УДК 004.43, 004.27

Вычисления на сетях Слепцова

Зайцев Д.А. (Международный гуманитарный университет, Одесса, Украина)

Выполнен обзор работ, формирующих теоретические основы вычислений на сетях Слепцова и представляющих особенности рисования, компиляции и компоновки программ на языке сетей Слепцова, а также массово параллельные архитектуры вычисляющей памяти для реализации процессоров сетей Слецова. Сеть Петри выполняется экспоненциально медленнее и является частным случаем сети Слепцова. Рассмотрена универсальная сеть Слепцова, содержащая 13 позиций и 26 переходов, представляющая собой прототип процессора сетей Слепцова. Приведены примеры программ на языке сетей Слепцова для эффективного умножения, RSA шифрования/дешифрования, вычисления функции нечёткой логики и решения уравнения Лапласа. Преимуществами вычислений на сетях Слепцова являются наглядный графический язык, сохранение естественного параллелизма предметной области, мелкая грануляция параллельных вычислений, формальные методы верификации параллельных программ, быстрые массово-параллельные архитектуры, реализующие модель вычислений.

Ключевые слова: *сеть Слепцова, сеть Петри, машина Тьюринга, клеточный автомат, параллельные вычисления, универсальные вычислительные модели*

1. Введение

В последнее время многие исследователи предлагают новые модели гипер-вычислений, такие как квантовые вычисления, вычисления на мембранах клеток, на импульсных нейронах и ДНК [10], способные преодолеть препятствие неподдающихся задач. Сети Петри известны более полувека как модель параллельных систем и вычислений [1,4], однако их универсальные расширения выполняются экспоненциально медленнее машины Тьюринга, особенно при реализации арифметических операций. Концепция сети Слепцова, предложенная четверть века назад, недавно обрела своё второе рождение [13] благодаря своей способности к быстрой реализации основных арифметических операций. Запуск перехода в нескольких экземплярах на шаге приводит к универсальным структурам, которые выполняются за полиномиальное время [16]. В вычислениях на сетях Слепцова программа, написанная на языке сетей Слепцова с сохранением естественного параллелизма предметной

области, выполняется на процессоре сетей Слепцова, который реализует параллельное срабатывание переходов в нескольких экземплярах, обеспечивая ультра-быстродействие.

Концепция алгоритма была впервые формализована Аланом Тьюрингом в 1936 году в форме абстрактной машины, которую традиционно называют машиной Тьюринга. Универсальную машину Тьюринга, которая выполняет заданную машину Тьюринга, рассматривают как прототип традиционного компьютера. Кроме машины Тьюринга появились другие универсальные модели вычислений: рекурсивные функции Клини, нормальные алгорифмы Маркова, системы перезаписи тегов Поста, регистровые машины Минского и другие. Разнообразие моделей объясняется специфическими требованиями различных областей применения. Новые модели задействуют возможности массивно-параллельных вычислений, присущие таким простым структурам как элементарные клеточные автоматы, универсальность которых была доказана Мэтью Куком в 2004 году. Кроме того, Турлок Нири и Демием Вудз построили в 2008 году универсальные машины Тьюринга малого размера, которые выполняются в полиномиальное время. Однако способ программирования клеточных автоматов, описанный Мэтью Куком, не задействует массово-параллельные вычисления. Концепция сети Слепцова [13] выправляет основной недостаток сети Петри [4] состоящий в инкрементном характере вычислений, что делает вычисления на сетях Слепцова [13,14] перспективным подходом для достижения ультра-быстродействия параллельных вычислений. В статье [13] представлен обзор работ, которые используют сети Слепцова (сети позиций/переходов с многоканальными переходами или множественной стратегией запуска переходов).

2. Определение сети Слепцова

Сеть Слепцова – это двудольный ориентированный мультиграф, дополненный динамическим процессом [13]. Обозначим сеть Слепцова как $N = (P, T, W, R, \mu_0)$, где P и T это непересекающиеся множества вершин называемых *позиции* и *переходы* соответственно, отображение F описывает *дуги* соединяющие вершины, отношение R представляет приоритеты переходов, и отображение μ_0 задаёт начальное состояние (*маркировку*).

Отображение $W : (P \times T) \to \mathrm{N} \bigcup \{-1\}, (T \times P) \to \mathrm{N}$ задаёт дуги, их тип и кратность; нулевое значение соответствует отсутствию дуги, положительное значение – *регулярной дуге* с указанной кратностью, а минус один – *ингибиторной дуге* которая проверяет маркировку позиции на ноль. N обозначает множество неотрицательных целых чисел. Чтобы избежать

вложенных индексов, обозначим $w^-_{j,i} = w(p_j, t_i)$ и $w^-_{i,j} = w(t_i, p_j)$. Отображение $\mu : P \to \mathrm{N}$ представляет маркировку позиций.

В графической форме позиции изображают в виде кругов, а переходы – в виде прямоугольников (квадратов). Ингибиторную дугу представляют небольшим полым кругом на её конце, а маленький заполненный круг обозначает аббревиатуру цикла. Кратность регулярных дуг большая, чем единица, подписывается над дугой, и маркировка позиции большая нуля записывается внутри позиции. Примеры сетей Слепцова, реализующих основные арифметические и логические операции, приведены на Рис. 6, который будет обсуждаться в последующих разделах.

Чтобы оценить *кратность возбуждения* на каждой входящей дуге позиции, введём следующую вспомогательную операцию

$$x \succ y = \begin{cases} x / y, if \ \ y > 0 \\ 0, if y = -1, \ x > 0, \\ \infty, if y = -1, \ x = 0. \end{cases}$$

Во избежание аномалий с бесконечным числом экземпляров перехода, запретим в дальнейшем использование переходов, не содержащих входящих регулярных дуг.

Поведение (динамика) сети Слепцова может быть описана соответствующим уравнением состояний аналогично [6]. Настоящая работа рассматривает поведение сети как результат применения следующего *правила запуска перехода*:

– количество экземпляров перехода t_i возбужденных на текущем шаге равняется

$$v_i = v(t_i) = \min_j (\mu_j \succ w^-_{j,i}), 1 \le j \le m, \ w^-_{j,i} \ne 0$$

– когда переход $t_i, v_i > 0$ срабатывает, при $u_i \le v_i$, он
  - ✓ извлекает $u_i \cdot w^-_{j,i}$ фишек из каждой своей входной позиции p_j для регулярных дуг $w^-_{j,i} > 0$;
  - ✓ добавляет $u_i \cdot w^+_{i,k}$ фишек в каждую свою выходную позицию p_k, $w^+_{i,k} > 0$;

– сеть останавливается, если отсутствуют возбужденные переходы.

Когда переход, имеющий единственную регулярную входящую дугу кратности a из позиции p и единственную исходящую дугу кратности b в позицию p', срабатывает, при $u_i = v_i$, он осуществляет следующие вычисления: $\mu(p) = \mu(p) \bmod a$; $\mu(p') = \mu(p') + b \cdot (\mu(p) \ div \ a)$. А именно, он реализует деление на a с остатком и умножение на b. Выбирая либо a, либо b равными единице, получаем чистое умножение либо чистое деление соответственно.

В сети Петри только один переход срабатывает на шаге, в то время как в сети Салвицки [8] (или синхронной сети в соответствии с [4]) на шаге срабатывает максимальное множество возбужденных переходов. Однако в обеих сетях (Петри и Салвицки) лишь один экземпляр каждого перехода срабатывает на шаге.

Переход может восприниматься как некоторое виртуальное событие (действие). Количество реально запущенных действий зависит лишь от количества доступных ресурсов, представленных входными позициями перехода. Зачем же необходимо ограничивать количество экземпляров перехода до единицы, когда имеющиеся в наличии ресурсы позволяют запустить действия одновременно? Более того, классический последовательный порядок запуска переходов может быть получен как частный случай путем присоединения к каждому переходу посредством цикла отдельной позиции, содержащей одну фишку.

Известны разнообразные расширения сетей Петри, такие как приоритетные, ингибиторные, временные, нагруженные (раскрашенные), иерархические и вложенные сети Петри [4,5,9,11]. Иногда они лишь усложняют восприятие основной модели. Наша цель состоит в получении гипер-производительности за счёт минимальной модификации, которая состоит во множественной стратегии запуска перехода. Что касается других моделей близких к сетям Петри, следует отметить системы перезаписи множеств, а также системы сложения и замещения векторов. Кроме того, для изучения протоколов используются последовательные взаимодействующие процессы Хоара, близкие к сетям Петри и позволяющие взаимные преобразования [11]. Существенным преимуществом сетей Слепцова (и сетей Петри) является графическое представление структуры и поведения систем.

3. Универсальная сеть Слепцова как процессор

Для программирования на языке сетей Слепцова (Петри, Салвицки) используем концепцию выделенных контактных позиций. Перед запуском сети загружаем исходные данные в контактные позиции, а когда сеть остановиться, извлекаем выходные данные из контактных позиций. Иногда более детальная специализация, различающая отдельные подмножества входных и выходных позиций, является удобной. В этом случае используем декомпозицию сети на кланы (функциональные подсети) [11]. Заметим, что данные различных типов следует закодировать целыми неотрицательными числами для их последующей обработки сетью.

Полнота по Тьюрингу расширенных сетей Петри подразумевает существование универсальной сети, которая выполняет произвольную заданную сеть. В последнее время была построена серия универсальных сетей малого размера [10,12,16], каждая из которых может рассматриваться как прототип процессора в вычислениях на сетях Слепцова [13]. Программа такого компьютера, представленная в форме (ингибиторной, приоритетной) сети Слепцова и закодированная в целых числах, подаётся в качестве исходных данных на процессор, который представляет собой аппаратную реализацию универсальной сети. Рис. 1 иллюстрирует описанный подход.

Рис. 1. Универсальная сеть как процессор

Главным препятствием на пути широкой реализации парадигмы вычислений на сетях Петри [7] является тот факт, что все известные универсальные сети малого размера выполняются в экспоненциальное время (в зависимости от числа шагов заданной сети). В статье [16] построена универсальная сеть Слепцова малого размера, которая выполняется в полиномиальное время; сеть содержит 13 позиций и 26 переходов. Техника композиции универсальной сети Слепцова [16] использует обратный поток управления и подсети умножения и деления на константу, которые в последующем являются удобным средством разработки технологии программирования на сетях Слепцова [14]. Существенная разница в вычислительной сложности между универсальными сетями Слепцова и Петри достаточно легко объяснима: во время обработки кода ленты машины Тьюринга, который представляет собой экспоненту от ширины рабочей зоны, универсальная сеть Петри извлекает одну фишку на шаге, в то время как универсальная сеть Слепцова извлекает все фишки. В результате мы приходим к быстрым арифметическим операциям, а именно умножению и делению, которые являются основными для кодирования/декодирования заданной сети.

Известным примером графической технологии программирования является p-технология [2], однако её средства не содержат формальные модели параллельных вычислений, что затрудняет верификацию параллельных программ.

В статье [7] предложена концепция вычисляющей памяти для массово параллельной реализации аппаратных процессоров сетей Слепцова, которая обеспечит ультра-производительность при выполнении программ написанных на языке сетей Слепцова. Отдельно от проблемы эффективной аппаратной реализации стоят вопросы разработки технологии программирования на сетях Слепцова сохраняющей естественный параллелизм предметной области. Несомненным преимуществом таких программ является мелкая грануляция параллельных процессов, а также возможность применения формальных методов верификации параллельных программ [1,4,11] в процессе управляемой моделью разработки.

4. Универсальная сеть Слепцова выполняющаяся за полиномиальное время

Представленная в статье [16] универсальная сеть Слепцова получена путём моделирования слабо-универсальной машины Тьюринга с 2 состояниями и 4 символами ленты, построенной Нири и Вудз и обозначенной СУМТ(2,4). Так как поведение машины Тьюринга детерминировано, используем детерминированные сети Слепцова, в которых все переходы занумерованы и на шаге срабатывает переход с наименьшим индексом в максимальном числе экземпляров. Для визуального отображения порядка (приоритета) позиций используем дуги, соединяющие переходы; если существует дуга из перехода t в переход t', это означает, что индекс перехода t должен быть меньше индекса перехода t' (меньший индекс обозначает более высокий приоритет). Особенностью программирования на сетях Слепцова является использование обратного потока управления, представленного движением нулевой маркировки; таким образом, начальная маркировка позиций потока управления равна 1. Для проверки нулевой маркировки используется ингибиторная дуга, которая не ограничивает кратность срабатывания перехода.

Исходной информацией для моделирования является функция переходов СУМТ(2,4) и кодирование состояний и символов ленты [16], заданные Табл. 1. Изначально на ленте СУМТ(2,4) записано бесконечное повторение пустых слов: $b_l = 0001$ влево и $b_r = 010001$ вправо от рабочей зоны. В соответствии с функцией кодирования ленты

$s(x_{k-1}x_{k-2}...x_0) = \sum_{i=0} s(x_i) \cdot r^i$, коды левого и правого пустых слов равняются: $s(b_l) = 167$ и $s(b_r) = 13596$.

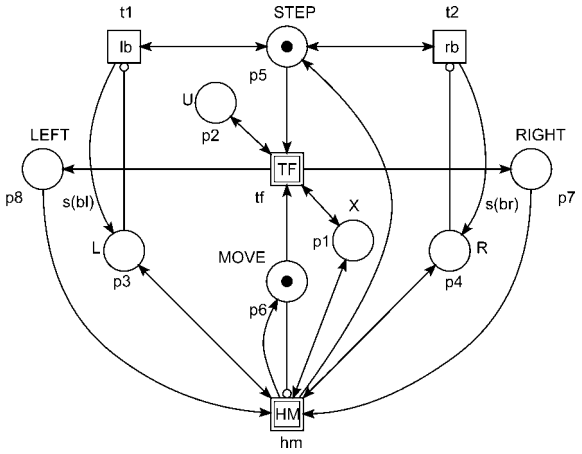
Построенная универсальная сеть Слепцова обозначена как УСС(13,26) в соответствии с количеством её позиций и переходов соответственно. Общая схема УСС(13,26) представлена на Рис. 2 а). Подсети изображены как квадраты с двойной линией границ. Для некоторых вершин кроме их номеров указаны также мнемонические имена. Использованные подсети TF и HM представлены на Рис. 2 б), в) соответственно. Вершины с одинаковыми именами (номерами) логически обозначают одну и ту же вершину и при окончательной компоновке объединяются по всем компонентам сети. Изображение полученной в результате сети выглядит довольно запутанно [16] и для её формального представления рекомендуется использовать параметрические выражения, использованные для описания бесконечных сетей [15], в частности, универсальных сетей, полученных в результате моделирования клеточных автоматов.

Таблица 1. Слабо универсальная машина Тьюринга СУТМ (2,4) и ее кодирование

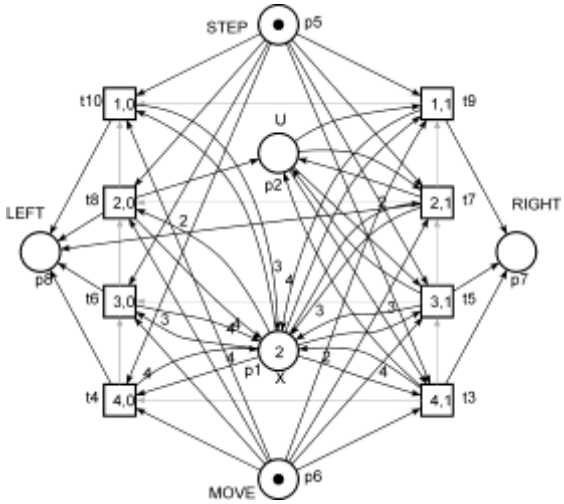
$\Sigma \setminus \Omega$		u_1	u_2
	$s(\Sigma) \setminus s(\Omega)$	0	1
0	1	3,left,0	4,right,0
1	2	4,left,1	3,left,1
∅	3	4,left,0	1,right,1
1	4	4,left,0	2,right,1

Позиция U содержит код состояния $s(u)$, позиция X содержит код текущего символа ленты $s(x)$, а позиции L и R содержат коды левой и правой частей рабочей зоны ленты соответственно по отношению к текущей ячейке. При моделировании текущего шага машины, позиция STEP запускает подсеть TF, которая моделирует функцию переходов СУМТ(2,4). Подсеть TF получает коды нового состояния головки $s(u')$ и нового символа текущей ячейки ленты $s(x')$ в позициях U и X соответственно. Подсеть TF также помещает фишку в позицию RIGHT или позицию LEFT для указания правого либо левого движения головки соответственно. Фишка извлекается из позиции MOVE при завершении работы подсети TF и запускает подсеть HM, которая моделирует требуемое движение управляющей головки вдоль ленты. В конце моделирования текущего шага, фишка возвращается в

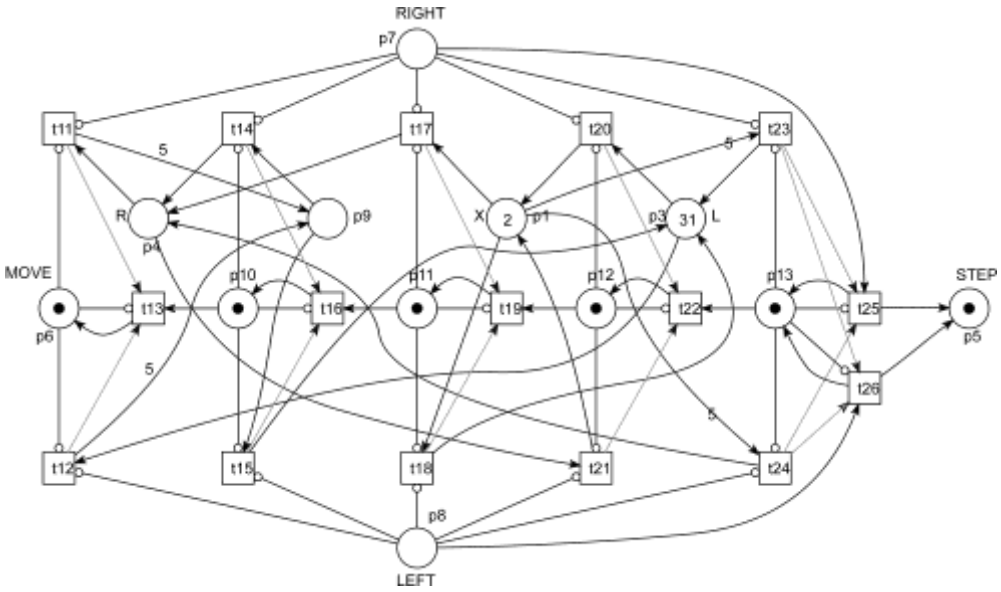
позицию STEP, что позволяет начать моделировать следующий шаг. Заметим, что позиции LEFT и RIGHT очищаются заключительными переходами подсети HM.



а) общая схема;



б) подсеть функции переходов TF;



в) подсеть движений управляющей головки HM.

Рис. 2. Покомпонентное представление УСС(13,26)

Лента машины представлена позициями L, X, и R содержащими коды левой части рабочей зоны, символа текущей ячейки и правой части рабочей зоны соответственно. Движение головки влево моделируется как (a) $R'' = 5 \cdot R' + X'$, (b) $X'' = L' \% 5$, (c) $L'' = L' / 5$ а движение головки вправо, как (a) $L'' = 5 \cdot L' + X'$, (b) $X'' = R' \% 5$, (c) $R'' = R' / 5$; фактически выполняются одинаковые последовательности вычислений, в которых L и R меняются местами. HM

представляет собой объединение трёх частей, индуцированных следующими последовательностями переходов: переходы $t_{11}, t_{14}, t_{17}, t_{20}, t_{23}$ моделируют движение влево; переходы $t_{12}, t_{15}, t_{18}, t_{21}, t_{24}$ моделируют движение вправо; переходы $t_{13}, t_{16}, t_{19}, t_{22}, t_{25}, t_{26}$ представляют инверсный поток управления. Например, при движении влево: переход t_{11} реализует умножение на константу 5: $\mu(p_9) := 5 \cdot R, R := 0$; переход t_{14} перемещает результат обратно из позиции p_9 в позицию R: $R := 5 \cdot R, \mu(p_9) := 0$; переход t_{17} добавляет X к R с очисткой X: $R := 5 \cdot R + X, X := 0$; переход t_{20} подготавливает операнд для деления, перемещая L в X: $X := L, L := 0$; переход t_{23} реализует деление с остатком на константу 5: $X := L\%5, L := L/5$.

В статье [16] доказано, что УСС(13,26) моделирует СУМТ(2,4) за время $O(12 \cdot k) \approx O(k)$. Емкостная сложность оценивается как $\log_2 5^k = k \cdot \log_2 5 \approx O(k)$, где k – это число шагов СУМТ(2,4). Принимая во внимание полиномиальную сложность выполнения заданной машины Тьюринга на УСС(13,26) и полиномиальную сложность выполнения сети Слепцова на машине Тьюринга, получаем общую полиномиальную сложность представленной универсальной сети Слепцова.

Заметим, что УСС(13,26) моделирует элементарный клеточный автомат номер 110. В статье [15] построена серия сетей, непосредственно моделирующих клеточные автоматы, для получения универсальных конструкций с массовым параллелизмом выполнения.

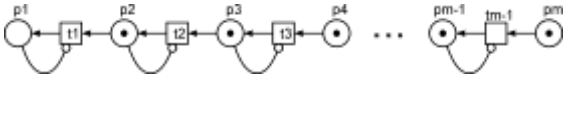
5. Принципы программирования на сетях Слепцова

Программирование на сетях Слепцова [14] представляет собой композицию данных и потоков управления, дополненную подстановкой перехода для описания иерархической структуры программы (сети). Предполагаем, что для комбинирования потока управления с подсетью, которая подставляется вместо перехода, используется специальная пара позиций: *Start* – чтобы запустить подсеть и *Finish* – чтобы индицировать ее завершение.

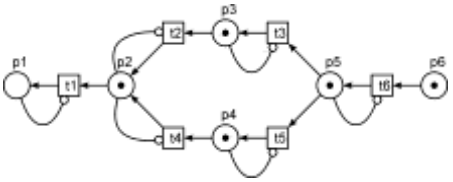
Потоки управления представлены в инверсной форме «с движущимся нулём» потому что нулевая маркировка легко проверяется ингибиторной дугой и не ограничивает количество возбужденных экземпляров перехода. Таким образом, изначально все позиции потока управления содержат фишку.

Для моделирования стандартных потоков управления классических языков программирования предложены шаблоны, представленные на Рис. 3 для последовательности а), ветвления б), цикла в), и параллельного выполнения г). Кроме того, произвольная подсеть

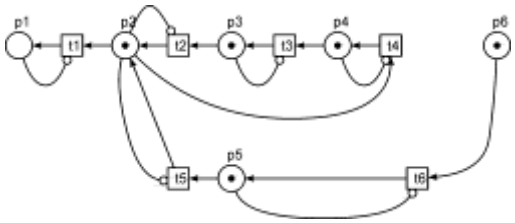
с маркировками, принадлежащими $\{0,1\}^m$, снабженная парой позиций запуска/завершения, может рассматриваться как поток управления.



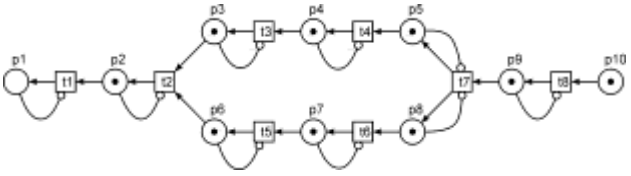
а) последовательность;



б) ветвление;



в) цикл (“while”);



г) параллельное выполнение (split t_2 and join t_7).

Рис. 3. Стандартные шаблоны потока управления

Каждый из переходов на Рис. 3 может быть замещён (подставлен) подсетью, использующей входную позицию для запуска и выходную позицию для завершения. Кроме того, дополнительные инцидентные переходам позиции могут быть добавлены для представления требуемых входные и выходных переменных. В шаблонах с альтернативными переходами (Рис. 3 б, в) предполагается либо внешнее управление посредством переменных, содержащих условия в случае детерминированного выбора, либо недетерминированный выбор.

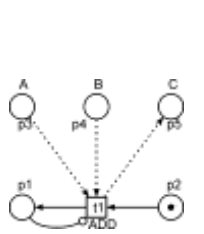
Для вычисления выражений подход управления потоками данных может быть более эффективным, и соответствующая подсеть запускается и завершается с помощью внешнего потока управления. Это вопрос грануляции вычислений: либо использовать строгую композицию подсетей, запускаемых потоком управления, либо рисовать заново сильно взаимосвязанное переплетение потоков данных и управления каждый из которых может быть чётко неразличим.

Программа, нарисованная на языке сетей Слепцова, получается путём композиции потоков управления и данных с использованием модульного подхода. Подстановка перехода модулем определяется указанием имени модуля (подсети) и соединения (отображения) его входных и выходных позиций с позициями исходной сети которые представляют переменные и (или) поток управления.

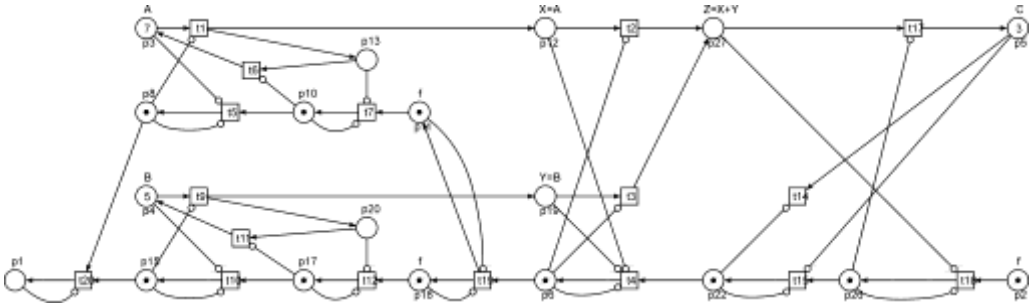
Также предполагается, что перед запуском модуля все его входные данные скопированы в его входные позиции, а после завершения работы модуля или пред его следующим запуском

все выходные данные перемещены из его выходных позиций. Для этого предложено использовать пунктирные дуги, введенные в [6] для обозначения краткого и удобного способа работы с данными.

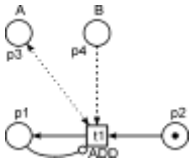
Штриховая входящая дуга модуля обозначает подсеть COPY используемую для копирования значения входной переменной в соответствующую входную позицию модуля перед запуском его работы посредством позиции *Start*. А штриховая исходящая дуга модуля обозначает последовательность подсетей CLEAN, MOVE для замещения значения переменной результатом, полученным в соответствующей выходной позиции модуля. В случае если модуль имеет несколько входных (выходных) позиций, подсети COPY (CLEAN, MOVE) вставляются с использованием параллельного шаблона выполнения. В примере, представленном на Рис. 4 а) и б), подсеть а), содержащая штриховые дуги расширяется в соответствующую низкоуровневую сеть б).



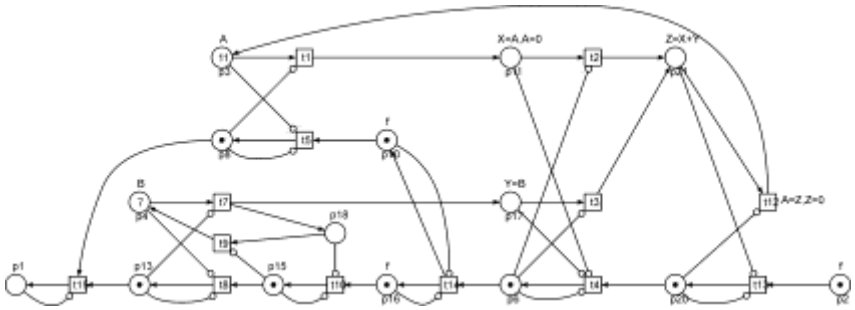
а) c=ADD(a,b);



б) расширение of a);



в) a=ADD(a,b);



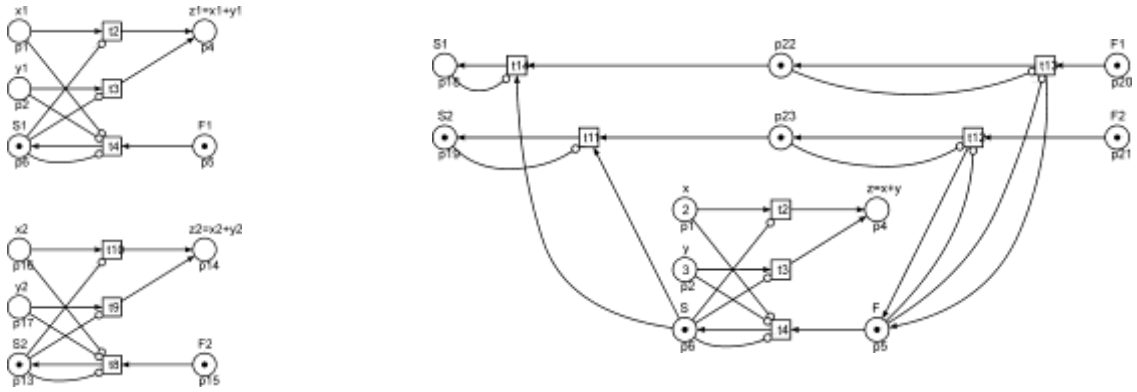
г) расширение of в).

Рис. 4. Примеры расширения пунктирных дуг

Введенные для сокращения размера вспомогательных подсетей пунктирные дуги обозначают MOVE как для входных, так и для выходных переменных; подсеть MOVE является сокращенной и обеспечивает очистку значения своей входной переменной. Таким образом, значение входной переменной не сохраняется и значение выходной переменной добавляется. Когда переменная является как входной, так и выходной, используется пунктирная двунаправленная дуга, которая расширяется как MOVE перед запуском модуля и

MOVE после завершения модуля. Соответствующий пример представлен на Рис. 4 в) и г), где подсеть в) содержащая пунктирную дугу расширяется в низкоуровневую сеть г).

Таким образом, модуль сети может рассматриваться как процедура языка программирования, чьи контактные позиции соответствуют формальным параметрам. Копирование переменных штриховыми и пунктирными дугами соответствует подстановке фактических входных параметров и извлечению фактических выходных параметров. Остается открытым вопрос о способе реализации обращения к модулю, которое может быть выполнено либо в стиле вставки полной копии подсети, либо в стиле вызов-возврат с единственной копией модуля и переключением потоков управления при вызове-возврате из различных мест. Разница между двумя указанными подходами проиллюстрирована Рис. 5 для двух обращений к модулю ADD (без рассмотрения передачи используемых данных). В то время как стиль вызова-возврата является более компактным из-за наличия единственной копии модуля, стиль вставки является более привлекательным с точки зрения параллельного исполнения [14].



а) вставка б) вызов-возврат

Рис. 5. Примеры реализации подстановки перехода для z1=ADD(x1,y2); z2=ADD(x2,y2)

Программа, представляющая собой композицию модулей, в конечном счёте компилируется в простую («плоскую») ингибиторную приоритетную сеть Слепцова, которая загружается на компьютер сетей Слепцова. Наилучшая производительность достигается в случае, когда все переходы запускаются независимо на основе их локальных условий возбуждения; для разрешения конфликтов предложено использовать арбитра, который блокирует инцидентные позиции срабатывающего перехода. Описанный способ исполнения требует рисования программ инвариантных к порядку запуска переходов.

6. Эффективная реализация арифметических и логических операций

Для формального описания операции подстановки перехода с учётом внешнего потока управления и входных/выходных переменных введена концепция *модуля* [6,13,14]. Модуль представляет собой подсеть с контактными (входными и выходными) позициями, которые представляют собой единственный способ взаимодействия с окружающим миром для замкнутого модуля либо комбинируется с использованием глобальных переменных для открытого модуля. Работа модуля контролируется парой выделенных позиций: позиция *Start* (*s*) запускает модуль и позиция *Finish* (*f*) индицирует завершение работы модуля. Извлечение фишки из первой позиции потока управления *Start* запускает движение нулевой маркировки до тех пор пока ноль ни прибудет в последнюю позицию потока управления *Finish.*

На графы потоков управления наложены следующие ограничения. Предположим, что все действия модуля контролируются его потоком управления, и ингибиторные дуги используются для запуска переходов. Таким образом, когда все позиции потока управления содержат фишку, переходы модуля отключены. В результате, перед извлечением фишки из позиции *Start*, а также после прибытия нулевой маркировки в позицию *Finish* все переходы модуля неактивны.

Для обеспечения реентерабильности (повторного входа) модуля предполагается, что когда модуль запускается, все его позиции данных, за исключением входных позиций, имеют нулевую маркировку, а когда модуль завершается, все его позиции данных, за исключением выходных позиций, имеют нулевую маркировку. На Рис. 6 представлены сети, реализующие основные операции: копирование, логические и арифметические. По сравнению со статьёй [6] подсети сокращены на одну позицию и один переход в предположении, что не разрешено использовать входную позицию *Start* для управления переходами внутри подсети, но позиция *Finish* не используется для управления внутри подсети.

Используя метод анализа и классификации всех разрешенных последовательностей переходов, использованную в [13], доказано, что каждая из подсетей, представленных на Рис. 6 реализует соответствующую операцию.

Для (ингибиторных) сетей Петри операции умножения и деления являются наиболее сложными с точки зрения времени вычисления; соответствующие подсети были изучены в статье [13]. Именно их сложность обуславливает экспоненциальную медлительность вычислений на сетях Петри. Представленные в [16] компоненты универсальной сети Слепцова эффективно реализуют умножение и деление на константу (равную 5). Рассмотрим

умножение и деление для произвольных пар заданных целых неотрицательных чисел, используя умножение и деление на константу (равную 2).

Мы выбираем простые школьные алгоритмы умножения и деления «в столбик». Лучшие известные алгоритмы умножения и деления могут также быть закодированы сетью Слепцова. В алгоритме умножения для нахождения текущей цифры множителя используется остаток от деления на два: $d = y\,\%\,2$; затем множитель перевычисляется как $y/ = 2$ для работы со следующей цифрой на следующем проходе основного цикла. Множимое умножается на два: $x* = 2$, что представляет собой его сдвиг влево. Когда текущая цифра *d* множителя равна 1, сдвинутое множимое добавляется к результату. Алгоритм может быть оптимизирован, чтобы избежать перевычисления, когда новое значение *y* равно нулю, но это приводит к усложнению сети. Алгоритм закодирован сетью Слепцова и представлен на Рис. 7.

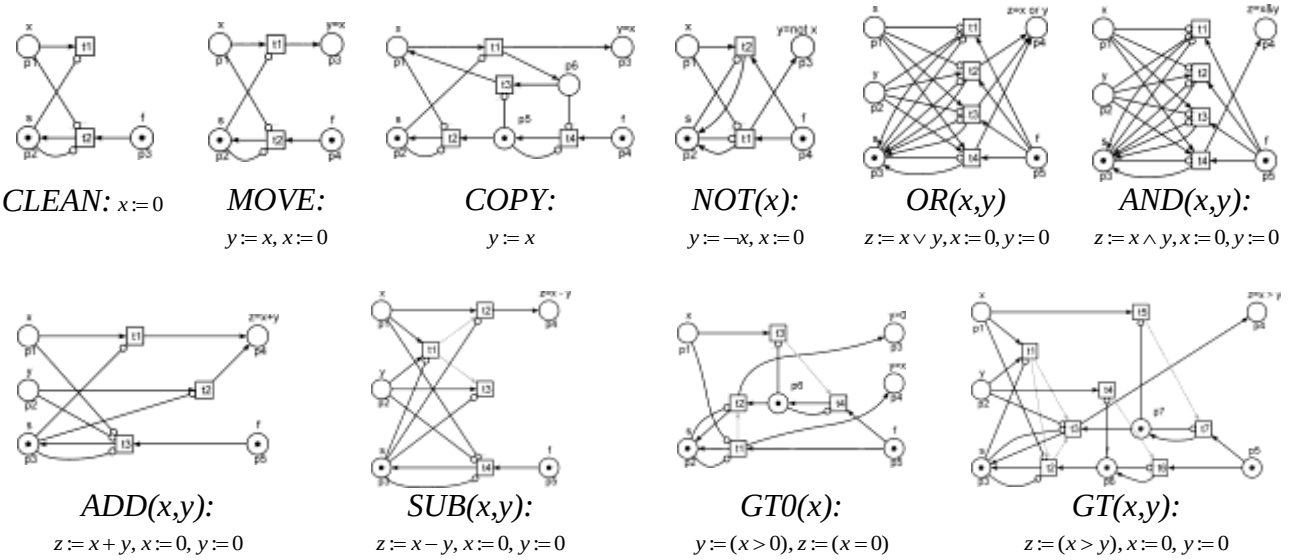


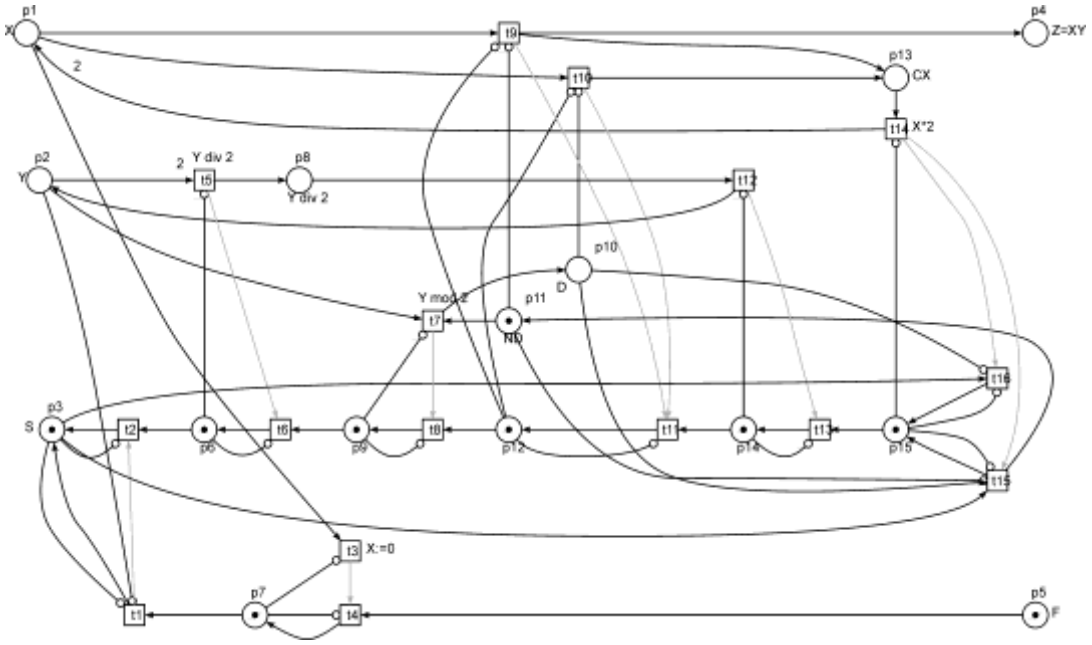
Рис. 6. Модули, реализующие основные операции: копирование, логические и арифметические

Для сетей Слепцова изображенных на Рис. 6, 7 доказано [13] что:

– CLEAN, MOVE, COPY, NOT, OR, AND, ADD, SUB, GT0, GT реализуют соответствующие операции с временной и емкостной сложностью равной константе;

– MUL реализует умножение неотрицательных целых чисел *x* и *y* с временной сложностью $O(11 \cdot \log_2 y + 3)$ и постоянной емкостной сложностью равной 15 (по линейной шкале);

– DIV [13] реализует деление неотрицательных целых чисел x и y с временной сложностью $39\cdot(\log_2 x-\log_2 y)+19$ и постоянной емкостной сложностью равной 48.



MUL(x,y): $z := x \cdot y,\ x := 0,\ y := 0$ *;*

Рис. 7. Модуль умножения за полиномиальное время

Например, чтобы вычислить $3\cdot 2=6$ множитель 3 загружается в позицию X (p_1) и число 2 загружается в позицию Y (p_2). Для запуска вычислений фишка извлекается из позиции S (p_3). Тогда единственная разрешённая последовательность срабатывания переходов $t_1t_5t_6t_8t_{10}^3t_{11}t_{12}t_{13}t_{14}^3t_{16}t_1t_6t_7t_8t_9^6t_{11}t_{13}t_{14}^6t_{15}t_2t_3^{12}t_4$ приводит к получению результата 6 в позиции Z (p_4) что индицируется изъятием фишки из позиции F (p_5).

Заметим, что модуль сети Слепцова для умножения, показанный на Рис. 7, умножает заданные натуральные числа X и Y, используя для этих целей умножение и деление на константу 2 выполняемые одним переходом сети Слепцова; умножение и деление на константу 2 обычно эффективно реализуют как сдвиг двоичного кода влево и вправо соответственно.

В реальных реализациях сети Слепцова (также как сети Петри), программных или аппаратных, мы должны рассматривать сложность указанных процедур. Предположение последовательного вычислительного устройства [12,13], просматривающего переходы и позиции в цикле даёт множитель $O(|P|\cdot|T|\cdot\log_2 l)$ для сетей Петри, где l – это максимальное число шагов. Для сетей Слепцова дополнительная сложность умножения и деления на

константу равную степени 2 также оценивается как $O(\log_2 l)$; для произвольной константы сложность сублинейна [13]. Если мы предполагаем устройство вычисляющей памяти, которое независимо реализует каждый переход [7], мы избавляемся от множителя $|P|\cdot|T|$ и получает ультра-параллельный процессор сетей Слепцова со сложностью шага $O(\log_2 l)$.

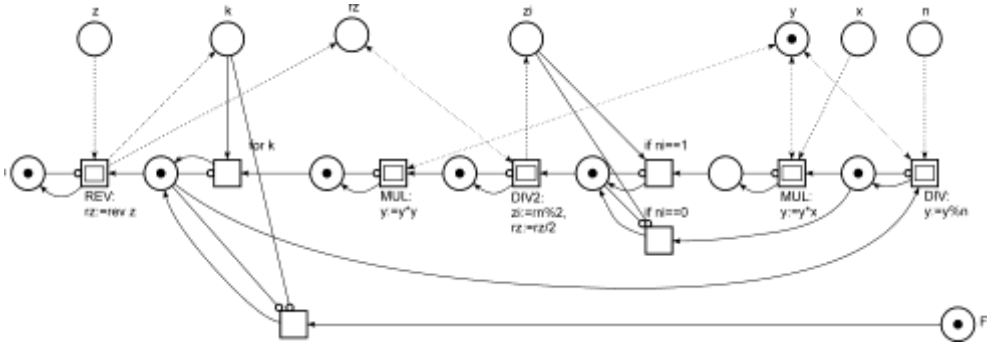
Рассмотренные оценки слишком скрупулёзны для обычного сравнения арифметических алгоритмов. Предполагая, что x и y содержат около n двоичных цифр $x \approx y < 2^n$ и используя логарифмическую шкалу с множителем $\log_2(x \cdot y) \approx 2 \cdot n$, получаем сложность умножения $O(2^{2 \cdot n})$ для сетей Петри и $O(n^2)$ для сетей Слепцова. Для операции деления предположим, что x содержит $2 \cdot n$ двоичных цифр и y содержит n двоичных цифр. Тогда мы получаем аналогичную сложность деления $O(2^{2 \cdot n})$ для сетей Петри и $O(n^2)$ для сетей Слепцова. Таким образом, сети Слепцова выполняются экспоненциально быстрее в сравнении с сетями Петри. Для операций умножения и деления сети Слепцова выполняются за полиномиальное время, в то время как сети Петри требуют экспоненциальное время.

Заметим, что некоторые сети Слепцова могут интерпретироваться как сети Петри, дающие тот же самый результат, но с определенным замедлением вычислений. Исследование указанного типа эквивалентности является направлением дальнейших работ.

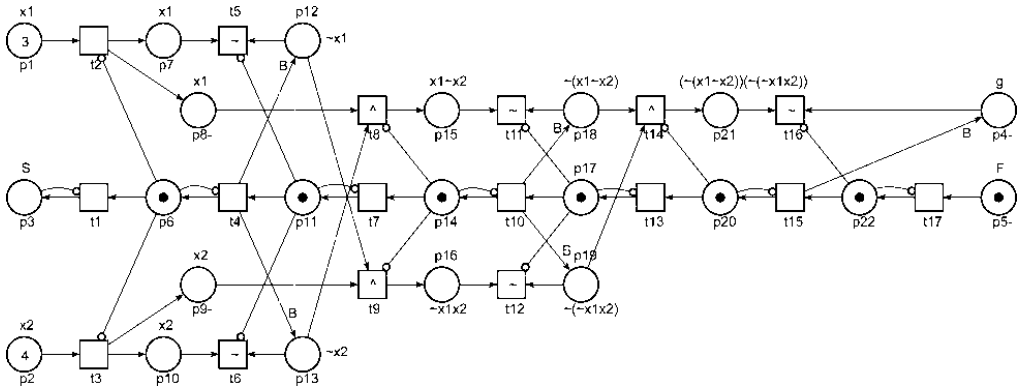
7. Перспективы практической реализации

На основе описанных ранее универсальной сети и сетей, выполняющих арифметические операции, сделан основной вывод, что сети Слепцова выполняются экспоненциально быстрее сетей Петри [13], что позволяет рекомендовать их в качестве модели параллельных вычислений для последующей практической реализации.

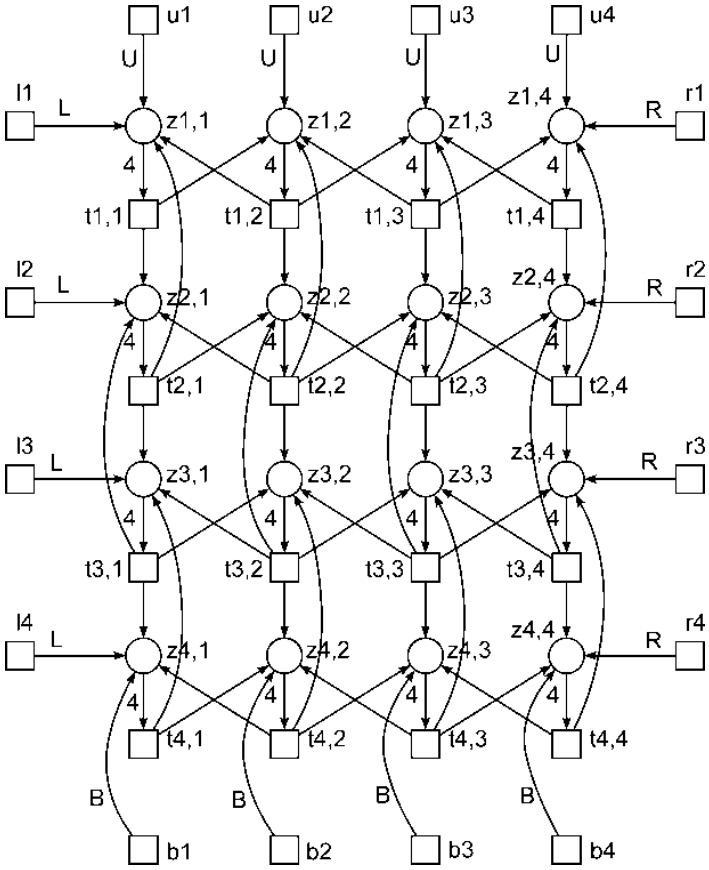
Вычисления на сетях Слепцова обретают все новые области применения, представленные в работах [13,14]. Мы завершаем настоящую работу тремя примерами программ на языке сетей Слепцова для достаточно разнообразных областей приложения, таких как шифрование данных с открытым ключом, мягкие вычисления на основе нечёткой логики и решение уравнений математической физики (Рис. 8 а–в). Первая из сетей использует изученные подсети для арифметических операций и подсеть REV для инверсии побитового представления z в то время как вторая и третья сети реализуют требуемые логические и арифметические операции непосредственно соответствующими переходами сети Слепцова.



а) RSA шифрование/дешифрование $y = x^z \bmod n$



б) вычисление функции нечёткой логики $\varphi = x_1\overline{x}_2 \vee \overline{x}_1 x_2 = \overline{(\overline{(x_1\overline{x}_2)} \wedge \overline{(\overline{x}_1 x_2)})}$;



в) решение уравнения Лапласа $\frac{\delta^2\varphi}{\delta x^2} + \frac{\delta^2\varphi}{\delta y^2} = 0$.

Рис. 8. Примеры программ на языка сетей Слепцова

В первую очередь мы рекомендуем использовать вычисления на сетях Слепцова для тех областей применения, в которых параллельный стиль программирования может принести значительные ускорения вычислений.

Эффективная практическая реализация вычислений на сетях Слепцова требует разработки соответствующих специализированных систем автоматизации программирования и аппаратной реализации процессоров сетей Слепцова. Кроме того, необходимо дальнейшее развитие теоретических методов доказательства корректности программ на языке сетей Слепцова и разработка универсальных сетей, которые используют массовый параллелизм.

Преимуществами вычислений на сетях Слепцова являются наглядный графический язык, сохранение естественного параллелизма предметной области, мелкая грануляция параллельных вычислений, формальные методы верификации параллельных программ, быстрые массово-параллельные архитектуры, реализующие модель вычислений.

Список литературы

1. Ачасова С.М., Бандман О.Л. Корректность параллельных вычислительных процессов. Н.: Наука, 1990, 254 с
2. Вельбицкий И.В. Технология программирования. Киев: Техіка, 1984. 279 с.
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб: БХВ-Петербург, 2002, 608 с.
4. Котов В.Е. Сети Петри. М: Наука, 1984, 160 с.
5. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой, Науч. мир, 2004, 207 с.
6. Зайцев Д.А. Универсальная сеть Петри. Кибернетика и системный анализ, № 4, 2012, 24-39.
7. Зайцев Д.А. Парадигма вычислений на сетях Петри. Автоматика и телемеханика, № 8, 2014, 19–36.
8. Burkhard H.-D. Ordered Firing in Petri Nets, Jornal of Information Processing and Cybernetics, no. 2, 1981, 71-86.
9. Jensen K., Kristensen L.M. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, Berlin, 2009.
10. Neary T., Cook M. (ed.) Proceedings Machines, Computations and Universality (MCU 2013), Zurich, Switzerland, Electronic Proceedings in Theoretical Computer Science 128, 2013.
11. Zaitsev D.A. Clans of Petri Nets: Verification of protocols and performance evaluation of networks, LAP LAMBERT Academic Publishing, 2013, 292 p.
12. Zaitsev D.A. Toward the Minimal Universal Petri Net. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 44 (1), 2014, 47–58.

13. Zaitsev D.A. Sleptsov Nets Run Fast, IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016, Vol. 46(5), 682–693.
14. Zaitsev D.A., Jürjens J. Programming in the Sleptsov Net Language for Systems Control, Advances in Mechanical Engineering, 2016, Vol. 8(4), 1–11.
15. Zaitsev D.A. Simulating Cellular Automata by Infinite Petri Nets, Journal of Cellular Automata, 2017.
16. Zaitsev D.A. Universal Sleptsov Net, International Journal of Computer Mathematics, 2017.