© Колегов К.С. 2022

Задача о загрузке судна

ЛАБОРАТОРНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Введение. Проблема оптимальной загрузки судна относится к классической задаче о рюкзаке (ранце). В случае, когда партий груза на складе в порту заведомо больше вместимости судна, получаем задачу о неограниченном рюкзаке (unbounded knapsack problem). Существует несколько точных и приближенных методов решения этой задачи. Из точных способов, как правило, используются метод динамического программирования (МДП) и метод ветвей и границ (МВГ). Ознакомиться с МВГ, к примеру, можно в работе [1]. В [2] предложена одна из эффективных модификаций МДП. Зачастую, применяются комбинации нескольких подходов для получения более эффективного алгоритма [3, 4]. В [3, 4] также выполнено сравнение нескольких алгоритмов между собой. Комбинированный параллельный алгоритм рассмотрен в [4]. В [5, 6] распараллелены два алгоритма, относящиеся к семейству ДП, средствами OpenMP и pthreads на многоядерной архитектуре. При использовании 4-х ядер авторами [5, 6] получено ускорение в 2,5–3 раза, а на 8-ми ядрах — в 5–7 раз. Примечательно, что для рассмотренных подходов ускорение расчета на 16-ти ядрах не превышает ускорение в случае использования 8-ми ядер [5]. Работа [7] посвящена разработке параллельных приложений для решения бинарной задачи о рюкзаке, когда предмет либо кладется в рюкзак в количестве 1 шт., либо нет. Авторы [7] сравнивают на примере такой задачи технологии OpenMp и MPI для архитектур с общей и распределенной памятью, соответственно. В [8] для этой же задачи (0–1 knapsack problem) выполнено сопоставление двух версий оптимального параллельного алгоритма: для многоядерного ЦП на основе OpenMp и вычислительной системы на базе графической карты с применением программно-аппаратной платформы CUDA.

Целью данной работы является рассмотрение вопроса о возможности улучшения неэффективного метода полного перебора за счет использования многоядерной архитектуры.

Постановка задачи. Задача о загрузке — это задача о рациональной загрузке судна (самолета, автомашины и тому подобное), которое имеет ограничения по объему или грузоподъемности. Каждый помещенный на судно груз приносит определенную прибыль. Смысл в том, чтобы выбрать такой вариант загрузки судна какими-либо товарами, при котором будет получена наибольшая прибыль. Формализуем описанную задачу. Пусть W — грузоподъемность нашего судна. В наличии есть N наименований партий груза. Пусть m_n – количество партий n-го наименования (n = 0, 1, …, N-1). Считаем,

что число m_n ограничено лишь грузоподъемностью транспорта, $m_n = 0, 1, \ldots, [W/w_n]$, квадратными скобками здесь обозначается целочисленное деление, w_n — вес одной партии груза n-го наименования. Прибыль, которую приносит одна партия груза n-го наименования, обозначим c_n. Тогда общая задача имеет вид следующей целочисленной задачи линейного программирования. Необходимо выбрать число m_n партий каждого типа так, чтобы максимизировать общую прибыль $\sum_{n=0}^{N-1} c_n m_n$. При этом должно выполняться условие $\sum_{n=0}^{N-1} w_n m_n \le W$. Отметим также, что $c_n > 0$, $w_n > 0$, $m_n \ge 0$ (все числа целые). Вектор m_n, соответствующий максимальной прибыли C_{max}, обозначим как M_{best}.

Алгоритм программы расчета. Временная сложность метода полного перебора $O(N!)$, то есть он работоспособен для небольших значений N. С ростом N задача становится неразрешимой данным методом за приемлемое время. Для небольших портов число видов партий груза N, как правило, невелико. Из плюсов можно отметить то, что такой алгоритм решения задачи прост в реализации и хорошо параллелится на многоядерных или многопроцессорных системах. Вполне возможно, что использование высокопроизводительных вычислений решит проблему продолжительности счета.

Число вариантов загрузки $N_v = \prod_{n=0}^{N-1} (M_n + 1)$, где $M_n = [W/w_n]$, M_n — максимально возможное количество партий груза n-го наименования, которое может быть взято на судно без превышения его грузоподъемности. Теперь запишем соотношение, которое свяжет i-й вариант погрузки с уникальным видом вектора m_n,

$$m_n = \begin{cases} i \bmod (M_n + 1), & n = 0 \\ \left[i \Big/ \prod_{k=0}^{n-1} (M_k + 1) \right] \bmod (M_n + 1), & n > 0 \end{cases},$$

где $i = 0, 1, \ldots, N_v - 1$. Использование данной формулы при реализации алгоритма перебора позволит отказаться от рекурсии. Перебор всех вариантов выполняется в параметрическом цикле (последовательная версия алгоритма изображена в виде блок-схемы на рис. 1). Такой цикл достаточно просто и эффективно параллелится средствами OpenMP, так как в описанном способе отсутствуют зависимости по данным. Этот цикл будет выполняться одновременно несколькими потоками, у каждого из которых имеется своя зона ответственности. То есть часть вариантов перебирается нулевым потоком, другая часть — первым и так далее. Каждый поток запоминает лучший найденный вариант загрузки в процессе перебора. После того, как все потоки отработали, выбирается наилучшее решение из N_{thr} вариантов, где N_{thr} равняется количеству логических или физических ядер (при отсутствии у

микропроцессора функции виртуализации HyperThreading). На базе предложенного здесь алгоритма написана программа расчета на языке Си++. Применение технологии OpenMP позволяет перейти от последовательной версии программы к параллельной и обратно без значительных изменений в ее коде.

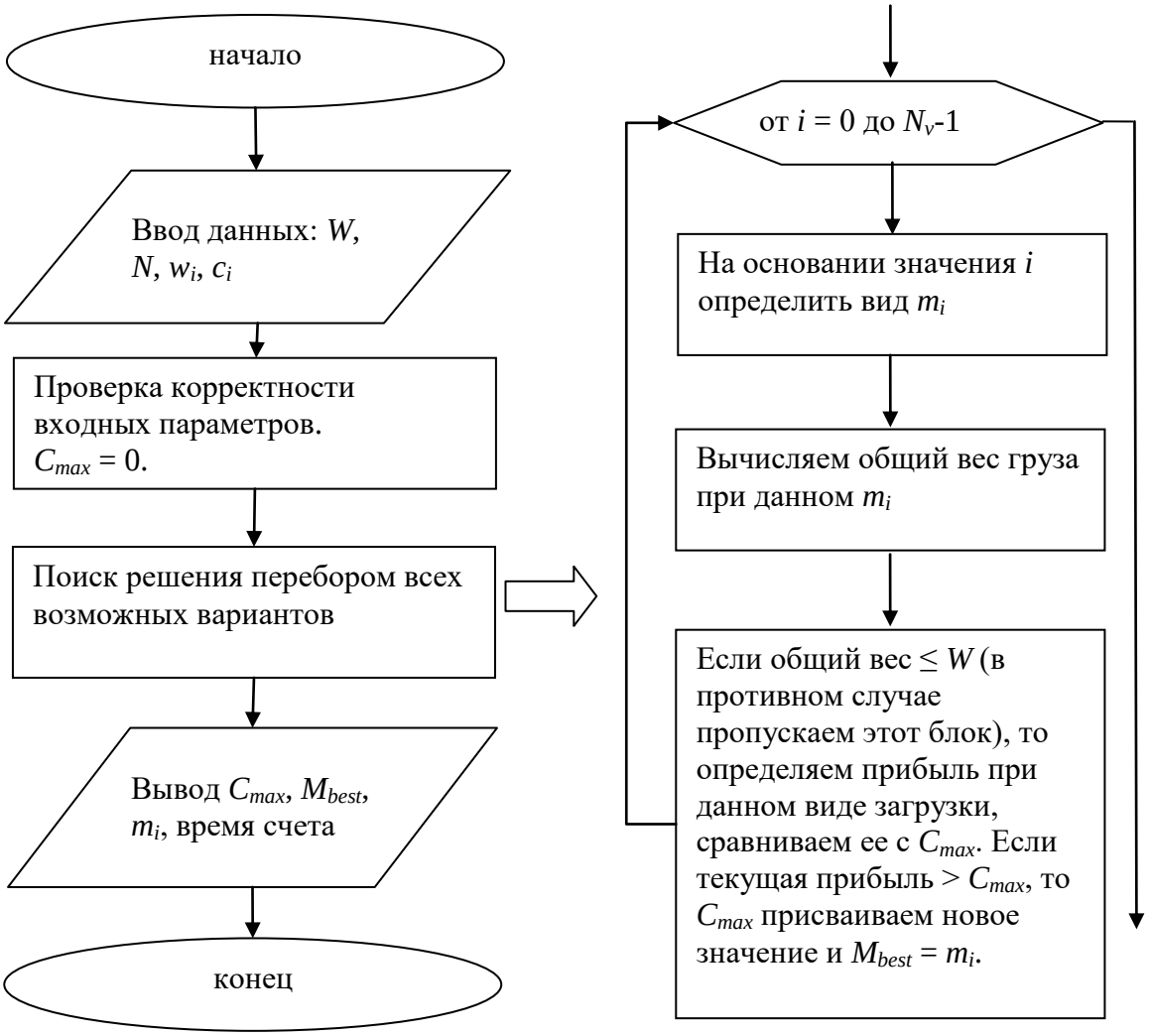

Рис. 1. Блок-схема алгоритма перебора

Задание. Напишите программу (листинг 1) и проанализируйте метод распараллеливания на основе OpenMP. Повторите расчеты по аналоги с работой [9] и подготовьте отчет в Excel. Предоставьте для проверки код программы и отчет.

```
#include <fstream>
#include <string>
#include <iostream>
#include <ctime>
#include <clocale>
#include <iomanip>
#include <cmath>
```

```
#include <omp.h>
#include <cstring>

#define num_proc_threds 6

using namespace std;

int main() {
	setlocale(LC_ALL, "Russian_Russia.1251");
//общее число потоков, количество потоков для использования
	int totalNumThreads = 1, usingNumThreads = 1;
	#pragma omp parallel
	{
		#pragma omp single
			totalNumThreads =  omp_get_num_procs();
	}
	cout << "Maximal number of threads = " << totalNumThreads << endl;//вывод количества потоков в системе
	cout << "What number of threads do you want to use?" << endl;
	cin >> usingNumThreads;
	if (usingNumThreads>totalNumThreads) {
		cerr << "Wrong number of threads have been entered!" << endl;
		cin.get();
		return -1;
	}
	else omp_set_num_threads(usingNumThreads);

	cout << "Number of using threads = " << usingNumThreads << endl;

	if (num_proc_threds < usingNumThreads) {
		cerr << "Too much of threads!" << endl;
		cin.get();
		return -1;
	}
//грузоподъемность судна, количество наименований груза, счетчик
	int W = 0, N = 0, i = 0;
	clock_t jobTime;//время работы программы
	jobTime = clock();
	//СЧИТЫВАЕМ ВХОДНЫЕ ПАРАМЕТРЫ ИЗ ФАЙЛА
	string str;
	ifstream file("input.txt", ios::in);
	getline(file, str);//отступаем строчку с подписями данных
	getline(file, str);//считываем параметры задачи
	sscanf(str.c_str(), "%u %u", &W, &N);
	if ((W<1) || (N<1)) {
```

```
                cerr << "Ошибка: W<1 или N<1" << endl;
                cin.get();
                return -1;
        }

        //выделяем память под вектора
        int *w;
        int *c;
        int *bestM;
        w = new int[N];
    //инициализируем нулями вектор для хранения веса груза
        memset(w, 0, sizeof(*w)*N);
        c = new int[N];
        memset(c, 0, sizeof(*c)*N);//... цены груза
        bestM = new int[N];
        memset(bestM, 0, sizeof(*bestM)*N);
        if (!w || !c || !bestM) {
                cerr << "Error: Lack of memory" << endl;
                cin.get();
                return -1;
        }

        getline(file, str);//отступаем строчку с подписями данных
        for (i = 0; (!file.eof()) && (i<N); i++) {
                getline(file, str);//считываем входные данные
                int x = 0, y = 0;//промежуточные переменные
                sscanf(str.c_str(), "%u %u", &x, &y);
                if ((x < 0) || (y < 0)) {
                        cerr << "Error: in file input.txt negative value!" << endl;
                        cin.get();
                        return -1;
                }
                w[i] = x; c[i] = y;
        }
        file.close();
        if (i<N) {
                cerr << "Error: lack of input data in file input.txt" << endl;
                cin.get();
                return -1;
        }

        cout << endl << "capacity of ship = " << W << ", number of cargos = "
<< N << endl;

        for (int i = 0; i<N; i++) {//вывод массивов на экран
```

```
                cout << setw(3 + (short)log10((double)N) - (short)log10((double)i)) //форматируем вывод, выравниваем столбцы
                << "w[" << i << "]= " << w[i] << " " //выводим w
                << setw(3 + (short)log10((double)N) - (short)log10((double)i)) //форматируем вывод, выравниваем столбцы
                << "c[" << i << "]= " << c[i] << " "//выводим c
                << endl;//выводим m
        }
        //РАСЧЕТ
    // максимально возможная прибыль по кажд. потоку
        int maxC[num_proc_threds] = { 0 };
    // вес взятый к погрузке по кажд. потоку
        int  loadW[num_proc_threds] = { 0 };
    //позиция наилучшего варианта по кажд. потоку
        long long i_maxPos[num_proc_threds] = { 0 };
        long long NumVar = 1; //число вариантов загрузки судна
    //определяем количество вариантов
        for (int n = 0; n < N; n++) NumVar *= (long long) W / w[n] + 1;
        cout << "Number of variants: " << NumVar << endl;
        //выполняем расчет для всех возможных вариантов
        #pragma omp parallel
        {
            int m;
    //распараллеливаем цикл для обработки несколькими потоками
            #pragma omp for
            for (long long i = 0; i < NumVar; ++i) {
                m = i % (W / w[0] + 1);
                //цена и вес груза
                int price = 0, weight = 0;
    //цена всего груза для данного варианта погрузки
                price += c[0] *  m;
    //масса всего груза для данного варианта погрузки
                weight += w[0] *  m;
            //в зависимости от значения i определяем вариант загрузки
                for (int n = 1; n < N; n++) {
                // временная переменная для хранения произведения
                        int P = 1;
                        for (int k = 0; k < n; k++) P *= W / w[k] + 1;
                        m = (i / P) % (W / w[n] + 1);
                        //считаем сумму произведений
                // цена всего груза для данного варианта погрузки
                        price += c[n] * m;
    //масса всего груза для данного варианта погрузки
                        weight += w[n] * m;
                }
```

```
                        // номер текущего потока
                        int tid = omp_get_thread_num();
        //если найдено новое оптимальное решение, то перезаписываем ответ
                        if (
                                ((price>maxC[tid]) && (weight <= W))
                                ||
                                ((price == maxC[tid]) &&
                                        (loadW[tid]>weight))
                        )
                                {
                        //запоминаем новое максимальное значение прибыли
                                maxC[tid] = price;
                        //запоминаем новый вес товара к погрузке
                                loadW[tid] = weight;
                                i_maxPos[tid] = i;
                                }
                        }
                }
                //ОПРЕДЕЛЯЕМ ГЛОБАЛЬНЫЙ МАКСИМУМ
                int maxC_global = maxC[0];
                int loadW_global = loadW[0];
                long long i_maxPos_global = i_maxPos[0];
                for (int j = 1; j < usingNumThreads; j++)
                        if ((maxC[j] > maxC_global)||((maxC[j]==
maxC_global)&&(loadW[j]<loadW_global))) {
                        maxC_global = maxC[j];
                        loadW_global = loadW[j];
                        i_maxPos_global = i_maxPos[j];
                        }
        //в зависимости от значения i_maxPos_global определяем вариант загрузки
                bestM[0] = i_maxPos_global % (W / w[0] + 1);
                for (int n = 1; n < N; n++){
                // временная переменная для хранения произведения
                        int P = 1;
                        for (unsigned short k = 0; k < n; k++) P *= W / w[k] + 1;
                        bestM[n] = (unsigned short)((i_maxPos_global / P) % (W /
w[n] + 1));
                }
                //ВЫВОД РЕЗУЛЬТАТА
                ofstream result("output.txt", ios::out);
                if (result.fail()) {
                        cerr << "Error: can't create result file output.txt" << endl;
                        cin.get();
                        return -1;
                }
```

```
            for (int i = 0; i<N; i++) {//вывод массивов в файл
                result << setw(3 + (short)log10((double)N) -
(short)log10((double)i)) //форматируем вывод, выравниваем столбцы
                << "w[" << i << "]= " << w[i] << " " //выводим w
                << setw(3 + (short)log10((double)N) -
(short)log10((double)i)) //форматируем вывод, выравниваем столбцы
                << "c[" << i << "]= " << c[i] << " "//выводим c
                << "m[" << i << "]= " << bestM[i] << endl;//выводим m
            }
            result.close();
            //вывод остальных результатов
            cout << "max profit = " << maxC_global << endl;
            cout << "weight of the goods = " << loadW_global << endl;
            jobTime = clock() - jobTime;//время в тактах микропроцессора
            cout << "Execution time: " << (double)jobTime / CLOCKS_PER_SEC
<< " c" << endl;
            delete[] w; delete[] c; delete[] bestM;
            cin.get();
            return 0;
        }
```

На рис. 2 приведен пример содержания файла input.txt (слева) и output.txt (справа).

```
W n
55 8
w c
5 5
9 9
6 6
7 7
8 8
4 4
3 3
10 15
```

```
w[0]= 5  c[0]= 5 m[0]= 1
w[1]= 9  c[1]= 9 m[1]= 0
w[2]= 6  c[2]= 6 m[2]= 0
w[3]= 7  c[3]= 7 m[3]= 0
w[4]= 8  c[4]= 8 m[4]= 0
w[5]= 4  c[5]= 4 m[5]= 0
w[6]= 3  c[6]= 3 m[6]= 0
w[7]= 10  c[7]= 15 m[7]= 5

```

Рис. 2. Содержания файла input.txt (слева) и output.txt (справа).

Корректность результата можно проверить с помощью модуля «Поиск решения» в Excel [10]. На рис. 3 представлена таблица, в которой белые ячейки заполняются данными. В зеленой ячейке (*D10*) содержится целевая функция *=СУММ(D2:D9)*, рассчитывающая общий доход по всем грузам. Доход в лиловых ячейках (*D2:D9*) рассчитывается по формуле *=B2*C2*. Для расчета массы партии в оранжевых ячейках (*E2:E9*) задана формула *=A2*C2*. В желтой ячейке (*E10*) общая масса груза рассчитывается по формуле *=СУММ(E2:E9)*.

© Колегов К.С. 2022

Синие ячейки (*C2:C9*) для начала можно заполнить, к примеру, нулями. Далее они будут автоматически пересчитаны с помощью модуля «Поиск решения» (см. инструкцию в [10]). Настройка этого модуля показана на рис. 4. Ограничения вводятся с помощью кнопки добавить. Для запуска этого модуля нажмите кнопку *Выполнить*, затем в появившемся окошке выберите пункт «Сохранить найденное решение» и нажмите OK. В итоге, как видно из рис. 3, результаты совпадают с результатами проведенных вычислений (рис. 2).

	A	B	C	D	E	F	G
1	масса груза	прибыль	к погрузке	доход	масса партии		
2	5	5	1	5	5		
3	9	9	0	0	0		
4	6	6	0	0	0		
5	7	7	0	0	0		
6	8	8	0	0	0		
7	4	4	0	0	0		
8	3	3	0	0	0		
9	10	15	5	75	50	Грузоподъемность	
10				80	55	55	

Рис. 3. Проверка результата в Excel

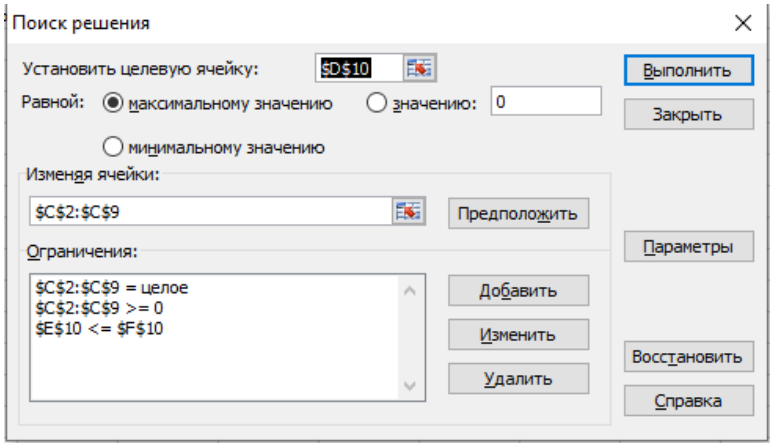

Рис. 4. Настройка модуля «Поиск решения»

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Martello S., Toth P. An exact algorithm for large unbounded knapsack problems // Operations Research Letters. — Vol. 9, Issue 1. — 1990. — Pp. 15–20.

2. Andonov R., Poirriez V., Rajopadhye S. Unbounded knapsack problem: Dynamic programming revisited // European Journal of Operational Research. — Vol. 123, Issue 2. — 2000. — Pages 394–407.
3. Poirriez V., Yanev N., Andonov R. A hybrid algorithm for the unbounded knapsack problem // Discret. Optim. — Vol. 6, no. 1. — 2009. — Pp. 110–124.
4. Посыпкин М.А., Сигал И.Х. Комбинированный параллельный алгоритм решения задачи о ранце // Известия РАН, Теория и системы управления. — № 4. — 2008. — С. 50–58.
5. Novoa C., Qasem A., Rashid H. An Evaluation of Parallel Knapsack Algorithms on Multicore Architectures // CSC. — 2010.
6. Rashid H., Novoa C., McKenney M., Qasem A. Efficient parallel solutions to the integral knapsack problem on current chip-multiprocessor systems // International Journal of Parallel, Emergent and Distributed Systems. — Vol. 27, no. 1. — 2012. — Pp. 19–44.
7. Crawford M., Toth D. Parallelization of the knapsack problem as an introductory experience in parallel computing // Journal of Computational Science Education. — Vol. 4, Issue 1. — 2013. — Pp. 35–39.
8. Li K., Liu J., Wan L. & et. al. A cost-optimal parallel algorithm for the 0–1 knapsack problem and its performance on multicore CPU and GPU implementations // Parallel Computing. — Vol. 43. — 2015. — Pp. 27–42.
9. Демьянович Л.А., Колегов К.С. Реализация параллельного алгоритма полного перебора для решения задачи о загрузке судна // Инновационное развитие транспортно-логистического комплекса Прикаспийского региона [Текст]: материалы V Международной научно-практической конференции (Астрахань, 27-28 мая 2016 г.). — Астрахань: Каспийский институт морского и речного транспорта филиал ФГБОУ ВО "ВГУВТ", 2016.— С. 7-12 (http://www.kimrt.ru/_ld/1/136_DemyanovichKole.pdf)
10. Колегов К.С. Лабораторная работа: «Создание модели работы судна при трамповом судоходстве в Excel» http://www.kimrt.ru/_ld/0/94_11._____.pdf