УДК 004.722

КУЛАКОВ Ю.А.,
ГОРОБЕЦ А.Н.

АЛГОРИТМ СОЗДАНИЯ ТОПОЛОГИИ БИНОМИАЛЬНЫЙ ГРАФ В ПОЛНОСВЯЗНОЙ СИСТЕМЕ

В данной работе предложен алгоритм формирования топологии биномиальный граф в полносвязной системе. Параллельно выполняющиеся части алгоритма формируют кольцевую топологию, а затем заполняют в каждом узле соответствующие списки, в соответствии с правилами, которые описывают данную топологию.

In this paper we described an algorithm for the constructing binomial graph topology in a fully connected system. Parallel execution of the algorithm parts form a ring topology, and then fill in lists of links to nodes, in accordance with rules that describe binomial graph topology.

Создание топологии биномиальный граф.

Для систем, которые состоят из набора параллельно выполняющихся процессов, важным параметром является задержка при обмене информацией. Обмен сообщениями между процессами происходит при помощи исполнительной среды. Конфигурация данной среды должна оптимальным образом распределять нагрузку на узлы системы. В качестве топологии, которая будет снижать нагрузку на пересылки, была предложена топология биномиальный граф [2]. Для её построения, в исходном алгоритме [7] в качестве базовой топологии указано дерево процессов. Выбор такой топологии не случаен – такая топология на самом деле очень легка для создания – процессы, решающие сложные задачи, или изначально разработанные как многопоточные создают новые потоки, запускают на выполнение другие задачи. В процессе работы выстраивается иерархия процессов, которая и представляет собой дерево[1].

Подобное решение также вызвано тем, что при инициации параллельной исполнительной среды, для которой будет строиться данная топология, системный загрузчик изначально создает дерево процессов, которое содержит необходимую для данного алгоритма информацию. У каждого процесса в дереве есть идентификатор своего родительского процесса (как указатель или определенное уникальное значение), идентификаторы дочерних процессов.

В топологии кольцо каждый узел знает идентификатор узла, который следует за ним, и идентификатор предыдущего узла. Такая топология равномерно распределяет нагрузку, связанную с передачей сообщений между узлами. Помимо этого, в каждом элементе кольца формируются два массива идентификаторов узлов. Данные узлы расположены на расстоянии степеней двойки от данного узла, в направлении по часовой и против часовой стрелки.

Алгоритм построения биномиального графа из дерева

Для создания выше описанной топологии используется алгоритм, в котором можно выделить два под алгоритма: алгоритм создания кольца процессов из дерева процессов, и алгоритм создания биномиального графа из кольца процессов [3, 6].

Алгоритм построения кольца процессов из дерева процессов представлен в работе [7].

Основной идеей первого алгоритма является создание цепочек из родителей и их первых потомков, и соединение этих цепочек в кольцо.

Создание цепочек происходит с помощью правил 1 и 2 из алгоритма 1. Правило 1 может быть использовано любым процессом, у которого есть хотя бы один процесс-наследник. При выполнении его, первый процесс-наследник записывается как следующий процесс в кольце. Посылая сообщение со своим идентификатором, процесс устанавливает себя как предыдущий элемент кольца для своего первого дочернего процесса при помощи правила 2.

Следует отметить, что в результате получится цепочки, с «листом» дерева на одном конце, и каждый лист дерева будет являться конечной точкой какой-либо цепочки.

Объединение цепочек в кольцо сводиться к поиску для каждого «листа» первого свободного процесса для установки его как следующего в кольце [5]. Правило 3 может быть выполнено для листа, и в результате будет посла-

но сообщение Info в родительский процесс. Правило 4 описывает каким образом поток должен реагировать на получение сообщения Info.

Во втором алгоритме каждый узел регулярно посылает идентификаторы своих соседей друг другу (правило 1). Когда процесс получил идентификатор процесса на расстоянии 2^i, он сохраняет его в соответствующем массиве, и пересылает его на расстояние 2^i в обоих направлениях.

Построение топологии биномиальный граф в среде со случайной конфигурацией

Для реализации алгоритма в системе, в которой не создано дерево процессов программы, необходимо видоизменение связей среды для создания дерева. С учетом того, что конфигурация дерева не имеет кардинального значения для алгоритмов, описанных выше, алгоритм построения топологии дерева сводится к выбору корневого узла и пошагового формирования веток дерева.

Для определения корневого узла будем исходить из того, что алгоритм построения кольца из древовидной структуры выполняется параллельно для каждой ветки, соответственно выбор корневым узлом узла с наибольшей степенью положительно скажется на конфигурации дерева.

Алгоритм создания дерева будет выглядеть следующим образом:

1) Поиск узла с наибольшей степенью и установка его как корневого

2) Установка всех связанных с текущим узлом узлов как дочерних узлов в дереве.

3) Всем дочерним узлам текущий узел указывается как родительский узел в дереве. Все дочерние узлы добавляются в список.

4) Пока в списке есть хоть один узел:

– Первый узел в списке устанавливается как текущий;

– Все узлы, которые непосредственно связаны с текущим, не отмеченные как обработанные и не находятся в списке указываются как дочерние для текущего;

– Всем дочерним узлам текущего узла текущий узел указывается как родительский в дереве. Все дочерние узлы добавляются в список;

– Текущий узел отмечают как обработанный и удаляют из списка.

Основным преимуществом данного алгоритма является отсутствие комплексного преобразования сети. Дерево формируется на базе уже существующих связей.

Алгоритм создания топологии биномиальный граф для полносвязной системы

Алгоритмы, рассмотренные выше, являются избыточными для сильно связных систем. Особенно явно это проявляется в полносвязных системах, так как необходимые для биномиального графа связи уже присутствуют, и создание промежуточных топологий с переконфигурированием системы будет явно излишним. Для подобных систем необходим алгоритм, который выделит необходимые связи из уже имеющихся, и создаст соответствующие списки соседей слева и справа для каждого из узлов. Также алгоритм должен выполняться параллельно на каждом из узлов, что ускорит построение. Топологии биномиальный граф.

В общем случае, для каждого из N узлов необходимо создать два списка – CW и CCW, которые содержат узлы, находящиеся на расстоянии $\{(i+1) \bmod n, (i+2) \bmod n, \dots, (i+2^k) \bmod n \mid 2^k <= n\}$ и $\{(n-i+1) \bmod n, (n-i+2) \bmod n, \dots, (n-i+2^k) \bmod n \mid 2^k <= n\}$ соответственно[4].

Связи узлов на расстоянии 1 формируют кольцевую топологию, конкретная реализация которой не имеет особого значения для формирования остальной части списков.

Каждый конкретный узел может сформировать связь на расстоянии 2^i для любых двух узлов на расстоянии 2^{i-1} слева и справа. Узлы получают сообщение, в котором указан идентификатор узла и расстояние до него, и устанавливают соответствующие значения в своих списках CW и CCW.

Алгоритм построения топологии биномиальный граф в полносвязной системе будет состоять из двух частей. Первая часть алгоритма случайным образом будет формировать кольцевую топологию, выбирая соседние узлы на расстоянии 1. Вторая часть алгоритма будет формировать соответствующие списки CW и CCW для каждого из узлов.

Исходными данными для алгоритма является список идентификаторов доступных узлов из текущего узла. Алгоритм выполняется параллельно на каждом из узлов топологии.

Алгоритм построения топологии биномиальный граф в полносвязной сети.

1) Послать в случайный узел запрос наличия «соседа справа». При получении в ответе своего идентификатора установить узел, в который отправлялся запрос, как текущий «сосед слева» и добавить его идентификатор в список CW.

2) Ожидать запроса наличия «соседа справа». При получении такого запроса, если «сосед справа» не определен, установить узел, который прислал запрос, как текущий «сосед справа», добавить его идентификатор в список CCW и отправить идентификатор текущего «соседа справа» в ответе.

3) Отправить идентификатор текущего «соседа справа» в узел с идентификатором указанным в качестве текущего «соседа слева».

Отправить идентификатор текущего «соседа слева» в узел с идентификатором указанным в качестве текущего «соседа справа».

4) Ожидать получения идентификатора узла от текущего «соседа справа». При получении, если полученный идентификатор не содержится в списке CW и не является текущим «соседом слева», то установить полученный идентификатор как текущий «сосед слева» и добавить его в список CW. Иначе установить сигнал завершения.

Ожидать получения идентификатора узла от текущего «соседа слева». При получении, если полученный идентификатор не содержится в списке CCW и не является текущим «соседом справа», то установить полученный идентификатор как текущий «сосед справа» и добавить его в список CCW. Иначе установить сигнал завершения.

5) Если не установлен сигнал завершения, перейти к пункту 3.

Пункты 1 и 2 являются первой частью алгоритма. Они выполняются параллельно на всех узлах, и случайным образом формируют кольцо узлов. Пункты 3 и 4 выполняются в цикле, формируя списки CW и CCW узлов, до тех пор, пока не будет достигнуто максимальное значение расстояния ({(i+1) mod n, (i+2) mod n, … , (i+2^k) mod n | 2^k <= n} и {(n-i+1) mod n, (n-i+2) mod n, … , (n-i+2^k) mod n | 2^k <= n} для соответствующих списков). Это условие выполнится, когда для каждого из узлов будут достигнуты наиболее удаленные в соответствии со структурой топологии узлы.

Данный алгоритм, в полносвязной системе будет формировать топологию биномиальный граф более оптимальным способом. Это достигается за счет отсутствия промежуточных преобразований топологии, параллельного выполнения частей алгоритма на каждом из узлов.

Пример работы алгоритма

Для создания выше описанной топологии в полносвязной системе на 8 узлов будут произведены следующие действия. Каждый узел будет иметь уникальный идентификатор и список связей со всеми узлами в системе. На рисунке 1 показано исходное состояние системы.

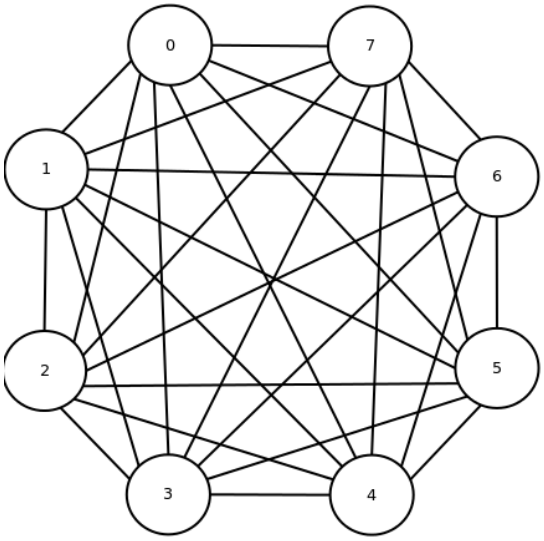

Рис. 1. Исходное состояние системы

На первом этапе проводится формирование кольца узлов случайным образом. Результат выполнения первого этапа представлен на рисунке 2.

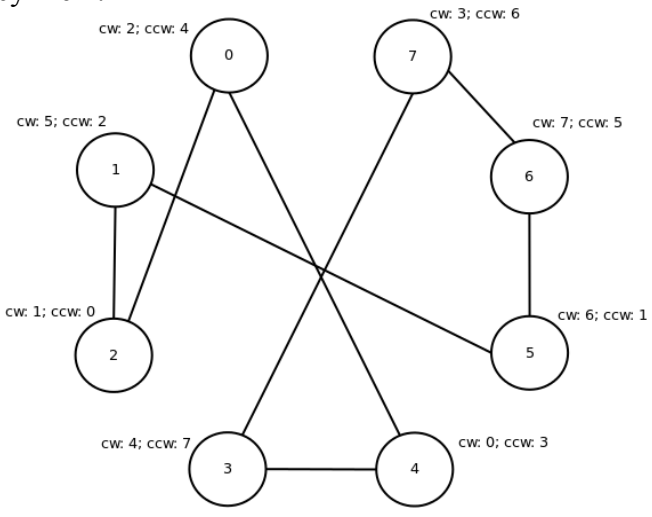

Рис. 2. Результат работы первого этапа алгоритма

Как видно из рисунка, у каждого из узлов начали формироваться списки соседних узлов

по часовой стрелке (CW) и против часовой стрелки (CCW).

На следующем этапе продолжается формирование данных списков. На первом шаге в список добавляются узлы, которые находятся на расстоянии 2. Как описывалось выше, это происходит в результате отправки текущего узла из списка соседних узлов слева в узел, который является текущим в списке соседних узлов справа, и наоборот. Результат первого шага второй части алгоритма представлен на рисунке 3.

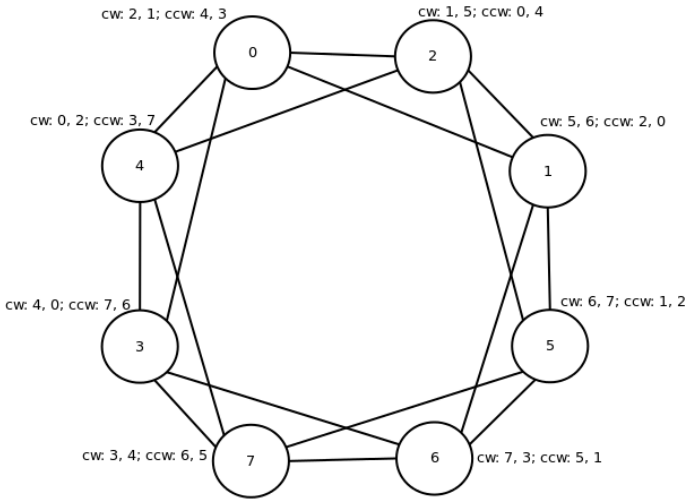

Рис. 3. Результат первого шага второй части алгоритма

Условие завершения работы второго этапа алгоритма не выполняется, следовательно, производится следующий шаг второй части алгоритма. Результат его работы представлен на рисунке 4.

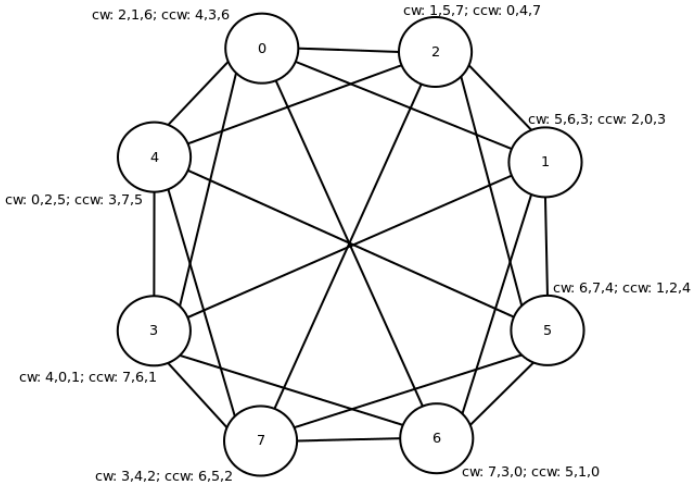

Рис. 4. Результат работы второго шага второй части алгоритма

Как видно из рисунка 4, текущий узел из списка CW является также текущим узлом в списке CCW, что является условием окончания работы второй части алгоритма.

В результате работы алгоритма в каждом из узлов было сформировано два списка узлов, в которых содержатся узлы, находящиеся на расстоянии 2^k, что собственно формирует список связей для топологии биномиальный граф.

Список литературы

1. Sutter, H. Software and concurrency revolution / H. Sutter, J. Laurus // Queue. – New York: ACM, 2005. – Vol. 3, № 7. – P. 54-62. – ISSN 1542-7730.
2. T. Angskun, G. Bosilca, and J. Dongarra. Binomial graph: A scalable and fault-tolerant logical network topology. – Heidelberg: Parallel and Distributed Processing and Applications, ISPA 2007 – Vol. 4742/2007 of Lecture Notes in Computer Science – P. 471–482. Springer Berlin /.
3. Lemarinier P. Constructing Resiliant Communication Infrastructure for Runtime Environments / Pierre Lemarinier George Bosilca, Camille Coti, Thomas Herault, and Jack Dongarra (University of Tennessee Knoxville, Universite Paris Sud, INRIA) – 2009 P. 4-5.
4. Bacon, D.F. Compiler transformations for high performance computing / D.F. Bacon, S.L. Graham, O.J. Sharp // ACM Computing Surveys. – New York: ACM, 1994. – Vol. 26, № 4. – P. 345-420. – ISSN 0360-0300.
5. Hendren, L. J. Parallelizing programs with recursive data structures / L. J. Hendren, A. Nicolau // IEEE Transactions on Parallel and Distributed Systems. – Piscataway, New Jersey, USA: IEEE Press, 1990. – Vol. 1, № 1. – P. 35-47. – ISSN 1045-9219.
6. Herault T. A model for large scale self-stabilization. / Thomas Herault, Pierre Lemarinier, Olivier Peres,. Laurence Pilard, Joffroy Beauquier // IEEE International on Parallel and Distributed Processing Symposium, march 2007 – P. 1–10, – IPDPS 2007.
7. Bosilca, G. Constructing Resiliant Communication Infrastructure for Runtime Environments / George Bosilca, Camille Coti, Thomas Herault, Pierre Lemarinier, Jack Dongarra // University of Tennessee Knoxville University of Tennessee Knoxville, Universite Paris Sud, INRIA