Паралелно Програмиране

Езици и Библиотеки (API) за паралелно програмиране

доц. д-р Александър Пенев

Паралелност на различните нива на системата

- На ниво Език за програмиране:
  - Канали (Channel);
  - Съпрограми (Coroutines);
  - Фючърси и „Обещания“ (Futures and Promises);
- На ниво ОС:
  - Многозадачност – кооперативна многозадачност и превантивна (preemptive) многозадачност;
  - Време деление, което заменя последователната „пакетна обработка“ на задачите с едновременна употреба на система
  - Процеси (Process);
  - Нишки (Thread);
- На ниво хардуер и мрежа: по начало са паралелни (различни у-ва);

Езици за Паралелно Програмиране (ЕПП)

Езици с елементи на ПП

- Ada – ключова дума task и др. вградени в езика;
- Cilk, Cilk++, Cilk Plus – C и C++ базирани езици, в които чрез допълнителни ключови думи се дефинира паралелизъм (Fork-Join концепцията);
- Cω – C Omega е изследователски език, разширяваш възможностите на C# с асинхронни комуникации;
- Clojure – диалект на Lisp, изпълняван на Java VM. Силно функционален, поддържа STM и др.;
- Co-array Fortran;
- Concurrent Pascal;
- Eiffel;
- Elixir – функционален език. Конкурентност на базата на леки нишки, които могат да комуникират чрез съобщения;
- Emerald – използва нишки и монитори;

Езици с елементи на ПП

- Erlang – използва асинхронни предавания на съобщения и „nothing shared” подход;
- Gambit Scheme – функционален. Прилага мощен и прост модел на предаване на съобщения, както и вдъхновен от Erlang модел на конкурентност;
- Go – CSP модел, goroutines (вариант на съпрограми), асинхронен, channels, и др.;
- Java;
- Julia;
- Occam – Базиран на Communicating Sequential Processes (CSP);
- Occam-π – модерен вариант на occam, като са добавени и идеи от π-calculus;

Езици с елементи на ПП

- Orc – недетерминиран, разпределен и конкурентен език;
- P;
- Pict – π-calculus базиан;
- Rust – паралелизъм базиран на Send, Sync и Thread;
- Scala – реализира Erlang-стил actors върху JVM;
- SequenceL – чисто функционален, автоматично паралелизиращ;
- Unified Parallel C;
- Modula-2 – наследник на Pascal. Има поддръжка на coroutines;
- Modula-3 – поддържа threads, mutexes, condition variables;
- SuperPascal – базиран на Concurrent Pascal и Joyce;
- VHSIC Hardware Description Language (VHDL)—IEEE STD-1076;
- ...

Пример Occam

```
-- occam
PROC write.string(CHAN output, VALUE string[])=
  SEQ character.number = [1 FOR string[BYTE 0]]
    output ! string[BYTE character.number]

write.string(terminal.screen, "Hello World!")

var ch
PAR
  terminal.keyboard ? ch
  terminal.screen ! "."
```

Библиотеки за Паралелно Програмиране (API)

Библиотеки (API) за ПП

- OpenMP;
- MPI, MPI-2;
- Java concurrency framework;
- Task Parallel Library for .NET;
- Threading Building Blocks (TBB);
- CUDA;
- OpenCL;
- OpenHMPP;
- Apache Hadoop;
- Apache Spark;
- Apache Flink;
- Apache Beam;
- ...

Примери и Често Използвани ЕПП и API

C++11

(Futures, Promises, ...)

Фючърси (Futures) C++11

- Atomic;
- Thread;
- Mutex;
- Condition variable;
- Future & Promises;
- ...

Фючърси (Futures) C++11

```
#include <iostream>          // std::cout
#include <future>            // std::async, std::future
#include <chrono>            // std::chrono::milliseconds

// a non-optimized way of checking for prime numbers:
bool is_prime(int x) {
  for (int i=2; i<x; ++i) if (x%i==0) return false;
  return true;
}
```

Фючърси (Futures) C++11

```
int main() {
  // call function asynchronously:
  std::future<bool> fut = std::async(is_prime, 444444443);

  // do something while waiting for function to set future:
  std::cout << "checking, please wait";
  std::chrono::milliseconds span(100);
  while (fut.wait_for(span)==std::future_status::timeout)
    std::cout << '.' << std::flush;

  bool x = fut.get(); // retrieve return value

  std::cout << "\n444444443 " << (x?"is":"is not") << " prime.\n";

  return 0;
}
```

Обещания (Promises) C++11

```
#include <iostream>          // std::cout
#include <functional>        // std::ref
#include <thread>            // std::thread
#include <future>            // std::promise, std::future

void print_int(std::future<int>& fut) {
  int x = fut.get();
  std::cout << "value: " << x << '\n';
}

int main() {
  std::promise<int> prom;                      // create promise
  std::future<int> fut = prom.get_future(); // engag. with future
  std::thread th1(print_int, std::ref(fut));// send future to thr.
  prom.set_value(10);                          // fulfill promise
  th1.join();            // (synchronizes with getting the future)
  return 0;
}
```

C#

(Threads, Parallel.For, PLINQ, Tasks, Futures, ...)

- Threads (System.Threading);
- Task Parallel Library (TPL) – Tasks.Parallel, Parallel.For, Parallel.ForEach, Parallel.Invoke, ... (System.Threading.Tasks);
- PLINQ;
- Работа с Task обекти, представящи асинхронни операции;
- Futures;
- async + await, yield;

Tasks

```
using System.Threading; using System.Threading.Tasks;
class Program {
  static void Main(string[] args) {
    Task<int[]> parent = new Task<int[]>(() => {
      var results = new int[3];
      new Task(() => {Thread.Sleep(15000); results[0] = 0;},
        TaskCreationOptions.AttachedToParent).Start();
      new Task(() => results[1] = 1,
        TaskCreationOptions.AttachedToParent).Start();
      new Task(() => results[2] = 2,
        TaskCreationOptions.AttachedToParent).Start();
      return results;
    });
    // ...
```

Tasks

```
    // ...

    parent.Start();
    var finalTask = parent.ContinueWith(
      parentTask => {
        foreach (int i in parentTask.Result)
          Console.WriteLine(i);
      });
    finalTask.Wait();
  }
}
```

Parallel.For

```
using System.Linq;
using System.Threading; using System.Threading.Tasks;

class Program {
  static void Main(string[] args) {
    Parallel.For(0, 10, i => {
      Thread.Sleep(1000);
    });
    var numbers = Enumerable.Range(0, 10);
    Parallel.ForEach(numbers, i => {
      Thread.Sleep(1000);
    });
  }
}
```

PLINQ

```
var source = Enumerable.Range(1, 10000);

// Opt in to PLINQ with AsParallel.
var evenNums = from num in source.AsParallel()
               where num % 2 == 0
               select num;

Console.WriteLine("{0} even numbers out of {1} total",
                  evenNums.Count(), source.Count());
```

Резултат:

```
5000 even numbers out of 10000 total
```

Java

(Threads, Futures, Streams, ...)

Поддръжка на паралелни примитиви в езика и във framework-а

- Синхронизирани методи (като част от ЕП – ключова дума);
- Threads (java.lang.Thread и java.lang.Runnable);
- Futures, (java.util.concurrent);
- Atomic и Locks (java.util.concurrent.atomic, java.util.concurrent.locks);
- Streams (java.util.stream);

Синхронизирани методи (lock)

```
public synchronized void critical() {
  // some thread critical stuff here
}

public void add(String site) {
  synchronized (this) {
    if (!crawledSites.contains(site)) {
      linkedSites.add(site);
    }
  }
}
```

Thread

```
class PrimeThread extends Thread {
  long minPrime;

  PrimeThread(long minPrime) {
    this.minPrime = minPrime;
  }

  public void run() {
    // compute primes larger than minPrime ...
  }
}

// Create and start thread
PrimeThread p = new PrimeThread(143);
p.start()
```

Runnable

```
class PrimeRun implements Runnable {
  long minPrime;
  PrimeRun(long minPrime) {
    this.minPrime = minPrime;
  }

  public void run() {
    // compute primes larger than minPrime ...
  }
}

// Create a thread and start it running
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Locks

```
class X {
   private final ReentrantLock lock = new ReentrantLock();

   public void m() {
     lock.lock();  // block until condition holds
     try {
       // ... method body
     } finally {
       lock.unlock()
     }
   }
 }
```

Брояч без Atomics

```
public class SafeCounterWithLock {
  private volatile int counter;

  public synchronized void increment() {
    counter++;
  }
}
```

Брояч с Atomics

```
public class SafeCounterWithoutLock {
  private final AtomicInteger counter = new AtomicInteger();

  public void increment() {
    while(true) {
      int existingValue = counter.getValue();
      int newValue = existingValue + 1;
      if(counter.compareAndSet(existingValue, newValue)) {
        return;
      }
    }
  }
}
```

Streams

```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Lokesh");

memberNames.stream().filter((s) -> s.startsWith("A"))
                    .forEach(System.out::println);
```

Резултат:

```
Amitabh
Aman
```

Streams

```
memberNames.stream().filter((s) -> s.startsWith("A"))
                    .map(String::toUpperCase)
                    .forEach(System.out::println);
```

Резултат:

```
AMITABH
AMAN
```

Streams

```
memberNames.stream().sorted()
                    .map(String::toUpperCase)
                    .forEach(System.out::println);
```

Резултат:

```
AMAN
AMITABH
LOKESH
RAHUL
SALMAN
SHAHRUKH
SHEKHAR
```

Streams

```
List<String> memNamesInUppercase =
memberNames.stream().sorted()
                    .map(String::toUpperCase)
                    .collect(Collectors.toList());

System.out.print(memNamesInUppercase);
```

Резултат:

```
[AMAN, AMITABH, LOKESH, RAHUL, SALMAN, SHAHRUKH, SHEKHAR]
```

Streams

```
Optional<String> reduced = memberNames.stream()
                           .reduce((s1,s2) -> s1 + "#" + s2);

reduced.ifPresent(System.out::println);
```

Резултат:

```
Amitabh#Shekhar#Aman#Rahul#Shahrukh#Salman#Lokesh
```

Streams

```
public class StreamBuilders {
  public static void main(String[] args){
    List<Integer> list = new ArrayList<Integer>();
    for(int i = 1; i< 10; i++){
      list.add(i);
    }
    // Here creating a parallel stream
    Stream<Integer> stream = list.parallelStream();
    Integer[] evenNumbersArr = stream
      .filter(i -> i%2 == 0).toArray(Integer[]::new);
    System.out.print(evenNumbersArr);
  }
}
```

OpenMP

(Fork-Join, #pragma, ...)

Базиран основно Fork-Join

- Използва `#pragma` директиви на компилатора, за да “подскаже” на компилатора как искаме при възможност да се разпаралели програмата;
- Ако компилатора не поддържа OpenMP или не са зададени правилните параметри на компилатора, то директивата се игнорира и програмата се компилира както нормално;
- Shared memory многозадачен модел;
- Базиран на Fork-Join модела;
- Могат да се задават бариери и много други с параметрите на директивата `#pragma omp ...`

OpenMP пример

```
void simple(int n, float *a, float *b) {
  int i;

  #pragma omp parallel for
  for (i=1; i<n; i++) /* i is private by default */
    b[i] = (a[i] + a[i-1]) / 2.0;
}
```

MPI

(Комуникация, Синхронизация, ...)

MPI

- Комуникация (MPI_Send, MPI_Recv, MPI_Sendrecv и др.);
- Разпределяне на работата;
- Глобални редукции (MPI_Reduce, MPI_Op_create, MPI_Allreduce, MPI_Reduce_scatter, MPI_Scan и др.);
- Синхронизация (MPI_Barrier, синхронни комуникации и др.);
- Дефиниране на потребителски типове;
- Дефиниране на виртуални топологии;
- Паралелен В/И (MPI-2);
- Стартиране на процеси (MPI-2)
- И др.;

MPI – пример (Hello World)

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    MPI_Init(NULL, NULL); // Initialize the MPI environment
    int world_size; // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int world_rank; // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    char processor_name[MPI_MAX_PROCESSOR_NAME]; // Name
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    printf("Hello world from proc %s, rank %d out of %d procs\n",
           processor_name, world_rank, world_size);

    MPI_Finalize(); // Finalize the MPI environment.
}
```

MPI – пример (Hello World)

host_file:
```
host1name
host2name
host3name
host4name
```

```
> export MPIRUN=/.../mpirun
> export MPI_HOSTS=host_file
> mpirun -n 4 -f host_file ./mpi_hello_world
Hello world from proc host2name, rank 1 out of 4 procs
Hello world from proc host1name, rank 0 out of 4 procs
Hello world from proc host4name, rank 3 out of 4 procs
Hello world from proc host3name, rank 2 out of 4 procs
```

MPI – пример (Редукции, SUM и AVG)

```
float *rand_nums = NULL;
rand_nums = create_rand_nums(num_elements_per_proc);

// Sum the numbers locally
float local_sum = 0;
int i;
for (i = 0; i < num_elements_per_proc; i++) {
  local_sum += rand_nums[i];
}

// Print the random numbers on each process
printf("Local sum for process %d - %f, avg = %f\n",
       world_rank, local_sum, local_sum / num_elements_per_proc);
```

MPI – пример (Редукции, SUM и AVG)

```
// Reduce all of the local sums into the global sum
float global_sum;
MPI_Reduce(&local_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0,
           MPI_COMM_WORLD);

// Print the result
if (world_rank == 0) {
  printf("Total sum = %f, avg = %f\n", global_sum,
         global_sum / (world_size * num_elements_per_proc));
}
```

MPI – пример (Редукции, SUM и AVG)

```
> mpirun -n 4 ./reduce_avg 100

Local sum for process 0 - 51.385098, avg = 0.513851
Local sum for process 1 - 51.842468, avg = 0.518425
Local sum for process 2 - 49.684948, avg = 0.496849
Local sum for process 3 - 47.527420, avg = 0.475274
Total sum = 200.439941, avg = 0.501100
```

POSIX Threads, Boost.Thread, TBB, ...

(Нишки, Синхронзация, ...)

Поддържат

- Поддържат създаване и управление на Нишки (Threads);
- Fork-Join;
- Mutexes;
- Condition variables;
- Синхронизация, read/write locks и бариери;
- Различни готови за използване „паралелни“ базови структури от данни;
- И др.;

pthreads – пример

```
#include <pthread.h>
#include <stdio.h>

/* this function is run by the second thread */
void *inc_x(void *x_void_ptr) {
  /* increment x to 100 */
  int *x_ptr = (int *)x_void_ptr;
  while(++(*x_ptr) < 100);

  printf("x increment finished\n");

  /* the function must return something - NULL will do */
  return NULL;
}
```

pthreads – пример

```
int main() {
  int x = 0, y = 0;

  /* show the initial values of x and y */
  printf("x: %d, y: %d\n", x, y);

  /* this variable is our reference to the second thread */
  pthread_t inc_x_thread;

  /* create a second thread which executes inc_x(&x) */
  if(pthread_create(&inc_x_thread, NULL, inc_x, &x)) {
    fprintf(stderr, "Error creating thread\n"); return 1;
  }
```

pthreads – пример

```
    /* increment y to 100 in the first thread */
    while (++y < 100);

    printf("y increment finished\n");

    /* wait for the second thread to finish */
    if(pthread_join(inc_x_thread, NULL)) {
      fprintf(stderr, "Error joining thread\n"); return 2;
    }

    /* results - x is now 100 thanks to the second thread */
    printf("x: %d, y: %d\n", x, y);

    return 0;
}
```

Boost – пример

```
#include <boost/thread.hpp>
#include <boost/chrono.hpp>
#include <iostream>
void wait(int seconds) { boost::this_thread::sleep_for(
                           boost::chrono::seconds{seconds}); }
void thread() {
  for (int i = 0; i < 5; ++i) {
    wait(1);
    std::cout << i << '\n';
  }
}
int main() {
  boost::thread t{thread};
  t.join();
}
```

TBB – пример

```
#include "tbb/task_group.h"
using namespace tbb;
int Fib(int n) {
  if (n<2) {
    return n;
  } else {
    int x, y;
    task_group g;
    g.run([&]{x=Fib(n-1);}); // spawn a task.
    g.run([&]{y=Fib(n-2);}); // spawn another task.
    g.wait();                // wait for both tasks to
                             // complete.
    return x+y;
  }
}
```

CUDA

(Compute Unified Device Architecture)

CUDA

- Това е платформа за паралелни изчисления и API създадена от Nvidia за работа и програмиране на техните GPGPU;
- Поддържа езиците C, C++ и Fortran;
- Платформата включва и множество силно оптимизирани библиотеки с математически и други функции като cuBLAS, cuFTT, cuRAND и др;
- Включени са и средства за създаване, дебъгиране, анализ и оптимизация на приложения базирани на CUDA;
- Подобно на OpenCL работата се базира на създаване на kernel-и и работа с тях;

Пример – Hello, CUDA!

```
#include <stdio.h>

const int N = 16;
const int blocksize = 16;

__global__
void hello(char *a, int *b) {
 a[threadIdx.x] += b[threadIdx.x];
}

int main() {
   char a[N] = "Hello \0\0\0\0\0\0";
   int b[N] = {15, 10, 6, 0, -11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
   char *ad; int *bd;
   const int csize = N*sizeof(char);
   const int isize = N*sizeof(int);
   printf("%s", a);
```

Пример – Hello, CUDA!

```
    cudaMalloc( (void**)&ad, csize );
    cudaMalloc( (void**)&bd, isize );
    cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
    cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

    dim3 dimBlock( blocksize, 1 );
    dim3 dimGrid( 1, 1 );
    hello<<<dimGrid, dimBlock>>>(ad, bd);
    cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
    cudaFree( ad );
    cudaFree( bd );

    printf("%s\n", a);

    return EXIT_SUCCESS;
}
```

Пример 2

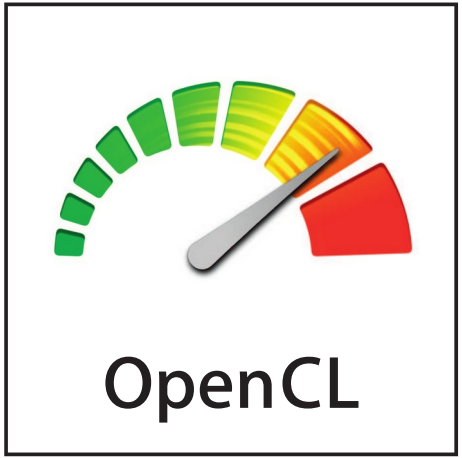
```
import pycuda.compiler as comp
import pycuda.driver as drv
import numpy
import pycuda.autoinit
mod = comp.SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")
multiply_them = mod.get_function("multiply_them")
a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)
dest = numpy.zeros_like(a)
multiply_them(drv.Out(dest), drv.In(a), drv.In(b), block=(400,1,1))
print dest-a*b
```

OpenCL

(Open Computing Language)

OpenCL

- Това е стандартна софтуерна рамка (framework) за писане на хетерогенни и силно паралелни приложения;
- Използват се езици базирани на C99 и C++11, разширени с ключови думи и примитиви за програмиране на широк клас устройства;
- Поддържа CPU, GPU, Digital signal processors (DSPs), Field-programmable gate arrays (FPGAs) и други процесори и хардуерни ускорители;
- Всяка програма се разделя на две части – базова (написана на host езика на основната програма) и kernel-и написани на OpenCL C или OpenCL C++, които се компилират от вградения в OpenCL библиотеката компилатор;



OpenCL пример – умножение на вектор и матрица (OpenCL C kernel)

```
// matvec.cl
__kernel void matvec(__global const float *A,
                     __global const float *x,
                     uint ncols, __global float *y)
{
  size_t i = get_global_id(0);              // Global id row index
  __global float const *a = &A[i*ncols]; // I'th row ptr
  float sum = 0.f;                          // For dot product
    for (size_t j = 0; j < ncols; j++) {
      sum += a[j] * x[j];
    }
    y[i] = sum;
}
```

OpenCL пример – главна програма

```
#include <CL/cl.hpp>
#include <iostream>
#include <fstream>
int main() {
  std::vector<cl::Platform> platforms;
  cl::Platform::get(&platforms);
  auto platform = platforms.front();

  std::vector<cl::Device> devices;
  platform.getDevices(CL_DEVICE_TYPE_CPU, &devices);
  auto device = devices.front();

  std::ifstream demoFile("matvec.cl");
  std::string src(std::istreambuf_iterator<char>(demoFile),
                  (std::istreambuf_iterator<char>()));
```

```
cl::Program::Sources sources(1,
  std::make_pair(src.c_str(), src.length() + 1));

cl::Context context(device);
cl::Program program(context, sources);
auto err = program.build("-cl-std=CL1.2");

float A[16][3]; float x[3]; float y;
cl::Buffer memBufA(context, CL_MEM_HOST_READ_ONLY, sizeof(A));
cl::Kernel kernel(program, "matvec", &err);
kernel.setArg(0, memBufA); // Other params and result ...

cl::CommandQueue queue(context, device);
queue.enqueueWriteBuffer(memBufA, GL_TRUE, 0, sizeof(A), A);
queue.enqueueWriteBuffer(memBufX, GL_TRUE, 0, sizeof(x), x);
queue.enqueueTask(kernel);
queue.enqueueReadBuffer(memBufY, GL_TRUE, 0, sizeof(y), y);

cout << y;
}
```

OpenHMPP, OpenACC, C++ AMP

(Коделети, кернели, #pragma, GPU, ...)

HMPP (Hybrid Multicore Parallel Programming)

- Работа с хетерогенен хардуер;
- Базиран е на подход подобен на OpenMP с използване на `#pragma` анотации;
- Хетерогенните изчисления се програмират в т.нар. **коделети** – прости функции написани на базовия език (C или Fortran), които се компилират до езика на „target“ платформи като CUDA, OpenCL и др.;
- Може да се използва съвместно с OpenMP, MPI и др.;

OpenACC, C++ AMP, ...

- Аналогично OpenACC (open accelerators), предоставя подобен подход, но се поддържат C, C++ и Fortran. Коделетите тук се наричат **kernels**;

- Подобно е предназначението и подхода в C++ AMP. Първоначално той е разработка на Microsoft и се базира на превеждане на прости функции, ламбда изрази и други маркирани с ***restrict(amp)*** до извиквания на API функции базирани на DirectX 11. По-късно се появяват и реализации реализирани на базата на Clang/LLVM и OpenCL;

OpenHMPP – пример

```
#pragma hmpp simple1 codelet, args[outv].io=inout, target=CUDA
static void matvec(int sn, int sm, float inv[sm],
                   float inm[sn][sm], float *outv){
  int i, j;
  for (i = 0 ; i < sm ; i++) {
    float temp = outv[i];
    for (j = 0 ; j < sn ; j++) {
      temp += inv[j] * inm[i][j];
  }
  outv[i] = temp;
}
int main(int argc, char **argv) {
  int n;
  /* codelet use */
  #pragma hmpp simple1 callsite, args[outv].size={n}
  matvec(n, m, myinc, inm, myoutv);
}
```

Въпроси?

apenev@uni-plovdiv.bg