ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 42 за 2009 г.

Ефимкин К.Н.

Эвристический алгоритм распределения заданий

Рекомендуемая форма библиографической ссылки: Ефимкин К.Н. Эвристический алгоритм распределения заданий // Препринты ИПМ им. М.В.Келдыша. 2009. № 42. 16 с. URL: http://library.keldysh.ru/preprint.asp?id=2009-42

К. Н. Ефимкин

Эвристический алгоритм распределения заданий♦

Аннотация

В работе описывается эвристический алгоритм распределения заданий (балансировки нагрузки) по процессорам (исполнителям) с целью эффективного использования этих процессоров. Приведены оценки и результаты практических экспериментов с алгоритмом.

K.N. Efimkin

Heuristic algorithms for jobs scheduling

Abstract

Heuristic algorithms for jobs scheduling (load balancing) on processors (executors) for the purpose of effective usage of these processors is described. Some estimation and experimental results are presented.

♦ Работа выполнена при финансовой поддержке гранта РФФИ N09-01-00329

1. Введение

В работе описывается эвристический алгоритм распределения заданий (балансировки нагрузки) по процессорам (исполнителям) с целью эффективного использования этих процессоров.

Эта задача относится к давно исследуемому классу задач составления расписаний, подробно рассматриваемым, например, в [1]. Неформально, задача составления расписаний может быть определена следующим образом. С помощью некоторого множества ресурсов или обслуживающих устройств необходимо выполнить некоторую систему заданий, при этом целью является оптимизация заданной меры эффективности с учетом свойств заданий и ресурсов.

В конкретных постановках задач составления расписаний свойства ресурсов, заданий и остальных объектов задаются по-разному, в результате чего известно весьма большое число постановок и результатов, полученных для них. С точки зрения сложности решения, большинство задач составления расписания являются NP-полными, хотя существуют и постановки, для которых известны алгоритмы решения с полиномиальными оценками. Например, в работе [2] приведен псевдополиномиальный алгоритм для решения задачи о расписании для двух процессоров.

В данной работе рассматривается постановка задачи составления расписания, ориентированная на распределение независимых заданий в многопроцессорной вычислительной системе с одинаковыми процессорами (балансировку загрузки процессоров). Более точная формулировка задачи и алгоритм ее решения, основанный на применении 2-х простых эвристик, приводится в разделе 2. Задачи такого типа возникают, например, при разработке параллельных программ для многопроцессорных систем с общей памятью. Конкретным мотивом для проведения данного исследования стали работы в ИПМ им. М.В. Келдыша РАН по созданию компилятора с новой версии непроцедурного языка программирования Норма+ для многопроцессорных кластерных многоядерных систем, где такую задачу требуется решать (предыдущая версия системы доступна в [3], описание входного языка Норма для нее – в [4,5]).

В разделе 3 обсуждаются некоторые свойства описываемого алгоритма, оценки для него и некоторые результаты его применения. В Приложении приведен текст программы на Фортране, реализующий алгоритм.

2. Эвристический алгоритм распределения заданий

Сформулируем более точно решаемую задачу.

2.1. Постановка задачи распределения заданий

Пусть задано:

1) множество $P = \{P_1, \ldots, P_m\}$ одинаковых процессоров, $m \geq 2$ произвольно и фиксировано;

2) множество заданий $\Gamma = \{\Gamma_1, \ldots, \Gamma_n\}$ $n \geq 1$ произвольно и фиксировано;

3) задания независимы;

4) множество времен выполнения заданий $\{\tau_i\}$, $i = 1,\ldots,n$, причем $\tau_i < \infty$;

5) дополнительные ресурсы [1] не используются, множество $R = \varnothing$.

Необходимо построить алгоритм составления расписания S выполнения заданий без прерываний отдельных заданий, при котором минимизируется время выполнения всех заданий.

Предлагаемый алгоритм основан на перестановке заданий между процессорами, основанной на двух эвристических правилах.

2.2. Правило 1. “Сильное” улучшение

Пусть заданы два множества заданий: Γ_1, выполняющееся на P_1, и Γ_2, выполняющееся на P_2, таких, что $\Gamma_1 \subseteq \Gamma$ и $\Gamma_2 \subseteq \Gamma$, а $\Gamma_1 \cap \Gamma_2 = \varnothing$. Обозначим через Γ_{max} то множество из Γ_1 или Γ_2, для которого максимальна $\Sigma\tau_i$ по $\Gamma_к$, где $к \in \{1,2\}$; а через Γ_{min} – оставшееся множество, для которого минимально $\Sigma\tau_i$ по $\Gamma_к$, $к \in \{1,2\}$.

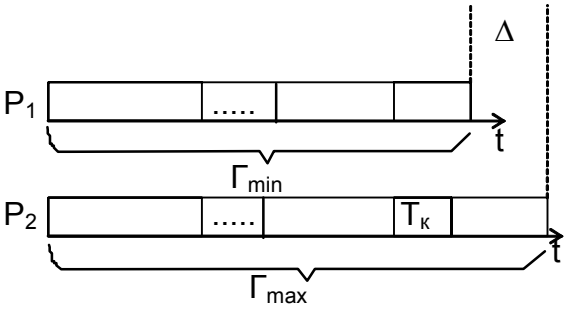

Рис.1.1

Правило 1 переносит целиком одно задание в другой процессор, если выполняются следующие условия. Среди $T_к \in \Gamma_{max}$ найдем такое, для которого $\tau_k < \Delta$, где $\Delta = \left| \sum_{\Gamma_{max}} \tau_i - \sum_{\Gamma_{min}} \tau_i \right|$.

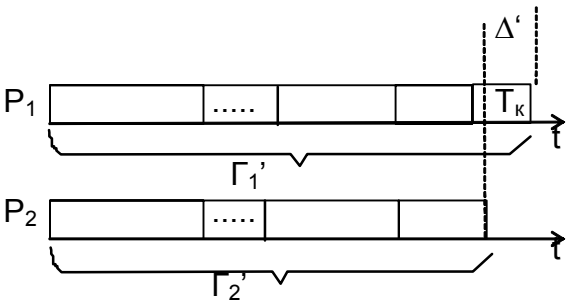

Рис.1.2

Образуем два множества $\Gamma_1' = \Gamma_{min} \cup T_к$ и $\Gamma_2' = \Gamma_{max} / T_к$ (см. Рис. 1.1. и Рис.1.2.)

Лемма 1. Применение ***Правила 1*** приводит к уменьшению максимума из $\sum_{\Gamma 1}\tau_i, \sum_{\Gamma 2}\tau_i$.

Доказательство.

$$\text{Max}(\sum_{\Gamma 1'}\tau_i, \sum_{\Gamma 2'}\tau_i) < \text{Max}(\sum_{\Gamma 1}\tau_i, \sum_{\Gamma 2}\tau_i) = \sum_{\Gamma}\tau_i /2+\Delta'/2<\sum_{\Gamma}\tau_i /2+\Delta/2 \Leftrightarrow \Delta'<\Delta.$$

Пусть $\Gamma_{max}=\Gamma_2$, а $\Gamma_{min} = \Gamma_1$ тогда без ограничения общности: $\sum_{\Gamma 2'}\tau_i = \sum_{\Gamma 2}\tau_i - \tau_k$ и $\sum_{\Gamma 1'}\tau_i = \sum_{\Gamma 1}\tau_i + \tau_k$.

Отсюда $\sum_{\Gamma 2'}\tau_i - \sum_{\Gamma 1'}\tau_i = \sum_{\Gamma 2}\tau_i - \tau_k - \sum_{\Gamma 1}\tau_i - \tau_k = \Delta\text{-}2\tau_k$, но известно, что $\tau_k<\Delta$. Поэтому $-\Delta<\sum_{\Gamma 2'}\tau_i - \sum_{\Gamma 1'}\tau_i <\Delta$, или $\left|\sum_{\Gamma 2'}\tau_i - \sum_{\Gamma 1'}\tau_i\right|<\Delta$. Отсюда, $\Delta'<\Delta$, что и требовалось доказать.

2.3. Правило 2. “Слабое” улучшение

Пусть заданы два множества заданий: Γ_1, выполняющееся на P_1, и Γ_2, выполняющееся на P_2, таких, что $\Gamma_1\subseteq\Gamma$ и $\Gamma_2\subseteq\Gamma$, а $\Gamma_1\cap\Gamma_2=\varnothing$. Обозначим через Γ_{max} то множество из Γ_1 или Γ_2, для которого максимальна $\Sigma\tau_i$ по $\Gamma_к$, где $к\in\{1,2\}$; а через Γ_{min} – оставшееся множество, для которого минимально $\Sigma\tau_i$ по $\Gamma_к$, $к\in\{1,2\}$.

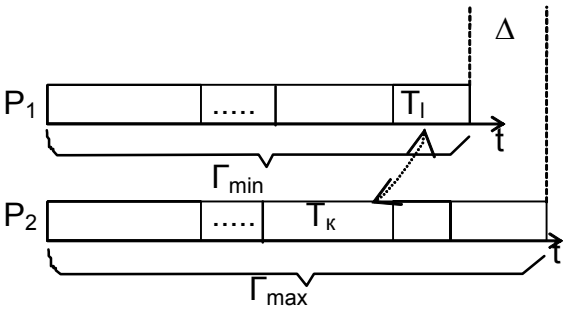

Рис.2.1

Правило 2 переставляет два задания между процессорами, если выполняются следующие условия. Среди $T_к\in\Gamma_{max}$ и $T_l\in\Gamma_{min}$ найдем такие, для которых $\tau_к\text{-}\tau_l<\Delta$, где

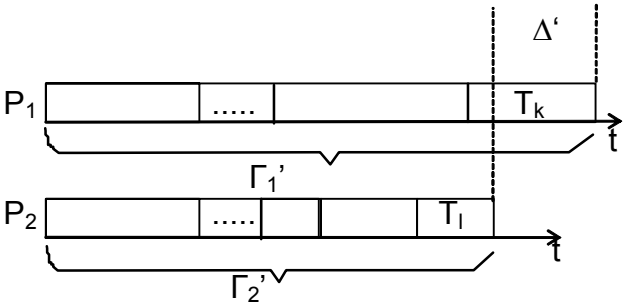

Рис.2.2

$\Delta = \left| \sum_{\Gamma_{max}} \tau_i - \sum_{\Gamma_{min}} \tau_i \right|$. Образуем два

множества $\Gamma_1' = \Gamma_{min} \cup T_к / T_1$ и

$\Gamma_2' = \Gamma_{max} \cup T_l / T_к$. (см. Рис. 2.1. и

Рис.2.2.)

Лемма 2. Применение ***Правила 2*** приводит к уменьшению максимума из $\sum_{\Gamma 1} \tau_i, \sum_{\Gamma 2} \tau_i$.

Доказывается аналогично Лемме 1, если принять $\tau_к = \tau_к - \tau_l$.

2.4. Алгоритм для случая m = 2 процессоров

1. Разобьем произвольным образом множество заданий Г на подмножества Γ_1, Γ_2, выполняющихся соответственно на процессорах P_1, P_2, таких, что $\Gamma_1 \cup \Gamma_2 = \Gamma$, а $\Gamma_1 \cap \Gamma_2 = \varnothing$.

2. Возьмем в качестве Γ_{max} то множество из Γ_1 или Γ_2, для которого максимальна $\Sigma\tau_i$ по $\Gamma_к$, где $к \in \{1,2\}$; а в качестве Γ_{min} – оставшееся множество, для которого минимально $\Sigma\tau_i$ по $\Gamma_к$, $к \in \{1,2\}$.

3. Применим к получившимся множествам Γ_{max} и Γ_{min} ***Правило 1***, а если применить его нельзя, то ***Правило 2***. Если перестановка заданий по какому либо правилу произошла, переходим к п. 2., если применить ни одно из правил нельзя, то алгоритм завершен.

Лемма 3. Последовательное применение ***Правила 1*** и ***Правила 2*** к расписанию (п. 3 алгоритма для m=2 процессоров) приведет к улучшению оценки расписания.

Доказательство.

Утверждение следует из Лемм 1 и 2, так как на каждом следующем шаге $\max(\sum_{\Gamma 1} \tau_i, \sum_{\Gamma 2} \tau_i)$ меньше, чем на предыдущем.

2.5. Алгоритм для случая m > 2 процессоров

1. Разобьем произвольным образом множество Γ на подмножества $\Gamma_1,\ldots,\Gamma_m$, выполняющихся, соответственно, на процессорах $P_1,\ldots,P_m$, такие, что $\cup\Gamma_k=\Gamma$, k = 1 ,..., m, а попарное пересечение $\Gamma_l\cap\Gamma_\kappa=\varnothing$, k,l = 1 ,..., m, k≠l.

2. Введем множество MIN, определенное на множестве натуральных чисел от 1 до m. Положим сначала MIN = ∅.

3. Возьмем в качестве Γ_{max} множество, для которого максимальна $\Sigma\tau_i$ по $\Gamma_к$, k = 1 ,..., m, а в качестве Γ_{min} – множество, для которого минимальна $\Sigma\tau_i$ по $\Gamma_к$, к = 1 ,..., m, k∉ MIN.

4. Применим к получившимся множествам Γ_{max} и Γ_{min} ***Правило 1***, а если это невозможно - ***Правило 2***. Если хотя бы одно правило было применено, переходим к п. 2, если нет - переходим к п. 5.

5. Добавим к множеству MIN новый элемент $IND(\Gamma_{min})$, MIN = MIN ∪ $IND(\Gamma_{min})$, где операция $IND(\Gamma_k)$ определяет номер k∈{1..m} данного множества. Переходим к п. 6.

6. Если мощность множества MIN = m-1 то алгоритм завершается. Если нет , то переходим к п. 3.

3. Свойства алгоритма

В данном пункте рассматриваются некоторые характеристики качества предложенного эвристического алгоритма, в первую очередь сложность и эффективность.

Под сложностью будем понимать функцию, зависящую от основных параметров из постановки задачи (m – количество процессоров, и n – количество заданий), и оценивающую количество элементарных операций (типа сравнения, сложения и т.п.). Под эффективностью будем понимать относительное отклонение решения, полученное данным эвристическим алгоритмом, от оптимального решения (получаемого полным перебором вариантов).

3.1. Верхняя оценка алгоритма

Получим верхнюю оценку сложности алгоритма для случая m > 2.

В описании алгоритма, приведенном в п. 2.5, есть два вложенных циклических действия: п. 3– п.4 и п. 3– п.6. Действиями, не входящими в эти циклы, при оценке можно пренебречь. Оценим сложность действий внутри циклов.

Оценка действий п.3 не превышает $C1\ (n + m)$, поскольку суммируется n чисел и ищется максимум и минимум среди m чисел.

Оценка действий п.4 складывается из оценок Правила 1 и Правила 2, которые равны n и $n^2/2$ соответственно. Поэтому оценка п.4 не превышает $C2\ (n + n^2/2)$.

Действиями п. 6 можно пренебречь.

В результате оценка количества операций для п.3–п.4 не превышает $C1(n+m)+C2(n+n^2/2)$, а число повторений цикла п.3–п.4 (число применений правил) не превышает константы $C3$, зависящей от свойства заданий. Таким образом, оценка для цикл п.3–п.4 есть $C4(n+m+n^2/2)$.

Число повторений цикла п.3–п.6 не превышает m-1, поэтому итоговая верхняя оценка сложности алгоритма – $C(n+m+n^2/2)(m-1)$.

На практике эта оценка достигается чрезвычайно редко. Кроме того, в алгоритме возможен ряд оптимизаций, которые улучшают базовый алгоритм, например:

- задания, распределяемые по процессорам, целесообразно упорядочивать по убыванию времени выполнения (веса), что сокращает перебор в Правилах 1 и 2;

- первоначальную загрузку процессоров (разбиение на подмножества множества заданий Г), целесообразно делать алгоритмом равномерной загрузки.

3.2. Эффективность алгоритма

Оценка эффективности основана на экспериментальных данных. Эксперименты проводились на персональном компьютере AMD Athlon 64, 2.2. GHz, 1 GB RAM.

Задавая различные множества времен выполнения заданий $\{\tau_i\}$, i=1,…,n, и варьируя n – количество заданий, m – количество процессоров, проводилось сравнение решения, полученного эвристическим алгоритмом, и полученного методом полного перебора.

В Таблице 1 приведены обобщенные результаты экспериментов.

m, число процессоров	n, число заданий	диапазон изменения веса задания	отклонение от оптимума, max	отклонение от оптимума, среднее
2	2 – 30	0 – 10000	12%	2%
3	2 – 20	0 – 10000	11%	4%
4	2 – 15	0 – 10000	10%	3%
5	2 – 15	0 – 10000	12%	2%

Табл. 1. Обобщенные результаты применения алгоритма

В 49,92% случаев решение, полученное эвристическим алгоритмом, в точности совпадает с оптимальным, полученным методом полного перебора.

В 24,22% случаев решение, полученное эвристическим алгоритмом, отличается от оптимального, полученного методом полного перебора, не более, чем на 1%, в 25,5% случаев отклонение находится в пределах от 1 до 10%, и лишь в 0,35% случаев отличается более, чем на 10%.

Ниже в Таблице 2 приведены времена работы алгоритма при построении расписания для 810 работ, число процессоров варьируется.

Время работы алгоритма, сек.	m, число процессоров
3.63281	800
24.03125	700
78.96094	600
114.64844	500
98.21875	400
32.43750	300
10.03906	200
0.81250	100
0.49000	50

Табл. 2. Времена работы алгоритма при распределении 810 работ.

Следует отметить, что время работы алгоритма зависит от свойств работ (диапазон изменения значений весов, количества и распределения “крупных” и “мелких” работ и т.п.). В проведенных экспериментах встречались ситуации, когда обработка “неудобного” для планирования множества работ занимала больше времени, при меньших m и n, чем “удобное”, при больших значениях m и n.

В исследованиях принимал участие А.В. Жучков.

Приложение. Текст программы

```
C Размещение n работ r1,r2,...,rn по m процессорам
      PROGRAM WORKS

C-----------------------------
C ВХОД: n - число работ        (задается в PARAMETER)
C       m - число процессоров (задается в PARAMETER )
           INTEGER m,n
           PARAMETER(n=6000,m=50)
C       файл MN.DAT: в строке i (i=1,n) - размер работы i
(целое)
C ВЫХОД:файл SHEDULE.REZ
C       в строке j,j=1,m: процессор j<--список назначенных работ
C       файл MN.REZ - число процессоров, исходные размеры работ,
C                     величина загрузки каждого процессора,
C                     максимальная загрузка, средняя загрузка
C-----------------------------

      INTEGER r(n),rez(n)
C     Рабочие массивы
      INTEGER diapazons(m,n)
      INTEGER icount(0:m)
      INTEGER zagruzka(m),rab(n)

      INTEGER numberr(n), minvek(m)
      LOGICAL yes1,yes2

C  Чтение весов работ
      open(36,file='MN.DAT')
      do i=1,n
       read(36,102) r(i)
      enddo
102   FORMAT(i14)
      close(36)

C Инициализация массива загрузок процессоров работами
      do i=1,m
        zagruzka(m)= 0
```

```
      enddo
      do i=1,n
        numberr(i) = i
        rab(i)     = r(i)
      enddo

C Сортировка работ и Начальное размещение работ
      CALL Sort(rab,n,numberr)
      i=1
      j=1
222   rez(numberr(i))=j
       IF(i.EQ.n) GOTO 333
       IF(j.LT.m)THEN
        j=j+1
        i=i+1
        GOTO 222
       ELSE
        j=1
        i=i+1
        GOTO 222
       ENDIF
333    CONTINUE

C Основной алгоритм: применение двух эвристических правил
C Подсчет начальной загрузки процессоров работами

      do i=1,n
       zagruzka(rez(i))=zagruzka(rez(i))+r(i)
      enddo

C Находим 2 процессора i и j с минимальной и максимальной
C загрузкой и пытаемся обработать их правилами 1 и 2
C Если хотя бы одно применимо - применяем и все заново для всех

C Если ни одно неприменимо - берем следующие 2 и повторяем всю
C процедуру заново для  всех кроме минимального
C  И так до тех пор пока не останется 1 процессор

      ichek=0

C Вектор minvek(i) = 0 если i-й процессор рассматривать не надо
C Сначала все minvek(i)=1

1     CONTINUE

      do k=1,m
        minvek(k)=1
      enddo

555   CONTINUE

C Не пора ли заканчивать обработку?
      itogo=0
```

```
      do k=1,m
      IF(minvek(k).EQ.1) itogo=itogo+1
      enddo
C Пора
      IF(itogo.EQ.1)GOTO 999
C Не Пора - ищем процессоры с минимальной и максимальной
C           загрузкой среди оставшихся
      do k=1,m
       IF(minvek(k).EQ.1) THEN
         max=zagruzka(k)
         min=zagruzka(k)
         i=k
         j=k
         GOTO 33
       ENDIF
      enddo
33    CONTINUE
      do l=2,m
       IF(minvek(l).EQ.0)GOTO 99

       IF (max.LT.zagruzka(l)) THEN
         max=zagruzka(l)
         j=l
       ENDIF
       IF (min.GT.zagruzka(l)) THEN
         min=zagruzka(l)
         i=l
       ENDIF
99     CONTINUE
      enddo

C Для 2-х процессоров i и j

      CALL RULE1(i,j,yes1,n,rez,r,m,zagruzka)

C Правило 1 применилось - все заново
      IF(yes1) GOTO 1
C Правило 1 не применилось - пробуем применить 2-е правило

      CALL RULE2(i,j,yes2,n,rez,r,m,zagruzka)

C Правило 2 применилось - все заново
      IF(yes2) GOTO 1
C Ни одно из правил не применилось - минимальный процессор i
C или максимальный процессор j (через раз) не рассматриваем
C и повторяем процедуру

      IF(ichek.GT.0)THEN
        minvek(i)=0
      ELSE
        minvek(j)=0
      ENDIF
        ichek=-ichek
```

```
      GOTO 555

999   CONTINUE

C Подсчет итоговых характеристик загрузки процессоров работами
      s=0.0
      maxzagr=0
      do i=1,m
        s=s+zagruzka(i)
        IF(maxzagr.LT.zagruzka(i)) maxzagr=zagruzka(i)
      enddo
      s=s/m

      open(45,file='MN.REZ')
        write(45,11) m
        write(45,111) (r(J),J=1,n)
  111 FORMAT(' Исходные работы',/,10(i7))
  11  FORMAT(' Число процессоров = ',I7)
        write(45,776)
        write(45,777) (zagruzka(J),J=1,m)
  776 FORMAT(' Итоговая загрузка ')
  777 FORMAT(10(I8))
        write(45,778) maxzagr,s
  778 FORMAT(' Максимальная загрузка = 'I9,
     >' Средняя загрузка = ',F25.3)
        close(45)

C  Вывод файла-результата: процессор<--работы
        open(46,file='SHEDULE.REZ')
        do i=1,m
          icount(i)=1
        enddo
        do i=1,n
          diapazons(rez(i),icount(rez(i))) = i
          icount(rez(i))                   = icount(rez(i))+1
        enddo
        do i=1,m
         write(46,151) i,(diapazons(i,j),j=1,icount(i)-1)
        enddo
  151 FORMAT('Процессор ',I4,' Работы ',100(i5))
       END

       SUBROUTINE Sort(a,n,numberr)
       INTEGER a(n)
       INTEGER numberr(n)
       INTEGER r

       do m1=n,2,-1
         r=a(1)
         k=numberr(1)
           do i=2,m1
               if(a(i).le.r)then
                  a(i-1)=a(i)
```

```
                numberr(i-1)=numberr(i)
              else
                a(i-1)=r
                numberr(i-1)=k
                r=a(i)
                k=numberr(i)
              endif
            enddo
          a(m1)=r
          numberr(m1)=k
        enddo
      RETURN
      END

       SUBROUTINE RULE1(i,j,yes,n,rez,r,m,zagruzka)
C  i   - минимальная загрузка j - максимальная загрузка
C  yes - TRUE если RULE1 применилось иначе FALSE
C  n   - число работ
C  m   - число процессоров
C  rez[1:n] - размещение работы r[i] на процессор j
C  r[1:n] - размер (вес) работ (целые числа)
C  zagruzka[1:m] - текущая загрузка процессоров

       LOGICAL yes
       INTEGER rez(n),r(n)
       INTEGER zagruzka(m)
       INTEGER delta

       yes=.FALSE.
       IF(zagruzka(i).EQ.zagruzka(j)) GOTO 999
       delta=zagruzka(j)-zagruzka(i)
       maxdel=0
C Поиск ll - номера работы которую можно перенести
       ll=0
       do l=1,n
C Если l-я работа назначена k-му процессору и она меньше delta,
C      то ее можно пренести в процессор kk
C Но лучше среди всех таких найти самую большую
        IF((rez(l).EQ.j).AND.(r(l).LT.delta)
     *                  .AND.(r(l).GT.maxdel))THEN
          maxdel=r(l)
          ll=l
        ENDIF
       enddo
       IF(ll.NE.0)THEN
         zagruzka(j) =zagruzka(j) -r(ll)
         zagruzka(i)=zagruzka(i)+r(ll)
         rez(ll)=i
         yes=.TRUE.
       ENDIF
999   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RULE2(i,j,yes,n,rez,r,m,zagruzka)
C  i    - минимальная загрузка j - максимальная загрузка
C  yes - TRUE если RULE1 применилось иначе FALSE
C  n    - число работ
C  m    - число процессоров
C  rez[1:n] - размещение работы r[i] на процессор j
C  r[1:n] - размер (вес) работ (целые числа)
C  zagruzka[1:m] - текущая загрузка процессоров

      LOGICAL yes
      INTEGER rez(n),r(n)
      INTEGER zagruzka(m)
      INTEGER delta

      yes=.FALSE.
      IF(zagruzka(i).EQ.zagruzka(j)) GOTO 999
      delta=zagruzka(j)-zagruzka(i)
      maxdel=0
C ll и mm - номера работ, которые можно поменять местами
      ll=0
      mm=0
      do l1=1,n
      do l2=1,n
C Если разность l1-й работы, назначенной i-му процессору, и l2-й
C      работы, назначенной j-му процессору, меньше delta,
C      то их можно поменять между этими процессорами
C Но лучше среди всех таких разностей найти самую большую
      IF((rez(l1).EQ.i).AND.(rez(l2).EQ.j)) THEN
        itemp=r(l2)-r(l1)
        IF((itemp.LT.delta).AND.(itemp.GT.maxdel))THEN
         maxdel=itemp
         ll=l1
         mm=l2
        ENDIF
      ENDIF
      enddo
      enddo
      IF(ll.NE.0)THEN
       zagruzka(i) = zagruzka(i) - r(ll) + r(mm)
       zagruzka(j) = zagruzka(j) + r(ll) - r(mm)
       rez(ll)= j
       rez(mm)= i
       yes=.TRUE.
      ENDIF
  999 CONTINUE
      RETURN
      END
```

Литература

1. М. Гэри, Д. Джонстон. Вычислительные машины и труднорешаемые задачи. М., Мир, 1982, 416 с.

2. С.С. Ерчев, И.Б. Задыхайло. Псевдополиномиальный алгоритм с оценками для решения задачи о наилучшем расписании для двухпроцессорной системы с независимыми заданиями. Препринт ИПМ им.М.В.Келдыша РАН, 1988. №153, 20 с.

3. Система НОРМА. http://www.keldysh.ru/pages/norma/.

4. А.Н.Андрианов, А.Б.Бугеря, К.Н.Ефимкин, И.Б.Задыхайло. Норма. Описание языка. Рабочий стандарт. Препринт ИПМ им.М.В.Келдыша РАН, 1995. №120, 50 с.

5. И.Б.Задыхайло, К.Н.Ефимкин. Содержательные обозначения и языки нового поколения. Информационные технологии и вычислительные системы, 1996, №2, с. 46-58.