic Conf. (Italy, July, 1999). — Trento, 1999. — P. 3–15.
6. Kapitonova J.V., Letichevski A.A., and Konozenko S.V. Computations in aps // Theoret. Computer Sci. — 1993. — **119**. — P. 145–171.
7. Капитонова Ю.В., Летичевский А.А., Волков В.А. Дедуктивные средства системы алгебраического программирования // Кибернетика и системный анализ. — 2000. — № 1. — С. 17–35.
8. Глушков В.М. О задачах теории автоматов и искусственного интеллекта // Кибернетика. — 1970. — № 2. — С. 3–17.
9. Дегтярев А.И., Лялецкий А.В. Логический вывод в САД // Математические основы систем искусственного интеллекта. — Киев: Ин-т кибернетики АН УССР, 1981. — С. 3–11.
10. Коваль В.М., Сергієнко І.В. СКІТ — український суперкомп'ютерний проект // Вісн. НАН України, 2005. — № 8. — С. 3–13.

Поступила 19.11.2009