PARALLEL CALCULATIONS IN THE SYSTEMS OF REAL TIME AND BATCH OPERATION

Nazarov S.
Dr.Sci.Tech., professor, chief specialist of the Moscow research television institute

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ И ПАКЕТНОЙ ОБРАБОТКИ

Назаров С.
Д-р техн. наук, профессор, главный специалист
Московского научно-исследовательского телевизионного института

Abstract

The Onboard Computing Systems (OCS) find broad application in the aircraft and space equipment and also in land and water mobile objects. Development of the onboard equipment is characterized by continuous increase in number of solvable tasks, increase in their complexity, expansion of intellectual and adaptive opportunities. For the period of the solution of the tasks assigned to OCS tight temporary restrictions are imposed. Performance of these of the requirement results in need of the organization of parallel computation processes. The similar situation develops at the organization of computation process supercomputers. Even at their high performance the tasks intended for them, can take considerable time and be carried out, as a rule, repeatedly. In this regard even within the most powerful supercomputers optimization of computation process for the purpose of rational expenditure of computing resources at the simultaneous requirement of performance of a certain set of tasks (package) in the set temporary restrictions is required. In article the approach to the organization of computation process based on methods of network planning and management, the theory of counts and schedules is offered.

Аннотация

Бортовые вычислительные системы (БВС) находят широкое применение в авиационной и космической технике, а также в наземных и водных подвижных объектах. Развитие бортового оборудования характеризуется постоянным увеличением числа решаемых задач, повышением их сложности, расширением интеллектуальных и адаптивных возможностей. На время решения задач, возлагаемых на БВС, накладываются жесткие временные ограничения. Выполнение этих требования приводит к необходимости организации параллельных вычислительных процессов. Подобная ситуация складывается при организации вычислительного процесса суперкомпьютерах. Даже при высокой их производительности предназначенные для них задачи, могут занимать значительное время и выполняться, как правило, неоднократно. В связи с этим даже в рамках мощнейших суперкомпьютеров требуется оптимизация вычислительного процесса с целью рационального расходования вычислительных ресурсов при одновременном требовании выполнения определенного набора задач (пакета) в заданные временные ограничения. В статье предлагается подход к организации вычислительного процесса, основанный на методах сетевого планирования и управления, теории графов и расписаний.

Keywords: multiprocessor system, information and connected tasks, tier-parallel form, parallel computation process.

Ключевые слова: многопроцессорная система, информационно-связанные задачи, ярусно-параллельная форма, параллельный вычислительный процесс.

1. Параллельные вычислительные процессы в системах реального времени

1.1. Постановка задачи

Пусть задана структура приложения, состоящего из некоторого множества информационно-связанных частей (задач) с известным (ожидаемым) временем выполнения каждой задачи. Предполагается, что это время определяется элементарным вычислителем (процессором или ядром) многопроцессорной бортовой вычислительной системы. Структуру подлежащего выполнению приложения удобно представить графом, например, как это показано на рис. 1.1, который будем далее использовать для иллюстрации решения поставленной задачи:

$G = \{< z_i, t_i > | i = 1, \dots, M \}$, где z_i – номер задачи (вершины) в графе (первая цифра), t_i – время решения задачи (вторая цифра в вершине графа), M - количество задач в пакете G.

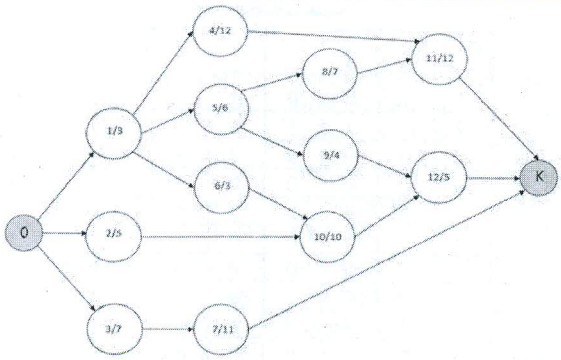

Рис. 1.1. Граф решаемых задач

В нашем примере вычисления начинаются в вершине О и завершаются в вершине К (здесь вершины О и К – фиктивные, используются как исходная и завершающая точки процесса выполнения задач). Дуги графа представляют собой передачу результатов вычислений от задачи z_i к задаче z_j Длительность передачи является весом дуги, обозначим ее соответственно t_{ij} . Поскольку предполагается, что решение набора задач производится на мультипроцессоре и оперативная память является общей для всех процессоров, будем считать, что $t_{ij} = 0$.

Требуется организовать вычислительный процесс многопроцессорной (многоядерной) бортовой вычислительной системы таким образом, чтобы выполнить все задачи набора за минимально возможное в данной системе время, т.е. найти такое расписание выполнения задач по процессорам:

$Schedule_Z = \{z_i, t_{н(i)}, t_{з(i)}, n_i\}$, которое обеспечивает $min\ T_Z$ при заданном количестве вычислителей, где $t_{н(i)}$ – время начала выполнения задачи z_i, $t_{з(i)}$ – время завершения выполнения задачи z_i , n_i – номер процессора, выделенного для выполнения задачи z_i.

1.2. Решение задачи

1.2.1. Проверка возможности организации параллельного вычислительного процесса

Известно, что существующий в графе задач G критический путь определяет минимальный срок выполнения всех работ, т.е. в нашем случае минимальное время выполнения заданного набора задач. Для нахождения перечня вершин, образующих критический путь, и его длины можно использовать различные методы, в том числе метод линейного программирования.

Потенциальную параллельность выполнения заданного набора задач можно определить, преобразовав граф задач в ярусно-параллельную форму. Ярусно-параллельная формаграфа (ЯПФ) – деление вершин ориентированного ациклического графа на перенумерованные подмножества V_j такие, что, если дуга идет от вершины $v_l \in V_j$, к вершине $v_m \in V_k$,то обязательно $j < k$.

Каждое из множеств V_j называется ярусом ЯПФ, j – его номером, количество вершин $|V_j|$ в ярусе – его шириной. Количество ярусов в ЯПФ называется её высотой, а максимальная ширина её ярусов – шириной ЯПФ. Для ЯПФ графа алгоритма важным является тот факт, что операции, которым соответствуют вершины одного яруса, не зависят друг от друга (не находятся в отношении связи), и поэтому заведомо существует параллельная реализация алгоритма, в которой они могут быть выполнены параллельно на разных устройствах вычислительной системы. Поэтому ЯПФ графа алгоритма может быть использована для подготовки такой параллельной реализации алгоритма.

Минимальной высотой всех возможных ЯПФ графа является его критический путь. Построение ЯПФ с высотой, меньшей критического пути, невозможно. Если в составе яруса могут быть вершины, находящиеся в различных отношениях (например, параллельности или альтернативы, что типично для граф-схем параллельных алгоритмов), ярус называется сечением, а ЯПФ – множеством сечений. Наличие более одного отношения между вершинами сечения существенно усложняет большинство алгоритмов обработки.

Для получения ЯПФ предварительно необходимо построить матрицу смежности исходного графа. Матрица смежности – это квадратная матрица размерностью *(M+1)* × *(M+1)*, (где *M* – число вершин графа), однозначно представляющая его структуру. Обозначим ее как $A = |a_{ij}|$, где каждый элемент матрицы определяется следующим образом: $a_{ij} = 1$, если есть дуга (i, j), $a_{ij} = 0$, если нет дуги (i, j). В нашем примере матрица смежности будет иметь следующий вид, приведенный на рис. 1.2.

Алгоритм распределения модулей системы по уровням:

1. Находим в матрице нулевые строки. В нашем случае это только одна строка с номером 13.
2. Вершина с этим номером образует нулевой уровень ЯПФ.
3. Вычеркиваем столбцы с номерами найденных вершин. В нашем случае – столбец 13.
4. Находим в матрице нулевые строки (7, 11, 12). Это вершины 1-го уровня.
5. Вычеркиваем столбцы с номерами 7, 11, 12.
6. Находим в матрице нулевые строки (3, 4, 8, 9 и 10). Это вершины 2-го уровня.
7. Вычеркиваем столбцы с номерами найденных вершин.
8. Находим в матрице нулевые строки (2, 5, 6). Это вершины 3-го уровня.
9. Вычеркиваем столбцы с номерами 5, 6.
10. Вершина с номером 1 образует 4-й уровень.

номер вершины															
номер вершины		0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	5	0	0	0	0	0	0	0	0	1	1	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	12	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис. 1.2. Матрица смежности

Полученная таким образом ЯПФ заданного набора задач представлена на рис. 1.3. Алгоритм определения критического пути в ЯПФ графа приведен в [1]. В нашем случае это довольно просто сделать перебором всех возможных путей. Такой путь длиной в 28 единиц процессорного времени образует последовательность вершин $W_{кр} = \{1, 5, 8, 11\}$. Из рисунка следует, что высота ЯПФ равна четырем, минимальная ширина – $B_{мин} = 1$, а максимальная – $B_{макс} = 5$. Таким образом граф задач далеко не прямоугольный, что неудобно для организации параллельного вычислительного процесса. Визуально из рис. 1.3 понятно, что граф можно легко переформатировать, не меняя связей между вершинами, например, можно переместить вершину 3 на 4-й уровень. Что же касается вершины 2, то для ее перемещения на уровень 4, нужно проверить, не приведет ли этот шаг к задержке выполнения вершины 10. А если и задержит, то это не должно привести к увеличению длины критического пути.

Действительно, начало времени выполнения вершины 10 зависит от времени завершения вершин 2 и 6. Однако пока неизвестно, какая вычислительная производительность будет выделена на каждый ярус, ответить на этот вопрос невозможно. Поэтому окончательное расписание вычислительного процесса можно составить только после решения вопроса о количестве процессоров, выделенных для решения задач, представленных графом G.

Далее будем считать, что можно использовать некоторое количество процессоров в заданном диапазоне. Предварительно целесообразно определить минимальную вычислительную мощность $T_{мин}$, которая позволить реализовать заданный набор задач. Ее легко найти, просуммировав времена реализации всего набора задач:

$$T_{мин} = \sum_{i=1}^{M-1} t_i.$$

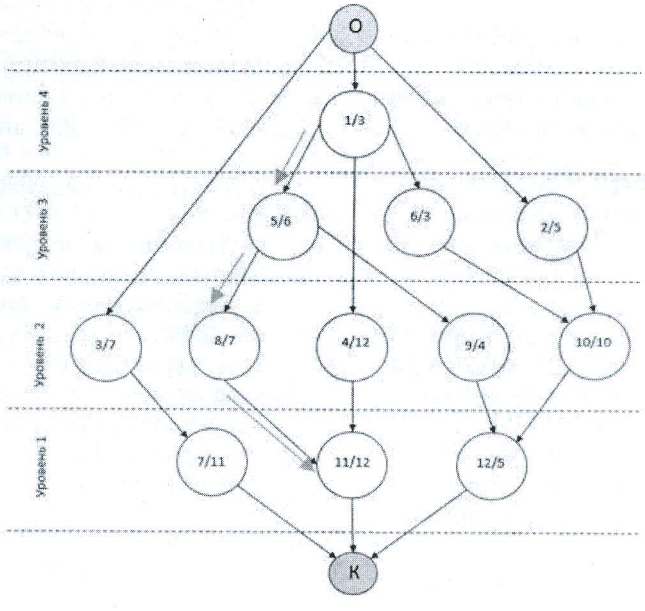

Рис. 1.3. Граф задач в ЯПФ

1.2.2. Выбор количества процессоров для параллельного вычислительного процесса

В нашем примере $T_{мин}$= 85. При этом вычислительная нагрузка по ярусам не равномерна и составляет следующие значения:

$$T_1 = 28, T_2 = 40, T_3 = 14, T_4 = 3 \quad (1.1)$$

Также неравномерна и ширина ярусов графа G:

$$B_1 = 3, B_2 = 5, B_3 = 3, B_4 = 1 \quad (1.2)$$

Очевидной является необходимость перестройки ЯПФ графа решаемых задач. Однако, как следует из рис. 1.3, вершины 2 и 3 можно передвинуть в четвертый ярус, а вершину 4 – в третий ярус, получив тем самым более прямоугольную ЯПФ (рис. 1.4). После такой реорганизации графа вычислительная нагрузка по ярусам составит следующие значения:

$$T_1 = 28, T_2 = 21, T_3 = 21, T_4 = 15 \quad (1.3)$$

Ширина ярусов становится одинаковой и равна трем. Для дальнейшего упрощения задачи можно поступить следующим образом. Выделим отдельный процессор на реализацию критического пути в графе. Его загрузка составит 28 единиц. Для выполнения оставшейся вычислительной работы, равной 57 единицам, потребуется как минимум 2 процессора. Однако при этом неизбежно превышение времени выполнения всего набора задач по сравнению с найденным временем критического пути. Кроме того, встает вопрос о том, какие из оставшихся задач (кроме входящих в критический путь) на какой процессор необходимо назначить. При этом желательно обеспечить равномерность загрузки процессоров.

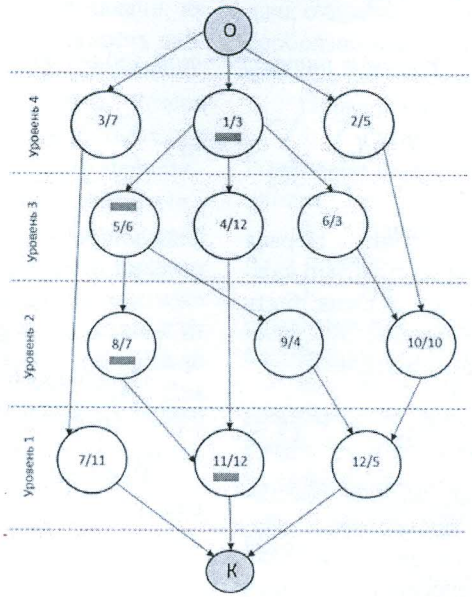

Рис. 1.4. Преобразованная ЯПФ

Рассмотрим возможную постановку задачи в данной ситуации. Размещение задач по процессорам удобно представить двудольным графом вида $G = (Z, P, E)$, где E – множество дуг, отображающих размещение множества задач. Требуется так распределить множество задач $Z' = \{z_2, z_3, z_4, z_6, z_7, z_9, z_{10}, z_{12}\}$ по процессорам P (второй и третий), чтобы разница в загрузке процессоров была минимальной и все задачи Z были выполнены. Если ограничений на размещение задач по процессорам нет, то в таком графе каждая вершина $z_i \in Z'$ соединена дугами со всеми вершинами P (рис. 1.5). Поставим в соответствие дугам E графа G множество переменных $X = \{x_{ij} | i \in I, j = 2,3\}$. Каждая переменная x_{ij} образуется по правилу x_{ij} = 1, если будет принято решение выполнять задачу z_i на втором процессоре p_2 (первый процессор выделен на задачи критического пути). В противном случае x_{ij} = 0 и задача выполняется на третьем процессоре.

Применительно к графовой интерпретации задача сводится к отысканию частичного графа $G_o = (Z, P, E_o)$, где $E_o \subseteq E$. При этом множество найденных дуг E_o определит значение булевых переменных $X_o = \{x_{ij} | i \in I, j = 2,3\}$.

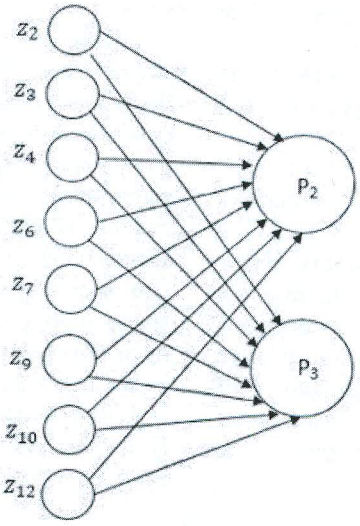

Рис. 1.5. Графовая интерпретация задачи о назначениях

$$C = (\sum_{i \in I} \sum_{j=2} t_{ij} * x_{i2} - \sum_{i \in I} \sum_{j=3} t_{ij} * x_{i3} \rightarrow min \quad (1.4)$$

Определим ограничения задачи. Первая группа ограничений связана с тем фактом, что каждая задача в результате решения должна быть назначена только на один вычислитель. Это условие записывается следующим соотношением:

$$\forall i \sum_{j=2}^{j=3} x_{ij} = 1. \quad (1.5)$$

Вторая группа ограничений должна обеспечить равномерную нагрузку процессоров. Поскольку общая загрузка второго и третьего процессоров составляет 57 единиц, целесообразно задач следующие ограничения по загрузке каждого процессора:

$$\sum_{i \in I} t_{ij} * x_{ij} \leq 28, j = 2,3. \quad (1.6)$$

Поскольку все переменные задачи – двоичные, вторая группа ограничений имеет следующий вид:

$$\forall x_{ij} \in \{0,1\} \quad (1.7)$$

Решение легко получить в электронных таблицах. По полученному решению распределение задач по процессорам представлено на рис. 1.6.

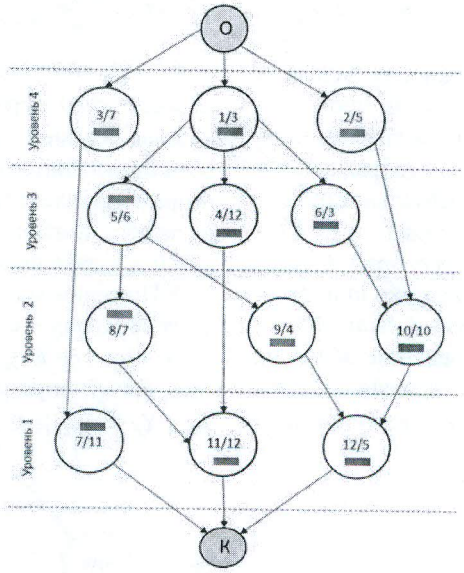

Рис. 1.6. Распределение задач по процессорам (– 1-й процессор, –2-й процессор, – 3-й процессор)

1.2.3.Разработка расписания

Согласно полученному решению загрузка процессоров распределяется следующим образом: процессор 1 – 28; процессор 2 – 26; процессор 3 – 29. Однако анализируя граф, представленный на рис. 8, легко видеть, что на первом и четвёртом уровнях занято не три, а два процессора. Кроме того, такое распределение задач по процессорам не учитывает их информационной связности. Поэтому при разработке расписания необходимо учитывать следующие два обстоятельства:

1. Задержка в завершении задач, не лежащих на критическом пути, до определенного момента может не влиять на срок завершения всего набора задач. Такие задачи обладают резервом времени – таким промежутком времени, на который может быть отсрочено завершение задачи без нарушения сроков завершения критического пути.

2. При составлении расписания с целью сокращения возможных простоев можно переносить выполнение запланированной к выполнению задачи с одного процессора на другой, если при этом не меняется последовательность выполнения, обусловленная информационными связями задач. Такая возможность связана с тем, что процессоры имеют общую память.

Резерв времени показывает, сколько имеется в запасе времени для выполнения данной задачи, на которое можно увеличить продолжительность данной задачи, не изменяя при этом продолжительности пути, которому принадлежит задача. Резерв времени задачи z_i определяется как разность между наиболее поздним $t_i^{\text{п}}$ и ранним $t_i^{\text{р}}$ сроками выполнения задачи и временем выполнения самой задачи : $R_i = t_i^{\text{п}} - t_i^{\text{р}} - t_i$. Поздний срок $t_i^{\text{п}}$ – это такой срок завершения задачи, превышение которого вызовет задержку в реализации задач, лежащих на критическом пути. Определим резервы времени, исходя из условия, что время завершения задачи не должно увеличивать величину критического пути. Для этого вначале определяем наиболее ранние сроки начала выполнения задач, рассматривая вычислительный процесс, начиная с вершины О:

1) Для задач уровня 4:
$$t_1^{\text{р}} = 0;\ t_2^{\text{р}} = 0;\ t_3^{\text{р}} = 0.$$
2) Для задач уровня 3 (здесь надо учитывать наиболее раннее выполнение задач предшествующего уровня):
$$t_4^{\text{р}} = 3;\ t_5^{\text{р}} = 3;\ t_6^{\text{р}} = 3.$$
3) Для задач уровня 2:
$$t_8^{\text{р}} = 9;\ t_9^{\text{р}} = 9;\ t_{10}^{\text{р}} = 6.$$
4) Для задач уровня 1:
$$t_7^{\text{р}} = 7;\ t_{11}^{\text{р}} = 16;\ t_{12}^{\text{р}} = 16.$$

После этого определяем наиболее поздние сроки завершения задач, начиная с вершины К:

1) Для задач уровня 1:
$$t_7^{\text{п}} = 28;\ t_{11}^{\text{п}} = 28;\ t_{12}^{\text{п}} = 28.$$
2) Для задач уровня 2:
$$t_8^{\text{п}} = 16;\ t_9^{\text{п}} = 23;\ t_{10}^{\text{п}} = 23.$$
3) Для задач уровня 3:
$$t_4^{\text{п}} = 16;\ t_5^{\text{п}} = 9;\ t_6^{\text{п}} = 13.$$
4) Для задач уровня 4:
$$t_1^{\text{п}} = 3;\ t_2^{\text{п}} = 13;\ t_3^{\text{п}} = 17.$$

Определяем резервы времени для выполняемых задач, которые могут быть использованы для снижения возможных простоев процессоров:

1) Для задач уровня 4:
$$R_1 = 3 - 0 - 3 = 1;\ R_2 = 13 - 0 - 5 = 8;\ R_3 = 17 - 0 - 7 = 10.$$
2) Для задач уровня 3:
$$R_4 = 16 - 3 - 12 = 1;\ R_5 = 9 - 3 - 6 = 0;\ R_6 = 13 - 3 - 3 = 7.$$
3) Для задач уровня 2:
$$R_8 = 16 - 9 - 7 = 0;\ R_9 = 23 - 9 - 4 = 10;\ R_{10} = 23 - 6 - 10 = 7.$$
4) Для задач уровня 1:
$$R_7 = 28 - 7 - 11 = 10;\ R_{11} = 28 - 16 - 12 = 0;\ R_{12} = 28 - 16 - 5 = 7.$$

Построим диаграмму загрузки процессоров, исходя из поярусной последовательности выполнения задач. Очевидно, что задачи с нулевым значением резерва времени принадлежат критическому пути и должны выполняться без задержек. Следующими должны выполняться задачи с минимально возможными резервами времени и т.д. При этом следует учитывать наличие информационной связи между задачами. Построить оптимальный план распределения задач по процессорам с учетом этих условий достаточно сложно, поскольку данная задача относится к классу NP-полных. Поэтому целесообразно использовать эвристические алгоритмы, дающие достаточно хорошее решение.

Пример эвристического алгоритма:

1. Рассматриваем задачи высшего уровня N. Назначаем на процессоры задачи, готовые к выполнению.
2. Переходим к следующему уровню N = N−1 (уровень 3). Если N=0, конец.
3. На процессор с минимальным временем освобождения назначаем задачу яруса, имеющую минимальный резерв времени на выполнение.
4. Если не все задачи текущего уровня назначены на процессоры, переходим к п.3 алгоритма. Если все задачи текущего уровня назначены на процессоры, переходим к п. 2.

Построим расписание, используя этот алгоритм.

1. Рассматриваем задачи высшего уровня N=4. Назначаем на процессоры задачи, готовые к выполнению. В нашем случае это задачи z_1, z_2, z_3.
2. Переходим к следующему уровню N = N−1 (уровень 3).
3. На процессор с минимальным временем освобождения (в нашем случае процессор 1, красный цвет на диаграмме) назначаем задачу яруса, имеющую минимальный резерв времени на выполнение. Это задача z_5.
4. На процессор с минимальным временем освобождения (в нашем случае процессор 1, красный цвет на диаграмме) назначаем задачу яруса, имеющую минимальный резерв времени на выполнение. Это задача z_5.
5. Так как не все задачи уровня 3 назначены на процессоры выбираем задачу текущего уровня с минимальным резервом времени. Это задача z_4. Назначаем ее на процессор с минимальным временем освобождения (процессор 3).
6. Оставшуюся задачу уровня 3 z_6 назначаем на процессор 2. Все задачи текущего уровня назначены. Переходим к п. 2 алгоритма, N= 3 – 1. Переход на уровень 2.
7. На процессор 1 назначаем задачу z_8, имеющую нулевой резерв времени. На процессоры 2 и 3 назначаем задачи z_{10} и z_9. Все задачи уровня 2 назначены. Переходим к п. 2 алгоритма, N= 2 – 1. Переход на уровень 1.

8. На процессор 1 назначаем задачу z_{11}, имеющую нулевой резерв времени. На процессоры 2 и 3 назначаем оставшиеся задачи.

9. Все задачи уровня 1 назначены. Переходим к п. 2 алгоритма, N= 1 – 1 =0. Конец.

Как видно из диаграммы (рис. 1.8), время реализации заданного пакета задач составляет 31 единицу времени при длине критического пути 28 единиц.

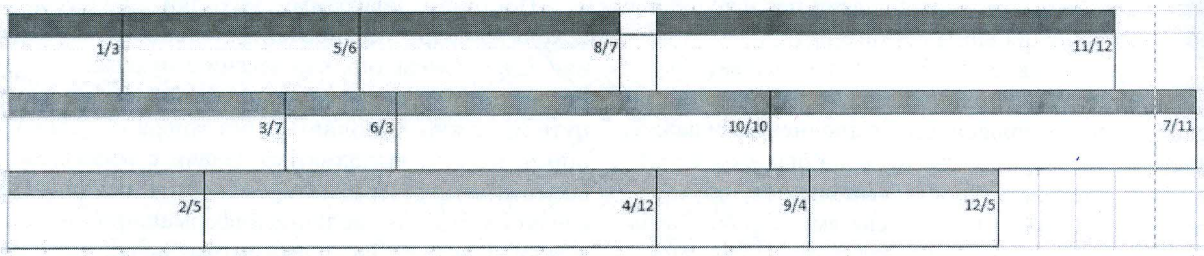

Рис. 1.8. Диаграмма загрузки процессоров для поярусного выполнения задач

Дальнейшее улучшение расписания можно получить, учитывая тот факт, что в полученном расписании загрузка процессоров неравномерная и, следовательно, имеется возможность улучшения плана за счет увеличения загрузки третьего процессора. Это можно получить, если вместо задач 9 и 12 на этом процессоре выполнять задачу 7, а задачи 9 и 12 выполнять на втором процессоре. Для проверки возможности такой замены нужно пересчитать времена наиболее позднего завершения этих задач, исходя из реально возможного значения критического пути, который в нашем примере равен 29.Такая проверка показывает, что перестановка возможна. Скорректированный план загрузки процессоров представлен на рис. 1.9. Заметим, что еще одну возможность оптимизации плана дает допустимость прерывания задачи и продолжение ее решения на другом процессоре.

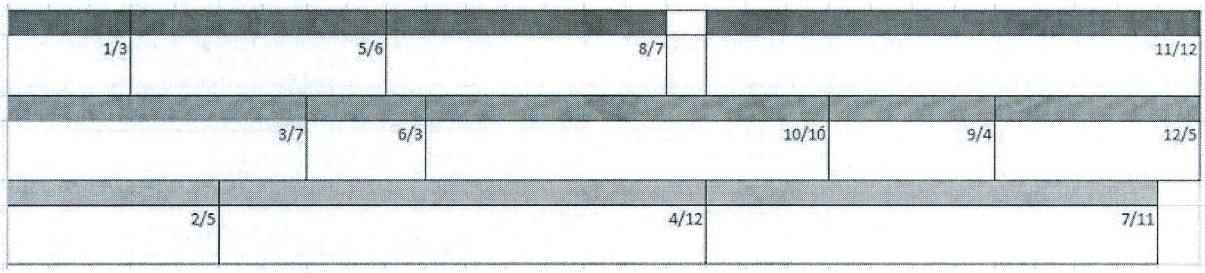

Рис. 1.9. Итоговая диаграмма загрузки процессоров

2. Параллельные вычислительные процессы в системах пакетной обработки

2.1. Представление структуры приложения и постановка задачи

Будем считать, что выполнение приложения производится на суперкомпьютере IBM типа Power или z. Известна структура приложения, состоящего из некоторого множества информационно-связанных частей (задач) с известным (ожидаемым) временем выполнения каждой задачи. Предполагаем, что каждая задача выполняется отельным логическим (виртуальным сервером) в логическом разделе LPAR, в который устанавливается гостевая операционная система. Считаем, что это время определяется элементарным вычислителем (процессором) многопроцессорного компьютера. Взаимодействие LPAR внутри физического сервера обеспечивается общей оперативной памятью или использованием виртуальной сети – HiperSocket [2]. Структуру подлежащего выполнению приложения удобно представить взвешенным ориентированным графом, как это показано на рис. 2.1, который будем далее использовать в качестве иллюстративного примера решения поставленной задачи (общность предлагаемого решения задачи от этого не изменится):

$G = \{< s_i, s_j >, t_{ij} | \, j > i, i = 0, 1, \dots, M-2, j = 1,2, \dots, M-1 \}$, где s_i, s_j – номера событий (вершины) в графе, t_{ij} – время решения задачи (вес соответствующей дуги), M – количество задач в пакете G. Считаем, что временные характеристики приложения известны по результатам его разработки и определены из условия выполнения каждой задачи приложения на одном процессоре (в одном LPAR) суперкомпьютера, выбранного для реализации этого приложения.

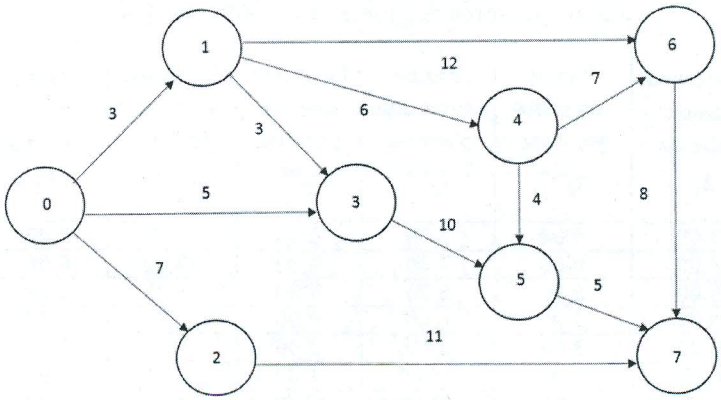

Рис. 2.1. Структура приложения

Требуется разработать план $P_{opt}(G)$ реализации заданного набора задач (ЗНЗ), обеспечивающий выполнение этих задач за время $T(G)$, не превышающее требуемого директивного значения T_d при условии минимизации ресурсов $Rs(G)$, занятых в процессе выполнения ЗНЗ. Таким образом, требуется найти такой план

$$P_{opt}(G) \in \{P(G) | \, T(G) \leq T_d\} \qquad (2.1)$$

при котором обеспечивается

$$min \, Rs(G) = Rs(G) | \, P_{opt}(G) \qquad (2.2)$$

Здесь правая часть выражения (2.1) означает множество возможных планов организации вычислительного процесса, которые обеспечивают завершение ЗНЗ за требуемое время. Оптимальным из этого множества будет план, для которого выполняется условие (2.2). Правая часть выражения (2.2) означает, что оптимальный план реализуется с минимальными затратами ресурсов системы.

Заметим, что решению задачи в постановке (2.1) – (2.2) или несколько иной форме посвящено достаточно много публикаций. Например, в [3] дана схема планирования работы параллельного суперкомпьютера, в фундаментальной монографии [4] даются методы статического планирования и балансировки нагрузки в параллельных и распределенных системах. Здесь содержится обзор и подробное обсуждение широкого спектра тем: от теоретических основ до практических методов планирования и распределения нагрузки. Значительное количество публикаций посвящено распараллеливанию алгоритмов и программ (например, [5]) а также методам параллельных вычислений при решении ряда задач оптимизации, методам организации оптимальных параллельных вычислительных процессов управления и информационного обслуживания, методам диспетчирования и синхронизации, применением методов параллельного программирования при разработке GRID-технологий [6]. В отличие от подобных публикаций, целью настоящей статьи является организация вычислительного процесса в суперкомпьютерах с динамически управляемыми разделами. При этом задача в постановке (2.1) – (2.2) заключается в минимизации требуемого числа разделов (т.е. минимизация требуемых ресурсов) при условии выполнения ЗНЗ в требуемое директивное время.

Для дальнейшей формализации задачи будем использовать понятия сетевого планирования и управления. В нашем случае работы представляются дугами графа $< s_i, s_j >$, или просто (i, j), причем для любой дуги $j > i$. Обмен информацией между задачами инициируется событиями, например, событие s_1 инициирует завершение работы $< s_0, s_1 >$, длительностью t_{01}, возможность запуска вычислений $< s_1, s_4 >, < s_1, s_6 >$, и $< s_1, s_{31} >$. Организация вычислительного процесса при выполнении данного приложения сводится к определению временных параметров сетевого графика и к его оптимизации по длительности выполнения всего комплекса задач и затратам вычислительных ресурсов в соответствии с заданными требованиями. Следует иметь ввиду, что, как правило, на суперкомпьютере выполняется в пакетном режиме несколько различных приложений, и вычислительные ресурсы для реализации каждого из них ограничены.

2.2. Решение задачи

2.2.1. Определение величины критического пути и резервов времени по отдельным вычислительным работам.

Результаты расчета сетевой модели выполнения пакета задач, представленного на рис. 2.1, приведены в табл. 2.1. Работы, не имеющие резерва времени и выделенные жирным шрифтом, образуют критический путь T_{kr} = 24.

Таблица 2.1

Таблица расчетов параметров сетевого графика

Работа (ij)	Продолжительность работы t_{ij}	Ранее начало работы $t_{ij}^{р.н}$	Ранее окончание работы $t_{ij}^{р.о}$	Позднее начало работы $t_{ij}^{п.н}$	Позднее окончание работы $t_{ij}^{п.о}$	Общий резерв работы R_{ij}	Частный резерв работы r_{ij}
(01)	3,00	0,00	3,00	0,00	3,00	**0,00**	0,00
(02)	7,00	0,00	7,00	6,00	13,00	6,00	0,00
(03)	5,00	0,00	5,00	4,00	9,00	4,00	0,00
(13)	3,00	3,00	6,00	6,00	9,00	3,00	0,00
(14)	6,00	3,00	9,00	3,00	9,00	**0,00**	0,00
(16)	12,00	3,00	15,00	4,00	16,00	1,00	0,00
(27)	11,00	7,00	18,00	13,00	24,00	6,00	0,00
(35)	10,00	6,00	16,00	9,00	19,00	3,00	3,00
(45)	4,00	9,00	13,00	15,00	19,00	6,00	0,00
(46)	7,00	9,00	16,00	9,00	16,00	**0,00**	0,00
(57)	5,00	16,00	21,00	19,00	24,00	3,00	3,00
(67)	8,00	16,00	24,00	16,00	24,00	**0,00**	0,00

2.2.2. Минимизация длины критического пути

Имеющийся резерв времени по работам, не лежащим на критическом пути, свидетельствует о том, что имеется возможность изъятия ресурсов, выделенных на некоторые работы, с целью добавления ресурсов на работы, лежащие на критическом пути. Это приведет к уменьшению длины критического пути, т.е. к сокращению всего цикла выполняемых вычислений. В нашем примере суммарный резерв времени составляет значение

$$R = \sum_{i=0}^{M-2} \sum_{j=1}^{M-1} R_{ij} = 32.$$

Передача ресурса от какой-либо работы означает увеличение ее выполнения на величину изъятого ресурса, добавление ресурса к другой работе, лежащей на критическом пути, означает уменьшение ее выполнения на величину добавленного ресурса. Например, изъятие половины резерва, т.е. трех единиц ресурса от работы (02) означает ее удлинение до 10 единиц времени. Это означает, что для ее выполнения нужен будет виртуальный процессор со следующим значением доли физического процессора

$$V_{02} = \frac{t_{02}}{t_{02} + 0{,}5 * R_{02}} = \frac{7}{7 + 0{,}5 * 6} = 0{,}7.$$

Изъятый ресурс можно добавить с целью сокращения критического пути, например, к работе (67). При этом время выполнения этой работы уменьшится с 8 до 5 единиц времени, но для выполнения этой работы потребуется виртуальный процессор со следующим значением доли физического процессора

$$V_{67} = \frac{t_{67}}{t_{67} - 0{,}5 * R_{02}} = \frac{8}{8 - 0{,}5 * 6} = 1{,}6.$$

Для решения задачи минимизации критического пути необходимо построить модель линейного программирования с целью определения значений T_{kr} для различных вариантов перераспределения ресурсов. Удобно это сделать в электронных таблицах, например, Excel. Воспользуемся для этого формализацией задачи поиска критического пути, предложенной в [7]. Исходные данные удобно представить в форме таблицы (табл. 2).

Таблица 2.2

Исходные данные для решения задачи поиска критического пути

	Переменные	X_{01}	X_{02}	X_{03}	X_{13}	X_{14}	X_{16}	X_{27}	X_{35}	X_{45}	X_{46}	X_{57}	X_{67}	Огранич. функция	Ограничение
	Значения переменных	1	1	1	1	1	1	1	1	1	1	1	1		
Номера узлов	0	1	1	1										3	1
	1	-1			1	1	1							2	0
	2		-1					1						0	0
	3			-1	-1				1					-1	0
	4					-1				1	1			1	0
	5								-1	-1		1		-1	0
	6						-1				-1		1	-1	0
	t_{ij}	3	7	5	3	6	12	11	10	4	7	5	8		

Задача поиска критического пути на основе табл. 2 заключается в определении значения функции

$$T_{kr} = \sum_{i=0}^{i=6} \sum_{j=1}^{j=7} t_{ij} \cdot x_{ij} \rightarrow \max \quad (2.3)$$

при следующих ограничениях:

$$1 \cdot x_{01} + 1 \cdot x_{02} + 1 \cdot x_{03} = 1; \quad (2.4)$$

$$-1 \cdot x_{01} + 1 \cdot x_{13} + 1 \cdot x_{14} + 1 \cdot x_{16} = 0 \quad (2.5)$$

$$-1 \cdot x_{02} + 1 \cdot x_{27} = 0; \quad (2.6)$$

$$-1 \cdot x_{03} - 1 \cdot x_{13} + 1 \cdot x_{35} = 0; \quad (2.7)$$

$$-1 \cdot x_{14} + 1 \cdot x_{45} + 1 \cdot x_{46} = 0; \quad (2.8)$$

$$-1 \cdot x_{35} - 1 \cdot x_{45} + 1 \cdot x_{57} = 0; \quad (2.9)$$

$$-1 \cdot x_{16} - 1 \cdot x_{46} + 1 \cdot x_{67} = 0; \quad (2.10)$$

$$\forall x_{ij} \in \{0, 1\} | \; i = 0,1, \ldots 6; j = 1,2, \ldots 7. \quad (2.11)$$

Решение задачи (2.3) – (2.11) для исходного графа сетевой модели показано на рис. 2.2.

	Переменные	x_{01}	x_{02}	x_{03}	x_{13}	x_{14}	x_{16}	x_{27}	x_{35}	x_{45}	x_{46}	x_{57}	x_{67}	Огранич. функция	Ограничение
	Значения переменных	1	0	0	0	1	0	0	0	0	1	0	1		
Номера узлов	0	1	1	1										1	1
	1	-1			1	1	1							0	0
	2		-1					1						0	0
	3			-1	-1				1					0	0
	4					-1				1	1			0	0
	5								-1	-1		1		0	0
	6						-1				-1		1	0	0
	t_{ij}	3	7	5	3	6	12	11	10	4	7	5	8		

T_{kr}	24

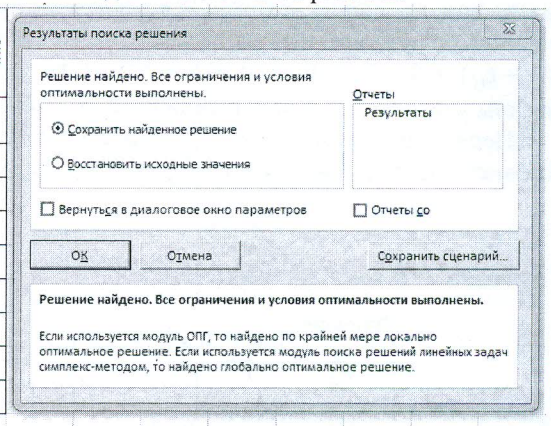

Рис. 2.2 Критический путь исходного графа сетевой модели

Определим теперь, как изменится величина критического пути, если 4 единицы резерва изъять у работы (45) и передать их для выполнения работе (67). При этом задача (45) будет выполняться 8 единиц времени, а задача (67) – 4 единицы времени. Решение задачи в этом случае показано на рис. 3 и дает значение критического пути, равное 22. Если рассчитанное значение критического пути окажется больше директивного, можно рассмотреть другие варианты использования резервов других работ, пока не будет полученное требуемое значение критического пути. Если этого не удается сделать за счет резервов в выполнении работ, необходимо увеличить количество ресурсов для их выполнения (в рассматриваемой задаче мы исходили из предположения, что каждой задаче выделяется один процессор).

2.2.3. Минимизация количества ресурсов выполнения пакета без увеличения длины критического пути

Будем считать, определенный выше критический путь, равный 24 единицам времени, соответствует директивному времени решения заданного пакета задач, т.е. $T_{kr} = T_d$. В этом случае становится актуальной задача минимизации ресурсов (в нашем случае количества процессоров), необходимых для реализации всех задач при условии непревышения длины критического пути. Другими словами, длительности выполнения всех работ пакета нужно изменить так, чтобы длина любого пути в графе была в идеале равна длине критического пути. Практически все вычислительные работы будут увеличены с учетом возможны резервов времени для их выполнения.

Для формализации задачи введем дополнительные обозначения:

t_{ij}^x – время выполнения работы (ij) после минимизации количества выделяемых ресурсов;

T_i^x – время свершения событий в графе работ (события отождествляются с вершинами графа);

N – исходное число процессоров, которое позволяет выполнить все работы по принципу: один процессор на одну работу;

N_m – минимальное количество процессоров, которое будет получено в результате решения оптимизационной задачи (пока еще без учета возможности их параллельной работы). Значение минимального количества процессоров определяется как сумма долей физических процессоров в виртуальных единичных процессорах логических разделах LPAR, т.е.

$$N_m = \sum_{i=0}^{i=6} \sum_{j=1}^{j=7} d_{ij},$$

где d_{ij} – доля физического процессора соответствующего виртуального процессора LPAR, необходимая для выполнения работы (ij) после минимизации количества выделяемых ресурсов,

$$d_{ij} = 1 - \frac{\left(t_{ij}^x - t_{ij}\right)}{t_{ij}^x}.$$

В качестве целевой функции задачи выбираем время занятости процессоров (полное машинное время) на решение пакета задач – его нужно минимизировать за счет использования имеющихся резервов времени на выполнение отдельных задач. Таким образом, целевая функция имеет следующий вид:

$$\sum_{i=0}^{i=6}\sum_{j=1}^{j=7} t_{ij} - \sum_{i=0}^{i=6}\sum_{j=1}^{j=7}(t_{ij}^{x} - t_{ij}) \rightarrow min \quad (2.12)$$

Первое слагаемое этой функции представляет собой полные затраты машинного времени на решение всего пакета задач. Второе – экономию машинного времени при условии того, что можно увеличить время решения некоторых задач за счет имеющегося резерва времени на их выполнение (здесь $(t_{ij}^{x} - t_{ij}) \geq 0$). Рассмотрим ограничения, которые должны учитываться при решении этой задачи.

Первый вид ограничений связан с принятым условием использования для решения задачи только имеющихся резервов для выполнения задач пакета. Отсюда следует система ограничений для продолжительности выполнения работ следующего вида:

$$t_{ij} + R_{ij} \geq t_{ij}^{x} \geq t_{ij}. \quad (2.13)$$

Второй вид ограничений должен обеспечить такое изменение длительности выполнения всех работ пакета, чтобы длина любого пути в графе была в идеале равна длине критического пути. В рассматриваемом примере таких путей шесть (перечислим их последовательностями вершин графа): $P_1 : 0 – 2 – 7$; $P_2 : 0 – 3 – 5 – 7$; $P_3 : 0 – 1 – 6 – 7$; $P_4 : 0 – 1 – 3 – 5 – 7$; $P_5 : 0 – 1 – 4 – 5 – 7$; $P_6 : 0 – 1 – 4 – 6 – 7$. Ограничения на длину этих путей имеют следующий вид:

$$L_i(P_i) \leq T_{kr} | 1 = 1,2,\ldots,M-1 \quad (2.14)$$

Решение задачи (2.12) – (2.14) показано на рис. 2.4 и 2.5.

Операция	Продолжительность работы t_{ij}	R_{ij}	$t_{ij} + R_{ij}$	t^{x}_{ij}	$t^{x}_{ij} - t_{ij}$	Исходное число процессоров N	Минимальное число процессоров Nm	Путь	Сумма работ по пути
(01)	3	0	3	3	0	1	1,00	0-2-7	24
(02)	7	6	13	13	6	1	0,54	0-3-5-7	24
(03)	5	4	9	9	4	1	0,56	0-1-6-7	24
(13)	3	3	6	6	3	1	0,50	0-1-3-5-7	24
(14)	6	0	6	6	0	1	1,00	0-1-4-5-7	24
(16)	12	1	13	13	1	1	0,92	0-1-4-6-7	24
(27)	11	6	17	11	0	1	1,00		
(35)	10	3	13	10	0	1	1,00		
(45)	4	6	10	10	6	1	0,40		
(46)	7	0	7	7	0	1	1,00		
(57)	5	3	8	5	0	1	1,00		
(67)	8	0	8	8	0	1	1,00		
	81			101	20	12	9,92		
	Fцел	61							

Рис. 2.4. Представление задачи (2.12) – (2.14) в электронных таблицах

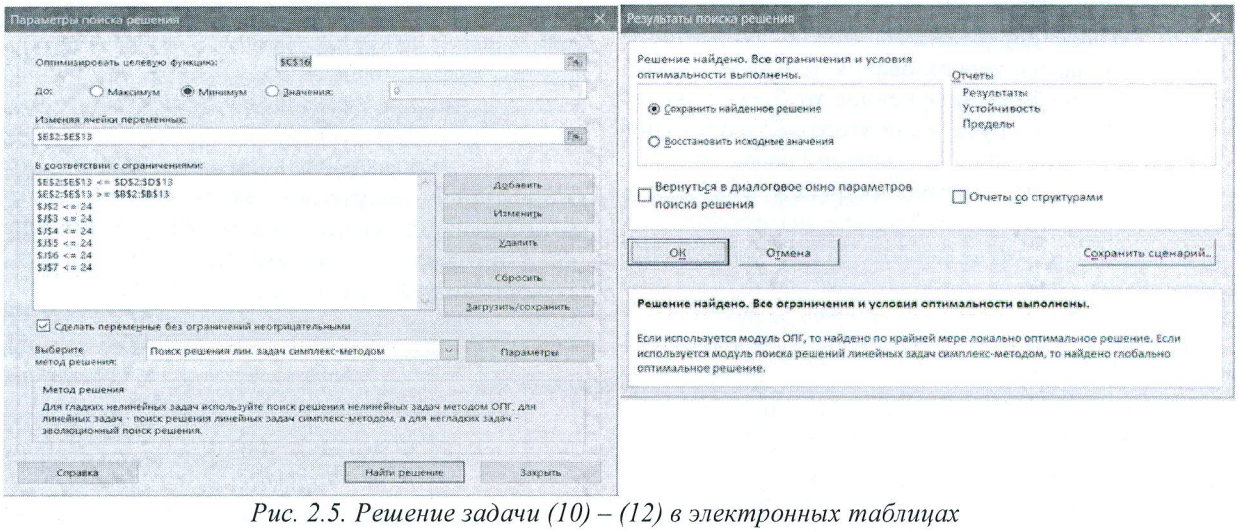

Рис. 2.5. Решение задачи (10) – (12) в электронных таблицах

Как видно из результата решения задачи минимизации использования ресурсов, число процессоров для реализации пакета задач без увеличения длины критического пути можно сократить с 12 до 10. Однако структура пакета свидетельствует о возможной параллельности работы этих процессоров при выполнении задач пакета.

2.2.4. Возможности организации мультипроцессорного выполнения пакета задач, представленного сетевой моделью

Потенциальную параллельность выполнения заданного набора задач можно определить, преобразовав граф задач в ярусно-параллельную форму.

Для получения ЯПФ пакета задач, представленного в форме сетевой модели, как показано на рис. 1, его предварительно необходимо преобразовать, заменив дуги графа (работы) вершинами. В преобразованном графе (рис. 2.6) номера вершин соответствуют работам, длительность которых пересчитана в соответствии с решением задачи (2.12) – (2.14) по рис. 2.4.

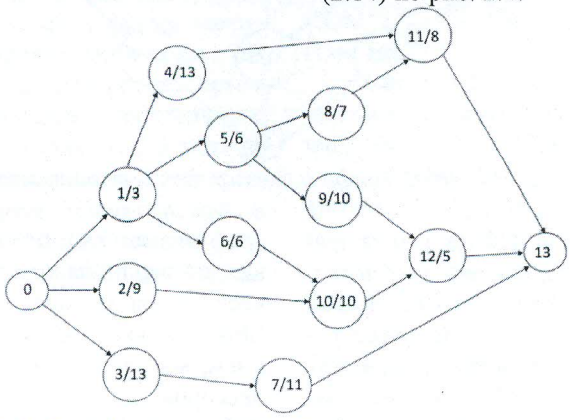

Рис. 2.6. Преобразованный граф пакета задач приложения

Получить ЯПФ можно, построив матрицу смежности графа. В нашем примере матрица смежности будет иметь следующий вид, приведенный на рис. 2.7.

Алгоритм распределения модулей системы по уровням:

1. Находим в матрице нулевые строки. В нашем случае это только одна строка с номером 13.

2. Вершина с этим номером образует нулевой (низший) уровень ЯПФ.

3. Вычеркиваем столбцы с номерами найденных вершин. В нашем случае – столбец 13.

4. Находим в матрице нулевые строки (7, 11, 12). Это вершины 1-го уровня.

5. Вычеркиваем столбцы с номерами 7, 11, 12.

6. Находим в матрице нулевые строки (3, 4, 8, 9 и 10). Это вершины 2-го уровня.

7. Вычеркиваем столбцы с номерами найденных вершин.

8. Находим в матрице нулевые строки (2, 5, 6). Это вершины 3-го уровня.

9. Вычеркиваем столбцы с номерами 5, 6.

10. Вершина с номером 1 образует 4-й уровень.

ЯПФ графа задач пакета, поученная на основе матрицы смежности представлена на рис. 2.8.

номер вершины		0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	5	0	0	0	0	0	0	0	0	1	1	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	12	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис. 2.7. Матрица смежности

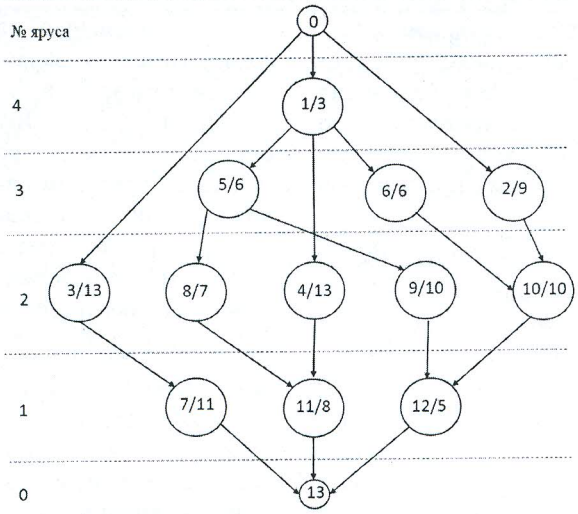

Рис. 2.8. ЯПФ графа пакета задач приложения

Построенный граф задач в ЯПФ далеко не прямоугольный, что неудобно для организации параллельного вычислительного процесса. Визуально из рис. 2.8 понятно, что граф можно легко перестроить, не меняя связей между вершинами, например, можно переместить вершины 2 и 3 на 4-й уровень, а вершину 4 – на 3-й. После таких преобразований граф примет вид, показанный на рис. 2.9. Здесь у каждой вершины курсивом дано число долей физического процессора, которое необходимо разделу LPAR выполнения задачи в соответствии с решением, минимизирующим количество ресурса процессоров для выполнения всего пакета задач (рис. 2.4).

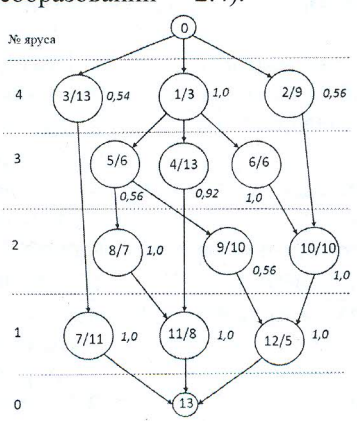

Рис. 2.9. ЯПФ пакета задач приложения

По ЯПФ (рис. 2.9) легко построить план реализации вычислительного процесса во времени. На рис. 2.10 сверху показана шкала времени (равная длине критического пути), под которой обозначены временные промежутки реализации задач пакета. В нижней части диаграммы приведены сведения о необходимом количестве логических разделов LPAR и количестве долей физических процессорных модулей (здесь необходимо округление с учетом дискретности выделения 0,1), образующих виртуальные процессоры разделов. Вертикальными стрелками показаны передачи данных между разделами. Конкретное представление найденного плана выполнения вычислительных работ по реализации ЗНЗ $P_{opt}(G)$ зависит от особенностей диспетчера вычислительной системы и в статье не рассматривается.

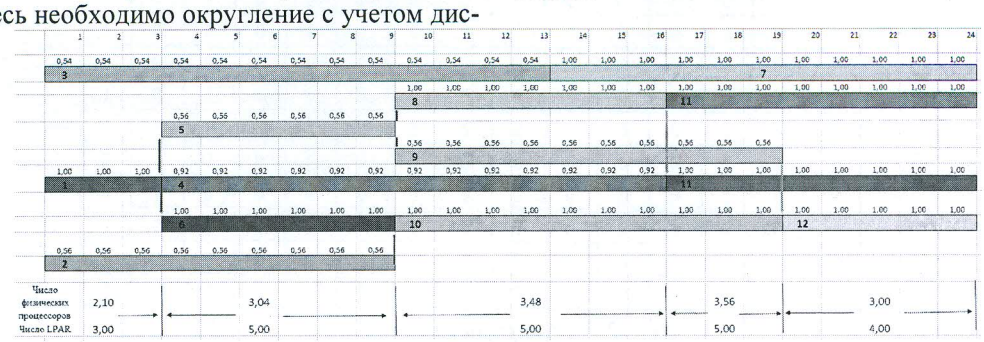

Рис. 2.10. Диаграмма выполнения вычислительных работ пакета

Заключение

Несмотря на принципиальные различия систем реального времени и пакетной обработки в данной работе дан достаточно общий подход к организации вычислительного процесса в подобных системах. Представлена совокупность математических моделей, формулировок задач и подходов к их решению, позволяющих построить расписание параллельного вычислительного процесса для реализации информационно-связанных задач на многопроцессорных вычислительных системах. Даны модели наборов решаемых задач в форме нагруженного графа и в ярусно-параллельной форме, решение задач о назначениях задач на процессоры и алгоритм составления расписания параллельного вычислительного процесса. Показано, что методы сетевого планирования и управления, методы организации мультипроцессорного планирования и управления могут быть успешно использованы в достаточно широком классе вычислительных систем.

СПИСОК ЛИТЕРАТУРЫ:

1. Назаров С.В. Операционные системы специализированных вычислительных комплексов: теория построения и системного проектирования. – М.: Машиностроение, 1989. – 400 с.

2. Руководство по внедрению IBM HiperSockets. [Электронный ресурс]. URL: http://www.redbooks.ibm.com/abstracts/sg246816.html

3. Polychronopoulos C.D. and Kuck D.J. Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. IEEE TRANSACTIONS ON COMPUTERS, VOL. C-36, NO 12, pp. 1425–1439 (Dec 1987).

4. Shirazi B., Hurson A., Krishna M. Scheduling and Load Balancing in Parallel and Distributed Systems. 1995, Wiley-IEEE Computer Society Pr. 520 P.

5. Карпов В. Е. Введение в распараллеливание алгоритмов и программ. Компьютерные исследования и моделирование 2010. Т. 2 № 3 С. 231–272.

6. Барский А.Б. Параллельное программирование. 2-е изд. – М.: Интуит, 2016. – 345 с.

7. Вагнер Г. Основы исследования операций. Том 1. Пер. с англ. Изд. «МИР». – М.: 1972. – 336 с.

STABILITY OF SHELLS OF VARIOUS SHAPES OF VARIABLE THICKNESS FROM A NONLINEAR ELASTIC MATERIAL DURING CREEP

Sadigov I.

УСТОЙЧИВОСТЬ ОБОЛОЧЕК РАЗЛИЧНОЙ ФОРМЫ ПЕРЕМЕННОЙ ТОЛЩИНЫ ИЗ НЕЛИНЕЙНО-УПРУГОГО МАТЕРИАЛА ПРИ ПОЛЗУЧЕСТИ

Садыгов И.Р.

Abstract

Shells of various shapes are now widely used in various sectors of construction and industry as the most optimal designs with high load-bearing capacity, which can be equipped with stiffening ribs, and significantly increase their strength characteristics.

The paper considers the stability issues of shells having a variable thickness and made of nonlinear elastic material. In the case of taking into account physical nonlinearity and the manifestation of creep, the functional of the total energy of the calculated shell is determined.

Аннотация

Оболочки различной формы в настоящее время нашли широкое применение в различных отраслях строительства и промышленности как наиболее оптимальные конструкции, обладающие высокой несущей способностью, которые могут снабжены рёбрами жёсткости, и в значительной степени повышают свои прочностные характеристики.

В работе рассмотрены вопросы устойчивости оболочек, имеющих переменную толщину и изготовленных из нелинейно – упругого материала. В случае учёта физической нелинейности и проявления ползучести определяется функционал полной энергии рассчитываемой оболочки.

Keywords: shell, stability, stiffening ribs, nonlinear elastic material, functional, creep, critical load, variable thickness, unit columnar functions, strain rate, program complexes, allowable stress.

Ключевые слова. Оболочка, устойчивость, рёбра жёсткости, нелинейно–упругий материал, функционал, ползучесть, критическая нагрузка, переменная толщина, единичные столбчатые функции, интенсивность деформаций, программные комплексы, допускаемое напряжение.

Оболочка – это тело, обладающее такими свойствами, при которых у него один из размеров значительно меньше остальных размеров. Как правило, оболочка ограничена двумя или более криволинейными поверхностями.

Оболочка может быть изготовлена из различных материалов: дерева, полиэтилена, стали, кожи, бетона, органического стекла, ткани, полимеров и многих других. Наиболее часто применяются оболочки в авиастроении и аэрокосмической отрасли, в которых они используются как корпуса ракет, самолетов, баки с горючим, ёмкости для технологических жидкостей и т.п.

На рис. 1 представлены различные виды оболочек.