ТЕХНИЧЕСКИЕ НАУКИ

УДК 681.3.012

С.В. Назаров, д-р техн. наук, профессор, Московский научно-исследовательский телевизионный институт (Москва, Россия) (e-mail: nazarov@mniti.ru)

ОПТИМИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА БОРТОВЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Актуальность тематики параллельных вычислений была осознана достаточно давно при решении сложных научно-технических задач, как в связи с низкой надежностью и производительностью компьютеров, так и в связи с появлением многопроцессорных систем и многоядерных процессоров. Технология обеспечения надежности и высокой производительности на основе параллельных вычислений естественным образом стала преобладающей в бортовых вычислительных системах (БВС). В настоящее время такие системы находят широкое применение в авиационной и космической технике, а также в наземных и водных подвижных объектах. Эффективность выполнения поставленных задач, безопасность, эксплуатационная пригодность и ряд других важных качеств подвижных объектов в значительной мере определяются способностью бортовой вычислительной системы выполнять свои функции. Развитие бортового оборудования характеризуется постоянным увеличением числа решаемых задач и повышением их сложности, расширением интеллектуальных и адаптивных возможностей. Это неизбежно приводит к усложнению БВС, ее операционной системы и специального программного обеспечения. На время решения большинства задач, возлагаемых на БВС, накладываются жесткие временные ограничения. Выполнение этих требований приводит к необходимости организации параллельных вычислительных процессов. В данной работе представлена совокупность математических моделей, формулировок задач и подходов к их решению, позволяющих построить расписание параллельного вычислительного процесса для реализации информационно-связанных задач на многопроцессорных бортовых вычислительных системах. Даны модели наборов решаемых задач в форме нагруженного графа и в ярусно-параллельной форме, решение задач о назначениях решаемых задач на процессоры и алгоритм составления расписания параллельного вычислительного процесса.

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

DOI: 10.21869/2223-1560-2018-22-2-6-17

Ссылка для цитирования: Назаров С.В. Оптимизация вычислительного процесса бортовых вычислительных систем // Известия Юго-Западного государственного университета. 2018. Т. 22, № 2(77). С.6-17.

***

Введение

Большая часть алгоритмов (программ), используемых в современных бортовых вычислительных системах (БВС), обладает достаточным внутренним параллелизмом, который может быть реализован многопроцессорными (многоядерными) бортовыми системами. Сложность этого подхода связана с тем, что архитектура аппаратных средств многопроцессорной системы зачастую фиксируется на этапе проектирования и остается неизменной на время выполнения программ. Это означает, что разработчик должен выбирать соответствующую систему для предполагаемых приложений. С этой целью разработчик должен разбить приложение на части, рассмотреть возможности параллелизма при выполнении приложения и выбрать соответствующую систему для своего приложения. Ограничением такого подхода является недостаток существующих многопроцессорных систем, заключающийся в отсутствии адаптивности на стадии разработки и во время выполнения программы [1 – 8].

Программируемая разработчиком логическая матрица предлагает более гибкое решение, потому что с ее помощью аппаратные средства можно переконфигурировать с новыми функциями и использовать повторно с различными приложениями. Некоторые поставщики программируемых пользователем логических матриц (компания Xilinx) предлагают специальную функцию, называемую динамическим и частичным переконфигурированием [9]. Это означает, что часть аппаратных средств системы может быть изменена во время выполнения программы, а оставшаяся часть остается действующей и неизменной.

В данной статье рассмотрим возможное решение в рамках первого подхода, позволяющее на основе анализа приложения определить целесообразную структуру многопроцессорной системы с выполнением требований на время выполнения приложений.

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, который будем далее использовать для иллюстрации решения поставленной задачи [10]:

$$G = \{< z_i, t_i > | i = 1, \dots, M \},$$

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.

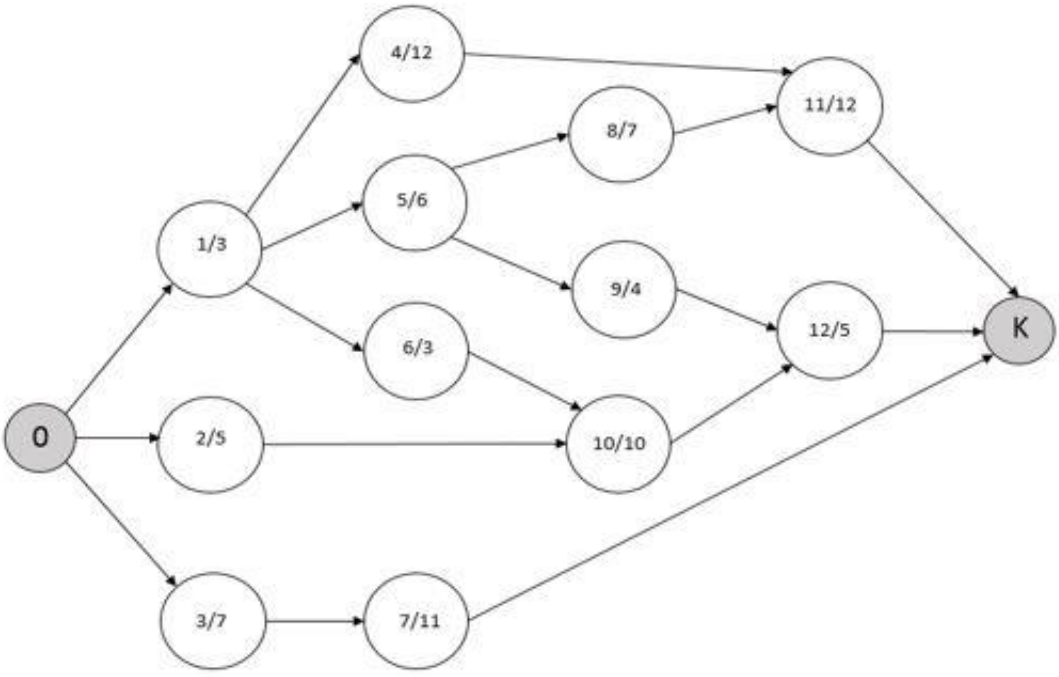

Рис. 1. Граф решаемых задач

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.

2. Решение задачи

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 форма графа (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 $(M+1) \times (M+1)$, (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. 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		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. Матрица смежности

Полученная таким образом ЯПФ заданного набора задач представлена на рис. 3. Алгоритм определения критического в ЯПФ приведен в [11]. В нашем случае это довольно просто сделать перебором всех возможных путей. Такой путь длиной в 28 единиц процессорного времени образует последовательность вершин $W_{\text{кр}} = \{1, 5, 8, 11\}$. Из рисунка следует, что высота ЯПФ равна четырем, минимальная ширина – $B_{\text{мин}} = 1$, а максимальная – $B_{\text{макс}} = 5$. Таким образом граф задач далеко не прямоугольный, что неудобно для организации параллельного

вычислительного процесса. Визуально из рис. 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*.

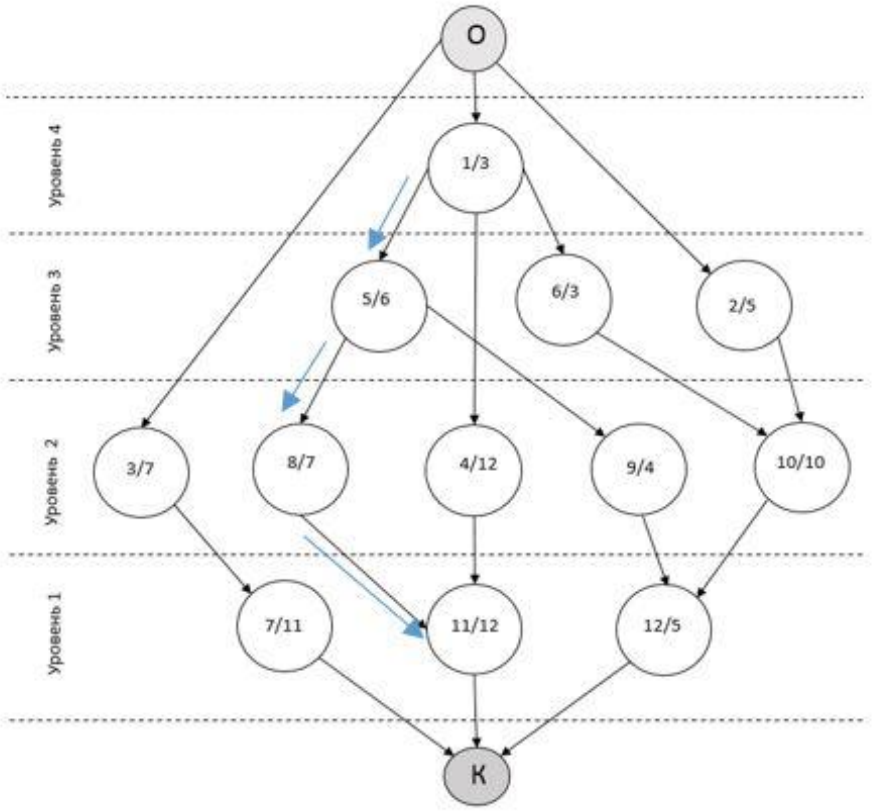

Рис. 3. Граф задач в ЯПФ

2.2. Выбор количества процессоров для параллельного вычислительного процесса

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

$$T_{\text{мин}} = \sum_{i=1}^{M-1} t_i.$$

В нашем примере $T_{\text{мин}}$= 85. При этом вычислительная нагрузка по ярусам не

равномерна и составляет следующие значения:

$$T_1 = 28, T_2 = 40, T_3 = 14, T_4 = 3. \quad (1)$$

Также неравномерна и ширина ярусов графа *G*:

$$B_1 = 3, B_2 = 5, B_3 = 3, B_4 = 1. \quad (2)$$

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

$$T_1 = 28, T_2 = 21, T_3 = 21, T_4 = 15. \quad (3)$$

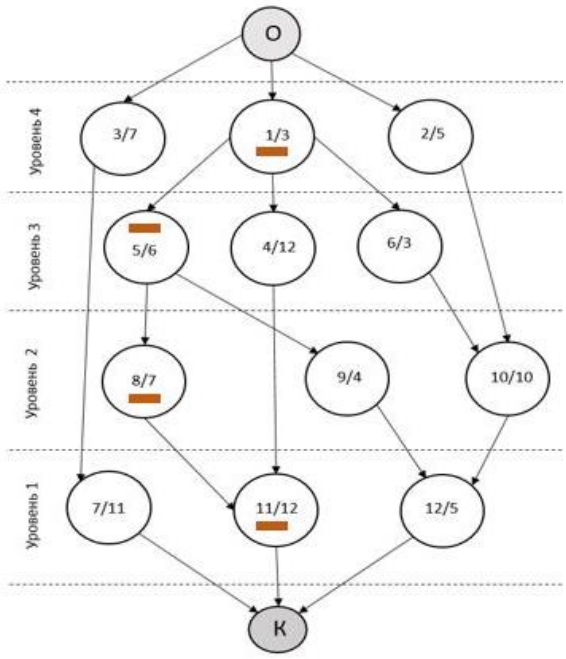

Рис. 4. Преобразованная ЯПФ

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.

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. 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 $X_o = \{x_{ij} | i \in I, j = 2,3\}$.

$$C = (\sum_{i \in I} \sum_{j=2} t_{ij} * x_{i2} -$$

$$- \sum_{i \in I} \sum_{j=3} t_{ij} * x_{i3} \rightarrow min. \qquad (4)$$

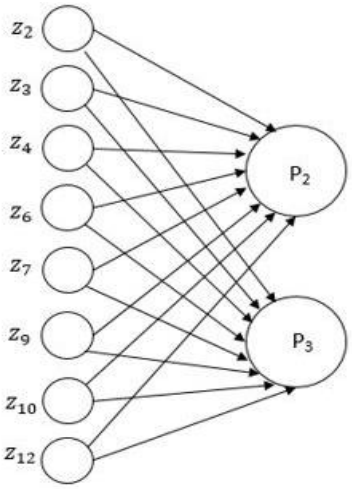

Рис. 5. Графовая интерпретация задачи о назначениях

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:

$$\forall i \sum_{j=2}^{j=3} x_{ij} = 1. \qquad (5)$$

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

$$\sum_{i \in I} t_{ij} * x_{ij} \leq 28 \quad j = 2,3. \qquad (6)$$

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

$$\forall x_{ij} \in \{0,1\}. \qquad (7)$$

Решение легко получить в электронных таблицах (рис. 6). По полученному решению распределение задач по процессорам представлено на рис. 7.

	A	B	C	D	E
1	t_i	значение	x_{22}	x_{23}	
2	t_2	5,00	1,00	0,00	1,00
3	t_3	7,00	1,00	0,00	1,00
4	t_4	12,00	1,00	0,00	1,00
5	t_6	3,00	0,00	1,00	1,00
6	t_7	11,00	0,00	1,00	1,00
7	t_9	4,00	1,00	0,00	1,00
8	t_{10}	10,00	0,00	1,00	1,00
9	t_{12}	5,00	0,00	1,00	1,00
10					
11					
12					
13		Цел. Функция	-1,00		
14		Ограничения по загрузке процессора 2	28	28	
15		Ограничения по загрузке процессора 3	29	28	
16					
17					
18					
19					
20					
21					
22					
23					

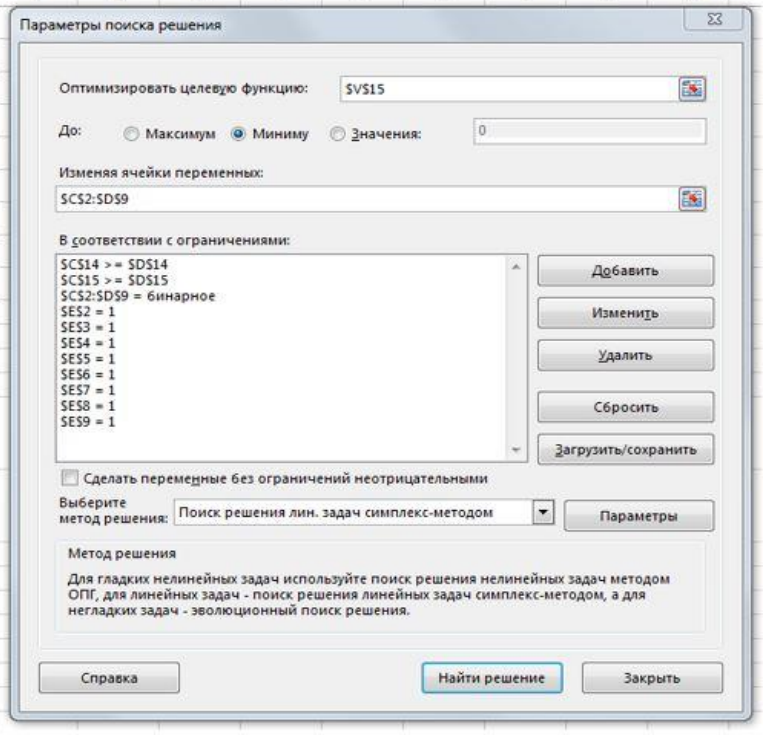

Рис. 6. Решение задачи (4) – (7) о распределении задач по процессорам

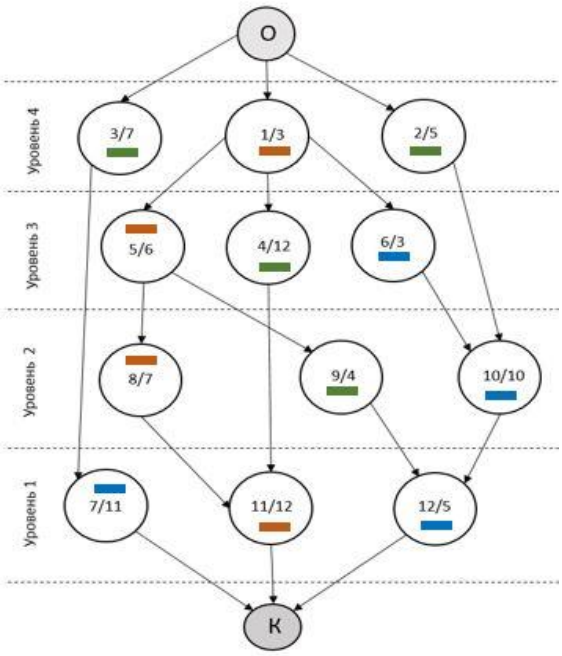

Рис. 7. Распределение задач по процессорам:
– 1-й процессор, –2-й процессор,
– 3-й процессор)

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:

Для задач уровня 4:

$$t_1^{\text{р}} = 0;\ t_2^{\text{р}} = 0;\ t_3^{\text{р}} = 0.$$

Для задач уровня 3 (здесь надо учитывать наиболее раннее выполнение задач предшествующего уровня):

$$t_4^{\text{р}} = 3;\ t_5^{\text{р}} = 3;\ t_6^{\text{р}} = 3.$$

Для задач уровня 2:

$$t_8^{\text{р}} = 9;\ t_9^{\text{р}} = 9;\ t_{10}^{\text{р}} = 6.$$

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:

Для задач уровня 1:

$t_7^{\text{п}} = 28;\ t_{11}^{\text{п}} = 28;\ t_{12}^{\text{п}} = 28.$

Для задач уровня 2:

$t_8^{\text{п}} = 16;\ t_9^{\text{п}} = 23;\ t_{10}^{\text{п}} = 23.$

Для задач уровня 3:

$t_4^{\text{п}} = 16;\ t_5^{\text{п}} = 9;\ t_6^{\text{п}} = 13.$

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:

Для задач уровня 4:

$R_1 = 3 - 0 - 3 = 1;$

$R_2 = 13 - 0 - 5 = 8;$

$R_3 = 17 - 0 - 7 = 10;$

Для задач уровня 3:

$R_4 = 16 - 3 - 12 = 1;$

$R_5 = 9 - 3 - 6 = 0;$

$R_6 = 13 - 3 - 3 = 7;$

Для задач уровня 2:

$R_8 = 16 - 9 - 7 = 0;$

$R_9 = 23 - 9 - 4 = 10;$

$R_{10} = 23 - 6 - 10 = 7.$

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:

Рассматриваем задачи высшего уровня N. Назначаем на процессоры задачи, готовые к выполнению.

Переходим к следующему уровню N = N−1 (уровень 3). Если N=0, конец.

На процессор с минимальным временем освобождения назначаем задачу яруса, имеющую минимальный резерв времени на выполнение.

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.

Рассматриваем задачи высшего уровня N=4. Назначаем на процессоры задачи, готовые к выполнению. В нашем случае это задачи z_1, z_2, z_3.

Переходим к следующему уровню N = N−1 (уровень 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.

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.

Так как не все задачи уровня 3 назначены на процессоры выбираем за-

дачу текущего уровня с минимальным резервом времени. Это задача z_4. Назначаем ее на процессор с минимальным временем освобождения (процессор 3).

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.

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.

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

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

Как видно из диаграммы (рис. 8), время реализации заданного пакета задач составляет 31 единицу времени при длине критического пути 28 единиц.

Рис. 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. 9. Заметим, что еще одну возможность оптимизации плана дает допустимость прерывания задачи и продолжение ее решения на другом процессоре.

Рис. 9. Итоговая диаграмма загрузки процессоров

Заключение

Приведенный пример оптимизации параллельного вычислительного процесса в бортовых вычислительных системах позволяет сделать следующие выводы:

Для планирования параллельного вычислительного процесса в бортовых вычислительных системах возможно использование ярусно-параллельных форм представления наборов решаемых задач. Такое представление позволяет предварительно определить требуемое количество процессоров и минимально возможное время реализации заданного набора задач.

Предварительное распределение решаемых задач по процессорам системы можно получить решением задачи о назначениях методом булевого линейного программирования.

Определение наиболее ранних и наиболее поздних моментов начала и завершения задач в ряде случаев позволяет улучшить первоначальное распределение задач по процессорам.

Улучшение полученного плана реализации задач и устранение обнаруженных простоев процессоров слабо поддается формализации и требует “ручной доработки”.

Список литературы

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ – Петербург, 2004. 608 с.

2. Боресков А. В. [и др.] Параллельные вычисления на GPU. Архитектура и программная модель CUDA. М.: Изд-во МГУ, 2012. 336 с.

3. Бурцев В.С. Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ. М.: Изд-во “Нефть и газ”, 1997. 150 с.

4. Таненбаум Э. Распределенные системы. Принципы и парадигмы, Ван Сеен. М.СПб.: Питер, 2003. 877с.

5. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. М. Издательский дом «Вильямс», 2003. 512 с.

6. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Н.Новгород, 2001. 386 с.

7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: BHV, 2002. 486 с.

8. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем, СПб.: Петербург, 2002. 360 с.

9. Гохрингер Д., Хюбнер М., Бекер Ю. Архитектура адаптивных многопроцессорных системна кристалле: новая степень свободы при проектировании систем и в поддержке при выполнении // Мир радиоэлектроники. М.: Техносфера, 2012. С. 146 – 173.

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

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

Поступила в редакцию 28.02.18

UDC 681.3.012

S.V. Nazarov, Doctor of Engineering Sciences, Professor, Moscow Research TV Institute Joint Stock Company (Moscow, Russia)(e-mail: nazarov@mniti.ru)

OPTIMIZATION OF COMPUTING PROCESS OF THE ONBOARD COMPUTING SYSTEMS

Relevance of scope of parallel calculations was realized for a long time at the solution of complex scientific and technical challenges, as in connection with low reliability and productivity of computers, and in connection with emergence of the multiprocessor systems and multinuclear processors. The technology of ensuring reliability and high efficiency on the basis of parallel calculations naturally became prevailing in the onboard computing systems (OCS). Now such systems find broad application in aircraft and space equipment, and also in land and water mobile objects. Efficiency of performance of objectives, safety, operational suitability and some other important qualities of mobile objects considerably are defined by ability of the onboard computing system to carry out the functions. Development of the onboard equipment is characterized by constant increase in number of the solved tasks and increase of their complexity, expansion of intellectual and adaptive opportunities. It inevitably leads to complication of BVS, its operating system and the special software. For the period of the solution of the majority of the tasks assigned to BVS rigid temporary restrictions are imposed. Performance of these of the requirement results in need of the organization of parallel computing processes. In this work set of mathematical models, formulations of the tasks and approaches to their decision allowing to construct the schedule of parallel computing process for realization of the information and connected tasks on the multiprocessor onboard computing systems is presented. Models of sets of the solved tasks in the form of the loaded count and in a yarusno-parallel form, the solution of tasks on purposes of the solved tasks to processors and algorithm of drawing up the schedule of parallel computing process are given.

Key words: *the multiprocessor system, information and connected tasks, yarusno-parallel form, parallel computing process.*

DOI: 10.21869/2223-1560-2018-22-2-6-17

For citation: Nazarov S.V. Optimization of Computing Process of the Onboard Computing Systems. Proceedings of the Southwest State University, 2018, vol. 22, no. 2(77), pp. 6-17 (in Russ.).

***

Reference

1. Voevodin V.V., Voevodin Vl.V. Parallel'nye vychislenija. Sankt-Peterburg, 2004, 608 p.

2. Boreskov A. V. i dr. Parallel'nye vychislenija na GPU. Arhitektura i programmnaja model' CUDA. Moscow, 2012, 336 p.

3. Burcev V.S. Parallelizm vychislitel'nyh processov i razvitie arhitektury superJeVM. Moscow, 1997, 150 p.

4. Tanenbaum Je., Van Seen M. Raspredelennye sistemy. Principy i paradigmy. Sankt-Peterburg, 2003. 877 p.

5. Jendrjus G.R. Osnovy mnogopotochnogo, parallel'nogo i raspredelennogo programmirovanija. Moscow, 2003, 512 p.

6. Gergel' V.P., Strongin R.G. Osnovy parallel'nyh vychislenij dlja mnogoprocessornyh vychislitel'nyh sistem. N.Novgorod, 2001, 386 p.

7. Voevodin V.V., Voevodin Vl.V. Parallel'nye vychislenija. Sankt-Peterburg, 2002, 486 p.

8. Nemnjugin S.A., Stesik O.L. Parallel'noe programmirovanie dlja mnogoprocessornyh vychislitel'nyh sistem. Sankt-Peterburg, 2002, 360 p.

9. Gohringer D., Hjubner M., Beker Ju. Arhitektura adaptivnyh mnogoprocessornyh sistemna kristalle: novaja stepen' svobody pri proektirovanii sistem i v podderzhke pri vypolnenii. Mir radiojelektroniki. Moscow, 2012, pp. 146 – 173.

10. Karpov V.E. Vvedenie v raspараllelivanie algoritmov i programm. Komp'juternye issledovanija i modelirovanie, 2010, vol. 2, no. 3, pp. 231 – 272.

11. Nazarov S.V. Operacionnye sistemy specializirovannyh vychislitel'nyh kompleksov: teorija postroenija i sistemnogo proektirovanija. Moscow, 1989, 400 p.