EVALUATION OF THE EFFECTIVENESS OF MONITORING AND RESTORING DATA INTEGRITY IN INFORMATION SYSTEMS FOR VARIOUS PURPOSES

O.P. Shemtov, D.M. Simonenko, R.V. Fadeev, P.A. Novikov

In modern conditions of warfare, a significant role is assigned to information systems for various purposes. To increase the efficiency of the use of resources in information systems for various purposes, a method for monitoring and restoring data integrity has been developed, requiring an assessment of efficiency and an assessment of redundancy with various consequences of destructive information and technical impact.

Key words: network-centric war, information and technical impact, information system, hashing function, redundant modular codes.

Shemetov Oleg Petrovich, employee, oleg.shem15@mail.ru, Russia, Krasnodar, Krasnodar Higher Military School named after S.M. Shtemenko,

Simonenko Danil Mikhailovich, employee, dichenko.sa@yandex.ru, Russia, Krasnodar, Krasnodar Higher Military School named after S.M. Shtemenko,

Fadeev Roman Viktorovich, employee, roma-70594@yandex.ru, Russia, Krasnodar, Krasnodar Higher Military School named after S.M. Shtemenko,

Novikov Pavel Arkadievich, candidate of technical sciences, employee, novikov.p.ark@yandex.ru, Russia, Krasnodar, Krasnodar Higher Military School named after S.M. Shtemenko

УДК 004.4'24
DOI: 10.24412/2071-6168-2022-10-230-237

АВТОМАТИЗАЦИЯ ФОРМИРОВАНИЯ МАКСИМАЛЬНОГО ПАРАЛЛЕЛИЗМА В ПРОЦЕССЕ

О.С. Крюков, А.Г. Волошко, А.Н. Ивутин

В работе рассматривается необходимость создания автоматизированного средства распараллеливания вычислительных процессов, предложен метод и алгоритмы формирования минимально последовательной модели нелинейного процесса. В основу подхода положено представление процесса в виде модели расширенной сети Петри с семантическими связями.

Ключевые слова: сети Петри, моделирование, модель процесса, структура процесса, параллелизм, минимально последовательное представление.

Введение. Предприятия различных сфер экономики сталкиваются с необходимость постоянного совершенствования своих процессов для повышения конкурентоспособности, снижения затрат или ускорения вывода новых продуктов на рынок. Независимо от целей, очевидной необходимостью является анализ и совершенствование как процессов, так и технологий. Одним из ключевых конкурентных преимуществ в современном мире является цифровизация как можно большего числа происходящих процессов как для их мониторинга и анализа, так и для их ускорения и снижения ошибок. Однако, это приводит к формированию ряда информационных процессов, которые также нуждаются в анализе и оптимизации.

В сфере вычислительной техники улучшение характеристик производительности долгое время достигалось путем повышения тактовой частоты процессора, однако по данному варианту уже достигнуты пределы его применения, связанные с ограничениями, вызванными как характеристиками оборудования, так и технологическим процессом их производства. Другим способом является использование параллельных и распределенных вычислительных систем для одновременного исполнения нескольких процессов. Подобный вариант позволяет в идеальном случае при одной и той же тактовой частоте достигать кратного увеличения производительности.

К основным проблемам внедрения параллелизма относится сложность программирования, так как программисту необходимо учитывать способ обмена данными между параллельными процессам, накладные расходы на формирование параллельных участков и передачу данных, возможности архитектуры, возможности и ограничения различных технологий программирования и т.д. Данные аспекты приводят к снижению эффективности разрабатываемого программного обеспечения, а также приводят к дополнительным финансовым и временным затратам на разработку систем. Кроме того, остаются вопросы масштабирования и возможности использования имеющихся программ при изменении аппаратного обеспечения.

Вследствие этого возникает необходимости создания универсального средства, способного в автоматическом режиме выполнять построение параллельной версии последовательного алгоритма, тем самым повышая скорость выполнения работы программистом.

Подходы к формированию параллельных процессов. Распараллеливание процессов можно выполнять на трех уровнях представления:

на уровне отдельные операций;

на уровне блоков – линейной последовательности операций, завершающейся передачей управления в другой блок;

на уровне более крупных структур (задач и групп задач).

После выбора уровня представления процесса производится непосредственно его распараллеливание. Далее рассматриваются методы распараллеливания исходя из представления на уровне операций, однако данные методы верны и для других уровней.

Так как операции процессов зачастую являются «потребителями» результатов других операций, то любая из таких операций не может быть начаться раньше, чем закончится выполнение всех ее «поставщиков». Следовательно, при распараллеливании должен соблюдаться верный порядок операций, что позволит гарантировать корректность результата.

Вариантом, позволяющим при распараллеливании учесть последовательность операций, является граф операции-операнды, отображающий информационные зависимости [1]. В этом графе множество вершин отображает множество операций, а каждая дуга (u, v), означает, что в операции v используются результаты операции u. В таком представлении операции, между которыми нет дуг, могут выполнять параллельно. Определение независимых операций выполняется на основе построения ярусов параллельной формы. Все позиции, принадлежащие одному ярусу, не связаны дугами и, следовательно, независимы, при этом если дуга из вершины яруса i ведет в вершину яруса j, то $i < j$ [1]. На рис. 1 представлен граф операции-операнды некоторого процесса в соответствии с распределением операций по ярусам.

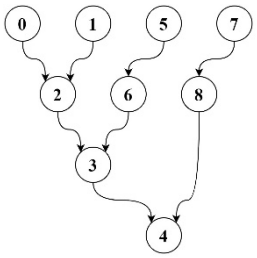

Рис. 1. Пример графа операции-операнды

Применение подобного метода моделирования параллельных структур может приводить к возникновению множества вариантов схем вычислений, каждую из которых нужно проанализировать и выбрать лучшую [1]. Например, на рис. 2 представлены некоторые варианты параллельных схем выполнения процесса, граф зависимостей которого изображен на рис. 1.

Решение этой проблемы описано в работе [2]. Автор предлагает применение различных алгоритмов оптимизации для автоматизированного построения лучших схем параллельного выполнения.

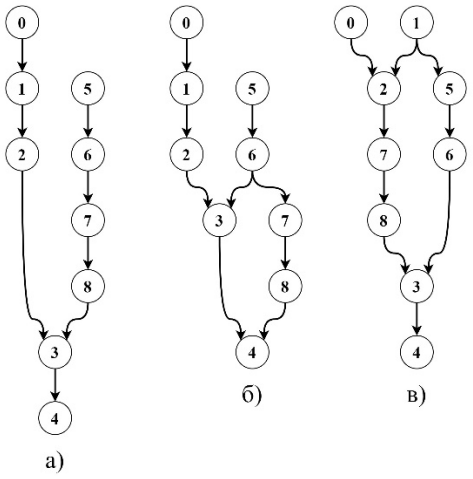

Рис. 2. Варианты параллельных схем

Другой проблемой применения анализа информационных зависимостей, описываемой в работе [1], является невозможность его применения для процессов, имеющих в своей структуре условия, приводящие к возникновению ветвлений и циклов. Например, на рис. 3 представлен граф некоторого процесса: черным цветом обозначена последовательность выполнения операций, синим – линейные зависимости, красным – нелинейная зависимость, обусловленная наличием цикла. Наличие нелинейной зависимости делает невозможным анализ процесса на основе построения ярусов параллельной формы, так как в два различных момента времени одна и та же операция может зависеть от разных «поставщиков».

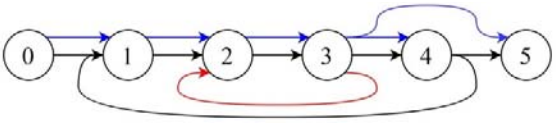

Рис. 3. Пример нелинейных зависимостей

В работе [3] описывается возможное решение данной проблемы, заключающееся в объединении структур, формирующих ветвления и циклы, в одну вершину графа с сохранением зависимостей. На рис. 4 показан пример преобразования схемы нелинейных зависимостей рис. 3 к линейному виду.

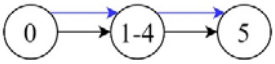

Рис. 4. Пример преобразования к линейному виду

В соответствии с этим, следующей проблемой, которая должна быть решена при распараллеливании, является определение нелинейных структур. Для этого процесс, помимо реализации представления в форме графа операции-операнды, необходимо представление процесса в форме, позволяющей оценить зависимости по управлению.

Расширенные сети Петри с семантическими связями. Моделью, позволяющей представить вычислительный процесс как последовательность типовых операций, связанных друг с другом по управлению и/или информационно, является расширенная сеть Петри с семантическими связями (РСПСС) [11]. Простейшая РСПСС может задаваться следующим множеством:

$$\Pi = \{A, \{Z^C, \tilde{R}^C, \hat{R}^C\}, \{Z^S, \tilde{R}^S, \hat{R}^S\}\},$$

где A – конечное множество позиций; Z^C – конечное множество переходов по управлению; Z^S – конечное множество переходов по семантическим связям; $\tilde{R}^X$ – матрицы смежности, отображающие множество позиций в множество переходов по управлению; $\hat{R}^X$ – матрицы смежности, отображающие множество переходов по управлению в множество позиций.

Кроме того, задаются следующие функции переходов:: $I_A(Z^X)$ – входная функция переходов; $O_A(Z^X)$ – выходная функция переходов.

Все фишки подобной сети также имеют управляющую и семантическую часть, то есть являются составными. Кроме того, при необходимости, данная модель может быть дополнена некоторыми ограничениями, позволяющими учесть различные аспекты процесса:

цвет позиций и переходов;

время и логика срабатывания переходов.

Раскрашенная сеть позволяет выполнять распределение выполнения команд между несколькими исполнителями. Временные характеристики, накладываемые на переходы по управлению, определяют время выполнения предшествующей ему операции. Время, связанное с переходом по семантическим связям, позволяет учесть задержку на передачу информации между командами, например, когда распределенная система вынуждена передавать данные по сети с невысокими показателями скорости обмена.

Структуры РСПСС. В модели процесса, построенной в соответствии с определением РСПСС, можно выделить три вида структур, формируемых в результате различных комбинации позиций и переходов по управлению [4, 5]:

линейный участок:

$$Ln = \{a_{i(a)}\}, |Ln| \geq 1;$$

цикл:

$$C = \{a_{cB}, (a_{cond}, a_{out}), Cp\},$$

где a_{cB} - начальная позиция, a_{cond} - условие (branch-позиция), a_{out} - внешняя позиция, в которую передается управление из цикла, Cp - тело цикла;

ветвление:

$$Br = \{a_{xB}, \{B_{x1}, \ldots, B_{xn}\}, a_{xe}\},$$

где a_{xB} - начальная позиция ветвления, B_{xi} - ветви ветвления, a_{xe} - конечная позиция ветвления.

Линейный участок описывает структуру по управлению, подразумевающую выполнение некоторой операции сразу после окончания выполнения предыдущей.

Циклы подразумевают возвращение по управление на начальную позицию, наличие условия, выводящего управление за пределы цикла, и тела цикла, содержащего последовательность линейных участков, ветвлений и циклов [4].

Ветвление характеризуется наличием условия, описывающем варианты действий согласно его результату, причем количество таких вариантов может варьироваться от 2 до n, и содержит в себе начальную позицию, конечную позицию, а также ветви, каждая из которых может содержать последовательность линейных участков, ветвлений и циклов [5].

Распараллеливание процессов на основе РСПСС. Метод распараллеливания процессов основан на построении из исходной РСПСС множества семантических уровней и дальнейшего его преобразования в эквивалентную минимально последовательную сеть (МПС), то есть такую РСПСС, которая обладает минимальным числом последовательных участков. Подобное представление позволяет изначально оценить эффективность применения параллельной реализации алгоритма. Кроме того, в дальнейшем, основываясь на полученной МПС и применяя оптимизационные методы, сформированные параллельные участки можно перераспределить в соответствии с требуемым числом исполнителей.

Две расширенные сети Петри с семантическими связями являются эквивалентными если у них совпадают множества позиций и переходов по семантическим связям, и для каждого перехода по семантическим связям совпадают входные и выходные функции. С точки зрения функционирования исходная и модифицированная сети Петри эквивалентны, если при любой последовательности входных позиций, в которых появляется фишка, управляющая срабатыванием перехода, в выходных позициях сетей количество фишек одинаково [6].

Множество семантических уровней представляет собой многоуровневую структуру, где каждый уровень содержит множество позиций исходной сети. Физический смысл семантического уровня заключается в том, что позиции, принадлежащие одному уровню, являются семантически независимыми и могут выполняться параллельно. При этом, информация, принадлежащая *i*-му уровню иерархии может потребоваться на *j*-ом уровне, при условии $j < i$.

Как уже было сказано, наличие нелинейности в процессе, в соответствии с определенным методом формирования параллельного представления, усложняет его. Помимо уже рассмотренного способа упрощения сети на основе сворачивания структур [3] существуют и другие решения, предлагаемые для выполнения задачи распараллеливания нелинейного представления: слияние циклов и рассмотрение линейных участков полученных тел циклов [7, 8], формирование решётчатого графа информационных зависимостей для синхронизации операций [9], разворачивание циклов в линейное представление в зависимости от числа итераций [10] и т.д. Однако, данные методики не предусматривают наличие в структуре модели ветвлений, а также не предоставляют универсального решения для всех типов циклов.

Таким образом, для осуществления распараллеливания нелинейной структуры с использованием модели РСПСС наилучшим вариантом представляется внедрение иерархии в модель, то есть осуществление замены нелинейную структуру целиком одной позицией с сохранением семантических связей. В результате осуществляется преобразование сети к иерархическому виду, где верхним уровнем является исходная сеть, без циклов и ветвлений, то есть являющаяся линейным представлением, а последующими уровнями – тела циклов и ветвлений. Каждый уровень иерархии необходимо распараллеливать отдельно.

Общий алгоритм распараллеливания представляется следующим образом.

1. Структурный анализ исходной сети – определение нелинейных структур, усложняющих структуру сети, посредством последовательного анализа позиций сети для определения их отношения к той или иной нелинейной структуре.

2. Преобразование текущего уровня иерархии к линейному виду – замена всех, вложенных в текущий уровень циклов и ветвлений на одну позицию (позицию-структуру) с сохранением семантических связей.

3. Распараллеливание полученной линейной структуры на основе построения множества семантических уровней путем создания новых переходов по управлению между каждой соседней парой семантических уровней.

4. Распараллеливание следующего уровня иерархии – рассматриваются циклы и ветвления, объединенные на текущем уровне в одну позицию. При распараллеливании уровня, соответствующего циклу, выполняется распараллеливание тела цикла, ветвлению – каждой ветви по отдельности.

5. Восстановление прежнего уровня представления – позиции-структуры заменяются изначальными структурами, формируются переходы, связывающие начальные и конечные позиции структур, в соответствии с переходами, определенными для позиций-структур.

Структурный анализ РСПСС выполняется в соответствии с описанным в работах [4, 5] алгоритмами. Рассмотри подробнее этапы 2-3 алгоритма.

Преобразование к линейному представлению. Для описания алгоритма преобразования РСПСС к линейному представлению введем следующие переменные: *nA* – множество свободных позиций (не входящие в циклы и ветвления) текущего уровня иерархии, *cP* – множество новых позиции, представляющих циклы, включенные в текущий уровень иерархии, *bP* – множество новых позиций, представляющих ветвления, включенные в текущий уровень иерархии.

Преобразование к линейному представлению для всех уровней иерархии выполняется следующим образом:

1. Сформировать новое множество *nA* – объединение линейных участков.
2. Сформировать множества новых позиций *cP* и *bP*.
3. Сформировать множество переходов по семантике для текущего представления:

3.1. Для позиции $p \in nA$ создать переход *z* для которого:

$$I_A(z) = \{a_i\} \mid a_i \in nA \wedge \exists z^S_{i(z^S)}\left(I_A(z^S_{i(z^S)}) = a_i \wedge O_A(z^S_{i(z^S)}) = p \right) \vee$$

$$\vee a_i \in cP \wedge \exists z^S_{j(z^S)}\left(\exists a_{i(a)}(a_{i(a)} \in C_{a_i}) \wedge I_A(z^S_{j(z^S)}) = a_i \wedge O_A(z^S_{j(z^S)}) = p \right) \vee$$

$$\vee a_i \in bP \wedge \exists z^S_{k(z^S)}\left(\exists a_{i(a)}(a_{i(a)} \in Br_{a_i}) \wedge I_A(z^S_{k(z^S)}) = a_i \wedge O_A(z^S_{k(z^S)}) = p \right);$$

$$O_A(z) = p.$$

3.2. Для ветвления $p \in bP$ создать переход *z* для которого: $I_A(z) = \{a_i\} \mid$

$$a_i \in nA \wedge \exists z^S_{i(z^S)}\left(\begin{array}{c} I_A(z^S_{i(z^S)}) = a_i \wedge \\ \exists a_{i(a)}\left(a_{i(a)} \in Br_p \wedge O_A(z^S_{i(z^S)}) = a_{i(a)} \right) \end{array} \right) \vee$$

$$\vee a_i \in cP \wedge \exists z^S_{j(z^S)}\left(\begin{array}{c} \exists a_{i(a)}\left(a_{i(a)} \in C_{a_i} \wedge I_A(z^S_{j(z^S)}) = a_{i(a)} \right) \wedge \\ \wedge \exists a_{j(a)}\left(a_{j(a)} \in Br_p \wedge O_A(z^S_{j(z^S)}) = a_{j(a)} \right) \end{array} \right) \vee$$

$$\wedge a_i \in bP \wedge \exists z^S_{k(z^S)}\left(\begin{array}{c} \exists a_{i(a)}\left(a_{i(a)} \in Br_{a_i} \wedge I_A(z^S_{k(z^S)}) = a_{i(a)} \right) \wedge \\ \wedge \exists a_{j(a)}\left(a_{j(a)} \in Br_p \wedge O_A(z^S_{k(z^S)}) = a_{j(a)} \right) \end{array} \right); \quad O_A(z) = p.$$

3.3. Для цикла $p \in cP$ создать переход *z* для которого: $I_A(z) = \{a_i\} \mid$

$$a_i \in nA \wedge \exists z^S_{i(z^S)}\left(\begin{array}{c} I_A(z^S_{i(z^S)}) = a_i \wedge \\ \exists a_{i(a)}\left(a_{i(a)} \in C_p \wedge O_A(z^S_{i(z^S)}) = a_{i(a)} \right) \end{array} \right) \vee$$

$$\vee a_i \in cP \wedge \exists z^S_{j(z^S)}\left(\begin{array}{c} \exists a_{i(a)}\left(a_{i(a)} \in C_{a_i} \wedge I_A(z^S_{j(z^S)}) = a_{i(a)} \right) \wedge \\ \wedge \exists a_{j(a)}\left(a_{j(a)} \in C_p \wedge O_A(z^S_{j(z^S)}) = a_{j(a)} \right) \end{array} \right) \vee$$

$$\vee a_i \in bP \wedge \exists z^S_{k(z^S)}\left(\begin{array}{c} \exists a_{i(a)}\left(a_{i(a)} \in Br_{a_i} \wedge I_A(z^S_{k(z^S)}) = a_{i(a)} \right) \wedge \\ \wedge \exists a_{j(a)}\left(a_{j(a)} \in C_p \wedge O_A(z^S_{k(z^S)}) = a_{j(a)} \right) \end{array} \right); \quad O_A(z) = p.$$

4. Объединить *nA*, *cP* и *bP* в единое множество позиций для последующего распараллеливания.

Так как, для дальнейшего распараллеливания важно только сохранение семантических связей, то восстановление переходов по управлению не осуществляется.

Распараллеливание линейного участка. При распараллеливании линейного участка между каждой парой соседних семантических уровней формируются новые переходы по управлению на основании семантических связей. Помимо простых (примитивных) переходов по управлению, для которых $|I_A(z^C_{i(z^C)})|=|O_A(z^C_{i(z^C)})|=1$, у параллельной РСПСС, в отличии от последовательной, имеются непримитивные переходы, то есть переходы следующего вида [11]:

• fork ($|I_A(z^C_{i(z^C)})|=1, |O_A(z^C_{i(z^C)})|>1$) – порождающая операция, которая описывает действие, в результате которого в системе запускается один или несколько дополнительных (параллельных) процессов;

• join ($|I_A(z^C_{i(z^C)})|>1, |O_A(z^C_{i(z^C)})|=1$) – поглощающая операция, которая описывает действие, в результате которого в системе несколько параллельных процессов объединяются в один;

• synchro ($|I_A(z^C_{i(z^C)})|>1, |O_A(z^C_{i(z^C)})|>1$) – синхронизирующая операция, которая характеризует действие, в результате которого в системе два и более параллельных процессов ожидают завершения друг друга, после чего их независимое выполнение продолжается.

Для описания алгоритма распараллеливания линейного представления введем следующие переменные: *starts* – множества начальных позиций с точки зрения семантики, *L* – множество семантических уровней, *outs* – множество позиций, которые могут являться выходами для формируемых переходов по управлению, *united* - множество позиций, для которых уже определены переходы, G_S – множество позиций, для которых осуществляется формирование одного перехода типа synchro.

Сеть, преобразованная в линейное представление, подвергается распараллеливанию следующим образом.

1. Формирование множества *starts*.

2. Заполнение семантических уровней *L* на основе последовательности операций с точки зрения семантики. При этом нулевому уровню принадлежат конечные с точки зрения семантики позиции.

3. Удаление повторяющихся позиций из семантических уровней. Очевидно, что из двух позиций $a_{i(a)} \in L_i$ и $a_{i(a)} \in L_j$, при условии $i<j$, необходимо оставить в позицию, принадлежащую уровню иерархии с более высоким номером, так как в соответствии с правилами построения уровней это означает, что для выполнения операций на уровне $i<k<j$, уже требуются результаты операции $a_{i(a)}$.

4. Далее рассматриваются все семантические уровни поочередно, кроме самого верхнего, начиная с нулевого. Множество *united* изначально пустое.

4.1. Формируется множество outs:

$$outs=\{a_{i(a)}\} \mid a_{i(a)} \in L_i \wedge a_{i(a)} \notin starts \wedge a_{i(a)} \notin united .$$

4.2. Если множество *outs* пустое, то формирование переходов по управлению не требуется. Рассмотрение уровня иерархии завершается.

4.3. Если $|L_{i+1}|=1$ и $|outs|=1$ и $\exists z^S_{i(z^S)}\left(L_{i+1} \subseteq I_A(z^S_{i(z^S)}) \wedge outs_0 \in O_A(z^S_{i(z^S)})\right)$, то добавляется примитивный переход *z*, для которого: $I_A(z)=L_{i+1}$ и $O_A(z)=outs$.

4.4. Если $|L_{i+1}|=1$ и $|outs|>1$, то добавляется переход *z* типа fork, для которого:

$$I_A(z)=L_{i+1};$$

$$O_A(z)=\{a_{i(a)}\} \mid a_{i(a)} \in outs \wedge \exists z^S_{i(z^S)}\left(\begin{array}{l} a_{i(a)} \in O_A(z^S_{i(z^S)}) \wedge \\ \wedge L_{i+1} \subseteq \mathrm{I}_A(z^S_{i(z^S)}) \end{array}\right).$$

4.5. Если $|L_{i+1}|>1$ и $|outs|=1$, то добавляется переход z типа join, для которого:

$$I_A(z)=\{a_{i(a)}\} \mid a_{i(a)} \in L_{i+1} \wedge \exists z^S_{i(z^S)}\left(\begin{array}{l} outs_0 \in O_A(z^S_{i(z^S)}) \wedge \\ \wedge a_{i(a)} \in I_A(z^S_{i(z^S)}) \end{array}\right);\ O_A(z)=outs .$$

4.6. Если $|L_{i+1}|>1$ и $|outs|>1$, то необходимо выделить несколько групп позиций G_s, для которых формируются переходы *z* типа synchro. В начале формирования каждой группы в множество G_s помещается случайная, еще нераспределенная позиция из множества *outs*.

$$G_s=\{a_i\}\ |a_{i(a)}\in outs\wedge\exists a_{j(a)}\in G_s\wedge$$

$$\exists a_{k(a)}\in L_{i+1}\wedge\exists z^S_{i(z^S)}\exists z^S_{j(z^S)}\begin{pmatrix}a_{k(a)}\in I_A(z^S_{i(z^S)})\wedge a_{k(a)}\in I_A(z^S_{j(z^S)})\wedge\\ \wedge a_{i(a)}\in O_A(z^S_{i(z^S)})\wedge a_{j(a)}\in O_A(z^S_{j(z^S)})\end{pmatrix}.$$

Анализ позиций множества *outs* выполняется заново при каждом обнаружении новой позиции, принадлежащей текущей группе.

$$I_A(z)=\{a_{i(a)}\}\ |a_{i(a)}\in L_{i+1}\wedge\exists a_{j(a)}\in G_s\wedge\exists z^S_{i(z^S)}\begin{pmatrix}a_{i(a)}\in I_A(z^S_{i(z^S)})\wedge\\ \wedge a_{j(a)}\in O_A(z^S_{i(z^S)})\end{pmatrix};$$

$$O_A(z)=G_s.$$

4.7. Позиции, для которых были сформированы переходы по управлению, переносятся из множества *outs* в множество *united*.

4.8. Если $|outs|>0$, то остались позиции, для которых не определены переходы, в таком случае алгоритм продолжается с шага 7, но вместо L_{i+1} рассматривается следующий уровень, иначе начинается рассмотрение следующего семантического уровня.

В результате выполнения данных шагов будет получено множество переходов по управлению для текущего уровня иерархии. Осуществив восстановление прежнего уровня представления сети, можно получить минимально последовательную модель сети, которая отображает максимально возможный параллелизм в процессе. Анализ подобного представления может позволить оценить необходимость распараллеливания процесса в целом, а также служить основой для дальнейших оптимизаций.

Заключение. Применение описанных алгоритмов распараллеливания вычислительных процессов с применением расширенной сети Петри с семантическими связями позволит выполнить оптимизацию, в результате чего может быть получена модель более эффективной реализации процесса. Основываясь на получаемой в результате модели, отображающей максимально возможный параллелизм операций, применяя различные оптимизационные алгоритмы, можно сформировать модели процессов, рассчитанных на выполнение заданным числом исполнителей.

Исследование выполнено при финансовой поддержке РФФИ и Правительства Тульской области в рамках научного проекта 19-41-710003 р_а.

Список литературы

1. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

2. В.В. Слободской. Задача распараллеливания ациклического алгоритма // Вестник Нижегородского университета им. Н.И. Лобачевского, 2008, № 5. С. 113 – 119.

3. Марлей В. Е., Воробьев В. И., Крылов Р. А., Петров М. Ю., Быков Я. А. Автоматизация распараллеливания программ на основе анализа информационных связей // Программные продукты и системы. 2005. №1. С. 2–6.

4. О.С. Крюков. Структурный анализ моделей процессов и выявление циклических структур // Известия Тульского государственного университета. Технические науки. 2021. Вып. 4. С. 322 – 330.

5. О.С. Крюков. Автоматизация выявления ветвлений в моделях процессов // Известия Тульского государственного университета. Технические науки. 2021. Вып. 4. С. 435 – 440.

6. М.П. Маслаков, А.Г. Дедегкаев. Метод минимизации модифицированных сетей Петри на основе их эквивалентных преобразований // Universum: Технические науки: электрон. научн. журн. 2013. № 1 (1) [Электронный ресурс]. URL: https://7universum.com/ru/tech/archive/item/788 (дата обращения: 15.05.2022).

7. Д.Б. Борзов, С.А. Дюбрюкс, В.С. Титов. Метод объединения и разделения циклических участков последовательных наследуемых программ // Известия высших учебных заведений. Приборостроение. №2. 2009. С. 60 –65.

8. П.Ю. Ткачев, Д.Б. Борзов, И.Е. Чернецкая. Метод и алгоритм поиска линейных участков внутри циклов с последующим распараллеливанием // Известия Юго-Западного государственного университета. № 5 (62). 2015. С. 16 – 20.

9. С.А. Гуда. Автоматическая расстановка синхронизаций при распараллеливании гнезд циклов на основе решетчатого графа // Вестник РГУПС, №3/2019. С. 81 – 87.

10. Ткачев П.Ю., Борзов Д.Б. Метод распараллеливания циклов со счетчиком // Приборостроение. 2015. №2. С. 104 – 108.

11. Anna Voloshko, Oleg Kryukov. Extended Petri Nets Based Approach for Simulation of Distributed Manufacturing Processes // The 9th Mediterranean Conference on Embedded Computing (MECO 2020), 2020. P. 508 – 511.

Крюков Олег Сергеевич, аспирант, ol_kryukov97@mail.ru, Россия, Тула, Тульский государственный университет,

Волошко Анна Геннадьевна, канд. техн. наук, доцент, atroshina@mail.ru, Россия, Тула, Тульский государственный университет,

Ивутин Алексей Николаевич, д-р техн. наук, профессор., заведующий кафедрой, alexey.ivutin@gmail.com, Россия, Тула, Тульский государственный университет

AUTOMATION OF THE FORMATION OF THE MAXIMUM PARALLELISM IN THE PROCESS

O.S. Kryukov, A.G. Voloshko, A.N. Ivutin

The paper considers the need to create an automated tool for parallelizing computational processes, proposes a method and algorithms for the formation of a minimally consistent model of a nonlinear process. The methodology is based on the representation of the process in the form of an extended Petri net model with semantic relations.

Key words: Petri nets, modeling, process model, process structure, parallelism, minimally sequential representation.

Kryukov Oleg Sergeevich, postgraduate, ol_kryukov97@mail.ru, Russia, Tula, Tula State University,

Voloshko Anna Gennad'evna, candidate of technical sciences, docent, atroshina@mail.ru, Russia, Tula, Tula State University,

Ivutin Alexey Nikolaevich, doctor of technical sciences, professor, head of department, alexey.ivutin@gmail.com, Russia, Tula, Tula State University

УДК 62-51
DOI: 10.24412/2071-6168-2022-10-237-245

ЦИФРОВОЙ СКОЛЬЗЯЩИЙ РЕЖИМ В СЛЕДЯЩЕМ ПРИВОДЕ С ЛИНЕЙНОЙ ПОВЕРХНОСТЬЮ СКОЛЬЖЕНИЯ

Н.Н. Макаров, Е.В. Плыкина

Рассмотрено применение цифрового режима скольжения для линейной системы с линейной поверхностью скольжения. Под цифровым скользящим режимом понимается такой режим работы системы, при котором дискретное по времени управление переводит изображающую точку в фазовом пространстве на поверхность скольжения к началу следующего такта управления. Приведены условия существования такого режима и конечные соотношения для вычисления управляющего воздействия, реализующего скользящий режим работы системы. Рассмотрен пример реализации такого режима в электрическом следящем приводе. Результаты моделирования демонстрируют перспективность использования цифрового скользящего режима в проектировании следящих приводов.

Ключевые слова: следящая система, дискретное управление, скользящий режим.

Введение. Известно, что применение в следящей системе релейного регулятора, работающего в скользящем режиме [1, 2], позволяет обеспечить не только высокую, при определённых условиях практически идеальную, точность слежения, но и робастность системы по отношению к параметрам объекта управления. В частности, оптимальные по быстродействию системы [2], в полной мере обладают указанными свойствами. При практической реализации оптимальных по быстродействию регуляторов возникают, однако, некоторые трудности. Первая из них связана с тем, что для управления необходимо использовать полный вектор состояния системы, вторая обусловлена нелинейностью используемых обратных связей. Широкое распространение цифровых регуляторов, основанных на использовании микроконтроллеров, с одной стороны расширяет возможности применения сложных нелинейных законов