Автоматика и телемеханика, № 8, 2014

© 2014 г. Д.А. ЗАЙЦЕВ, д-р техн. наук (daze@acm.org)
(Международный гуманитарный университет, Одесса, Украина)

ПАРАДИГМА ВЫЧИСЛЕНИЙ НА СЕТЯХ ПЕТРИ

Показано, что пардигма вычислений на сетях Петри обеспечит существенное ускорение вычислений и сокращение трудоемкости разработки программного обеспечения за счет массового параллелизма и асинхронного характера процессов обработки информации. Разработан язык программных сетей Петри, предложены альтернативные подходы реализации парадигмы на микро- и макроуровнях, уточнены оценки сложности ранее построенной универсальной сети Петри, являющейся прототипом соответствующего процессора. Указаны направления практической реализации парадигмы: разработка технологии программирования на нагруженных сетях Петри, технологии трансляции нагруженных сетей Петри в ингибиторные сети Петри и эффективных аппаратных процессоров, исполняющих программы на языке ингибиторных сетей Петри. Использование класса сетей Слепцова с кратным запуском перехода на шаге обеспечивает быстрые вычисления.

1. Введение

Сеть Петри [1–7] широко известна как средство моделирования дискретных параллельных процессов. Кроме того, сеть Петри все чаще применяют как язык программирования [8–13] параллельных вычислений [14]. С построением универсальной сети Петри в явном виде [15, 16] создаются объективные предпосылки для всесторонней разработки парадигмы вычислений на сетях Петри в замкнутом виде, не использующей других формализмов для организации вычислений; настоящая статья посвящена предварительной проработке основных направлений исследований.

2. Сети Петри и моделирование систем

Сеть Петри [1, 2] представляет собой двудольный ориентированный граф, на котором задан динамический процесс. Одна доля вершин, называемых позициями и изображаемых в виде окружностей, моделирует условия; вторая доля вершин, называемых переходами и изображаемых в виде прямоугольников, моделирует события. Динамические элементы, именуемые фишками, размещаются внутри позиций и перемещаются в сети в результате срабатывания переходов. Позиции, переходы, дуги и фишки рассматриваются либо как элементарные, либо как нагруженные дополнительными атрибутами и функциями, определяя множество различных классов сетей Петри [3, 5, 6].

Сети Петри традиционно применяют для моделирования систем в широком спектре приложений [3–7]. Элементарная сеть Петри располагается между конечным автоматом и машиной Тьюринга, позволяя применять аналитические методы исследования [1, 2]. Нагруженные сети Петри [3, 5] часто

рассматривают как графический язык систем имитационного моделирования и используют подход проверки модели (*model-checking*) либо статистический анализ.

Как правило, классические сети Петри применяют для доказательства корректности (верификации) систем, а нагруженные сети – для оценки эффективности систем.

3. Программирование на языке ингибиторных сетей Петри

Так как доказана полнота по Тьюрингу ингибиторной сети Петри [1, 2], то можно программировать на языке сетей Петри. Напомним, что в ингибиторной сети Петри (ИСП) добавлен специальный тип дуги, направленной из позиции в переход, для проверки маркировки позиции на ноль. Многие авторы рассматривают сеть Петри как графическую подложку асинхронных параллельных языков программирования [8–13]. Но язык сетей Петри достаточно часто используют как вспомогательный промежуточный язык, который, в конечном счете, транслируется в инструкции некоторого классического процессора.

На рис. 1 показана сеть Петри, реализующая операцию сложения целых чисел: *а* – исходные данные помещены в позиции x и y, попадание фишки в позицию s запускает вычисления; *б* – полученный результат наблюдается в позиции z когда позиция f маркирована. Ингибиторная дуга обозначена полым кругом на ее конце; дуга со сплошным кругом является проверочной и представляет собой аббревиатуру двух дуг противоположного направления. Позиции сети Петри обозначены буквами p, переходы – буквами t. Представленная в подрисуночной подписи последовательность срабатывания переходов не является единственной допустимой: порядок запуска переходов $t2$ и $t3$ допускает произвольные перестановки.

Современная управляемая моделью разработка (*model-driven development*) использует преимущество моделей в форме сетей Петри, которые трансформируются в спецификации систем в процессе проектирования. Однако заключительный этап предполагает трансляцию сети Петри в последовательность инструкций, что нивелирует многие преимущества.

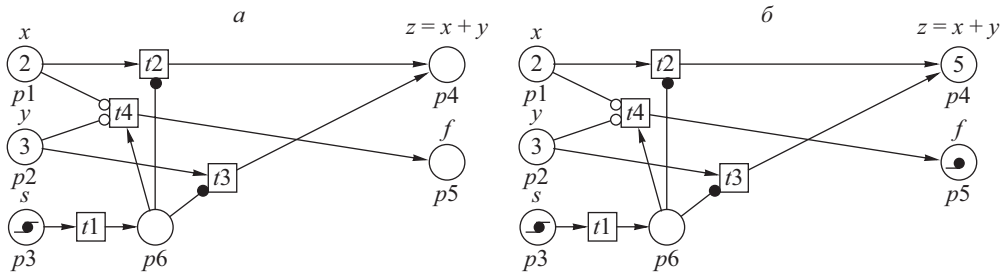

Рис. 1. Ингибиторная сеть Петри сложения целых чисел *ADD*: *а* – начальное состояние; *б* – конечное состояние; последовательность срабатывания переходов $t1t2^2t3^3t4$.

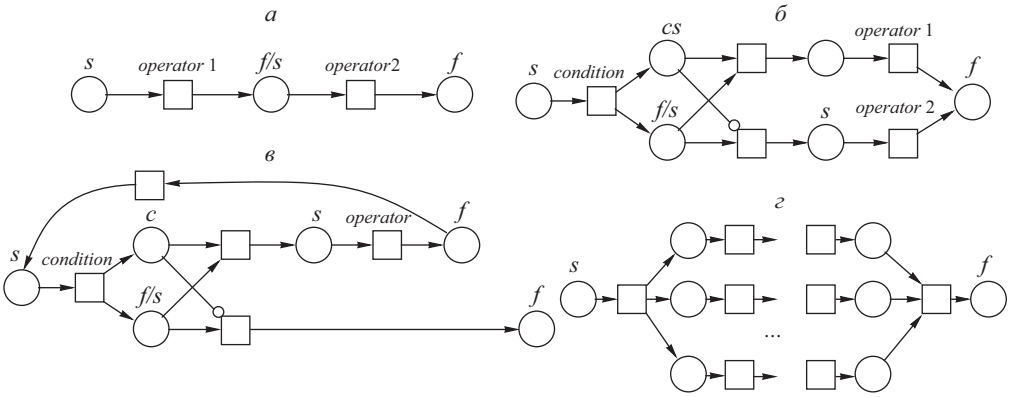

Рис. 2. Представление операторов языка программирования сетями Петри: *а* – последовательность; *б* – ветвление; *в* – цикл; *г* – параллельное выполнение.

Асинхронный подход противопоставляют традиционному параллельному подходу, в котором алгоритмы являются изначально последовательными и требуются методы их распараллеливания для исполнения на мультипроцессорных системах и кластерах. В асинхронном подходе алгоритмы разрабатывают, сохраняя естественный параллелизм предметной области таким образом, что их последующее распараллеливание не требуется.

На рис. 2 основные алгоритмические конструкции языка параллельного программирования представлены ингибиторной сетью Петри [15]: *а* – последовательность *operator*1; *operator*2; *б* – ветвление *if* (*condition*) *operator*1 *else operator*2; *в* – цикл *while* (*condition*) *operator*; *г* – параллельное выполнение *parbegin . . . parend*. Буквой *c* обозначен результат вычисления условия *condition*. Однако написание программ непосредственно на языке сетей Петри позволяет выполнять произвольные комбинации потоков данных и потоков управления; более того, потоки управления и данных могут быть настолько тесно интегрированы, что становятся неразличимыми по отдельности.

Сеть потока управления формируется путем композиции представленных на рис. 2 графических шаблонов для последовательности, ветвления, цикла и параллельного выполнения. Использованы две основные операции для композиции потока управления: последовательная композиция и подстановка перехода. Операторная сеть имеет входные и выходные контактные позиции вместе с обязательной парой позиций: входной – запуск (s) и выходной – завершение (f); другие контактные позиции используются для представления входных и выходных переменных. Последовательная композиция выполняется путем слияния позиции завершения (f) предыдущего оператора с позицией запуска (s) следующего оператора, как показано на рис. 2,*а*.

4. Организация работы с переменными и вызова процедур

В [15–18] предложен ряд соглашений, позволяющих легко транслировать традиционные программы в ИСП, а также разрабатывать программы в виде ИСП непосредственно. Систему соглашений назовем программной инги-

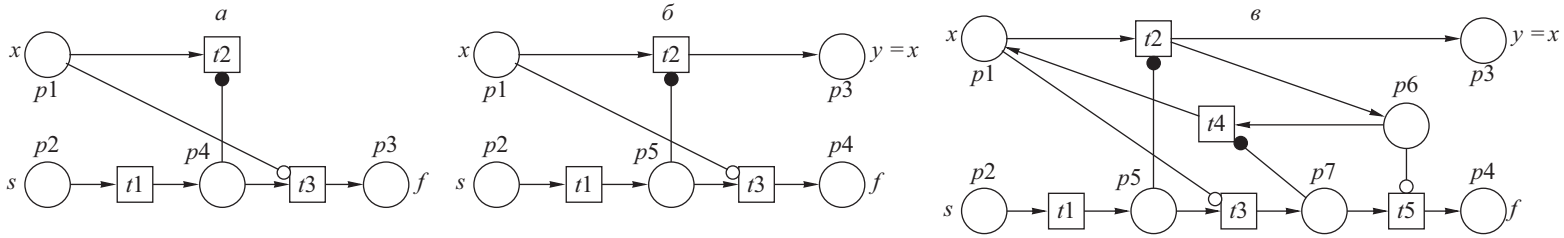

Рис. 3. Вспомогательные сети копирования значений переменных: *а* – *CLEAN* ($x := 0$); *б* – *MOVE* ($y := x, x := 0$); *в* – *COPY* ($y := x$).

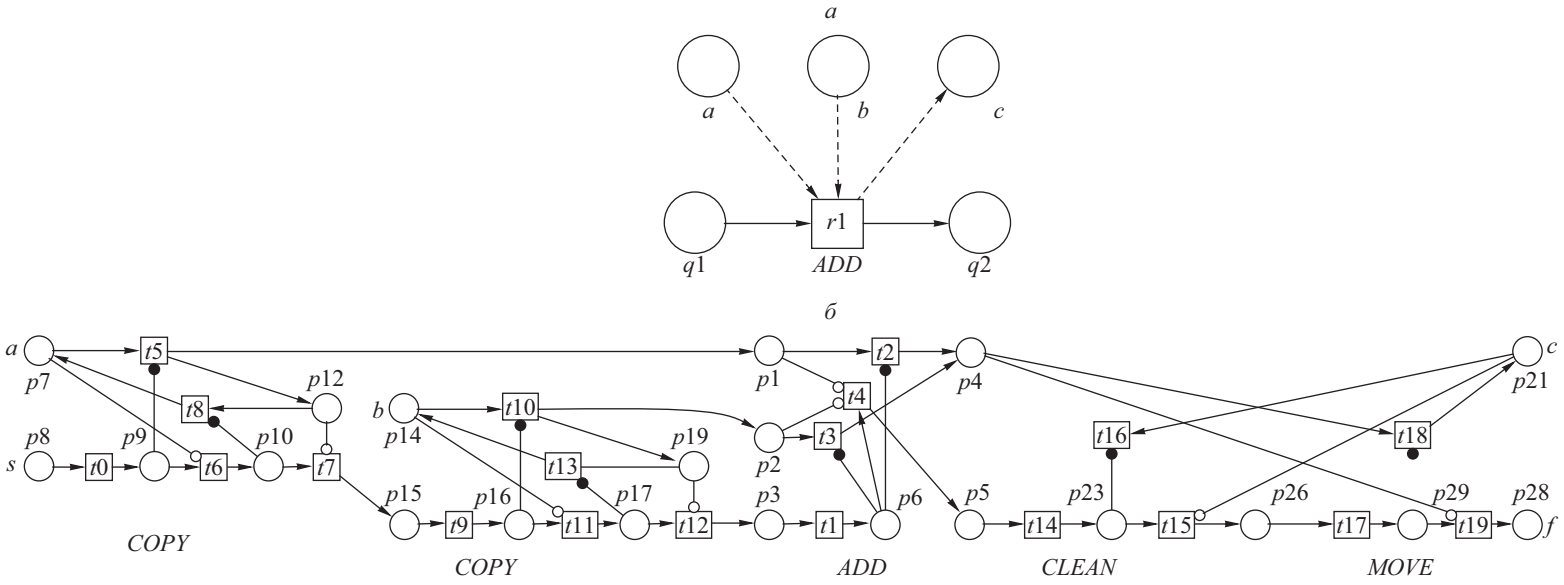

Рис. 4. Пример использования оператора (подсети) – компиляция в низкоуровневую сеть: *а* – типовой вызов *ADD* ($c := a+b$); *б* – результирующая низкоуровневая сеть.

биторной сетью Петри (ПИСП). ПИСП получают в результате композиции сети, описывающей поток (потоки) управления и множества позиций, представляющих глобальные статические переменные. Использованы глобальные целочисленные переменные; каждая переменная представлена соответствующей позицией сети Петри. Кроме того, параметры операторов представлены входными и выходными контактными позициями. Перед запуском оператора переменные копируются в его входные позиции, а после завершения заменяются значениями из выходных позиций. В дальнейших реализациях возможно поразрядное хранение переменных.

Специфической особенностью ПИСП являются штриховые и пунктирные дуги, соединяющие позиции переменных с переходами операторов. Они обеспечивают присваивание значений переменным. На рис. 3 представлены основные вспомогательные сети копирования переменных. Эти сети могут использоваться либо независимо, либо как расширение штриховых (пунктирных) дуг.

Подстановка перехода представляет собой использование операторов (процедур). Переход замещается соответствующей подсетью; имя подсети записывается возле перехода. Как и в моделирующей системе *CPN Tools* [5], подстановка перехода требует указания:

а) имени подсети;

б) отображения контактных позиций.

Количество контактных позиций и их типы (входная, выходная, входная/выходная) должны совпадать. Например, подстановка, изображенная на рис. 4, задается следующим отображением: $r1 \to ADD$, $a \to x$, $b \to y$, $q1 \to s$, $c \to z$, $q2 \to f$. Буквами q на рис. 4 обозначены позиции потока управления сети высокого уровня. При композиции программы отображение контактных позиций может быть указано неявно, что в большинстве случаев не является неоднозначным.

ПИСП может быть полностью собрана с использованием функциональных подсетей (кланов) [7], содержащих только входные и выходные позиции. Для этого в таких вспомогательных сетях как *COPY* и *MUL* значение из входной позиции может быть предварительно скопировано в некоторую внутреннюю позицию.

Входные штриховые дуги оператора обозначают расширение в подсеть *COPY*; выходные – *CLEAN* и *MOVE*. На рис. 4 показан пример полной подстановки с копированием значений переменных: ПИСП, изображенная на рис. 4,*а*, компилируется в низкоуровневую сеть, представленную на рис. 4,*б*, с использованием подсетей рис. 1 и рис. 3,*а*–3,*в*. Для нескольких входных (выходных) переменных создаются цепочки подсетей *COPY* (*CLEAN* и *MOVE*).

Пунктирные линии используются в случае, когда значение переменной потребляется полностью оператором; они раскрываются как подсети *MOVE*. Когда переменная является как входной, так и выходной для некоторого оператора, используется двунаправленная штриховая дуга, которая обозначает *MOVE* перед и *MOVE* после выполнения оператора. Присваивание *ASSIGN* раскрывается как *CLEAN* и *COPY*. Варианты расширения в низкоуровневые сети для различных типов копирующих дуг проиллюстрированы на рис. 5.

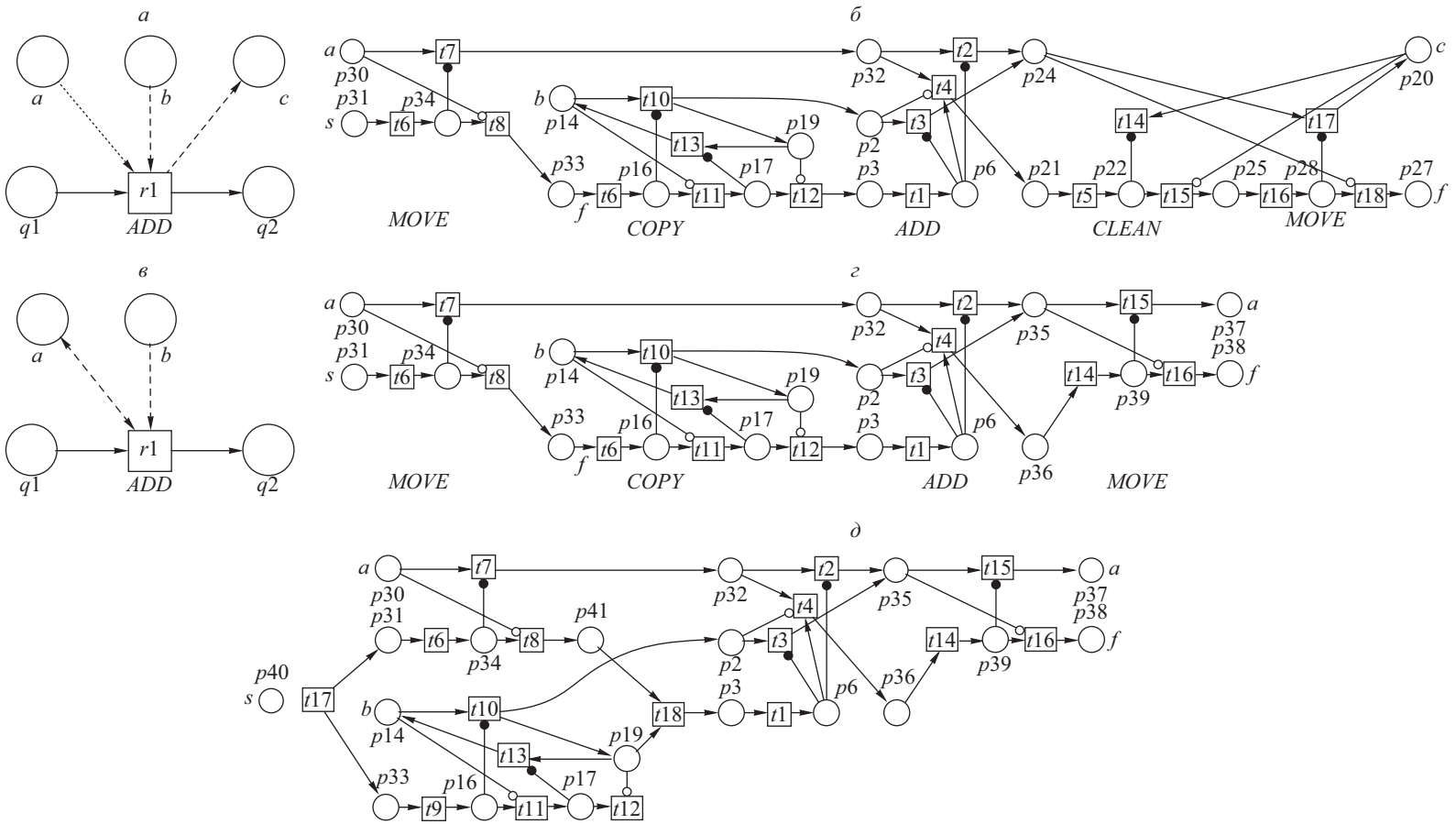

Рис. 5. Варианты расширения для различных типов копирующих дуг: *а* – вызов ADD ($c := a + b, a := 0$); *б* – расширение *а*; *в* – вызов ADD ($a := a + b$); *г* – расширение *в*; *д* – расширение *в* с параллельным копированием.

Задача оценки размера ПИСП в элементарных позициях, переходах и дугах является нетривиальной. Сеть содержит как элементарные переходы, так и составные и, кроме того, при отображении позиций выполняется их совмещение и расширение пунктирных и штриховых дуг.

Когда выполняется подстановка перехода (или оценивается его размер), рассматриваются штриховые (пунктирные) дуги. Анализ ПИСП позволяет выделить типовые комбинации копирующих дуг; наиболее распространенные из них представлены на рис. 5. Формула оценки достаточно простая: сумма переходов; сумма дуг; сумма позиций минус количество объединяемых позиций. Так, для оператора OP, имеющего ms позиций с mi входными позициями и mo выходными позициями, создается цепочка из $mi-1$ подсетей $COPY$ перед OP и цепочка $mo-1$ связок $CLEAN$, $MOVE$ после OP; позиции запуска и завершения при этом не рассматриваются.

Оценим количество объединенных позиций (в случае $mi > 1$, $mo > 1$):

$$mfp = mi + mo + mcc + mcm + mcmcm,$$

где использованы следующие аббревиатуры:

mcc: $COPY,COPY$,
mcm: $CLEAN,MOVE$,
$mcmcm$: $CLEANMOVE,CLEANMOVE$

и

$$mcc = mi - 2, \quad mcm = 2 \cdot (mo - 1), \quad mcmcm = mo - 2.$$

Таким образом, общее число позиций вычисляется как

$$mse = ms + (mi - 1) \cdot mCOPY + (mo - 1) \cdot (mCLEAN + mMOVE) - mfp.$$

Найденный размер ПИСП может быть далее сокращен без реорганизации ПИСП за счет использования модифицированных правил подстановки перехода, что является аналогичным оптимизации во время компиляции. Рассмотренная подстановка перехода аналогична инлайновой подстановке кода процедур. При обычном вызове код процедуры не дублируется, а реализуется передача и возврат управления. Например, если оператор ADD вызывается 9 раз, то соответствующая подсеть вставляется 9 раз в результирующую низкоуровневую сеть. Альтернативой является одна копия оператора (подсети) и переключение потока управления при его вызове. Различия иллюстрирует рис. 6; переменные опущены для краткости.

В случае, изображенном на рис. 6,*а*, каждый из k экземпляров вызова оператора OP приводит к полной вставке подсети OP (вместе с фрагментами копирования переменных). В случае, изображенном на рис. 6,*б*, вставляется лишь единственная копия подсети OP и добавляются $2k$ переходов и $k + 2$ позиции соответственно для вызова оператора ci, для запоминания номера вызова qi и возврата ri. В общем случае переключение вызова оператора отделено от копирования переменных, которые могут отличаться для

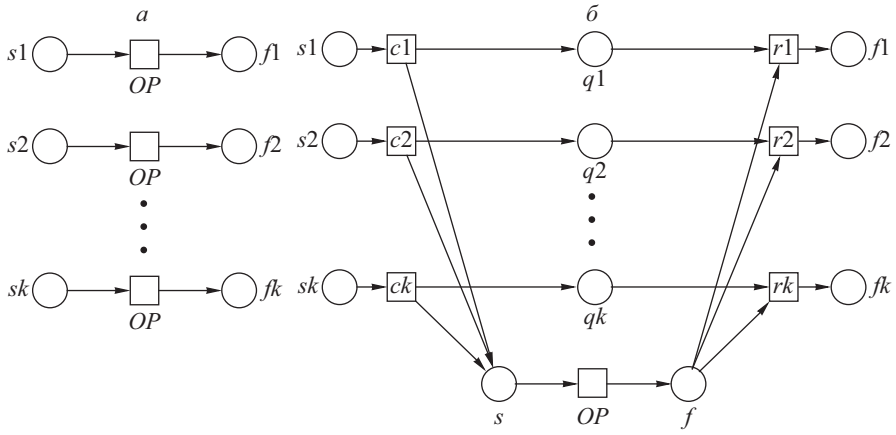

Рис. 6. Реализация подстановки перехода: *а* – схема индивидуальной вставки; *б* – схема вызова-возврата.

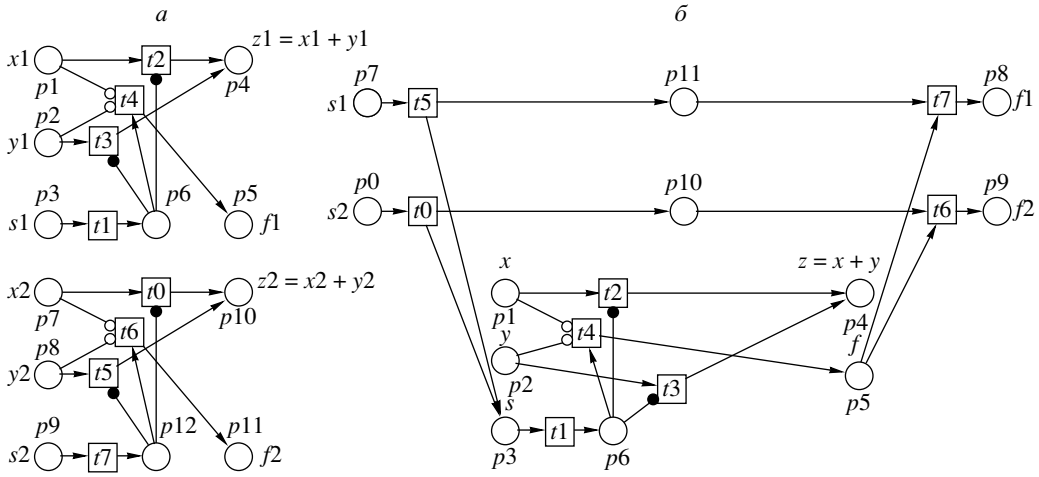

Рис. 7. Пример реализации подстановки перехода: *а*– вставка; *б*– вызов-возврат: $x1$, $x2$ копируются в x; $y1$, $y2$ копируются в y; $z1$, $z2$ копируются из z.

различных вызовов. Конкретный пример реализации подстановки перехода представлен на рис. 7.

Оценим сокращение общего количества позиций, переходов и дуг соответственно:

$$RP = (k-1)(ms-2) - k - 2,$$
$$RT = (k-1)ns - 2k,$$
$$RA = (k-1)(as-2) - 3k,$$

где ms – количество позиций подсети, ns – количество переходов, as – количество дуг. Заметим, что для небольших сетей сокращение может быть нулевым и даже отрицательным.

5. Универсальная сеть Петри как прототип процессора сетей Петри

Концепция универсальной сети Петри (УСП) аналогична концепциям других универсальных систем, например универсальной машине Тьюринга или универсальному нормальному алгоритму Маркова. УСП принимает на вход заданную сеть Петри и ее начальную маркировку и производит на выходе достижимую конечную маркировку и последовательность запуска переходов заданной сети. Основная идея состоит в том, что как входная программа, так и средства ее исполнения специфицированы с помощью одной и той же формальной системы, для УСП – с помощью некоторого класса сетей Петри.

В качестве рабочей формальной системы выбран класс ингибиторных сетей Петри. Граф ингибиторной сети Петри [1, 2] – это четверка $G = (P, T, B, D)$, где $P = \{p_1, \ldots, p_m\}$ – конечное множество вершин, называемых позициями, $T = \{t_1, \ldots, t_n\}$ – конечное множество вершин, называемых переходами, а отображения $B : P \times T \to \mathbb{N} \cup \{-1\}$ и $D : T \times P \to \mathbb{N}$ задают входящие и исходящие дуги переходов соответственно и их кратность, $\mathbb{N}$ – множество целых неотрицательных чисел; нулевое значение отображений B, D обозначает отсутствие дуги, ненулевое – кратность дуги, специальное значение (-1) обозначает ингибиторную дугу.

Динамика ингибиторной сети представляет собой пошаговый процесс изменения ее маркировки в результате срабатывания переходов [1, 2] и может быть формально описана системой [15], традиционно называемой уравнением состояния:

$$\begin{cases} q_j^k = q_j^{k-1} - x(b_{l,j}) + d_{l,j}, \quad 1 \leqslant j \leqslant m, \\ u(t_i) = \bigwedge\limits_{j=\overline{1,m}} \left(y(b_{i,j}) \wedge (q_j^{k-1} = 0) \vee \bar{y}(b_{i,j}) \wedge (q_j^{k-1} \geqslant b_{i,j}) \right), \\ u(t_l) = 1, \quad 1 \leqslant l \leqslant n, \\ k = 1, 2, \ldots, \\ x(b) = \begin{cases} b, & b \geqslant 0, \\ 0, & b = -1, \end{cases} \qquad y(b) = \begin{cases} 0, & b \geqslant 0, \\ 1, & b = -1. \end{cases} \end{cases}$$

Первая строка системы описывает изменение маркировки при срабатывании перехода t_l; функция $u(t_i)$ во второй строке представляет условие возбуждения перехода t_i на текущем шаге k, а третья строка задает недетерминированный выбор срабатывающего перехода t_l из множества возбужденных, четвертая строка задает порядок следования шагов; вспомогательные отображения x и y служат для задания декремента маркировки и распознавания ингибиторной дуги соответственно.

Универсальную сеть Петри можно рассматривать как прототип процессора, который выполняет программы, написанные на языке ингибиторных сетей Петри. Для построения УСП в явном виде [15] был выбран наиболее удобный для формальных доказательств способ: алгоритм перевычисления маркировки в соответствии с уравнением состояний представлен на Си-подобном языке, а затем закодирован с использованием методики, рассмотренной в предыдущих разделах, ингибиторной сетью Петри (рис. 8).

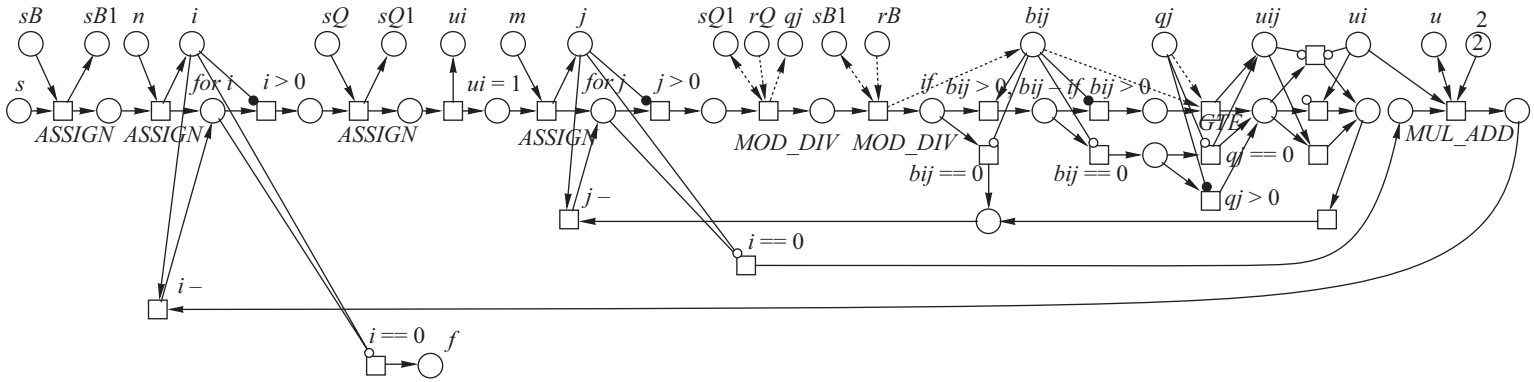

Рис. 8. Фрагмент УСП для нахождения возбужденных переходов.

```
/* традиционная реализация шага сети Петри */
for(i=n; i>0; i--)
{
    for(j=m; j>0; j--)
    {
      <обработка дуг, соединяющих переход i и позицию j>
    }
}
```

Недостатком указанного подхода с практической точки зрения является его низкая эффективность, так как не использованы средства асинхронного параллельного стиля программирования с множественными потоками данных и управления. Практическая реализация процессоров сетей Петри требует разработку эффективных универсальных сетей, обладающих как минимальным размером [16], так и низкой вычислительной сложностью за счет асинхронной параллельной реализации и специального кодирования данных.

Недетерминированность поведения классических сетей Петри предоставляет возможность гибкого назначения имеющихся ресурсов для выбора оптимальной трассы выполнения (последовательности запуска переходов). Большинство представленных в статье сетей являются детерминированными по конечному результату (в тупиковой маркировке), кроме того, в [16] введен и применен класс детерминированных ингибиторных сетей Петри.

6. Уточнение оценок сложности универсальной сети Петри

ИСП можно представить [15] как $N = (m, n, B, D, Q_0)$, где $m = |P|$ и $n = |T|$ – количества позиций и переходов соответственно, B и D – матрицы входных и выходных дуг переходов соответственно, вектор Q_0 задает начальную маркировку. Уточним указанные в [15] оценки сложности, которые предполагают ограниченность обрабатываемой сети X.

Предположим, что $\hat{a}$ – максимальная кратность дуги и $\hat{q}$ – максимальная маркировка позиции, и получим количество бит для хранения компонентов матриц $nba = \lceil \log_2 \hat{a} \rceil$ и вектора $nbq = \lceil \log_2 \hat{q} \rceil$. Тогда общий объем памяти, необходимый для хранения ИСП, оценивается так:

$$SC = 2m \cdot n \cdot nba + m \cdot nbq + nbm + nbn, \quad nbn = \lceil \log_2 \hat{n} \rceil, \quad nbm = \lceil \log_2 \hat{m} \rceil.$$

Маркировка сети кодируется как

$$sQ = \varphi(Q) = \sum_{j=0}^{m-1} r^j \cdot q_j < r^m, \quad \text{где} \quad r = \hat{q} + 1.$$

Количество битов для хранения маркировки

$$SCq = \log_2 r^m \approx \log_2 \hat{q}^m = m \cdot \log_2 \hat{q} = m \cdot nbq.$$

Матрица B (как и матрица D) кодируется как

$$sB = \varphi(B) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} r^{m \cdot i + j} \cdot b_{i,j} < r^{m \cdot n}, \quad \text{где} \quad r = \hat{a} + 1.$$

Количество битов для хранения матрицы

$$SCa = \log_2 r^{m \cdot n} \approx \log_2 \hat{a}^{m \cdot n} = m \cdot n \cdot \log_2 \hat{a} = m \cdot n \cdot nba.$$

Учитывая, что верхняя граница маркировки позиции УИСП может быть оценена как $SC = \max(SCq, SCa)$, получим общую оценку емкостной сложности

$$SC_{UIPN} = m_{UIPN} \cdot SC = 450 \cdot SC = \text{const} \cdot SC = \text{const}.$$

Таким образом, УСП исполняет ограниченную сеть X с постоянной емкостной сложностью. Константа может быть улучшена (уменьшена), так как, например, маркировки позиций потоков управления принадлежат множеству $\{0, 1\}$; возможно также указание меньших верхних границ и для ряда других позиций.

Заметим, что последовательности запуска переходов не рассматривались, так как их сохранение не является обязательным. Результаты могут быть уточнены для сохраненной последовательности запуска Z:

$$SCz = k \cdot nbn + nbk, \quad nbk = \lceil \log_2 k \rceil.$$

Полученная сложность является линейной по отношению к числу шагов обрабатываемой сети X. Методика хранения разреженных матриц может быть рекомендована в случае, когда менее чем одна треть элементов матрицы ненулевая, потому что хранение элемента использует примерно тройное пространство для хранения индексов.

Оценка временно́й сложности требует предварительного рассмотрения организации процесса выполнения сети Петри. В случае когда ИСП рассматривается как самовыполняющаяся, каждый шаг занимает одну единицу времени. Процесс может быть ассоциирован с некоторой абстрактной машиной, содержащей средства параллельной оценки условий возбуждения всех переходов, а также обособленного арбитра для выбора срабатывающего перехода и схемы запуска перехода, что требует три элементарных такта машины.

Однако даже в этом случае возможны различные интерпретации, что требует дальнейшего уточнения. На каждом шаге при срабатывании перехода выполняются операции декремента и инкремента маркировки позиций. Если сложность этих операций не учитывается (принимается равной единице), то процесс их реализации соответствует использованию унарной системы счисления посредством дописывания и стирания единиц. Тогда емкостная сложность должна оцениваться по линейной шкале. В случае использования позиционной (двоичной) системы счисления, сложность выполнения операции пропорциональна логарифму хранимого числа.

В случае, когда ИСП предполагается выполняемой на некоторой последовательной скалярной машине, каждый шаг требует в худшем случае $m \cdot n + n + 2 \cdot m \approx O(m \cdot n)$ единиц времени: найти возбужденные переходы, выбрать срабатывающий переход и запустить его соответственно. Для k шагов сложность равняется $O(k \cdot m \cdot n)$, но, принимая во внимание размер маркировки позиции, получаем

$$TC = O(k \cdot m \cdot n \cdot \log_2 \hat{q}).$$

Анализ основных циклов УСП [15] показывает три вложенных цикла (в худшем случае) с коэффициентами повторения k, m, n, что дает $O(k{\cdot}m{\cdot}n)$, но каждый подстановочный переход предполагает по крайней мере копирование маркировки в режиме перемещения каждой фишки, что дает общую оценку

$$TC_{UIPN} = O(k \cdot m \cdot n \cdot \hat{q}).$$

Таким образом, УСП моделирует сеть X за время $\hat{q}/\log_2 \hat{q}$. Для ограниченных сетей Пери $\hat{q} = \text{const}$ и не зависит от числа шагов сети X, т.е. временна́я сложность УСП является линейной.

Для получения оценок в худшем случае для неограниченных сетей Петри выразим $\hat{q}$ через k; запуск перехода добавляет не более $\hat{a}$ фишек в каждую позицию маркировки. Таким образом, в качестве верхней границы $\hat{q}$ выбрано $k{\cdot}\hat{a}$ (подразумевая нулевую начальную маркировку). В процессе кодирования маркировки ингибиторной сети Петри рассматривается ее запись в позиционной системе счисления с основанием $k \cdot \hat{a}$ и m позициями; тогда верхней границей кода маркировки обрабатываемой сети X является $(k \cdot \hat{a})^m$ или k^m для ординарной сети. Итак,

$$TC_{UIPN} = \frac{(k \cdot \hat{a})^m}{\log_2 k + \log_2 \hat{a}} \cdot TC,$$

что означает, что УСП моделирует сеть X за время $k^m/\log_2 k$, так как $\hat{a} = {}$ $= \text{const}$, где k – количество запущенных переходов целевой сети. В случае самовыполняющейся (параллельной) ИСП сложность домножается на $m \cdot n$; множитель не зависит от k и рассматривается как константа.

Полученный результат имеет достаточно простое объяснение: для абстрактной машины предполагается, что сложность выполнения операции пропорциональна логарифму длины операндов, но в настоящее время УСП реализует все операции посредством серий примитивов инкремента и декремента как в унарной системе счисления. Указанные рассуждения раскрывают дальнейшие направления исследований по построению эффективных УСП.

Большинство практически используемых ИСП являются ограниченными, что приводит к достаточно оптимистичным линейным оценкам. Для неограниченных ИСП сложность остается полиномиальной, хотя и с достаточно высокой степенью полинома, равной количеству позиций обрабатываемой сети. Построенная с использованием сложных цепочек кодирования сетей Петри в битеговые системы и прямой эмуляции машин Тьюринга минимальная универсальная сеть Петри [16] насчитывает всего 56 вершин (14 позиций и 42 перехода), однако ее вычислительная сложность является экспоненциальной.

7. Макро- и микроуровни реализации парадигмы вычислений на сетях Петри

Парадигма вычислений на сетях Петри может быть реализована на микроуровне либо на макроуровне. В случае реализации на макроуровне сеть Петри координирует фрагменты последовательного кода, написанные на обыч-

ном языке программирования, которые нагружают переходы, дуги и фишки. Таким образом, процессор сетей Петри добавляется к кластеру обычных процессоров.

Реализация на микроуровне предполагает полное описание алгоритмов сетью Петри, в котором не используются никакие другие средства (кроме сетей Петри) для представления потоков данных и потоков управления и операций над ними. Имеются определенные неудобства в описании арифметических операций сетями Петри и традиционно неэффективный способ их реализации, что препятствует практическому применению микроуровневого подхода.

Определенный разрыв между теорией асинхронного параллельного программирования с использованием сетей Петри и программирования для существующих компьютеров остается незаполненным на протяжении многих лет. Пожалуй, единственной областью, которая практически использует языки программирования на основе сетей Петри, являются программируемые логические контроллеры (ПЛК) [8]. Экспериментальные реализации языков программирования, основанных на сетях Петри [9–13], либо предполагают компиляцию в классические языки, либо интерпретацию программ на программно реализованных процессорах сетей Петри. Таким образом, используемые архитектуры аппаратных средств не позволяют в полной мере реализовать преимущества асинхронного подхода, так как асинхронный параллельный процессор эмулируется на последовательном синхронном оборудовании.

В последнее время построены универсальная сеть Петри (УСП) [15, 16] и сети Петри, которые исполняют заданную машину Тьюринга (СПМТ) [17] и нормальный алгоритм Маркова (СПНАМ) [18], обеспечивая преемственность концепций. УСП рассматривается как прототип процессора сетей Петри (ПСП), реализованного на сетях Петри, а СПМТ и СПНАМ – как прототипы сопроцессоров, выполняющих алгоритмы, записанные в последовательностях инструкций или продукций (подстановок). Таким образом, программа, написанная на языке сетей Петри, исполняется УСП, а другие фрагменты кода исполняются СПМТ, СПНАМ и другими специализированными сетями.

На рис. 8 представлен фрагмент УСП – подсеть, вычисляющая условия возбуждения переходов. Сеть построена по программе на языке Си и использует единственный поток управления, представленный продвижением единственной фишки из позиции s в позицию f. Переменные расположены в поименованных позициях в верхней части рисунка. Порядок обработки соответствует рассмотренному фрагменту кода на языке Си. Подсеть MUL_ADD используется для кодирования (шифрования) данных, подсеть MOD_DIV – для декодирования; подсеть GTE реализует сравнение целых чисел. Зашифрованные структуры данных имеют префикс s; например, зашифрованная маркировка обозначена sQ. Основание позиционной системы счисления, выбранной для шифрования, обозначено с помощью префикса r; например, основание системы счисления для шифрования маркировки обозначено rQ. В обозначениях арифметических операций использован синтаксис языка Си.

Сложившаяся ситуация характеризует выполненные подготовительные работы для начала производства процессоров сетей Петри и разработки компьютеров и кластеров с их использованием. И хотя все чаще встречаются

реальные программы, написанные на языке сетей Петри [8–13], что-то препятствует их быстрому развитию. Некоторые из препятствий будут выявлены и обсуждены в разделе 8.

8. Построение эффективной аппаратной реализации (процессора) сетей Петри

Во-первых, неудобно писать программы в элементарных классах сетей Петри, например в ингибиторных сетях Петри. Нагруженные сети Петри являются удобными и мощными, но их аппаратная реализация слишком трудоемка.

Во-вторых, конструкции, использованные в теоретических построениях, часто неэффективны в практических применениях. Построенная УСП [15] использует последовательные алгоритмы реализации арифметических операций, основанные на примитивах инкремента и декремента и рассматривает единственный поток управления, что крайне неэффективно, хотя удобно и наглядно в теории для индуктивных доказательств. Таким образом, необходимо разрабатывать новые модификации УСП, которые максимально задействуют преимущества асинхронного стиля обработки.

Итак, имеются два основных препятствия: отсутствие технологии программирования на сетях Петри и эффективной аппаратной реализации сетей Петри, использующей преимущества унифицированного асинхронного стиля обработки.

В реализациях на микроуровне класс ингибиторных сетей Петри задействован в качестве аналога системы команд процессора, а классы нагруженных сетей Петри – в качестве языков программирования, предназначенных для разработки приложений. Требуется разработать как технологию программирования в нагруженных сетях Петри, так и технологию трансляции нагруженных сетей Петри в ингибиторные сети Петри.

Программные процессоры сетей Петри используют последовательный подход, состоящий в проверке всех переходов на каждом шаге, что требует в худшем случае проверить все позиции для каждого перехода. Таким образом, если имеется трасса из k переходов, то для ее выполнения требуется около $m \cdot n \cdot k$ операций, как было показано в разделе 5. Для достижения массового параллелизма каждый переход из множества возбужденных должен запускаться независимо, что требует преодоления границ последовательного способа восприятия событий.

Новый тип аппаратного обеспечения, непосредственно и эффективно реализующего сеть Петри, может быть назван вычисляющей памятью (ВП). ВП запоминает структуру сети Петри и ее текущее состояние и обеспечивает независимое вычисление условий возбуждения по всем переходам. Требуется единственная локальная синхронизация в случае конфликта нескольких переходов за фишку. Блокирование инцидентных позиций в момент срабатывания перехода и сигнализация другим переходам (использующим эти позиции) перевычислить их условия возбуждения обеспечит целостность системы. Наиболее близким аналогом ПСП являются программируемые логические матрицы (ПЛМ), но они работают в синхронном режиме.

Синхронная и асинхронная концепции только превалируют одна над другой в различных формальных системах, но, как правило, не проявляются в чистом виде. Поведение классической сети Петри подразумевает синхронизацию на шаге, когда проверяются все переходы и только один из них выбирается и запускается. Класс синхронных сетей Петри, в которых запускается максимальное множество возбужденных переходов на шаге, является полным по Тьюрингу. Однако требуется новый класс сетей Петри без определенных шагов (бесшаговый), в котором все переходы функционируют независимо, но результат их запуска в некоторых заданных моментах времени наблюдения совпадает.

Разрыв в технологии программирования компенсируется посредством эффективной реализации основных арифметических, логических и других операций. Например, в первую очередь должны быть решены вопросы представления целых и вещественных чисел для обеспечения эффективных алгоритмов и разработка самих алгоритмов. Позиции представлены в виде счетчиков ограниченной емкости, что существенно модифицирует общую организацию процессоров по сравнению с прототипами. Асинхронная поразрядная реализация арифметических операций в параллельно-последовательном режиме существенно снижает временну́ю сложность.

9. Быстрые вычисления на сетях Слепцова

Экспоненциальная сложность универсальных сетей [15, 16] является основным препятствием на пути широкого применения сетей Петри в качестве общецелевого языка параллельного программирования. Практический аспект проблемы обусловлен экспоненциальной сложностью реализации в форме сетей Петри арифметических операций умножения и деления.

Анатолий Ильич Слепцов предложил использовать кратный запуск перехода на шаге при наличии кратных условий его возбуждения [19]. В дальнейшем концепция кратного запуска была развита при исследовании временны́х сетей с многоканальными переходами [20]. В [21] показано, что универсальная сеть, построенная в классе сетей с кратным запуском переходов, имеет полиномиальную временну́ю сложность. Таким образом, складываются объективные предпосылки для особого именования указанного класса сетями Слепцова: они вычисляют быстро, в то время как традиционные сети Петри вычисляют медленно.

В сети Слепцова переход t, имеющий входящую дугу кратности x из позиции p_1 и исходящую дугу кратности y в позицию p_2, срабатывает с кратностью q_1/x на текущем шаге, что приводит к следующей маркировке: $q_1 = q_1 \bmod x$, $q_2 = q_2 + y \cdot (q_1/x)$; деление выполняется нацело. При $x = 1$ переход реализует умножение на y, а при $y = 1$ – деление на x за один шаг работы сети.

10. Заключение

Выполнен обзор работ по организации вычислений на сетях Петри и предварительная проработка соответствующей парадигмы вычислений на основе

построенных универсальных систем, являющихся прототипами новых процессоров.

По сравнению с известными платформами и языками программирования для параллельных и распределенных вычислений [14] парадигма вычислений на сетях Петри обеспечивает мелкое гранулирование спецификаций на уровне отдельных событий, что является потенциальным источником массового параллелизма, наглядный графический язык и единство формализма, используемого как для программирования, так и для верификации программ, что особенно важно при реализации подхода управляемой моделью разработки (*model-driven development*). Использование класса сетей Слепцова с кратным запуском перехода на шаге обеспечивает быстрые вычисления.

Краеугольными камнями для дальнейшего продвижения парадигмы вычислений на сетях Петри являются: технология программирования на нагруженных сетях Петри, технология трансляции нагруженных сетей Петри в ингибиторные сети Петри и эффективные аппаратные процессоры, исполняющие программы на языке ингибиторных сетей Петри.

Основными преимуществами, которые оправдывают усилия на реализацию новой парадигмы вычислений, являются значительные ускорения вычислений за счет массового параллелизма обработки информации и сокращение сроков разработки программного обеспечения за счет сохранения параллелизма предметной области.

СПИСОК ЛИТЕРАТУРЫ

1. *Питерсон Дж.* Теория сетей Петри и моделирование систем. М.: Мир, 1984.
2. *Котов В.Е.* Сети Петри. М.: Наука, 1984.
3. *Слепцов А.И., Юрасов А.А.* Автоматизация проектирования управляющих систем гибких автоматизированных производств / Под ред. Б.Н. Малиновского. Киев: Техника, 1986.
4. *Ачасова С.М., Бандман О.Л.* Корректность параллельных вычислительных процессов. Новосибирск: Наука, 1990.
5. *Jensen K.* Colored Petri Nets—Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag, 1997. V. 1–3.
6. *Ломазова И.А.* Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. М.: Науч. мир, 2004.
7. *Zaitsev D.A.* Clans of Petri Nets: Verification of Protocols and Performance Evaluation of Networks. LAP LAMBERT Acad. Publishing, 2013.
8. *Peng S.S., Zhou M.Ch.* Petri Net Based PLC Stage Programming for Discrete-event Control Design // Systems, Man, and Cybernetics. 2001 IEEE Int. Conf. V. 4. 2001. P. 2706–2710.
9. *Rossmann J., Eilers K.* Translating Robot Programming Language Flow Control into Petri Nets // Emerging Technologies and Factory Automation (ETFA). 2011 IEEE 16th Conf. Digital Object Identifier. P. 1–7.
10. *Palomeras N., Ridao P., Carreras M., et al.* Using Petri Nets to Specify and Execute Missions for Autonomous Underwater Vehicles // Intelligent Robots and Systems. 2009. IROS 2009. IEEE/RSJ Int. Conf. 10–15 Oct. 2009. P. 4439–4444.
11. *Dodd R.B.* Coloured Petri Net Modelling of a Generic Avionics Mission Computer // Air Operations Division: Defence Science and Technology Organisation. Australia. DSTO-TN-0692. 2006. 94 p.

12. *Usher M., Jackson D.* A Petri Net Based Visual Programming Language // Systems, Man, and Cybernetics. 1998. IEEE Int. Conf. V. 1. 1998. P. 107–112.
13. *Iordache M.V., Antsaklis P.J.* Petri Nets and Programming: A Survey // Amer. Control Conf. 2009. ACC '09. 2009. P. 4994–4999.
14. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
15. *Зайцев Д.А.* Универсальная сеть Петри // Кибернетика и системный анализ. 2012. № 4. С. 24–39.
16. *Zaitsev D.A.* Toward the Minimal Universal Petri Net // IEEE Trans. Systems, Man, and Cybernetics: Systems. 2014. V. 44. No. 1. 2014. P. 47–58.
17. *Зайцев Д.А.* Ингибиторная сеть Петри, исполняющая произвольную заданную машину Тьюринга // Системні дослідження та інформаційні технології. 2012. № 2. С. 26–41.
18. *Зайцев Д.А.* Ингибиторная сеть Петри, выполняющая произвольный заданный нормальный алгорифм Маркова // Моделирование и анализ информационных систем. 2011. Т. 18. № 4. С. 80–93.
19. *Слепцов А.И.* Уравнения состояний и эквивалентные преобразования нагруженных сетей Петри (алгебраический подход) // Формальные модели параллельных вычислений. Докл. и сообщ. Всесоюзн. конф. Новосибирск: 1988. С. 151–158.
20. *Зайцев Д.А., Слепцов А.И.* Уравнение состояний и эквивалентные преобразования временных сетей Петри // Кибернетика и системный анализ. 1997. № 5. С. 59–76.
21. *Zaitsev D.A.* Small Polynomial Time Universal Petri Nets. arXiv:1309.7288

Статья представлена к публикации членом редколлегии А.А. Лазаревым.

Поступила в редакцию 16.11.2013