УДК 006.72 **DOI** 10.15622/sp.48.10

А.А. Воевода, Д.О. Романников

АСИНХРОННЫЙ АЛГОРИТМ СОРТИРОВКИ МАССИВА ЧИСЕЛ С ИСПОЛЬЗОВАНИЕМ ИНГИБИТОРНЫХ СЕТЕЙ ПЕТРИ

Воевода А.А., Романников Д.О. **Асинхронный алгоритм сортировки массива чисел с использованием ингибиторных сетей Петри.**

Аннотация. В настоящее время задачи ускорения вычислений и/или их оптимизация является достаточно актуальной задачей. Среди направлений решения вышеприведенной задачи в статье рассматривается применение подхода распараллеливания и асинхронизации алгоритма сортировки. Предлагается метод сортировки, основанный на принципе разбиения всего массива на множество независимых пар чисел и их параллельное и асинхронное сравнение, что отличает предлагаемый алгоритм от традиционных алгоритмов сортировки (таких как быстрая сортировка, сортировка слиянием, вставками и другие). Алгоритм реализован с использованием сетей Петри как наиболее подходящего инструмента для описания асинхронных систем, а также приведен пример его работы. В статье выполнена оценка быстродействия алгоритма для наилучшего и наихудших случаев. В наилучшем случае алгоритм выполняется за 2 или 3 условных такта в зависимости от разбиения массива на пары соседних элементов. В наихудшем случае – за n или за 3n/2, где n – число элементов. Принципы распараллеливания и асинхронизации, использованные при построении алгоритма, также могут быть применены для других алгоритмов.

Ключевые слова: алгоритмы сортировки, пузырьковая сортировка, сети Петри, асинхронность, параллельные вычисления.

1. Введение. В настоящее время решение большинства современных задач требует больших вычислительных ресурсов. При этом развитие вычислительных ресурсов выполняется по многим направлениям: аппаратный уровень, где, например, увеличивают число ядер процессора; программное обеспечение, где применяют различные виды распараллеливания; и архитектурный уровень, где вычислительные системы проектируются из расчета распределения вычислительной нагрузки на многие независимые вычислительные ресурсы [1-3]. Стоит отметить, что для всех вышеприведенных направлений развития чаще всего применяются синхронные схемы их работы.

Одной из частных задач, которая решается при оптимизации использования вычислительных ресурсов и их ускорения, является задача сортировки. Для решения данной задачи разработано множество алгоритмов (сортировка пузырьком (*bubble sorting*), сортировка слиянием (*merge sorting*), быстрая сортировка (*quick sorting*), чет-нечет сортировка (*odd-even sorting*) и др.) [4-5]. В частности, в [5] приводится многопоточная реализация чет-нечет сортировки (*odd-even sorting*), заключающаяся в разбиении и параллельном сравнении и перестанов-

ке элементов массива на пары, начинающиеся с четных и не четных элементов. Битонная сортировка (*bitonic sorting*) [5, 6], основанная на последовательном приведении массива к парам битонных последовательностей и дальнейшем их слиянии, является популярным алгоритмом часто применяемая в GPU (*Graphics Processing Unit*). В [5] ранг сортировка (*rank sorting*), в основе которой лежит подсчет количества элементов с меньшим и большим значениями и последующая их расстановка по пред рассчитанным позициям. В [6] предлагается *AA-sort* алгоритм, основанный на сортировке расческой (*comb sorting*), разбиении всего массива элементов на множество подгрупп и их параллельной сортировке с последующим слиянием. В [7] приводится сравнение различных видов сортировок на современных вычислительных устройствах (в том числе и на GPU).

Стоит отметить попытки решить задачу сортировки при помощи машинного обучения [8], в частности, в [9] предлагается нейронная машина Тьюринга, с помощью которой алгоритмом сортировки пирамидой (*heap sorting*) сортируется массив из 20 элементов, а в [10] показана возможность сортировки массива из 10 элементов с использованием только нейронной сети.

При распараллеливании процессов дополнительные возможности возникают в случае их асинхронности. Для создания асинхронных систем с целью максимального использования существующих вычислительных ресурсов применяются сети Петри. Так в [11] рассмотрено решение задач организации связи между таксофонами и центром дистанционного контроля, в [12, 13] решаются задачи автоматики, среди которых есть задачи управления оборудованием водонапорных станций [13] и задача управления обжигом печи [13], а в [14] приводится пример решения задачи управления роботом-манипулятором, которая далее была развита в [13, 15, 16].

В связи с этим задача адаптации существующих алгоритмов для возможности их исполнения в параллельном и асинхронном режиме является актуальной.

Распараллеливание [1-3] и асинхронность могут быть применены и к задаче сортировки. В статье предлагается асинхронный алгоритм сортировки массива чисел, основывающийся на разбиении всего множества чисел массива на независимые пары и их сравнении. Можно рассматривать предлагаемый метод как адаптацию пузырьковой сортировки (*bubble sorting*) [4] к «многопузырьковому виду».

В отличие от традиционных алгоритмов сортировки, к которым можно отнести алгоритмы быстрой сортировки, сортировку вставками, слиянием и другие [4-6], предлагаемый алгоритм способен эффективно

исполняться за счет возможности максимального распараллеливания вычислений, то есть основным требованием к разрабатываемому алгоритму сортировки является эффективное использование возможности современных устройств, а именно – сортировка массивов в параллельном асинхронном режиме с целью непосредственного ускорения самой сортировки и более рационального использования ресурсов.

2. Алгоритм многопузырьковой сортировки. Основной принцип предлагаемого метода сортировки заключается в том, что из массива случайным образом выбираются пары соседних чисел, которые затем сравниваются (к примеру, в массиве из 100 элементов на одном условном такте (в силу того, что алгоритм является асинхронным, понятие тактов к нему не применимо, но под условным тактом тут и далее подразумевается исполнение возможных операций сравнения без сравнения повторяющихся пар чисел) одновременно могут сравниваться от 33 до 50 пар). После сравнения данная пара чисел блокируется для того, чтобы избежать их повторного сравнения. Если числа были переставлены в результате сравнения, то снимается блокировка с соседних чисел (слева и справа), так как имеет смысл выполнять сравнение для измененных пар. В случае если сравниваемая пара чисел не была переставлена, то есть левый элемент меньше, чем правый, то блокировка чисел соседних с рассматриваемой парой не снимается. Алгоритм заканчивает свою работу в случае запрета на сравнение для всех пар соседних чисел массива.

Схематичное представление алгоритма сортировки массива из четырех трехразрядных чисел приведено на рисунке 1. Числа массива на рисунке 1 изображены вертикальной последовательностью разрядов, представленных в виде окружностей. Секции сравнения, нарисованные на рисунке прямоугольниками, выполняют сравнение *i*-тых разрядов соседних чисел. Каждый разряд числа имеет левого и правого соседей и так как сравнение происходит попарно между соседними разрядами, каждый разряд числа (кроме крайних) относится к двум секциям сравнения, что представлено на рисунке как окружность, разделенная вертикальной чертой между двумя секциями сравнения.

В начальном состоянии метки находятся только в месте *start* и в местах представляющих разряды чисел в массиве. Работа приведенной системы начинается со срабатывания самого верхнего перехода, при котором метка из места *start* переходит в каждое из мест *c1-c4,* которые показывают доступность для сравнения соответствующих элементов массива. Изначально переходы, ведущие к местам *d1-d3*, показывающие, что соответствующие секции сравнения могут начать работу, доступны для срабатывания из-за того, что места *b1-b3* не содержат

меток. В начальных условиях места *a1-a3* также не содержат меток. Остальная логика работы части схемы с блокировками раскрывается ниже в процессе описания ее работы.

Все секции сравнения имеют одинаковое строение и отличаются лишь тем, какие разряды они представляют на рисунке 1.

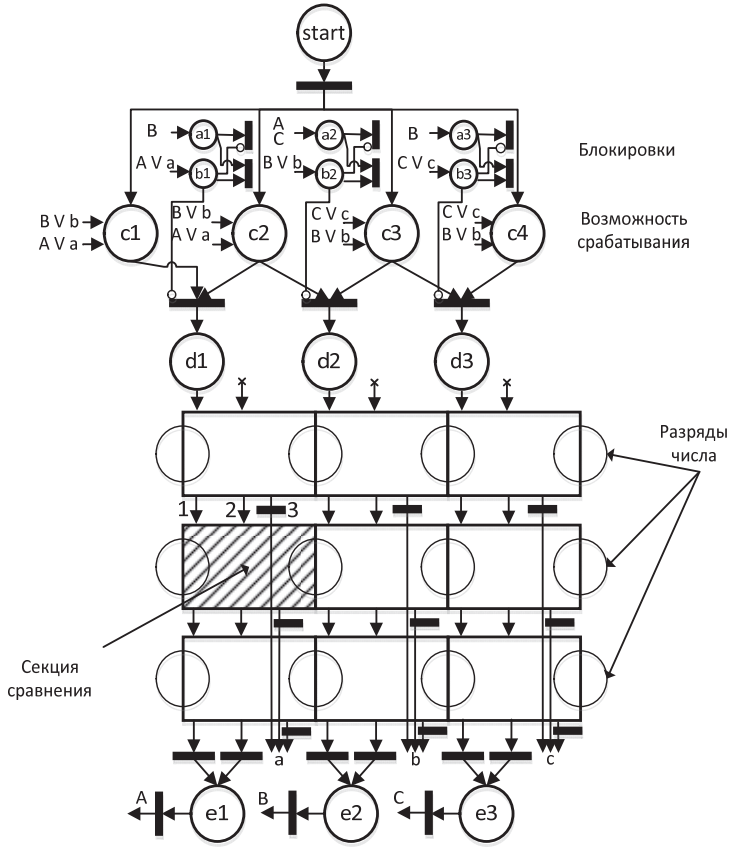

Рис. 1. Асинхронный алгоритм сортировки четырех трехразрядных чисел методом всплывающих пузырьков

Каждая секция сравнения содержит два входа – команду сравнения и команду перестановки разрядов. Самые верхние секции сравнения также содержат по два входа, но вход перестановки не соединен ни с каким местом из-за отсутствия необходимости смены всех чисел без предварительного сравнения (на рисунке изображен крестиком). Результатом сравнения двух разрядов могут быть не-

сколько исходов (изображены в виде переходов, исходящих из секции сравнения, и отмеченные цифрами 1-3): стрелка, отмеченная единицей, соответствует необходимости продолжения сравнения следующих разрядов. И она срабатывает при условии, что сравниваемые значения равны; двойка – поменять нижестоящие значения разрядов местами. Срабатывает при условии, что значение разряда слева больше чем значение разряда справа; тройка – закончить сравнение данной пары чисел. Срабатывает при условии, что левое число меньше правого. Результатом сравнения двух чисел является наличие меток в местах *e1-e3*. В вышеприведенных местах метки могут оказаться при равенстве сравниваемых чисел (срабатывание левого перехода) или перестановки чисел (правый переход). Далее метки из мест *e1-e3* переходят в места *a1-a3, b1-b3, c1-c4*. Рассмотрим работу переходов, связанных с местами *a1-a3, b1-b3,* на примере срабатывания переходов *a*, *A*.

Из места *e1* (переход *A*) метка переходит в места *c1*, *c2,* показывающие, что соответствующие числа массива не заняты (выполнено сравнение: числа равны или левое меньше правого), а также в место *b1* – для блокировки повторного срабатывания сравнения только что сравниваемых чисел, и в *a2* для снятия блокировки с соседних элементов (для мест, у которых есть левый и правый соседние элементы, также есть переходы для снятия блокировки с левого элемента).

При срабатывании перехода *a* (правое число больше левого) метка выполняет аналогичные переходы за исключением снятия блокировки с соседних элементов. Это связано с тем, что переход *a* означает, что числа упорядочены и не было выполнено перестановки. Окончанием работы приведенного алгоритма является блокировка переходов, ведущих к местам *d1-d3*, то есть наличие меток в местах *b1-b3*. Данного условия достаточно, так как метки из мест *b1-b3* могут быть удалены только при сравнении пар чисел, что не может быть из-за их блокировки. Переходы от меток *a1-a3*, *b1-b3* необходимы для того, чтобы снять блокировку с пар чисел. Возможны несколько вариантов: 1) место *b1* содержит метку, запрещающую сравнение первой пары чисел. Тогда она может быть снята, если в результате сравнения второй пары чисел сработал переход *B* и метка перешла в место *a1*. При этом срабатывает «нижний» переход для уничтожения блокирующей сравнение метки; 2) место *b1* не содержит метки, но в результате сравнения второй пары чисел сработал переход *B* и метка перешла в место *a1*. При этом срабатывает «верхний» переход (место *a1* содержит метку, а *b1* соединено ингибитор-

ной – запрещающей дугой с переходом) для уничтожения блокирующей сравнение метки.

На рисунке 2 приведена секция сравнения (на рисунке 1 она обозначена прямоугольником) двух *i*-тых разрядов. Входами для секции являются два перехода: команда сравнения чисел показана переходом *t1*, команда перестановки разрядов – переходом *t2*. Изначально метки со значениями разрядов чисел содержаться в местах *p1*, *p2*. Для большей наглядности на данном рисунке двунаправленной дугой показана пара дуг, идущих в противоположные направления.

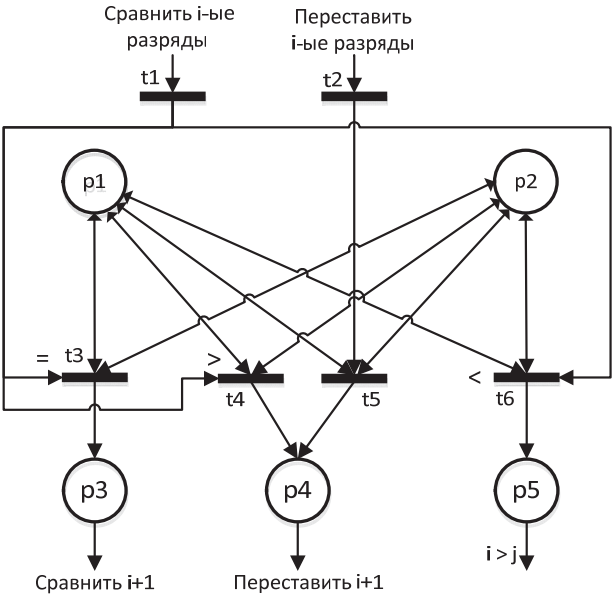

Рис. 2. Секция сравнения *i*-тых разрядов соседних чисел

При команде сравнения становится доступным для срабатывания один из переходов: *t3*, при условии, что значения в местах *p1* и *p2* одинаковы; *t4*, если значение в *p1* больше, чем в *p2*; *t6*, если значение в *p2* больше, чем в *p1*. При команде перестановки срабатывает переход *t5*, при котором метки меняются местами. В место *p3* метка попадает при условии, что значения в *p1* и *p2* одинаковы, в *p4* – если значение в *p1* больше чем в *p2* или при команде перестановки и в *p5* при условии, что значение в *p1* меньше чем в *p2*.

3. Реализация алгоритма сортировки. В качестве программного пакета для реализации приведенного алгоритма с использованием сетей Петри был выбран CPN Tools 4.0.1.

Реализация основной схемы (рисунок 1) разбита на несколько частей из-за громоздкости схемы и приведена на рисунке 3 и 4 (причем часть 2 на рисунке 4 является расширением части 1 на рисунке 3 и должна находиться вместо многоточия на рисунке 3). На рисунке 4 представлена часть схемы с набором секций сравнения для сортировки массива из четырех трехразрядных разрядов, на рисунке 3 часть с логикой блокировок.

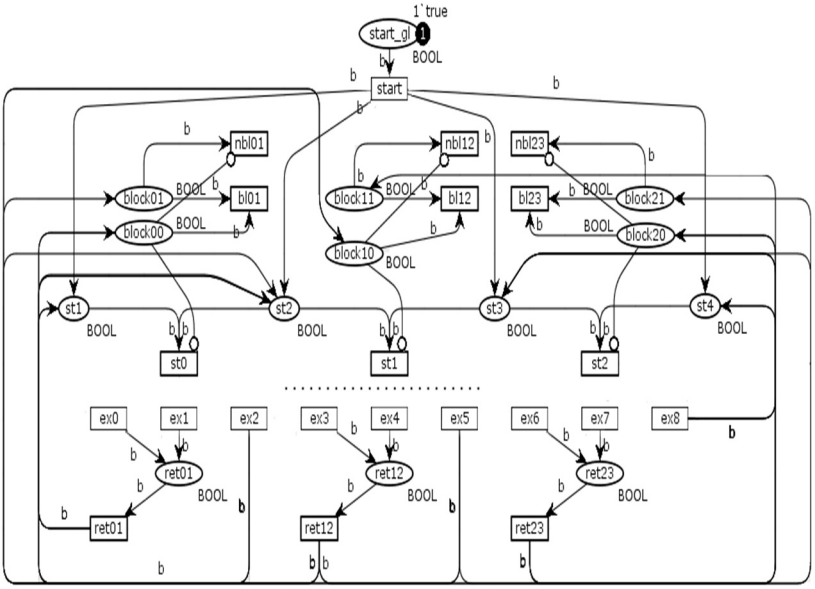

Рис. 3. Реализация алгоритма сортировки в сетях Петри: часть 1

Рассмотрим работу приведенной схемы. Работа алгоритма начинается с срабатывания перехода *start* (дуги, соединяющие переходы и метки, снабжены переменными, которые «переносят» значения меток) и перехода меток в места *st1-st4*. Переходы *st0-st2* при этом становятся доступными для срабатывания из-за того, что места *block00-block20* соединены ингибиторными дугами с переходами и не содержат меток. Через переходы *st0-st2* метки попадают в места *st0_in-st2_in* на рисунке 4.

На рисунке 4 приведена часть сети Петри предлагаемого алгоритма, изображенного на рисунке 1.

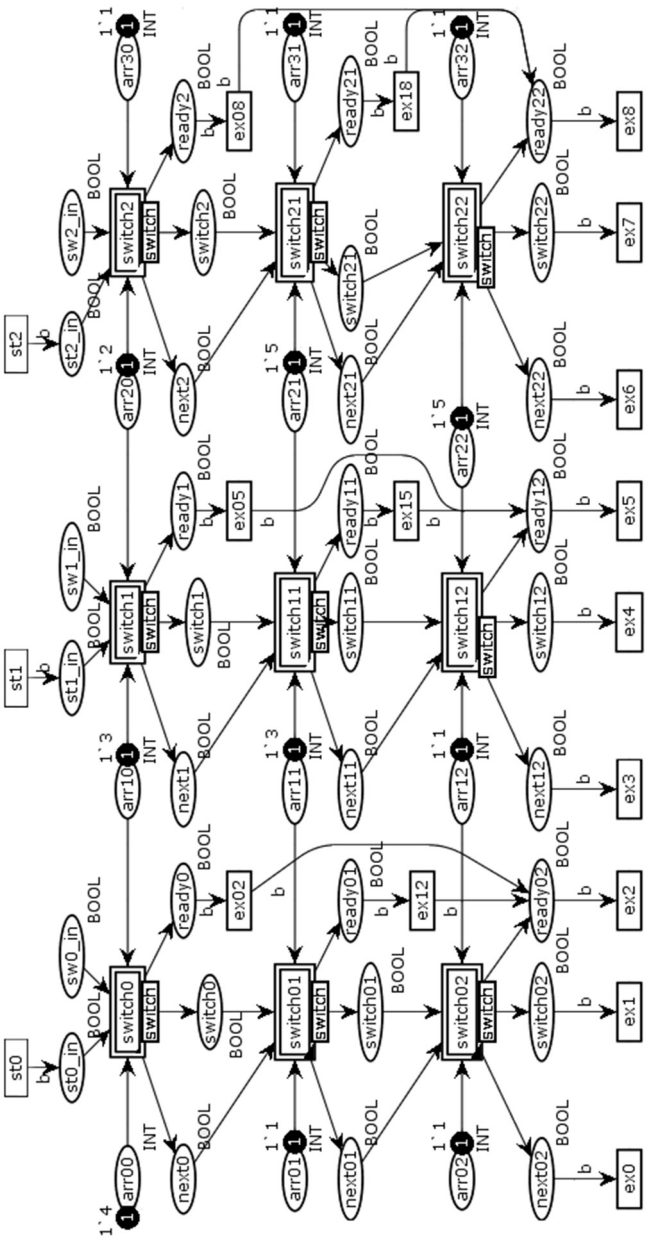

Рис. 4. Реализация алгоритма сортировки в сетях Петри: часть 2

В данной конкретной реализации использованы иерархические сети Петри, и переходы, отмеченные надписью *switch* в прямоугольнике, являются подсетью основной сети Петри, содержащие секции сравнения. Для реализации иерархических сетей Петри в CPN Tools необходимо соединить места в основной сети с входными/выходными местами в подсети. Таким образом, все места на рисунке 4 соединены с аналогичными местами с соответствующими секциями сравнения (рисунок 5). Например, для перехода *switch0* входные места *st0_in* и *sw0_in* соединены с местами *start* и *switch* соответственно, места *arr00* и *arr10* с *p1*, *p2* соответственно, а места *next0*, *switch0* и *ready0* с местами *next_out, switch_out* и *ready_out* соответственно. Остальные секции имеют аналогичное строение. Сами разряды чисел содержатся в местах *arr00-arr32 (array)* и представлены метками (в CPN Tools метки обозначаются записью «1`2», где 1 – число меток, 2 – их значение).

После последнего слоя секций сравнения метки могут находиться либо в местах *ret01-ret23*, либо в *ready02-ready22*, что соответствует тому, что сравниваемые элементы равны или были переставлены (*ret**), либо упорядочены (*ready**). При любом исходе метки переходят в места *st1-st4* (в зависимости от секции) и блокируют сравнение. Если метка оказалась в местах *ret01-ret23*, то дополнительно снимается блокировка для соседних элементов путем перехода метки в места *block01-block21* и срабатывания переходов *bl01-bl23*. Если в результате работы секции сравнения оказалось, что элементы упорядочены, то разблокировка соседних элементов не выполняется. Остановка работы приведенной сети выполняется, когда все пары соседних чисел заблокированы для сравнения, то есть метки находятся в местах *b1-b3*.

4. Реализация секции сравнения. На рисунке 5 представлена реализация секции сравнения двух разрядов (рисунок 2) соседних чисел. Данная схема является детализацией перехода *switch* на рисунке 4, а именно *switch0*. Места *p1*, *p2* связаны с местами *arr01*, *arr11*. Места *start* и *switch* показывают наличие команды на начало сравнения и перестановки соответственно. Выходные места *next_out, switch_out* и *ready_out* показывают результаты исполнения секции сравнения, два из которых (*next_out, switch_out*) передаются в следующие секции сравнения, а *ready_out* «собирается» в *ex2* в зависимости от секции.

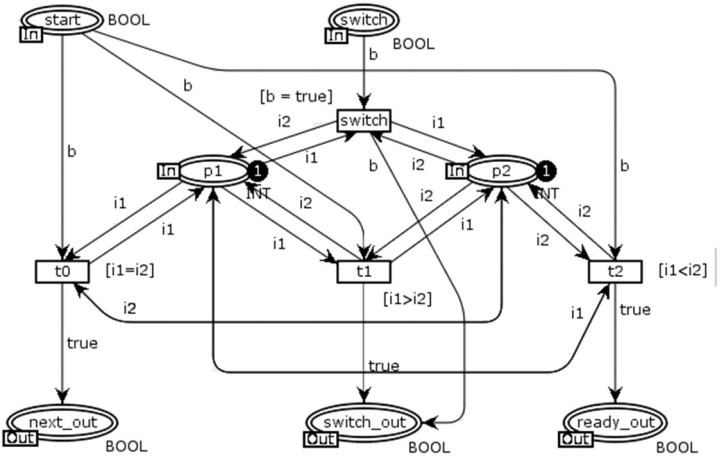

Рис. 5. Реализация секции сравнения алгоритма сортировки в сетях Петри

Переход *t0* срабатывает при условии, что метки в местах *p1* и *p2* равны (проверка осуществляется с помощью защитного условия «*i1=i2*» на переходе). При этом переменные *i1, i2* (с помощью которых выполняется «перенос» меток с места к переходу) не меняют местами исходные метки. При переходе *t1*, который срабатывает, если значение метки в *p1* больше, чем в *p2* (защитное условие «*i1>i2*»), исходные метки меняются местами (переменные *i1* и *i2* возвращаются в разные места). Переход *t2* срабатывает, если значение в *p1* меньше, чем в *p2* (защитное условие «*i1<i2*»).

5. Пример работы алгоритма. Рассмотрим работу полученной реализации алгоритма на примере сортировки следующего массива:

$$\{13, 12, 16, 15, 14, 1\},$$

который состоит только лишь из шести элементов для компактности изложения. В сетях Петри в случае возможности срабатывания нескольких переходов срабатывание любого из доступных является равновероятным событием. Поэтому при сортировке массива предложенным алгоритмом нет однозначной последовательности перестановок. Для вышеприведенного массива результат работы алгоритма приведен в таблице 1, где в левой колонке показан номер условных тактов (номеров срезов состояния массива) от начального до того момента как массив будет отсортирован. Сравниваемые пары на условных тактах выделены. В запуске, приведенном в средней колонке, на первом условном такте сравниваются пары чисел {12, 16} и {14, 15} (далее по

номерам индексов элементов массива считая с нулевого индекса). Так как в результате первого такта 1–4 элементы массива заблокированы, то на втором условном такте доступными для сравнения остаются пары {0, 1}, {2, 3} и {4, 5}. Третий условный такт сравнения совпадает с первым. На четвертом такте пара с индексами {0, 1} – заблокирована, остальные доступны для сравнения. На пятом такте сравниваются 1 и 2 элементы, и далее на шестом сравнение завершается парой {0, 1}.

Таблица 1. Сортировка массива из шести элементов методом всплывающих пузырьков

Номер условного такта	Состояние массива (запуск 1)	Состояние массива (запуск 2)
0	13 12 16 15 14 1	13 12 16 15 14 1
1	13 **12 16 14 15** 1	**12 13 15 16 1 14**
2	**12 13 14 16 1 15**	12 **13 15 1 16** 14
3	12 **13 14 1 16** 15	12 13 **1 15 14 16**
4	12 13 **1 14 15 16**	12 **1 13 14 15** 16
5	12 **1 13** 14 15 16	**1 12 13 14 15 16**
6	**1 12** 13 14 15 16	-

В данном примере сортировка массива выполнена за 6 условных тактов. В виду случайности выбора пар при различных запусках количество условных тактов может меняться. К примеру, тот же самый массив может быть отсортирован за другое количество условных тактов (таблица 1, запуск 2). В худшем случае сортировка будет выполнена за девять условных тактов.

6. Оценка быстродействия алгоритма. Рассмотрим оценку быстродействия приведенного алгоритма. Асимптотическая сложность классического алгоритма пузырьковой сортировки $O(n^2)$ [4], параллельный вариант чет-нечет сортировки $O(log(2n))$. Рассмотрим наилучший и наихудший возможные случаи оценки.

В наилучшем случае массив уже отсортирован, тогда после сравнения соседних элементов массива будут заблокированы непосредственно сравниваемые элементы, а метки для снятия блокировки с соседних элементов не будут переходить в места для снятия блокировок. Иллюстрация вариантов разбиения массива на пары соседних элементов на примере массива из шести элементов приведена на рисунке 6. В верхней его части представлены два варианта, когда весь массив разбивается через один элемент. В таких случаях для разбиения на пары всего массива требуется по три условных такта. В нижней части рисунка представлено разбиение, в котором нет пропусков элементов. В данном случае для полного разбиения массива требуется всего

два условных такта. Таким образом, для полной сортировки всего массива требуется 3 и 2 тактов соответственно.

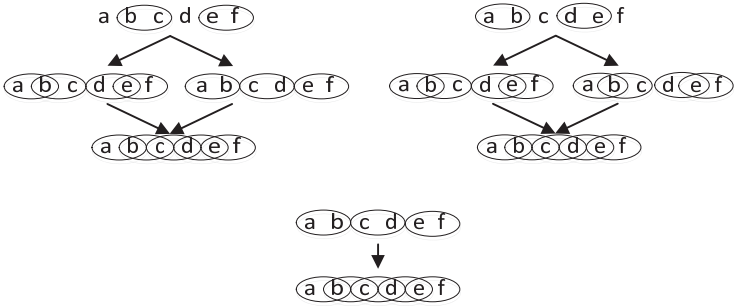

Рис. 6. Возможные варианты разбиения массива на пары чисел

В наихудшем случае массив отсортирован в обратном порядке, если учитывать, что в реализации пузырьковой сортировки необходимо совершить $n\text{-}1 + n\text{-}2 + n\text{-}3 \ldots + 1 = (n + 1 - 1)*n/2 = n^2/2$ сравнений [4]. Учитывая, что в предлагаемом алгоритме в среднем одновременно может выполняться $n/2$ или $n/3$ действий (перестановка или сравнение), то общее количество действий будет $(n^2/2)/(n/2) = n$ или $(n^2/2)/(n/3) = 3n/2$ в зависимости от разбиения массива на пары соседних элементов.

Для оценки быстродействия в среднем следует оценить математическое ожидание функции распределения возможного количества пар соседних элементов. Например, для массива из 16 элементов 8 пар возможны в 1 комбинации, 7 пар – 28 комбинаций, 6 пары – 32 комбинации, 5 пар – 1 комбинация. Тогда математическое ожидание примерно соответствует шести парам. То есть на одном условном такте выполняется анализ шести пар чисел, что лежит между худшим и лучшим случаями.

7. Заключение. В статье продемонстрирована возможность использования сетей Петри с целью распараллеливания процессов обработки информации, что в данном случае выполнено на примере классической задачи сортировки массивов. Приводится обобщенный алгоритм пузырьковой сортировки, который позволяет выполнять параллельную асинхронную сортировку массива элементов. В приведенной в работе реализации алгоритма с использованием сети Петри используется представление числа в поразрядном виде, и, в частности, возможна обработка чисел в двоичном виде. При необходимости числа могут быть свернуты и тогда они будут представлены в виде одного

слоя секций сравнения (рисунок 1). Приведен иллюстративный пример работы сортировки массива.

Принципы, примененные для построения алгоритма, могут быть применены к другим типам сортировки, например, сортировки слиянием или быстрой сортировки, что и является целью дальнейших исследований.

Выполнена оценка быстродействия приведенного алгоритма для наилучшего и наихудших случаев. В наилучшем случае, когда массив уже отсортирован, рассматриваемый алгоритм выполняется за 2 или 3 условных такта в зависимости от разбиения массива на пары соседних элементов. В наихудшем случае – за n или за $3n/2$, где n – число элементов.

Литература

1. *Wilkinson B., Allen M.* Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition) // Pearson Education. 2005. 431 p.
2. *Kirk D.B., Hwu W.W.* Programming Massively Parallel Processors, Second Edition: A Hands-on Approach // Morgan Kaufmann. 2012. 496 p.
3. *Малышкин В. Э., Корнеев В. Д.* Параллельное программирование мультикомпьютеров // Издательство НГТУ. 2011. 296 с.
4. *Cormen T., Leiserson C., Rivest R., Stein C.* Introduction to Algorithms: 3rd Edition // The MIT Press. 2009. 1328 p.
5. *Jan B., Montrucchio B., Ragusa C., Khan F. G., Khan O.* Fast parallel sorting algorithms on GPUs // International Journal of Distributed and Parallel Systems (IJDPS). 2012. pp. 107–118.
6. *Inoue H., Moriyama T., Komatsu H., Nakatani T.* AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors // Proceedings on 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007). 2007. pp. 189–198.
7. *Capannini, G., Silvestri F., Baraglia R.* Sorting on GPUs for large scale datasets: A thorough comparison // Information Processing and Management. 2011. vol. 48(5). pp. 903–917.
8. *Haykin S.* Neural Networks and Learning Machines, 3rd Edition // Pearson Education. 2009. 938 p.
9. *Graves A. Wayne G., Danihelka I.* Neural Turing Machines // The Computing Research Repository 1410. 5401. 2014. pp. 1–24.
10. *Воевода А.А., Полубинский В. Л., Романников Д.О.* Сортировка массива целых чисел с использованием нейронной сети // Научный Вестник НГТУ. 2016. №2(63). С. 151–157.
11. *Коротиков С.В., Саркенов Д.О.* Применение спецификации эквивалентности в моделировании сеанса связи таксофона и центра дистанционного контроля и управления таксофонами раскрашенной сетью Петри // Сб. науч. тр. НГТУ. 2007. № 3 (49). С. 97–104.
12. *Марков А.В.* Автоматизация проектирования и анализа программного обеспечения с использованием языка UML и сетей Петри: дисс. канд. техн. наук // Новосибирск. 2015. 176 с.

13. *Воевода А.А., Марков А.В., Романников Д.О.* Разработка программного обеспечения: проектирование с использованием UML диаграмм и сетей Петри на примере АСУ ТП водонапорной станции // Труды СПИИРАН. 2014. №3(34). С. 218–232.
14. *Марков А.В.* Поиск манипулятором кратчайшего пути в лабиринте // Сб. науч. тр. НГТУ. 2011. №4(66). С. 75–90.
15. *Марков А.В., Воевода А.А.* Развитие системы “Перемещение манипулятора в пространстве с препятствиями” при помощи рекурсивных функций // Автоматика и программная инженерия. №2(4). 2013. С. 35–41.
16. *Марков А.В.* Свойства инверсии сетей Петри // Сб. науч. тр. НГТУ. 2014. №4(78). С. 139–152.

Воевода Александр Александрович — д-р техн. наук, профессор, профессор кафедры автоматики, Новосибирский государственный технический университет. Область научных интересов: полиномиальный синтез, сети Петри, UML диаграммы. Число научных публикаций — 200. voevoda@ucit.ru; пр. Карла Маркса 20, Новосибирск, 630073; р.т.: +79139223092.

Романников Дмитрий Олегович — к-т техн. наук, доцент, доцент кафедры автоматики, Новосибирский государственный технический университет. Область научных интересов: машинное обучение, нейронные сети, сети Петри. Число научных публикаций — 45. dmitry.romannikov@gmail.com; пр. Карла Маркса 20, Новосибирск, 630073; р.т.: +7 961 223 8567.

Поддержка исследований. Работа выполнена при финансовой поддержке Министерства образования и науки РФ (проект № 2014/138).

A.A. VOEVODA, D.O. ROMANNIKOV

ASYNCHRONOUS SORTING ALGORITHM FOR ARRAY OF NUMBERS WITH THE USE OF INHIBITORY PETRI NETS

Voevoda A.A., Romannikov D.O. **Asynchronous Sorting Algorithm for Array of Numbers With the use of Inhibitory Petri Nets.**

Abstract. Currently the tasks of computations speed-up and/or their optimization are actual ones. Among the ways to solve these tasks is a method of parallelization and asynchronization of a sorting algorithm, which is considered in the article. We offer a sorting method that is based on the principle of dividing an array into a set of independent pairs of numbers and their parallel and asynchronous comparison, which distinguishes the offered method from the traditional sorting algorithms (like quick sorting, merge sorting, insertion sorting and others). The algorithm is realized with the use of Petri nets as the most suitable tool for describing asynchronous systems. Examples of its work are given. The performance of the algorithm is evaluated for the best and the worst cases. In the best case the algorithm is executed for 2 or 3 conditional tacks depending on an array partition into the pairs of adjacent elements. In the worst case –for n or 3n/2, where n is the number of elements. Parallelization and asynchronization principles, used during the algorithm construction, can also be used for different algorithms.

Keywords: sorting algorithms, bubble sorting, Petri nets, asynchronous, parallel processing.

Voevoda Alexandr Aleksandrovich — Ph.D., Dr. Sci., professor, professor of automation department, Novosibirsk State Technical University. Research interests: polynomial synthesis, UML diagrams, Petri nets. The number of publications — 200. voevoda@ucit.ru; 20, Karl Marx Avenue, Novosibirsk, 630073; office phone: +79139223092.

Romannikov Dmitry Olegovich — Ph.D., associate professor, associate professor of automation department, Novosibirsk State Technical University. Research interests: machine learning, neural networks, Petri nets.. The number of publications — 45. dmitry.romannikov@gmail.com; 20, Karl Marx Avenue, Novosibirsk, 630073; office phone: +7 961 223 8567.

Acknowledgements. This research is supported by RFBR (grant 2014/138).

References

1. Wilkinson B., Allen M. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition). Pearson Education. 2005. 431 p.
2. Kirk D.B., Hwu W.W. Programming Massively Parallel Processors, Second Edition: A Hands-on Approach. Morgan Kaufmann. 2012. 496 p.
3. Malyshkin V.Je., Korneev V.D. *Parallel'noe programmirovanie mul'tikomp'juterov* [Parallel programming of multicomputers]. Izdatel'stvo NGTU. 2011. 296 p. (In Russ.).
4. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms: 3rd Edition. The MIT Press. 2009. 1328 p.
5. Jan B., Montrucchio B., Ragusa C., Khan F. G., Khan O. Fast parallel sorting algorithms on GPUs. *International Journal of Distributed and Parallel Systems (IJDPS)*. 2012. pp. 107–118.
6. Inoue H., Moriyama T., Komatsu H., Nakatani T. AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors. Proceedings on 16th International Con-

ference on Parallel Architecture and Compilation Techniques (PACT 2007). 2007. pp. 189–198.
7. Capannini, G., Silvestri F., Baraglia R. Sorting on GPUs for large scale datasets: A thorough comparison. *Information Processing and Management*. 2011. vol. 48 (5). pp. 903–917.
8. Haykin S. Neural Networks and Learning Machines: 3rd Edition. Pearson Education. 2009. 938 p.
9. Graves A. Wayne G., Danihelka I. Neural Turing Machines. The Computing Research Repository 1410.5401. 2014. pp. 1–24.
10. Voevoda A.A., Polubinskij V. L., Romannikov D.O. Array of integers sorting with a using of a neural network. *Nauchnyj Vestnik NGTU – Science Bulletin of NSTU*. 2016. vol. 2 (63). pp. 151–157. (In Russ.)
11. Korotikov S.V., Sarkenov D.O. Application Specification equivalence in the modeling session payphone and remote monitoring and control center payphones colored Petri net. *Sb. nauch. tr. NGTU – Collection of scientific works of NSTU*. 2007. № 3 (49). pp. 97–104. (In Russ.).
12. Markov A.V. *Avtomatizacija proektirovanijai analiza programmnogo obespechenijas ispol'zovaniem jazyka UML i setej Petri diss. kand. tehn. nauk* [Design automation and the analysis of the software with use of the UML language and Petri nets. Ph.D. thesis]. Novosibirsk: 2007. 176 p. (In Russ.).
13. Voevoda A.A., Markov A.V., Romannikov D.O. Software development: design using UML diagrams and Petri nets on the example of PCS pumping station. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2014. vol. 3 (34). pp. 218–232. (In Russ.).
14. Markov A.V. Search manipulator shortest path in a maze. *Sb. nauch. tr. NGTU – Collection of scientific works of NSTU*. 2011. vol. 4(66). pp. 75–90.
15. Markov A.V. Voevoda A.A. Development of the system "manipulator move in space with obstacles" with recursive functions. *Avtomatika i programmnaja inzhenerija – Automation and Software Engineering*. 2013. no. 2(4). pp. 35–41.
16. Markov A.V. Properties of Petri nets inversion. *Sb. nauch. tr. NGTU – Collection of scientific works of NSTU*. 2014. vol. 4(78). pp. 139–152.