ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.КЕЛДЫША
РОССИЙСКОЙ АКАДЕМИИ НАУК

В.М.Михелев

ПРОК - ПРОцессоры в Кольце запросов.
(Архитектура многопроцессорного комплекса)

МОСКВА
2000

УДК 681.31

Резюме

Описывается вариант multithreading архитектуры вычислительных машин, когда поток вычислений управляется передачей данных - значений объектов и признаков синхронизации. Передача данных процессору, выбранному для выполнения очередной цепочки команд, сопровождается передачей регистрового контекста, в котором эти данные были получены. Для общения процессоров с памятью и между собой, а также для выбора свободного процессора используются кольцевые коммутаторы.

Abstract

A variant of computer multithrerading architecture is described. To control of program execution the computer uses

instructions of data passing. The data are passed together with a register environment. To interact with memory and anather processor and to look for a free processor it is used ring commutators.

1. ВВЕДЕНИЕ.

Почти двадцать лет тому назад Деннис сформулировал идею потоковой - dataflow машины [1,2], в которой последовательность выполнения инструкций определялась не последовательностью их записи, а динамически задаваемым потоком данных. Такой подход давал возможность распараллелить вычисления на уровне инструкций, циклов, процедур, то есть обеспечивал низко уровневый - fine grain параллелизм. Однако плата оказалась чрезмерно высокой [3]. Отметим лишь некоторые трудности, связанные с реализацией потоковых машин. С одной стороны было отмечено увеличение в 2 - 3 раза необходимого количества инструкций по сравнению с машинами фон-неймановской архитектуры. Далее, поскольку каждое данное в памяти должно иметь имя, сохраняющее свою уникальность во время решения задачи, то для задач, обрабатывающих большой объем информации, проблема задания такого имени может стать непреодолимым препятствием. Существенной проблемой для потоковых машин является также необходимость в большой ассоциативной памяти, реализовать которую на существующей технологической базе по меньшей мере сложно. Кроме того, поскольку данные в виде токенов передаются без регистрового контекста, а команды выполняются "хаотично", то разумное использование регистров в процессе вычисления практически невозможно, что, естественно, увеличивает количество обращений к памяти. В литературе отмечаются также трудности, связанные с реализацией критических областей.

В настоящее время считается более перспективным multithreading [3,4] - реализацию fine grain параллелизма путем вычисления на одном процессоре последовательности команд программы - thread. Последовательность обычно кончается командой, для выполнения которой в данный момент не хватает данных. Началом же последовательности служит либо команда разветвления вычислений, либо команда, ожидающая данное. Параллелизм при таком подходе возникает за счет одновременного вычисления нескольких последовательностей.

В работе рассматривается вариант mutithreading архитектуры, решающий следующие проблемы:

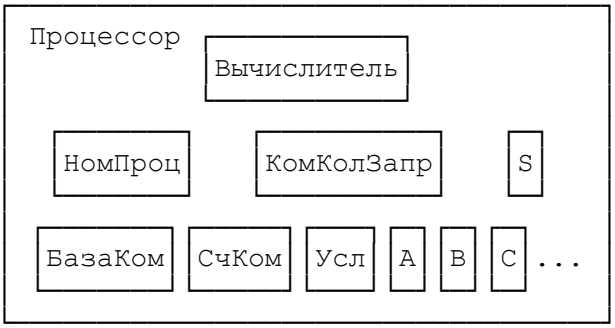
- выбор свободного процессора и обращение к памяти из множества процессоров:
- передачу выбранной для вычисления последовательности того регистрового контекста, в котором выполнялась команда, инициирующая вычисление этой последовательности;
- синхронизацию, связанную с защитой критических областей;
- общение между процессорами.

В предлагаемой архитектуре последовательность вычислений определяется передачей данных: значений операндов и признаков синхронизации. Распараллеливание реализуется автоматически командами передачи данных. При обращении к объектам памяти учитывается значение тега, разрешающего записывать, читать или изменять объект.

2. ЭЛЕМЕНТЫ АРХИТЕКТУРЫ.

2.1. Процессор.

Компьютер содержит N физических процессоров и M виртуалбных, количество которых определяет возможный параллелизм вычислений.



Каждый физический процессор содержит:

Вычислитель - устройство для выполнения операций,

БазаКом - база команд, добавляется к СчКом при выборе команды,

S - регистр результата,

СчКом - счетчик адреса команд,

Усл - признак результата - может быть использован лишь во время работы процессора, в котором он установлен,

A,B,C,... - регистры, могут использоваться для хранения данных во время работы процессора. Размер регистра - четыре байта.

НомПроц - каждому физическому процессору в процессе вычислений присваивается номер того виртуального процессора, который в данный момент замещается физическим. Исходное значение номера - ноль.

КомКолЗапр - регистр команды, переданной каким-либо процессором для исполнения данному через кольцо запроса.

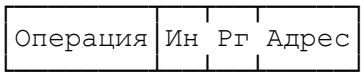
2.2. Памяти.

В дальнейшем предполагается существование двух видов памятей:

- локальная память команд процессора. Эта память не базируется.

- общая память команд и данных. В качестве базы команд используется значение регистра A, которое запоминается в БазаКом выбранного свободного процессора командами, распараллеливающими вычисления, а базы данных - регистр B.

Команды программы могут быть разбиты на две группы. Часто повторяемые, например, некоторые циклы или подпрограммы загружаются в локальные памяти процессоров. Эти части программы для всех процессоров одинаковы. Остальная программа загружается в общую память. Формат команды:



Все команды одноадресные, но в большинстве своем двухоперандные.

Первый операнд - результат выполнения текстуально предыдущей команды.

Второй операнд

- либо значение, выбираемое по указанному в команде адресу,

- либо адрес команды, которой передается результат или признак,

- либо литерал,

- либо регистр.

Ин - индикатор способа адресации. Индикатор содержит

признаки:

a - признак автоматического базирования. Если это адрес команды передачи данных, то значение регистра A заносится в БазаКом выбранного свободного процессора, если данных, то он автоматически базируется регистром B.

c - адрес в локальной памяти команд процессора. Этот адрес не базируется.

w - при выполнении команды объект в памяти рассматривается как рабочий, используемый для реализации рандеву двух операндов. При обращении к памяти адрес автоматически базируется регистром B.

d - косвенная адресация в команде передачи значения, использующая дескриптор. Адрес самого дескриптора задается в команде парой регистр-адрес. Дескриптор - три элемента памяти. В первом элементе указывается адрес команды, которой передается значение или признак, во втором - база программы, а в третьем - база соответствующих программе данных. При передаче данных по дескриптору в тот момент, когда запрашивается свободный процессор, устанавливается база программы, то ес
овторяется. Если
же не совпадают теги, то физический процессор освобождается, а соответствующий ему виртуальный переходит в состояние поиска со счетчиком адреса той команды программы, которая выдала запрос.

При обращении к другому процессору запрос считается нереализованным в следующих ситуациях:

- отсутствует физический процессор, которому передается команда, то есть он находится среди виртуальных,

- в принимающем команду процессоре регистр КомКолЗапр занят.

В первом случае запрос попадает в очередь нереализованных запросов и затем возвращается процессору, выдавшему этот запрос. Последний обращается к кольцу запросов процессоров и в тот момент, когда искомый виртуальный процессор находится в соответствующей ему позиции, переводит его из состояния ожидания, если он в этом состоянии находился, в состояние поиска. При этом сам он переходит в виртуальное состояние поиска. Во втором случае принимающий команду процессор возвращает ее в кольцо запросов.

В приведенной кольцевой схеме время передачи команды внутри кольца от позиции к позиции должно быть, очевидно, не меньше времени обращения к кэш памяти.

В том случае, когда объект, участвующий в операции, состоит из нескольких элементов памяти, например, двойная точность или последовательность байтов процессор выполняет несколько запросов и образует из них результат.

Из приведенной схемы видно, что положение процессоров в кольце по отношению к памяти неравнозначно: запрос от ближайшего к памяти процессора проходит лишь одну позицию в то время как от наиболее удаленного - количество позиций, равное числу процессоров.

Недостатком чисто кольцевых схем является увеличение времени обращния к памяти на количество тактов, необходимых для передачи запроса от процессора к памяти. Затрачиваемое время увеличивается пропорционально увеличению числа процессоров. Кроме того в том случае, когда в соответствующей памяти позиции кольца нет запроса к памяти, например, позиция свободна или занята обращением к процессору, а в кольце такие запросы есть, то память выполняет холостой цикл.

Эта проблема может быть решена, например, при помощи древовидного кольцевого коммутатора. Ниже представлен один из

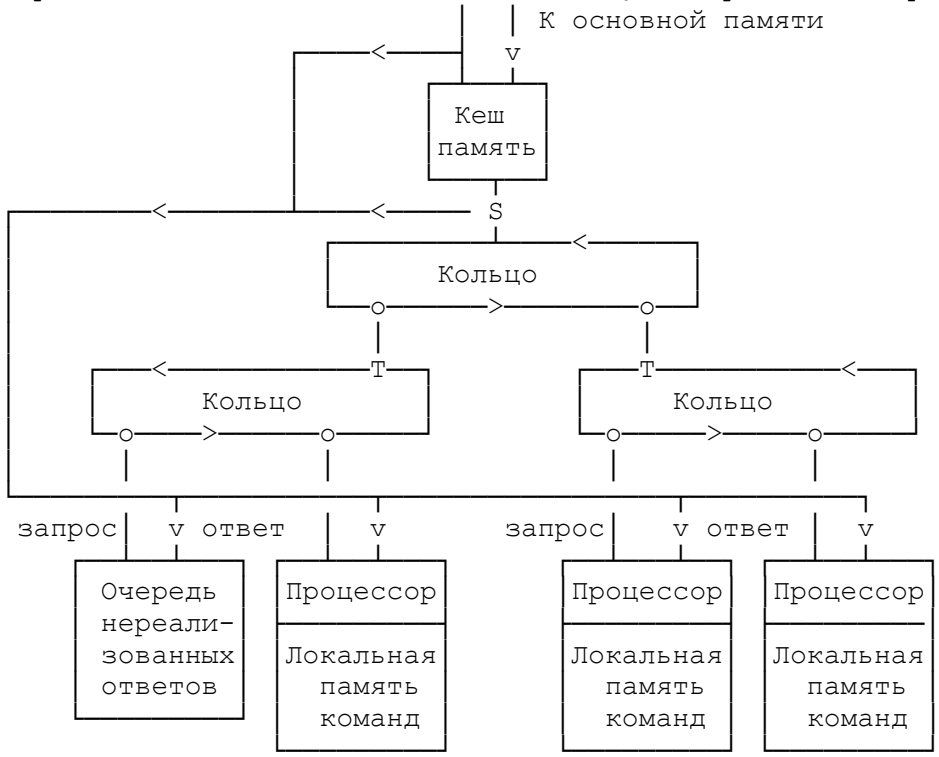
возможных вариантов решения проблемы. На схеме для простоты представлено только три процессора.

Здесь

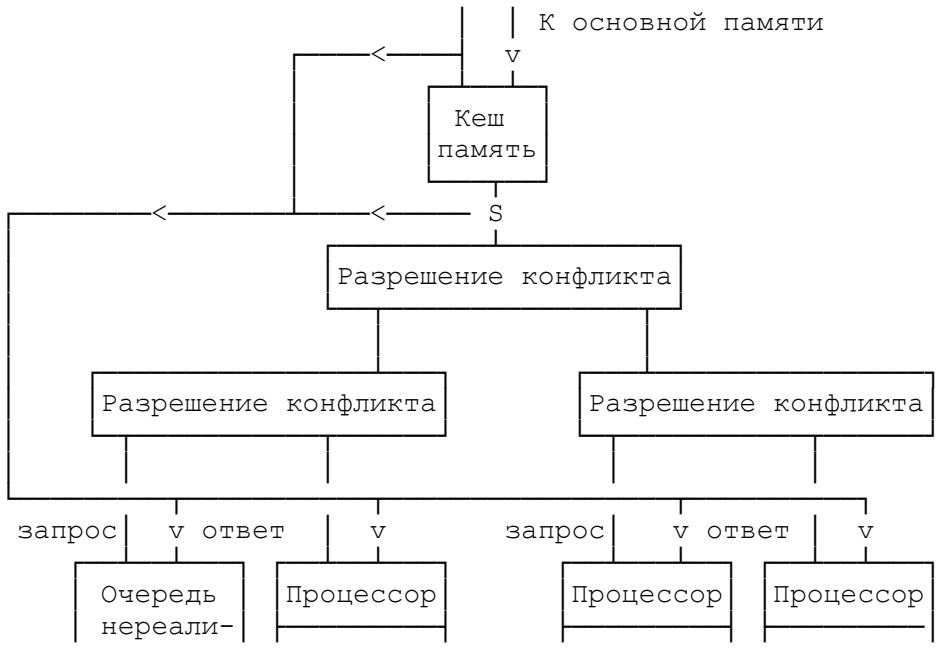
T - позиция передачи данных следующему кольцу, когда соответствующая позиция свободна;

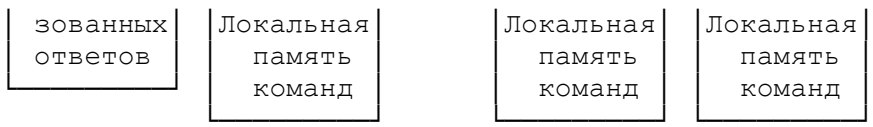
S - переключатель, передающий команду кольца либо памяти, либо процессору, то есть непосредственно в шину ответов.

Для таких схем время задержки пропорционально не количеству процессоров, а его логарифму. Очевидно, что через кольцо может передаваться не вся команда запроса, а лишь номер процессора, а остальная информация может коммутироваться непосредственно на входе памяти, используя выбранный номер.



Дальнейшее ускорение работы подобных схем может быть достигнуто за счет замены колец потенциальными устройствами для разрешения конфликтных ситуаций между, например, двумя запросами.





Возможен еще один вариант ускорения обращения к памяти за счет увеличения числа блоков памяти. В этом случае в каждом из блоков, очевидно, должны хранится объекты, адреса которых не совпадают, например, отличающиеся по некоторому модулю.

2.4. Взаимодействие процессоров.

Процессор может выполнять операции над регистрами другого процессора, номер которого ему известен. В команде, выполняющей эту операцию, адрес должен иметь индикатор "p". Выполнение команды сводится к выдаче в кольцо запросов команды с соответствующим кодом операции и номером процессора, которому эта команда предназначена. В этом процессоре команда попадает в регистр команды кольца запросов - КомКолЗапр при условии, конечно, что он свободен, и выполняется после выполнения команды wait. Если же этот регистр занят, то команда остается в кольце запросов и повторяется процессором, ее выдавшим.

2.5. Команды.

Ниже приведен не полный перечень команд, а лишь те, которые используются в программах для выполнения арифметических, логических операции и операций передачи данных. Все команды делятся на семь групп.

1. Получающие значения, которые в качестве значений операндов используют результат выполнения текстуально предыдущей команды и значение по адресу. Последнее может быть передано командой передачи значения. Результат, если специально не оговорено, остается в регистре результата S.

В операциях, мнемоника которых начинается с r, первым операндом считается операнд, выбираемый по адресу в команде. Результат записывается по этому же адресу. Очевидно, что в таких командах выполняется два обращения к памяти, одно из которых - чтение, а другое - запись. Для того, чтобы при этом в интервале между чтением и записью объект не был бы прочитан или изменен, используется команда кольца - чтение с изменением тега на cl, а соответствующей ей команде записи разрешается писать с таким тегом.

add rand sub rsub mult rmult div rdiv mod - арифметические команды с целыми аргументами.

fadd rfadd fsub rfsub fmult rfmult fdiv rfdiv - арифметические команды с дейсвтвительными аргументами.

or ror and rand not - логические команды. В команде not второй операнд отсутствует.

ren - результатом выполнения этой команды служит результат текстуально предыдущей команды. Второй операнд - признак синхронизации. Команда используется для синхронизации работы отдельных частей программы и в силу этого в адресе должен быть указан адрес рабочего объекта.

emp - результатом служит результат предыдущей выполненной команды, которая может быть как текстуально предыдущей, так и передающей значение или признак.

nop - результат - значение второго операнда, которое передается команде командами передачи значения. Первый операнд игнорируется. Обычно команда nop служит для организации входа, через который передается значение следующей за nop команде. Команда выполняется вслед за командой, передающей ей значение.

gener - генератор значений. Команда gener используется для параллельного выполнения команд тела цикла с параметром в

интервале: исходное значение регистра C .. 0. Для каждого значения параметра тело цикла выполняется независимо на собственном процессоре. Номер выбранного процессора, как указано выше, зависит от значения Усл.

Команда выполняется следующим образом. Если значение в регистре C больше или равно нулю, запрашивается свободный процессор, в который устанавливается счетчик адреса не единицу больше адреса команды gener, в его регистр результата переписывается значение регистра C, а остальные регистры принимают то значение, которое они имели в процессоре, выполняющем команду gener. Затем значение в регистре C уменьшается на единицу и, если оно снова не отрицательно, команда повторяется. Повторение может быть прервано в случае появления команды в регистре КомКолЗапр. В этой ситуации сначала выполняется команда кольца запроса, а затем снова, если это необходимо повторяется команда gener. Когда значение регистра C становится отрицательным, в Усл заносится 1 и процессором выполняется команда по адресу, заданному в команде gener.

Если в момент очередного запроса свободный процессор отсутствует, то после выдачи этого запроса выдается дополнительный запрос процессора со счетчиком адреса команды gener. После чего команда завершается с освобождением процессора. Команда gener, очевидно, будет в дальнейшем продолжена после того как дополнительный запрос захватит освободившийся процессор.

2. Передающие значение, которые передают полученное в качестве первого операнда значение или признак синхронизации команде по адресу и, возможно, текстуально следующей. При передаче по адресу при наличии одновременно передачи текстуально следующей команде запрашивается свободный процессор, поскольку текущий выполняет текстуально следующую команду.

Команды этой группы позволяют распараллелить вычисления. Мнемоника команд передачи данных.

hA[(N ! C)], где

A={r,v} - передача по адресу в команде.

r - передается признак синхронизации,

v - передается значение первого операнда.

N={r,v} - передача текстуально следующей команде. Наличие признака r означает лишь, что будет выполнена текстуально следующая команда. При отсутствии N свободный процессор не запрашивается, а вычисления продолжаются на текущем процессоре.

C={c,nc} передача по адресу при наличии условия. Свободный процессор не запрашивается.

c - передача происходит, когда в Усл занесена 1,

nc - когда в Усл занесен 0.

При невыполнении условия следующей команде передается первый операнд. Например,

hvv - передача первого операнда команде по адресу и текстуально следующей.

hr - передача признака по адресу. Текстуально следующая команда не выполняется.

hvc - значение передается команде по адресу, когда в Усл занесена 1, в противном случае - текстуально следующей команде.

3. Устанавливающие отношение между операндами.

eq ge ne le lt gt - после выполнения операции отношения в случае false в Усл заносится 0, в случае true - 1. На регистре результата остается значение первого операнда.

4. Присваивающие результат значение.

ass - результат выполнения текстуально предыдущей команды и тег Rd, если адрес относится к общей памяти, заносятся в

объект по заданному в команде кадресу. Команда не может быть выполнена, если до записи тег в памяти был отличен от Cl или Wr, и запрос как неудовлетворенный возвращается процессору.

gtv - результатом становится значение операнда по адресу в команде. Если адресуется элемент памяти, то его тег должен быть равен Rd. В противном случае запрос возвращается процессору как неудовлетворенны. Первый операнд игнорируется.

prnm - значение НомРег присваивается по адресу.

cond - значение по адресу присваивается Усл.

5. Изменяющие значение базовых регистров. Выполнение операции над регистрами не изменяет значение регистра результата. На нем сохраняется значение первого операнда. Пусть R={A,B,C,...}.

svR - сохранение регистра. Значение указанного в операции регистра и признак Rd (если запись в память) заносятся по заданному в команде адресу. Если адресуется элемент памяти, то его тег должен быть равен Cl или Wr В противном случае запрос возвращается процессору как неудовлетворенный.

ptR - восстановление значения регистра. В указанный в операции регистр устанавливается значение по адресу в команде. Если адресуется элемент памяти, то тег должен быть равен Rd. В противном случае запрос возвращается процессору как неудовлетворенный.

incR - увеличение значения регистра на единицу.

decR - уменьшение значения регистра на единицу.

addR,subR,multR,divR - выполняется операция над регистром и значением по адресу в команде. Результат операции записывается в регистр. Если адресуется элемент памяти, то тег должен быть равен Rd. В противном случае запрос возвращается процессору как неудовлетворенный.

push - сохранение всех регистров в памяти. Адрес памяти задается во втором операнде.

pop - восстановление всех регистров из памяти. Адрес памяти задается во втором операнде.

6. Команды изменяющие тег.

clwr rdup uprd rdcl - команда не выполняется, если состояние тега не соответствует требованию команды. В этом случае процессору возвращается признак Усл равный 1, в противном случае - 0. Это позволяет использовать эти команды для проверки состояния памяти.

7. Команды, прерывающие работу процессора. Команды этой группы не изменяют значение регистра результата.

wait - процессор находится в состоянии ожидания до тех пор пока не появится команда в регистре КомКолЗапр. Эта команда выполняется и управление передается следующей за wait команде.

stop - переход процессора в пул свободных процессоров.

3. ПРИМЕРЫ И ПРИЕМЫ ПРОГРАММИРОВАНИЯ.

Во всех приведенных ниже примерах будем считать, что объекты в памяти адресуются с точностью до слова. Все примеры записываются на языке некоторого гипотетического ассемблера. Поля в предложениях ассемблера называются:

Опер - операция,
И - индикатор способа адресации,
Р - идентификатор регистра,
Адр - поле адреса.

3.1. Пример цикла с выходом по условию.

Сначала рассмотрим следующий простой пример.

```
MM: array[0..1000] of integer;
i: integer:=0;
Sum: integer:=0;
  ...
while i*i<NN
 loop
   Sum+=MM[i];
   i+=2
 end loop;
```

Предполагается, что программа установлена в общей памяти процессоров. Напомним, что индикатор "a" означает автоматическое базирование команд в общей памяти БазаКом, а данных - регистром B.

	Опер	И	Р	Адр	Комментарий
	ptC	l		0	0 -> i, i в регистре C
	gtv	l		0	
	ass	a		Sum	0 -> Sum, Rd -> тег Sum
	RdUp	a		Sum	Разрешение изменения Sum
Cycle	emp				
	gtv	r		C	Получение i*i в S
	mult	r		C	
	ge	a		NN	
	hrc	a		Exit	Выход по концу цикла
	addC	l		2	i <- i+2
	hrr	a		Cycle	Захват свободного процессора
	gtv	a	C	MM-2	Чтение MM[i]
	radd	a		Sum	Накопление в Sum
	stop				Возврат процессора

Следует иметь в виду, что в момент выхода по концу цикла сумма в Sum может быть еще не накоплена. Для того, чтобы значением Sum можно было воспользоваться, необходимо определить этот момент и изменить в Sum тег. Один из вариантов решения этой проблемы приведен в следующем примере.

3.2. Перемножение матриц с определением конца выполнения операции.

Рассмотрим пример - перемножение матриц.

```
MM,NN,RR: array[1..20] of integer;
 ...
for i in 1..20
  loop
    for j in 1..20
      loop sum:=0;
        for k in 1..20
          loop RR[i,j]+=MM[i,k]*NN[k,j]
          end loop;
      end loop
  end loop
```

Для того, чтобы зафиксировать момент окончания перемножения матриц, будем накапливать в регистре E процессора, инициализирующего сам процесс перемножения матриц, общее количество сложений (20*20*20), выполняемых при вычислении всех элементов матрицы. Номер этого процессора будем хранить в регистре G всех процессоров, участвующих в процессе вычисления. Очевидно, что его значение будет передаваться всякий раз при запросе очередного процессора командой gener. Будем считать, что программа расположена в локальных памятях процессоров (индикатор

C). Поскольку взаимодействие процессоров, помимо указанного выше для определения конца перемножения матриц, не требуется, то можно выбирать процессоры с произвольным номером. Для этого перед командой gener будем присваивать нулевое значение регистру D. В дальнейшем в комментариях используется запись (Рег,Проц), которая означает "регистр Рег процессора, номер которого хранится в регистре Проц".

```
        | push  |a|  | SvAr | Сохранение регистров в SvAr
        | prnm  |r|  |   G  | Занесение номера процессора в G
        | ptE   |l|  | 8000 | 20*20*20
        | ptC   |l|  |   19 | Цикл по i 19..0, 19 -> C
        | cond  |l|  |    0 | Запрос процессора с произв. номером
        | gener |c|  |  End | Ожидание конца перемножения матриц
        | multC |l|  |   20 | i в регистре C
        | ptH   |r|  |    C | i*20 в регистре H
        | ptC   |l|  |   19 | Цикл по j 19..0, 19 -> C
        | cond  |l|  |    0 | Запрос процессора с произв. номером
        | gener |c|  | RtPr |
        | ptE   |r|  |    C | j в регистре E
        | ptF   |r|  |    H |
        | addF  |r|  |    E | i*20+j в F адрес [i,j]
        | gtv   |l|  |    0 |
        | ass   |a|F |   RR | 0 -> RR[i,j], Cl -> Rd
        | rdup  |a|F |   RR | Разрешение накопления в RR[i,j]
        | ptC   |l|  |   19 | Цикл по k 19..0, 19 -> C
        | cond  |l|  |    0 | Запрос процессора с произв. номером
        | gener |c|  | RtPr |
        | addH  |r|  |    C | i*20+k, k в регистре C
        | ptH   |a|H |   MM | MM[i,k]  в регистр H
        | multC |l|  |   20 |
        | addC  |r|  |    E | k*20+j
        | gtv   |a|C |   NN | Выборка NN[k,j]
        | mult  |r|  |    H | NN[k,j]*MM[i,k]
        | radd  |a|F |   RR | Накопление суммы в RR[i,j]
        | decE  |p|G |      | (E,G) счетчик конца перемн матриц
        | stop  | |  |      |
RtPr    | nop   | |  |      |
        | stop  | |  |      | Возврат процессора
End     | nop   | |  |      |
        | wait  | |  |      | Ожидание выполнения команды decE
        | eqE   |l|  |    0 |
        | hrnc  |c|  |  End |
        | pop   |a|  | SvAr | Восстановление регистров из SvAr
              ...             Конец перемножения матриц
```

Накопление суммы в памяти данных при помощи команды radd существенно увеличивает время, необходимое для вычисления элемента матрицы R[i,j]. Можно, однако, для накопления суммы воспользоваться регистром процессора, инициирующего вычисление элемента матрицы, то есть процессора, выполняющего самую внутреннюю команду gener. Его номер будем хранить в регистре F. Введем следующие изменения. Сумму будем накапливать в регистре (A,F), поскольку он не используется для базирования, а количество еще не использованных слагаемых в регистре (D,F).

Ниже приведено распределение регистров в процессорах, выполняющих команды gener.

В цикле по i.
- C - i,
- H - i*20,
- G - номер процессора, задающего вычисление всей матрицы.

В цикле по j.

A - элемент матрицы, накопление суммы,
D - счетчик числа слагаемых,
H - i*20,
E - j,
F - номер процессора, вычисляющего элемент матрицы,
G - номер процессора, задающего вычисление всей матрицы,
матрицы.
В цикле по k.
C - k,
H - i*20,
E - j,
F - номер процессора, вычисляющего элемент матрицы,
G - номер процессора, задающего вычисление всей матрицы,

	push	a		SvAr	Сохранение регистров в SvAr
	prnm	r		G	Занесение номера процессора в G
	ptE	l		400	20*20 число элементов матрицы
	ptC	l		19	Цикл по i 19..0, 19 -> C
	cond	l		0	Запрос процессора с произв. номером
	gener	c		End	
	multC	l		20	i в регистре C
	ptH	r		C	i*20 в регистре H
	ptC	l		19	Цикл по j 19..0, 19 -> C
	cond	l		0	Запрос процессора с произв. номером
	gener	c		RtPr	
	ptE	r		C	j в регистре E
	ptC	l		19	Цикл по k 19..0, 19 -> C
	cond	l		0	Запрос процессора с произв. номером
	ptD	l		20	Счетчик числа слагаемых
	ptA	l		0	В A - элемент матрицы
	prnm	r		F	Занесение номера процессора в F
	gener	c		EndE	
	addH	r		C	i*20+k, k в регистре C
	ptH	a	H	MM	MM[i,k] в регистр H
	multC	l		20	
	addC	r		E	k*20+j
	ptC	a	C	NN	Выборка NN[k,j]
	multC	r		H	NN[k,j]*MM[i,k]
	addA	p	F	C	Накопление суммы в (A,F)
	decD	p	F		(D,F) количество неиспользованных
	stop				слагаемых
EndE	nop				
	addH	r		E	i*20+j, в H адрес [i,j]
	wait				Ожидание выполнения команды decD
	eqD	l		0	
	hrnc	c		EndE	
	gtv	r		A	
	ass	a	H	RR	A -> RR[i,j], Cl -> Rd
	decE	p	G		(E,G) количество невычисленных
	stop				элементов
RtPr	nop				
	stop				Возврат процессора
End	nop				
	wait				Ожидание выполнения команды decE
	eqE	l		0	
	hrnc	c		End	
	pop	a		SvAr	Восстановление регистров из SvAr
					Конец перемножения матриц

3.3 Упорядочивание массива целых чисел.

Ниже приведена программа, реализующая следующий алгоритм

упорядочивания по возрастанию элементов массива целых чисел. Массив разбивается на пары, начиная с первого элемента, и каждая пара упорядочивается индивидуально. Затем массив вновь разбивается на пары, но начиная не с первого элемента, а со второго, и вновь производится упорядочивание внутри каждой пары. Следующий цикл разбиения начинается опять с первого элемента и так далее. Легко видеть, что для упорядочивания потребуется количество циклов, не превышающее размер массива.

Реализующая алгоритм программа призвана проиллюстрировать возможность обмена данными между процессорами. Каждой паре выделяется процессор, который сначала считывает в свои регистры соответствующую пару из массива в памяти, упорядочивает элементы пары и затем в случае четной итерации левый элемент пары заносит в регистр неполной пары, правый элемент пары переписывает в регистр левого элемента и запрашивает правый элемент из регистра неполной пары у процессора, упорядочивающего следующую пару. Номер этого процессора известен, поскольку генератор, запрашивающий процессоры, выдает им последовательные номера. В случае нечетной итерации производятся аналогичные действия, но в обратном направлении. Если число элементов массива четное, то для последней пары заносится величина большая или равная максимальному значению элемента в упорядочиваемом массиве.

Предлагаемая программа построена таким образом, что массив читается только один раз перед началом цикла и записывается, когда его элементы уже упорядочены. Все перестановки производятся с использованием регистров.

В программе используются следующие константы:

Cnt - количество элементов массива M.

PrCnt=Cnt/2 - количество пар элементов. Равно числу одновременно работающих процессоров.

Even - {0,1} - четность числа элементов массива.

Max - величина большая или равная максимальному значению элемента в массиве.

Регистры распределены следующим образом:

C - номер предыдущего процессора,
D - левый элемент пары,
E - правый элемент пары,
F - регистр неполной пары,
G - номер следующего процессора,
H - количество перестановок.

```
      | ptC    |l|  |PrCnt  |
      | cond   |l|  |    1  | Процессорам выделяются посл. номера
      | gener  |c|  | EndE  | Цикл по i={PrCnt..0},i в регистре C
      | svC    |r|  |    F  | Чтение i*2 и i*2+1 переставляемых
      | multF  |l|  |    2  |  элементов M в регистры D и E
      | ptD    |a|F |    M  | M[i] -> D - левый элемент пары
      | ptE    |a|F |  M+1  | M[i+1] -> E - правый элемент пары
      | gtv    |r|  |    C  |
      | ne     |l|  |PrCnt-1|
      | hvc    |c|  |NoLst  |
      | gtv    |l|  | Even  | Последняя пара массива M
      | eq     |l|  |    1  | Запись в регистр неполной пары
      | hrc    |c|  | MaxV  |  если нечетное число элементов в M
      | ptF    |a|F |  M+2  |  - последний элемент
      | hr     |c|  | Cont  |  иначе
MaxV  | ptF    |l|  |  Max  |  - Max
Cont  | prnm   |r|  |    C  | Установка номера процессора,
      | svC    |r|  |    G  |  номер в регистре C
      | inkG   | |  |       | В G номер следующего процессора
```

```
        | decC  |   |   |       | В C номер предыдущего процессора
        | ptH   | l |   |     0 | В H счетчик числа перестановок
Cycle   | emp   |   |   |       | Начало цикла перестановок
        | gtv   | r |   |     D |
        | le    | r |   |     E |
        | hvc   | c |   | NoEx  |
        | ptD   | r |   |     E | E -> D
        | ptv   | r |   |     E | D=S -> E
NoEx    | emp   |   |   |       |
        | gtv   | l |   |   Cnt | Определение конца перестановок
        | eq    | r |   |     H |
        | hrc   | c |   | Write |
        | inkH  |   |   |       |
        | gtv   | r |   |     H | Получение следующей пары
        | mod   | l |   |     2 | Определение четности номера
        | eq    | l |   |     0 |  перестановки
        | hrc   | c |   | SvRt  |
SvLf    | svD   | r |   |     F | Сохранение левого элемента пары
        | svE   | r |   |     D | Перенос правого элемента в левый
        | gtv   | r |   |     G |
        | eq    | l |   | PrCnt |
        | hrc   | c |   | RPair | Крайняя правая пара
        | wait  |   |   |       | Пока не прочтен сохраненный элемент
        | ptE   | p | G |     F | Чтение с ожиданием регистра (F,G)
        | hrc   | c |   | Cycle |  в качестве правого элемента пары
RPair   | ptD   | r |   |     F | Запись из F элемента неполной пары
        | hrc   | c |   | Cycle |  в качестве правого элемента
SvRt    | svE   | r |   |     F | Сохранение правого элемента пары
        | svD   | r |   |     E | Перенос левого элемента в правый
        | gtv   | r |   |     C |
        | lt    | l |   |     0 |
        | hrc   | c |   | LPair | Крайняя левая пара
        | wait  |   |   |       | Пока не прочтен сохраненный элемент
        | ptD   | p | G |     F | Чтение с ожиданием регистра (F,C)
        | hrc   | c |   | Cycle |  в качестве левого элемента пары
LPair   | ptD   | r |   |     F | Запись из F элемента неполной пары
        | hrc   | c |   | Cycle |  в качестве левого элемента
Write   | dec   | G |   |     G | Запись M[i]  и M[i+1]
        | svG   | r |   |     C | В G и C номер пары
        | multG | l |   |     2 |
        | svD   | a | G |     M | M[i] <- D - левый элемент пары
        | svE   | a | G |  M+1  | M[i+1] <- E - правый элемент пары
        | gtv   | r |   |     C |
        | ne    | l |   | PrCnt-1 |
        | hvc   | c |   |  Exit |
        | gtv   | l |   |  Even | Последняя пара массива M
        | eq    | l |   |     1 | Запись из  регистра неполной пары
        | hrc   | c |   |  Exit |  если нечетное число элементов в M
        | ptF   | a | G |  M+2  |  - в последний элемент
Exit    | stop  |   |   |       | Освобождение процессора
```

Следует иметь в виду, что в процессе работы программы используется такой ресурс, как кольцо запроса процессоров. Количество позиций кольца должно быть не меньше размера массива. В противном случае при использовании этого алгоритма массив должен быть поделен на части. Продолжение упорядочивания учитывает уже полученный порядок внутри частей.

3.4. Параллельные вычисления с синхронизацией при помощи рабочих объектов.

В процессе вычисления выражения промежуточные результаты,

присваиваемые рабочим объектам, обычно используются для объединения независимо вычисляемых частей выражения. В случае, когда эти части вычисляются параллельно на разных процессорах, рабочие объекты могут быть использованы и для синхронизации вычисления.

Синхронизация выполняется входящей в вычисление выражения командой с двумя операндами (например, mult), одним из которых является рабочий объект, то есть командой с индикатором адреса, равным "w". По сути дела к команде с синхронизацией обращаются дважды. Выполнение команды начинается с выдачи в кольцо запросов команды "чтение рабочего объекта с возможной записью" значения операнда. Если чтение невозможно (для первого из пришедших операндов тег равен Cl - запись в рабочий объект еще не прошла), то операнд записывается с тегом Rd, а в качестве ответа в кольцо возвращается команда, освобождающая выдавший команду процессор. Если чтение выполняется (второе обращение к команде с синхронизацией со вторым пришедшим операндом), то возвращается значение рабочего объекта, а сам рабочий объект освобождается путем записи ему тега, равного Cl.

Рассмотрим пример. Допустим, что в First и Second переданы признаки синхронизации, задающие начало вычислений на тех же процессорах, которые передали эти признаки. Будем считать, что программа находится в основной памяти и, следовательно, адреса в командах передачи данных базируются БазаКом.

(M+N)*(P-15)

First	nop				Первый процессор
	gtv	a		M	
	add	a		N	
	hv	a		Mlt	Передача операнда
Second	nop				Второй процессор
	gtv	a		P	
	sub	l		15	
Mlt	mult	w		Wa	Умножение на рабочий объект Wa с синхронизацией вычислений

Очевидно, что часть программы, применяющая для синхронизации рабочие объекты, требует, во избежание одновременного их использования несколькими процессорами, собственного множества таких объектов. Назовем часть программы, вход в которую требует подключения собственного множества рабочих объектов, областью с синхронизацией. Легко видеть, что необходимым и достаточным условием того, что выбранная часть программы является областью с синхронизацией, служит наличие только одного входа и по крайней мере одного объекта синхронизации. Области с синхронизацией могут быть вложенными. В момент входа в область ей должны выделяться требуемые рабочие объекты. В качестве примера рассмотрим следующий цикл.

```
for i in 0..1000
  loop
    M[i]:=f(i)+g(i)
  end loop
```

В приведенной ниже программе используется массив рабочих объектов BsWa, в регистре D находится номер выбранного (свободного) рабочего объекта. Объект свободен, если его тег равен Cl, занят - если Wr. Функции f(i) и g(i) вычисляются параллельно.

```
        | gener |l|  |  1000  | Цикл по i 1000..0, i в регистре C
        | ptD   |l|  |    -1  | Исходное состояние D -1
Cont    | emp   |  |  |        | Поиск свободного рабочего объекта
        | incD  |  |  |        |
        | gtv   |r|  |     D  |
        | mod   |l|  |   100  | Одновременно не более 100 точек M[i]
        | ptD   |r|  |     S  | Новое значение D
        | clwr  |a|S |  BsWa  | Разрешена ли запись ?
New     | hvc   |a|  |  Cont  |
        | ...   |  |  |        | Захвачена строка D в массиве BsWa
        | hv    |a|  |  EvG   | Параллельное вычисление f(i) и g(i)
        | ...   |  |  |        | Вычисление f(i), возврат в ResF
EvG     | ...   |  |  |        | Вычисление g(i)
ResG    | nop   |  |  |        | Результат g(i)
ResF    | add   |w|D |  BsWa  | f(i)+g(i)
        | ass   |a|C |     M  | Запись в M[i]
        | RdCl  |a|D |  BsWa  | Освобождение рабочего объекта
        | stop  |  |  |        | Освобождение процессора
```

3.5. Вызов процедуры.

Вызов процедуры передачей данных отличается тем, что любая передача параметра приводит к началу возможных вычислений. Второе отличие - все процедуры должны быть реентерабельными. Это означает, что каждому вызову процедуры соответствует своя локальная память данных.

Очевидно также, что команда, принимающая возвращаемое процедурой значение, должна выполнять функцию синхронизации, а при возвращении значения должны быть восстановлены состояния тех регистров, которые необходимы для дальнейших вычислений.

Вообще говоря, команды, реализующие вызов и возврат из процедуры, записываются в соответствии с соглашениями о связи вызывающей и вызываемой процедур. Один из возможных выриантов такого соглашения реализован в приведенном ниже примере. Соглашение состоит в выделении при каждом вызове процедуры локальной памяти, содержащей, в частности, значения сохраняемых регистров, и адрес точки возврата. В дальнейшем будем считать, что программа расположена в основной памяти,

- все процедуры адресуются с нуля,
- адреса меток указываются по отношению к базе, а адрес самой процедуры устанавливается загрузчиком по отношению к началу памяти,
- программа, выделяющая локальную память для вызываемой процедуры, возвращает ее базу в регистре D,
- локальная память начинается с дескриптора возврата и дескриптора параметров,
- непосредственно вызов процедуры состоит в передаче параметров.

Рассмотрим следующий простой пример.

```
 proc RlSgn(x,y) returns integer
   if x*y>0
     then return 0
     else return 1
   end if;

...p*RlSgn(m,n)... (* вызов RlSgn *)
```

Допустим, что аргумент m записан в локальной памяти вызывающей процедуры по адресу MM, аргумент n - по адресу NN, а p - по адресу PP. Локальная память данных в вызывающей процедуры базируется регистром B. Размер элемента в этой памяти равен 4

байтам. Ниже приведены фрагменты процедуры, расположенной в основной памяти, из которой вызывается процедура RlSgn. Напомним, что база локальной памяти вызываемой процедуры возвращается программой, выделевшей эту область, в регистре D, а сама эта память, естественно, находится в общей памяти. Все объекты памяти вначале имеют тег Cl.

```
        | gtv   |l| |    Rt   | Запись дескриптора возврата в
        | ass   | |D|     0   |  начало локальной памяти RlSgn
        | svA   | |D|     4   |
        | svD   | |D|     8   |
        | gtv   |l|   RlSgn   | Запись дескриптора параметров
        | ass   | |D|    16   | База программы RlSgn
        | svD   | |D|    20   | База локальной памяти RlSgn
        | push  | |D|    24   | Сохранение всех регистров
        | gtv   |l|  RlSgn#1  | RlSgn.1 вход первого параметра
        | ass   | |D|    12   |
        | gtv   |a|  MM       |
        | hvr   |d|D|    12   | Передача первого параметра
        | rdcl  | |D|    12   | Восстановление разрешения присваив
        | gtv   |d|  RlSgn#2  | RlSgn.2 вход второго параметра
        | ass   | |D|    12   |
        | gtv   |a|  NN       |
        | hvr   |d|D|    12   | Передача второго параметра
          ...
        | gtv   |a| | PP      |
        | hvr   |a| | Mlt     |
          ...
Rt      | nop   | | |         | Возврат из RlSgn
        | pop   | |D|    24   | Восстановление регистров
Mlt     | mult  |w| | Wmlt    | Wmlt - рабочий объект в локальной
          ...                   памяти вызывающей программы
```

Процедура RlSgn

```
RlSgn   | proc  | | |         | Начало процедуры
RlSgn#1 | nop   | | |         |
RlSgn#2 | mult  |w|   WRlSgn  | WRlSgn - рабочий объект в
        | gt    |l|       0   |  локальной памяти RlSgn
        | gtv   |l|       0   |
        | hvc   |d|B|     0   | Возврат по дескриптору возврата
        | gtv   |l|       1   |
        | hv    |d|B|     0   | Возврат по дескриптору возврата
        | endp  | | |         |
```

Если бы процедура RlSgn располагалась в локальной памяти процедур, то в дескрипторы параметров вместо базы процедуры необходимо было бы записать ноль, так как в этой памяти дескрипторы не базируются регистром A.

4. ЗАКЛЮЧЕНИЕ.

Предлагаемая архитектура является сильно видоизмененным вариантом архитектуры, рассмотренной в [5]. Автор отошел от предлагаемой pipeline архитектуры в пользу multithreading, найдя достаточно простые и эффективные решения проблем обращения к памяти и выбора свободного процессора. Кроме того, принятые решения допускают быструю передачу данных между процессорами, что позволяет уменьшить количество обращений к памяти. Дальнейшие усовершенствования могут возникнуть после решения задач на модели.

Работа выполнена при поддержке Российского Фонда Фунда-

ментальных Исследований.

ЛИТЕРАТУРА.

1. Dennis, Jack B. Data flow supercomputers.
   Computer 13, 11 (Nov. 1980).
2. Gard, J. R., Kirkham, C. C., Watson, I.
   The Manchester prototype dstaflow computer.
   CACM 28, 1 (january 1985).
3. G.M.Papandoupulos, K.R.Traub
   Multithreading: A Revisionist View of Dataflow
   Architectures
   ACM SIGARCH V. 19, N 3 (May 1991)
4. R.Alverson, D.Callahan, D. Cummings, B.Koblenz,
   A.Porterfield, B.Smith.
   The Tera computer system.
   Proceedings of the 1990 ACM International
   Confirence on Supercomputing, June 1990.
5. В.М.Михелев.
   Архитектура машин с явным управлением данными.
   Препринт ИПМ РАН N% 49, 1999г.