Обзор систем и языков параллельного программирования

Ткачёва А.А., м.н.с., ИВМ и МГ СО РАН

Летняя школа, 2015

Проблема представления алгоритма в виде параллельной программы

Можно выделить 2 способа представления алгоритма:

Императивный (в виде набора инструкций, которые нужно выполнить в терминах, понятных компьютеру)

Плюсы: накладные расходы при реализации минимальны

Минусы: программирование в терминах, далеких от терминов предметной области

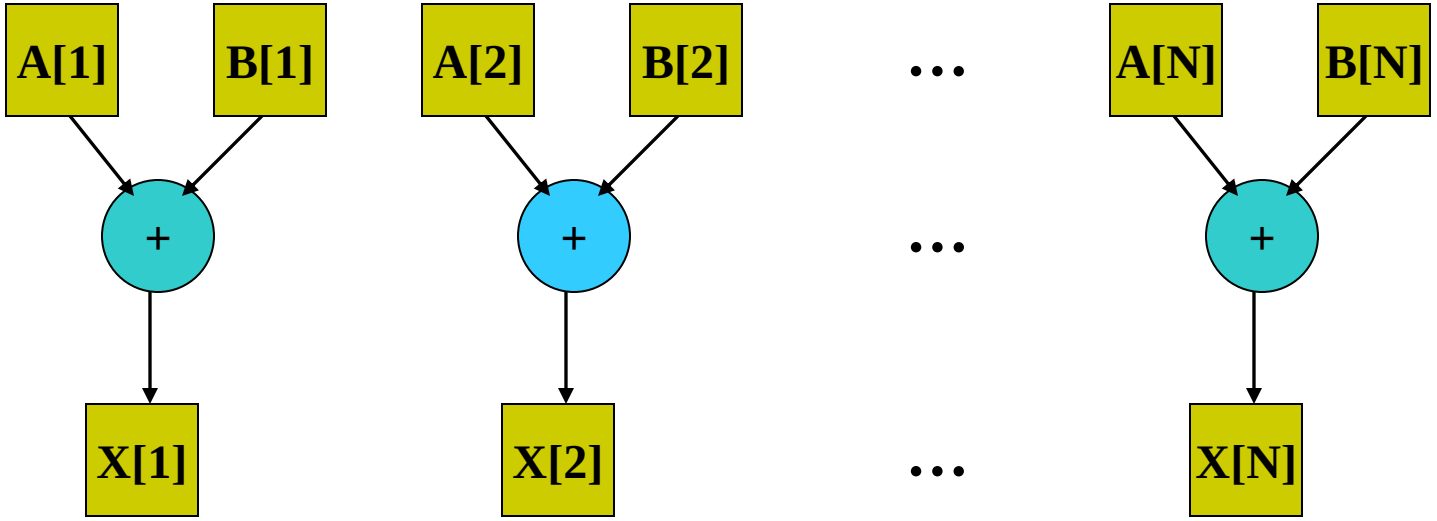
Декларативный (описывается сам алгоритм, а не способ его реализации)

Плюсы: программирование в терминах предметной области

Минусы: накладные расходы при реализации существенны

Пример

Задача: сложение векторов A и B в вектор X размера N



Императивное представление

Текст программы на C

```
for (i=0;i<N;i++)
    X[i] = A[i] + B[i];
```

Порядок исполнения инструкций

```
X[0] = A[0] + B[0];
X[1] = A[1] + B[1];
…
X[N] = A[N] + B[N];
```

Императивное представление

Текст программы на C

```
for (i=0;i<N;i++)
    X[i] = A[i] + B[i];
```

Порядок исполнения инструкций

```
X[0] = A[0] + B[0];
X[1] = A[1] + B[1];
…
X[N] = A[N] + B[N];
```

Параллелизм задан не явно

Императивное представление: как параллельно исполнять?

В императивном представление параллелизм задан не явно.

- Анализ информационных зависимостей и распараллеливания на стадии компиляции
- Введения дополнительных языковых средств, для задания параллелизма

Императивное представление

Текст программы на C + OpenMP

```
#pragma omp  for parallel
for (i=0;i<N;i++)
    X[i] = A[i] + B[i];
```

Порядок исполнения инструкций

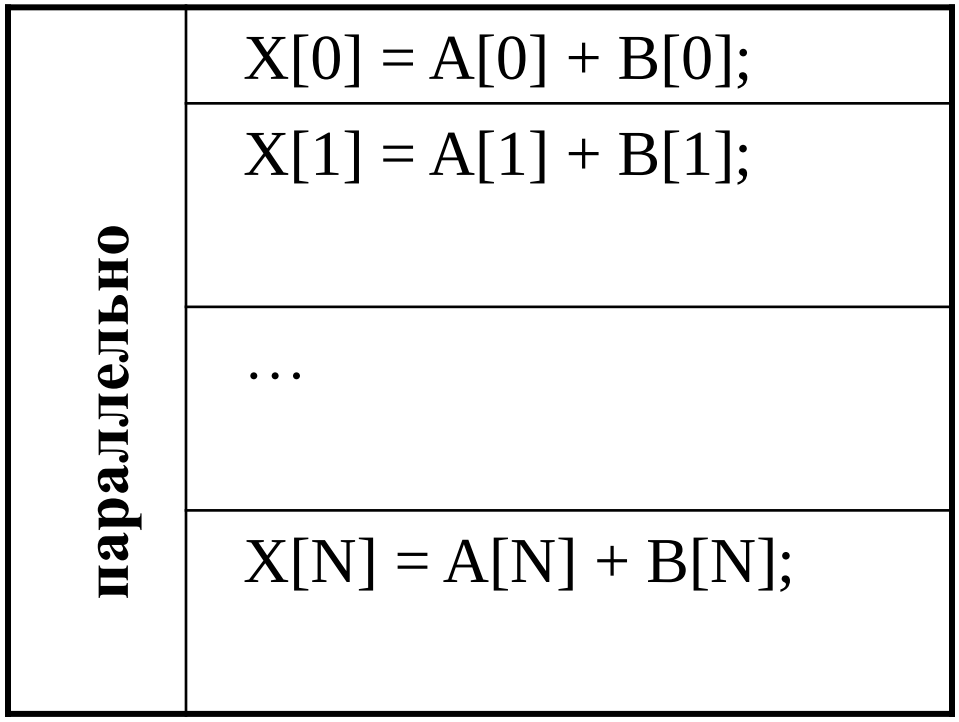
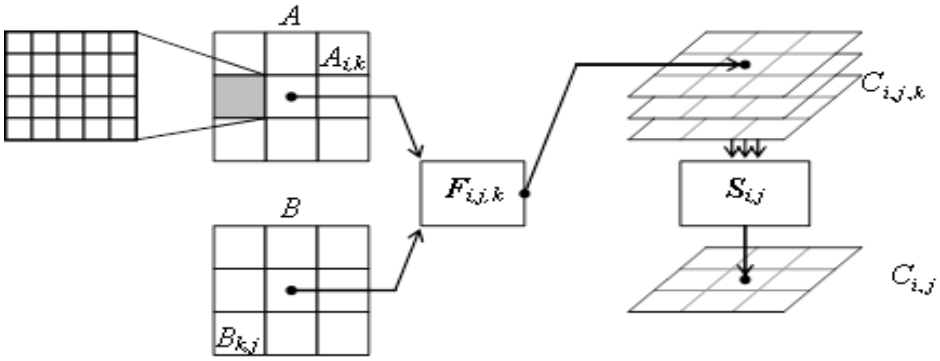


Схема фрагментированного алгоритма умножения матриц

$C_{i,j,k} = A_{i,k} \cdot B_{k,j}$ $C_{i,j} = \sum_k C_{i,j,k}$



Декларативное представление

Использование декларативного представления удобно для человека, но при этом декларативная форма, как правило, допускает множество вариантов исполнения, обладающих разными характеристиками (время выполнения, потребление памяти). В общем случае характеристики конкретного варианта исполнения автоматически сложно предсказуемы и выбор наилучшего (или близкого к нему) затруднителен.

Декларативное представление

Чтобы уменьшить накладные расходы при реализации, в языки и системы, использующие декларативное представление, вводятся дополнительные императивные средства.

Пример. LISP

LISP – язык функционального программирования

Введены средства when, loop + макросы для расширения синтаксиса языка

Цели обзора

Рассмотреть системы и языки параллельного программирования с точки зрения вводимых дополнительных языковых средств.

Структура анализа системы или языка параллельного программирования

1. Класс решаемых задач
2. Модель представления алгоритма
3. Модель исполнения программы
4. На какую архитектуру ориентирован
5. Какие динамические свойства программы поддерживаются
6. Какие языковые средства вводятся в язык для задания параллелизма или уменьшения недетерминизма

Классификация архитектур Флина

SISD (Single Instruction \ Single Data)

SIMD (Single Instruction \ Multiple Data)

MISD (Multiple Instruction \ Single Data)

MIMD (Multiple Instruction \ Multiple Data)

Классификация параллельных архитектур

1) С общей памятью

1) С распределенной памятью

Динамические свойства параллельных программ

1. Переносимость
2. Балансировка загрузки
   - статическая
   - динамическая
3. Коммуникации на фоне счета
4. Отказоустойчивость

Акторная модель

Система параллельного программирования Charm++

1. Численные алгоритмы
2. Акторная модель (единица программирования charm)
3. Message driven
4. На MIMD с или без общей памяти
5. Динамические свойства
    - •Переносимость
    - •Динамическая и начальная балансировка (заданные базовые стратегии, интерфейс для задания своей стратегии)
    - •Отказоустойчивость (check point)
6. Structured Dagger – методы, которые не обращаются к другим функции

Пример Charm++

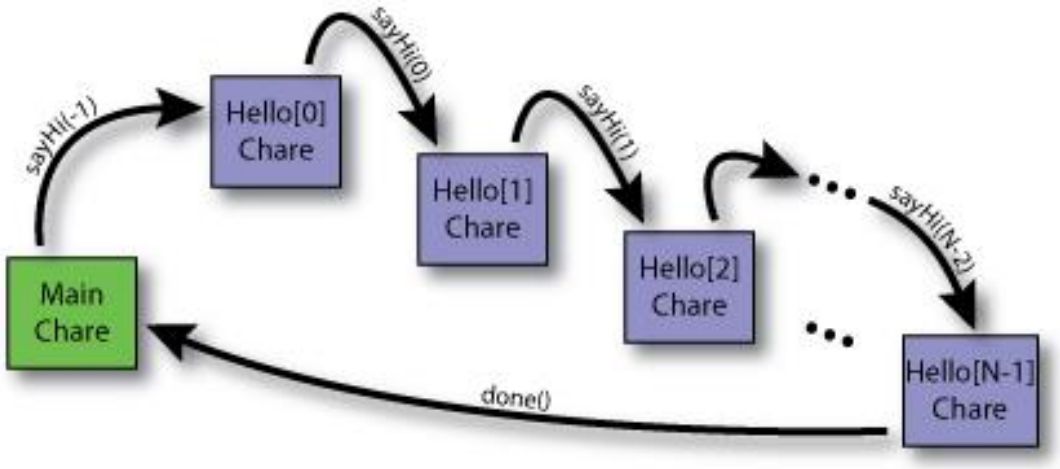


Figure: Control Flow of the Array "Hello World" Program

Пример программы: charm main

Source File *(main.C)*

```
/* readonly */ CProxy_Main mainProxy;
/* readonly */ int numElements;

// Entry point of Charm++ application
Main::Main(CkArgMsg* msg) {
numElements = 5; // Default numElements to 5
  // There should be 0 or 1 command line
arguements.
  // If there is one, it is the number of "Hello"
  // chares that should be created.
  if (msg->argc > 1)
    numElements = atoi(msg->argv[1]);

  // We are done with msg so delete it.
  delete msg;

  // Display some info about this execution
  // for the user.
  CkPrintf("Running \"Hello World\" with %d
elements "
           "using %d processors.\n",
           numElements, CkNumPes());
```

Header File *(main.h)*

```
class Main : public CBase_Main {

public:

  /// Constructors ///
  Main(CkArgMsg* msg);
  Main(CkMigrateMessage* msg);
 /// Entry Methods ///
  void done();
};
```

Interface File *(main.ci)*

```
mainmodule main {
  readonly CProxy_Main mainProxy;
  readonly int numElements;

  extern module hello;

  mainchare Main {
    entry Main(CkArgMsg* msg);
    entry void done();
  };
};
```

```
// Set the mainProxy readonly to point to a
  // proxy for the Main chare object (this
  // chare object).
  mainProxy = thisProxy;
 // Create the array of Hello chare objects. NOTE:
The
  // 'helloArray' object that is returned by 'ckNew()' is
  // actually a Proxy object to the array.
  CProxy_Hello helloArray =
CProxy_Hello::ckNew(numElements);
  // Invoke the "sayHi()" entry method on the first
  // element of the helloArray array of chare objects.
  helloArray[0].sayHi(-1);
}
// Constructor needed for chare object migration
(ignore for now)
// NOTE: This constructor does not need to appear in
the ".ci" file
Main::Main(CkMigrateMessage* msg) { }
// When called, the "done()" entry method will cause
the program
// to exit.
void Main::done() {
  CkExit();
}
```

Header File *(hello.h)*

```
class Hello : public CBase_Hello {

 public:

  /// Constructors ///
  Hello();
  Hello(CkMigrateMessage *msg);

  /// Entry Methods ///
  void sayHi(int from);
};

#endif //__HELLO_H__
```

Interface File *(hello.ci)*

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi(int);
  };

};
```

Пример: charm hello

Source File *(hello.C)*

```
extern /* readonly */ CProxy_Main mainProxy;
extern /* readonly */ int numElements;

Hello::Hello() {
  // Nothing to do when the Hello chare object
is created.
  // This is where member variables would be
initialized
  // just like in a C++ class constructor.
}

// Constructor needed for chare object
migration (ignore for now)
// NOTE: This constructor does not need to
appear in the ".ci" file
Hello::Hello(CkMigrateMessage *msg) { }
```

```
void Hello ::sayHi(int from) {

  // Have this chare object say hello to the user.
  CkPrintf("\"Hello\" from Hello chare # %d on "
          "processor %d (told by %d).\n",
          thisIndex, CkMyPe(), from);

  // Tell the next chare object in this array of chare
objects
  // to also say hello. If this is the last chare object
in
  // the array of chare objects, then tell the main
chare
  // object to exit the program.
  if (thisIndex < (numElements - 1))
    thisProxy[thisIndex + 1].sayHi(thisIndex);
  else
    mainProxy.done();
}
```

Функциональное программирование

Haskell

1. Широкий спектр, рассматривались численные алгоритмы
2. Чистые функции

Вид поддерживаемого параллелизма:

- Параллелизм доступный внутри выражений

$$F\ x + G\ y$$

- Параллелизм по данным

Haskell

3. Ленивые вычисления, мемоизация, информационные зависимости
4. SIMD , общая память
6. Гранулярность, специализированные библиотеки

OpenTS

1. Алгоритмы численного моделирования
2. Программа представляется в функциональном стиле в виде набора функций без побочных эффектов (могут быть на C, Fortran)
3. Data-flow
4. MIMD
5. Переносимость, динамическое распараллеливание
6. Гранула – размер вычислений в функции без побочных эффектов, настройка на кэш, и т.д.

SISAL

2. Единица программирования функция без побочных эффектов + единственное присваивание.
3. data-flow
6. Специализированные циклы: for-loop (все итерации цикла независимы), for-initial loop (цикл, в котором итерации цикла зависят друг от друга)

SISAL. Пример

- for initial
- i := array_liml(A);
- tmp := A[i];
- running_sum := tmp
- while i < array_limh(A) repeat
- i := old i + 1;
- tmp := A[i];
- running_sum := **old** running_sum + tmp
- returns value of running_sum
- array of running_sum
- end for

Direct Acyclic Graph

SMP Superscalar

1. Алгоритмы численного моделирования
2. Программа на C
3. DAG (direct acyclic graph)
4. Системы с общей памятью
5. Гранула – размер данных
6. Точки синхронизации (подождать вычисления какого-то фрагмента)

Пример. Умножение матриц

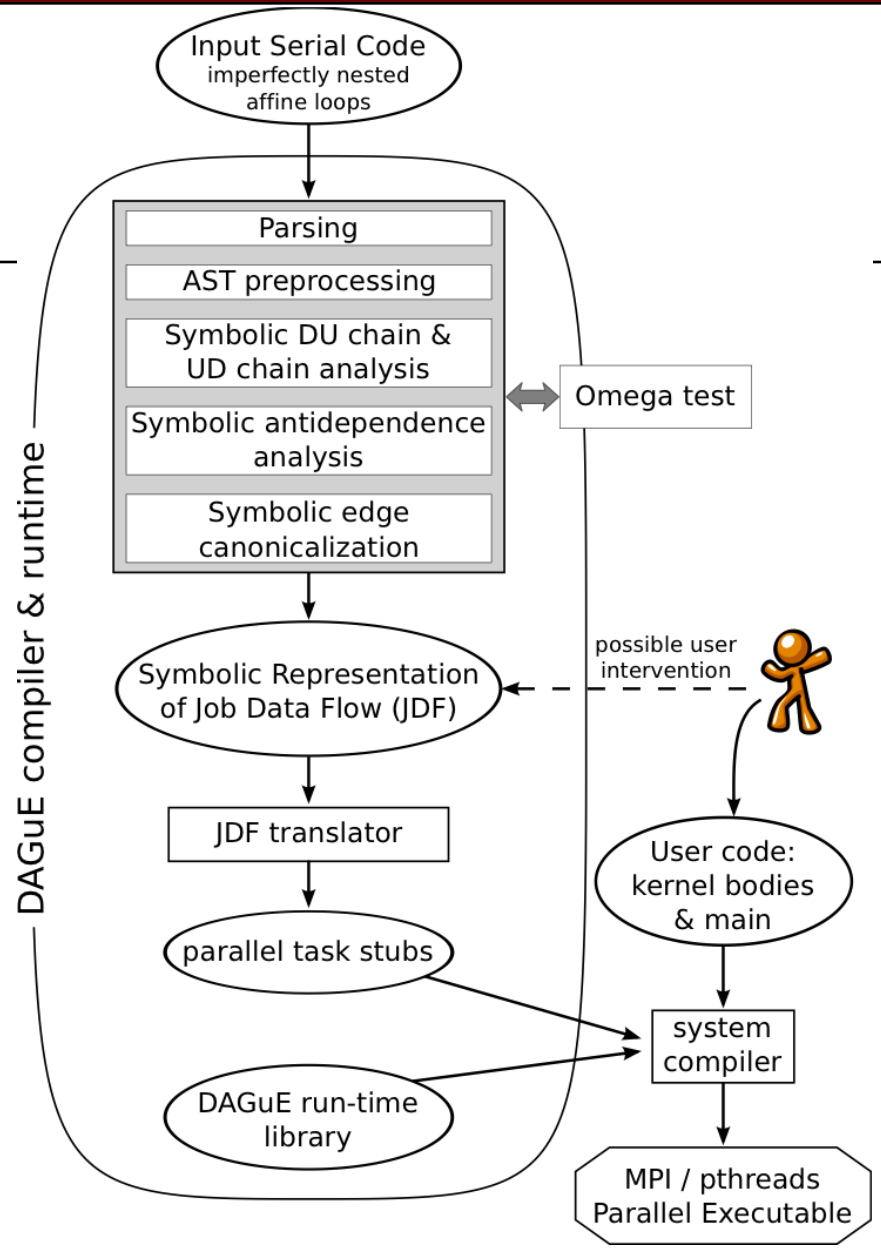
```
#pragma gss task input (A, B) inout (c)

static void block_addmultiply( double C[BS][BS], double A[BS][BS], double B[BS][BS]){
        int i, j, k_;
        for (i = 0; i < BS; i++)
                for (j = 0; j < BS; j++)
                        for (k =0; k < BS; k++)
                                C[i][j] += A[i][k] * B[k][j];
}

int main (int argc, char **argv){
        int i, j, k;
        initialize (argc, argv,  A, B, C);
        for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                        for (k =0; k < N; k++)
                                block_addmultiply(C[i][j], A[i][k], B[k][j]);
        ...
}
```

DPLASMA + PaRSEC

1.Библиотека линейной алгебры на плотных матрицах

3. DAG (direct acyclic graph)

4. Системы с общей и распределенной памятью


Input Serial Code
imperfectly nested
affine loops
DAGuE compiler & runtime
Parsing
AST preprocessing
Symbolic DU chain &
UD chain analysis
Symbolic antidependence
analysis
Symbolic edge
canonicalization
Omega test
Symbolic Representation
of Job Data Flow (JDF)
possible user
intervention
JDF translator
parallel task stubs
DAGuE run-time
library
User code:
kernel bodies
& main
system
compiler
MPI / pthreads
Parallel Executable

Заключение

Как показал обзор ввод дополнительных императивных средств – один из методов уменьшить накладные расходы, связанные с реализацией.

Виды средств: приоритеты, точки синхронизации, гранулярность вычислений, циклы специального вида, ограничение круга решаемых задач.

Спасибо за внимание!