Автоматическая генерация MPI кода для диалогового высокоуровневого оптимизирующего распараллеливателя

Е.Н. Кравченко, И. Скиба

Южный федеральный университет, Ростов-на-Дону

Введение

В связи с распространением многоядерных процессоров и многопроцессорных систем актуальной задачей становится не только написание параллельных программ, но и автоматическое и полуавтоматическое преобразование написанных последовательных программ в параллельные программы. В данной работе описан программный модуль MPIProducer, разработанный в рамках проекта ДВОР, позволяющий автоматически (полуавтоматически) генерировать MPI код для программ, содержащих циклы специального вида. Автоматическая генерация MPI была реализована ранее М. Гуревичем в рамках проекта ОРС.

На данный момент модуль MPIProducer генерирует MPI код для программ удовлетворяющих следующим требованиям: программа не содержит операторов безусловного перехода (goto, break, continue); распараллеливаемый цикл не содержит вложенных циклов, распараллеливаемый цикл канонизирован; данные, обрабатываемые распараллеливаемым циклом являются одномерными массивами; все индексные выражения линейно зависят от счетчиков циклов; распараллеливаемый цикл содержит только входные зависимости или не содержит их вообще.

В будущем планируется модифицировать модуль MPIProducer. Модификация предполагает следующее: распараллеливание циклов, содержащих истинные зависимости и антизависимости; распараллеливание циклов, содержащих вложенные циклы; распараллеливание циклов, оперирующих скалярными переменными и двумерными массивами.

Диалоговый высокоуровневый оптимизирующий распараллеливатель

Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) – проект, разрабатываемый группой студентов и аспирантов, который предназначен для автоматизации распараллеливания программ. Подающаяся на вход ДВОР программа сначала разбирается, и по ней строится дерево из объектов внутреннего представления ДВОР, затем к этой программе возможно применение преобразований из библиотеки преобразований ДВОР, построение графов для вспомогательного анализа, генерация результирующего кода. Прототипом ДВОР была Открытая распараллеливающая система (ОРС).

Информационные зависимости в программе

Если при допустимых индексных выражениях два вхождения u и v обращаются к одной и той же ячейке памяти, то говорят, что эти вхождения порождают информационную зависимость. При этом вхождение v зависит от вхождения u, если вхождение v обращается к некоторой ячейке памяти позже, чем вхождение u.

Пример 1. Следующий фрагмент программы содержит информационную зависимость

```
for(i = 1; i < 10; i++)
  x[i]= x[i-1]+y[i];
   u       v
```

имеется зависимость вхождения v от u т.к., например, к ячейке памяти x[5] обращается сначала вхождение u на итерации 5, а затем вхождение v на итерации 6.

Конец примера 1.

Пусть вхождение v зависит от вхождения u. Различают четыре типа информационной зависимости:

- если u является генератором, а v – использованием, то зависимость такого типа называется истинной (true dependence) или потоковой зависимостью (flow dependence);
- если u является использованием, а v – генератором, то зависимость такого типа называется антизависимостью (antidependence);
- если u и v являются генераторами, то зависимость такого типа называется выходной зависимостью (output dependence) или зависимостью по выходу;
- если u и v являются использованиями, то зависимость такого типа называется входной зависимостью (input dependence) или зависимостью по входу.

Гнездование циклов

В составе ДВОР разработано преобразование гнездование циклов. Данное преобразование заменяет цикл

```
for (i = 0 ; i < n; i = i + 1)
     loopBody(i);
```

на следующий фрагмент программы:

```
for (j = 0; j < h; j = j + 1)
     for (i = 0; i < n / h; i = i + 1)
          loopBody(i + j * (n / h));
for (i = 0; i < n - (n / h) * h; i = i + 1)
     loopBody((n / h) * h + i);
```

Если n – константа и не делится без остатка на h, то ко второму циклу полученного фрагмента программы можно применить раскрутку цикла.

Если n делится на h, то гнездование циклов выглядит более просто. Преобразование заменяет исходный цикл циклом следующего вида:

```
for (j = 0; j < h; j = j + 1)
     for (i = 0; i < n / h; i = i + 1)
          loopBody(i + j * (n / h));
```

Если `loopBody(i)` содержит помеченные операторы, то в копиях тела цикла должны вводиться новые метки.

Применение данного преобразования может быть целесообразно при автоматическом (полуавтоматическом) распараллеливании программ.

Генерация MPI кода

Пример 2. Программа, содержащая цикл, который необходимо выполнить параллельно

```
void main()
{
     int a[20000], b[20000], j, sum;

     // Здесь заполняются массивы a и b
     for(j = 0; j < 10000; j++)
          b[j] = a[j] * a[j + 1];
}
```

Конец примера 2.

Требуется модифицировать данную программу с помощью библиотеки MPI так, чтобы цикл for выполнялся параллельно на нескольких узлах распределенной системы.

В данном примере при распараллеливании цикла for, требуется переслать на все узлы кроме 0, части массивов x и y, т.к. они используются для вычисления массива a. Также по завершении циклов на всех узлах требуется переслать полученные на каждом узле части массива a 0-му узлу. В процессе генерации MPI кода к распараллеливаемому циклу применяется гнездование циклов. Каждая итерация, полученного в ходе преобразования цикла выполняется на отдельном узле вычислительной системы.
После генерации MPI кода MPIProducer выдает результат, приведенный в примере 3.
Пример 3. Результат работы MPIProducer

```
#include "mpi.h"

int MPIPRODUCER_rank;
int MPIPRODUCER_numprocs;

int main(int argc, char **argv)
{
    int a[20000], b[20000], i, j, sum, MPIPRODUCER_it;
    MPI_Status MPIPRODUCER_status;
    MPI_Datatype MPIUSERDATATYPE;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &MPIPRODUCER_numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &MPIPRODUCER_rank);
    if(MPIPRODUCER_rank==0)
    {
        // Здесь заполняются массивы a и b
    }
    // Обработка цикла
    {
        // 0-й узел рассылает данные
        if(MPIPRODUCER_rank==0)
        {
            for(MPIPRODUCER_it = 1; MPIPRODUCER_it <
MPIPRODUCER_numprocs; MPIPRODUCER_it++)
            {
                MPI_Type_vector(1000, 1, 1, MPI_INT,
&MPIUSERDATATYPE);
                MPI_Type_commit(&MPIUSERDATATYPE);
                MPI_Send(a + 1 + MPIPRODUCER_it * 1000, 1,
MPIUSERDATATYPE, MPIPRODUCER_it, MPIPRODUCER_it, MPI_COMM_WORLD);
                MPI_Type_vector(1000, 1, 1, MPI_INT,
&MPIUSERDATATYPE);
                MPI_Type_commit(&MPIUSERDATATYPE);
                MPI_Send(a + 0 + MPIPRODUCER_it * 1000, 1,
MPIUSERDATATYPE, MPIPRODUCER_it, MPIPRODUCER_it, MPI_COMM_WORLD);
            }
        }
        // Прием данных
        if(MPIPRODUCER_rank!=0)
        {
            MPI_Type_vector(1000, 1, 1, MPI_INT, &MPIUSERDATATYPE);
            MPI_Type_commit(&MPIUSERDATATYPE);
            MPI_Recv(a + 1 + MPIPRODUCER_rank * 1000, 1,
MPIUSERDATATYPE, 0, MPIPRODUCER_rank, MPI_COMM_WORLD,
&MPIPRODUCER_status);
            MPI_Type_vector(1000, 1, 1, MPI_INT, &MPIUSERDATATYPE);
            MPI_Type_commit(&MPIUSERDATATYPE);
```

```
            MPI_Recv(a + 0 + MPIPRODUCER_rank * 1000, 1,
MPIUSERDATATYPE, 0, MPIPRODUCER_rank, MPI_COMM_WORLD,
&MPIPRODUCER_status);
        }
        // Выполнение цикла на каждом узле
        for (j = 0; (j)<(1000); j++)
        {
            (b)[((MPIPRODUCER_rank)*(1000))+(j)] =
((a)[((MPIPRODUCER_rank)*(1000))+(j)])*((a)[(((MPIPRODUCER_rank)*(10
00))+(j))+(1)]);
        }
        // Пересылка результатов на 0-й узел
        if(MPIPRODUCER_rank!=0)
        {
            MPI_Type_vector(1000, 1, 1, MPI_INT, &MPIUSERDATATYPE);
            MPI_Type_commit(&MPIUSERDATATYPE);
            MPI_Send(b + 0 + MPIPRODUCER_rank * 1000, 1,
MPIUSERDATATYPE, 0, MPIPRODUCER_rank + MPIPRODUCER_numprocs,
MPI_COMM_WORLD);
        }
        // Сбор результатов
        if(MPIPRODUCER_rank==0)
        {
            for(MPIPRODUCER_it = 1; MPIPRODUCER_it <
MPIPRODUCER_numprocs; MPIPRODUCER_it++)
            {
                MPI_Type_vector(1000, 1, 1, MPI_INT,
&MPIUSERDATATYPE);
                MPI_Type_commit(&MPIUSERDATATYPE);
                MPI_Recv(b + 0 + MPIPRODUCER_it * 1000, 1,
MPIUSERDATATYPE, MPIPRODUCER_it, MPIPRODUCER_it +
MPIPRODUCER_numprocs, MPI_COMM_WORLD, &MPIPRODUCER_status);
            }
        }
    }
    if(MPIPRODUCER_rank==0)
    {
    }
    MPI_Finalize();
}
```

Конец примера 3.

Литература

1. www.ops.rsu.ru
2. www.rsusu1.rnd.runnet.ru/tutor/method/index.html
3. Штейнберг Б.Я. «Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью», Издательство Ростовского университета, Ростов-на-Дону, 2004