Паралелно Програмиране

Модели за паралелно програмиране.
Координация в паралелните алгоритми.

доц. д-р Александър Пенев

Модели за Паралелно Програмиране

Модели за Паралелно Програмиране

- Споделена памет (Shared memory)
  Задачите използват общо адресно пространство (обща памет), в която могат да пишат и четат асинхронно. Използват се различни механизми за защита и контрол на достъпа до общата памет
- Нишки (Threads)
  Нишките са подпрограми в главната програма. Те комуникират помежду си през глобална памет. Примери за този модел са Posix Threads, OpenMP и др,
- Предаване на съобщения (Message Passing)
  Множество от задачи, използващи своя собствена локална памет за изчисленията. Обмен на данни става чрез изпращане и получаване на съобщения. Трансфера на данни обикновено е кооперативен т.е. изпращащата страна трябва да има получател. Пример: MPI
- Неявно взаимодействие (Implicit interaction)
  В неявен модел, никакво взаимодействие между процесите не е видимо за програмиста. Вместо това компилаторът и/или изпълнителят е отговорен за неговото изпълнение
- Данново паралелен (Data Parallel)
  Множество от задачи, работещи колективно върху обща структура от данни. Всяка задача извършва едно и също действие със своята част от данните
- Хибриден (Hybrid)

Модели за Паралелно Програмиране

- Една програма много даннни (Single Program Multiple Data, SPMD);
- Много програми много данни (Multiple Program Multiple Data, MPMD);

Модели за паралелно програмиране

Моделите за паралелно програмиране съществуват като още една абстракция над архитектурите на хардуера и паметта.

Те не са свързани с конкретен хардуер или архитектура и могат да бъдет реазлизирани безпроблемно върху всяка една;

Споделена памет (Shared memory)

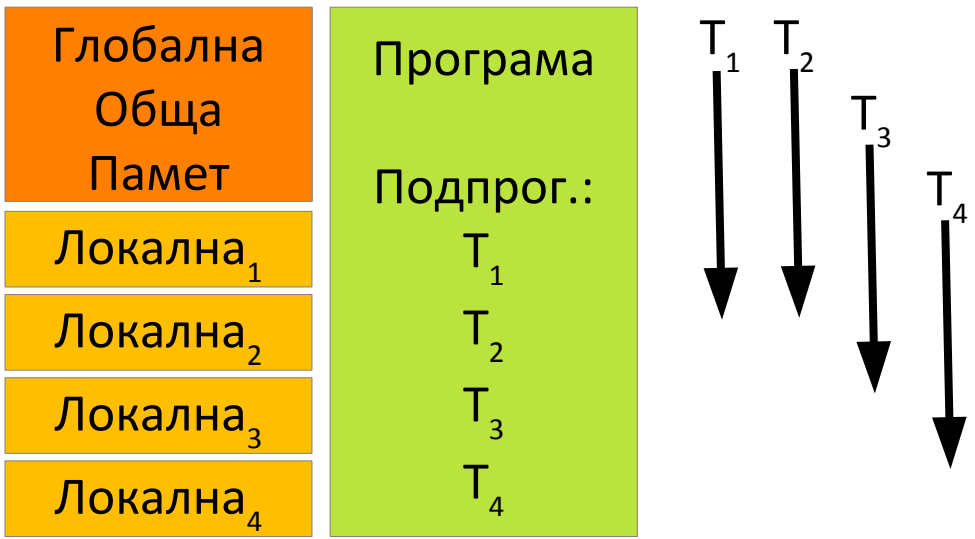
- В този модел задачите споделят общо адресно пространство (обща памет), в което те четат и записват асинхронно;
- За контрол на достъпа до споделената памет могат да се използват различни механизми като заключване и семафори;
- Предимство на този модел от гледна точка на програмиста е, че липсва понятието „собственост“ на данни, така че не е необходимо да се уточнява изрично комуникацията на данните между задачите. Разработването на програми е максимално опростено;
- Важен недостатък (от гледна точка на производителността) е, че става трудно да се разбере и управлява местоположението на данните;

Споделена памет – реализация

- В платформите за споделена памет компилаторите превеждат променливите на програмата в действителни адреси на паметта, които са глобални;

Нишки (Threads)

- В много нишковия модел на паралелно програмиране, един процес може да има множество, едновременни пътища за изпълнение;
- Нишките са подпрограми в главната програма. Те комуникират помежду си през глобална памет;
- Примери за този модел са Posix Threads, OpenMP и др.;



Нишки (Threads)

- Основната програма се изпълнява от ОС;
- Изпълнява някаква последователна (серийна) работа, след това създава задачи (нишки), които могат да бъдат планирани и изпълнявани от ОС едновременно;
- Всяка нишка има локални данни, но също така споделя всички ресурси програмата;
- Работата на нишката може най-добре да се опише като подпрограма в основната програма. Всяка нишка може да изпълни всяка подпрограма едновременно с другите нишки;
- Нишките комуникират помежду си чрез глобална памет, което изисква конструкции за синхронизация, за да се гарантира, че повече от една нишка не актуализира един и същ глобален адрес в даден момент;
- Нишките могат да се създават и унищожават по време на прог.;

Нишки (Threads) – реализация

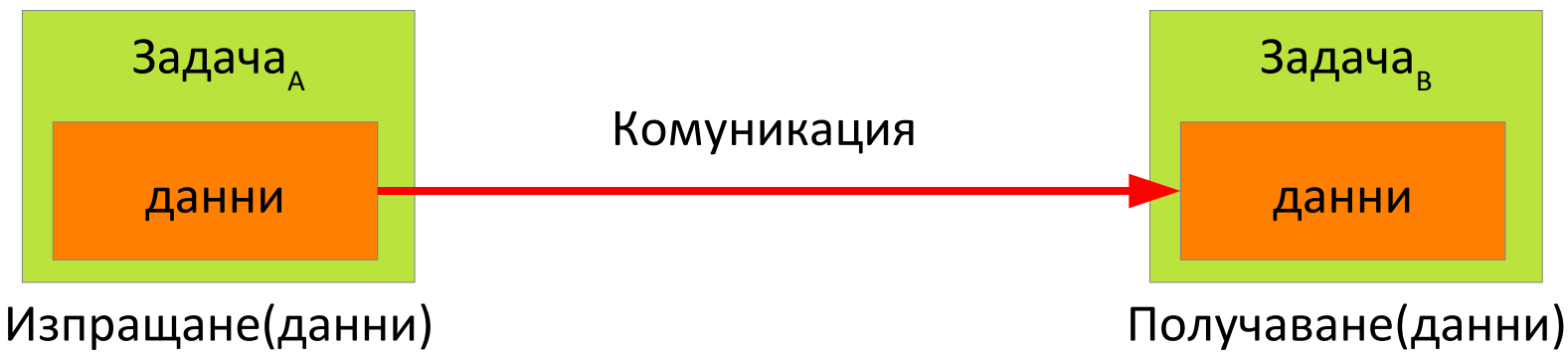
От гледна точка на програмирането, реализациите на нишки обикновено включват:

- Библиотека от подпрограми (методи), които се извикват от паралелен изходен код;
- Набор директиви за компилатора, вложени в сериен или паралелен изходен код;

И в двата случая програмистът е отговорен за определянето на целия паралелизъм.

Предаване на съобщения (Message Passing)

- Множество от задачи, използващи своя собствена локална памет за изчисленията;
- Обмен на данни става чрез изпращане и получаване на съобщения;
- Трансфера на данни обикновено е кооперативен т.е. изпращащата страна трябва да има получател;
- Пример: MPI



Предаване на съобщения – реализация

- Реализира се като библиотека от функции, които се използват в изходния код на паралелните програми;
- Програмистът отговаря за определянето на целия паралелизъм;
- През 80-те години на миналия век възникват множество библиотеки за предаване на съобщения. Това са различни (и несъвместими помежду си) реализации;
- През 1992 г. е създаден **MPI forum** с основната цел да се създаде стандартен интерфейс за предаване на съобщения;
- MPI (и MPI-2) е "фактически" индустриален стандарт за предаване на съобщения в паралелни приложение (на HPC);
- За архитектури със споделена памет, MPI реализациите обикновено не използват мрежова комуникация, а вместо това те използват споделена памет за по-добра произвдителност;

Неявно взаимодействие (Implicit interaction)

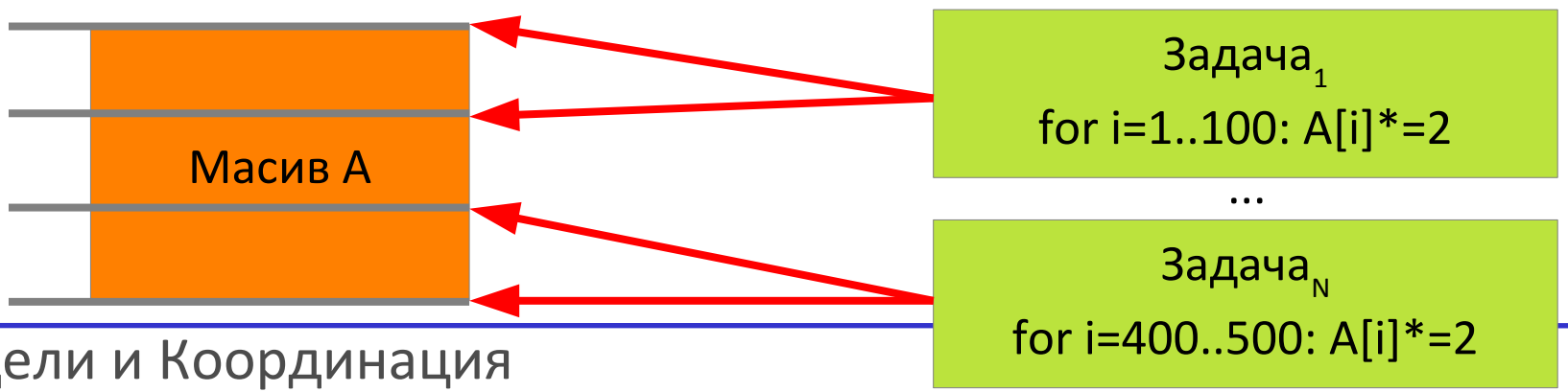
- В неявен модел, никакво взаимодействие между процесите не е видимо за програмиста;
- Вместо това компилаторът и/или изпълнителят е отговорен за неговото изпълнение;
- Примери за този вид модел може да се видят в някои домейн-специфични езици и функционалните езици;
- Този тип паралелизъм се контролира трудно;

Неявно взаимодействие (Implicit interaction)

- Други примери за този модел са:
  - Автоматичната паралелизация на ниво компилатор;
  - Супер-скаларното изпълнение;
  - ILP;

Данново паралелен (Data Parallel)

- По-голямата част от паралелната работа се фокусира върху извършването на операции върху набор от данни;
- Наборът от данни обикновено е организиран в обща структура, като например масив;
- Набор от задачи работят съвместно върху една и съща структура на данни, но всяка задача работи на различен дял от тази структура;
- Задачите изпълняват една и същата операция над своя дял от работата, например "умножете с 2 всеки елемент от масив".



Данново паралелен (Data Parallel)

- В архитектурите със споделена памет всички задачи имат достъп до структурата от данни, чрез глобална памет;
- В разпределените архитектури на паметта структурата от данни е разделена и се намира като "парчета" в локалната памет на всяка задача;

Данново паралелен – реализация

- Програмирането с данново паралелен модел, обикновено се осъществява чрез писане на програма с данново паралелни конструкции;
- Те са или част от езика и съответно компилатора ги превежда (много често до MPI извиквания), или са реализирани както библиотеки съдържащи класове и методи/функции от данново паралелни типове/конструкции;

Хибриден модел (Hibrid)

- Комбинация от два или повече от другите модели;
- Например: MPI+OpenMP или MPI+Pthreads или MPI+DataParallel;

Координация в Паралелните Алгоритми

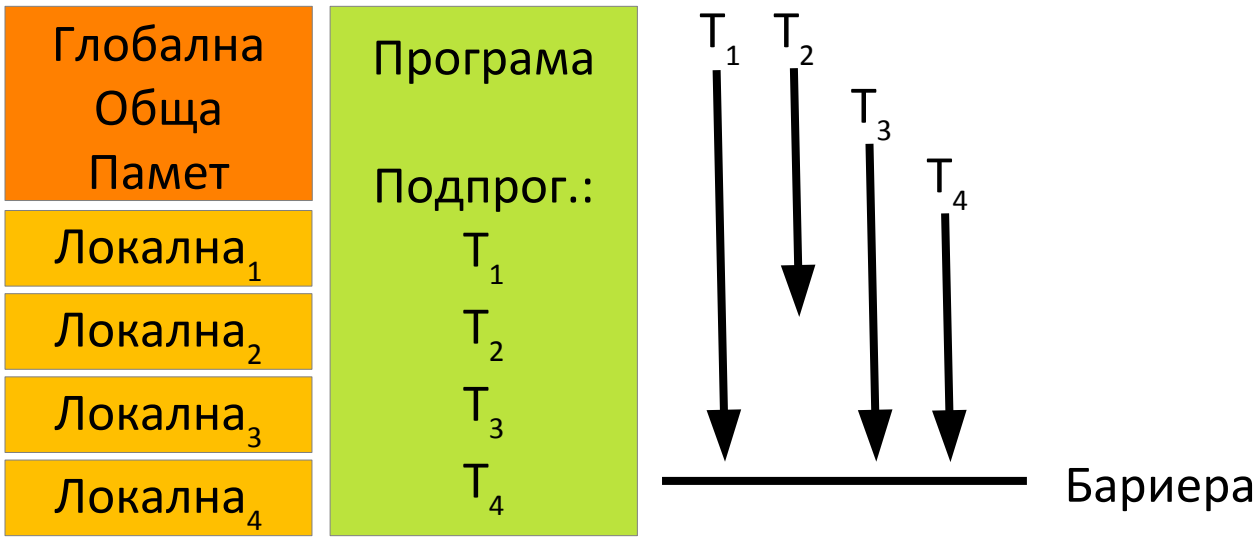
Видове координация / синхронизация (на процеси и на данни)

- Бариери (Barrier)
  Бариера в сорс кода за група нишки или процеси се мястото, в което всички нишки/процеси трябвада спрат, докато всички други (от групата) не достигнат тази “бариара”;
- Заключвания/Семафори (Lock/Semaphore)
  Обикновено се използва за сериализиране (защита) на достъпа до глобални данни или общ код;
- Синхронни комуникационни операции (Synchronous communication operations);

Бариери (Barrier)

- Обикновено се предполага, че всички задачи се изпълняват;
- Всяка задача изпълнява своята работа, докато достигне бариерата;
- След това задачата спира или "блокира";
- Когато последната задача достигне бариерата, всички задачи са синхронизирани;
- Какво става след бариерата може да варира в зависимост от реализацията, например може да се продължи с изпълнението на последователната (серийната) част от кода; може задачите да продължат работата си и др.;

Бариери (Barrier)



Бариери (Barrier) – пример (C++11)

```
void DoWork() {
    Tasks& tasks;
    int n_threads;
    vector<thread*> workers;

    barrier task_barrier(n_threads);

    for (int i = 0; i < n_threads; ++i) {
      workers.push_back(new thread([&] {
        bool active = true;
        while(active) {
          Task task = tasks.get();
          // perform task
```

Бариери (Barrier) – пример (C++11)

```
            // perform task
            ...
            task_barrier.arrive_and_wait();
          }
        });
    }
    // Read each stage of the task until all are complete.
    while (!finished()) {
      GetNextStage(tasks);
    }
}
```

Заключвания/Семафори (Lock/Semaphore)

- Може да участват произволен брой задачи;
- Обикновено се използва за сериализиране (защита) на достъпа до глобални данни или общ код;
- Само една задача в даден момент може да използва (притежава) заключването / семафора / флага;
- Първата задача, която “заключи” променливата и задава стойност, след това тази задача може безопасно (серийно) да получи достъп до защитените данни или код;
- Други задачи могат да се опитат да придобият заключване, но трябва да изчакат, докато задачата, която “държи”заключването го освободи;

Заключване – пример (C#)

```
using System; using System.Threading;
class Program {
  static readonly object _object = new object();
  static void A() {
    lock (_object) {
      Thread.Sleep(100);
      Console.WriteLine(Environment.TickCount);
    }
  }
  static void Main() {
    for (int i = 0; i < 10; i++) {
      ThreadStart start = new ThreadStart(A);
      new Thread(start).Start();
    }
  }
}
```

Заключвания/Семафори (Lock/Semaphore)

- Има блокиращи и не блокиращи семафори;
- Има “бинарни” семафори, наречени мутекс (mutex);
- В общия случай семафора съдържа броя на свободните ресурси, ако те са повече от един, като при използване (aquire) променливата се намаля с 1, а при освобождаване (release) се увеличава с 1. Блокирането на задачи е когато ресурси вече няма т.е. семафора е 0;
- Реализацията на семафори и други подобни, обикновено използва специални (атомарни) инструкции (от тип Test-and-Set) в съвременните процесори, като например BTS, CMPXCHG и др.;

Синхронни комуникационни операции

- Включва само онези задачи, изпълняващи комуникационна операция;
- Когато дадена задача извършва комуникационна операция е необходима някаква форма на координация с другата задача(и), участващи в комуникацията. Например, преди дадена задача може да извърши операция за изпращане, първо трябва да получи потвърждение от получаващата задача, че е добре да изпрати;