Алгоритмы асинхронных круговых турниров для многозадачных приложений обработки данных

С.В. Востокин, И.В. Бобылева

***Аннотация* — В статье рассматривается методика синтеза графов зависимостей задач для многозадачных приложений, выполняющих параллельную асинхронную обработку данных по принципу кругового спортивного турнира. Описаны следующие составные элементы методики: семейство алгоритмов круговых турниров определено в форме алгоритмического скелета; предложен метод синтеза графа кругового турнира по базовому последовательному алгоритму; показано дальнейшее использование синтезированных графов для реализации турниров в форме параллельного и асинхронного вычислительного процесса. Рассмотрена графическая интерпретация процедур круговых турниров, на основе которой можно строить аналитические оценки времени выполнения турниров. Методика иллюстрируется тремя примерами построения конкретных алгоритмов турниров: турниром без дополнительных ограничений, простым сортирующим турниром, оптимизированным сортирующим турниром. Построены алгоритмы конструирования графов зависимостей задач для перечисленных турниров. Показан принцип реализации асинхронных параллельных вычислений по данным графам. Выполнен расчет числа раундов и ускорения с использованием предложенной графической иллюстрации процесса проведения турниров. Рассмотренная методика синтеза графов зависимостей задач в асинхронных круговых турнирах и построенные по ней алгоритмы ориентированы на широкий класс параллельных архитектур, включающий вычислительные кластеры, грид-системы предприятий и гибридные облака.**

***Ключевые слова* — многозадачные вычисления, параллелизм задач, круговой турнир, алгоритмический скелет.**

I. Введение

Многозадачные приложения – распространенная архитектура для организации параллельных и распределенных вычислений [1]. Такая архитектура поддерживается, например, в стандартизированном инструменте параллельного программирования систем с общей памятью OpenMP [2], а также CilkPlus [3] и TBB [4]. В многозадачных приложениях высокопроизводительных распределенных вычислений используются платформы BOINC [5], Everest [6], HTCondor [7], X-Com[8] и др. Перечисленные инструменты и платформы автоматизируют управление вычислениями, реализуя механизмы назначения задач на ресурсы, обеспечения надежных и отказоустойчивых вычислений, преодоления проблем гетерогенности вычислительной среды. Инструментальная поддержка многозадачных вычислений реализована для автономного компьютера, кластерной системы, грид-систем масштаба предприятия и масштаба сети Интернет.

Наличие готовых механизмов управления многозадачными вычислениями значительно упрощает разработку прикладных программ. Тем не менее приложение для многозадачных вычислений должно самостоятельно управлять порядком запуска задач. В простейшем случае чрезвычайной параллельности управление сводится к одновременному запуску и одновременной обработке результатов всех задач [9]. Однако на практике обработка результатов задач и запуск новых задач могут выполняться в темпе вычислений. В данном случае управление порядком запуска задач может представлять сложность для прикладного программиста.

Для уменьшения сложности написания многозадачных приложений разрабатываются специальные инструменты и технологии автоматизации программирования, такие как системы управления потоком работ [10], которые автоматизируют описание графов зависимостей задач. В данной работе для представления зависимостей между задачами применяется подход на основе алгоритмических скелетов [11, 12]. В нем мы описываем многозадачные вычисления в форме комбинации последовательных алгоритмов, являющихся параметрами функций высшего порядка. Неявно заданные функции высшего порядка описывают управление вычислениями, при этом подразумевается, что их семантика понятна программисту. Известным примером реализации этой концепции в области высокопроизводительной потоковой обработки данных на кластерных системах является модель MapReduce [13].

Работа посвящена описанию алгоритмических скелетов для многозадачных алгоритмов типа

Статья получена 04 февраля 2020 года.
С.В. Востокин, доктор технических наук, доцент, Самарский национальный исследовательский университет имени академика С.П. Королева, профессор кафедры информационных систем и технологий (e-mail: easts@mail.ru).
И.В. Бобылева, Самарский национальный исследовательский университет имени академика С.П. Королева, аспирант кафедры информационных систем и технологий (e-mail: ikazakova90@gmail.com).

«асинхронный круговой турнир», цель которых – организация попарной обработки элементов информационного массива для более широкого класса вычислительных систем, включающего кластерные системы, грид-системы и гибридные облачные системы [14]. Попарная многозадачная обработка данных встречается во многих приложениях, однако она сложна даже в случае отсутствия дополнительных ограничений на порядок выполнения задач [15] и может представлять проблему при задании специальных ограничений [16]. Мы рассматриваем алгоритмы, в которых один и тот же элемент данных не может обрабатываться в двух одновременно исполняющихся задачах, что может использоваться при сортировке и анализе частоты повторений элементов в больших массивах [17].

Работа состоит из следующих частей. В начале рассматриваются алгоритмический скелет «асинхронный круговой турнир» (ART – asynchronous round-robin tournament) и постановка задачи управления вычислениями. Затем излагается общий метод синтеза графов турниров по базовым последовательным алгоритмам. Далее описываются три конкретных алгоритма типа ART: без дополнительных ограничений, простой сортирующий и оптимизированный сортирующий турнир. Приводится расчет ускорений для данных алгоритмов. В заключении представлены основные результаты работы.

II. Асинхронный круговой турнир

При описании алгоритмического скелета ART мы используем аналогии со спортивным круговым турниром. Пусть имеется M команд-участников турнира. Требуется построить расписание игр такое, чтобы каждая команда сыграла со всеми другими командами по одной игре, при этом запрещено планировать несколько игр с участием одной команды на одно время. Асинхронность турнира подразумевает, что команда может сразу же приступать к новой игре, как только она и команда-соперник закончила очередную игру или закончена процедура подготовки. В отличие от описываемого турнира, спортивные турниры обычно делятся на раунды. Асинхронность же существенна при вычислениях в распределенной гетерогенной среде.

С точки зрения программиста-пользователя алгоритмический скелет ART представляет собой функцию высшего порядка, параметрами которой являются определяемые программистом процедуры prepare() и play(): подготовка prepare(i) команды $i \in \{0,1,\ldots,M-1\}$ к игре play(i,j), где $j \in \{0,1,\ldots,M-1\} \wedge i<j$.

Листинг 1. Алгоритмический скелет ART

```
ART(
   prepare(integer)
   play(integer,integer)
)
```

Моделью управления вызовами процедур prepare() и play() для программиста-пользователя может служить последовательный алгоритм, например:

```
для i от 0 до M-1 выполнять prepare(i), затем
для i от 1 до M-1, для j от 0 до j<i выполнять play(j,i).
```

Алгоритмический скелет ART реализует параллельный асинхронный аналог подобного последовательного алгоритма. Для получения конкретного алгоритма на основе алгоритмического скелета ART программисту-пользователю потребуется только определить процедуры prepare() и play(). Например, определяя процедуру prepare(i) как ввод i-ого элемента массива A[0], A[1], A[i], … , A[M-1], а процедуру play(i,j) как перестановку значений элементов массива A[i] и A[j] в случае, если A[i] > A[j], он получит асинхронный и параллельный алгоритм сортировки элементов массива.

III. Метод синтеза графа турнира по базовому последовательному алгоритму

В работе [18] нами показано, что точная семантика асинхронного параллельного управления играми турнира может быть задана с использованием модели акторов и концепции алгоритмического скелета. Работа алгоритма [18] основана на интерпретации ориентированного ациклического графа зависимостей игр турнира, закодированного двумя матрицами I_1 и I_2 размерности M×M следующим образом. Дуга $(i,j) \rightarrow (I_1[j,i], I_1[i,j])$ на графе зависимости игр турнира обозначает, что игра play(I_1[j,i], I_1[i,j]) играется непосредственно после игры play(i,j). Аналогично дуга $(i,j) \rightarrow (I_2[j,i], I_2[i,j])$ обозначает, что игра play(I_2[j,i], I_2[i,j]) играется непосредственно после игры play(i,j). Если после игры play(i,j) больше не играются игры с участием команд i, j или играется только одна игра (кодируемая в матрице I_1 или I_2), то не используемые элементы матриц I_1 или I_2 заполняются специальными значениями NA (см. рис. 1). Суть управления играми турнира заключается в том, что ведется подсчет количества сыгранных игр, предшествующих данной игре. Как только сыграны все предшествующие игры, запускается данная игра. Таким образом определение скелета ART сводится к определению матриц I_1 и I_2.

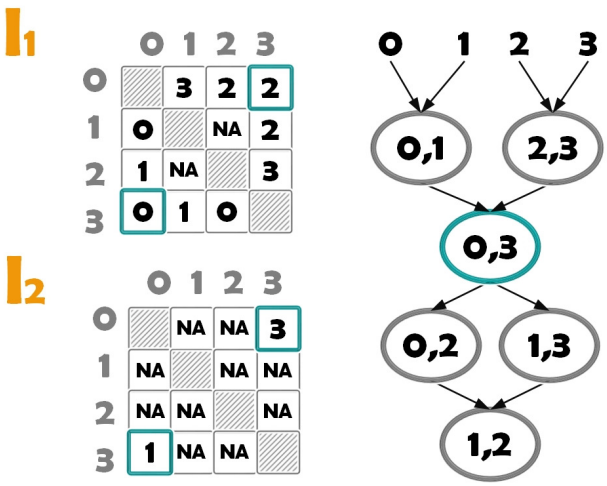

Рис. 1. Кодирование графа зависимостей игр турнира

Рассмотрим принцип работы алгоритма, заполняющего матрицы I_1 и I_2. Пусть имеется некоторый алгоритм, определяющий последовательность игр турнира. Между играми получившейся последовательности имеются информационные зависимости: для команды, участвующей в некоторой игре важен только порядок игр, в которых принимает участие именно она. Порядок остальных игр для рассматриваемой команды не существенен. Учитывая это строится искомый граф зависимостей игр турнира: взяв некоторую команду i нужно соединить дугами все ближайшие вершины последовательности, содержащие i (рис. 2). Распараллеливание простейших турниров 4-х команд разных типов показано на рис. 3.

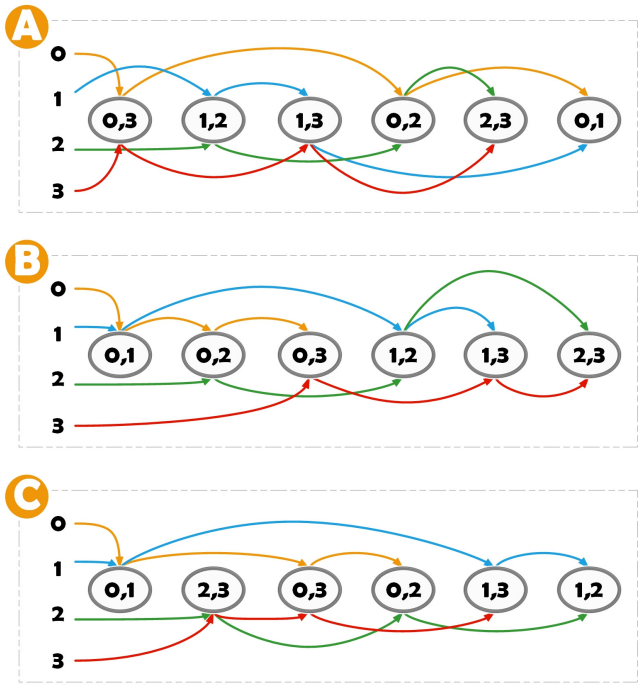

Рис. 2. Иллюстрация распараллеливания простейших турниров 4-х команд, исходная последовательность игр: (a) – турнир без дополнительных ограничений; (b) – простой сортирующий турнир; (c) – оптимизированный сортирующий турнир

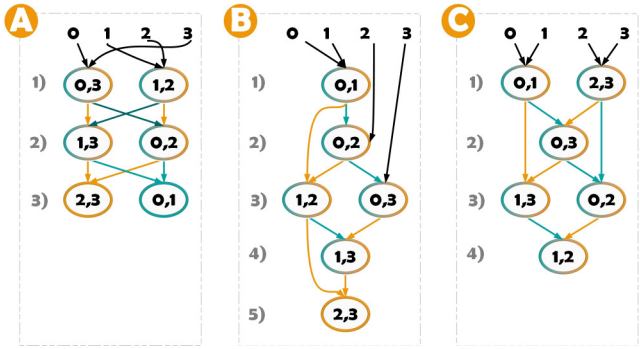

Рис. 3. Иллюстрация распараллеливания простейших турниров 4-х команд, графы турниров в ярусно-параллельной форме: (a) – турнир без дополнительных ограничений; (b) – простой сортирующий турнир; (c) – оптимизированный сортирующий турнир

С учетом описанного принципа обобщенный алгоритм TDAG (tournament directed acyclic graph) записывается, как показано ниже.

Листинг 2. Обобщенный алгоритм TDAG

```
переменные: массивы I1[M,M], I2[M,M], D1[M,M],
  D2[M,M], B[M,2], C[M,M]
  начало
для i от 0 до M-1 выполнять
  | B[i,0] := NA
  | B[i,1] := NA
  | для j от 0 до M-1 выполнять
  | | D1[i,j] := NA
  | | D2[i,j] := NA
  | | I1[i,j] := NA
  | | I2[i,j] := NA
  | | если i<j то C[i,j] := false
выполнить build()
для i от 0 до M-1, для j от i+1 до M-1 выполнять
  | если C[i,j]=false то
  | |    build определен неправильно, завершение
  | если D1[i,j] ≠ NA то
  | | если I1[D1[i,j],D1[j,i]] = NA то
  | | | I1[D1[i,j],D1[j,i]] := i
  | | | I1[D1[j,i],D1[i,j]] := j
  | | если I2[D1[i,j],D1[j,i]] = NA то
  | | | I2[D1[i,j],D1[j,i]] := i
  | | | I2[D1[j,i],D1[i,j]] := j
  | если D2[i,j] ≠ NA то
  | | если I1[D2[i,j],D2[j,i]] = NA то
  | | | I1[D2[i,j],D2[j,i]] := i
  | | | I1[D2[j,i],D2[i,j]] := j
  | | если I2[D2[i,j],D2[j,i]] = NA то
  | | | I2[D2[i,j],D2[j,i]] := i
  | | | I2[D2[j,i],D2[i,j]] := j
конец
```

В алгоритме помимо вычисляемых матриц I_1 и I_2, описанных выше, используются следующие вспомогательные матрицы. D_1[M,M], D_2[M,M] – матрицы, показывающие после каких игр играется данная игра: игра(i,j) играется после игры (D[j,i], D[i,j]). B[M,2] – матрица, показывающая в какой игре (B[i,0], B[i,1]) последний раз участвовала команда i. C[M,M] – матрица из элементов над главной диагональю: если C[i,j]=true, то игра (i,j) сыграна.

Данный алгоритм является обобщенным и может быть описан следующей схемой:

Листинг 3. Схема обобщенного алгоритма TDAG

```
TDAG(
   build() {
      game(integer, integer)
   } ).
```

Для того, чтобы определить конкретный алгоритм из данного обобщенного алгоритма необходимо в качестве параметра задать другой алгоритм build(), в котором в качестве вспомогательного вызывается алгоритм game(). Алгоритм build() и определяет исходную

распараллеливаемую последовательность игр турнира. Алгоритм game() определен в обобщенном алгоритме TDAG, как показано ниже.

Листинг 4. Алгоритм game()
входные переменные:
$team_0$, $team_1$ в вызове game($team_0$, $team_1$)
начало
если $C[team_0,team_1]$ = true □ $team_0 \geq team_1$ то
  build определен неправильно, завершение
иначе C[i,j] := true
если $B[team_0,0] \neq NA$ то
  $D_1[team_1,team_0] := B[team_0,0]$
  $D_1[team_0,team_1] := B[team_0,1]$
если $B[team_1,0] \neq NA$ то
  $D_2[team_1,team_0] := B[team_1,0]$
  $D_2[team_0,team_1] := B[team_1,1]$
$B[team_0,0] := team_0$; $B[team_0,1] := team_1$
$B[team_1,0] := team_0$; $B[team_1,1] := team_1$
конец

Обобщенный алгоритм TDAG вначале строит исходный граф зависимостей игр турнира на основе матриц D_1 и D_2. Затем инвертирует исходный граф, меняя ориентацию дуг на противоположную. В итоге получается искомый граф в матрицах I_1 и I_2. Дополнительно контролируется правильность работы алгоритма build() с использованием матрицы C.

Далее рассматриваются некоторые конкретные варианты определения build() в обобщенном алгоритме TDAG.

IV. Турнир без дополнительных ограничений

Построим асинхронный вариант традиционного спортивного кругового турнира с использованием обобщенного алгоритма TDAG. Особенностью данного турнира является то, что в нем не налагается дополнительных ограничений на порядок игр. При этом все игры должны быть сыграны за минимальное количество раундов.

Различные методы составления турнирных сеток для шахматных круговых турниров были опубликованы ещё в конце 19 века Шуригом, Бергером и др. Для составления данного и далее других алгоритмов круговых турниров воспользуемся модификацией графической интерпретации турнира по Лукасу [19]. Предлагаемая нами графическая интерпретация основана на перемещении и взаимодействии воображаемых цепочек, звенья которых помечены цифрами. Цифры соответствуют номерам команд. Взаимное положение звеньев, когда звенья одной или разных цепочек находятся друг над другом, соответствует игре той пары команд, чьи номера записаны на данных звеньях.

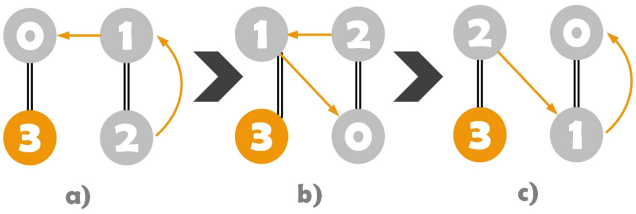

Рис. 4. Графическая иллюстрация построения алгоритма перечисления игр 4-х команд в круговом турнире без дополнительных ограничений

Пример составления турнира 4-х команд данным методом показан на рис. 4. Имеется две цепочки. Первая цепочка включает команды 0, 1, 2. Вторая цепочка состоит из одной команды 3. Для формирования игр первого раунда цепочка должна быть уложена, как показано на рис. 4a, а именно половина звеньев (0-1) – в верхнем ряду, вторая половина (2) – в нижнем ряду. Звено 3 (с максимальным номером) постоянно занимает положение в левой части нижнего ряда. Лежащие друг над другом звенья (0 и 3; 1 и 2) образуют игры первого раунда. Для формирования игр второго раунда передвинем звенья цепи так, чтобы звено с минимальным номером заняло положение звена с максимальным номером, а за ним переместились связанные звенья. Тогда цепь займет положение рис. 4b. Лежащие друг над другом звенья (1 и 3; 0 и 2) образуют игры второго раунда. Повторим перекладку элементов цепи описанным методом еще раз. Получим третий раунд игр в парах 2 и 3, 0 и 1 (рис. 4c). Еще одна перекладка вернет цепь в первоначальное положение, поэтому 3 раунд игр является последним.

Для длинных цепочек укладка выполняется аналогично. Например, для шести команд имеем укладку: верхний ряд слева-направо 0-1-2, нижний ряд справа-налево 3-4. Одиночный элемент 5 – в крайней левой позиции нижнего ряда. Для цепочек с нечетным числом элементов крайняя левая позиция нижнего ряда остается незаполненной, используется одна сплошная цепь.

Алгоритм build() по описанной графической интерпретации строится таким образом, чтобы для каждого раунда, начиная с первого, перечислить все пары раунда на рис. 4 слева направо. Такой алгоритм записывается, как показано ниже. Сопоставление рис. 2a и 4 позволяет убедиться, что формируется необходимая последовательность игр.

Листинг 5. Алгоритм build() турнира без ограничений
начало
для r от 1 до M + (M mod 2) – 1 выполнять
  если M mod 2 = 0 то выполнить game(r-1,M-1)
  для p от 1 до (M + (M mod 2))/2 – 1 выполнять
    i := (r+p-1)mod(M+(M mod 2)–1)
    j := (M+(M mod 2)-p+r-2)mod(M+(M mod 2)-1)
    если i > j то обменять значения переменных i,j
    выполнить game(i,j)
конец

В данном алгоритме внешний цикл соответствует раундам, а внутренний – парам команд в раунде слева-направо по рис. 4. Поправка M mod 2 позволяет обобщить алгоритм для четного и нечетного числа команд (a mod b – бинарная операция взятия остатка от деления a на b). Перекладка элементов цепи моделируется сложением либо вычитанием по модулю M+(M mod 2)-1, где M – количество команд. Обмен значений i и j требуется, так как мы условились, что в game() первым передается меньший номер, а вторым – больший.

Утверждение 1. Минимальное время в раундах R(M) выполнения турнира без дополнительных ограничений для M команд $M \geq 2$, $M \in \boldsymbol{N}$ – R(M)=M-1, если M – четное – R(M)=M; если M – нечетное. Ускорение при параллельной организации игр – S(M)=M/2, если M – четное; S(M)=(M-1)/2, если M – нечетное.

Доказательство. Очевидно следует из описанного метода построения алгоритма: число раундов турнира определяется числом итераций цикла по r в алгоритме. Ускорение находится как отношение количества игр турнира к числу раундов S(M)=M(M-1)/2R(M).

V. Простой сортирующий турнир

Пусть каждая из команд 0, 1, 2, … , M-1 хранит некоторое произвольное число, а игра заключается в том, чтобы команды предъявили друг другу хранимые числа и обменялись числами, если команда с меньшим номером хранила до игры большее число. Тогда очевидно, что сортировка хранимых чисел по возрастанию (номера команд перечисляются также по возрастанию) выполняется следующим образом: команда 1 играет с командой 0; команда 2 играет с командами 0, 1; команда 3 играет с командами 0, 1, 2; команда M-1 играет с командами 0, 1, 2, … , M-2. Такая последовательность игр кодируется приведенным ниже алгоритмом.

Листинг 6. Алгоритм build() сортирующего турнира

```
начало
для i от 1 до M-1, для j от 0 до j<i
└  выполнять game(j,i)
конец
```

Утверждение 2. Минимальное время в раундах R(M) выполнения простого сортирующего турнира M команд $M \geq 2$, $M \in \boldsymbol{N}$ – R(M)=2M-3. Ускорение при параллельной организации игр – $S(M)=(M^2-M)/(4M-6)$.

Доказательство. Рассмотрим графическую интерпретацию турнира, основанную на перекладывании воображаемой цепи со звеньями, помеченными номерами команд в простом сортирующем турнире (рис. 5).

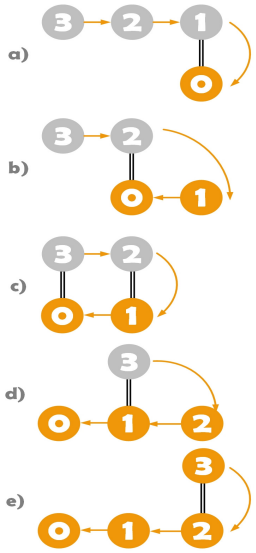

Рис. 5. Графическая иллюстрация построения алгоритма перечисления игр 4-х команд в простом сортирующем турнире

Начальное положение цепи, первый раунд: верхний ряд – звенья 3-2-1, нижний ряд справа — звено 0. Играют команды 0 и 1 (рис. 5a). Далее звенья верхнего ряда «скользят» по звеньям нижнего ряда. Достигая края цепочки, звенья переходят в нижний ряд. Второй раунд: верхний ряд – звенья 3-2, нижний ряд – звенья 0-1. Звено 1 перешло в нижний ряд. Играют команды 0 и 2 (рис. 5b). Третий раунд: верхний ряд – звенья 3-2, нижний ряд – звенья 0-1. Ряды оказываются друг над другом. Играют команды 0 и 3, 1 и 2 (рис. 5c). Четвертый раунд: звено 2 переходит в нижний ряд, звено 3 оказывается над звеном 1. Играют команды 1 и 3 (рис. 4d). Пятый раунд: звено 3 проходит над звеном 2, играют команды 2 и 3 (рис.5e).

Анализируя описанное выше перемещение воображаемой цепи можно заметить, что от начального положения (рис. 5a) до положения, в котором хвост цепи длины M совмещается с её началом, проходит M-1 раундов (так как над начальным звеном цепи должны пройти все остальные звенья). Затем, чтобы завершить перекладку, потребуется еще M-2 раунда (так как конечное звено цепи должно последовательно пройти над всеми остальными элементами цепи, кроме начального). В сумме имеем R(M)=2M-3, а зная, что общее число игр равно M(M-1)/2, получаем ускорение $S(M)=M(M-1)/2R(M)=(M^2-M)/(4M-6)$.

VI. Оптимизированный сортирующий турнир

Идея оптимизации алгоритма из раздела V основана на двух наблюдениях. Во-первых, можно разделить массив на части, по отдельности отсортировать каждую из частей, затем объединить части в общую последовательность. Этим достигается то, что на начальных этапах турнира одновременно играется больше игр, чем в простом сортирующем турнире. Рассмотрим данный эффект по рис. 6. Разобьем цепь команд на две подцепи: 0-1, 2-3. Тогда в первом раунде можно провести 2 игры в парах команд 0 и 1, 2 и 3

(рис. 6a). Во втором раунде начнем слияние двух отсортированных подцепей: в верхнем ряду голова «большей» подцепи начинает взаимодействовать с хвостом «меньшей» подцепи, проводится игра команд 0 и 3 (рис. 6b). В третьем раунде верхняя подцепь смещается дальше вправо, проводятся игры команд 0 и 2, 1 и 3 (рис. 6c). В четвертом раунде цепи завершают взаимодействие игрой команд 1 и 2 (рис. 6d). Теперь две подцепи можно объединить в одну отсортированную цепь. Перечисление игр в приведенной последовательности показано на рис. 2c, где видно, что удалось уменьшить число раундов по сравнению с простым сортирующим турниром рис. 2b.

Рис. 6. Графическая иллюстрация построения алгоритма перечисления игр 4-х команд в оптимизированном сортирующем турнире

Во-вторых, наблюдая процесс слияния более длинных подцепей, например, длиной по четыре звена, можно заметить, что хвост «большей» цепи и голова «меньшей» цепи в начальный момент слияния не задействованы. Поэтому они могут участвовать в других слияниях. Иными словами, можно начинать слияние не полностью сформированных цепей, тем самым дополнительно увеличивая параллелизм.

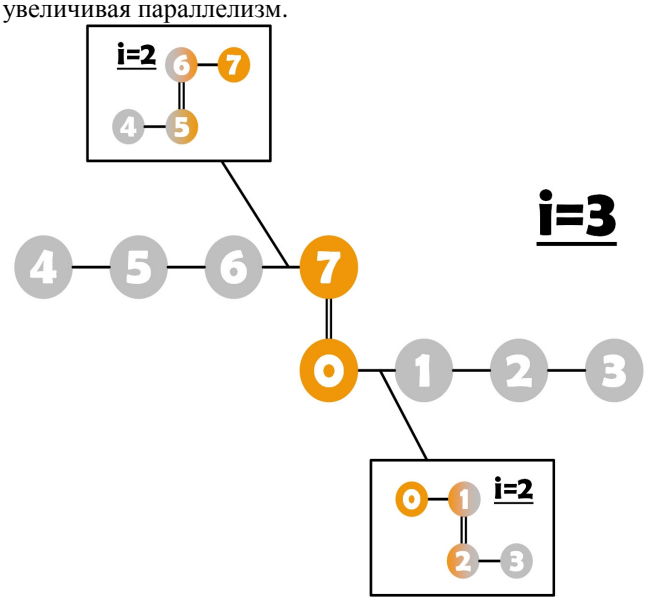

Рис. 7. Иллюстрация процесса слияния не полностью сформированных подцепей при построении оптимизированного сортирующего турнира

Поясним это по рис. 7. В момент начала слияния подцепей 4-5-6-7 и 0-1-2-3, когда проводится игра команд 0 и 7, одновременно может завершаться формирование самих подцепей: может проводиться игра команд 1 и 2 в подцепи 0-1-2-3, а также может проводиться игра команд 5 и 6 в подцепи 4-5-6-7.

Из приведенного описания следует рекурсивная процедура перечисления игр оптимизированного сортирующего турнира. Вначале нужно разделить исходную последовательность номеров команд на две по возможности равные подпоследовательности. Затем нужно рекурсивно сформировать списки игр для каждой из подпоследовательностей. В итоге получим две цепочки из номеров команд, которые объединим, как описано выше. Точный алгоритм этой процедуры (gen_optim) записан в листинге 7. Вызов процедуры gen_optim из процедуры build() обобщенного алгоритма TDAG имеет вид gen_optim(0, M-1).

Листинг 7. Алгоритм процедуры gen_optim

```
вход: range_begin, range_end
      в вызове gen_optim(range_begin, range_end)
начало
если range_end - range_begin = 0 то
⌊ возврат в точку вызова gen_optim
если range_end - range_begin = 1 то
| выполнить game(range_begin, range_end)
⌊ возврат в точку вызова gen_optim
range_begin^0 := range_begin
range_end^0 := range_begin+(range_end-range_begin+1)/2-1
range_begin^1 := range_end^0+1
range_end^1 := range_end
вызвать рекурсивно gen_optim(range_begin^0, range_end^0)
вызвать рекурсивно gen_optim(range_begin^1, range_end^1)
для i от range_end^1 до range_begin^1
| для j от range_begin^0 до range_end^0
⌊ ⌊ выполнять game(j,i)
конец
```

Утверждение 3. Минимальное время в раундах R(M) выполнения оптимизированного сортирующего турнира M команд $M=2^k$, $k \geq 2$, $k \in \boldsymbol{N}$ – $R(M)=3/2M-2$. Ускорение при параллельной организации игр – $S(M)=(M^2-M)/(3M-4)$.

Доказательство. Пусть алгоритм вызывается для значений $M=2^k$, $k \geq 2$, $k \in \boldsymbol{N}$. Будем отсчитывать уровни рекурсивного вызова процедуры gen_optim следующим образом. Самый глубокий уровень, на котором обрабатывается диапазон из 2-х чисел, – уровень 1; уровень, на котором обрабатывается диапазон из 4-х чисел, — уровень 2; уровень, на котором обрабатывается диапазон из 8-и чисел, — уровень 3; и так далее до вызова процедуры gen_optim из алгоритма build. Можно заметить, что начиная с уровня $i \geq 3$ некоторые игры, формируемые на уровне i, будут выполняться одновременно с играми, формируемыми нижележащим уровнем i-1. Определим сколько раундов игр будет сформировано именно на уровне i, без учета игр уровня i-1. Пусть m – длина цепочки номеров команд на некотором уровне рекурсии $i \geq 3$. Тогда от начала формирования новых раундов до выравнивания верхней и нижней подцепочек на уровне i будет сформировано m/4+1 новых раундов. После этого, при дальнейшем движении подцепочек на уровне i будет сформировано m/2-1 раундов. То есть уровень рекурсии $i=\log_2 m$ для диапазона длины m даст приращение количества раундов турнира $m/2+m/4 = \frac{3}{4}\, m = \frac{3}{4}\, 2^i$. Это означает, что общее число раундов R(M) равно 1 плюс

сумма ¾ 2^i для i от 2 до $\log_2 M$. Суммируя члены геометрической прогрессии получаем $R(M)=3/2M–2$. Тогда ускорение определяется как $S(M)=M(M-1)/2R(M)=(M^2-M)/(3M-4)$.

VII. Заключение

В работе предложен общий метод синтеза алгоритмических скелетов для асинхронного параллельного управления вычислениями по принципу кругового спортивного турнира «каждый с каждым». Метод проиллюстрирован тремя примерами синтеза алгоритмических скелетов: круговым турниром без наложения дополнительных ограничений на порядок игр, простым сортирующим турниром и оптимизированным сортирующим турниром. В последних двух турнирах введены ограничения, позволяющие использовать их в задачах параллельной сортировки и для вычисления частоты использования слов в больших текстовых массивах. Для разработанных алгоритмических скелетов получены оценки времени параллельного выполнения и ускорения.

Предполагается дальнейшее экспериментальное исследование предложенного метода и построенных по нему алгоритмических скелетов для реализации многозадачных приложений обработки данных в гибридных облачных системах.

Библиография

[1] I. Raicu, I.T. Foster, and Y. Zhao, “Many-task computing for grids and supercomputers,” *2008 workshop on many-task computing on grids and supercomputers*. *IEEE*, 2008, pp. 1-11.

[2] E. Ayguadé, N. Copty, A. Duran, J. Hoeflinger and Y. Lin, “The design of OpenMP tasks,” *IEEE Transactions on Parallel and Distributed Systems,* 2009, vol. 20, no. 3, pp. 404-418.

[3] A. D. Robison, “Composable parallel patterns with intel cilk plus,” *Computing in Science & Engineering,* 2013, vol. 15, no. 2, pp. 66-71.

[4] M. Voss, R. Asenjo, J. Reinders, *Pro TBB: C++ Parallel Programming with Threading Building Blocks*. Apress, 2019.

[5] О. С. Заикин, М. А. Посыпкин, А. А. Семёнов, Н. П. Храпов, “Опыт организации добровольных вычислений на примере проектов OPTIMA@ home и SAT@ home.” *Вестник Нижегородского университета им. НИ Лобачевского,* 2012, vol. 5, no. 2.

[6] O. Sukhoroslov, S. Volkov, A. Afanasiev , “A web-based platform for publication and distributed execution of computing applications,” *14th International Symposium on Parallel and Distributed Computing*. *IEEE*, 2015, pp. 175-184.

[7] D. Thain, T. Tannenbaum, M. Livny, “Distributed Computing in Practice: The Condor Experience,” *Concurrency and Computation: Practice and Experience*, 2005, vol. 17, no. 2-4, pp. 323-356.

[8] В. В. Воеводин, Ю. А. Жолудев, С. И. Соболев, К. С. Стефанов, “Эволюция системы метакомпьютинга X-Com,” *Вестник Нижегородского университета им. НИ Лобачевского*, 2009, no. 4.

[9] J.C. Régin, M. Rezgui, A. Malapert , “Embarrassingly parallel search,” *International Conference on Principles and Practice of Constraint Programming*, 2013, pp. 596-610.

[10] И. В. Лазарев, О. В. Сухорослов, “Использование workflow-методологии для описания процесса распределенных вычислений,” *Труды Института системного анализа Российской академии наук 14,* 2005, pp. 26-70.

[11] M. Cole, “Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming,” *Parallel computing*, 2004, vol. 30, no. 3, pp. 389-406.

[12] H. González□Vélez, M. Leyton, “A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers,” *Software: Practice and Experience*, 2010, vol. 40 no.12, pp. 1135-1160.

[13] J. Dean,S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM,* 2008, vol. 51, no. 1, pp. 107-113.

[14] S. Goyal, “Public vs private vs hybrid vs community-cloud computing: a critical review,” *International Journal of Computer Network and Information Security*, 2014, vol. 6, no 3, pp. 20.

[15] C. Moretti, J. Bulosan, D. Thain, , P.J. Flynn, “All-pairs: an abstraction for data-intensive cloud computing,” *International Symposium on Parallel and Distributed Processing. IEEE,* 2008, pp. 1–11.

[16] W. Suksompong, “Scheduling asynchronous round-robin tournaments,” *Operations Research Letters*, 2016, vol. 44, no 1, pp. 96-100.

[17] S.V. Vostokin, I.V. Bobyleva, “Using the bag-of-tasks model with centralized storage for distributed sorting of large data array,” *CEUR Workshop Proceedings,* 2019, vol. 2416, pp. 199-203.

[18] S.V. Vostokin, I.V. Bobyleva, “Building an Algorithmic Skeleton for Block Data Processing on Enterprise Desktop Grids,” *Communications in Computer and Information Science,* 2019, vol. 1129 CCIS, pp. 678-689.

[19] E. Lucas, “Les jeux de demoiselles,” *Récréations Mathématiques (in French),* Paris: Gauthier-Villars, 1883, pp. 161–197.

Asynchronous round-robin tournament algorithms for many-task data processing applications

S.V. Vostokin, I.V. Bobyleva

Abstract — **The article discusses the constructing of tasks dependencies graphs for many-task applications that perform parallel asynchronous data processing on the principle of round-robin sport tournament. The following components of the technique are described: the family of round robin algorithms is defined in the form of an algorithmic skeleton; a method for synthesizing a graph of a round-robin tournament using the basic sequential algorithm is proposed; it is shown how to use the graphs for the implementation of parallel and asynchronous computing process. A graphical interpretation of the round-robin tournament procedures is considered, on which it is possible to build analytical estimates of the tournament time. The technique is illustrated by three examples of constructing specific tournament algorithms: the tournament without additional restrictions, the simple sorting tournament, and the optimized sorting tournament. We build the algorithms for constructing task dependencies graphs for these three tournaments. The principle of implementing asynchronous parallel computing according to the graphs are considered. Using the graphical interpretation, the number of rounds and the speedup was calculated for each of the listed tournaments. The technique is recommended for a wide class of parallel architectures including clusters, desktop grids, and hybrid clouds.**

Keywords— many-task computing, task parallelism, round-robin tournament, algorithmic skeleton.

REFERENCES

[1] I. Raicu, I.T. Foster, and Y. Zhao, “Many-task computing for grids and supercomputers,” 2008 workshop on many-task computing on grids and supercomputers. IEEE, 2008, pp. 1-11.

[2] E. Ayguadé, N. Copty, A. Duran, J. Hoeflinger and Y. Lin, “The design of OpenMP tasks,” IEEE Transactions on Parallel and Distributed Systems, 2009, vol. 20, no. 3, pp. 404-418.

[3] A. D. Robison, “Composable parallel patterns with intel cilk plus ,” Computing in Science & Engineering, 2013, vol. 15, no. 2, pp. 66-71.

[4] M. Voss, R. Asenjo, J. Reinders, Pro TBB: C++ Parallel Programming with Threading Building Blocks. Apress, 2019.

[5] O. S. Zaikin, M. A. Posyipkin, A. A. Semënov, N. P. Hrapov, “Opyit organizatsii dobrovolnyih vyichisleniy na primere proektov OPTIMA@ home i SAT@ home.” Vestnik Nijegorodskogo universiteta im. NI Lobachevskogo, 2012, vol. 5, no. 2.

[6] O. Sukhoroslov, S. Volkov, A. Afanasiev , “A web-based platform for publication and distributed execution of computing applications,” 14th International Symposium on Parallel and Distributed Computing. IEEE, 2015, pp. 175-184.

[7] D. Thain, T. Tannenbaum, M. Livny, “Distributed Computing in Practice: The Condor Experience,” Concurrency and Computation: Practice and Experience, 2005, vol. 17, no. 2-4, pp. 323-356.

[8] V. V. Voevodin, YU. A. Joludev, S. I. Sobolev, K. S. Stefanov, “Evolyutsiya sistemyi metakompyutinga X-Com,” Vestnik Nijegorodskogo universiteta im. NI Lobachevskogo, 2009, no. 4.

[9] J.C. Régin, M. Rezgui, A. Malapert , “Embarrassingly parallel search,” International Conference on Principles and Practice of Constraint Programming, 2013, pp. 596-610.

[10] I. V. Lazarev, O. V. Suhoroslov, “Ispolzovanie workflow-metodologii dlya opisaniya protsessa raspredelennyih vyichisleniy,” Trudyi Instituta sistemnogo analiza Rossiyskoy akademii nauk 14, 2005, pp. 26-70.

[11] M. Cole, “Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming,” Parallel computing, 2004, vol. 30, no. 3, pp. 389-406.

[12] H. González□Vélez, M. Leyton, “A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers,” Software: Practice and Experience, 2010, vol. 40 no.12, pp. 1135-1160.

[13] J. Dean,S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, 2008, vol. 51, no. 1, pp. 107-113.

[14] S. Goyal, “Public vs private vs hybrid vs community-cloud computing: a critical review,” International Journal of Computer Network and Information Security, 2014, vol. 6, no 3, pp. 20.

[15] C. Moretti, J. Bulosan, D. Thain, , P.J. Flynn, “All-pairs: an abstraction for data-intensive cloud computing,” International Symposium on Parallel and Distributed Processing. IEEE, 2008, pp. 1–11.

[16] W. Suksompong, “Scheduling asynchronous round-robin tournaments,” Operations Research Letters, 2016, vol. 44, no 1, pp. 96-100.

[17] S.V. Vostokin, I.V. Bobyleva, “Using the bag-of-tasks model with centralized storage for distributed sorting of large data array,” CEUR Workshop Proceedings, 2019, vol. 2416, pp. 199-203.

[18] S.V. Vostokin, I.V. Bobyleva, “Building an Algorithmic Skeleton for Block Data Processing on Enterprise Desktop Grids,” Communications in Computer and Information Science, 2019, vol. 1129 CCIS, pp. 678-689.

[19] E. Lucas, “Les jeux de demoiselles,” Récréations Mathématiques (in French), Paris: Gauthier-Villars, 1883, pp. 161–197.