Уфа: УГАТУ, 2006 Вестник УГАТУ Т. 7, № 1 (14). С. 93–102

УДК 629.7.05:681.324

М. И. АХМЕТОВ, В. Н. ЕФАНОВ

ПРИНЦИПЫ РАЗРАБОТКИ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ БОРТОВЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Рассматриваются проблемы создания высокопроизводительных бортовых вычислительных систем реального времени и принципы их решения с использованием многопроцессорных структур с параллельной архитектурой. Предлагается аналитическая модель для описания вычислительных процессов, протекающих в многопроцессорной вычислительной системе. Приводится методика диспетчеризации вычислительных процессов в бортовой многопроцессорной системе на базе модифицированного генетического алгоритма, механизм адаптации которого направлен на достижения компромисса между скоростью сходимости и качеством получаемого локально-оптимального решения. *Бортовые вычислительные системы; многопроцессорные структуры; параллельная архитектура; аналитическая модель; макрокоманда; арифметико-логический граф; диспетчеризация; генетический алгоритм*

Мощным стимулом к возрождению отечественной авиационной индустрии может стать разработка летательных аппаратов пятого поколения, создаваемых в соответствии с программой «Перспективный авиационный комплекс фронтовой авиации» (ПАК ФА). При этом одной из главных составляющих, обеспечивающих боевую эффективность перспективного многофункционального истребителя, является бортовое радиоэлектронное оборудование (БРЭО). Тендер на разработку БРЭО для ПАК ФА выиграли две российские авиаприборостроительные компании: научно-производственный центр «Технокомплекс» и корпорация «Аэрокосмическое оборудование». Кафедра авиационного приборостроения УГАТУ уже в течение ряда лет активно сотрудничает с корпорацией «Аэрокосмическое оборудование» в области создания авионики нового поколения [1–5]. В данной статье обобщены результаты исследований, направленных на разработку высокопроизводительных бортовых информационно-вычислительных систем реального времени.

ПРОБЛЕМЫ СОЗДАНИЯ БОРТОВЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ НА БАЗЕ МНОГОПРОЦЕССОРНЫХ СТРУКТУР С ПАРАЛЛЕЛЬНОЙ АРХИТЕКТУРОЙ

Бортовой измерительно-вычислительный комплекс как система реального времени включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой код, пригодный для обработки этих показаний на процессорах, и, наконец, сами процессоры с программой, реагирующей на события, происходящие на объекте. Система реального времени ориентирована на обработку внешних событий. Причем реакция системы на событие должна быть своевременной, т. е. в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени. Кроме того, система должна успевать реагировать на одновременно происходящие события. Даже если два или большее число внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение временных интервалов, критических для этих событий.

Специфика бортовых систем реального времени (БСРВ) заключается в том, что они не допускают никаких задержек реакции системы ни при каких условиях, так как:

- результаты могут оказаться бесполезными в случае опоздания;

• может произойти катастрофа в случае задержки реакции;

• стоимость опоздания может оказаться бесконечно большой.

Таким образом, БСРВ представляет собой аппаратно-программный комплекс, реагирующий в предсказуемые моменты времени на непредсказуемый поток внешних событий.

Именно это приводит к коренным отличиям в структуре БСРВ, функциях ее ядра, построении устройств ввода-вывода по сравнению с системами реального времени общего назначения. Такие системы, особенно многопользовательские, ориентированы на оптимальное распределение вычислительных ресурсов между пользователями и задачами (системы разделения времени). В БСРВ подобная задача отходит на второй план — все отступает перед главной задачей — успеть среагировать на события, происходящие на объекте.

Еще одно отличие состоит в том, что системы общего назначения по своей природе должны быть универсальными, чтобы не зависеть от конкретных технических или программных средств или продуктов отдельных производителей. Применение БСРВ всегда связано с аппаратурой, объектом, событиями, происходящими на объекте, т. е. применение БСРВ всегда конкретно.

Отмеченные отличия оказывают существенное влияние на архитектуру и свойства операционных систем (ОС). Рассмотрим, какие параметры имеют ключевое значение с точки зрения оценки свойств ОС БСРВ.

Почти все производители систем реального времени приводят такой параметр, как время реакции системы на прерывание (interrupt latency). Это комплексный параметр, интегрально оценивающий способность системы вовремя отреагировать на внешние события. События, происходящие на объекте, регистрируются датчиками, информация с которых передается в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло соответствующее событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события. Таким образом, время реакции системы на прерывание охватывает интервал от момента возникновения события на объекте до выполнения первой инструкции в программе обработки этого события. При оценке величины этого параметра необходимо учесть следующее обстоятельство. Длительность интервала от момента возникновения события на объекте до генерации прерывания никак не зависит от ОС БСРВ и целиком определяется аппаратурой, а длительность интервала времени от возникновения запроса на прерывание до выполнения первой инструкции по его обработке определяется целиком свойствами операционной системы и архитектурой бортового вычислительного комплекса. Это время нужно уметь оценивать в худшей для системы ситуации, т. е. в предположении, что процессор загружен, что одновременно может происходить несколько прерываний, что система может выполнять какие-то действия, блокирующие прерывания.

Второй ключевой параметр ОС БСРВ — это время переключения контекста. В операционные системы реального времени заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все БСРВ являются многопроцессными и многозадачными. Чтобы уметь оценивать возможности системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче), т. е. время переключения контекста.

Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.). Следует заметить, что в условиях существующей тенденции на сближение ОС различных классов значение этого параметра уменьшается. Тем не менее, он остается важным, и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики.

Используя значения этих параметров, разработчик БСРВ должен выбрать такой механизм реального времени, который в максимальной степени устранит ограничения, связанные с особенностями архитектуры бортового вычислительного комплекса. Среди таких механизмов можно назвать различные модели межпроцессного взаимодействия, распределение памяти, принципы диспетчеризации в многозадачных и многопроцессных системах, обработку ошибок, автоматический контроль для обеспечения высокой готовности и безопасности.

В традиционных системах для организации межпроцессного взаимодействия (IPC — Inter Process communication) используется, как правило, модель «разделяемой памяти» (share memory). Большинству разработчиков хорошо известны трудности, связанные с управлением межпроцессным взаимодействием такого типа, в частности, необходимость обеспечения когерентности кэш-памятей, чтобы предотвратить опасность сбоя системы, когда один процесс изменяет данные, жизненно важные для другого процесса. Еще один недостаток межпроцессного взаимодействия через разделяемую память заключается в том, что данный метод плохо подходит для распределенных систем и характеризуется низкой степенью совместимости с механизмом управления памятью. Проблема обеспечения стабильности работы системы с IPC через разделяемую память требует достаточно длительного периода времени для отладки программного обеспечения и выявления ошибок.

В этих условиях для БСРВ более перспективной может оказаться модель межпроцессного взаимодействия через «передачу сообщений» (message passing). Согласно этой модели, процедура взаимодействия заключается в том, что один процесс посылает другому процессу сообщение. При этом может потребоваться лишь четыре системных вызова. Данный подход прост и надежен, прохождение сообщений в процессе работы системы может легко отслеживаться, что делает программное обеспечение прозрачным. Кроме того, при организации межпроцессного взаимодействия через передачу сообщений появляется возможность естественным образом распределять приложение между многочисленными центральными процессорами и процессорами цифровой обработки сигналов (DSP-процессоры). В то же время необходимо отметить, что IPC через передачу сообщений не отличается высокой скоростью. Данное обстоятельство имеет существенное значение с точки зрения организации режима реального времени. В связи с этим модель межпроцессного взаимодействия через передачу сообщений может быть рекомендована для перспективных БСРВ, использующих высокоскоростные каналы информационного обмена.

К числу базовых инструментов разработки сценария работы БСРВ относятся также механизмы назначения приоритетов процессов (задач) и алгоритмы планирования (диспетчеризации).

Чтобы гарантировать способность БСРВ вовремя реагировать на внешние события, рассмотрим формализованный способ пространственно-временной декомпозиции вычислительного алгоритма, реализуемого бортовым аппаратно-программным комплексом. Предлагаемый способ позволяет описывать в заданной вычислительной среде состояние вычислительного процесса с точностью до такта макрокоманд.

АНАЛИТИЧЕСКАЯ МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА В МНОГОПРОЦЕССОРНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Основная проблема, возникающая при разработке аналитических моделей вычислительных процессов, заключается в их высокой размерности. Это связано, в первую очередь, с наличием большого числа макрокоманд, таких как арифметико-логические операции, операции чтения-записи данных в буферное ОЗУ. Число подобных операций, в зависимости от сложности вычислительного алгоритма, может достигать нескольких тысяч. Рассмотрим возможности сокращения размерностей разрабатываемых моделей, базирующиеся на исследовании особенностей реализуемых вычислительных алгоритмов и специфических алгебраических свойств множества макрокоманд.

Вычислительные алгоритмы бортовых систем относятся к рекурсивным арифметико-логическим алгоритмам. Это алгоритмы, содержащие повторяющиеся операции одного и того же типа над последовательно поступающими данными. Как правило, многопроцессная и многозадачная реализация вычислительного алгоритма характеризуется конечным числом дискретных состояний (контекстов) $q_1, q_2, \ldots q_M$, в каждом из которых выполняется своя ветвь вычислительного процесса. Переключение контекста осуществляется по текущей информации о состоянии вычислительного процесса x, внешней среды μ, а также по командам на переключение q_c, поступающим из внешних систем. Аналитически процедуру переключения контекста можно выразить следующим образом:

$$q = \rho_K\left(x, \mu, q_c\right), \qquad (1)$$

где q — символьная переменная, принимающая значения из области $\Omega = \{q_1, q_2, \ldots q_M\}$, ρ_K — оператор квантования, который переменным величинам x, μ, и дискретным сигна-

лам q_c ставит в соответствие значение символа $q \in \Omega$.

Опишем состояние каждого из характерных фрагментов вычислительного алгоритма уравнениями следующего вида:

$$\begin{aligned} x_i(k+1) &= A_i x_i(k) + B_i u_i(k); \\ y_i(k) &= C_i x_i(k); \quad i = 1, 2, \ldots, N. \end{aligned} \tag{2}$$

Здесь векторы $x_i(k)$ описывают текущее состояние i-го фрагмента вычислительного процесса на k-м временном такте, векторы $u_i(k)$ определяют состояние внешних воздействий, влияющих на ход вычислительного процесса, векторы $y_i(k)$ содержат массивы выходных данных.

Взаимодействие отдельных фрагментов вычислительного алгоритма применительно к каждому из возможных контекстов можно описать следующим уравнением:

$$u(k) = LE(q)\, y(k) + L_0 E_0(q)\, g(k), \tag{3}$$

где $u(k)$, $y(k)$ — прямые суммы векторов $u_i(k)$ и $y_i(k)$, $g(k)$ — вектор внешних по отношению к БСРВ воздействий, $L = \|L_{ij}\|_{N\times N}$, $L_0 = \|L_{j0}\|_{N\times 1}$ — блочные матрицы, $E(q) = \text{blockdiag}\{E_1(q), \ldots, E_N(q)\}$ — блочно-диагональная матрица, элементы которой являются единичными или нулевыми матрицами в зависимости от того, принимает или не принимает участие в реализации q-го контекста соответствующий фрагмент вычислительного алгоритма; аналогичный смысл имеет матрица $E_0(q)$, элементы которой определяют участие в вычислительном процессе соответствующих внешних воздействий.

Объединяя (2) и (3), получим математическую модель вычислительного алгоритма в следующем виде:

$$\begin{aligned} x(k+1) &= A_c(q)\, x(k) + B_c(q)\, u(k); \\ y(k) &= Cx(k). \end{aligned} \tag{4}$$

Здесь $x(k)$ — прямая сумма векторов $x_i(k)$,

$$\begin{aligned} A_c(q) &= [A + BLE(q)\, C], \\ B_c(q) &= BL_0 E_0(q), \\ A &= \text{blockdiag}\{A_1, A_2, \ldots, A_N\}, \\ B &= \text{blockdiag}\{B_1, B_2, \ldots, B_N\}, \\ C &= \text{blockdiag}\{C_1, C_2, \ldots, C_N\}. \end{aligned}$$

Поставим в соответствие вычислительному алгоритму (4) арифметико-логический граф, определяющий последовательность выполнения арифметических и логических операций, операций чтения-записи данных в буферные ОЗУ, вызова констант из ПЗУ и т. д. Этот ориентированный граф включает следующие типы вершин (рис. 1):

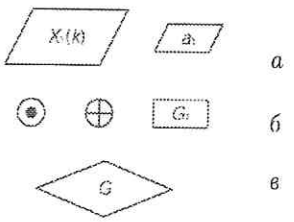

Рис. 1. Типы вершин графа

• вершины-истоки, соответствующие операциям чтения из буферных ОЗУ и ПЗУ или кэш-памяти процессора (рис. 1, *а*);

• арифметические вершины, соответствующие операциям сложения или умножения двух операндов — в том числе и умножения на логическую переменную (рис. 1, *б*);

• логические вершины, соответствующие определенным логическим операторам (например, условный оператор), — эти операторы определяют связи между операторами по управлению, задавая состав и порядок выполнения операторов (рис. 1, *в*).

Рассматриваемый граф является взвешенным, т. е. его ветвям присваиваются веса. С этой точки зрения ветви делятся на два класса:

• ветви с постоянными весами;

• ветви с логическими весами (соответствуют логическим переменным).

Поясним способ построения подобного графа на следующем примере. Пусть вычислительный алгоритм описывается следующей совокупностью арифметико-логических уравнений:

• если $u_1(k) < A$, то выполняется первый фрагмент

$$\begin{aligned} x_1(k+1) &= a_1 x_1(k) + b_1 u_1(k); \\ y_1(k+1) &= c_1 x_1(k+1); \end{aligned} \tag{5}$$

• если $u_1(k) \geqslant A$, то выполняется второй фрагмент

$$\begin{aligned} x_2(k+1) &= a_2 x_2(k) + a_3 x_3(k) + b_2 u_1(k); \\ x_3(k+1) &= a_4 x_2(k) + a_5 x_3(k) + b_3 u_1(k); \\ y_2(k+1) &= c_2 x_2(k+1) + c_3 x_3(k+1). \end{aligned} \tag{6}$$

Для анализируемого примера арифметико-логический граф будет иметь вид, представленный на рис. 2.

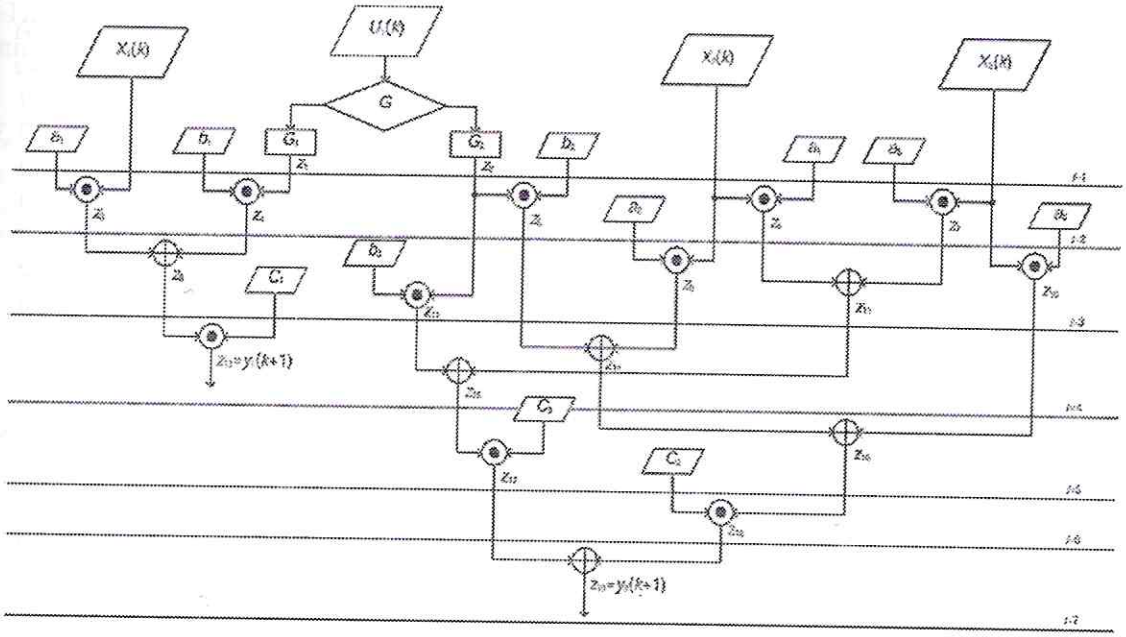

Рис. 2. Арифметико-логический граф алгоритма

Перейдем теперь к исследованию алгебраических свойств множества операций, образующих вычислительный алгоритм. Обозначим результаты выполнения операций, декомпозированных до уровня макрокоманд микропроцессора, через z_i, где i — номер соответствующей операции ($i = 1, 2, \ldots, r$). Поставим в соответствие графу следующую аналитическую модель:

$$z = \Pi_1 z + \Pi_2 v, \qquad (7)$$

где z — обобщенный вектор результатов операций; $v = [x(k); u(k)]^T$ — вектор исходных данных; Π_1,Π_2 — матрицы, определяющие последовательность выполнения операций в соответствии с вычислительным алгоритмом в зависимости от результатов выполнения предыдущих операций и исходных данных, соответственно.

Для графа, представленного на рис. 2, векторы из системы уравнений (7) принимают вид: $z = [z_1, z_2, \ldots, z_{19}]^T$, $\nu = [x_1, x_2, x_3, u_1]^T$; матрицы Π_1 и Π_2 приведены на рис. 3.

Основными характеристиками рекурсивных алгоритмов служат параллелизм и конвейерность. Алгоритм обладает параллелизмом уровня d, если максимально возможное количество операций, выполняемых за один отсчет времени (например, за такт исполнения макрокоманды), равно d. В свою очередь, алгоритм характеризуется конвейерностью длительности f, если количество отсчетов времени, за которое реализуется алгоритм, равно f.

Введенные понятия позволяют сопоставить свойства алгоритма с особенностями вычислительной системы, на которой он выполняется. В общем случае длительность исполнения алгоритма, выраженная в тактах макрокоманд, не может быть меньше его конвейерности, даже при условии, что на каждом временном такте задействуется число процессоров, равное соответствующему уровню параллелизма алгоритма. Поскольку уровень параллелизма может меняться в зависимости от такта вычислительной процедуры, то для обеспечения минимально возможной длительности исполнения алгоритма число процессоров в вычислительной системе должно быть не меньше, чем максимальный уровень параллелизма. Однако на тех тактах вычислений, для которых уровень параллелизма меньше максимального, отдельные процессоры будут незагруженными, что приводит к снижению производительности вычислительной системы.

Обозначим через d_i уровень параллелизма на i-м такте вычислительного алгоритма ($i = 1, 2, \ldots, f$), т. е. число операций, которые могут выполняться параллельно на данном такте. Отметим, что нумерация операций в пределах одного уровня параллелизма является произвольной. Такты алгоритма также отмечены на рис. 2.

Основной результат, связанный с разработкой аналитической модели вычислительного процесса, содержится в следующей теореме.

$$\Pi_1 = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & b_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & b_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & c_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{bmatrix}; \quad \Pi_2 = \begin{bmatrix}
0 & 0 & 0 & G_1 \\
0 & 0 & 0 & G_2 \\
a_1\alpha_1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & a_4\alpha_2 & 0 & 0 \\
0 & 0 & a_5\alpha_2 & 0 \\
0 & 0 & 0 & 0 \\
0 & a_2\alpha_2 & 0 & 0 \\
0 & 0 & a_3\alpha_2 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

Рис. 3

Теорема. Пусть $H_j = I_r - e_j\Pi_j^{(1)}$, где $\Pi_j^{(1)}$ — j-я строка матрицы Π_1, e_j — j-й координатный вектор, $r = \dim z$ — размерность вектора z, тогда решение системы (7) может быть представлено в следующем виде:

$$z = \prod_{i=f}^{1}\left[(d_i+1)\,I_r - \sum_{j=p_i+1}^{p_i+d_i} H_j\right]\Pi_2 v, \qquad (8)$$

где $$p_i = \sum_{k=1}^{i-1} d_k.$$

Доказательство. Тривиальное решение системы (7) имеет вид $z = (I_r - \Pi_1)^{-1}\Pi_2 v$. Однако вследствие чрезвычайно высокой размерности вектора z это решение оказывается неприемлемым. В то же время особенности построения вектора z приводят к тому, что матрица $(I_r - \Pi_1)$ является нижней треугольной с единичной главной диагональю, поскольку результаты операций, выполняемых на некотором i-м временном такте, могут быть получены только на основе использования результатов ранее выполненных операций. В связи с тем, что любая матрица вида $(I_r - \Pi_1)$ может быть представлена как произведение матриц H_j, т.е. $(I_r - \Pi_1) = H_2H_3\ldots H_r$, для обратной матрицы справедливо следующее выражение:

$$(I_r - \Pi_1)^{-1} = H_r^{-1}H_{r-1}^{-1}\ldots H_2^{-1}. \qquad (9)$$

Разобьем полученное произведение обратных матриц на группы сомножителей, относящихся к операциям, выполняемым на одном и том же временном такте:

$$H_r^{-1}H_{r-1}^{-1}\ldots H_2^{-1} = \prod_{i=f}^{1}\prod_{j=p_i+d_i}^{p_i+1} H_j^{-1}. \qquad (10)$$

Поскольку нумерация операций в пределах каждой совокупности, объединяющей операции, допускающие параллельное выполнение, является произвольной, то матрицы H_j^{-1} из соответствующей совокупности являются перестановочными, т. е.

$$\prod_{j=p_i+d_i}^{p_i+1} H_j^{-1} = \prod_{j=p_i+1}^{p_i+d_i} H_j^{-1}. \qquad (11)$$

В свою очередь, $H_j^{-1} = I_r + e_j\Pi_j^{(1)}$, причем произведение этих матриц дает нижнюю треугольную матрицу с единичной диагональю и с ненулевыми элементами, расположенными в строках с номерами $j = (p_i + 1, \ldots, p_i + d_i)$. Полученные таким образом нижние треугольные матрицы могут быть выражены в виде суммы единичной матрицы и элементарных матриц $e_j\Pi_j^{(1)}$, соответствующих ненулевым строкам:

$$\prod_{j=p_i+1}^{p_i+d_i} H_j^{-1} = I_r + \sum_{j=p_i+1}^{p_i+d_i} e_j\Pi_j^{(1)}. \qquad (12)$$

Преобразуем последнее выражение следующим образом:

$$I_r + \sum_{j=p_i+1}^{p_i+d_i} e_j \Pi_j^{(1)} = (d_i+1) I_r - \sum_{j=p_i+1}^{p_i+d_i} \left(I_r - e_j \Pi_j^{(1)} \right) = (d_i+1) I_r - \sum_{j=p_i+1}^{p_i+d_i} H_j.$$

Объединяя полученное соотношение с (9)–(11), приходим к результату, сформулированному в теореме:

$$z = (I_r - \Pi_1)^{-1} \Pi_2 v = \prod_{i=f}^{1} \left[(d_i+1) I_r - \sum_{j=p_i+1}^{p_i+d_i} H_j \right] \Pi_2 v.$$

Проиллюстрируем результат, содержащийся в приведенной теореме, на примере аналитического решения системы (7) для рассматриваемого алгоритма (5)–(6). Для данного примера число операций $r = 19$, конвейерность $f = 7$. Уровень параллелизма меняется в зависимости от вычислительного такта алгоритма следующим образом: $d_i = \{2; 5; 5; 3; 2; 1; 1\}$, где $i = 1, 2, \ldots, 7$ (см. рис. 2). Следовательно, решение будет иметь вид

$$z = \prod_{i=7}^{1} \left[(d_i+1) I_{19} - \sum_{j=p_i+1}^{p_i+d_i} H_j \right] \Pi_2 v.$$

Подставив численные значения величин в последнее выражение и выполнив необходимые вычисления, мы можем найти аналитические зависимости для финальных операций алгоритма — z_{13} и z_{19}, не выполняя промежуточных вычислений:

$$z_{13} = c_1 a_1 \alpha_1 x_1 + G_1 b_1 c_1 u_1,$$
$$z_{19} = (c_3 a_4 + c_2 a_2)\, \alpha_2 x_2 + (c_3 a_5 + c_2 a_3)\, \alpha_2 x_3 + G_2 (b_2 c_2 + b_3 c_3)\, u_1.$$

МЕТОДИКА ДИСПЕТЧЕРИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ В БОРТОВОЙ МНОГОПРОЦЕССОРНОЙ СИСТЕМЕ

Перейдем теперь к построению модели распределения загрузки между процессорами в многопроцессорной вычислительной системе. При этом основное внимание будем уделять характеру распределения фрагментов алгоритма управления между отдельными вычислительными блоками. Обозначим через $O = \{1, 2, \ldots, r\}$ множество номеров операций, составляющих алгоритм управления. Разобьем это множество на совокупность подмножеств $O_i = \{p_i + 1, \ldots, p_i + d_i\}$ $(i = 1, 2, \ldots, f)$ номеров операций, допускающих параллельное выполнение для i-го отсчета времени. Иначе говоря, O_i — i-я группа параллельных операций. Пусть в состав многопроцессорного вычислителя входит s $(1 \leqslant s \leqslant r)$ процессоров. Тогда каждая группа параллельных операций алгоритма может быть реализована данной совокупностью процессоров за $(k_i + 1)$ машинных тактов выполнения макрокоманд, если d_i не делится нацело на s, или за k_i тактов — в противном случае. Здесь $k_i =]d_i/s]$ — число полных тактов ($]d_i/s]$ означает целую часть дроби d_i/s), на которых загруженными оказываются все s процессоров. Если d_i не делится нацело на s, то последний $(k_i + 1)$-й такт этой последовательности временных тактов является неполным, т. е. на нем загруженными оказываются только $(d_i - sk_i)$ процессоров.

Выделим теперь в составе подмножеств O_i совокупности номеров операций, реализуемых j-м $(j = 1, 2, ..., s)$ процессором: $O_i^j = \left\{ l_{i,1}^j, l_{i,2}^j, \ldots \right\}$. В состав каждой такой совокупности входит $(k_i + 1)$ номеров, если j-й процессор оказывается загруженным на последнем (неполном) временном такте, и k_i номеров — в противном случае. Подобное разбиение подмножеств должно удовлетворять двум условиям:

1. $\bigcup_{j=1}^{s} O_i^j = O_i$ — это условие означает, что все операции i-й группы параллельных операций должны быть выполнены.

2. $O_i^l \cap O_i^k = \varnothing$ — каждая операция выполняется только на одном из процессоров $(l, k = 1, 2, ..., s)$.

Результаты распределения загрузки определяют такие важнейшие показатели эффективности вычислительного процесса, как длительность $T_и$ исполнения алгоритма, выраженную в тактах макрокоманд, и среднюю загрузку процессоров ω. Для оптимизации этих показателей воспользуемся модифицированным генетическим алгоритмом, суть которого сводится к следующим положениям.

Каждому варианту распределения r операций вычислительного алгоритма между s

процессорами поставим в соответствие вектор длиной $r \cdot s$ бит. Этот вектор-хромосому можно условно разбить на r групп, i-я из которых содержит информацию о номере процессора, выполняющего операцию с номером i. С этой целью в i-й группе зададим только один единичный бит, занимающий позицию $(i-1) \cdot s + j$. Это означает, что операция с номером i выполняется на j-м ($j = 1, 2, ..., s$) процессоре.

Состав начальной популяции формируется на основе случайной выборки с учетом того, что не все возможные варианты хромосом являются допустимыми. Допустимость определяется как двумя условиями, сформулированными ранее, так и тем, что процессор не может выполнять больше одной операции одновременно, т. е. в пределах одного машинного такта. Объем начальной популяции M определяется желаемой широтой поиска ($2 \leqslant M \leqslant s^r$).

Специфика вычисления функции пригодности применительно к рассматриваемому случаю заключается в том, что критерий оптимальности включает два показателя: длительность $T_{\text{и}}$ исполнения алгоритма и среднюю загрузку процессоров ω. Эти показатели рассчитываются по следующим формулам:

$$T_{\text{и}} = \sum_{i=1}^{f} \sum_{l=1}^{k_i+1} \max_j T_{ij}^l,$$
$$\omega = \frac{\sum_{i=1}^{f} \sum_{l=1}^{k_i+1} \sum_{j=1}^{s} T_{ij}^l}{s T_{\text{и}}}, \qquad (13)$$

где T_{ij}^l — длительность исполнения j-м процессором на l-м машинном такте макрокоманды, включающей транзакции записи, чтения, пересылки и операцию обработки данных и относящейся к i-му шагу вычислительного алгоритма.

Как правило, время, затрачиваемое на пересылку данных, превышает длительность вычислительной операции. Поэтому величина $\max_j T_{ij}^l$ зависит от топологии коммуникационной среды и принятого протокола обмена.

Чтобы узнать, есть ли на i-м шаге вычислительного алгоритма ($i > 1$) необходимость в передаче данных между процессорами, требуется проанализировать структуру вычислительного алгоритма. Это можно сделать с помощью матрицы смежности графа операций алгоритма. Для вершины, соответствующей некоторой операции p_i, выполняющейся на i-м шаге, нужно найти смежные ей вершины для $(i-1)$-го шага. Затем нужно определить, на каких процессорах выполняется операция p_i и смежные с ней операции. Если среди номеров этих процессоров есть отличающиеся, то между соответствующими процессорами необходима транзакция. Эту процедуру нужно повторить для всех операций i-го шага алгоритма.

Если на i-м шаге выполняется транзакция, то ее длительность можно оценить по формуле

$$T_i = \frac{m \cdot V}{b} + L, \qquad (14)$$

где m — длина пути между процессорами, b — пропускная способность канала «точка–точка» (байт/с), V — объем передаваемой информации в байтах, L — латентность (время, затрачиваемое на запуск процесса передачи данных, которое зависит от используемого сетевого программно-аппаратного обеспечения).

Длину пути m_{ij} между процессорами i и j можно определить с помощью матрицы $S_{\text{МВС}}$, задающей топологию коммуникационной среды. Эта длина определяется как минимальная степень, в которую нужно возвести матрицу $S_{\text{МВС}}$, чтобы ее элемент S_{ij} стал отличным от нуля

$$m_{ij} = \min \deg S_{\text{МВС}}^m \ (S_{ij} \neq 0). \qquad (15)$$

Очевидно, что больше всего времени занимает транзакция между наиболее удаленными друг от друга процессорами. Следовательно, при расчете длительности исполнения алгоритма необходимо на каждом машинном такте выявить транзакцию, соответствующую самому длинному пути. Применительно к этой транзакции определяется максимальная для данного машинного такта длительность макрокоманды.

Следует отметить, что длительность исполнения алгоритма с условными переходами, как и средняя загрузка, зависит от числа контекстных переключений. В этом случае при расчете соответствующих показателей можно воспользоваться суммарными характеристиками

$$T_{\text{и}} = \sum_{k=1}^{N} T_{\text{и}_k}, \quad \omega = \frac{\sum_{k=1}^{N} \omega_k T_{\text{и}_k}}{T_{\text{и}}}. \qquad (16)$$

Здесь $T_{\text{и}_k}$ — длительность исполнения ветви алгоритма, соответствующей k-му контексту.

Для расчета функции пригодности, отвечающей введенным показателям, воспользуемся методом обобщенных рангов. С этой целью производится ранжирование особей, входящих в рассматриваемую популяцию, сначала по первому, а затем по второму показателю. В качестве функции пригодности берется сумма рангов, присвоенных данной особи. Очевидно, что у наилучшей хромосомы в популяции функция пригодности будет иметь максимальное значение. Однако если эта хромосома попадет в другую популяцию, то значение ее пригодности изменится в соответствии с теми рангами, которые она получит в новой популяции.

Следующий этап генетического алгоритма — это отбор особей в соответствии с их функцией пригодности. Процедура отбора позволяет реализовать механизм адаптации параметров алгоритма оптимизации на основе компромисса между скоростью сходимости и качеством получаемого локально-оптимального решения. Суть предлагаемого механизм адаптации сводится к тому, что вероятность отбора особей гибко меняется в зависимости от предыстории поиска. С этой целью используется нормальный закон распределения вероятности отбора. Математическое ожидание принимается равным значению функции приспособленности, наилучшей для данного поколения хромосомы популяции. Если в очередном поколении произошла смена наилучшей хромосомы, то дисперсия принимает максимальное значение, расширяя тем самым диапазон поиска. Если же на протяжении нескольких поколений более предпочтительная хромосома не находится, то дисперсия уменьшается, в простейшем случае пропорционально числу поколений

$$D = D_{\max} - \beta \cdot g, \qquad (17)$$

где $D_{\max}$ — максимальное значение дисперсии; β — коэффициент, определяющий скорость сходимости алгоритма; g — число «неудачных» поколений.

Описанный механизм формирует предпосылки для элитного отбора, сохраняющего наилучшую из найденных хромосом популяции.

Рассмотренный принцип отбора используется в двух случаях:

- перед этапом кроссинговера для выбора скрещиваемых особей;
- после применения всех операторов генетического алгоритма для отбора наиболее пригодных особей в следующее поколение.

Оператор кроссинговера применяется по отношению к паре хромосом, прошедших отбор. В простейшем случае реализуется одноточечный кроссинговер. Для случайно выбранной пары хромосом определяется точка разрыва $k \in \left\{sd_1, s(d_1+d_2), ..., s\sum_{i=1}^{f-1} d_i\right\}$. Она выбирается также случайно и только на границах тех участков хромосом, которые кодируют операции, допускающие параллельное исполнение. Затем биты из двух выбранных хромосом, расположенные после точки разрыва, меняются местами. В результате образуются две хромосомы потомков и среди них случайным образом выбирается один. Этот процесс повторяется для всех хромосом заранее определенное количество раз. Более гибкой является схема, когда точки разрыва выбираются отдельно для каждого участка хромосомы и кроссинговер осуществляется независимо в пределах этих участков — многоточечный кроссинговер.

Наконец, после осуществления кроссинговера к хромосомам применяется оператор мутации. Его действие состоит в случайном изменении (на противоположное) значения каждого бита с заданной вероятностью мутации $P_{\text{мут}}$. Использование классической схемы мутации в нашем случае неприемлемо, поскольку может приводить к недопустимым вариантам загрузки процессоров. В связи с этим предлагается осуществлять мутацию не одного бита, а всего гена, состоящего из s бит и кодирующего номер некоторой операции p_i. Его код заменяется другой допустимой комбинацией с последующей проверкой следующего условия. Если машинный такт, на котором выполняется операция p_i, не содержит других операций, то мутация заканчивается. В противном случае значение гена, кодирующего некоторую другую операцию из этого же такта, заменяется исходным значением гена, соответствующего p_i.

Предложенный механизм адаптации обеспечивает еще одно важное преимущество — задает логически обоснованный критерий остановки поиска. Обычно в качестве такого критерия используется либо произвольно заданное число поколений, либо желаемое значение функции пригодности, при достижении которых поиск прекращается. В нашем случае поиск прекращается, если дисперсия уменьшается до такой величины, когда вероятность изменения лучшего из найденных вариантов загрузки становится пренебрежимо малой.

ЗАКЛЮЧЕНИЕ

Современный уровень требований, предъявляемых к бортовым информационно-вычислительным системам, обуславливает чрезвычайно высокую трудоемкость задач, связанных с детальной проработкой их функционального состава и коммуникационной среды, устанавливающей принципы информационного обмена между основными устройствами бортовой аппаратуры. В связи с этим необходимо, чтобы разработка вычислительных алгоритмов, планирование совместно протекающих вычислительных процессов и выбор структурной организации аппаратной части осуществлялись в рамках единой процедуры, обеспечивающей направленное формирование архитектуры БСРВ с заданным набором свойств. С этой целью в данной статье предлагается аналитическая модель вычислительного процесса в многопроцессорной вычислительной системе, позволяющая оценивать эффективность распределения операций вычислительного алгоритма между процессорами. Поиск оптимального варианта такого распределения целесообразно осуществлять с использованием модифицированного генетического алгоритма, механизм адаптации которого направлен на достижение компромисса между скоростью сходимости и качеством получаемого локально-оптимального решения. Все это обеспечивает необходимые предпосылки для разработки высокопроизводительных бортовых вычислительных систем пятого поколения.

СПИСОК ЛИТЕРАТУРЫ

1. **Бодрунов, С. Д.** Авионика пятого поколения и перспективы российского авиаприборостроения / С. Д. Бодрунов, В. Н. Ефанов // Сб. тр. II Всерос. науч.-техн. конф. Нац. ассоц. авиаприборостроителей. М., 1999. С. 14–35.
2. **Бодрунов, С. Д.** Концепция системной интеграции БРЭО на базе интеллектуальных информационных технологий /С. Д. Бодрунов, В. Н. Ефанов // Тез. докл. III Всерос. науч.-техн. конф. Нац. ассоц. авиаприборостроителей. СПб., 2000. С. 11–15.
3. **Ефанов, В. Н.** Пути повышения эффективности применения летательных аппаратов на базе быстросчетных моделей и средств искусственного интеллекта / В. Н. Ефанов, С. Д. Бодрунов // Мир авионики: Журн. Рос. приборостроительн. альянса. 2002. № 2. С. 33–36.
4. **Ефанов, В. Н.** Исследовательский стенд для оптимизации траекторий полета / В. Н. Ефанов, С. Д. Бодрунов // Авиационные технологии XXI века: Новые концепции летательных аппаратов и моделирование полета: Тез. докл. VII Междунар. симп. (ASTEC'02), Берлин, 2002. С. 35–36.
5. **Ефанов, В. Н.** Открытые архитектуры в концепции авионики пятого поколения / В. Н. Ефанов, С. Д. Бодрунов // Мир авионики: Журн. Рос. приборостроительн. альянса. 2004. № 5. С. 4–17.

ОБ АВТОРАХ

Ахметов Марат Искандарович, аспирант каф. авиац. приборостроения Дипл. инж. по орг. и технол. защиты информации (УГАТУ, 2003). Лауреат стип. Нац. ассоц. авиаприборостроит. (2004). Работает над дис. в обл. интеграции бортового оборудования.

Ефанов Владимир Николаевич, проф., зав. каф. авиационного приборостроения. Дипл. инж. по пром. электронике (УАИ, 1973). Д-р техн. наук по управлению в техн. системах (УГАТУ, 1995). Исследования в обл. интеллектуализ. комплексов бортового оборудования.