ЭФФЕКТИВНЫЙ АЛГОРИТМ ПЛАНИРОВАНИЯ ЗАДАЧ, ДОПУСКАЮЩИХ ПАРАЛЛЕЛЬНЫЙ СЧЕТ НА РАЗНОМ ЧИСЛЕ ПРОЦЕССОРОВ. ЕГО ИСПОЛЬЗОВАНИЕ В СИСТЕМЕ DVM

М.Н. Притула

Одним из достоинств DVM-системы является то, что получаемые параллельные программы могут настраиваться при запуске на количество выделенных для них процессоров. Такое свойство программ позволяет запускать их на произвольном числе процессоров, компоновать сложные программы из имеющихся простых программ, повысить эффективность использования параллельных систем коллективного пользования за счет более гибкого распределения процессоров между отдельными программами. К сожалению, этим свойством не обладают DVM-программы, использующие механизм подзадач, поскольку он требует от программиста самому распределить процессоры между подзадачами. Распределение вручную зачастую затруднено вследствие большого количества подзадач, заметного разброса их сложности, необходимости осуществлять распределение на различное количество процессоров.

Постановка исследуемой задачи:

Есть M процессоров, N задач, для каждой задачи известно:

1. минимальное количество процессоров, которое может быть использовано для счета задачи Kmin
2. максимальное количество процессоров, которое может быть использовано для счета задачи Kmax
3. функция зависимости времени исполнения от количества процессоров time(k) > 0 такая, что для всех k из диапазона [Kmin, Kmax - 1] выполнено: time(k) * k <= time(k + 1) * (k + 1)

Требуется составить оптимальное - с минимальным финишным временем - расписание прохождения задач, где для каждой задачи будет указано стартовое время, группа процессоров для ее счета. Задачи не могут считаться одновременно используя один и тот же процессор. Для всех процессоров из группы, назначенной для счета задачи времена старта счета этой задачи совпадают, равно как и времена счета (продолжительность) задачи - синхронное выполнение одной задачи.

Как известно, задача составления многопроцессорного расписания NP-полна, а, значит, исследуемая задача NP-трудна, так как является обобщением NP-полной задачи.

Для более традиционной постановки, в которой для каждой задачи необходимо выделить только один процессор, имеется множество эвристических алгоритмов. Однако в литературе не удалось найти алгоритм автоматического распределения процессоров между задачами, требующими для своего выполнения не одного, а нескольких процессоров.

Для описания алгоритма введем несколько дополнительных обозначений:

Proc(x, d) - множество всех процессоров, свободных с момента времени x и, как минимум, до момента (x + d)

Proc(x) - отображение такое, что Proc(x)(d) = Proc(x, d) для всех d и x

Tasks - множество всех задач

Kmin(t) - минимальное количество процессоров, которое может быть использовано для счета задач t

Kmax(t) - максимальное количество процессоров, которое может быть использовано для счета задач t

time(t, k) - время счета задачи t с использованием k процессоров

Алгоритм распределения задач можно описать следующим образом:

1. Положим tRestMin - сумма по всем задачам t из Tasks величин time(t, Kmin(t)) * Kmin(t)
2. Упорядочить задачи по убыванию величины time(t, Kmin(t)) * Kmin(t) - список sortedTasks, где t принимает все значения из Tasks
3. Положим tMax и tOccupied равными нулю
4. Взять задачу t из начала списка sortedTasks
5. Удалить задачу t из списка sortedTasks
6. Уменьшить tRestMin на величину time(t, Kmin(t)) * Kmin(t)
7. Положим множество notExamined равным [Kmin(t), Kmax(t)]
8. Для каждого момента времени x, являющегося либо началом отсчета времени, либо моментом изменения отображения Proc(x) в порядке возрастания выполнять:
    1. Для каждой длительности d, не меньшей time(t, k) для некоторого k, в порядке убывания выполнять:
        1. . Для каждого количества процессоров k, принадлежащему notExamined и не большему, чем мощность Proc(x, d), а также такому, что time(t, k) <= d в порядке возрастания выполнять:
            1. Положим tSuggested(x, k) равным максимуму из трёх величин: tMax, x + time(t, k), x + (tOccupied + tRestMin) / k
        2. Исключить из множества notExamined рассмотренные в цикле 8.1.1 количества процессоров
        3. Если множество notExamined пусто, то перейти к пункту 9

9. Пусть x0 - такое минимальное, что минимизирует tSuggested(x0, k) для некоторого k. Пусть k0 - минимальное из таких, что минимизируют tSuggested(x0, k0)
10. Положим tMax максимуму из tMax и x0 + time(t, k0)
11. Увеличим tOccupied на величину time(t, k0) * k0
12. Зарезервировать группу из k0 процессоров на время [x0, x0 + time(t, k0)], выделив подмножество из Proc(x0, d) с таким максимальным d, что мощность Proc(x0, d) не менее k0
13. Если список sortedTasks не пуст, то перейти к пункту 4
14. Конец

В результате будет составлено полное расписание прохождения задач на многопроцессорной системе. Алгоритм имеет алгоритмическую сложность асимптотически равную O(N * (N + M)), затраты по памяти асимптотически равны O(N + M). Предлагаемый алгоритм был реализован и включен в систему поддержки времени выполнения DVM-системы LibDVM в виде статической библиотеки функций и сравнен с двумя вариантами ручного распределения, использовавшимися при расчете многоблочной задачи по обсчету аэродинамики самолета в ИПМ им. Келдыша РАН и показал заметное ускорение работы на больших количествах процессоров. Время работы текущей реализации алгоритма на синтетическом случайном тесте с 10000 задачами, 2048 процессорами на среднепроизводительном домашнем компьютере с процессором Intel Core 2 Duo с тактовой частотой 2,33 ГГц составило 100 секунд.

Разработанный алгоритм может быть использован в планировщиках систем очередей для кластеров коллективного доступа.

Предполагается в дальнейшем встроить алгоритм в DVM-систему и использовать при автоматическом распараллеливании многоблочных программ.

В настоящее время для использования алгоритма автоматического отображения подзадач в DVM-программах следует выполнить следующие действия:

1. Завести массив типа integer размером на, как минимум, количество блоков - назовём его renum
2. Узнать количество процессоров в системе (например, парой вызовов DVM-системы NUMBER_OF_PROCESSORS() = 1; NP = NUMBER_OF_PROCESSORS())
3. Для генерации распределения подзадач описанным алгоритмом в зависимости от размерности блоков вставить вызов mproc_adv1_ для одномерных блоков, mproc_adv2_ для двумерных и так далее. Функции вида mproc_adv##n##_ имеют следующий прототип:

   int mproc_adv##n##_ (int *low_proc, int *high_proc, int *size, int *num_blocks, int *num_proc, int *renum);

   Где в первый аргумент – массив целых чисел – будет вписан нижний индекс номеров используемых для подзадачи процессоров; во второй соответственно верхний индекс номеров используемых для подзадачи процессоров; третий аргумент должен содержать размеры блоков по каждому измерению (например, для двумерных блоков размер i-го блока есть size[2 * i] по первому измерению и size[2 * i + 1] по второму); четвертый аргумент суть указатель на число блоков; пятый – указатель на число процессоров; шестой – массив, куда следует записать порядок прохождения подзадач для последующей его передачи системе LibDVM.
4. В директивах DVM необходимо включить полученную перенумерацию. Например, для Fortran-DVM программы, фрагмент кода:

   ```
   *DVM$ TASK_REGION TSA
   *DVM$ PARALLEL ( IB ) ON TSA( IB )
   DO 50 IB = 1,NBL
   CALL JACOBI(block(IB)%PA, block(IB)%PB,SIZE(1,IB),SIZE(2,IB))
   50 CONTINUE
   *DVM$ END TASK_REGION
   ```

   следует преобразовать так:

   ```
   *DVM$ TASK_REGION TSA
   *DVM$ PARALLEL (IB) ON TSA(renum(IB) )
   DO 50 IB = 1,NBL
   CALL JACOBI(block(renum(IB))%PA, block(renum(IB))%PB,SIZE(1, renum(IB)), SIZE(2, renum(IB)))
   50 CONTINUE
   *DVM$ END TASK_REGION
   ```
5. Затем таким образом модифицированный код следует подать на DVM-конвертер, который преобразует ее в Fortran-программу с вызовами LibDVM и ф-ций вида mproc_adv##n##_
6. Затем полученную Fortran-программу следует подать на вход Fortran-компилятору и связать с библиотеками LibDVM.

ЛИТЕРАТУРА:

1. Н.А.Коновалов, В.А.Крюков, А.А.Погребцов, Н.В.Поддерюгина, Ю.Л. Сазанов. Параллельное программирование в системе DVM. Языки Fortran-DVM и C-DVM. Труды Международной конференции "Параллельные вычисления и задачи управления" (PACO’2001) Москва, 2-4 октября 2001 г., 140-154 с.
2. Oliver Sinnen. Task Scheduling for Parallel Systems // John Wiley And Sons, Inc. 2007.

3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. - М.: Мир, 1982. - 416 с.