ИНФОРМАТИКА

2005 **апрель-июнь** **№ 2**

ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

УДК 519.68, 519.7

Н.Р. Торопов

ПАРАЛЛЕЛЬНАЯ ПРОВЕРКА ДНФ НА ТАВТОЛОГИЮ

Рассматривается одна из базовых задач логико-комбинаторных вычислений – проверка дизъюнктивной нормальной формы на тавтологию. Предлагаются два алгоритма параллельного решения этой задачи с использованием многопроцессорных систем. Приводятся результаты экспериментальных испытаний предложенных алгоритмов на суперкомпьютере семейства СКИФ, показывающие эффективность параллельных вычислений при решении логико-комбинаторной задачи.

Введение

Задача проверки дизъюнктивной нормальной формы (ДНФ) D на тавтологию ($D \equiv 1$) [1] и двойственная ей задача проверки выполнимости конъюнктивной нормальной формы (КНФ) $K = \overline{D}$ (проверка наличия корней логического уравнения $K = 1$) [2, 3] занимают важное место в логико-комбинаторных вычислениях. Эти ключевые операции часто фигурируют во внутренних циклах алгоритмов, решающих сложные логико-комбинаторные задачи, возникающие, например, при решении систем логических уравнений, в логическом проектировании, в частности при минимизации булевых функций и верификации логических схем. Естественно, что эффективность алгоритма в целом существенно определяется эффективностью выполнения базовых операций во внутренних циклах. Этим, в частности, объясняется интерес широкого круга специалистов к упомянутым задачам. Проводятся даже международные соревнования на выявление алгоритма, быстрее всех обнаруживающего тавтологию анализируемой ДНФ.

Удобной формой представления ДНФ $D = c_1 \vee c_2 \vee \ldots \vee c_m$, задающей некоторую булеву функцию $f(X)$, является троичная ($m \times n$) матрица $\boldsymbol{T}$, столбцы которой поставлены в соответствие переменным множества $X = \{x_1, x_2, \ldots, x_n\}$, а строки – конъюнкциям c_i из D. Элемент t_i^j принимает значение из множества $\{0, 1, -\}$: $t_i^j = 0$, если переменная $x_j \in X$ входит в конъюнкцию c_i со знаком инверсии; $t_i^j = 1$, если переменная x_j входит в конъюнкцию c_i без знака инверсии, и $t_i^j = -$, если переменная x_j не входит в конъюнкцию c_i. В дальнейшем третье значение троичного элемента t_i^j, отмеченное символом «-», иногда именуется «черточкой», чтобы отличить его от знака минуса, переноса и пр.

В матричной интерпретации упомянутые выше задачи сводятся к задаче проверки матрицы $\boldsymbol{T}$ на *вырожденность* [1]. Троичная матрица $\boldsymbol{T}$ вырождена, если представляемая ею ДНФ тождественно равна 1 ($D \equiv 1$), что равносильно утверждениям $f(X) \equiv 1$ и $K = \overline{D} = 0$.

Проверить троичную матрицу $\boldsymbol{T}$ на вырожденность – значит найти троичный вектор $\boldsymbol{w}$, ортогональный каждой строке этой матрицы (когда матрица $\boldsymbol{T}$ не вырождена, т. е. вектор $\boldsymbol{w}$ обращает уравнение $\overline{D} = 1$ в тождество) или убедиться в том, что такого вектора не существует (когда матрица $\boldsymbol{T}$ вырождена, т. е. $D \equiv 1$). Напомним, что два троичных вектора ортогональны, если у них есть хотя бы одна компонента, которая в одном векторе имеет значение 0, а в другом – значение 1.

В настоящей работе предлагаются два алгоритма параллельной проверки троичной матрицы на вырожденность с использованием многопроцессорных вычислительных систем. В качестве прототипа для параллельных модификаций был выбран простой алгоритм из работы [1], модернизированный для практического применения в качестве вспомогательного модуля при решении других задач [4, 5]. Модернизация касается в основном критерия выбора переменной, по которой производится расщепление искомого вектора $\boldsymbol{w}$.

1. Алгоритм проверки троичной матрицы на вырожденность

Анализ троичной матрицы $\boldsymbol{T}$ на вырожденность производится методом обхода дерева поиска троичного вектора $\boldsymbol{w}$, ортогонального всем строкам матрицы. Текущая ситуация, представляемая в узле дерева, характеризуется значением вектора $\boldsymbol{w}$, отдельные компоненты которого определены (т. е. имеют значение 0 или 1), а другие (отмеченные «-») подлежат определению в процессе движения вперед по дереву. Троичный вектор $\boldsymbol{w}$ однозначно определяет минор $\boldsymbol{M}(\boldsymbol{w})$ матрицы $\boldsymbol{T}$, расположенный на пересечении ее строк, неортогональных вектору $\boldsymbol{w}$, и столбцов, отмеченных его неопределенными компонентами.

В исходном состоянии в дереве поиска имеется всего один узел, отмеченный вектором $\boldsymbol{w}$, все n компонент которого не определены (имеют значение «-»), а соответствующий ему минор $\boldsymbol{M}(\boldsymbol{w})$ совпадает с матрицей $\boldsymbol{T}$. В процессе работы алгоритма на основании анализа минора $\boldsymbol{M}(\boldsymbol{w})$ в текущем узле дерева поиска последовательно определяются значения (0 или 1) еще неопределенных компонент вектора $\boldsymbol{w}$, минор $\boldsymbol{M}(\boldsymbol{w})$ *редуцируется* и при необходимости *расщепляется* с применением описанных ниже операций.

Заметим, что хотя минор $\boldsymbol{M}(\boldsymbol{w})$, составленный из неортогональных n-мерному вектору $\boldsymbol{w}$ строк троичной $m\times n$ матрицы $\boldsymbol{T}$, однозначно определяется этим вектором $\boldsymbol{w}$, для ускорения процесса анализа строки миноров отмечаются единицами m-мерного булева вектора-маски $\boldsymbol{y}$, который также сохраняется в узле дерева поиска. Таким образом, в k-м узле дерева хранится пара векторов $(\boldsymbol{w}_k, \boldsymbol{y}_k)$, характеризующих k-ю ситуацию. Заметим также, что для доказательства факта невырожденности матрицы $\boldsymbol{T}$ достаточно найти вектор $\boldsymbol{w}$, ортогональный строкам одного из исследуемых миноров, но для доказательства вырожденности матрицы нужно обойти все дерево поиска, чтобы убедиться в вырожденности всех отмеченных в его узлах миноров.

Редукция минора по строкам. Строки минора анализируются по очереди. Если выясняется, что в миноре есть «пустая» строка (содержащая только «черточки»), то это означает вырожденность минора. Если очередная i-я строка является *безальтернативной* (т. е. содержит всего одну отличную от «черточки» компоненту $t_i^j = \alpha \in \{0, 1\}$), то j-й компоненте вектора $\boldsymbol{w}$ присваивается инверсное значение α. После этого j-й столбец и i-я строка удаляются из минора вместе со всеми другими строками, ортогональными новому значению вектора $\boldsymbol{w}$ (путем корректировки значения вектора $\boldsymbol{y}$: $(t_k^j = \alpha) \rightarrow (y_k = 0)$).

Процесс редукции минора по строкам заканчивается, когда число строк или столбцов в нем станет равным нулю или выяснится, что среди оставшихся нет безальтернативных. Отсутствие строк в миноре означает, что найден троичный вектор $\boldsymbol{w}$, ортогональный каждой строке исходной матрицы $\boldsymbol{T}$, и задача решена, а отсутствие столбцов свидетельствует о вырожденности анализируемого минора, и для решения задачи в целом требуется проверить остальные миноры (осуществляется шаг «назад» по дереву обхода).

Редукция минора по столбцам. Столбцы непустого минора анализируются по очереди. Если выясняется, что анализируемый столбец «пустой» (содержит только «черточки»), то он удаляется из минора. Если же очередной j-й непустой столбец является *безальтернативным* (не содержащим нулей или не содержащим единиц), то без потери возможности найти вектор $\boldsymbol{w}$, ортогональный всем строкам исходной матрицы $\boldsymbol{T}$ (если таковой существует), его j-й компоненте присваивается значение α, отсутствующее в компонентах j-го столбца. Затем j-й столбец и все ортогональные обновленному значению вектора $\boldsymbol{w}$ строки удаляются из редуцируемого минора (путем соответствующей корректировки значения вектора $\boldsymbol{y}$) и анализ его столбцов повторяется, начиная с первого столбца, так как удаление строк из минора может породить новые безальтернативные и пустые столбцы. Процесс заканчивается, когда минор опустеет или выяснится, что в нем нет безальтернативных или пустых столбцов.

Если редукция по столбцам привела к изменению минора, то повторяется процесс его редукции по строкам, начиная с первой строки. При этом возможна цепная реакция, так как удаление безальтернативных строк из минора может привести к появлению безальтернативных столбцов, и наоборот. Иногда весь процесс проверки троичной матрицы $\boldsymbol{T}$ на вырожденность

может завершиться с применением лишь безальтернативных операций редукции без перебора вариантов, однако это случается не часто.

Расщепление вектора. Если очередная итерация редукции минора ($\boldsymbol{w}$, $\boldsymbol{y}$) оказалась нерезультативной, применяется операция расщепления вектора $\boldsymbol{w}$ на два вектора $\boldsymbol{w}_1$ и $\boldsymbol{w}_2$, ортогональных по некоторой из отмеченных черточками переменных, причем выбирается k-я переменная, соответствующая максимально определенному столбцу минора (содержащему минимальное число «черточек», чтобы минимизировать пересечение по строкам порождаемых миноров ($\boldsymbol{w}_1$, $\boldsymbol{y}_1$) и ($\boldsymbol{w}_2$, $\boldsymbol{y}_2$)). При этом $\boldsymbol{y}_1 = \boldsymbol{y} - \boldsymbol{t}^k(0)$ и $\boldsymbol{y}_2 = \boldsymbol{y} - \boldsymbol{t}^k(1)$, где через $\boldsymbol{y} - \boldsymbol{t}^k(\alpha)$ обозначена операция «вычеркивания» единиц в i-х компонентах вектора $\boldsymbol{y}$, если $t_i^k = \alpha$. В дереве поиска появляются два новых узла с отметками ($\boldsymbol{w}_1$, $\boldsymbol{y}_1$) и ($\boldsymbol{w}_2$, $\boldsymbol{y}_2$).

Затем итерация описанных выше операций, начиная с операции редукции минора по строкам, применяется к минору ($\boldsymbol{w}_1$, $\boldsymbol{y}_1$). Алгоритм заканчивает работу, когда в некотором узле дерева поиска находится вектор $\boldsymbol{w}$ или после обхода всех узлов дерева поиска, когда матрица вырождена. Далее описанный алгоритм именуется алгоритмом *S*, чтобы отличить его от параллельных алгоритмов *P* и *D*, предлагаемых в разд. 3.

2. Классификация параллельно решаемых задач

Все задачи по способам их распараллеливания условно можно разбить на следующие два класса.

Задачи класса C – это задачи, которые решаются *коллективно N* процессами. Каждый из них решает свою частную задачу над своими исходными данными до конца, а корневой процесс *собирает* результаты частичных решений и формирует из них итог решения общей задачи. Поскольку время решения задачи из класса *C* определяется временем «самого медленного» (последнего из закончивших работу) процесса, то при декомпозиции исходной задачи на частные подзадачи следует стремиться к равномерному распределению нагрузки между процессами.

Задачи класса I отличаются от задач класса *C* тем, что решение любой частной задачи, полученное *индивидуально* любым из процессов, является решением общей задачи. Когда какой-либо из процессов находит решение, все остальные должны прекратить работу, т. е. время решения задачи класса *I* определяется «самым быстрым» процессом.

Следует заметить, что параллельные многопроцессорные структуры типа СКИФ [6] ориентированы в основном на задачи класса *C* (реализованы эффективные операции по сбору данных корневым процессом у всех подчиненных, имеются операции синхронизации процессов и пр. [7]). В то же время при решении задач класса *I* приходится изыскивать специальные приемы для прерывания активных процессов в тот момент, когда один из процессов нашел решение. Одним из таких приемов является вставка во внутренний цикл программы, выполняющейся в подчиненных процессах, операции *MPI_IProbe*, реагирующей на факт посылки корневым процессом некоторого «сигнала прерывания». В случае «положительной пробы» (когда к моменту проверки корневой процесс уже послал «сигнал прерывания») подчиненный процесс должен обеспечить прием этого сигнала и прекратить работу. Место вставки выбирается с таким расчетом, чтобы обеспечить оптимальную скорость реакции подчиненных процессов на «сигнал прерывания» при допустимых накладных расходах.

Задача проверки троичной матрицы на вырожденность относится к одному из двух названых классов в зависимости от типа проверяемой матрицы. При доказательстве вырожденности матрицы решается задача класса *C* (время ее решения определяется «самым медленным» процессом, доказавшим, что последний из еще непроверенных миноров также оказался вырожденным), а при обнаружении вектора $\boldsymbol{w}$, ортогонального всем строкам матрицы, решена задача класса *I*, причем время её решения определяется «самым быстрым» процессом, первым обнаружившим вектор $\boldsymbol{w}$, все остальные должны при этом закончить работу.

Для организации обмена сообщениями между процессами, который реализуется при параллельной проверке троичной матрицы на вырожденность, разработан специальный класс операций с

буферами обмена, обеспечивающий упаковку в буфер и распаковку из него таких объектов, как число, булев или троичный вектор, булева или троичная матрица, массив троичных матриц.

3. Параллельная проверка троичной матрицы на вырожденность

При разработке параллельных модификаций алгоритма анализа троичной матрицы на вырожденность особое внимание уделяется организации равномерного распределения нагрузки между процессорами, участвующими в анализе, и минимизации накладных расходов на информационный обмен между ними. Предлагаются два алгоритма, различающиеся способами распределения нагрузки между корневым и подчиненными процессами. При описании алгоритмов приводятся сокращенные имена некоторых операций из системы MPI [7], представляющей среду программирования для суперкомпьютера семейства СКИФ [6].

3.1. Алгоритм P (с предварительным распределением нагрузки)

Предлагается алгоритм параллельной проверки троичной матрицы на вырожденность, предварительно распределяющий нагрузку между корневым и N подчиненными процессами с минимизацией числа обменов сообщениями между ними.

Корневой процесс начинает анализировать матрицу $\boldsymbol{T}$, исполняя алгоритм S, до тех пор, пока в дереве поиска вектора $\boldsymbol{w}$ не окажется N узлов, представляющих координаты миноров. Если к этому моменту корневой процесс не решит задачу до конца, анализ N полученных миноров распределяется между подчиненными процессами по следующему алгоритму.

Корневой процесс

К1. Посылает (операцией *Bcast*) всем процессам значение анализируемой матрицы $\boldsymbol{T}$ и, сформировав у себя в памяти транспонированную матрицу $\boldsymbol{T}^T$, переходит в состояние К2.

К2. Рассылает (операцией *Send*) координаты предварительно полученных миноров ожидающим сообщений подчиненным процессам. Разослав координаты всех миноров и отметив получившие их процессы в списке F загруженных, переходит в состояние К3.

К3. Принимает (операцией *Recv*) сообщения от активных подчиненных процессов, вычеркивая их из списка F. Если в полученном сообщении содержится вектор $\boldsymbol{w}$ или список F опустел, то посылает (операцией *Send*) всем подчиненным процессам сигнал END (прекратить работу) и переходит в состояние К4, иначе остается в состоянии К3.

К4. Принимает (операцией *Recv*) сообщения от еще оставшихся в списке F активных подчиненных процессов. Когда все сообщения получены, заканчивает работу с выдачей найденного вектора $\boldsymbol{w}$ или сигнала о вырожденности матрицы.

Подчиненный процесс

П1. Получает (операцией Bcast) значение анализируемой матрицы $\boldsymbol{T}$, формирует у себя в памяти транспонированную матрицу $\boldsymbol{T}^T$ и переходит в состояние П2.

П2. Принимает (операцией *Recv*) сообщения от корневого процесса. Если в принятом сообщении содержатся координаты минора, то переходит в состояние П3, а если сигнал END, то заканчивает свою работу.

П3. Анализирует полученный минор, посылает (операцией *Send*) корневому процессу соответствующее сообщение и возвращается в состояние П2. Посланное сообщение содержит вектор $\boldsymbol{w}$, если таковой находится, или будет пустым, если минор вырожден или в процессе его анализа обнаруживается (операцией *IProbe*), что корневым процессом был послан сигнал END.

Поскольку основная работа корневого процесса по предварительному формированию миноров локализована на начальном этапе и каждый из подчиненных процессов стартует всего

один раз, то итоговое «параллельное время» t_p, затраченное алгоритмом P на доказательство вырожденности матрицы $\boldsymbol{T}$, определяется суммой

$$t_p = t_0 + max\ \{t_i\}\ \ (i \in \{1, 3, \ldots, N\}),$$

где t_0 – время, затраченное корневым процессом на предварительное получение N миноров, а t_i – время работы i-го подчиненного процесса.

В случае невырожденной матрицы итоговое «параллельное время» t_p определяется по часам i-го процесса, первым нашедшего вектор $\boldsymbol{w}$: $t_p = t_0 + t_i$.

3.2. Алгоритм D (с динамическим распределением нагрузки)

Предлагается алгоритм с динамическим распределением нагрузки между подчиненными процессами. Распределение производится корневым процессом через магазин миноров по мере формирования последних при обходе дерева поиска вектора $\boldsymbol{w}$.

Корневой процесс

К1. Посылает (операцией *Bcast*) всем процессам значение исходной матрицы $\boldsymbol{T}$, формирует транспонированную матрицу $\boldsymbol{T}^T$ и, перед тем как перейти в состояние К2, организует список F свободных процессов и магазин M миноров. В исходном состоянии в список F попадают все подчиненные процессы, а в магазин M – координаты минора ($\boldsymbol{w}$, $\boldsymbol{y}$), представляющего исходную матрицу $\boldsymbol{T}$.

К2. Посылает (операцией *Send*) координаты очередного минора из магазина M одному из ожидающих сообщений подчиненных процессов, вычеркивая последний из списка F. Когда магазин M или список F опустеют, переходит в состояние К3. Если же при пустом магазине M список F полон, то переходит в состояние К4.

К3. Принимает (операцией *Recv*) сообщения от активных подчиненных процессов, отмечая их в списке F. Получив вектор $\boldsymbol{w}$, посылает всем остальным сигнал END (прекратить работу) и переходит в состояние К4, а в двух других ситуациях возвращается в состояние К2, предварительно разместив в магазине M координаты полученного в третьей ситуации минора.

К4. Принимает (операцией *Recv*) от не отмеченных еще в списке F активных подчиненных процессов сообщения. Когда все сообщения получены, заканчивает работу с выдачей найденного вектора $\boldsymbol{w}$, если таковой найден, или сигнала о вырожденности матрицы.

Подчиненный процесс

П1. Получает (операцией *Bcast*) значение анализируемой матрицы $\boldsymbol{T}$, формирует у себя в памяти транспонированную матрицу $\boldsymbol{T}^T$ и переходит в состояние П2.

П2. Принимает (операцией *Recv*) сообщения от корневого процесса. Если в принятом сообщении оказывается сигнал END, то заканчивает свою работу, а если в сообщении содержатся координаты минора, то переходит в состояние П3.

П3. Анализирует полученный минор и результаты анализа посылает корневому процессу. При этом в сообщении могут содержаться: *вектор* $\boldsymbol{w}$ (если найден); *пустое сообщение* (если минор вырожден или в процессе его анализа (операцией *IProbe*) обнаруживается, что корневой процесс уже послал сигнал END); *координаты наименьшего из миноров* и номер q (если вектор $\boldsymbol{w}$ расщеплен по переменной q на два). Послав сообщение в первых двух ситуациях, возвращается в состояние П2, а в третьей ситуации остается в состоянии П3, анализируя наибольший из полученных при расщеплении вектора $\boldsymbol{w}$ миноров.

4. Экспериментальные испытания

Описанные выше алгоритмы параллельной проверки троичной матрицы на вырожденность разрабатывались и отлаживались на модели, реализованной на персональном компьютере, а затем были испытаны на реальной многопроцессорной системе СКИФ [7].

Детальные исследования алгоритмов проверки выполнимости КНФ в пространстве параметров (m, n, r), проведенные в [1, 2], показали, что наибольшая трудоемкость падает на матрицы с фиксированным рангом (числом определенных компонент) строк $r = 3$ при $m = qn$. Здесь через q обозначен некоторый коэффициент, значение которого находится в пределах $(5 > q > 4)$ и лишь слегка уменьшается с ростом n. Испытания предлагаемых алгоритмов P и D показали, что с возрастанием значения n от 100 до 220 значение q изменяется от 4,61 до 4,32.

В табл. 1 и 2 показаны некоторые результаты испытаний параллельных алгоритмов P и D в сравнении с последовательным вариантом S. Испытания проводились на потоке псевдослучайных троичных матриц в пространстве параметров (m, n, r) по лучам, пересекающим точку экстремума ($m = 865$, $n = 200$, $r = 3$). Прочерк в столбце $\boldsymbol{w}$ означает, что проверяемая матрица вырождена, а символ * показывает, что обнаружен вектор $\boldsymbol{w}$, ортогональный всем строкам матрицы. Время в таблицах указывается в секундах.

Таблица 1

Сравнение быстродействия алгоритмов S, P и D при различных значениях числа n столбцов в матрице ($m = 865$, $r = 3$, $N = 10$)

n	$\boldsymbol{w}$	t_s	t_p	t_d
160	–	11,89	3,45	1,25
170	–	77,35	19,62	7,89
180	–	107,62	23,76	10,96
190	–	851,94	216,72	89,85
200	–	2564,86	752,49	335,31
210	*	876,71	193,43	83,62

Таблица 2

Сравнение быстродействия алгоритмов S, P и D при различных значениях числа m строк в матрице ($n = 200$, $r = 3$)

m	$\boldsymbol{w}$	t_s	$N = 5$		$N = 10$		$N = 15$		$N = 20$	
			t_p	t_d	t_p	t_d	t_p	t_d	t_p	t_d
840	*	137,60	137,55	70,98	63,72	23,89	23,71	4,47	9.01	3.67
850	*	10,94	10,96	3,21	10,96	3,91	10,39	2,31	10.98	1.94
860	*	1735,30	165,11	75,89	165,17	119,09	165,02	115,95	165.06	83.40
870	–	2273,13	913,25	460,01	652,47	230,14	652,28	154,98	392.84	115.25
880	–	1339,29	516,98	270,42	356,09	135,47	263,42	90,30	263.66	67.82
890	–	1269,12	490,41	256,73	338,19	128,98	258,30	85,70	248.63	64.39
900	–	1161,51	440,57	266,24	314,50	141,34	244,81	78,44	233.51	58.94
910	–	1055,53	411,35	213,70	**290,93**	**107,54**	221,57	71,33	213.48	53.61

Проведенные эксперименты показали, что параллельные алгоритмы *P* и *D* работают быстрее последовательного варианта *S*. В то же время алгоритм *P* не выдерживает конкуренции по быстродействию с алгоритмом *D*, так как при достаточно коротких сообщениях, которыми обмениваются процессы между собой, на быстродействие алгоритмов гораздо большее влияние оказывает равномерность распределения нагрузок между процессами, чем количество обменов сообщениями между ними.

Последнее замечание подтверждается данными табл. 3, где приведена информация о распределении нагрузок между 10 подчиненными процессами во время анализа алгоритмами *P* и *D* (910×200)-матрицы, итоги эксперимента с которой выделены жирным шрифтом в последней строке табл. 2. Во второй колонке табл. 3 показано время работы каждого из 10 подчиненных процессов, стартующих по одному разу при выполнении алгоритма *P*, а в третьей колонке приведено число стартов этих процессов при выполнении алгоритма *D* (суммарное время работы каждого из процессов при этом колеблется в пределах от 107,42 до 107,54 с). Разность между максимальным и минимальным временем работы подчиненных процессов для алгоритма *P* при разовой их загрузке составляет 263,82 = (290,93 – 27,11), в то время как для алгоритма *D* при многократной загрузке процессов эта разность не превышает 0,12 с.

Таблица 3

Время работы алгоритма *P* и число стартов алгоритма *D* для каждого из 10 подчиненных процессоров при анализе (910×200)-матрицы на вырожденность

Номер процесса	Время работы *P*	Число стартов в *D*
1	90,49	28 502
2	221,16	28 656
3	36,49	28 494
4	51,13	28 541
5	290,93	28 574
6	120,54	28 493
7	27,71	28 505
8	71,43	28 514
9	47,89	28 453
10	100,40	28 547

Заключение

Проведенные эксперименты показали, что *N*-кратное ускорение параллельной проверки ДНФ на тавтологию с использованием *N* процессоров, по сравнению с однопроцессорным вариантом, вполне достижимо при надлежащем распределении нагрузки между процессорами и минимизации длины обмениваемых ими сообщений.

При разработке эффективных параллельных алгоритмов для суперкомпьютера семейства СКИФ следует:

– оптимизировать потоки информации между процессами, участвующими в решении задачи;

– для задачи класса *C* стремиться к максимально равномерному распределению нагрузки между процессами, так как время решения задачи в целом определяется самым медленным процессом;

– для задачи класса *I* создать такие условия, чтобы какой-нибудь один из процессов (любой) как можно быстрее решил задачу, а все остальные сразу же прекратили свою работу.

Автор выражает благодарность Д.И. Черемисинову за помощь в освоении суперкомпьютера семейства СКИФ и в организации экспериментов с ним.

Список литературы

1. Закревский А.Д. Алгоритмы синтеза дискретных автоматов. – М.: Наука, 1971.

2. Уткин А.А. Экспериментальное исследование алгоритмов «выполнимость» // Автоматика и вычислительная техника. – 1960. – № 6. – С. 66–74.

3. Crawford J.M., Auton L.D. Experimental results on the crossover point in random 3-SAT // Artificial Intelligence. – 1996. – 81(1). – P. 31–57.

4. Торопов Н.Р. Минимизация систем булевых функций в классе ДНФ // Логическое проектирование: сб. науч. тр. – Мн: Ин-т техн. кибернетики НАН Беларуси, 1999. – Вып. 4. – С. 4–19.

5. Торопов Н.Р. Инверсия системы ДНФ // Логическое проектирование: сб. науч. тр. – Мн: Ин-т техн. кибернетики НАН Беларуси, 1999. – Вып. 4. – С. 20–33.

6. Принципы построения суперкомпьютеров семейства «СКИФ» и их реализация / С.М. Абрамов, Н.Н. Парамонов, В.В. Анищенко и др. // Информатика. – 2004. – № 1. – С. 89–106.

7. Gropp W., Lusk E., Skjellum A. Using MPI: Portable parallel programming with message passing interface. – MIT Press, 1999.

Поступила 21.04.05

Объединенный институт проблем информатики НАН Беларуси, Минск, Сурганова, 6

N.R. Toropov

A PARALLEL TAUTOLOGY DNF CHECKING

One basic logic-combinatorial task – tautology checking a disjunctive normal form (DNF) is discussed. Two algorithms for parallel tautology DNF checking by multi-processor systems are suggested. The results of experimental investigation of the suggested algorithms on supercomputers family SKIF testify the efficiency of the parallel calculations for solving this basic logic-combinatorial task.