Дальневосточный федеральный университет
Новосибирский государственный университет
Институт вычислительной математики и математической геофизики СО РАН

Асинхронная модель вычислений с управлением на основе строгого частичного порядка

С.Б. Арыков

ПаВТ-2016

Содержание

1. Введение
2. Асинхронная модель вычислений
   - Определения асинхронных моделей
   - Конструирование асинхронных программ
3. Система программирования Аспект
   - Архитектура системы
   - Язык программирования Аспект
   - Транслятор
   - Результаты тестовых испытаний

Введение


Непроцедурное представление алгоритмов
Процедурное представление алгоритмов
НОРМА
Пифагор
LuNa
Асинхронная модель вычислений
V-Ray
OPC

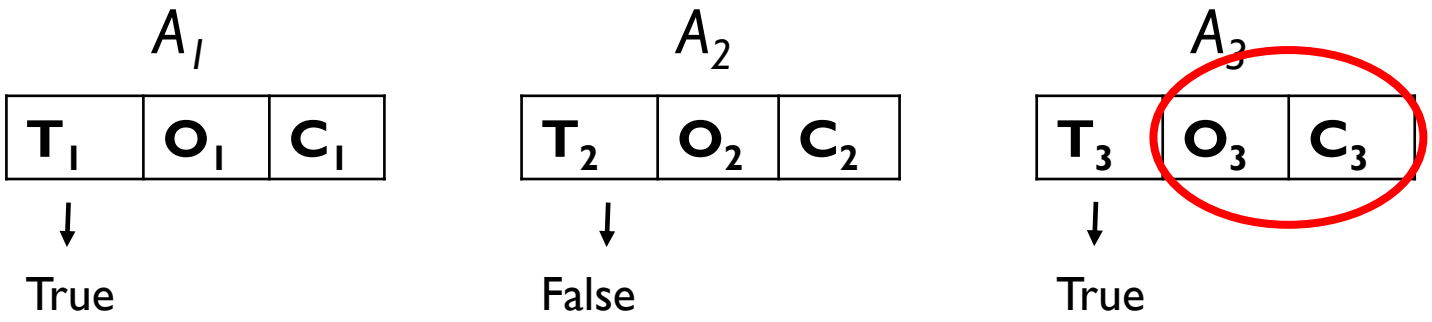
Асинхронная модель вычислений

Асинхронная модель вычислений

Котов В.Е, ***Нариньяни*** А.С. Асинхронные вычислительные процессы над памятью // Кибернетика. 1966. № 3. С. 64–71.

Асинхронная программа (*А*-программа) $P = (M, A)$, где:

- M – конечное множество *переменных*, $M = IM \cup CM$;
- A – конечное множество *А-блоков*, $A_k = (M_k, T_k, O_k, C_k)$



Асинхронная модель вычислений

Вальковский В.А., ***Малышкин*** В.Э. Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988. 129 с.

$IM = X \cup Y$

- X – множество простых переменных
- Y – множество структурных переменных (массивов)

A_k: (A_k, i), i $\in \mathrm{N}$

Асинхронная модель вычислений

Асинхронная модель вычислений с управлением на основе строгого частичного порядка:

1. Управляющий оператор заменён на бинарное отношение строгого частичного порядка
2. Каждый экземпляр А-блока исполняется не более одного раза

M = {x} ∪ {y = true, z = false}

T_1={y == true}
O_1 = {x = 0}
C_1 = {y = false; z = true}

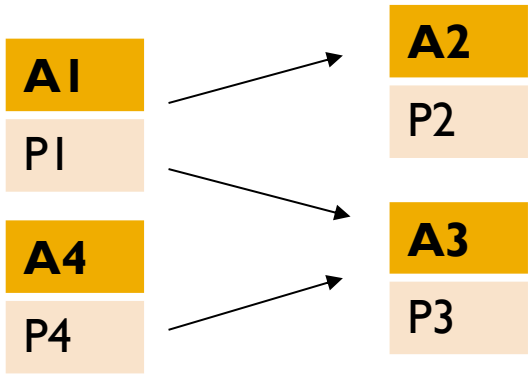
T_2 = {z == true}
O_2 = {x = x+1}
C_2 = {z = false}

$A_1 < A_2$

Конструирование асинхронных программ

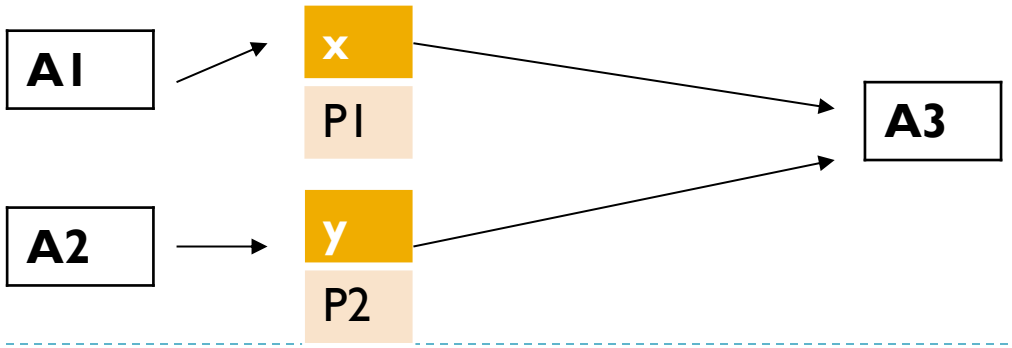
Определение управляющего оператора:

- Признак завершения А-блока



A1 < A2
A1 < A3
A4 < A3

- Признак готовности переменной



A1 < A3
A2 < A3

Алгоритм конструирования управляющих операторов

ВХОД: асинхронная программа *P* с управлением на основе строгого частичного порядка, экземпляр *A*-блока A^i_k.

ВЫХОД: код управляющего оператора C^i_k для экземпляра *A*-блока A^i_k.

ШАГИ:

1. Генерируется команда, которая установит признак завершения для A^i_k в значение ИСТИНА.
2. Формируется множество *S* экземпляров А-блоков, зависящих от A^i_k. В результате исполнения экземпляра A^i_k каждый экземпляр S^j_l потенциально может стать готовым к исполнению.

Алгоритм конструирования управляющих операторов

ШАГИ:

3. Для каждого экземпляра S^j_l:

- формируется множество U экземпляров А-блоков, которые должны исполниться до запуска S^j_l;
- генерируется команда создания переменной с именем $flag_{jl}$ и ей присваивается значение ИСТИНА;
- для каждого элемента U^p_q генерируется команда, объединяющая значение признака завершения U^p_q со значением переменной $flag_{jl}$ по принципу И;
- генерируется набор команд, который:
  - проверяет значение переменной $flag_{jl}$;
  - если оно истинно, проверяет значение триггер-функции T^j_l экземпляра S^j_l;
  - если значение триггер-функции T^j_l истинно, добавляет экземпляр S^j_l в множество готовых к исполнению экземпляров.

Система программирования Аспект

Общая архитектура системы

Язык программирования Аспект

```
program MultMatrix
preface {
   const int M = 3;
   const int N = 3;
};
data fragments
   double Frg [M][M];
code fragments
   Mult (in Frg X, Frg Y, Frg Z;
         out Frg Z)
   {
      for ( int i =0; i<M; ++i)
         for ( int k =0; k<M; ++k)
            for ( int j =0; j<M; ++j)
               Z[i][j] += X[i][k]*Y[k][j];
   };
```

```
task data
   Frg A[N][N];
   Frg B[N][N];
   Frg C[N][N];
task computations
   S[i][j][k]: Mult (A[i][k], B[k][j],
                     C[i][j], C[i][j])
      where i: 0..N -1, j: 0..N -1,
            k: 0..N -1;
task control
   S[i][j][k] < S[i][j][k +1];
end
```

Транслятор

```
inline void aplControlS ( int i, int j, int k)
{
   bool aplS_0 = ( true ) && (i >= (0) && i <= ((N -1)))
                          && (j >= (0) && j <= ((N -1)))
                          && ((k +1) >= (0) && (k +1) <= ((N -1)));
   If ( aplS_0 )
      esSpinLock ( aplSLock [i -(0)][j -(0)][( k +1) -(0)]);
   aplS [i -(0)][j -(0)][k -(0)] = true ;
      // Control for operation S
   if ( aplS_0 ) {
      bool res = true ;
      bool aplS0 = ( true ) && (i >= (0) && i <= ((N -1)))
                          && (j >= (0) && j <= ((N -1)))
                         && (k >= (0) && k <= ((N -1)));
       bool aplS1 = aplS0 ;
       aplS1 = aplS1 && aplS [i -(0)][j -(0)][k -(0)];
       res = res && ((! aplS0 && ! aplS1 ) | (! aplS0 && aplS1 ) |( aplS0 && aplS1 ));
      if ( res )
         esAddMassiveItemToQueue (0, i, j, (k +1));
   };
   if ( aplS_0 )
      esSpinUnlock ( aplSLock [i -(0)][j -(0)][( k +1) -(0)]);
};
```

Результаты тестовых испытаний

Таблица 1. Результаты измерений производительности программы умножения матриц

Реализация	**Количество ядер**				
	1	2	4	8	16
Аспект	148,98	74,73	37,52	18,98	9,68
C++/OpenMP	460,21	245,66	156,53	105,14	61,22

Таблица 2. Ускорение программы умножения матриц

Ускорение	**Количество ядер**				
	1	2	4	8	16
Аспект	1	1,99	3,97	7,84	15,39
C++/OpenMP	1	1,87	2,94	4,38	7,52

Спасибо за внимание!