МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ
(Вычислительные системы)
1979 год Выпуск 78

УДК 681.142.2

РЕАЛИЗАЦИЯ ЭКСПЕРИМЕНТАЛЬНОГО РАСПАРАЛЛЕЛИВАТЕЛЯ АЛГОРИТМОВ

В.Г. Кербель

Описывается реализация для однородной вычислительной системы МИНИМАКС [1] автоматического распараллеливателя алгоритмов специального вида [2]. Дается синтаксический и семантический анализ программ, записанных на специализированном входном языке, предлагаются эвристические методы распараллеливания и минимизации межмашинных взаимодействий, а также подготовки ветвей к параллельному счету. Иллюстрируется эффективность предложенных методов оптимизации.

1. Вводные замечания

Использование системы МИНИМАКС - многомашинного комплекса на базе М-6000 - как единой вычислительной установки приводит к необходимости распараллеливания вычислительного процесса.

Под параллельным процессом (программой) понимается организованная совокупность процессов (ветвей), выполняемых одновременно в разных элементарных машинах (ЭМ) системы [3].

Для описания параллельных процессов в системе МИНИМАКС используются специальные языковые средства, получившие название ОВС-языков. Последние построены на базе языковых средств АСВТ-М путем их расширения двумя функционально полными наборами системных макрооператоров [4,5]. Множество параллельных процессов условно разбивается на два класса.

К первому классу относятся процессы, реализация которых связана с решением сложной задачи с большим числом ветвей и интенсивными взаимодействиями между ними. Параллельные алгоритмы для таких задач обычно разрабатываются вручную, т.е. на уровне создания метода обработки информации, а распараллеливание вычислений основано чаще всего на распараллеливании циклов (см.

[3]). В записи этих алгоритмов используются макрооператоры групповых взаимодействий [4].

К второму классу относятся параллельные процессы, реализация которых связана с решением сложных задач, имеющих относительно редкие взаимодействия между ветвями, или с решением потока простых слабо связанных между собой по информации задач, но конкурирующих при использовании общих ресурсов системы. В данном классе процессов доминирующее положение занимают парные связи, для описания которых используются макрооператоры индивидуальных взаимодействий [5]. Параллельные алгоритмы для таких задач могут разрабатываться как вручную, так и с помощью предлагаемого в работе специального программного средства системы МИНИМАКС, обеспечивающего автоматически предварительную подготовку ветвей параллельной р-программы. Подготовка заключается в составлении расписания реализации блоков (операторов) по ветвям р-программы.

2. Входной язык и структура алгоритмов

Алгоритм, поступающий на вход распараллеливателя, рассматривается как орграф $G = \langle X, V \rangle$, вершины X которого соответствуют подалгоритмам A_i алгоритма $A = \bigcup_{i=1}^{N} A_i$, а дуги $V = (v_{ij})$ указывают на наличие информационно-временных связей между A_i и A_j. Каждая вершина $x_i \in X$ представляет собой последовательность из входного оператора a_i, подалгоритма A_i и выходного оператора b_i. Выполнение A_i начинается после выполнения a_i; b_i – после A_i. Входной оператор a_i считается выполненным, если не существует j, для которого $v_{ij} > 0$ (все входящие дуги активизированы). Выполнение b_i активизирует все исходящие из вершины дуги.

Подалгоритмы A_i представляют собой достаточно крупные программные блоки, которые могут быть записаны на любом языковом средстве АСВТ-М-6000. Каждый A_i характеризуется двумя числами: ориентировочным временем выполнения T_i и необходимым для реализации ресурсом системы R_i (ранг подалгоритма).

В описываемой реализации первой версии экспериментального распараллеливателя алгоритмов (системы ЭРА) предполагается, что любой подалгоритм A_i требует единицу ресурса – одну ЭМ (т.е. $R_i = 1$, $i = \overline{1,N}$), а структура A подобна сетевому графику [2] (рис. I).

Информационные зависимости между подалгоритмами записываются на специализированном входном ЯЗЫКЕ РАСПАРАЛЛЕЛИВАЮЩЕЙ СИСТЕМЫ (ЯРС).Синтаксис этого языка приведен в работах [2,6]. Следуя [6], приведем примеры операторов ЯРС.Пусть зависимость A_j от A_i имеет вид  . Тогда описания выходных связей A_i могут иметь вид:

A_i,OUTPUT(T_i:A_j:C_1,V_1, C_2, V_2), (1)

а входных связей для A_j :

A_j, INPUT(A_i: B_1, V_1, B_2, V_2), (2)

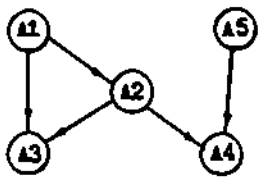


Рис. I.Пример алгоритма.

где C_1,C_2 – имена передаваемых массивов; B_1,B_2 – имена массивов,в которые помещается принятая информация.

Для задания имени и параметров распараллеливаемого алгоритма введен оператор имени NAME (N,L) , где N – количество подалгоритмов A_i, L – число ЭМ, на которых предполагается реализация алгоритма. Конец программы на ЯРС задается оператором =END).

Таким образом, программа на ЯРС есть последовательность операторов типа (I) или (I) и (2), перед которыми стоит NAME , а в конце =END).

Отметим, что оператор OUTPUT (блочный вывод) обладает функциональной полнотой для описания взаимодействий блоков, т.е. все взаимодействия между блоками могут быть описаны одним этим оператором. Оператор блочного ввода INPUT введен для удобства пользователей с целью автоматической проверки корректности описания информационных связей, а также для указания порядка оформления принимаемых массивов.

Ниже приведена запись на ЯРС алгоритма, структура которого представлена на рис.I. Использованы только операторы блочного вывода. Объемы передаваемой информации проставлены в условных единицах:

```
NAME (N,L)
A1,OUTPUT (T1:A2,A3:C1,V:A2:C2,V)
A2,OUTPUT (T2:A3:C3,3V:A4:C2,2V)
A5,OUTPUT (T5:A4:C5,5V)
A3,OUTPUT (T3)
A4,OUTPUT (T4)
=END)
```

3. Структура распараллеливателя

Распараллеливатель алгоритмов состоит из последовательно работающих функциональных блоков, включающих:

- настройку режима ввода программы на ЯРС;
- трансляцию, т.е. синтаксический и семантический анализ исходной программы, перекодировку и т.п.;
- упорядочение подалгоритмов для определения порядка их распределения по ветвям;
- выбор и настройку метода распределения;
- распределение подалгоритмов по ветвям р-программы и построение предварительного расписания;
- минимизацию обменов между ветвями генерируемой р-программы;
- выдачу окончательного расписания реализации подалгоритмов по ветвям р-программы.

Прохождение каждого алгоритма $A = \bigcup_{i=1}^{N} A_i$ через указанные блоки обеспечивает его разбиение на L подмножеств $G_1, G_2, \ldots, G_L$, которые образуют ветви эффективной в смысле [2] р-программы.

4. Настройка режима ввода

Ввод исходной программы, описывающей взаимосвязи подалгоритмов на языке ЯРС, возможен в режимах диалога (с клавиатуры) и с перфоленты. В обоих случаях предусмотрено совмещение трансляции с выдачей листинга программы с помощью основной управляющей системы. Настройку на нужный пользователю режим ввода система ЭРА осуществляет путем анализа управляющих операторов:

/C - ввод с перфоленты;
/C,L - ввод с перфоленты и выдача листинга;
/T - ввод с клавиатуры;
/T,L - ввод с клавиатуры и выдача листинга.

Пробелы в управляющем операторе недопустимы. Наличие пробелов воспринимается системой как ошибка, и выдается повторный запрос, после которого необходимо набить верный управляющий оператор. Запрос имеет вид:

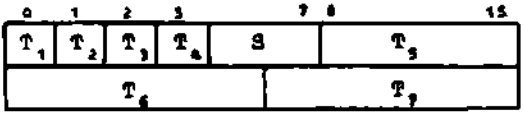
** СИСТЕМА ЭРА **

5. Синтаксический и семантический анализ

Для анализа операторов входного языка реализован модифицированный вариант ординарной TR-грамматики [7]. По этой модификации

правила с пустой левой частью [8] интерпретируется как переход к подмножеству правил-преемников. Введение связывающего правила TR-грамматики с признаком P списковой структуры позволяет поставить в соответствие в TR-таблице каждому подмножеству правил уровень, представляющий собой последовательность н е о б я з а т е л ь н о рядом стоящих строк TR-таблицы.

Правила модифицированной TR-грамматики представлены в памяти М-6000 двумя строками TR-таблицы следующей структуры:



где T_1 - элемент множества P = {0,1} ; T_2 - указатель типа правил (T_2=0 , если правило терминально); T_3 - указатель конца уровня (T_3=1 ,если уровень кончился); T_4 - указатель разомкнутых правил (T_4=1 , если правило разомкнуто); T_5 - символ из алфавита $\Sigma \cup V_N$ (Σ - терминальный, V_N - нетерминальный алфавит);T_6 - указатель интерпретируемой семантики; T_7 - адрес множества правил-преемников; S - указатель синтаксической ошибки.

Алгоритм функционирования синтаксического анализатора для модифицированной TR-грамматики отличается от использованного в [7] тем, что при p=1 ($p \in P$) игнорируется значение T_5 и выполняется переход к T_7 .

В процессе трансляции наряду с синтаксическими выявляются и семантические ошибки, появляющиеся в результате неверного описания связей между подалгоритмами. Считается, что связь между A_i и A_j описана неверно, если

- A_i передает данные A_j , но в записи оператора INPUT для A_j не указано имени A_i либо это имя указано, но в записи OUTPUT для A_i нет имени A_j;
- объемы пересылаемых данных между подалгоритмами A_i и A_j не совпадают в записях операторов INPUT и OUTPUT ;
- задана пересылка от A_i к A_i(петля);
- описания связей образуют цикл в графе G .

Семантические ошибки выявляются в процессе интерпретации I7 семантик, осуществляющих: сборку и перекодировку имен подалгоритмов; сборку чисел, указывающих время реализации подалгоритма и объемы пересылаемых данных; замену имени на номер; настройку соответствующих семантик на трансляцию оператора INPUT и OUTPUT ; запись T_1 на диагональ матрицы M ; проставление признака связано-

сти между A_i и A_j ; подсчет объема пересылаемых данных; поиск признаков связности для OUTPUT или проверку правильности описания связей для INPUT ; запись в соответствующие зоны памяти имени алгоритма A_i числа N подалгоритмов и числа L ветвей р-программы.

Результатом работы блока трансляции является матрица $M=(m_{ij})$ $(i,j=\overline{1,N})$ связности подалгоритмов и таблица имен A_i; при этом $m_{ii}=T_i$, $m_{ij}=\sum v_{ij}$ - суммарный объем передаваемых данных от A_i к A_j; если $m_{ij} \neq 0$, то $m_{ji}=0$, так как граф ориентирован.

6. Упорядочивание подалгоритмов

Известно [9], что результат эвристических методов распараллеливания (распределения вершин графа) зависит от формы представления графа. Распределение, наиболее близкое к оптимальному, достигается при упорядочивании вершин графа (подалгоритмов) по уровням.

В настоящей работе предлагается метод упорядочивания подалгоритмов по уровням за два просмотра матрицы М. Упорядочивание приводит к преобразованию графа (см. рис.1 и рис.2).

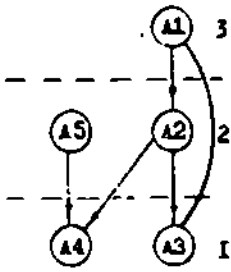


Рис.2. Преобразованный граф.

Суть метода поуровневого упорядочивания заключается в следующем: рассматриваются вектор уровня U и вектор исходящих из вершин графа дуг I , начальные значения которых нулевые. Значение $U(A_i)$, $i=\overline{1,N}$, указывает номер уровня, на который необходимо поместить подалгоритм A_i, а $I(A_i)$ указывает, сколько подалгоритмов зависит от A_i.

Просматривая i-й столбец матрицы смежности М графа G, подсчитываем число S исходящих дуг: если S=0 для A_i, то уровню $U(A_i)$ присваиваем значение I. За N^2 сравнений заполняем вектор I и находим номера всех подалгоритмов, которые надо поместить на первый уровень. Теперь для всех j таких, что $U(A_j)=1$, просматриваем j-е строки матрицы М и подалгоритмам A_i , для которых $m_{ij}>0$, присваиваем уровень 2. То же проделываем для $U(A_j)=2$ и соответствующим A_i присваиваем $U(A_i)=3$ и так до тех пор, пока не просмотрим все N строк. Отметим, что ни одна строка дважды не просматривается и поэтому уровни подсчитываются в сумме за $2 \cdot N^2$ сравнений.

После заполнения указанным способом векторов U и I упорядочиваем подалгоритмы. Вектору T присваиваем значение $T(A_i)=m_{ii}$,

вектором P указываем естественный порядок следования подалгоритмов, т.е. $P(A_i) = i$. Упорядочивание производим по убыванию уровней исходящих дуг и времени реализации подалгоритмов. Иными словами, A_i ставится в очередь на распределение перед подалгоритмом A_j, если истинна следующая дизъюнктивная форма:

$$(U(A_i) > U(A_j)) \vee ((U(A_i) = U(A_j)) \& (I(A_i) > I(A_j))) \vee$$
$$\vee ((U(A_i) = U(A_j)) \& (I(A_i) = I(A_j)) \& (T(A_i) > T(A_j))).$$

При перестановке элементов $U(A_i)$ и $U(A_j)$ переставляются также элементы $I(A_i) \rightarrow I(A_j)$ и $P(A_i) \rightarrow P(A_j)$. В результате перестановки вектор P укажет необходимый порядок назначения подалгоритмов по ветвям р-программы; $A_{p(i)}$ по возможности должен быть распределен раньше, чем $A_{p(j)}$, если $i < j$.

7. Выбор и настройка методов распараллеливания

В системе ЭРА предусмотрена программная настройка методов распараллеливания на время выполнения блоков. При этом один метод обеспечивает распараллеливание для случая, когда времена выполнения подалгоритмов почти равны, т.е. относительная погрешность не превышает заданной пользователем величины. Погрешность вычисляется по формуле:

$$\forall i = \overline{1,N} \quad \left| \frac{N \cdot T_i - \sum_{j=1}^{N} T_j}{\sum_{j=1}^{N} T_j} \right| \leq \frac{\varepsilon}{100}. \qquad (3)$$

Значение ε задается двузначным целым числом после выдачи системой ЭРА запроса:

ДОПУСТИМОЕ ОТКЛОНЕНИЕ ВРЕМЕНИ
•

При нарушении условия (3) реализуется другой метод, обеспечивающий плотный режим расписания реализации блоков по ветвям р-программы. Под плотным режимом понимается такое расписание, при котором ведется учет ресурса времени ЭМ и на освободившуюся ЭМ назначается подалгоритм, готовый к выполнению.

Указанное разделение методов распараллеливания обусловлено более быстрой работой первого из них. Построение расписания вторым методом может быть сведено к первому [10] за счет увеличения

размерности матрицы M, однако это не всегда возможно из-за большого расхода оперативной памяти.

8. Методы составления расписания

Методы распараллеливания вырабатывают расписание реализации подалгоритмов по ветвям р-программы в виде матрицы Y, в которой Y_i (i-й столбец) соответствует подмножеству G_i. Подмножество подалгоритмов A_i, содержащихся в G_i, будут реализованы на i-й машине системы.

Метод распараллеливания для случая равного времени реализации подалгоритмов состоит из следующих этапов:

I. i:=1, N, начиная с первого элемента вектора P, последовательно найдем такой номер i_1 подалгоритма, чтобы $P(i_1) > 0$. Если номер i_1 найден, то переходим к п.2, иначе к п.6.

2. Просматриваем i_1-ю строку матрицы M. Если для всех $j(j \neq i_1)$ значение $m_{i_1,j} \leq 0$, то переходим к п.3, иначе к п.5.

3. Увеличиваем счетчик S распределенных блоков и включаем номер i_1 в подмножество G_i. Если $i = L$, то переходим к п.4, иначе i:=i+1 и к п.5.

4. Для всех номеров i_j, включенных в последнем временном кванте в подмножества G_r ($r \in \overline{1,L}$), изменяем на минус знак у $P(i_j)$ и у всех элементов (кроме диагонального) столбца i_j. Если S = N, то переходим на п.6, иначе увеличиваем номер временного кванта на единицу и переходим к п.I.

5. Продолжаем поиск i_1 в векторе P. Если i_1 найден, то переходим к п.2, иначе к п.4.

6. Переходим на минимизацию обменов (см.раздел 9).

Отметим, что изменение знака у элементов вектора P фактически означает вычеркивание из графа тех вершин, для которых номера вошли в одно из подмножеств G_r. Результатом работы описанного метода распараллеливания является матрица Y, элемент $y(i,1)$ ($i \in \overline{1,k}$; $1 \in \overline{1,L}$; K - количество временных квантов) которой есть номер подалгоритма, назначенного для выполнения в i-й ветви р-программы в i-й квант времени; если $y(i,1) = 0$, то i-я элементарная машина ОВС, реализующая i-ю ветвь р-программы, в i-й квант времени не занята выполнением алгоритма A и может быть использована для других целей.

Квант времени определяется как $\tau = \sum_{i=1}^{N} T_i/N$. Так как задана величина ε допустимого отклонения времени выполнения подалгоритмов, то при выдаче расписания реализации алгоритма для l-й ветви будет отмечено потенциальное время простоя в i-м кванте времени $\tau_{i_1} = \max T_{i_j} - T_{i_1}$, где i_j - номера подалгоритмов, реализуемых в i-м кванте времени ($j = \overline{1,L}$).

При нарушении условия (3) система настраивается на метод плотного расписания, состоящий из следующих этапов:

I. Векторам K и C присваиваем начальные значения: K(i) = 1, C(i) = 0, $i = \overline{1,L}$; переходим на 2.

2. Находим значение $\alpha = \min C(i)$. В вектор K заносим номера подмножеств G_r, для которых $C(r) = \alpha$. Пусть занесено N_0 таких номеров и $N_1 = 1$. Если $\alpha = 0$, то переходим к п.3, иначе к п.6.

3. Начиная с первого элемента вектора P, отыскиваем такой номер i_1 подалгоритма, что $P(i_1) > 0$. Если i_1 найден, то переходим к п.4, иначе к п.9.

4. Просматриваем строку i_1 матрицы M. Если для всех $j = \overline{1,N}$ $(j \neq i_1)$ значение $m_{i_1,j} \leq 0$, то переходим к п.5, иначе к п.7.

5. Назначаем подалгоритм с номером i_1 в подмножество G_l $(l = K(N_1))$; K(1):=K(1) + 1; C(1):=C(1)+T_{i_1}. Если $N_0 = N_1$, то переходим к п.2, иначе $N_1 := N_1 + 1$ и переходим к п.7.

6. Для всех номеров i_j, включенных последними (на что указывает K(r)) в подмножества G_r, $r = \overline{1,N_0}$, изменяем знак на минус у $P(i_j)$ и у всех элементов (кроме диагонального) столбца i_j матрицы M. Переходим к п. 3.

7. Продолжаем дальнейший поиск i_1 в векторе P. Если i_1 найден, то переходим к п.4, иначе к п.8.

8. Находим значение $\gamma = \min_{i \in \Pi} C(i)$, где $\Pi = \{ i \,/ (C(i) > 0) \& \& (i \in \overline{(1,L \setminus K(N_1), K(N_0))}) \}$. В подмножества G_r $(r = \overline{K(N_1), K(N_0)})$ заносим время простоя $t = \gamma - \alpha$; выполняем K(r):=K(r) + 2. Переходим на 2.

9. Переходим на выдачу расписания.

В полученной матрице распределения T элемент y(i,1) означает

- номер подалгоритма, определенного для исполнения в l-й ветви p-программы на i-м шаге распределения, если $y(i,1) \neq 0$ и $y(i-1,1) \neq 0$;

- время простоя ЭМ, реализующей l-ю ветвь, если $y(i-1,1) = 0$;

- указатель времени простоя ЭМ, если $y(i,1) = 0$. Время простоя будет указано в $y(i+1,1)$.

Например, распараллеливание по двум ветвям алгоритма, структура которого изображена на рис.I, имеет следующий вид:

$$T_1 = \begin{pmatrix} I & 5 \\ 2 & 0 \\ 4 & 3 \end{pmatrix} \text{ при } c = 99; \qquad T_2 = \begin{pmatrix} I & 5 \\ 2 & 0 \\ 3 & 2 \\ 0 & 4 \end{pmatrix} \text{ при } c = 0;$$

$$T_1 = 2,\ T_2 = 3,\ T_3 = 5,\ T_4 = 4,\ T_5 = 3.$$

9. Минимизация обменов

При выполнении условия (3), т.е. когда времена T_i реализации подалгоритмов считаются равными, проводится минимизация обменов между ветвями полученного расписания. Выделение указанного случая обусловлено синхронизацией подалгоритмов перед их выполнением и после него в каждом кванте времени τ, что позволяет проводить локальную минимизацию обменов путем взаимных перестановок элементов подмножеств G_i. Конечно, при различных временах реализации подалгоритмов можно найти такое τ_0, что $T_i = k_i \cdot \tau_0$ для всех $i = \overline{1,E}$, и представить A_i как последовательность непосредственно связанных работ $A_{i1}, \ldots, A_{ik_i}$ с объемом пересылок $\tilde{V}$. В этом случае, если $\tilde{V}$ велико, использование расписания второго метода эквивалентно стягиванию всех A_{ij}, $j=\overline{1,K_i}$, в одну ветвь.

Минимизация обменов между ветвями р-программы проводится по квантам времени. Для этого строится матрица $Z = z(i,j)$, $i,j = \overline{1,L}$, характеризующая пересылки данных в каждую ветвь; строка Z^i матрицы Z отражает объем передаваемых данных из i-й ветви во все другие; столбец Z_i - объем принимаемых данных в i-ю ветвь. Сумма всех элементов Z есть объем пересылаемых данных между подалгоритмами, попавшими в рассматриваемый квант времени. Значение $z(i,i) = Z^i \cap Z_i$ есть объем пересылок внутри i-й ветви. Поэтому, максимизируя след матрицы Z только перестановкой строк Z^i, мы минимизируем объем обмена между ветвями.

В общем случае максимизация следа матрицы требует L! перестановок строк. Предлагаемый эвристический метод позволяет достаточно точно максимизировать след матрицы за $\phi = \alpha L(L-1)/2$ проверок условия (4), где α - число просмотров матрицы, выполненных для проверки условия (5). Количество перестановок строк может быть существенно меньше ϕ.

Итак, строки 2^i и 2^j матрицы Z переставляются, если

$$z(i,i) + z(j,j) - z(i,j) - z(j,i) < 0. \qquad (4)$$

След матрицы считается максимизированным, если

$$\forall i,j = \overline{1,L}\ (z(i,i) + z(j,j) - z(i,j) - z(j,i) \geq 0). \qquad (5)$$

Обмены между ветвями р-программы считаются минимизированными, если для всех квантов времени составленного расписания выполняется условие (5).

После минимизации обменов между ветвями, задаваемых матрицей Y_1, расписание примет вид:

$$Y_1' = \begin{pmatrix} 5 & 1 \\ 0 & 2 \\ 4 & 3 \end{pmatrix}.$$

В окончательном варианте (Y_1') требуется выполнить два обмена, в то время как в исходном распределении (Y_1) их требовалось 1+3+5=9 (см. программу на ЯРС в разделе 2).

10. Эффективность метода оптимизации

С помощью моделирования проверена эффективность эвристического алгоритма максимизации следа матрицы. На рис.3 приведен график зависимости $\alpha(L)$ - математического ожидания числа просмотров матрицы - в зависимости от ее размерности L. За один просмотр неравенство (5) может быть недостижимо. В частности, для размерности ?x? число просмотров α = 2,06, т.е. требуется 2·?·(?-1)/2 сравнений, что в 120 раз быстрее перебора.

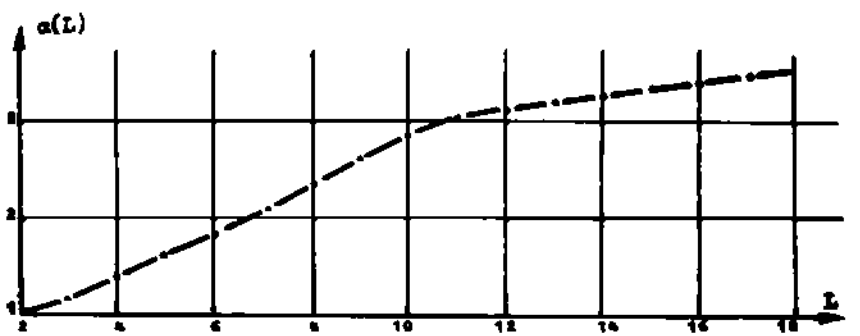


Рис. 3

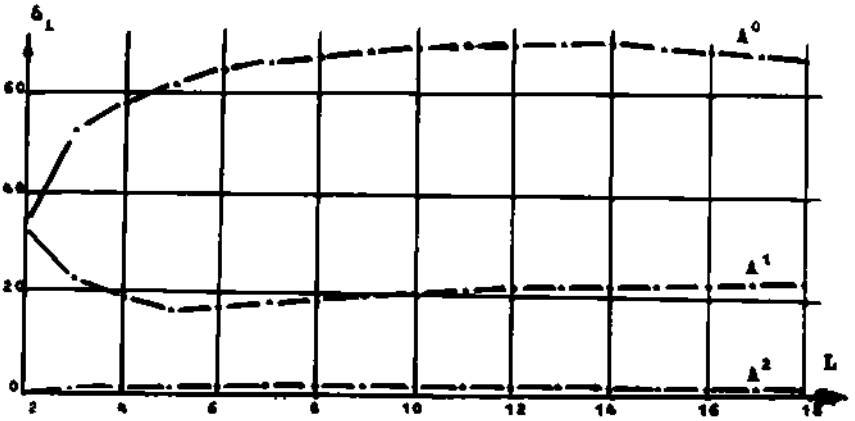


Рис. 4

Рис.4 иллюстрирует математическое отклонение максимума следа (получаемого различными эвристическими методами) от оптимального, где $\delta_L = [M(S_0 - S(A^1(Z)))/S_0] \cdot 100\%$. Здесь S_0 – оптимальное значение следа; $S(A^1(Z))(i=0,1,2)$ – значение следа матрицы Z после применения к ней i-го преобразования; A^0 соответствует тождественному преобразованию матрицы; A^1 – преобразование, при котором выполняется неравенство (5); A^2 – преобразование, при котором, начиная с первого столбца, максимальный элемент ставится на диагональ.

Например, для L = 7 отклонения δ_i составляют: $\delta_0 = 65,24\%$; $\delta_1 = 1,44\%$; $\delta_2 = 19,47\%$.

Комбинация преобразований A^1 и A^2 позволяет еще больше приблизиться к оптимальному значению следа:

$$\delta_{1(2)} = \left[M \left(\frac{S_0 - \max \left(S(A^1(A^2(Z))), S(A^1(Z)) \right)}{S_0} \right) \right] \cdot 100\% \leq 0,2\%.$$

При моделировании проводилось по тысяче экспериментов для каждого значения L. Использовался датчик случайных чисел с равномерным законом распределения. Матрица Z строилась следующим образом ($L = \overline{2,18}$):

I) для каждой строки i выбрасывается число $k_i \leq L$ ненулевых элементов;

2) выбрасываются места $j_1,\ldots,j_{k_i}$ ненулевых элементов строки i ($j_m \le L$, $m = \overline{1,k_i}$);

3) значения $z_1,\ldots,z_{k_i}$ ненулевых элементов матрицы Z выбрасываются и ставятся на места $j_1,\ldots,j_{k_i}$ соответственно ($0 < z_m \le 1000$, $m = \overline{1,k_i}$).

Указанным способом заполняются все строки матрицы Z, затем матрица дублируется и обрабатывается указанными преобразованиями.

II. Выдача расписания

Результатом работы системы ЗРА является расписание реализации подалгоритмов по ветвям параллельной программы, в котором указывается:

- относительное время запуска блоков;
- время простоя ЭМ при реализации расписания;

```
                ПРОГРАММА   ВОТ

  КОЛ-ВО БЛОКОВ  5      КОЛ-ВО ВЕТВЕЙ   2
СРЕДНЕЕ ВРЕМЯ РЕАЛИЗАЦИИ БЛОКА   5+-5   ПРОЦЕНТОВ

     ==========================================

                    ВЕТВЬ    1
       ИМЯ  БЛОКА      ВРЕМЯ ЗАПУСКА      ВРЕМЯ РЕШЕНИЯ

         A5                0                  5
         A4                5                  5
                           10                 5
КОЛ-ВО БЛОКОВ В ВЕТВИ  2, ВРЕМЯ ЗАНЯТОСТИ  15, ПРОСТОЯ -    5


     ==========================================

                    ВЕТВЬ    2
       ИМЯ  БЛОКА      ВРЕМЯ ЗАПУСКА      ВРЕМЯ РЕШЕНИЯ

         A1                0                  5
         A2                5                  5
         A3                10                 5
КОЛ-ВО БЛОКОВ В ВЕТВИ  3, ВРЕМЯ ЗАНЯТОСТИ  15, ПРОСТОЯ -    0
```

Рис. 5. Расписание при $T_i = 5$, $i = \overline{1,5}$.

ПРОГРАММА ВОТ

КОЛ-ВО БЛОКОВ 5 КОЛ-ВО ВЕТВЕЙ 2

ВЕТВЬ I

ИМЯ БЛОКА	ВРЕМЯ ЗАПУСКА	ВРЕМЯ РЕШЕНИЯ
A1	0	2
A2	2	3
A3	5	5

КОЛ-ВО БЛОКОВ В ВЕТВИ 3, ВРЕМЯ ЗАНЯТОСТИ 10, ПРОСТОЯ - 0

ВЕТВЬ 2

ИМЯ БЛОКА	ВРЕМЯ ЗАПУСКА	ВРЕМЯ РЕШЕНИЯ
A5	0	3
———	3	2
A4	5	4

КОЛ-ВО БЛОКОВ В ВЕТВИ 2, ВРЕМЯ ЗАНЯТОСТИ 9, ПРОСТОЯ - 2

Рис. 6. Расписание при $T_1 = 2$, $T_2 = T_3 = 3$, $T_4 = 5$, $T_5 = 4$.

- время выполнения блоков;
- время счета ЭМ для каждой ветви.

Расписание реализации блоков в ветвях р-программы отражает статическую схему распределения, удовлетворяя информационно-временным связям между блоками и предполагаемому времени выполнения каждого блока. Динамика реализации ветвей будет соответствовать статическому расписанию, если точно указано время решения каждого блока и эффективно заданы порядок пересылки данных и синхронизация ветвей параллельной программы на основе макрооператоров системных взаимодействий [4,5].

Примеры расписаний, выдаваемых системой ЭРА для матриц Y_1' и Y_2 (алгоритм рис.I), представлены на рис.5 и 6 соответственно. При отладке программы ей присвоено имя ВОТ.

При реализации экспериментального распараллеливателя алгоритмов (системы ЭРА) решен ряд задач по рациональному составлению расписания выполнения блоков по ветвям параллельной программы. Вырабатываемое системой ЭРА расписание реализации вычислительного процесса может быть применено для любой однородной вычислительной системы. Пользоваться системой ЭРА можно как в автономном режиме, при котором пользователю выдаются рекомендации о распределении блоков для составления эффективного расписания, так и в качестве модуля операционной системы для оперативного планирования и распараллеливания потока задач.

Блок трансляции запрограммирован на МИНИКОДЕ М-6000, остальные блоки – на ФОРТРАНе. Общий объем системы ЭРА, включая библиотеку, 13000_8 команд. Размер М оперативной памяти ЭМ, в которой работает ЭРА, должен удовлетворять неравенству:

$$N^2 + N \cdot L + L^2 + 6N + 3L + 13000_8 + C \le M,$$

где С – объем драйверов ввода/вывода в основной управляющей системе. Базовая реализация имеет ограничения на N и L: $N \le 32, L \le 5$. В работе [6] описаны действия по перенастройке системы ЭРА для увеличения значений N и L.

Литература

1. ВИНОКУРОВ В.Г., ДИМИТРИЕВ Ю.К., ЕВРЕИНОВ Э.В., КОСТЕЛЯНСКИЙ В.М., ЛЕХНОВА Г.И., МИРЕНКОВ Н.Н., РЯЗАНОВ В.В., ХОРОШЕВСКИЙ В.Г. Однородная вычислительная система из мини-машин. –Вычислительные системы, 1972, вып. 51. с.127–145.

2. КЕРБЕЛЬ В.Г., МИРЕНКОВ Н.Н. Автоматическое распараллеливание алгоритмов специального вида. –Вычислительные системы. Теории однородных вычислительных систем, 1975, вып.63. с.149–158.

3. ЕВРЕИНОВ Э.В., КОСАРЕВ Ю.Г. Однородные вычислительные системы высокой производительности. –Новосибирск: Наука, 1968. –308 с.

4. КОЛОСОВА Ю.И. Языки для записи параллельных процессов. Операторы группового взаимодействия. –Новосибирск, 1977, Отчет ИМ СО АН СССР.

5. КЕРБЕЛЬ В.Г., КОРНЕЕВ В.Д., КРЫЛОВ Э.Г. Языки для описания параллельных процессов. Операторы индивидуальных взаимодействий. –Новосибирск, 1977. Отчет ИМ СО АН СССР.

6. КЕРБЕЛЬ В.Г. Экспериментальный распараллеливатель алгоритмов. Версия I. –Новосибирск, 1977, Отчет ИМ СО АН СССР.

7. КОНСТАНТИНОВ В.И. Металанк ТВ-грамматик. –Кибернетика, 1974, № 6, с.84–89.

8. ВАЛЬЦИЦКИЙ И.В. Метаязык 2-грамматик.-Кибернетика, 1973, №3, с.47-63.

9. ВАХБАЗИН К.В., ТУЛЯКОВА Т.А. Обзор методов составления расписаний для многопроцессорных систем. -Зап. науч.семинаров ЛОМИ, 1975, т. 54, с.229-258.

10. МИРЕНКОВ И.Н. Алгоритмы планирования для диспетчера однородной вычислительной системы.-Вычислительные системы, 1970, вып.42. с. 34-46.

Поступила в ред.-изд.отд.
20 марта 1978 года